

- › Hands-on preparation and practice
- › Practical skills advancement for practitioners
- › Prescriptive guidance from expert voices

CERTIFICATION STUDY COMPANION SERIES

Web Development Essentials Study Companion



Getting Ready for
the LPI 030-100 Exam

›

Malhar Lathkar

Apress®

Certification Study Companion Series

Editorial Board

Ryan Byrnes

Apress, New York, NY, USA

Spandana Chatterjee

Apress, Pune, India

Melissa Duffy

Apress, New York, USA

Shaul Elson

Apress, New York, USA

Miriam Haidara

Apress, Dordrecht, The Netherlands

Celestin Suresh John

Apress, Pune, India

Susan McDermott

Suite 4600, Apress, New York, USA

Aditee Mirashi

Apress, Pune, Maharashtra, India

Malini Rajendran

Apress, London, UK

Shiva Ramachandran

Apress, New York, USA

James Robinson-Prior

Apress, London, Essex, UK

Anandadeep Roy

Apress, Pune, Maharashtra, India

Smriti Srivastava

Apress, New Delhi, Delhi, India

The Apress Certification Study Companion Series offers guidance and hands-on practice to support technical and business professionals who are studying for an exam in the pursuit of an industry certification. Professionals worldwide seek to achieve certifications in order to advance in a career role, reinforce knowledge in a specific discipline, or to apply for or change jobs. This series focuses on the most widely taken certification exams in a given field. It is designed to be user friendly, tracking to topics as they appear in a given exam and work alongside other certification material as professionals prepare for their exam.

More information about this series at

<https://link.springer.com/bookseries/17100>.

Malhar Lathkar

Web Development Essentials Study Companion

Getting Ready for the LPI 030-100 Exam

Apress®

Malhar Lathkar
Near Venkateshkrupa Hospital, Nanded, India

ISSN 2731-8761 e-ISSN 2731-877X
Certification Study Companion Series
ISBN 979-8-8688-2696-2 e-ISBN 979-8-8688-2697-9
<https://doi.org/10.1007/979-8-8688-2697-9>

© Malhar Lathkar 2026

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Unless otherwise expressly agreed by the Publisher in writing, the Publisher does not authorize use or reproduction of all or any part of this work in any manner for the purpose of training, fine-tuning, evaluating, or developing artificial intelligence models, technologies, or systems. In accordance with Article 4(3) of the EU Directive 2019/790 on copyright and related rights in the Digital Single Market, the Publisher expressly reserves all rights with respect to this work under the text and data mining exception of the directive.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com www.springeronline.com, or visit . Apress Media, LLC is a Delaware LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation. This Apress imprint is published by the registered company APress Media, LLC, part of Springer Nature. The registered company address is: 1 New York Plaza, New York, NY 10004, U.S.A.

Introduction

The modern world is powered by millions of web applications that enable us to perform most of our day-to-day activities like ordering food and groceries, booking travel tickets, and engaging with family and friends through social media and more. Software companies always need web developers with a sound understanding of the fundamental web technologies. The Web Development Essentials certification by the Linux Professional Institute is a highly respected credential, particularly in the open source ecosystem.

The **Linux Professional Institute (LPI)** is a non-profit global organization. Founded in 1999, it offers certification programs in a wide range of technologies. The certification tests validate the candidate's understanding and expertise in their chosen area. Over the years, LPI's vendor-neutral certifications have become a gold standard for aspiring professionals.

LPI offers the certifications in three categories: Essentials, Linux Professional, and Open Technology. To be eligible for the certificate, you must pass the examinations conducted at Pearson VUE test centers or online via OnVUE.

LPI's **Web Development Essentials (Exam code 030-100)** is an entry-level certification with no prerequisites and is aimed at testing the applicant's knowledge of the fundamental concepts of web development. The examination structure and the objectives are designed to validate a beginner's understanding of core web technologies: HTML, CSS, JavaScript, and Node.js. The examination is one hour long, during which 40 multiple-choice questions (MCQs) must be answered.

This book is proposed to serve as a study companion for anyone who wishes to take this certification. Hence, it is organized as per the official objectives specified by LPI, the certifying authority.

The statement of objectives has five distinct topics/sections, each of which carries a certain weighting value that indicates its relative

importance on the exam. The distribution of questions (out of 40) is proportional to the weights of topics. The section-wise distribution of weights (in brackets) is as follows:

030 – LPI Web Development Essentials Exam

- 031 Software Development and Web Technologies (6)
- 032 HTML Document Markup (8)
- 033 CSS Content Styling (8)
- 034 JavaScript Programming (10)
- 035 NodeJS Server Programming (8)

Note that LPI, at its discretion, may make changes in the statement of objectives, the topics, and their weightings from time to time without any prior notice. The latest version of the objectives of the LPI Web Development Essentials certification exam is available at www.lpi.org/our-certifications/exam-030-objectives/.

Accordingly, this book is also organized into five parts, each part mapped to one of the topics in the official objectives and having three chapters. Each part starts by stating the main aspects it covers, the subtopics and their weights, and a brief note about the contents of each chapter. Each chapter concludes by summarizing the main points covered. At the end of each part, a set of MCQs along with answers is included to help the reader review their understanding.

Two model question sets (40 questions each) on the entire scope of the certification objectives, and as per the official weight distribution, are included at the end of the book.

This book is not a textbook on web technologies; it's designed to be an exam guide. There's no substitute for the well-established process of learning to code: writing, debugging, and experimenting with the code on your own. The working examples used in this book are available on its GitHub repository. The readers are encouraged to use it to strengthen their understanding.

While this book is primarily a comprehensive study guide for the LPI Web Development Essentials certification, it is also designed to

serve as a practical handbook for your career as a professional web developer.

With the roadmap now clearly laid out, let your journey begin!

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub. For more detailed information, please visit <https://www.apress.com/gp/services/source-code>.

Acknowledgments

This is my third collaboration with Apress (Springer Nature), a globally renowned publisher. My previous works on FastAPI and Django have been well-received, and it remains a great privilege to be associated with such a distinguished brand.

Publishing a technical book is a collective effort. I would like to take this opportunity to acknowledge the invaluable contributions of the many individuals who made this project possible.

This book is part of the Apress Certification Study Companion Series. Credit goes to Divya Modi, with whom I worked closely on previous projects, for suggesting this book idea. My sincere thanks to James Robinson-Prior, the senior editor, and Ms. Abirami D. S., the production editor in charge of this project, for their support in bringing this content to fruition. I also extend my gratitude to the members of the editorial board and the production team.

I must thankfully acknowledge the efforts of Fabio Claudio Ferracchiati, my technical reviewer, for their diligence in identifying technical and typographical errors in the draft. Their expert insights have been instrumental in ensuring the accuracy and quality of this work.

It is often easy to take the support of our near and dear ones for granted, but this is the appropriate space to place my appreciation for them on record. My family (including the newest addition, my ever-cheerful and inquisitive granddaughter, Shrimayee) has always stood behind me like a rock.

Shri L. K. Kulkarni (LK Sir to his students and admirers), an eminent author and a distinguished science educator in his own right, has been an inspiring personality. My heartfelt thanks go to him for his unwavering support and mentorship.

Being a teacher at heart has been immensely helpful in my growth as an author. I owe a great deal to my students, as teaching is the best way to learn and gain deeper insights into a subject.

Finally, I would like to thank all my well-wishers, readers, colleagues, and collaborators. This book is a result of the encouragement and community support I have received along the way.

Table of Contents

Part I: Basics of Web Development

[Chapter 1: Introduction to Software Development](#)

[Machine Language](#)

[Assembly Language](#)

[High-Level Language](#)

[Software](#)

[Compiler](#)

[Interpreter](#)

[IDEs](#)

[Programming Paradigms](#)

[Imperative Paradigm](#)

[Object-Oriented Paradigm](#)

[Declarative Paradigm](#)

[High-Level Programming Languages](#)

[C](#)

[C++](#)

[Java](#)

[Python](#)

[SQL](#)

[HTML](#)

[CSS](#)

[JavaScript](#)

[Libraries](#)

[Version Control](#)

[Software Testing](#)

[Points to Remember](#)

[Summary.](#)

[Chapter 2: Web Application Architecture](#)

[Client-Server Model](#)

[WWW](#)

[DNS Server](#)

[Web Browsers](#)

[Web Standards](#)

[JSON](#)

[XML](#)

[ES Standards](#)

[HTML Living Standard](#)

[JS Frameworks](#)

[Single-Page Applications](#)

[API](#)

[REST](#)

[GraphQL](#)

[Databases](#)

[Packaging Web Applications](#)

[WebAssembly.](#)

[Points to Remember](#)

[Summary.](#)

[Chapter 3: Anatomy of the HTTP Protocol](#)

[Key Principles of HTTP](#)

[Request-Response Cycle](#)

[Request Methods](#)

[GET Method](#)

[POST Method](#)

[PUT Method](#)

[DELETE Method](#)

[Request Headers](#)

[HTTP Response](#)

[Status Code](#)

[Response Headers](#)

[Cookies](#)

[HTTP Servers](#)

[HTTP URLs](#)

[Virtual Host](#)

[HTTPS and TLS](#)

[REST Constraints](#)

[WebSocket](#)

[Points to Remember](#)

[Summary](#)

[Knowledge Check: Part I](#)

[Answers with Explanations](#)

Part II: HTML for Web Design

Chapter 4: Basic Tags in HTML

Types of HTML Tags

Structural Tags

DOCTYPE

Root Element

Head Section

Body Section

Presentation Tags

Heading Tags

Paragraph Element

Character Entities

List Tags

Ordered List

Unordered List

Nested Lists

<div> element

Points to Remember

Summary

Chapter 5: Semantic Tags in HTML

Anchor Element

Other URL Schemes in Hyperlinks

Internal Hyperlinks

Layout Elements

<header> Tag

[<nav> Tag](#)

[<main> Tag](#)

[<section> Tag](#)

[<article> Tag](#)

[<footer> Tag](#)

[<aside> Tag](#)

[Disclosure Elements](#)

[<dialog> Element](#)

[Multimedia Elements](#)

[Tag](#)

[<picture> Tag](#)

[<figure> Tag](#)

[<audio> Tag](#)

[<video> Tag](#)

[<iframe> Tag](#)

[Text-Level Semantic Tags](#)

[Points to Remember](#)

[Summary](#)

[Chapter 6: Tables and Forms in HTML](#)

[Tables](#)

[The <table> Element](#)

[The <tr> Element](#)

[The <td> Element](#)

[The <th> Element](#)

[The <thead> Element](#)

[The <tfoot> Element](#)

[The <tbody> Element](#)

[The <caption> Element](#)

[Spanning](#)

[Forms](#)

[The <form> Element](#)

[The <label> Element](#)

[The <input> Element](#)

[<select> Element](#)

[The <datalist> Element](#)

[<textarea> Element](#)

[Points to Remember](#)

[Summary.](#)

[Knowledge Check: Part II](#)

[Answers with Explanations](#)

Part III: CSS for Content Styling

[Chapter 7: CSS Basics](#)

[Before CSS](#)

[The Element](#)

[The <center> Element](#)

[The <big> and <small> Elements](#)

[The bgcolor Attribute](#)

[The background Attribute](#)

[The width and height Attributes](#)

[The border Attribute](#)

[Inline Style](#)

[The <style> Element](#)

[External Stylesheets](#)

[CSS Syntax](#)

[CSS Properties](#)

[Color](#)

[Background](#)

[Border](#)

[Font](#)

[Text](#)

[Margin](#)

[height and width](#)

[Units](#)

[Absolute Units](#)

[Relative Units](#)

[Points to Remember](#)

[Summary.](#)

[Chapter 8: CSS Selectors](#)

[Universal Selector](#)

[Element Selector](#)

[ID Selector](#)

[Class Selector](#)

[Attribute Selector](#)

[Combinators](#)

[Descendant Combinator](#) (" ")

[Child Combinator](#) (>)

[Subsequent Sibling Combinator](#) (~)

[Adjacent Sibling Combinator](#) (±)

[Pseudo-Classes](#)

[User Interaction](#)

[Structural](#)

[Form/Input State](#)

[Location](#)

[Pseudo-Elements](#)

[Inheritance](#)

[Specificity](#)

[!important](#)

[Points to Remember](#)

[Summary](#)

[Chapter 9: CSS Layouts](#)

[Box Model](#)

[Flow Layout](#)

[The float Property](#)

[The position Property](#)

[static](#)

[relative](#)

[absolute](#)

[fixed](#)

[sticky](#)

[The display Property](#)

[Flexbox](#)

[Grid Layout](#)

[Spanning](#)

[Responsive Web Design](#)

[Viewport](#)

[Fluid Layouts](#)

[Media Queries](#)

[Points to Remember](#)

[Summary](#)

[Knowledge Check: Part III](#)

[Answers with Explanations](#)

Part IV: JavaScript Programming

[Chapter 10: JavaScript Fundamentals](#)

[Enabling JavaScript](#)

[JavaScript Inside HTML](#)

[Basic Syntax](#)

[Keywords](#)

[Statements](#)

[Comments](#)

[Identifiers](#)

[Data Types](#)

[Primitive Types](#)

[Object Types](#)

[The typeof Operator](#)

[Variables](#)

[Strict Mode](#)

[Dynamic Typing](#)

[Operators](#)

[Arithmetic Operators](#)

[Comparison and Logical Operators](#)

[Compound Assignment Operators](#)

[Type Conversion](#)

[Implicit Conversion](#)

[Explicit Coercion](#)

[Conversion to String](#)

[Conversion to Number](#)

[Strict Equality Operator](#)

[Points to Remember](#)

[Summary](#)

[Chapter 11: JavaScript Control Structures](#)

[Conditional Statements](#)

[The if Statement](#)

[Nested if](#)

[The else if Statement](#)

[switch and case](#)

[Loops](#)

[The while Loop](#)

[The do while Loop](#)

[The for Loop](#)

[for...in](#)

[for...of](#)

[break and continue](#)

[Functions](#)

[Built-in Functions](#)

[User-Defined Functions](#)

[A Function with Parameters](#)

[A Function with a Return](#)

[Arrow Functions](#)

[Recursive Functions](#)

[Points to Remember](#)

[Summary.](#)

[Chapter 12: JavaScript and DOM](#)

[What Is the DOM?](#)

[Document Object](#)

[Document Properties](#)

[Document Methods](#)

[HTMLElement Object](#)

[HTMLElement Properties](#)

[DOM Traversal](#)

[DOM Events](#)

[Event Handler Properties](#)

[Event Listener Methods](#)

[Points to Remember](#)

[Summary](#)

[Knowledge Check: Part IV](#)

[Answers with Explanations](#)

Part V: Server Programming with Node.js

[Chapter 13: Node.js Basics](#)

[Installation](#)

[Node.js vs. JavaScript in a Browser](#)

[REPL Terminal](#)

[Script Mode](#)

[Node.js Modules](#)

[Node Package Manager \(npm\)](#)

[npm install](#)

[npm init](#)

[Callbacks](#)

[Error-First Callback](#)

[http Module](#)

[The createServer\(\) Function](#)

[The response Object](#)

[The request Object](#)

[Points to Remember](#)

[Summary](#)

[Chapter 14: Express.js Framework](#)

[Express: Important Features](#)

[Routing](#)

[Middleware](#)

[Static Assets](#)

[Request and Response Helpers](#)

[Template Engine Support](#)

[Installation](#)

[Hello Express](#)

[express-generator](#)

[Routing Mechanism](#)

[Path Parameters](#)

[Middleware](#)

[Built-in Middleware](#)

[express.json\(\)](#)

[express.urlencoded\(\)](#)

[express.static\(\)](#)

[express.Router\(\)](#)

[Template Engine](#)

[EJS Templates](#)

[EJS Syntax](#)

[Static Assets](#)

[Error Handling](#)

[Points to Remember](#)

[Summary](#)

[Chapter 15: Node.js and SQLite](#)

[RDBMS Concepts](#)

[SQL](#)

[SQL Data Types](#)

[SQLite](#)

[Installation](#)

[DB Browser for SQLite](#)

[SQLite Extension for VS Code](#)

[SQLite3 Fiddle](#)

[SQLite Command-Line Shell](#)

[sqlite3 Module](#)

[Connecting to a Database](#)

[Database Connection Object](#)

[db.run\(\)](#)

[db.close\(\)](#)

[db.get\(\)](#)

[db.all\(\)](#)

[db.each\(\)](#)

[Transaction Control](#)

[Other SQLite Modules](#)

[SQL Injection](#)

[SQLite in Express App](#)

[Points to Remember](#)

[Summary](#)

[Knowledge Check: Part V](#)

[Answers with Explanations](#)

[Model Test Paper I](#)

[Model Test Paper II](#)

[Epilogue: Beyond Essentials](#)

[Answer Keys](#)

[Index](#)

About the Author

Malhar Lathkar

brings over 35 years of experience as an independent software developer, entrepreneur, author, trainer, and mentor. Though formally trained in electronics at the postgraduate level and having begun his professional career as a lecturer at a degree college, he has successfully transitioned into software training and development as a self-taught expert.

A passionate educator at heart, Malhar has positively impacted the careers of countless students and professionals worldwide, particularly in Node.js, Python, and Java. He actively collaborates with various EdTech companies as a subject matter expert, contributing to the design of high-quality training programs.

A recognized author, Malhar has published several titles with prominent publishing houses, including his books on FastAPI and Django by Apress. He also provides specialized software training services to corporate clients, combining hands-on learning with industry-relevant skills. He is frequently invited to conduct workshops and deliver technical talks at various academic institutions.

Beyond his professional pursuits, Malhar enjoys Indian classical music and is an avid sports enthusiast.

About the Technical Reviewer

Fabio Claudio Ferracchiati

is a senior consultant and senior analyst/developer specializing in Microsoft technologies. He currently works at Telecom Italia (www.telecomitalia.it). Fabio holds several Microsoft certifications, including Microsoft Certified Solutions Developer (MCSD) for .NET, Microsoft Certified Application Developer (MCAD) for .NET, and Microsoft Certified Professional (MCP). He is also a prolific author and technical reviewer. Over the past decade, he has contributed numerous articles to both Italian and international technology magazines and has coauthored more than ten books covering a wide range of computing topics.

Part I

Basics of Web Development

Basics of Web Development

The Linux Professional Institute (LPI) conducts the Web Development Essentials Exam (Exam Code: 030-100), with its official objectives divided into five main topics. As a study companion for this certification exam, this book is organized according to these objectives, with five corresponding parts.

Part I is mapped to **topic 031: Software Development and Web Technologies**. It carries 6 weightage points and covers the fundamental concepts such as the basics of software development, different programming languages and their types, and core web technologies. This part has three chapters, one corresponding to each of the subobjectives 031.1, 031.2, and 031.3.

Chapter [1](#)

The first chapter is aimed at helping the candidate to get familiar with the fundamentals of software development, such as various programming paradigms and languages, IDE tools, and concepts like version control and software testing.

Chapter [2](#)

Chapter [2](#) concentrates on the role of the client-server model in web applications, various web browsers, the evolving web standards, front- and back-end technologies, the principle of APIs and their types, among other concepts.

Chapter [3](#)

The HTTP protocol drives the web. This chapter covers key terms, such as the request-response cycle, HTTP methods, HTTP status codes, headers and cookies, and the REST constraints. A solid understanding of these concepts is important for any web application developer.

Despite a modest weightage of 6 points (out of 40), this part lays a strong foundation for an aspiring web developer. The candidate is advised to test their knowledge of these chapters by solving the knowledge-check questions at the end of the third chapter. It is also followed by the answers, backed up by appropriate explanations.

1. Introduction to Software Development

Malhar Lathkar¹ 

(¹) Near Rachewad Hospital, Nanded, Maharashtra, India

The software is where the magic is. If you're going to have all this power be simple enough, appealing enough and cool enough, it's going to be because the software is right.

—Bill Gates

For all the rapid strides in computer technology in the last seven to eight decades (from 8-bit to 64-bit processors with multiple cores, powerful GPUs for faster graphics processing, compact and efficient storage devices with large volume, etc.), it is really the software—with a well-designed user interface and ability to tap into the enormous resources of the hardware—that makes the application more productive. So, in that sense, Bill Gates, one of the most decorated and successful software developers, is spot on. It is indeed the magic behind any computing device, such as your regular PC/laptop, mobile, or a stand-alone machine.

Modern digital computers are built around a microprocessor. The microprocessor can fetch a sequence of raw binary machine language instructions from memory and execute them with the help of an arithmetic and logical unit (ALU), a control unit (CU), and registers (accumulator, stack, and others).

Machine Language

A program (a set of instructions provided in a sequence to be executed by the processor) written in raw binary depends on the processor architecture. For a typical 8-bit microprocessor (Intel 8085), the binary code to add two numbers 10 and 20 (0x0A and 0x14 respectively in Hexadecimal form) would be as follows.

```
00111110 00001010
11000110 00010100
00110010 00000000 00100001
01110110
```

This code essentially loads 10 into the accumulator, adds 20 to it, and stores the result in the specified memory location before coming to a halt.

Each instruction in the preceding program has two components: an opcode that defines the type of operation that the ALU has to perform, and the operand, which is the data on which the opcode operates. See Figure [1-1](#).

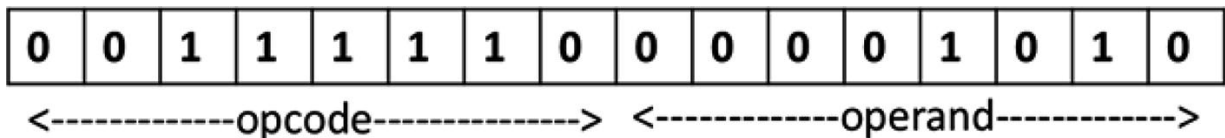


Figure 1-1 Opcode and operand

Assembly Language

Writing programs in machine code is tedious. The programmer has to deal with registers, memory addresses, and call stacks. The introduction of Assembly language was the first step toward simplifying development. Assembly language uses mnemonics and labels to represent opcodes and memory locations.

Here's the 8085 assembly language code equivalent to the preceding machine language program.

```
MVI A, 0AH
ADI 14H
STA 2100H
HLT
```

However, the assembly language code, like the machine code, is not compatible across processors of different architectures. The advent of higher-level programming languages such as FORTRAN and COBOL in the 1960s started making programmers' lives a little easier.

High-Level Language

With high-level programming languages, one can now write more human-readable instructions that can be easily ported across different computer architectures. They provide a convenient layer of abstraction, letting the programmer focus more on the processing logic than on the internals of the hardware. Each programming language defines a set of predefined keywords. A keyword resembles a word in English. The usage of keywords is controlled by well-defined rules called *syntax*.

The programmer can now use these keywords along with named locations (variables) to compose a sequence of instructions called the source code of the program. Hence, a program in C that is equivalent to the preceding assembly code that performs addition of two numbers.

```
#include <stdio.h>

int main() {
    int a=10;
    int b=20;
    int c=a+b;
    printf("c:%d", c);
    return 0;
}
```


Instructions in the source code are saved as normal text files. However, their extension indicates the language used to write the program. For example, the preceding C program is saved as `add.c`, where the `.c` extension lets you know that it is a C program. Similarly, `.cpp` implies that the source code is written in C++, the `.java` extension is used for program files written in Java, and so on.

The instructions in the source code are in turn translated into the machine code (also called *object code*) of the target processor with the help of a compiler or interpreter. Although both translate the source code to equivalent machine code, they use different approaches.

Software

A program is a set of instructions given to the computer (these instructions may be in any computer language) to perform a single specific task. The preceding C code is a program that adds two numbers. On the other hand, *software* is a much broader term. It may comprise a collection of programs, along with the associated data and design specifications.

Software is intended to perform multiple related operations. The Calculator app on your computer/mobile is an example of software, as it is a collection of programs that perform addition, subtraction, and so forth.

Similarly, a word processor such as Microsoft Word is a collection of interconnected programs. You can consider the action performed by each menu item or a button in the toolbar to be a single program; the result of one may be used by another.

Software is broadly classified into two categories: system software and application software. See Figure [1-2](#).

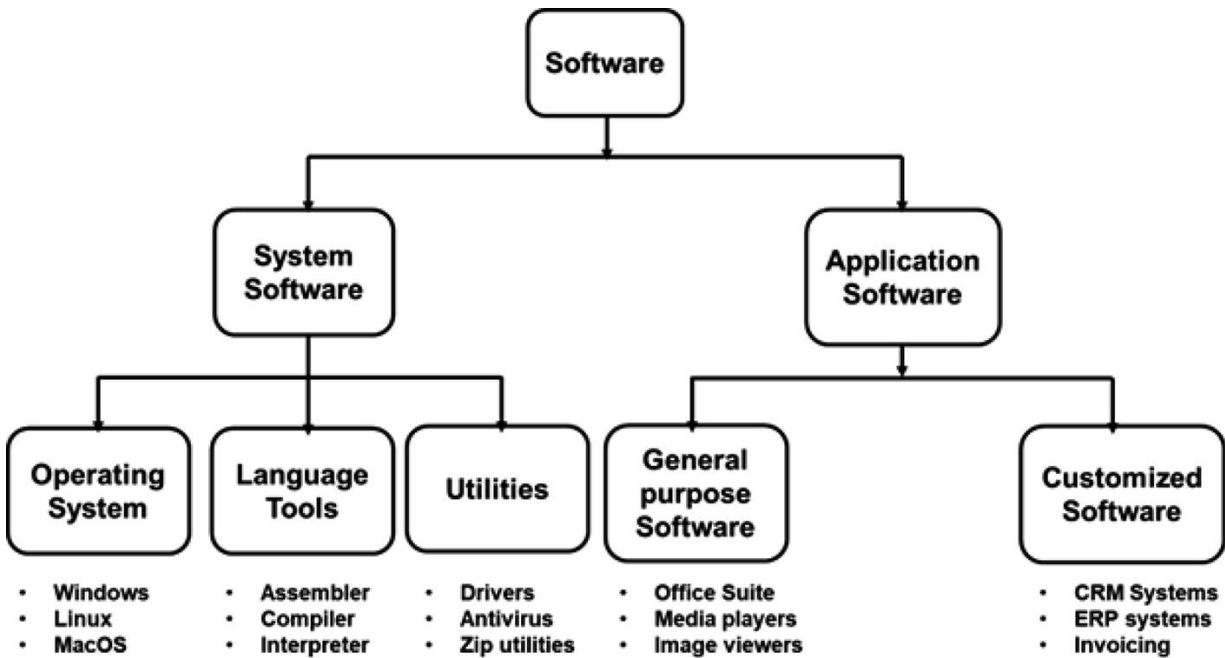


Figure 1-2 Types of software

System software manages hardware resources and facilitates their use by application software. The application software, in turn, is what the end user actually works with. While the system software provides a general platform for the application software to run, the latter is developed for a specific application area. MS Word is an application software for word processing. A web browser is also an application software whose objective is to browse the web pages on a website.

A general-purpose application software has a specific application area but is designed to be used by a wider user base. The constituents of the Microsoft Office suite of applications (such as Word, Excel, PowerPoint) fall in this category. Developers use integrated development environments (IDEs) such as Code::blocks or Visual Studio for streamlining the software development process. They are also examples of general-purpose application software.

Application software may also be customized to meet the requirements of a particular user. A client may approach the software developer to build invoicing software exclusively for his workflow.

The system software is further classified into three types: operating systems, compilers/interpreters, and utilities like device drivers.

The operating system (Windows, Linux, macOS, etc.) is the overall manager of the computer system. It is responsible for communicating with input/output devices, managing the memory resources, and handling the file system.

The code written in a programming language to be executed has to be translated into machine code. Assemblers, compilers, and interpreters specific to the hardware architecture/OS are used for this purpose.

Additionally, you often need to use other hardware-specific programs like device drivers and compression tools.

In short, the system software is a general-purpose set of tools that the application software takes advantage of.

Compiler

A compiler takes the complete source code into account, checks whether the keywords used in the code follow the language syntax, and builds a machine code equivalent of the source code. Note that once compiled, the original source code is no longer needed and the compiled code can be used repeatedly.

C, one of the earliest and still one of the most used programming languages, is compiler-based. Many others, inspired by C (such as C++, Objective-C, Go, Rust), are also compiler-based languages.

What makes it possible to run the same program in a high-level language (such as C) on any hardware platform? As mentioned earlier, you would use the predefined keywords to put the processing logic in the source code. Using the hardware-specific compilers for the given language, the same source code can be used on computers running on different processor architectures and operating systems. See Figure [1-3](#).

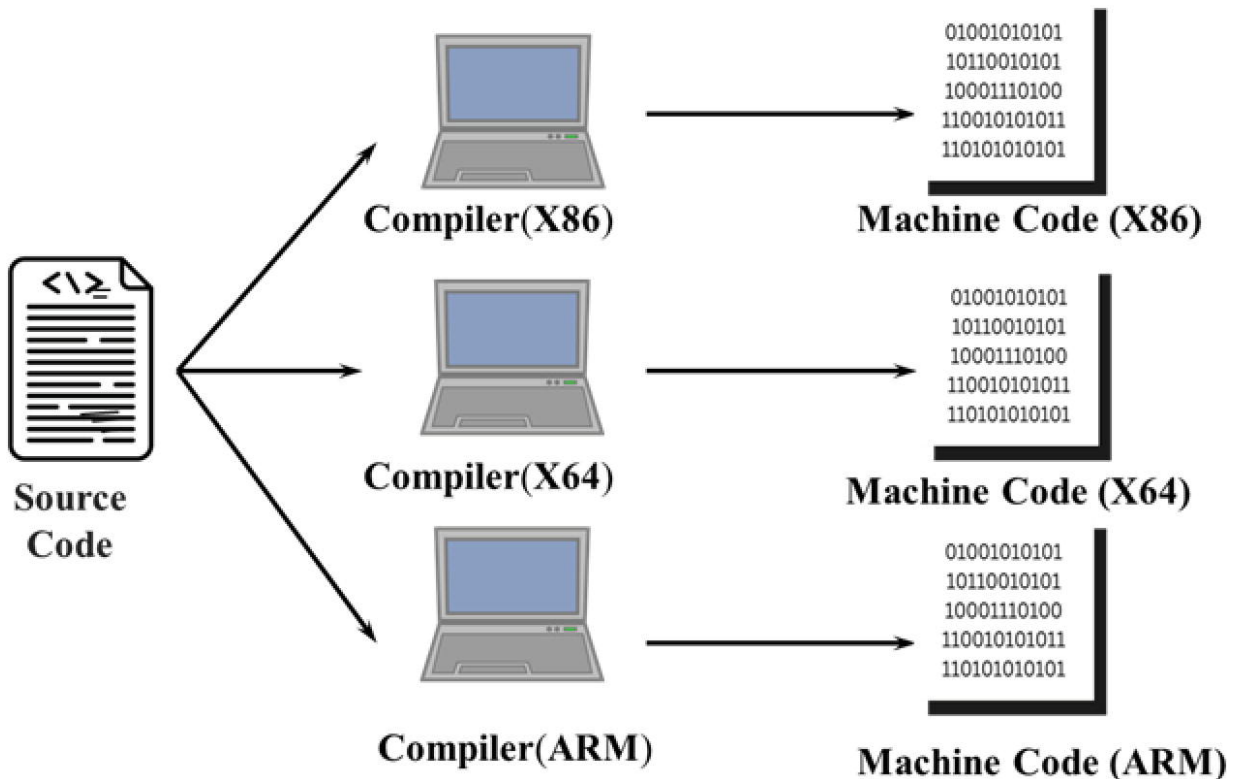


Figure 1-3 Language compiler

In other words, the source program in C is compiled for a specific hardware platform. The compiled machine code, however, cannot be used on other hardware platforms.

So, if you are using gcc (an open source, cross-platform C compiler on Windows), the C source code is compiled with the command as follows.

```
gcc -c add.c -o add.o
```

This results in the object version of the C source code. To obtain an executable version of the object code, it must be combined with the necessary libraries. On a Windows computer, add.exe is built with the following command.

```
gcc -o add.exe add.o
```

If you are working with a Linux-based system, use the following command.

```
gcc add.o -o add
```

Interpreter

Other equally popular programming languages such as Python, PHP, and Ruby are interpreter-based. The interpreter takes a different approach to converting source code into machine code. Unlike the compiler, it takes one instruction from the source code at a time, generates its equivalent machine-level instruction, which in turn is executed by the processor. The translated version of the source code is not retained. See Figure [1-4](#).

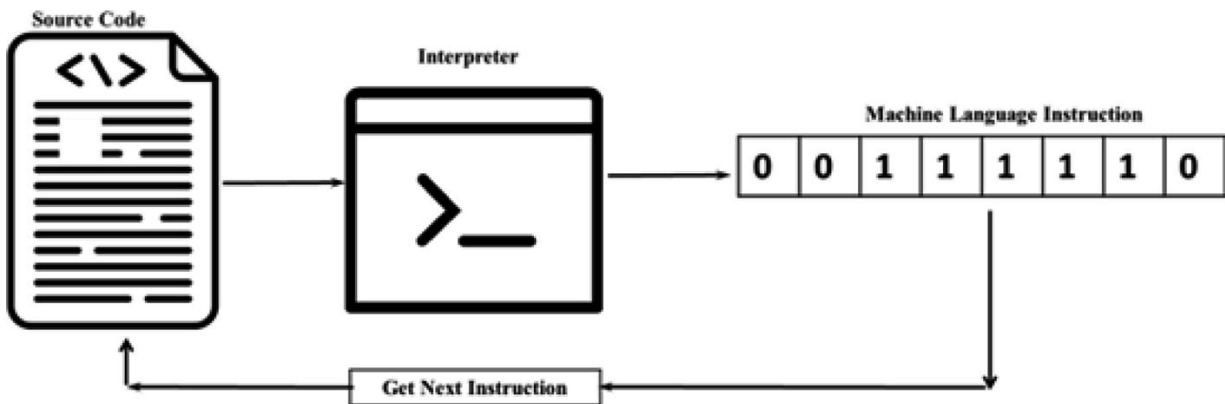


Figure 1-4 Interpreter

Many interpreter-based languages like Python provide an interactive programming environment called a REPL shell (REPL stands for read-evaluate-print-loop) inside which you can execute one instruction at a time in front of the Python prompt (`>>>`).

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print ("c:", c)
c: 30
```

Instead of executing one instruction at a time in the REPL shell, you can save a set of instructions in a text file with a .py extension and run the script with the following command.

```
python add.py
```

Java, another hugely popular programming language, follows the execution model that is a mix of compiler and interpreter. The Java source code (saved in a text file with the .java extension) is submitted to the Java compiler. Like a C compiler, it checks the source code for syntactic correctness and produces an intermediate, platform-independent bytecode. A Java interpreter (a.k.a. Java Virtual Machine, or JVM) then converts this bytecode to a raw binary code that is native to the specific hardware/operating system. Other languages that follow the same technique include C# and Kotlin. See Figure [1-5](#).

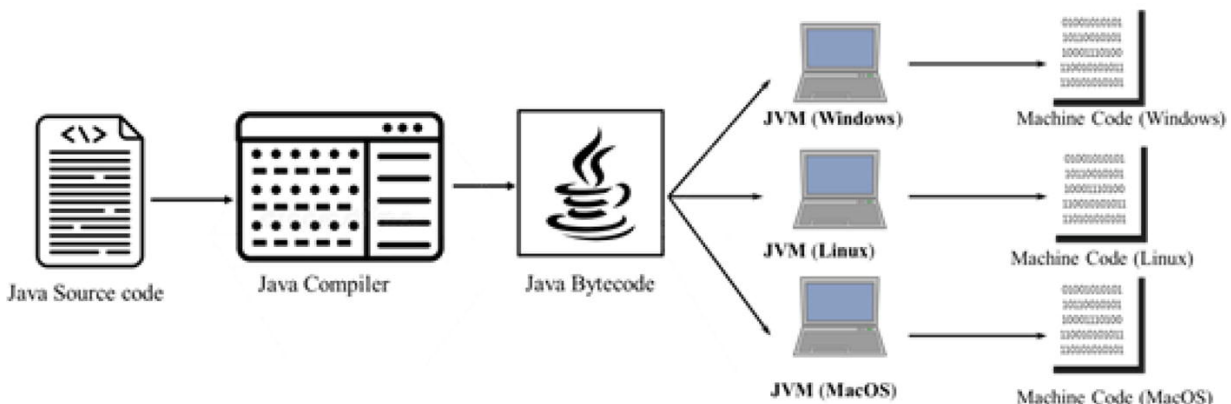


Figure 1-5 Java Virtual Machine (JVM)

Executing a Java program is thus a two-step process. The first step involves using the Java compiler to obtain the bytecode.

```
javac add.java
```

If successfully compiled, this generates a file with the .class extension and has the bytecode representation of the source code.

In the second step, use the Java executable to run the bytecode.

```
java add
```

Here, the Java executable is the Java Runtime (JRE) for the operating system on which the program is run. The JRE includes the virtual machine and the essential core libraries. Compilation is a one-time process. The same .class bytecode so generated is executed with the JRE (and JVM) of different OS platforms.

IDEs

As mentioned earlier, the source code is saved as a text file, with an appropriate extension indicating the programming language (the .c extension for a C program, .cpp for C++, .py for Python, etc.). As such, any text editor can be used to create the source program file. Operating systems often have built-in text editors. Notepad is a basic text editor app that is available on all versions of Windows OS. Linux comes with vi and Nano editors. Similarly, macOS has TextEdit.

However, using any of these general-purpose text editors for program development is not the most convenient solution. You open an editor, enter the code, and save it with the respective extension. Next, you need to use the compiler/interpreter to run it. But more often than not, the source code may contain errors, which the compiler will point out. To correct it, you should open the editor again, make changes, save, and compile it again till it is error-free.

IDE helps you avoid the hassle of repeatedly switching between editing and compiling. IDE stands for *integrated development environment*. The IDE software is a one-stop solution for editing the source code, compiling and debugging it, as well as packaging and deploying it. In other words, it offers an integrated environment with all the features required, from editing to executing, in one application itself.

IDEs generally have some very useful features. Syntax highlighting is one of the most important features. Elements like keywords and

variables are identified with distinct colors. The IDE also highlights syntax errors so that they can be spotted before running the program.

Modern IDEs provide intelligent autocomplete features, thereby helping speed up development. You also get refactoring and version control features in IDEs such as IntelliJ IDEA. See Figure 1-6.

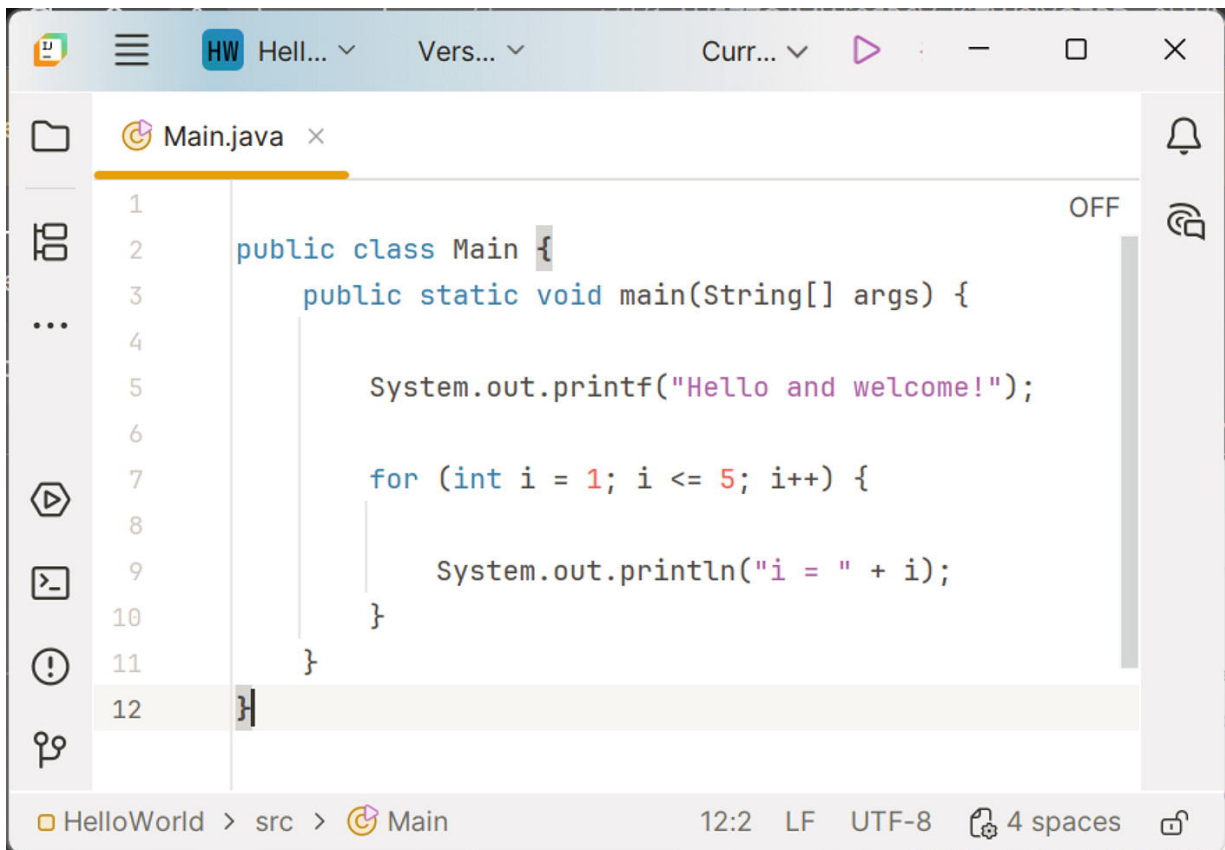


Figure 1-6 IntelliJ IDEA

Some IDEs are designed to work with a specific language, while others support multiple languages. Code::Blocks is an open source IDE for C, C++, and FORTRAN languages. PyCharm IDE is used exclusively for Python. Visual Studio supports development in the .NET family of languages, including VC++, C#, VB.NET, and F#.

Sometimes developers also prefer using source code editors instead of full-fledged IDEs. A source code editor is a lightweight tool that offers basic syntax highlighting. Some editors provide out-of-the-box

integration with Git version control. Most source code editors support multiple languages.

Notepad++ (Windows), Vim (Linux), Sublime Text, and Atom are some examples of source code editors. Visual Studio Code (VS Code), contrary to a general perception, is a source code editor and not an IDE. It is a cross-platform, free and open source editor developed by Microsoft and is extremely popular among developers. VS Code has a rich ecosystem of extensions (plug-ins) that helps with rapid development in many languages and frameworks. See Figure [1-7](#).

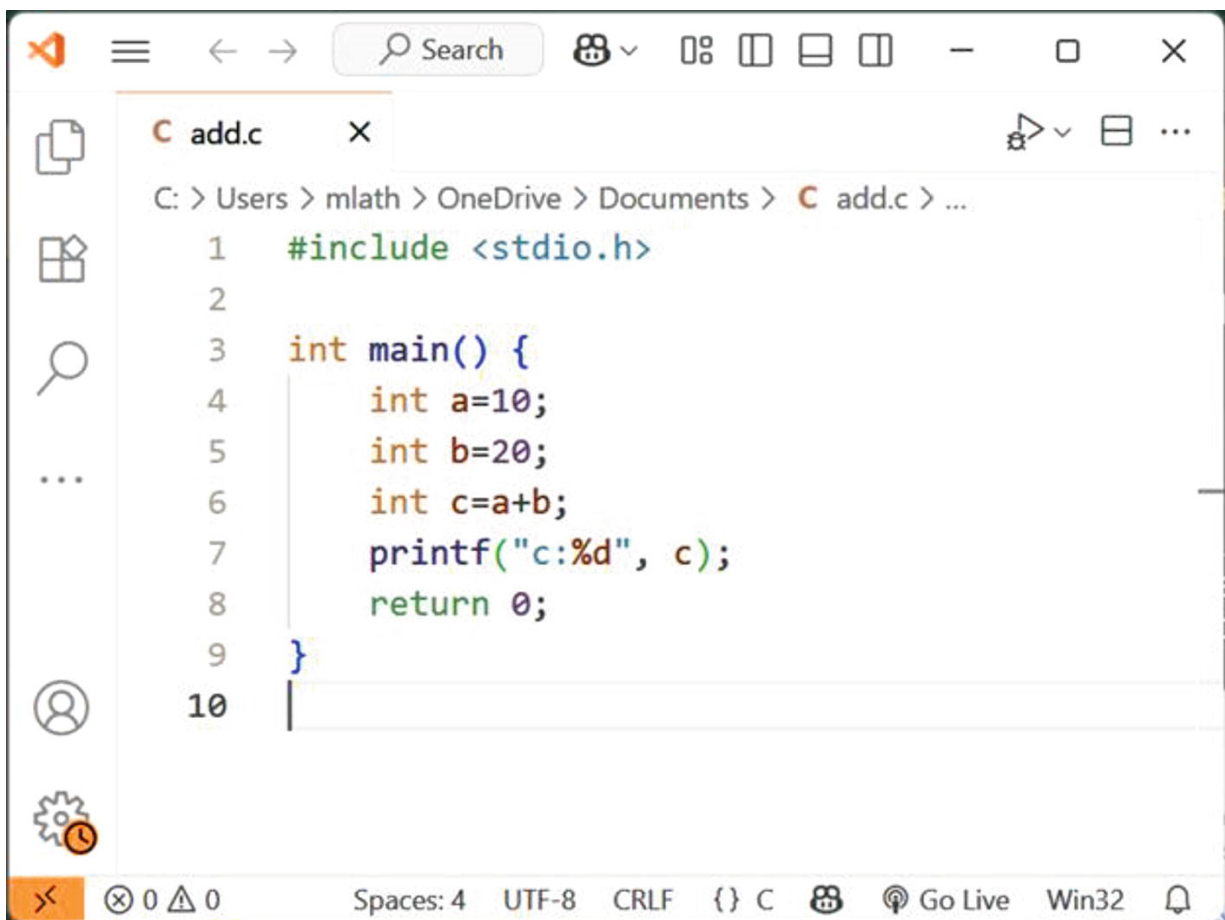


Figure 1-7 VS Code

The major point of difference between a full-fledged IDE and a source code editor is that the IDE works in close association with the compiler/interpreter, and hence provides debug and build tools (which the source code editor doesn't support).

Programming Paradigms

A casual Internet search tells us that the total number of programming languages runs into thousands. Since John Neumann introduced the concept of a stored program in the 1940s, a new language has come up every now and then, while some have become obsolete over time. Most languages appear similar—with almost identical keywords. However, each one of them follows a particular programming philosophy. In fact, the choice of keywords and other constituents of a language is governed by the philosophy, the approach that its creator has in mind.

The term programming paradigm refers to the approach or methodology for conceptualizing, implementing, and structuring a computer program. Each language supports a certain programming paradigm. Of course, some are multiparadigm languages.

Each of these paradigms (imperative, object-oriented, functional, etc.) adopts a unique approach toward programming. When a developer sets out to develop a solution, he has to decide which programming approach (paradigm) is best suited for the situation. This determines the choice of the programming language.

Imperative Paradigm

A program written in a language that follows the imperative paradigm is a sequence of explicit commands for the computer to execute. The focus is primarily on describing the steps to change the program's state to achieve the desired result. Think of imperative programming as a step-by-step guide for assembling a product, a recipe, or navigation instructions that lead the driver to the desired destination.

Procedural programming is a type of imperative approach where the programming logic is organized into more than one reusable subroutine (also called *procedures* or *functions*).

Structured programming methodology is also a subtype of imperative approach. It uses control flow constructs such as for, while, case, and so on.

Lower-level languages like assembly and machine code are inherently imperative. Early high-level languages like FORTRAN and COBOL are also classified as imperative languages.

C is the best-known example of a procedural language, as are the early versions of Python. Since most of the modern languages include control flow statements such as if-then, for, and while, they can be said to be structured in nature.

Object-Oriented Paradigm

The imperative paradigm places a lot of importance on the process, almost reducing the data on which the process is performed to secondary status, moving freely through different subroutines or functions. However, in the real world around us, the primary unit of processing is an object (a product, an invoice, an employee, etc.), each having a set of attributes and behavior associated with it. Moreover, they are not independent of each other; instead, they co-exist.

The object-oriented paradigm is based on this premise. In this case, the attributes (properties) and their functionality (methods or functions) are defined as a single unit called a *class*. A class serves as a template for all the objects having similar attributes and functionality. The fundamental principle of object-oriented programming methodology is that a class defines what attributes its objects have and how it behaves. The object, on the other hand, is an instance of the class.

An employee is characterized by several attributes, such as the name, designation, age, and date of appointment. The processing of employee data includes computing salary and taxes, applying increments, and processing promotions. In object-oriented terminology, you define an Employee class as a data type. See Figure [1-8](#).

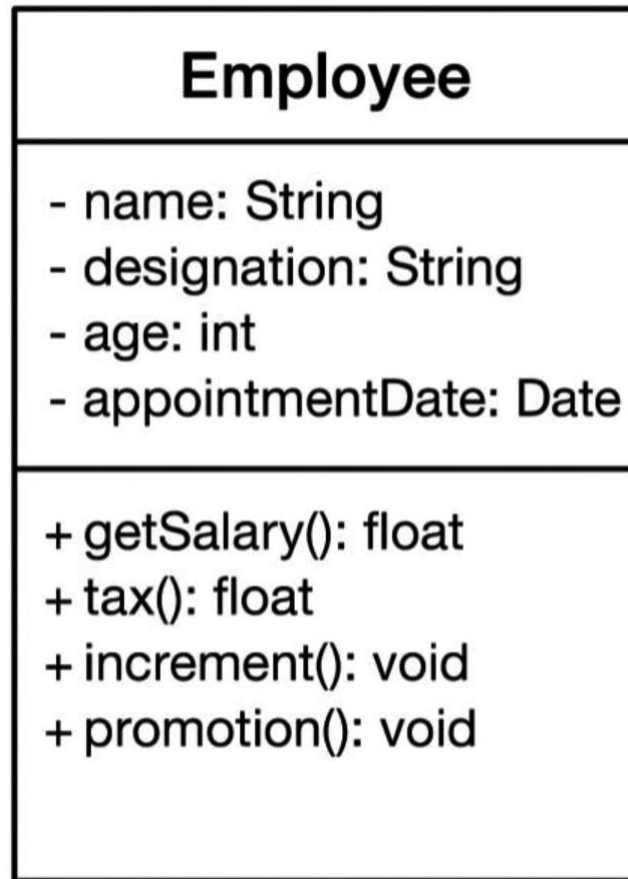


Figure 1-8 Employee class

One of the objects of the Employee class has the name John, age 28, and is working as a manager. Thus, individual employees in the organization are the objects of the Employee class.

Other principles of the object-oriented paradigm are based upon the concept of class and object, and their relationship. Encapsulation is an important concept of OOP that helps limit access to an object's data members only through methods. The principles of inheritance and polymorphism allow an existing class to be extended and reused. See Figure [1-9](#).

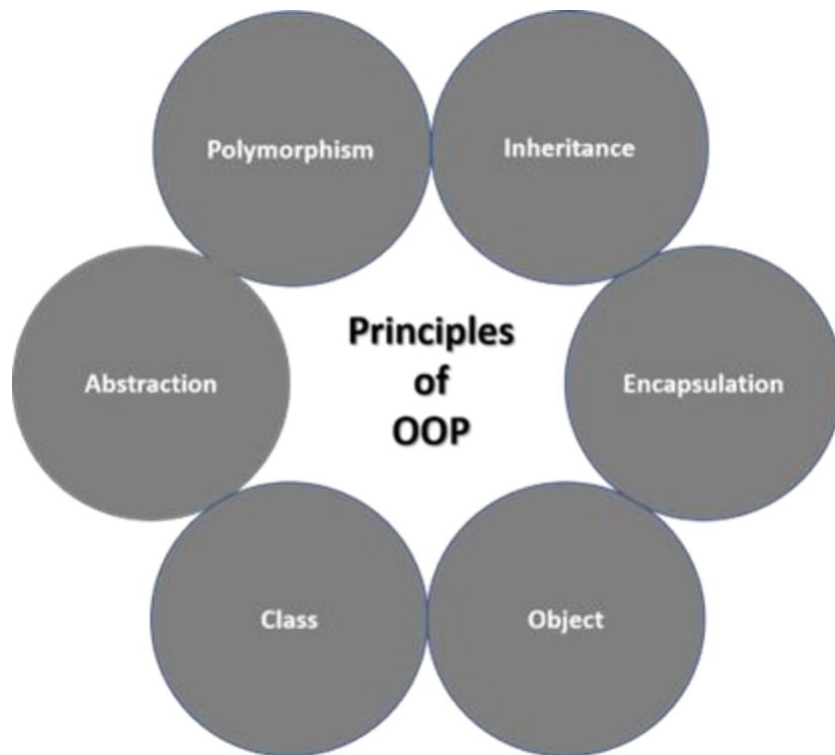


Figure 1-9 Principles of OOP

An American computer scientist, Alan Kay, is widely regarded as the pioneer of object-oriented methodology when he developed the SmallTalk language. In 1985, Bjarne Stroustrup added object-oriented capability to C and named it C++, which is still one of the most widely used languages. Java (developed by James Gosling in 1995) and Python (developed by Guido van Rossum in 1991), dominating today's software development ecosystem, are also prime examples of the object-oriented paradigm. JavaScript is said to implement prototype-based programming, which is a subtype of the object-oriented paradigm.

Declarative Paradigm

The declarative programming approach is directly in contrast to the imperative approach. The declarative paradigm focuses mainly on the properties of the desired result, but doesn't describe the step-by-step procedure to achieve it. In other words, it states what computation should be performed, but it doesn't specify how to accomplish it as an

imperative program does by elaborating it with various programming constructs such as conditionals, loops, and subroutines.

SQL is the best example of a declarative language. Assuming that a database has an Employee table, the following one-line SQL query is enough to output the records in ascending order of names in the table.

```
SELECT * FROM EMPLOYEE ORDER BY NAME
```

However, imperative/OOP languages such as C, C++, and Java implement algorithms in explicit steps to achieve a similar result. They would require lengthy code involving arrays, nested loops, and so forth.

HTML and CSS, languages widely used in front-end web development, are also declarative in nature. Functional programming is a subtype of the declarative paradigm. In this, the expected result is the value of a series of mathematical functions. Haskell, F# and Erlang are notable examples of functional languages.

High-Level Programming Languages

Today, computers and computing devices are everywhere—from businesses to education, from sports to forecasting, from warfare to healthcare, and more. This all-pervading use of computers has mainly resulted from advancements in programming languages and programming techniques. In the 1960s, the professional use of computers was facilitated by applications written in COBOL and FORTRAN. Over the years, many computer languages, following different paradigms, have been developed and are being used in different areas of application. Let's look at a brief overview of some of the widely used languages, including the ones used in web development.

C

Created more than 50 years ago (by Dennis Ritchie in 1972), C is still one of the most widely used languages. It has had a tremendous influence on the development of many modern languages like Java,

Python, PHP, Objective-C, and more in terms of syntax and other features. In that sense, C is often considered to be the mother of all programming languages.

Because of its ability to have direct access to the hardware resources, it is used extensively in systems programming for tasks such as the development of operating systems, device drivers, and other hardware-specific utilities. The Unix kernel, originally developed in assembly language, was rewritten in C by Dennis Ritchie and Ken Thompson (working at Bell Labs). This enabled Unix to be ported across different hardware platforms.

The specifications of the C language are periodically updated. Starting with ANSI C in 1989, C23 is the latest revision of the C standards published in 2024.

C++

As the name suggests, C++ is the upgraded version of C. Bjarne Stroustrup, a Danish computer scientist also working at Bell Labs, added object-oriented features to the original C to call it “C with classes.” He later renamed it C++ in 1985.

C++ is the first full-fledged implementation of the object-oriented paradigm. Being a superset of C, C++ can also be used to develop C-style procedure-oriented programs. Like C, C++ is also largely used in hardware-specific programming. C++ is widely used in the development of Linux and Windows operating systems. The Microsoft Visual C++ compiler (MSVC) is part of Visual Studio and is used to develop applications specifically for the Windows platform. VC++ runtime libraries are needed to run Windows applications.

The C++ standards have been periodically revised, with the latest stable release, C++23, published in October 2024.

Java

At Sun Microsystems, James Gosling, along with his colleagues Mike Sheridan and Patrick Naughton, started the Java Language Project with the intention of providing platform independence to application

programmers (i.e., the WORA principle: write once, run anywhere). Its first version was released in 1995. Over the years, Java's ownership has changed hands from Sun Microsystems to Oracle Corporation. Java follows a regular version update policy. Java 21 is the latest LTS (long-term support) version. Java SE 24 is the latest stable version, released in March 2025.

Java achieves platform independence with a unique execution model, with the Java software being a combination of JDK and JRE. The Java compiler (part of JDK) produces the intermediate bytecode version of the source code (the .class file). It is in turn interpreted and executed by the hardware-specific JVM (included in JRE). Although much of its syntax is similar to C/C++, Java doesn't have access to the hardware resources, owing to its platform-neutral architecture.

Oracle offers commercially licensed versions of the Java Development Kit, whereas OpenJDK is an open source implementation of the Java platform. OpenJDK is the official reference implementation and the most popular distribution of JDK.

Python

The popularity of the Python language has touched new heights in recent years. This is mainly due to its widespread use in the field of machine learning and artificial intelligence. There are many reasons behind its popularity; one important one is its clean syntax and, as a result, easy code readability. Compared to some of the other languages, Python is easy to learn. Its large community keeps the language continuously evolving.

Simplicity is one of the important principles of Python's design philosophy. Its multiparadigm nature makes it suitable for developing applications of different types: desktop, command line, even for scheduling CRON jobs that are executed automatically.

Guido van Rossum developed the early version of Python in 1991. He remains at the forefront of the language's ongoing development, although the Python Software Foundation looks after development and maintenance. Python's version 2.0 was published in October 2000.

Subsequently, support for Python 2.x was discontinued in 2017. Meanwhile, Python 3, with many new and advanced features, was released in 2008, and since then, periodic versions of Python 3.x are regularly released. Python 3.13 is the latest stable version.

Python is an open source and cross-platform language. It is an interpreter-based and dynamically typed language. With modern features like type hinting, support for asynchronous processing, and functional programming, Python has a very widespread application area.

SQL

SQL, short for Structured Query Language, is a domain-specific language. It is used to manage structured data stored in relational database management systems (RDBMS). The concept of relational databases was introduced by Edgar F. Codd. His relational model is based on relational algebra and relational calculus. In 1970, Raymond Boyce and Donald Chamberlin developed SQL to perform manipulations on data stored in IBM's RDBMS called System R.

Oracle Corp. (known as Relational Software Inc. in its earlier avatar) developed Oracle V2 as its first SQL-based relational database software. Many RDBMS products have come up since then. Some of the other popular relational databases are DB2, Ingres, Sybase, PostgreSQL, and MySQL.

SQL is a declarative language. It consists of statements of different types, such as data query language (DQL), data definition language (DDL), data control language (DCL), and data manipulation language (DML).

SQL specifications are updated from time to time. In 1986, the ANSI standard was published. Since then, periodic revisions have taken place, with SQL:2023 being the latest.

Although SQL is predominantly declarative in nature, different vendors ship their own procedural extensions with their RDBMS product. For example, Oracle's PL/SQL and Microsoft's T-SQL provide the procedural capabilities to their products, Oracle and SQL Server,

respectively. MySQL, an open source RDBMS, comes with SQL/PSM, its ANSI standard procedural extension.

HTML

Today, one cannot imagine the world without the Internet, websites, apps and devices, and so forth. Much of the credit for this revolution goes to Sir Tim Berners-Lee, who introduced most of the technologies that drive the world today. These include HTML, WWW, and HTTP.

While working at CERN Laboratories in 1989, Lee developed HyperText Markup Language, which used tags to design web documents. A website is a server that stores multiple web documents composed in HTML. He also developed a web browser that can interpret the HTML documents and render them on the client's computer screen. Lee also proposed HTTP (Hypertext Transfer Protocol) for the client to request a certain HTML document and the server to transmit the same. HTTP protocol is the foundation of data communication for the World Wide Web.

Tim Berners-Lee also founded the World Wide Web Consortium (W3C) that oversees the development and maintenance of international web standards, such as HTML and CSS. Over the years, the HTML standards have been revised from time to time. The HTML 5.2 standard is the latest, published in 2017.

CSS

Like HTML and JavaScript, CSS (Cascading Style Sheets) is an important constituent of the World Wide Web. CSS, along with HTML, is a declarative computer language. It is a type of markup language, used to define the styling of various elements in a web page. These elements include text color, background and borders, font properties such as type, size, and weight, positioning, alignment, and element padding.

Håkon Wium Lie and Bert Bos are recognized as the co-creators of cascading style sheets (CSS). Incidentally, when he proposed the idea of CSS in 1994, Lie was working at CERN alongside Tim Berners-Lee.

The CSS code is saved as a text file with the .css extension and is included in the HTML script. The style sheet defines a set of styling rules applicable to HTML elements identified by selectors such as id, attribute, and class.

W3C periodically revises the CSS standards. CSS3 is the current standard and was published in 2005.

JavaScript

JavaScript is an important constituent of the Web Platform, the collection of technologies developed by the World Wide Web Consortium (W3C). It is a widely used language for client-side (also called the *front end*) programming in a typical web application. According to one estimate, it is used in nearly 99 percent of websites on the World Wide Web.

JavaScript is a multiparadigm language. It supports imperative, prototype-based, and functional programming styles. One of its most important features is its ability to access and manipulate the web page's Document Object Model (DOM).

Contrary to what many would think, JavaScript is not related to Java. Java is used for developing much wider types of applications such as enterprise-level and desktop applications, and mobile apps. JavaScript is mainly a scripting language that can manipulate the web page behavior. Another key difference is that Java's code execution involves compilation into bytecode and interpretation by the JVM, as opposed to the execution of JavaScript code by a dedicated interpreter engine in the web browser.

While JavaScript is mainly embedded inside an HTML script and executed by the browser, it can also be run outside it. Node.js was developed in 2009 by Ryan Dahl as a JavaScript runtime system built on Google Chrome's V8 JavaScript engine, allowing JavaScript code to run from the command line. It is also used as a powerful server-side scripting environment.

JavaScript was developed by Brendan Eich while working for Netscape in 1995. The following year, the JavaScript implementation

was formalized as the ECMAScript standard. The current version of the ECMAScript standard is ES16, released in June 2025.

In addition to pure (vanilla) JavaScript, developers also prefer to use third-party libraries and frameworks built on top of JavaScript. JQuery, Angular, React, and Vue are some of the notable ones.

Libraries

A program is a sequence of instructions. The programmer uses the predefined keywords in a given language. To execute the program, it must be compiled into object code. Any compiler (or interpreter) understands only the keywords and operators available in the language. However, the program needs to perform many hardware/OS-dependent tasks such as reading input, displaying output, and reading and writing data in files, for which it has to rely on a collection of functions/classes.

Most compilers are shipped with a standard collection of functions or classes that are used very often. For example, a C compiler (such as GCC) has a collection of header files, each having one or more functions. The `stdio.h` header is a collection of library functions that read data from and write data to the console (`printf()`, `scanf()`). The `#include` directive makes these functions available for use.

```
#include <stdio.h>

int main() {
    printf("Hello World!");
    return 0;
}
```

Similarly, Java's `import` statement tells the JVM to use the package when executing the program.

```
import java.io.*;
public class Hello
{
```

```
public static void main(String args[ ])
{
    System.out.println("Hello World!");
}
}
```

However, there will be occasions when a program needs to use a predefined function which is not a part of the standard library. In such a case, these libraries should be installed. You also need to make sure that the installed third-party library is accessible to the compiler (for example by updating the CLASSPATH). In Java, the library packages are distributed as JAR files. In Python, a new library (module) can be easily made available to the interpreter with the PIP command.

```
pip install modulename
```

Once installed, it can be accessed in a Python program with an import statement.

```
import modulename
```

You can even import selected functions from the module.

```
from modulename import function1
```

Different languages have different keywords for the purpose. C# has the using keyword. To load a JavaScript file inside an HTML code, use the <script> tag.

```
<script src="myscript.js"></script>
```

In server-side JavaScript code (as in Node.JS), the require() statement is used to import modules. Modern ES6-compatible JavaScript uses the import statement.

```
import hello from "myscript.js";
```

The module to be imported (here, `myscript.js`) must have an export statement.

```
export default function hello {  
  return "Hello, World";  
}
```

Version Control

Software projects with a large scope are often collaborative efforts, where multiple developers work simultaneously. To ensure seamless coordination among the team members, various version control tools have been developed.

A version control system (VCS) records changes to one or more files over time so that you can fall back to any specific version if required. Different organizations employ different types of version control mechanisms. The two main types are centralized version control and distributed version control.

The centralized version control system (CVCS) works in a client-server environment. A single server stores the various versions of the source code. The server administrator controls the activity of other collaborators. Each team member connects to the server's repository to pull and commit changes. The main drawback of this system is the fact that if the server goes down, there is a risk of losing the team effort. It also requires constant network connectivity. Subversion (SVN) is an example of CVCS.

The distributed version control system (DVCS) is far more reliable and efficient. It has the concept of a remote repository of the code on the server, and its copy on each developer's computer—called a *local repository*. During development, everyone makes changes in the working copy of the repository and commits them to the local repository. The changes in the local repository are then pushed to the central repository. Changes from two or more independent branches are subsequently merged. The merging or integration operation

reconciles multiple changes made to a version-controlled collection of files.

Git is the most widely used version control system today. Developers often talk about Git and GitHub. Although they are different tools, they work together to facilitate modern software development. Git is free and open source software, installed locally on each developer's computer. GitHub, on the other hand, is a cloud-based service that provides a central location for Git projects and adds collaboration features. See Figure [1-10](#).

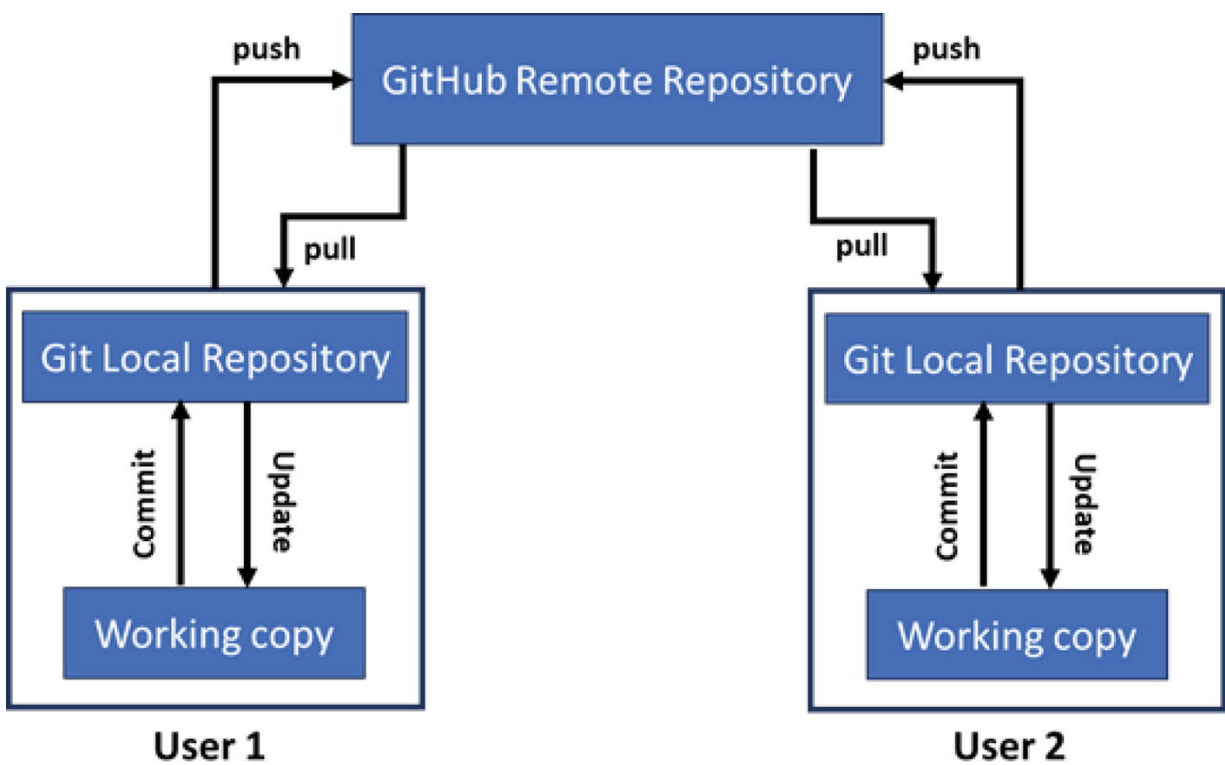


Figure 1-10 Git and version control

Git is a command-line tool. Transactions between the local and central repositories are done through commands like `git clone`, `git add`, `git push`, and `git commit`. Certain GUI tools like Git GUI and GitHub Desktop are also available.

Modern IDEs are tightly integrated with version control tools, particularly Git and GitHub. Instead of using the Git console commands or other GUI tools, you can perform Git operations from within the IDE

(for example, VS Code). Using appropriate extensions, you can configure the IDE to use other version control tools. See Figure [1-11](#).

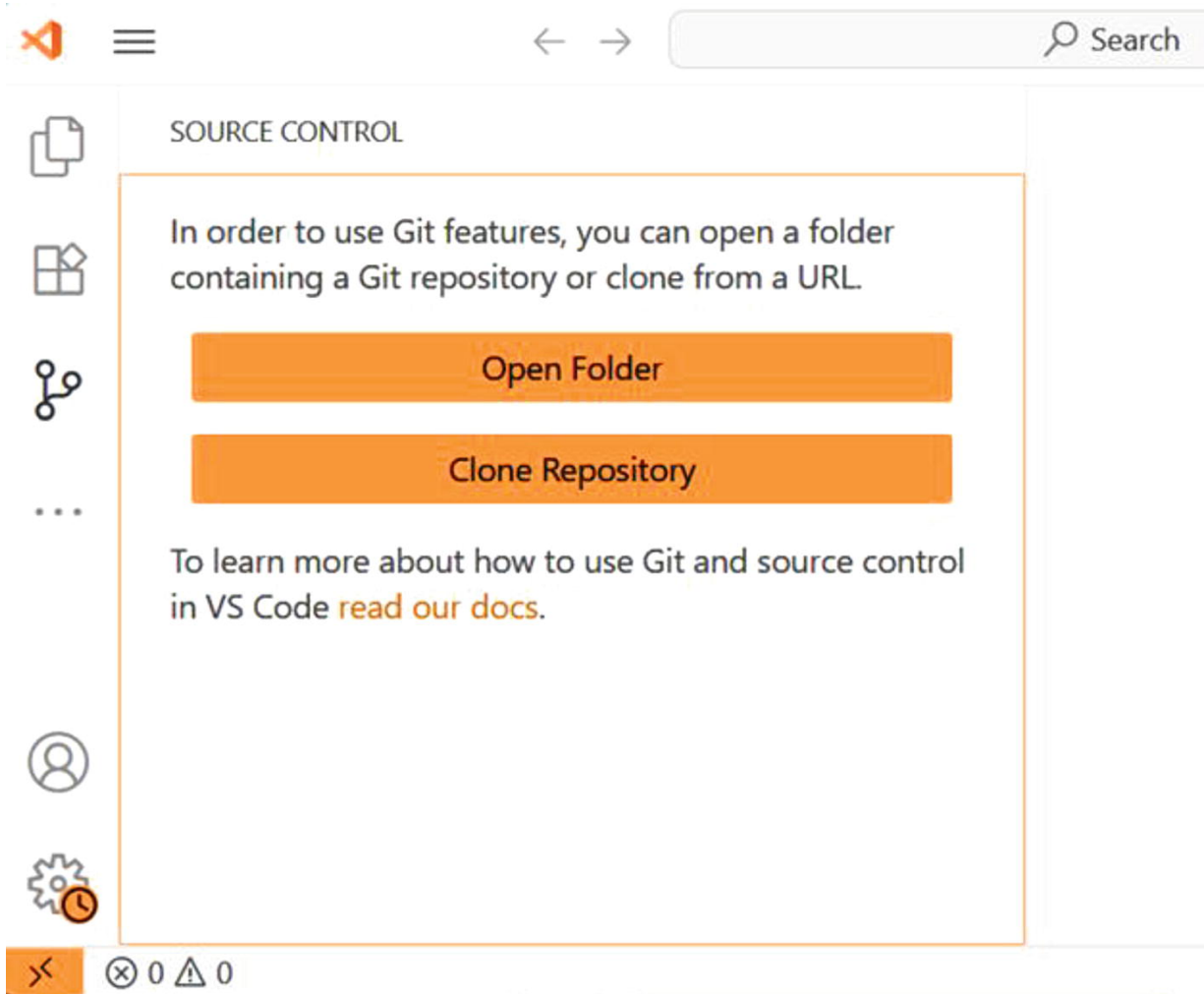


Figure 1-11 VS Code and source control

Software Testing

Developing enterprise software is a very elaborate process. Typical steps in the software development process are usually described as the software development life cycle (SDLC), and software testing is a crucial step. Starting with analysis of the existing system and performing requirement analysis, development goes through various stages such as design and development. Before the software solution is deployed, it needs very rigorous testing.

The testing phase helps developers trace and rectify any bugs in the software and ensure the product's consistency under various operating conditions. Developers employ various testing methodologies for different use cases. These include manual testing or using automated testing tools. Software testers use testing types such as unit tests, integration tests, functional and regression tests.

Various software testing tools are available. They include Selenium, JIRA, and TestRail. Postman app is a popular tool used for API testing.

Points to Remember

- Source code is the sequence of instructions written in a programming language.
- Computer languages are classified into machine language, assembly language, and high-level language.
- A program written in assembly or high-level language is translated into machine code with the help of a compiler or interpreter.
- The interpreter converts one source code instruction at a time, whereas the compiler generates an object code version of the source code.
- A paradigm is the approach a language uses for program development. Different paradigms are imperative, object-oriented, and functional.
- Among the many programming languages, those widely used are C/C++, Java, and Python. HTML and CSS are the scripting languages used in web page development. SQL is used to handle relational databases.
- A source program is saved as a text file. Although any text editor can be used for the purpose, programmers prefer using special software called an IDE.
- Software development is a collaborative process. For better coordination between the members in a development team, version control systems such as Git are widely used.

Summary

This chapter introduces the fundamental concepts of software development that an aspiring developer must be conversant with. Starting with the evolution and paradigms of computer languages, this chapter discussed the purpose of compilers and interpreters. Along with a brief introduction to the leading languages, the role of libraries and version control in software development has also been covered.

The next chapter focuses on the architecture of a web application, elaborating on the web technologies, web standards, and tools for packaging web applications.

2. Web Application Architecture

Malhar Lathkar¹ 

(¹) Near Rachewad Hospital, Nanded, Maharashtra, India

The history of employing computer systems for professional use dates back to the 1950s. Back then, computers were stand-alone machines that only a single user could use at a time. Subsequent advances in networking technology introduced the concept of distributed computing.

In a distributed system, several computer nodes communicate with each other by message passing. Each node has local memory. In addition, they share their resources with others. The nodes are assembled in different topologies such as client-server and peer-to-peer models.

Client-Server Model

The client-server model divides the workload of the application into two constituents. A server is a device that stores all the resources and makes them available for consumption by one or more clients. The client is either a hardware or an application that connects to the server through the network and sends a request for a certain resource. Upon receiving the request, the client application can process the response to provide the suitable output to the user. See Figure [2-1](#).

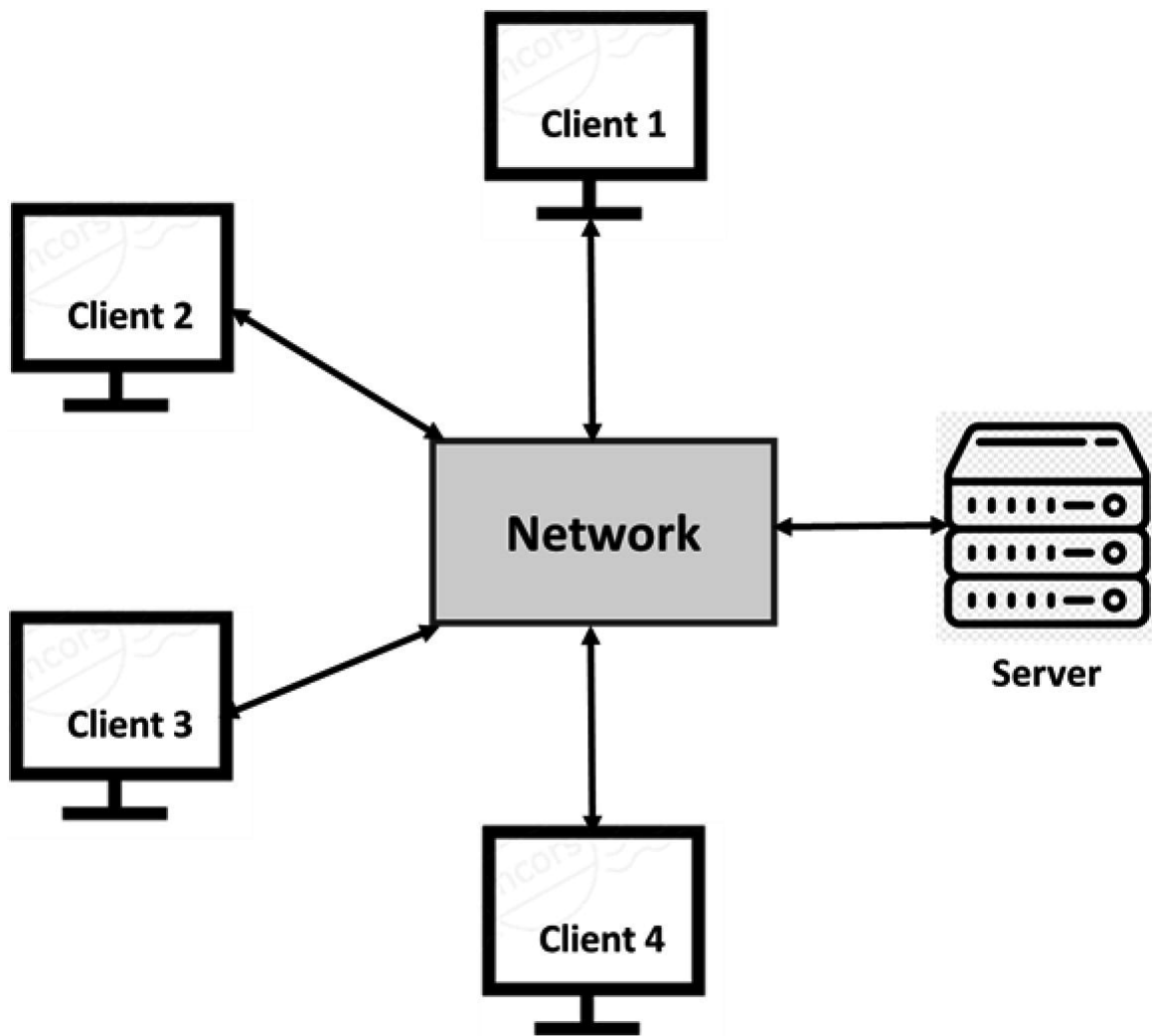


Figure 2-1 Client-server model

WWW

The World Wide Web (WWW) operates on the principle of client computing. This large, distributed information system is a collection of thousands of websites, each containing numerous web pages. The web browser software present on your computer acts as a client. It requests a resource located on one of the servers that hosts the website.

In the client-server model, the communication is initiated by the client. Here, the user enters a URL (Uniform Resource Locator) of the desired resource in the address bar of the web browser software. For instance, consider www.example.com/sample.html as the URL

entered. This request is intercepted by the Domain Name System (DNS) server, which plays a very important role in the function of the WWW.

Each web server in the WWW is assigned a unique IP address (Internet Protocol address). The original IPv4 scheme is a 32-bit number written as four sets of numbers separated by periods, like 192.168.1.1 (each set ranges from 0 to 255). The newer IPv6 (Internet Protocol version 6) scheme uses 128 bits and is a combination of numbers and letters separated by single or double colons. For example, if the IPv4 version of the IP address of a certain web server is 93.184.216.34, its IPv6 representation would be 0000:0000:0000:0000:0000:ffff:5db8:d822.

DNS Server

Naturally, it is not practically possible for the user to remember and use such complex alphanumeric IP addresses. This is where the DNS server comes into play. An IP address is assigned a human-friendly domain name. Here, the IP address 93.184.216.34 is mapped to a domain name example.com, which the user can easily remember. Upon intercepting the URL request from the client browser, the DNS server performs a DNS lookup to return the mapped IP address for the browser to use to connect to the server. See Figure [2-2](#).

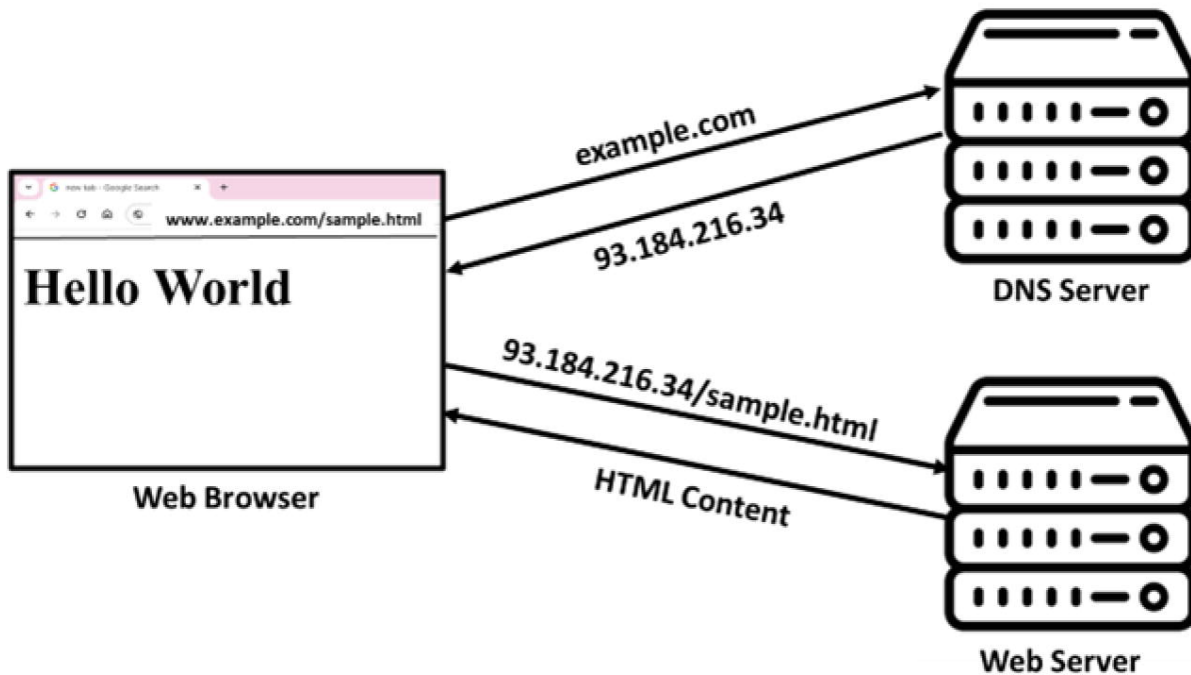


Figure 2-2 DNS server

Today, a web browser is one of the most used software applications as it is the go-to tool for accessing the Internet. Browsers are available on almost all types of devices, such as desktops, laptops, and smartphones. As Figure [2-2](#) shows, the browser receives the raw HTML data from the server. The browser's rendering engine interprets the HTML tags and constructs the Document Object Model (DOM) for the visual representation of the web page. Modern browsers also have a JavaScript engine to handle the programmatic DOM manipulations. Recent enhancements in browser technology include the integration of AI capabilities.

There are many web browser products available. Let's take a brief overview of some of the popular browsers.

Web Browsers

The following describes some of the more popular web browsers. Figure [2-3](#) shows their logos.

Internet Explorer: This was developed by Microsoft as the default web browser in the Windows operating system. However, Internet Explorer 11 is its last version, after which Microsoft discontinued it.

Edge: This is Microsoft's default browser since Windows 10. It was first released in 2015. Newer versions of Edge have AI capabilities as a result of integration with Microsoft Copilot.



Figure 2-3 Web browsers

Chrome: Developed by Google in 2008, Chrome is the most dominant product in the browser market. The V8 JavaScript engine, as part of the Chromium project, has been used to develop the Node.js runtime, which lets you use JavaScript for server-side programming. Recently, Chrome has also added AI mode, which is associated with Google Gemini.

Mozilla Firefox: This browser was developed by the Mozilla Foundation in 2002 as a free and open source web browser. Firefox is considered the successor of the now-extinct Netscape Navigator browser. Although it had a major user base until 2010, it has declined with the advent of Chrome.

Apple Safari: Apple developed Safari in 2003, and it is the default browser, preinstalled in its operating system products including macOS and iOS. It uses Apple's open source browser engine WebKit. There was also a Windows version of Safari, although it was discontinued in 2012. Safari 26.4 is its latest stable release (as of March 2026). It was released alongside the new macOS Tahoe, iOS 26, iPadOS 26, and visionOS 26.

Opera: Developed by a Norwegian company of the same name (Opera), this browser is a multiplatform browser, available for all the major OS (including Windows, Linux, Android and iOS). The Opera browser works on top of the Chromium project. Opera Mobile is its

variant for use with smartphones and tablets. It also has a more lightweight version called Opera Air.

Tim Berners-Lee developed HTML, a scripting language used to build web pages. He also developed a complete working prototype of the World Wide Web with the HTTP protocol, a web browser, and a web server. In 1994, he founded the World Wide Web Consortium (W3C), aimed at standardizing existing web technologies such as HTML and HTTP and developing new technologies such as CSS and JavaScript that would add styling and interactivity to web pages.

Web Standards

HTML, CSS, and JavaScript are the core technologies used in creating websites and web applications. W3C oversees periodic upgrades to the standards and specifications of HTML and CSS. W3C also works in coordination with ECMA International to standardize the use of scripting languages, JavaScript in particular. ECMAScript is a JavaScript standard intended to ensure the interoperability of web pages across different web browsers.

In addition to creating and maintaining web standards for the core web technologies, W3C also promotes their interoperability and recommends guidelines for using JavaScript effectively and accessibly within web documents. W3C is also involved extensively in defining the standards of other technologies that extend web capabilities, including SVG, XML, and JSON.

JSON and XML are the widely used data representation formats for structuring and exchanging data via APIs. JSON (JavaScript Object Notation) is a text-based, lightweight, language-independent format. XML (Extensible Markup Language) is also a text-based, human-readable, and machine-readable format standardized for storing, transmitting, and reconstructing data using self-descriptive, customizable tags. It provides a flexible framework for defining data structures, allowing systems to share information across different

platforms and applications. XML documents have a hierarchical structure based on nested tags.

JSON

Douglas Crockford originally specified the JSON format in the early 2000s. A JSON object is an unordered collection of key-value pairs, enclosed in curly braces {}. The key is a string enclosed in double quotes, followed by a colon. Its associated value can be a string, number, a Boolean (true/false), null, another object, or an array.

```
{
  "name": "John",
  "age": 30,
  "isEmployed": false,
  "Salary": null
}
```

XML defines a set of rules for encoding documents, as defined in the W3C's XML 1.0 Specification of 1998. The main purpose of XML is data serialization, the process of translating a data structure or object state into a format that can be stored or transmitted,) and reconstructed later (the process of deserialization).

XML

XML is a markup language. Like HTML, it also uses XML tags. However, unlike HTML tags, the XML tags are not pre-defined. Much like HTML, an XML document is hierarchically structured. On top of an XML document is a declaration that specifies the version and character encoding used in it. Every XML document must have exactly one root element. It comprises one or more XML elements, and each element is described with its attributes.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<students>
  <student id="S001">
    <firstName>John</firstName>
    <lastName>Doe</lastName>
    <subject>Computer Science</subject>
    <email>johndoe@example.com</email>
    <phone>123-456-7890</phone>
  </student>
  <student id="S002">
    <firstName>Jane</firstName>
    <lastName>Doe</lastName>
    <subject>Electronics</subject>
    <email>jane.doe@example.com</email>
    <phone>123-456-7890</phone>
  </student>
</students>
```

In this example, `<students>` is the root; each student element has the `firstName`, `lastName`, `subject`, `email`, and `phone` attributes.

ES Standards

The era of modern JavaScript was heralded with the introduction of ES6 (ES2015) revision. Many new features have since been added, including web APIs (such as the DOM API for manipulating HTML elements, the Fetch API that allows making network requests, the Geolocation API to access user location, etc.). An API (application programming interface) is a set of protocols that two different software applications follow to communicate with each other. Representational state transfer (REST) is a popular API architecture for HTTP-based data transfer. GraphQL and WebSocket protocols are also used to build APIs.

Here are the major ECMAScript versions of modern JavaScript and the features that they introduced.

ES6 (ES2015) introduced `let` and `const` keywords for variable declarations, arrow functions, classes, and promises.

ES8 (ES2017) added `async/await` for asynchronous programming.

ES15 (ES2024) introduced new features related to grouping, promises, and regular expressions.

ES16 (ES2025) is the latest version.

HTML Living Standard

While W3C is responsible for developing standards specifications for different web technologies, another body, the Web Hypertext Application Technology Working Group (WHATWG), maintains the continuously updated “Living Standard.” Browser vendors like Apple, Mozilla, Microsoft, and Opera (later joined by Google) came together to form WHATWG in 2004. As the name suggests, it maintains a “living document” that evolves with web development and browser implementations.

The HTML Living Standard is a comprehensive document that includes specifications about

- basic HTML elements, their attributes, and their correct semantic use.
 - DOM API for interacting with the document’s structure.
 - APIs such as `Fetch`, `Storage`, and `WebSockets`.
 - how to handle valid and invalid HTML syntax.
 - accessibility of web content.
-

JS Frameworks

Developers are increasingly favoring frameworks instead of the core (or “vanilla”) JavaScript in web development. A software framework is a reusable software environment that facilitates the development of software applications. The generic control flow of the application is already pre-decided. The developer only has to plug in the business

logic into the blocks. Hence, it results in rapid and scalable development.

Specifically, a JavaScript framework is a collection of libraries that provide a foundation for building web applications. The standardized structure of a framework, with ready-to-use components, helps developers work more efficiently and maintain consistency.

Various JavaScript frameworks can be broadly classified into two categories: front-end and back-end. Front-end, or client-side, frameworks focus on building the user interface of a web application that runs in a user's web browser. Back-end, or server-side, frameworks help write the server-side logic and APIs that run on a web server.

Let's take a quick overview some of the popular JS frameworks.

React: First developed at Facebook in 2011 and now maintained by Meta, React is the most popular JS tool for building dynamic UI for a web application with reusable components. Its notable features include a declarative UI that simplifies UI development significantly and the use of a virtual DOM that optimizes UI updates, leading to better performance. React uses JSX (JavaScript XML) to write HTML-like code directly within JavaScript, which makes component creation more intuitive and readable.

Angular: Originally named AngularJS, Google did a complete rewrite in 2014 and renamed it Angular. It is a free and open source single-page web application framework, based on TypeScript (a superset of JavaScript developed by Microsoft). Angular's component-based architecture makes it easier to manage and test individual pieces of an application. Dependency injection, two-way data binding, and extension of HTML with additional attributes called *directives* are some of the notable features of Angular.

Vue.js: Created by Evan You in 2014, Vue.js is an open source JavaScript framework based on the Model-View-ViewModel (MVVM) architectural pattern. This focuses on declarative rendering and component composition. While its core library focuses only on the view layer, it also supports advanced features such as routing and

state management. Vue.js keeps evolving; its latest 3.5 version was released in September 2024.

Node.js: Technically, Node.js is not a framework; it's a runtime environment built on top of the V8 Chrome engine for executing JavaScript code on a server. Node.js powers many server-side frameworks, including Express.js and NestJS.

Express.js: Recognized as a minimalist back-end web framework, Express.js is free and open source software. It is the de facto standard server framework for Node.js, designed for building web applications and APIs. It was first released by TJ Holowaychuk in May 2010, and the latest stable version is 5.1.0, released in March 2025.

NestJS: Node.js also powers the NestJS framework. It is a progressive framework that developers can use to build efficient, reliable, and scalable server-side applications. Its modular architecture is largely inspired by Angular. Kamil Myśliwiec published its initial version in February 2017, while its latest stable version (11.1.6) was released very recently in August 2025.

Next.js: Although built on top of React, Next.js is a hybrid framework, as it helps you to use both server-side and client-side rendering strategies. server-side rendering (SSR) is one of its prominent features. Because pages are rendered on the server at request time, this helps improve SEO and initial load performance. At the same time, another feature, Static Site Generation (SSG), causes the pages to be pre-rendered at build time. This is especially useful in cases where the content changes not so frequently, and results in extremely fast load times. Next.js was developed by Vercel, which is also responsible for its maintenance and ongoing development.

Single-Page Applications

Traditional web page applications, especially before the advent of modern JavaScript frameworks, were collections of multiple web pages, hyperlinked with each other. In fact, this approach is still very much in

use even today. However, adopting single-page applications has become more prevalent recently. Why?

In a traditional multi-page website, each navigation results in a new page being loaded, which often makes the application very slow. The SPA (single-page application) takes a different approach. As the name indicates, the application has only one page whose content keeps updating dynamically as the user interacts with it. The browser loads the page only once, and each user interaction refreshes only a part of the page.

In a typical web application, the back-end activity is managed by a server-side programming language like Java, PHP, or Python to fetch the required data from a database and render it into a web page with a new URL for the browser to load.

The SPA uses client-side rendering, primarily powered by JavaScript. When a user first visits such an application, the browser downloads all the required resources (HTML, CSS, and JavaScript files), or the entire application. Upon any user action (such as clicking a link or submitting a form), it invokes a certain JavaScript function. Unlike applications powered by Java/PHP/Python, it doesn't request a new HTML document from the server; it uses an API to fetch only the new data it needs. The fetched data is used to update the relevant parts of the DOM on the current page, without reloading the whole page. See Figure [2-4](#).

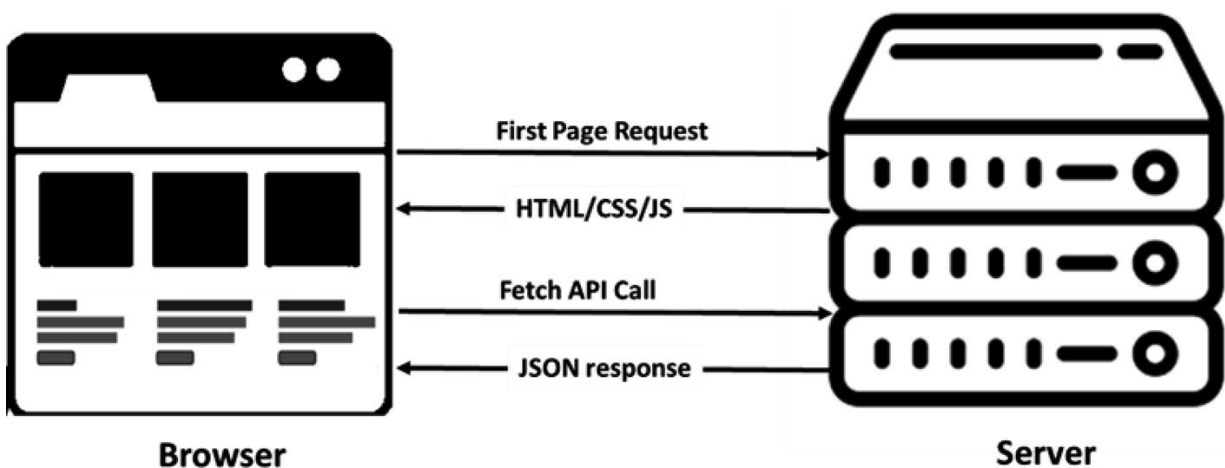


Figure 2-4 Single-page application

Different frameworks use different techniques for updating the components of the web page in an SPA. For example, React's virtual DOM helps with dynamic updates to page components, thereby improving the performance of the application UI.

API

Broadly speaking, the word interface refers to interaction between two isolated and independent environments on a predetermined set of rules and protocols. Your computer's CPU communicates with external devices such as printers, the camera, a mouse, and monitors through interfaces such as USB and HDMI. Likewise, an API lets two different software applications communicate with each other.

Today, web developers are embracing the API-first approach, thanks to advances in modern JavaScript and the emergence of efficient web frameworks. This is at variance with the traditional code-first approach. Let's try to understand the difference between the two.

In the code-first approach, development starts with back-end business logic and a database design suitable for it, and then the front end is designed accordingly. For example, a function (in Java, PHP, or Python) assumes roll number as its formal argument, fetches the marks from the database, computes and returns the result. Then comes the front-end design, using HTML and CSS to display the result.

In the API-first approach, the application development process starts with designing an API contract between the front end and the back end, agreeing beforehand on the data structure to be exchanged between the two. Back-end and front-end development can now proceed independently by mocking the API. This approach makes the application easily scalable and ensures better integration. See Figure [2-5](#).

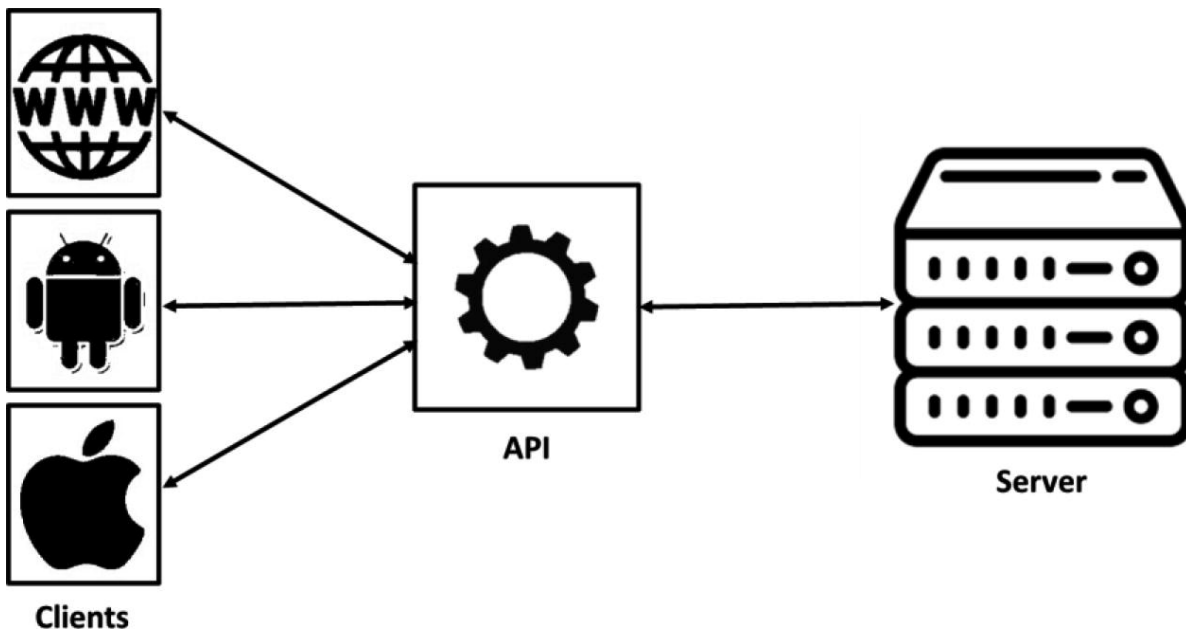


Figure 2-5 Application programming interface

Examples of APIs for fetching and processing real-time data are abundantly found around us. The API servers collect, aggregate, and process the real-time data related to some domain (such as weather, stock market, demography, etc.) and expose the data through their APIs. Here are some examples of front-end apps relying on the APIs.

Weather apps: Weather data providers such as OpenWeatherMap, Meteomatics, and AccuWeather disseminate information about the current conditions, forecasts, and alerts in XML/JSON data format through APIs, which the weather apps call through their UI.

Hotel booking apps: These apps use APIs provided by various aggregators of hotel accommodation data, such as the Amadeus Hotel API and the Hotelbeds API, to facilitate hotel booking services for their customers.

Stock trading apps: These apps are powered by APIs that stream real-time stock market data. In the United States, Interactive Brokers (IBKR) offers several APIs on top of which many apps operate. Another example is the Tradier API that provides access to market data. In the Indian context, services like Zerodha, ICICIdirect API, and Groww are prominently used.

Over the years, API design technology has seen different architectural styles. SOAP (Simple Object Access Protocol) and RPC (Remote Procedure Call) are some of the earlier approaches. Today, REST and GraphQL APIs are prominently used.

REST

Representational State Transfer, or REST, is one of the most widely used API architectures. REST APIs are based on a set of design principles, also called *constraints*, first recommended by Roy Fielding in the year 2000. These constraints define how a web application should function. The HTTP protocol is the backbone of REST, and it prescribes communication between the client and the server through the HTTP methods GET, POST, PUT, UPDATE, and DELETE.

The REST architecture lays down certain constraints to be followed in the design of the REST API. It emphasizes the concept of statelessness, where each request from client to server contains all the information needed to understand the request. A uniform interface requires that clients request resources on the server using a URI (Uniform Resource Identifier). The data is commonly exchanged in formats like JSON or XML. A subsequent chapter in this book takes a detailed look at the REST architecture and its constraints.

GraphQL

GraphQL API architecture is based on a query language (Graph Query Language). It was developed by Facebook in 2012. GraphQL today is a new API standard that provides a more efficient, powerful, and flexible alternative to REST. One limitation of REST APIs is that they can overfetch data from the server; GraphQL addresses this issue.

The GraphQL API is designed with GraphQL Schema Definition Language. The GSDL is used to define its schema, which outlines the available types and fields. The behavior of each field is then determined by its corresponding resolver functions, which can read (query), write (mutation), and subscribe to receive notifications of data changes. The resolvers are the root operation types. The object types are data

structures with fields of scalar types (numeric and string) as well as other objects. See Figure [2-6](#).

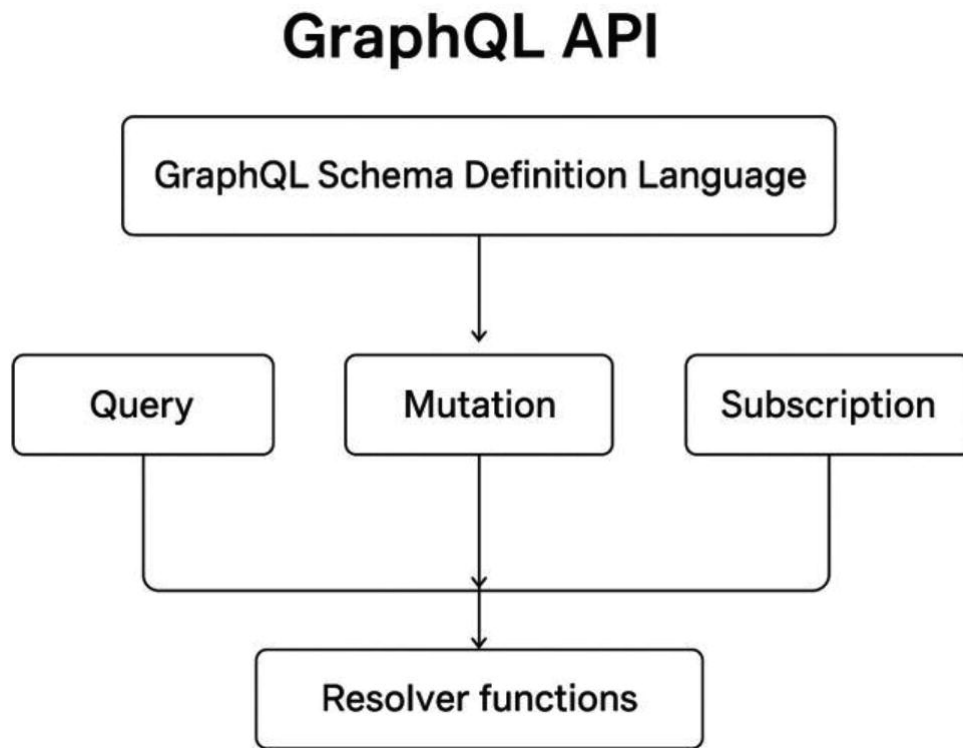


Figure 2-6 GraphQL architecture

In addition to Facebook, many other public API providers such as GitHub, Shopify, and Google Directions API use GraphQL.

Databases

Web applications are invariably data-driven. The data generated by the user needs to be persistently stored and retrieved in response to the user's queries. Most server-side languages like Java, Python, and Node.js can perform disk I/O operations to save and retrieve data to and from files. However, file operations are unstructured and hence suffer from disadvantages such as redundancy and lack of integrity.

To overcome these drawbacks, Dr. E. F. Codd proposed a relational model of data organization. Based on this relational model, various software products called *relational database management systems*

(RDBMS) have been developed. The two important concepts of a relational database are entity and relation.

In a relational database, the data is organized in tables, also known as *relations*. A table is a collection of one or more rows, sometimes called *rows* or *tuples*. Each row represents an instance of a certain entity. For example, a products table has many rows, each row describing a product. An entity is characterized by its attributes or fields. Each column of the relational table represents a field.

A table usually has one column (attribute) which has a unique value for each row. Such an attribute is called the *primary key*. When the primary key of one table appears as one of the attributes in another, it becomes a foreign key. These keys are used to establish relationships between the tables. They are also important for maintaining data integrity and avoiding data redundancy. See Figure [2-7](#).

	Name	Data type	Primary Key	Foreign Key
1	ProductID	INTEGER		
2	ProductName	TEXT		
3	Price	NUMERIC		

Products Table

	Name	Data type	Primary Key	Foreign Key
1	CustomerID	INTEGER		
2	CustomerName	TEXT		
3	ProductID	INTEGER		
4	Quantity	NUMERIC		

Customers Table

Figure 2-7 RDBMS

The data from the tables of relational databases is managed using SQL (Structured Query Language). SQL is a declarative language, developed by Donald Chamberlin and R. F. Boyce. Oracle, MySQL and its open source alternative MariaDB, MS SQL Server, SQLite, and PostgreSQL are some of the popular examples of relational databases.

Although relational databases are predominantly used by most software applications (web apps as well as stand-alone apps), they are found to be inefficient particularly in situations when the database schema is flexible, and the volume is huge. This is where NoSQL databases are used.

One of the most important features of a NoSQL database is its flexible schema. In a relational database, a table has a fixed, pre-defined structure. In a NoSQL database, each entity may have a variable number of attributes. NoSQL databases are of different types. MongoDB, the most popular NoSQL database, is a document-store database. Redis and Amazon DynamoDB are of the key-value store type. Cassandra and HBase are known as *column-oriented databases*. Newer versions of PostgreSQL have certain NoSQL features, allowing you to construct a hybrid database.

Another type of NoSQL database is in the limelight nowadays. It is called a *vector database*. They can handle vector embeddings, which are numerical representations of unstructured data like text, images, and audio. Rather than relational operations, embeddings are meant to perform similarity search. LLM-powered apps especially use vector databases. Chroma, Milvus, and Weaviate are some of the popular vector databases.

NoSQL databases are non-relational. Hence, it is not possible to use SQL for data manipulation operations (hence the name NoSQL). However, most NoSQL databases have their own version of query language. MongoDB Query Language (MQL) is the proprietary language that lets you interact with MongoDB databases. There is also the Cassandra Query Language (CQL).

Packaging Web Applications

After the application goes through various stages of development, such as design, programming, debugging, and testing, it needs to be packaged in a deployable form. A package contains all the dependencies of the application, including the code, the libraries and frameworks used, and the assets such as images.

If the front-end layer of your web application has only the raw HTML pages and the associated CSS and JavaScript files, they can be manually uploaded to the production server. However, if you are using

certain frameworks such as React or Angular, it is packaged using tools such as Webpack, Vite, or Parcel.

The back-end packaging tools depend on the server-side technology used. In traditional Java applications that use servlets and JSPs, the sources are bundled in a WAR (Web Application Archive) format. Modern Java web applications are developed with the Spring Boot framework. Tools like Maven and Gradle are used to create executable JARs (Java Archive) that can be distributed and deployed.

Python web applications are built with frameworks such as Django and Flask. Python's PIP utility is used to build a package that can be deployed.

Node.js applications generally define dependencies in a package.json file and then use tools like npm to create the distribution bundle.

The emergence of tools for container image creation, such as Docker, has changed how web applications are packaged. A container image bundles the application and all its dependencies into a single, isolated unit.

WebAssembly

As the name indicates, WebAssembly (wasm) aims to bring assembly-language programming to the World Wide Web. The wasm code is executed by the web browser on the client's machine. WebAssembly is hardware-independent as compared to assembly language.

WebAssembly acts as a portable compilation target. Its low-level, binary instruction format makes it possible to run C, C++, Rust, Go, or Python code on the browser alongside JavaScript.

Because of its binary format, wasm code execution is extremely fast. This high performance makes WebAssembly highly preferable for developing game engines, such as Unity and Unreal Engine, to run in the browser.

Points to Remember

- In a client-server model, the server stores all the resources and makes them available for one or more clients. The World Wide Web implements the client-server model of computing.
- WWW is a large, distributed network of servers that stores websites. Web browser software acts as the client. Google Chrome, Microsoft Edge, and Apple Safari are some of the popular browsers.
- A website is a collection of multiple web pages. HTML is used to construct the web page, whereas CSS helps in defining the styling properties of the web page elements. JavaScript is a popular scripting language that allows dynamic manipulation of the web page's DOM.
- HTML, CSS, and JavaScript are known as *web technologies*. The World Wide Web Consortium (W3C) oversees the development, upgrading, and maintenance of the standards for these technologies.
- Data shared by the user and generated during interaction is stored in a back-end database. Relational databases like Oracle and MySQL are the most commonly used database solutions. In applications where the database schema is flexible, NoSQL databases like MongoDB are preferred.
- Modern web application development emphasizes the API-first approach. Web APIs make the development process fairly flexible and scalable. The advances in modern JavaScript and JS frameworks have greatly contributed to the popularity of the API-first approach.
- REST architecture is today the de facto standard for API development. The trend of using GraphQL-based APIs is gaining popularity because of its flexibility.
- Another new trend in web application development is single-page applications. SPAs are powered by vanilla JavaScript or JavaScript-based frameworks such as React and Angular.
- Distribution and deployment are important steps in the development process. Different technology-specific tools are used for packaging. These include Vite and Vercel for client-side codebases, WAR and JAR

for Java-based server-side applications, PIP for Python, and npm for Node.js.

- WebAssembly is a powerful technology that lets assembly code run inside the client's browser alongside JavaScript.

Summary

This chapter discusses the important constituents of a web application, including client-side development tools like JS frameworks, types of APIs and databases used in the server-side part of the application, and approaches for packaging the web application for easy distribution.

This background makes it easier to understand the technical aspects of a web application, such as the features of the HTTP protocol and the REST architecture. This is discussed in the next chapter.

3. Anatomy of the HTTP Protocol

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

The giant structure of the World Wide Web is built on top of the HTTP protocol. Moreover, it is also the foundation of REST—the most popular API architecture. Let’s dive a little deeper into the concepts of the HTTP protocol and the REST architecture, particularly its constraints.

The Internet is made up of a large number of interconnected computer networks, spread across the globe. The communication between networks is governed by the Internet Protocol suite (TCP/IP). The TCP/IP model consists of four distinct layers below the user’s application. These layers are, starting from the bottom, the network access layer, the Internet layer, the transport layer, and the application layer at the top.

The application layer contains different protocols, including HTTP (HyperText Transfer Protocol), FTP (File Transfer Protocol), (SMTP (Simple Mail Transfer Protocol), and DHCP (Dynamic Host Configuration Protocol).

It is worth reiterating once again that HTTP is the foundation of data transfer in the client-server architecture in general and the World Wide Web in particular. The HTTP protocol is designed to send hypertext documents from the server in response to the client’s request. Hypertext documents include hyperlinks to other resources, thereby creating an interconnected “web” of documents a user can surf through.

The specifications of the HTTP protocol were formulated by Tim Berners-Lee, who is also credited with the development of HTML and a web browser. The Internet Engineering Task Force (IETF), in association with the World Wide Web Consortium (W3C), works toward ongoing upgrades of the HTTP protocol. The HTTP 1.0 standard was released in 1996. Subsequently, HTTP/2 and HTTP/3 versions were published in 2015 and 2022, respectively.

Key Principles of HTTP

The HTTP protocol that powers Internet communication is built upon certain important principles. Despite subsequent revisions in its specification, its core principles remain the pillars on which it works. Here’s a brief overview of the key principles.

Client-server model: HTTP, and hence the WWW, follows the client-server architecture of the computer network, where the web browser (acts as the client) sends a request, the server processes it and sends back the response for the browser to consume. Although a browser is the most common example of an HTTP client, you also find mobile apps, API testing tools, and web crawlers acting as clients. Performing various functions like caching, filtering, or load balancing, the proxies sit between the client and the server.

Request-response cycle: The client-server interaction goes through a series of steps, repeated for each request by the client. Each request message includes an HTTP verb (or method) implying the type of operation to be performed on the request, one or more headers, and the body of the request. The response returned by the server also contains the body, along with headers and a status code telling how the request was treated.

Stateless: Statelessness is a very crucial feature of the HTTP protocol. It requires each request to be executed independently, and the server should not retain details of any previous requests received from any client. This feature makes the server design simpler, scalable, and efficient. As per the specifications of HTTP 1.0, the connection between the client and the server had to be disconnected. However, later versions introduced reusable and persistent connections.

Media independent: HTTP is a media-independent protocol; hence, any type of data can be transferred as long as both the client and server support it. The server includes the Content-Type header in its response, specifying the MIME (Multipurpose Internet Mail Extensions) type of the data being sent. For example, the MIME type text/html is used for web pages, whereas application/pdf is used for PDF documents. This helps the client understand what type of response it receives and process it accordingly.

Extensible: The provision to include headers in the request as well as response makes the HTTP protocol highly extensible, making it possible to add functionality like cookies, session management, and authentication.

Request-Response Cycle

Under the HTTP protocol, the communication between the client and the server follows a request-response cycle. The web browser (also known as a *user agent* because it is responsible for retrieving the server's response) initiates the request. The request-response cycle goes through the following steps (also see Figure [3-1](#)).

1. **DNS lookup:** The DNS server maps the request URL to its IP address.
2. **Connection:** The client then uses this resolved IP address to establish a TCP connection with the web server (using port 80/443 for unencrypted/encrypted connection, respectively). The handshake between the two provides a channel for data transfer.

Figure 3-1: Request-Response Cycle

3. **Request:** The request is then processed by the server corresponding to the resolved IP address. The request data contains components such as the HTTP method (GET, POST, etc.), headers that pass additional metadata about the request, and the request body, usually the data submitted by the user in an HTML form.
4. **Server-side processing:** Upon the receipt of the request, the server either accepts or declines it. If accepted, the server obtains the client's IP address, parses the request data, and processes it, usually depending upon the request method. Finally, it formulates the HTTP response to be returned to the client.
5. **Response:** After processing the request, the server sends a message back to the client as the response. Similar to the request, the HTTP response may contain headers (such as cookies), a status code indicating how the request was handled, and the response body.
6. **Client-side processing:** The server's response is either a static HTML file or a raw HTML string. The client browser accepts the response and interprets it with its HTML engine. The response may also include references to other resources such as CSS or JavaScript code, which the browser uses to render the result.
7. **Disconnection:** HTTP is a stateless protocol. Hence, after the request is processed, either the client or the server terminates the TCP connection between the two.

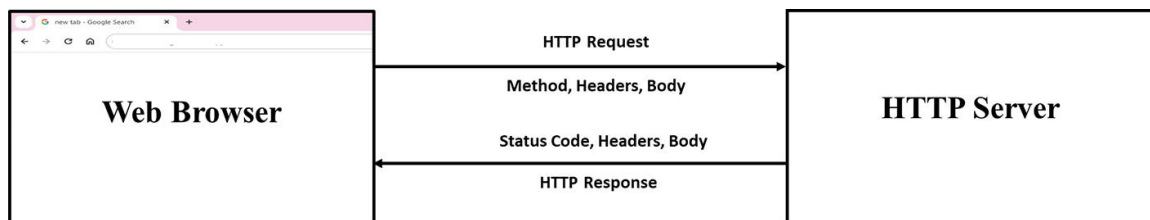


Figure 3-1 Request-response cycle

Request Methods

The client's request includes a request line, with the HTTP method, along with headers and the request body. It also states the version of the HTTP protocol being used, usually HTTP/1.1. A typical example of an HTTP request sent by a browser may include additional headers after the request line.

```
GET /test.html HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

A well-defined nomenclature of methods or verbs is an important aspect of the HTTP protocol, as the choice of a method indicates what type of action the client expects the server to perform on the requested resource. Each HTTP method (GET, POST, etc.) has its own syntax, but some characteristics such as idempotency, safety, and cacheability may be similar across multiple methods.

Let's discuss the different HTTP methods.

GET Method

By default, a request should use the GET method, whose purpose is to retrieve a certain resource identified by the given URL. The GET request doesn't have any body component. Since it intends to fetch or retrieve a resource and is not meant either to create or modify a resource, GET is considered a safe method.

For example, when a web browser requests <http://example.com/test.html>, it sends a GET request to fetch test.html from the server.

Similarly, the URL <http://example.com/test.html?name=John&age=25> also sends a GET request. A query string that includes one or more parameters in the form of key-value pairs is appended to the URL after the question mark (?).

The GET method is idempotent. A request method is considered idempotent if it is executed multiple times with the same input, it produces the same effect on the server.

POST Method

Using the POST method in an HTTP request implies that the client intends to create a new resource on the server. Obviously, this type of request needs to include a body that contains the attributes for a new resource to be created.

The user interaction starts by opening a web page that has an HTML form with its method attribute set to POST. The HTML form filled and submitted by the user becomes the request body. The encoded form data is sent along with the request to the URL set as the form's action attribute.

```
<form action="/users" method="POST">
  <div>
    <label for="username">Username:</label>
    <input type="text" id="username" name="username">
    <label for="email">Email:</label>
    <input type="email" id="email" name="email">
    <label for="password">Password:</label>
    <input type="password" id="password" name="password">
  </div>
  <button type="submit">Create Account</button>
</form>
```

Here, the form sends a POST request with the user's data to a /users endpoint.

```
POST /users HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 72

{
  "username": "A001",
  "email": "user@example.com",
  "password": "password123"
}
```

The POST method is neither safe nor idempotent. Not safe because it changes the state of the server, and not idempotent, as sending the same POST request will create multiple new resources.

PUT Method

The PUT method intends to update the attributes of a specified resource on the server. Hence, as with POST, the PUT request should also include a request body, along with the request set to PUT, headers like Content-Type and Content-Length, and the HTTP version. Similar to POST, the request body is populated with the user-submitted data in an HTML form.

The PUT request meant to update the email address of the user A001 would be as follows.

```
PUT /users/123 HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 72

{
  "username": "A001",
  "email": "amar@example.com",
  "password": "password123"
}
```

The PUT method is idempotent but not safe. This is because multiple identical requests have the same effect as a single request, and it updates an existing resource and modifies the server's state.

DELETE Method

As you would expect, the DELETE method in the HTTP request line asks the server to remove the specified resource. The request line usually specifies the Authorization header that carries the client's credentials.

```
DELETE /users/A001 HTTP/1.1
Host: example.com
Authorization: Bearer <Your-Auth-Token>
```

This request line asks the server to delete the user resource with username A001. The DELETE method is idempotent, but it is not safe. Once the resource is deleted, sending the same DELETE request will not change the server's state. Obviously, it is not safe because of its very purpose: to delete a resource.

These four request methods (GET, POST, PUT, and DELETE) are particularly important in web applications, especially RESTful APIs, as they handle the basic CRUD (create, read, update, delete) operations. In addition to these, the HTTP protocol also defines other methods, although they are used less frequently.

PATCH: This method is used for partial modifications to a resource. Compared to PUT, the PATCH method is more efficient for updating specific fields. Note that the PUT method replaces the entire representation of a resource.

HEAD: This method retrieves the headers for a resource, but not the body. It is particularly useful for checking a resource's metadata before downloading it.

OPTIONS: This method is often used to query a server for which methods (e.g., GET, POST, PATCH) are supported on a specific URL. This is especially important for handling cross-origin resource sharing (CORS).

Request Headers

The request and response messages in HTTP data exchange are in string format. One component found in both is the list of headers. Including headers in either facilitates transfer of additional metadata between a client and a server. Headers provide valuable context about the request/response body, such as the content type, the client's credentials, and the type of data being transferred. Both types of HTTP headers are sent as key-value pairs along with other parts, such as the method (in the request) and status code (in the response). Each header consists of a case-insensitive Header-Name followed by a colon (:) and then its corresponding Header-Value.

```
Header-Name: Header-Value
```

The end of the header section in the message is indicated by an empty line. Note that some headers appear in request as well as response messages.

The following are some of the most frequently used request headers.

Host: It specifies the domain name of the server along with the optional port number.

```
Host: www.example.com
```

User-Agent: It identifies the client's browser, operating system, and version. The server uses this metadata to send the content tailored to the client's capabilities, like

displaying a mobile-optimized version of a site.

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0
Safari/537.36
```

Accept: It tells the server what MIME types the client can process so the server can deliver the response in the most suitable format.

```
Accept: text/html, application/json, */*
```

Accept-Encoding: It specifies the list of compression algorithms the client supports.

```
Accept-Encoding: gzip, deflate
```

Accept-Language: It lists the human languages preferred by the client. This allows the server to deliver content in the user's language, if available.

```
Accept-Language: en-US
```

Content-Type: The POST and PUT requests have a body section, in which case the Content-Type header specifies the media type of the body data being sent. Note that this header is also used in responses.

```
Content-Type: application/json
```

Connection: This header manages persistent connections. If set to keep-alive, a client can make multiple requests without establishing a new TCP connection each time. The close value instructs the server to close the connection after the response.

```
Connection: keep-alive
```

Authorization: It is important for authenticating the client's credentials. It is typically used to access protected resources and includes authentication details, such as a token or username/password.

```
Authorization: Basic QWxhZGRpbjpwGVCuIHNLc2FtZQ==
```

Cookie: Cookies play a crucial role in web application development. Cookie is a request header that sends cookies set previously by the server (as a result of the Set-Cookie response header) back to the server.

```
Cookie: sessionId=12345; username=A001
```

HTTP Response

The previous section discussed what constitutes the HTTP request initiated by the client (i.e., a web browser). Let's now turn our attention to the other side: the HTTP response sent by the web server in reply to the client's request.

The HTTP response string has three main components: a status line, a list of response headers, and the message body.

The status line includes the protocol version (such as HTTP/1.1), a three-digit status code, and a short textual description of the status code.

Similar to the request headers, the HTTP response also uses headers to provide additional metadata about the response and the server.

As with the request, the body part of the response is also optional. It could be an HTML string, JSON data, an image, a file, or so forth, as identified by the Content-Type header.

Status Code

The server puts a status code in its response to let the client know how its request has been handled—whether it has been successfully processed, declined, the requested resource couldn't be found, or the server encountered any problem while doing so.

The status code is a three-digit number, each associated with a corresponding descriptive text. The HTTP protocol defines five categories or classes of the status codes, depending upon the nature of the responses. The first digit defines the class of response.

1xx Informational Response

Status codes from 100 to 199 are informational, indicating that the request was received and understood. Normally, they are not set directly. Responses with these status codes don't have a body. It is issued on a provisional basis, asking the client to wait for a final response.

The following are common informational status codes.

100 Continue: The server has received the request, and the client should proceed to send the request body.

101 Switching Protocols: As requested by the client, the server is changing the protocol.

102 Processing: It indicates that the server is processing the request, but no response is available yet.

2xx Successful Response

The category of status codes from 200 to 299 indicates the action requested by the client was accepted and completed successfully.

200 OK: This status code is the standard response for successful HTTP requests. For a GET request, the response includes the requested resource (such as a file), whereas

for the POST request, the response contains the result of the action.

201 Created: Usually occurs in a POST request, to let the client know that a new resource has been created.

202 Accepted: This status code implies that the request has been accepted for processing, but the processing has not been completed.

204 No Content: Another frequently encountered status code, which means that the server processed the request and is not returning any content.

3xx Redirection Response

If the response is in this class of status codes, it implies that the client needs to take certain additional action to complete its request. The status codes from 300 to 399 fall in this category. URL redirection, a technique commonly used to make a web page available under more than one URL address, is one use case where redirection response codes are used.

300 Multiple Choices: A response with this status code indicates that multiple options exist for the client to choose to access a certain resource.

301 Moved Permanently: Lets the user agent know that all requests should be directed to the given Unique Resource Identifier (URI).

302 Found: With this status code, the server asks the client to go to another URL, because the requested resource has moved temporarily elsewhere.

4xx Client Error Response

The 400–499 **response** status codes are reserved for reporting different errors in the client's request. Any type of request method can elicit a client-caused error response.

404 Not Found: This status code is frequently encountered whenever the requested resource could not be found (however, it may be available in the future).

400 Bad Request: Occurs if the server is unable to process the client's request because of reasons like malformed request syntax, or size too large, and so on.

403 Forbidden: Again, a very common status code, encountered even if the request is valid; the server understands the request but refuses to act on it, possibly because the user does not have the necessary permissions.

5xx Server Error Response

The class of response status codes from 500 to 599 represents the server's failure to fulfill the client's request. The server's response includes an explanation of the error and whether it is temporary or permanent. These response codes apply to any request method.

500 Internal Server Error: The server emits this response whenever it faces an unexpected condition.

503 Service Unavailable: It is raised if and when the server cannot handle the request (in most cases because it is down for maintenance), and is often a temporary

issue.

505 HTTP Version Not Supported: The server reports this response if it does not support the HTTP version used in the request.

501 Not Implemented: Usually, this status code implies that the request cannot be fulfilled now, but possibly in the future when a new feature of a web-service API will be added.

Response Headers

The server response also includes headers, just as the request does. A header is defined with the same syntax (i.e., it's a key-value pair, separated by a colon, and ends with a newline).

Many headers are common in requests and responses. Here are some of the response-only headers.

Set-Cookie: A response header that sends a cookie from the server to be stored in the user's browser. With each subsequent request, the browser sends this cookie back to the server.

```
Set-Cookie: session_id=abc123xyz; Max-Age=86400; Secure
```

Server: By including this header in the response, the web server discloses the software and its version used for handling the request.

```
Server: Apache/2.4.10 (Unix)
```

Accept-Ranges: This header indicates whether the server supports partial content requests. If its value is bytes, it means the server can serve partial content, so that the client can request a specific byte range of a resource. If this header's value is none, the server doesn't support partial content requests.

```
Accept-Ranges: bytes
```

WWW-Authenticate: This header is usually included when the status code is 401 Unauthorized, which asks the client for authentication.

```
WWW-Authenticate: Basic realm="Access to the staging server",  
charset="UTF-8"
```

This forces the client to send an access token and issue its next request.

Location: This header is used in redirection responses (the 3xx status codes) and informs the client of the new URL for its requested resource.

```
Location: https://www.example.com/new-url
```

Retry-After: often used with a 503 Service Unavailable status code, especially if the server is under maintenance; this response header defines the time period for the client to wait before resending the request.

Retry-After: 300

Cookies

Cookie (request header) and Set-Cookie (response header) were explained in the preceding sections. But what are cookies? Cookies play a crucial role in web application development.

One of the important properties of the HTTP protocol is that it is stateless. It means that each request from a browser to a server is independent, and the server doesn't hold on to the details of previous requests. While this simplifies the server design, developers felt the need to provide some mechanism by which the server can remember user-specific data between requests. Hence, the cookie technology was devised as a smart workaround for HTTP's statelessness.

The cookie mechanism is a two-step process, facilitated by the following two headers.

Set-Cookie response header: When the client visits a website for the first time, the server adds a Set-Cookie header to its response. This header contains a name-value pair (e.g., sessionId=abc123) and optional attributes that specify the cookie's life and the domain. This cookie data is stored as a text file on the client's machine.

Cookie request header: Thereafter, whenever the client visits the same domain, the client browser automatically pushes the stored cookie in a Cookie header in its request. The server can then retrieve this cookie data to identify the user and recall their state.

This cookie mechanism lets you add several features that otherwise wouldn't have been possible because of the stateless nature of the HTTP protocol. The following are some features.

Session management: To retain the session-specific data of the logged-in user, such as a shopping cart.

Personalized responses: Cookies help the website remember the user's preferences and usage patterns, based on which it can serve the relevant content.

Tracking: Tracks and analyzes the usage data, especially used by e-commerce and social media sites to push products and advertisements based on the user's historical data.

Cookie Attributes

The Set-Cookie response header pushes the cookie data into the client's machine. Along with the name-value pair, you can set a cookie with the following attributes.

Expires/Max-Age: To define the cookie's lifetime, use Max-Age to specify a duration in seconds, or use Expires to set its expiry date.

Domain: Specifies which hosts can receive the cookie.

Secure: The presence of this attribute ensures that the cookie is only sent over encrypted HTTPS connections.

HttpOnly: This protects against cross-site scripting (XSS) attacks.

SameSite: This attribute helps defend against cross-site request forgery (CSRF). It can be set to Strict, Lax, or None.

HTTP Servers

An HTTP server is a system consisting of hardware and software that stores resources in the form of web pages, files, images, and so forth, processes incoming HTTP requests, and delivers the required resource. HTTP servers support both the standard HTTP protocol and its secure version, HTTPS.

There are many products of the HTTP server type. The following are some of the widely used HTTP servers.

Apache HTTP server is an open source customizable, cross-platform web server available for use with all operating systems such as Unix, Linux, and Windows. It is usually distributed as an important constituent of the LAMP stack (Linux, Apache, MySQL, PHP). For Windows, the WAMP stack is also distributed for hosting web applications.

Nginx is a lightweight web server, also used as a reverse proxy and load balancer, providing efficient handling of high traffic and concurrent connections.

IIS is the acronym for Internet Information Services; this web server has been developed by Microsoft and is shipped with most versions of Windows OS.

LiteSpeed is a commercial web server designed as a high-performance alternative to Apache.

HTTP URLs

Resources on the Internet, such as HTML pages, CSS documents, and images, are identified by a URL (Uniform Resource Locator). Browsers send a request for a certain resource by mentioning its URL. The URL is composed of some parts.

It begins with a URL scheme, for example `http://` or `https://`, referring to the protocol used to request the resource.

After the double backslash (`//`) comes the authority part, which is optional and includes the domain and the port separated by a colon.

Next is the path to the resource, indicating the physical file location on the web server.

Another optional part is the parameter list. These parameters are a list of key-value pairs separated with the `&` symbol.

A URL may have an anchor that represents a sort of bookmark inside the resource.

Here are some examples of URLs.

<https://www.example.com>

<https://example.com/products?id=123&sort=asc>

<ftp://ftp.example.com/files/document.pdf>

On any web server filesystem, a certain location (folder) is defined to store all publicly accessible resources, such as web pages and images. This folder is called the *document root*. It acts as the entry point for a website. For the Apache server on the Linux operating system, /var/www/html is the default document root. Hence, a URL like <http://example.com/test.html> refers to test.html located in the /var/www/html folder, and translates to the physical path, such as /var/www/html/example.com/test.html.

Virtual Host

A virtual host is a single installation of an HTTP server that has been configured to host multiple websites using a single IP address. Each website has its own domain name. With virtual hosting, the server's resources, such as CPU and memory, are shared. This provides a cost-effective method for keeping each website's files, configurations, and data isolated from one another.

Virtual hosts can be either IP-based or name-based. In an IP-based virtual host, each site gets a different IP address. In the name-based setup, multiple websites share a single IP address.

The virtual host installation is configured with various important settings.

Listen directive: It specifies the IP address and port.

ServerName directive: It is the primary domain name for that virtual host.

ServerAlias directive: It refers to any other domain names to be handled by the virtual host.

DocumentRoot directive: It is the directory where a website's files are stored.

The following is a typical virtual configuration for Apache.

```
<VirtualHost *:80>
    ServerName www.example.com
    ServerAlias example.com
    DocumentRoot /var/www/example.com/public_html
    ErrorLog ${APACHE_LOG_DIR}/example.com_error.log
    CustomLog ${APACHE_LOG_DIR}/example.com_access.log
    combined
</VirtualHost>
```

HTTPS and TLS

A more secure variant of the standard HTTP protocol, HTTPS (short for HyperText Transfer Protocol Secure), was introduced with an additional layer of security. Initially, the security was enabled by Secure Sockets Layer (SSL). Later on, SSL was superseded by the TLS protocol (Transport Layer Security). In other words, HTTPS is a secure version of HTTP, powered by the TLS technology.

TLS is a cryptographic protocol. The three main features of TLS are encryption, authentication, and integrity. Secure communication between the client browser and the web server takes place through a process called a TLS handshake. Today, HTTPS is highly preferred over HTTP, and most modern browsers flag the URLs with the HTTP-only protocol (`http://`) as unsafe. HTTPS URLs start with `https://` and generally use port 443 by default. Browsers display a padlock in the address bar if the URLs are secure.

REST Constraints

As explained in the previous chapter, REST (Representational State Transfer), the most widely used API architecture, operates on the HTTP protocol. Other API development techniques such as SOAP and RPC are action-oriented protocols, whereas REST takes a resource-based approach. The HTTP request-response cycle in a REST API sends a request to fetch, create, modify, or delete a resource on the server. Accordingly, the server either transmits the resource in its response or sends its reference on being created, modified, or removed.

Another difference between SOAP - or RPC-based APIs and REST is that the former are protocols, while the latter (REST) simply lays down the principles to be followed in the API design. Rather, REST lays down certain conditions/criteria that the API must conform to. There are six design principles, formally known as REST constraints.

Uniform interface: This is the most crucial constraint that the REST API must fulfill. It requires that a URI identify each resource on the server and that requests for the same resource from different clients be the same. It also emphasizes that each message should include enough information for it to be processed. This is achieved by using appropriate HTTP headers in the request/response. Rather than a constraint, the uniform interface greatly simplifies the overall system architecture.

Client-server: The HTTP protocol, which is the backbone of the REST architecture, is based on the client-server model of network communication. This computing model helps maintain separation of concerns, letting the server and client-side processing logic evolve independently of each other.

Statelessness: This constraint is also a consequence of the fact that the REST architecture is built on top of the HTTP protocol, which requires that the server cannot hold any stateful information about the client's request. This constraint actually improves visibility, reliability, and scalability, as the server is not required to manage the session data.

Layered system: Because of the client-server constraint, the client is not connected directly to the server. This lets server functionality be developed into multiple

hierarchical and independent layers. This again improves the scalability of the system.

Cacheability: REST lets the server's response be marked as cacheable or non-cacheable. Cacheable responses can be reused by the client, which helps reduce network traffic and improve performance.

Code on demand: This sixth constraint is optional. Why? Most web applications are designed so that the APIs return either HTML, JSON, or XML responses. While REST does allow the server to send executable code, such as JavaScript, to the client (with the intention to extend or customize the client's functionality), this poses a potential security threat. Hence, this constraint is optional and is rarely implemented.

WebSocket

The HTTP protocol, and hence the REST API, has one major disadvantage because of its statelessness, which means the client has to reconnect with the server after each request. The WebSocket protocol, on the other hand, establishes a persistent HTTP connection between the client and the server, enabling uninterrupted communication between the two.

In the beginning, the client browser initiates a WebSocket handshake request to a server to upgrade the connection. The server acknowledges the handshake, accepts the upgrade request, and responds with an HTTP 101 Switching Protocols status code.

Once the handshake is accepted, both the client and server switch to the WebSocket protocol. The underlying TCP connection remains open, and both the client and server can now send and receive messages independently, enabling real-time, bidirectional communication.

Points to Remember

- The HTTP protocol follows a request-response cycle. The browser acts as a client that sends an HTTP request. The request consists of an HTTP method, request headers, and a request body.
- In HTTP communication, GET, POST, PUT, and DELETE methods are mainly used to perform fetch, create, modify, and remove operations on a resource stored on an HTTP server.
- The HTTP response body is associated with various status codes, classified based on how the server handled the request.
- Cookies play an important role in web application development. A cookie is a small text file stored on the client machine that stores data in key-value pairs pushed by the server in its response.
- A resource on the server is identified by a URL that maps it to its physical location in the server's filesystem.
- Different HTTP server software products are employed. Apache, IIS, and Nginx are some of the popular examples of web servers.

- HTTPS is the secure version of the standard HTTP, and it implements the TLS protocol.
 - The REST architecture is based on the principles of the HTTP protocol and is governed by six REST constraints.
 - WebSocket protocol facilitates two-way data exchange between the client and the server. The WebSocket request results in a persistent bidirectional communication channel.
-

Summary

The HTTP protocol is foundational to web application development. This chapter explains the important aspects of how the HTTP protocol works. The different request methods, response status codes, and the HTTPS and TLS protocols are discussed in detail. This chapter also covers the design principles and constraints of REST architecture.

Knowledge Check: Part I

1. **Which of the following best describes source code?**
 - A. The machine-readable binary executed by the CPU
 - B. Human-readable instructions written in a programming language
 - C. A compressed archive of application assets
 - D. The encrypted version of a database file
2. **Which programming paradigm focuses on breaking a program down into a collection of objects that contain both attributes and methods?**
 - A. Procedural programming
 - B. Functional programming
 - C. Object-oriented programming (OOP)
 - D. Declarative programming
3. **What is the primary difference between a Compiler and an Interpreter?**
 - A. Compilers are only used for web languages like HTML.
 - B. An interpreter translates the entire program into machine code before execution.
 - C. A compiler translates the whole program at once, while an interpreter translates it line by line during execution.
 - D. Compilers do not check for syntax errors.
4. **Why do developers use Version Control Systems like Git?**
 - A. To automatically write documentation for the code
 - B. To track changes in source code and allow multiple people to collaborate
 - C. To host the website on a public server

- D. To convert Python code into JavaScript

5.

What is the role of the client in a standard client-server model?

- A. To store the master database and process heavy logic
- B. To request resources and display the user interface (UI)
- C. To manage the physical hardware of the data center
- D. To act as a middleman between two different servers.

6.

Which of the following is an important characteristic of a single-page application (SPA)?

- A. The browser reloads the entire page every time a user clicks a link.
- B. It requires a separate HTML file for every possible view.
- C. It updates only the necessary parts of the page dynamically without a full reload.
- D. It cannot use JavaScript.

7.

Which type of database is characterized by using tables, rows, and columns with a fixed schema?

- A. NoSQL database
- B. Relational database (RDBMS)
- C. Key-value store
- D. Graph database

8. **What does WebAssembly (wasm) primarily allow developers to do?**

- A. Replace CSS for styling websites
- B. Run high-performance code (like C++ or Rust) in the web browser at near-native speed
- C.

- C. Create relational databases directly inside an HTML file
- D. Send emails using only HTML tags

9.

What is the main purpose of a content management system (CMS)?

- A. To compile C++ code into machine language
- B. To allow non-technical users to create, manage, and modify website content
- C. To monitor the battery health of the server hardware
- D. To replace the need for an IP address

10.

Which of these is an example of a NoSQL database?

- A. MySQL
- B. MariaDB
- C. MongoDB
- D. PostgreSQL

11.

Which of the following is the web browser developed by Apple?

- A. Chrome
- B. Safari
- C. Edge
- D. Firefox

12. **Which organization is responsible for standardization of the core web technologies?**

- A. ECMA
- B. WHATWG
- C. W3C

W3C

- D. CERN

13. **Which HTTP method is typically used to submit sensitive data, like a password in a login form?**
- A. GET
 - B. POST
 - C. HEAD
 - D. OPTIONS
14. **What does the HTTP status code 404 indicate?**
- A. Internal Server Error
 - B. Unauthorized Access
 - C. Not Found
 - D. Successful Request
15. **In the URL <https://www.example.com/images/logo.png>, which part refers to the specific resource path on the server?**
- A. <https://>
 - B. www.example.com
 - C. [/images/logo.png](https://www.example.com/images/logo.png)
 - D. [:443](https://www.example.com/images/logo.png)
16. **What is the primary purpose of an HTTP cookie?**
- A. To speed up the Internet connection
 - B. To store small pieces of data on the client-side to track sessions or preferences

- C. To encrypt the entire hard drive of the user
- D. To serve as a backup for the CSS file

17. **Which HTTP header tells the server what type of content the client can handle?**

- A. User-Agent
- B. Accept
- C. Host
- D. Set-Cookie

18. **What is the difference between static and dynamic web content?**

- A. Static content is written in JavaScript. Dynamic content is written in HTML.
- B. Static content is delivered exactly as stored on the server. Dynamic content is generated on the fly (e.g., via a database).
- C. Static content can only be viewed on desktop computers.
- D. Dynamic content is always faster to load.

19. **A status code starting with a "5" indicates what type of issue?**

- A. Client-side error (e.g., bad URL)
- B. Redirection
- C. Server-side error
- D. Successful information

20. **What is caching in the context of HTTP?**

- A. Deleting old files to save space
- B. Storing copies of frequently accessed resources to reduce load times and bandwidth

bandwidth

- C. Hiding the server's IP address
- D. Encrypting the connection between client and server

21. In the URL <https://example.com/products?id=123>, what is the query string?

- A. blog
- B. :8080
- C. /posts
- D. ?id=123

22. Which HTTP status code indicates a permanent redirect?

- A. 200
- B. 301
- C. 403
- D. 502

23. What is the document root on a web server?

- A. The main processor of the computer
- B. The directory on the server's file system where the website's public files are stored
- C. The first line of an HTML file
- D. The password for the administrator account

24. Which of the following Content-Type headers would be used for a JSON API response?

- A. application/json

- text/html
 - B. application/javascript
 - C. application/json
 - D. image/jpeg
25. **When using the GET method, where are the parameters usually sent?**
- A. Inside the HTTP request body
 - B. Encrypted in a separate file
 - C. Appended to the URL as a query string
 - D. They cannot be sent with GET
-

Answers with Explanations

- 1. **B**
Explanation: The source code is the sequence of instructions in a programming language saved as a text file. The extension indicates the language used. (Chapter [1](#))
- 2. **C**
Explanation: OOP is centered around objects that encapsulate state (attributes) and behavior (methods). (Chapter [1](#))
- 3. **C**
Explanation: Compilers build a stand-alone executable file from the source code, whereas interpreters execute the code directly by reading it on the fly. (Chapter [1](#))
- 4. **B**
Explanation: Version control systems like Git maintain a history of changes. Multiple developers can work together without overwriting each other's work. (Chapter [1](#))
- 5. **B**
Explanation: The client sends a request for a certain resource and renders the server's response to the user. (Chapter [2](#))
- 6. **C**
Explanation: SPAs load a single HTML page and update content dynamically via JavaScript. (Chapter [2](#))

7. **B**
Explanation: Data in relational databases like MySQL or PostgreSQL is organized in tables having a predefined structure. (Chapter [2](#))
8. **B**
Explanation: WebAssembly is a binary instruction format that enables running code in languages like C++ or Rust inside the browser alongside JavaScript. (Chapter [2](#))
9. **B**
Explanation: CMS platforms like WordPress or Drupal provide a graphical interface for managing content without writing code. (Chapter [2](#))
10. **C**
Explanation: MongoDB is a document-oriented NoSQL database that stores data in JSON-like formats.
11. **B**
Explanation: Safari is the default browser in Apple's operating systems like macOS and iOS. (Chapter [2](#))
12. **C**
Explanation: The objective of the World Wide Web Consortium (W3C) is to standardize HTML and HTTP and to develop new technologies such as CSS and JavaScript. (Chapter [2](#))
13. **B**
Explanation: The POST method sends data in encoded form in the request body, without exposing the data in the URL. (Chapter [3](#))
14. **C**
Explanation: A 404 error means the server could not find the specific resource (URL) requested by the client. (Chapter [3](#))
15. **C**
Explanation: The path follows the domain and points to a specific file or endpoint on the server. (Chapter [3](#))
16. **B**
Explanation: Cookies are sent by the server and stored as a text file on the client machine. This enables the websites to remember user preferences and track usage data. (Chapter [3](#))
17. **B**
Explanation: The Accept header specifies the MIME types that are acceptable for

Explanation: The Accept header specifies the MIME types that are acceptable for the response. (Chapter 3)

18.

B

Explanation: HTML/CSS composes the web content that doesn't change, while JavaScript can change page content dynamically based on user data or logic. (Chapter 3)

19.

C

Explanation: The 5xx range is reserved for errors that occur on the server side. (Chapter 3)

20.

B

Explanation: Caching allows responses to be reused by the client, which helps reduce network traffic and improve performance. (Chapter 3)

21.

D

Explanation: The query string starts with a ? and contains key-value pairs used to pass data to the server. (Chapter 3)

22.

B

Explanation: 301 is used when a resource has been moved to a new URL permanently. (Chapter 3)

23.

B

Explanation: The web server maps the base URL to this specific folder on the hard drive to find files like `index.html`. (Chapter 3)

24.

C

Explanation: `application/json` is the standard MIME type for JSON data. (Chapter 3)

25.

C

Explanation: GET parameters are visible in the address bar as part of the URL. (Chapter 3)

Part II

HTML for Web Design

HTML for Web Design

Topic 032: HTML Document Markup in the objectives of the Web Development Essentials Exam (Exam Code: 030-100) focuses on the structure of a web page document and the use of various HTML elements to construct a document. It intends to test the candidate's knowledge of basic and semantic tags as well as the controls used in the construction of an HTML form.

LPI allocates 8 weightage points to the topic 032. It is further arranged into four sub topics, each carrying two points.

- 032.1 HTML Document Anatomy
- 032.2 HTML Semantics and Document Hierarchy
- 032.3 HTML References and Embedded Resources
- 032.4 HTML Forms

Part II of this book corresponds to topic 032 of the official objectives. Three chapters in this part discuss the key aspects of HTML.

Chapter [4](#)

Chapter [4](#) briefly discusses how HTML has evolved over the years. It explains the basic structure of an HTML document. Elements are the basic building blocks of a document. This chapter explains various structural elements and some of the frequently used presentation tags.

Chapter [5](#)

Starting with why later revisions of the HTML standard introduced semantic tags, Chapter [5](#) explains, with ample examples, how they are used. It includes layout elements, disclosure elements, modal and non-modal dialogs, and multimedia elements.

Chapter [6](#)

Although HTML tables are not part of the official objectives, they are a significant part of this chapter, as they are an important tool for structured data presentation in an HTML document. Importantly, HTML forms and various form elements are discussed with real-life use cases.

At the end of Chapter [6](#) is a set of 25 practice questions (following a similar pattern as the LPI certification exam), along with their answers and appropriate explanations.

4. Basic Tags in HTML

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

Tim Berners-Lee conceptualized the World Wide Web in the late 1980s, and later launched the prototype in 1993. It was built on three fundamental technologies: the HTTP protocol as the backbone of the data transfer, HTML as the publishing language of the web with hyperlinked documents, and a web browser capable of retrieving the required document from the server and rendering it for the user.

HTML (HyperText Markup Language), as its expanded form indicates, is a markup language. A markup language is an encoding system that defines the formatting of a part of the text document. The markup mechanism was initially used in typesetting systems like LaTeX. Berners-Lee used the specifications of SGML (Standard Generalized Markup Language) as a basis for designing HTML. HTML is a descriptive markup language in which different parts of the document are marked by predefined labels, more commonly called *tags*.

Early versions of HTML were very lenient and flexible in enforcing the markup syntax, as the web browsers were built to display a page even with errors like unclosed tags. However, different browsers rendered the same page differently.

In the early 2000s, the World Wide Web Consortium (W3C) published HTML 4.0, designed on the principles of XHTML (Extensible HyperText Markup Language) instead of SGML. It recommended stricter syntax rules such as mandatory closing of tags and their proper nesting. Although the idea was to have much cleaner HTML code and

consistent browser behavior, it was not particularly well received by browser vendors.

WHATWG (Web Hypertext Application Technology Working Group), formed by companies like Microsoft and Apple, in association with W3C, published HTML5 recommendations in 2014. The latest version of HTML 5.2 was published in 2019. It is backward compatible and more flexible than HTML 4, but it also brings significant improvements.

Note that HTML is not a programming language (such as C/C++, Java, Python, etc.) because it cannot perform any logical operations. It only has tags to mark the appearance and formatting of a document; hence, it is not considered Turing complete.

Types of HTML Tags

HTML5 has more than a hundred tags. MDN (Mozilla Developer Network, now known as Mozilla Web Docs), widely recognized as an authentic, trusted, and comprehensive resource on web technology standards, maintains a list of 107 HTML tags (excluding deprecated and experimental tags). These tags can be broadly classified into the following types.

Structural tags establish the fundamental layout of an HTML document.

Presentation tags are used to define the appearance and styling of the text in the document.

List tags help you organize numbered or bulleted lists.

Table tags allow you to organize data in a tabular form.

Form tags are used to design forms that accept user input.

Multimedia and link tags are used to embed audio, video, or another HTML document and establish hyperlinks.

Semantic tags describe the meaning and purpose of the content. They are useful in improving the accessibility and readability of the document. Search engines use these tags to determine page rankings.

Paired tags have the separate opening and closing tag symbols (for example, `<html>` and `</html>`) used in most HTML.

However, there are others called self-closing tags. No separate closing symbol is required because they insert an item at a specific point on the page. The tag is an example. Also, these tags don't contain any nested elements; hence, they are called void tags.

A trailing slash (/) is optional but often included for consistency in the self-closing tag usage.

```

```

HTML tags are not case-sensitive. As a result, the browser treats <p> as well as <P> the same way. Some tags have one or more attributes. For example, the tag has the src attribute. Attributes are also case-insensitive. However, the attribute values are case-sensitive. The attribute values are enclosed in single or double quotes (' or "). They are interchangeable. In other words,

```

```

is the same as the following.

```
<img src='image.jpg' />
```

Like any other computer language, you can add certain text that the browser will not interpret. Such text is called a comment. Comments are used to add notes and explanations within the source code. To let the browser's rendering engine know that it is not to be acted upon, the comment text is put inside the opening tag <!-- and the closing tag -->. The browser ignores anything inside these tags.

```
<!-- This is a comment inside HTML script-->
```

Another term related to the tag in HTML is an element, often used interchangeably, although the two are different. A tag is a predefined marker, with one or more characters inside the angular brackets. For example, <p> is a tag that marks the beginning of a paragraph, the end being marked by the </p> tag. What falls between the two is an

element. So, the text between `<p>` and `</p>` is a paragraph element. See Figure [4-1](#).

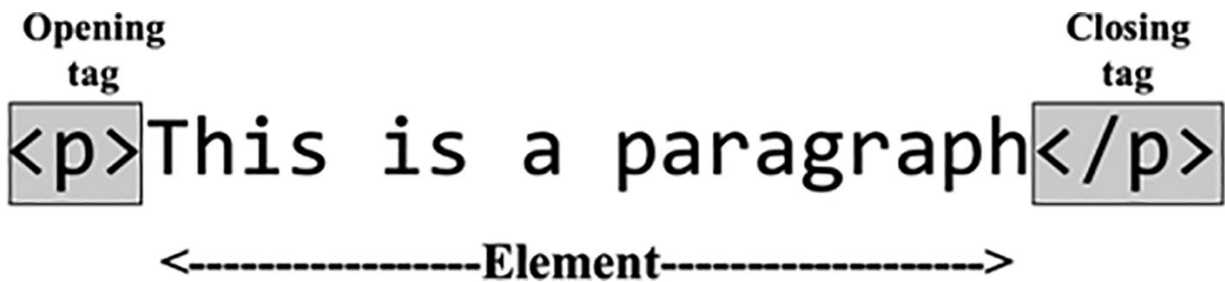


Figure 4-1 HTML element

Structural Tags

All the elements in an HTML document are organized in a nested manner inside two sections of the root element. HTML5 specification requires a DOCTYPE declaration to be put before the root element.

DOCTYPE

The HTML document script invariably starts with the following statement.

```
<!DOCTYPE html>
```

Here, DOCTYPE is not a tag (note the ! symbol). It merely tells the browser that the following document is written in HTML5. As a result, when the browser loads the document, it knows which HTML version to follow, and hence interprets tags and elements accordingly. Without a DOCTYPE declaration, browsers may switch into “quirk” or backward-compatible mode, behaving like the old browsers from the 1990s. What is more, the page is likely to be displayed differently on different browsers.

Root Element

Just below the DOCTYPE declaration is the root element. A document in any markup language has a single root or topmost element, inside

which all other elements are nested. In an HTML document, opening `<html>` and closing `</html>` tags form the root element. All other elements are put inside two sections: the head section and the body section.

```
<!DOCTYPE html>
<html>
  <head>
    <!--head section-->
  </head>
  <body>
    <!--body section-->
  </body>
</html>
```

Head Section

The HTML instructions inside the `<head>` and `</head>` pair form the head section of the document. Though nothing from this part is rendered on the web page, it plays a crucial role in how it is rendered and how the user interacts, and defines the relationship with external resources such as style sheets (CSS) and JavaScript code. The head section is also a place where useful metadata information is included. Note that the head section is always defined before the body section.

<title> element

This element is almost invariably defined in the head section. The text inside the `<title>` and `</title>` tags appears in the browser's title bar (not as a part of the web page). In the absence of a title tag in the HTML script, most browsers display "New Tab" as the title.

```
<head>
  <title>Hello World</title>
</head>
```

The title tag is one of the most important factors that search engines take into account while determining the page ranking. It is important that the title includes appropriate SEO-friendly keywords. The title should be descriptive, not short or generic.

<link> element

This element allows you to include external resources such as stylesheets and site icons (favicon) in the current document. Its rel attribute is mandatory. The href attribute specifies the URL of the external resource. For CSS, the rel attribute is set to "stylesheet", while its value is "icon" for including a favicon.

```
<head>

    <link rel="stylesheet" href="styles.css" />

    <link rel="icon" href="favicon.ico"
type="image/x-icon" />

</head>
```

The type attribute here is the MIME type of the resource.

<style> element

Although the standard practice for defining the styling attributes of various elements in an HTML document is to link an external CSS file, it is also possible to write CSS rules directly within the HTML document. The style definitions are enclosed within a <style> and </style> pair. Although this can be done in the head as well as the body section, it is recommended to put the styles in the head section.

```
<head>
    <style>
        p {
            color: #26b72b;
```

```
    }  
  </style>  
</head>
```

You will learn about the CSS rules in a subsequent chapter, but it is important to note that styles can be defined in the head section (as done here), in an external CSS file, or inline when an element appears in the body section.

<script> element

By default, the HTML script renders a static web page. Since HTML version 3.2, dynamic and interactive functionality can be added in HTML. JavaScript code that defines functions, classes, and event handlers can be put inside the `<script>` and `</script>` pair of tags.

You can either place JavaScript code directly inside the HTML page, or include an external file containing JavaScript code, much like a CSS file.

Note that the `<script>` element can be placed in the head as well as the body section, depending on how and when you want the JS to be executed. If it is put inside the `<head> .. </head>` pair, the JS code runs before the page loads. If it is inside the body section, it takes effect after the HTML instructions before the page appears.

The following snippet is an example of executable JavaScript code placed directly inside the `<script>` tag and as part of the head section.

```
<head>  
  <script>  
    alert("Hello World.");  
  </script>  
</head>
```

To link an external file with JavaScript code, set the `src` attribute of the script element to it. The value of `src` may be an absolute or relative path of a file with a `.js` extension, or it may even be its URL.

```
<head>  
  <script src="my-script.js"></script>  
</head>
```

The `<script>` tag also supports a type attribute. For HTML5, "text/javascript" is its default value. In modern JavaScript versions with the mechanism of importing and exporting modules, the type attribute is set to "module".

<meta> element

The information provided in the meta tag is very significant for search engines. You can include certain additional metadata about the contents of the current page in this tag. The following are important attributes of a meta element.

charset: Specifies the character encoding for the document. The most common values are UTF-8 and ISO-8859-1.

name and content: The name attribute specifies various types of metadata. Its possible values can be description, author, and viewport. You must also include the content attribute. It holds the value associated with the metadata defined by the name attribute.

```
<meta name="description" content="This page  
explains the use of meta tag in HTML.">
```

The viewport attribute is the user's visible area of a web page. It is set to width=device-width to cause the page width to match the screen width of the device. Also, initial-scale=1.0 sets the initial zoom level to none when the page is first loaded.

```
<meta name="viewport" content="width=device-width,  
initial-scale=1.0">
```

You can define an attribute with "author" as its value to specify the author of the current document.

```
<meta name="author" content="Author Name">
```

Another useful attribute is http-equiv, which emulates an HTTP header and provides instructions to the browser.

```
<meta http-equiv="refresh" content="30">
```

This meta instruction causes the page to be automatically refreshed every 30 seconds.

Body Section

The browser renders the various HTML elements defined inside the body section marked by the <body> and </body> pair of tags. The body section is always defined after the head section. It comprises various text formatting elements such as headings and paragraph tags; elements that organize the page (for example, <div>, , and <table>); tags that embed multimedia objects; and the HTML form.

The appearance and behavior of the body section can be customized by various attributes. Modern HTML5 specifications recommend that the styling attributes such as (bgcolor, text, and background) of the <body> tag are defined by CSS rules.

The <body> tag supports the following global attributes.

class: Used for styling with CSS or selecting with JavaScript.

id: A unique ID used for identification and styling.

style: Allows you to define inline CSS styles.

```
<body id="id001" class="homepage" style="border:
2px solid #555;">
    <h1>Welcome to My Website</h1>
</body>
```

The <body> tag also supports certain attributes that define actions to be executed in the form of some JavaScript code when a specific event occurs on it.

onload: Captures the event of the process of finishing a page being loaded and executes a script when the page is finished loading.

onunload: Triggers a JavaScript code when the user exits the page.

onresize: Attaches an event handler to run when the browser window is resized.

```
<body onload="alert('Hello, welcome to the  
page!');">  
    <h1>Hello World</h1>  
</body>
```

However, this approach is not recommended in modern web development trends. Instead, using `window.addEventListener()` with appropriate events is more preferred.

Presentation Tags

Most of the tags defined in the early versions of HTML were (and still are in the later versions) meant for rendering and structuring the text content. Tags related to multimedia and semantics were added much later.

Text in the `<body>` section that is not decorated by any tag is rendered by the browser using a font, mainly based on the user's settings and not governed by any HTML standard. Browsers generally render the text in Times New Roman or Arial, unless specified otherwise by CSS rules. Also, most browsers render the text in a font size of 16 px (approx. 12 pts) in black color on a white background.

The most commonly used presentation tags that control how the text appears are `<b>`, `<i>`, and `<u>` for making the text bold (thick), showing it in italics, and marking it underlined.

```
<body>  
This text is <b>bold.</b> This text is in  
<i>italic.</i> This is <u>underlined.</u>  
</body>
```

The browser renders the HTML script as follows.

This text is **bold**. This text is in *italic*. This text is underlined.

Modern HTML offers semantic equivalents to these tags. The `<strong>` tag indicates strong importance by making the text thick. The `<em>` tag emphasizes the text by rendering it in italics. The `<ins>` tag indicates inserted text that is typically underlined.

```
<body>
This text is <strong>bold.</strong> This text is
in <em>italic.</em> This is <ins>underlined.</ins>
</body>
```

Semantic tags are helpful to screen readers. Search engines use them to better understand your content.

Heading Tags

An HTML document must be hierarchically and logically structured with appropriately formatted captions. HTML provides six tags from `<h1>` to `<h6>`. To underline the importance of the page title or the name of a major section, it is marked with the `<h1>` tag. The text inside the `<h1>` and `</h1>` pair appears in a bigger font. The font size successively decreases for the rest of the heading tags, indicating the hierarchy of the document structure.

In this chapter, for example, the main section's heading, **Structural Tags**, is shown with the `<h1>` tag. The subtopics in this section, the head section and the body section, have the `<h2>` tag. Under the head section, the paragraphs explaining each element are shown with the `<h3>` tag.

Heading tags are semantic rather than presentation tags, as their objective is to help browsers understand the logical structure of the document. The font sizes of heading tags are relative to the base font size of the `<body>` element, which is 16 px. Accordingly, the font size of

the text in the `<h1>` element is 2em, 1.5em for `<h2>`, and so on. The font size of `<h4>` is 1em (i.e., 16 px, roughly equivalent to 12 point size, a unit traditionally used in the printing industry). Table [4-1](#) shows the comparison of font sizes of heading tags in em, px, and pt units.

Table 4-1 Font Sizes of Heading Tags in em, px, and pt Units

Figure [4-2](#) gives an idea about the relative font size of different heading levels.

`<h1>` - Hello World

`<h2>` - Hello World

`<h3>` - Hello World

`<h4>` - Hello World

`<h5>` - Hello World

`<h6>` - Hello World

Figure 4-2 Heading levels

Paragraph Element

In general, a paragraph is a cohesive text block that focuses on a single topic or concept, usually having one or more sentences. Any document (including the HTML document) is structured with multiple distinct paragraphs. HTML treats the text included in `<p>` and `</p>` tags as a paragraph.

```
<p>This is a paragraph.</p>
```

The `<p>` tag is a block-level element. It always starts a new line. Most browsers leave 1.2em of space between lines and 1em of space

above and below the paragraph. This helps in showing paragraphs distinctly.

The paragraph may contain other presentation tags or anchor tags. HTML ignores any whitespace in the text. As a result, any text flowing over multiple physical lines inside the `<p>` tag is rendered as a paragraph, as shown in the following example.

[Table](#)

You can, however, use the self-closing `<br>` tag to insert a line break without creating a new paragraph, as shown in the following example.

[Table](#)

Another tag worth explaining here, particularly in the context of an HTML paragraph, is the `<pre>` tag. All whitespace inside the `<p>` and `</p>` pair is ignored. But sometimes, we do need the spaces, the tabs, and line breaks to appear in the browser output, as in a code snippet. This is where the `<pre>` tag is useful. It displays the preformatted text, preserving all the extra spaces, tabs, and line breaks, as shown in the following example.

[Table](#)

Character Entities

The preceding example uses characters like `<` and `>` in the HTML code. The browser renders them as `<` and `>` respectively. In HTML, some characters have a specific meaning in the markup syntax. For example, the characters `<` and `>` denote the beginning and end of a tag. Similarly, single and double quotation marks (`'` and `"`) are used to assign a value to an attribute. If you intend to display these characters as a literal

without their respective HTML-specific meaning, you need to replace them with their respective character entities.

The character entity is a sequence of characters, usually starting with the & character and ending with the ; character. The character entity corresponding to < is < and > is represented by >.

Reserved characters may also be represented by decimal equivalents, starting with &#. In this reference scheme, < is replaced by < and " represents the double quote symbol (").

Table [4-2](#) shows some important reserved characters and their named and decimal entry equivalents.

Table 4-2 Reserved Characters and Named and Decimal Entry Equivalents

The following HTML snippet shows the usage of HTML entities.

```
<lt;body>
<p><lt;p> denotes paragraph tag</p>
<p>HTML stands for &quot;HyperText Markup
Language&quot;</p>
</body>
```

In your browser window, you should see the following output.

```
<p> denotes paragraph tag

HTML stands for "HyperText Markup Language"
```

List Tags

A well-structured document makes effective use of numbered and bulleted lists to improve readability and accessibility. You may want to display a collection of related items, such as steps in a process, points to be covered in the content to follow, navigation links, and so forth. Lists help screen readers and browsers understand the visual structure and

semantic meaning of a web page. HTML supports ordered lists and unordered lists through `<ol>` and `<ul>` tags.

Ordered List

Individual items in the list are put inside the `<li>` and `</li>` tags. If the order of items in a list is important, the `<li>` elements are placed inside `<ol>` and `</ol>` tags. Usually, an incrementing number is displayed to the left of each item in the list, as shown in the following example.

[Table](#)

By default, item numbering starts at 1. The `<ol>` tag does have the `type` attribute to change the numbering type.

- a: lowercase letters
- A: uppercase letters
- i: lowercase Roman numerals
- I: uppercase Roman numerals
- 1: numbers

[Table](#)

Unordered List

If the order of items in a collection is not particularly important, we use the `<ul>` tag. By default, the browser displays a disk symbol (●) on the left of each item, as shown in the following example.

[Table](#)

Older HTML versions (before HTML5) had the provision to set the `type` attribute of the `<ul>` tag to `circle` or `square` to change the bullet symbol. However, it has since been deprecated. Instead, you can use CSS to customize the list symbols.

Nested Lists

The term *nesting* is used frequently in computer programming. It refers to putting one programming construct (such as a loop) or a data structure (such as a list) inside another. In HTML, the tags are always used in a nested manner. Likewise, HTML list elements can be nested, where an element in an upper-level list is another list.

```
<!--Parent List-->
<ul>
  <li>Item 1
  <!--Child List 1-->
    <ul>
      <li>Item a</li>
      <li>Item b</li>
    </ul>
  </li>
  <li>Item 2
  <!--Child List 2-->
    <ol>
      <li>Item i</li>
      <li>Item ii</li>
    </ol>
  </li>
</ul>
```

Both list types, ordered and unordered, can be nested. Moreover, the parent and child lists can be of either type. This means you can have an ordered list made up of more than one unordered list, and vice versa. Moreover, the child elements can be a mix of both types as well, as shown in the following example.

[Table](#)

<div> element

The <div> tag allows you to create a block-level container element. Usually considered short for division, the <div> element plays a crucial role in the uniform styling of related content in a web page. It is non-semantic in nature as it doesn't describe the nature of its contents. Web pages are generally designed by nesting div elements with appropriate styling by consistent CSS rules. See Figure [4-3](#).

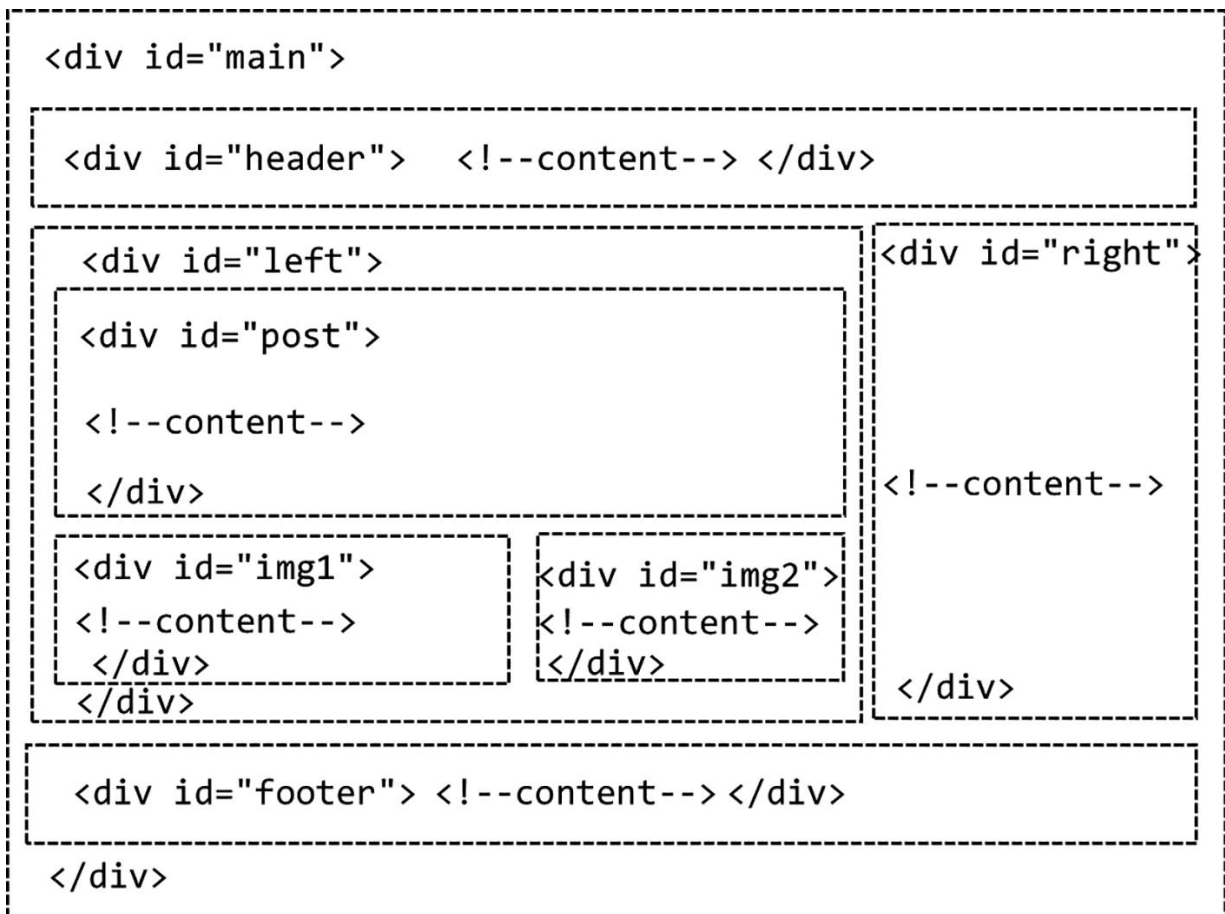


Figure 4-3 Nested div elements

Although the <div> element is quite versatile and is a common choice for web developers, excessive use often makes the HTML script cluttered and difficult to debug. It is a non-semantic element. HTML5 offers a rich collection of semantic container elements such as <header>, <nav>, <section>, <article>, and so forth. Hence, wherever

appropriate, it is recommended to use these semantic elements instead of `<div>`, because it improves the structure and accessibility of the document.

Points to Remember

- An HTML document is designed with predefined markup tags.
 - The DOCTYPE tag at the top indicates the HTML version used.
 - The page structure lies inside the root `<html>` element.
 - Any HTML page has two sections marked by `<head>` and `<body>` tags.
 - The `<head>` element mainly serves as a place to define relations with external resources such as CSS and JavaScript code.
 - The content actually rendered by the browser goes in the `<body>` element.
 - HTML has six heading tags with predefined font sizes.
 - The presentation tags such as `<b>` and `<u>` define the appearance of the text content.
 - `<p>` for paragraph and `<div>` for division are the two main structural tags.
 - To render an ordered/bulleted list of items, HTML provides `<ol>`, `<ul>`, and `<li>` tags.
-

Summary

Tags are the basic building blocks of a web page. The tags can be classified into different categories. This chapter discussed the use of the most frequently used tags. The contents in the head section of the page don't actually show up on the browser. It mainly establishes the relationships of the page with certain external resources such as CSS, and contains important metadata about it. The body section is what is actually displayed. The chapter also discussed the elements that define the document structure, such as headings, paragraphs, lists, and divisions.

In the next chapter, you will learn about the semantic tags in HTML.

5. Semantic Tags in HTML

Malhar Lathkar¹✉

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

The two important aspects of a language—whether a human language or a computer language—are syntax and semantics. While syntax refers to the formal rules governing how words (keywords in a computer language such as Java or Python or tags in HTML) may be combined, semantics refers to the meaning or interpretation of those combinations.

In the early days of the WWW, the web pages were static. Hence, basic tags like `<b>`, `<u>`, and `<i>` are only stylistic. But others, like the root element and the tags that define the main sections (head and body), are certainly semantic in nature, because they clearly indicate their meaning. The paragraph element marked by the `<p>` tag can also be considered semantic.

However, elements like `<div>` (a block-level container) and `<span>` (an inline container) are generic in nature. For example, you can use `<div>` for grouping, layout, or styling as well. There is nothing in its usage syntax that suggests its meaning. It is only through the CSS rules that you can understand what a `<div>` element stands for.

As the web grew, it became necessary that HTML code should be accessible, more structured, and machine-readable. Hence, HTML5 introduced several semantic tags that help enhance the visibility of a web page in search engine rankings (SEO).

Apart from the presentation tags and a few generic ones, most of the HTML tags are semantic. Even the presentation tags such as `<b>`, `<i>`, and `<u>` now have semantic equivalents (`<strong>`, `<em>`, and `<ins>`, respectively), and as such, they are generally not recommended anymore.

The tags for creating various elements in an HTML table, and various interactive elements in an HTML form are classified as semantic tags. They are discussed in another chapter. Those used in creating ordered/unordered lists have been explained in the previous chapter. This chapter focuses on the anchor element, elements used for defining the layout and high-level structure, tags used to embed multimedia elements such as audio and video, and certain text-level semantic tags.

Anchor Element

By definition, the World Wide Web (WWW) is a large collection of interlinked web pages, which may be on the same or different servers. The anchor element with its `href` attribute is at the core of the idea of the WWW, as it provides a hyperlink around the text (or image) marked by the `<a>` and `</a>` tags. In fact, the marked text itself is called *hypertext* (as in HTTP).

Here's the basic usage of the `href` attribute to create a hyperlink to another page.

```
<a href="other_page.html">Hypertext</a>
```

When the browser renders the anchor element, the pointer changes from an arrow to a hand as the mouse hovers over the hypertext. When clicked, the browser navigates to the URL assigned to the `href` attribute. Usually, the text inside the anchor tags is appropriately worded to let users know where they will be navigated to.

The page hyperlinked by the `href` attribute may have an absolute or relative URL. Unless specified otherwise, the browser assumes that the linked page is in the current folder. The absolute address can use

the file:// URL scheme, or it can be a root-relative URL.

Assume that the web pages are put inside a d:\MyWebSite folder as follows.

```
D:\MyWebSite>
|   index.html
|   introduction.html
|
└── topics
    |   http.html
    |   tags.html
```

The index page contains the links to the other pages, as shown in the following HTML code.

Table

The browser renders the marked text with a blue underline. The underline's color turns to purple if the hyperlink has been visited. For the active link, the color is red. However, these defaults may be overridden by modifying browser settings or by custom CSS styling.

The href attribute can be a URL that points to a website on the World Wide Web. Both the https:// and http:// URL schemes may be used. Just as with text content within the anchor tags, you may display an image (with the element) that has a hyperlink to any web page.

```
<!--Linked to an external URL-->
<a href="https://developer.mozilla.org/">Mozilla Developer Network</a>
<br>
<!--Hyperlink around an image-->
<a href="https://developer.mozilla.org/docs/Web/HTML"></a>
```

When the user follows a hyperlink, by default the linked page opens in the current window itself. However, specifying the target attribute of the <a> element helps you open it in a different location.

The target attribute can take any of the following values.

_self: This is the default value. Unless specified otherwise, the linked document opens in the same window/tab as it was clicked

_blank: The browser opens a new window or tab for the linked document to be rendered.

_parent: This causes the document to be opened in the parent frame

_top: The document opens in the full body of the window.

Other URL Schemes in Hyperlinks

Normally, the string value assigned to the href attribute is the path of a web page expressed in file:// or https:// URL scheme. However, HTML also allows you to use other URL schemes, such as telephone numbers and email addresses.

```
<!--opens the user's default email program-->
<a href="mailto:johndoe@example.com">Send email to me</a>
<br>
<!--Hyperlink pointing to the default phone app-->
<a href="tel:+919988776655">Call me</a>
<br>
<!--Hyperlink so send SMS-->
<a href="sms:+919988776655">Send SMS</a>
<br>
```

```
<!--WhatsApp link-->
<a href="https://wa.me/919988776655">Chat on WhatsApp</a>
```

Internal Hyperlinks

The anchor element lets you build hyperlinks that point not only to other documents but also to any place inside the same document. To create an internal hyperlink, assign the id attribute to the target element (the one you want the browser to navigate to when the hyperlink is clicked). Assign this id (prefixed with the # symbol) to the anchor (<a>) element's href attribute.

For example, the following is a paragraph element in an HTML document with the id "http".

```
<p id="http">
HTTP protocol is the foundation of data communication for the World Wide
Web. It defines how messages are formatted and transmitted, and how web
servers and browsers should respond to various commands.
</p>
```

To create a hyperlink that takes the browser to this paragraph, define the anchor element as follows.

```
<a href="#http">Know about HTTP protocol</a>
```

The target of an internal hyperlink may be a paragraph, div, section, or any layout element. Such internal hyperlinks are usually employed in the Table of Contents of a document, navigation menus, and so on. Note that simply stating href="#" creates a dummy hyperlink.

Layout Elements

HTML5 introduced many new layout-oriented semantic elements (such as header, section, nav, etc.) that clearly imply the purpose of their content. Instead of employing a clutter of <div> elements for various sections in the page layout, you can now use these elements with more mnemonic names. They improve the clarity of the document structure and help browsers and other assistive technologies to understand and interpret the content.

Let's discuss some of these elements.

<header> Tag

The header element represents introductory content, usually at the top of a page, an article, or a section, as shown in the following example.

[Table](#)

It can contain a site, page, or section title in an appropriate heading tag, an introductory line or paragraph, a logo, and navigation links. Screen reader software recognizes <header> as a "banner landmark" and uses it (all semantic elements, in general) to interpret the content for the visually impaired. The logo is inserted with the tag, which you will learn later. The <nav> tag helps in providing the navigation feature.

<nav> Tag

The navigation section provides hyperlinks to other documents on the server, links to sections in the same page, or other URLs. HTML defines the hyperlinks with the href attribute of the anchor element marked by the <a> tag, as shown in the following example.

[Table](#)

Generally, you place the `<nav>` element inside `<header>`, although it can also appear elsewhere, for example, in the footer section or sidebar. It can also be used in an `<article>` section as a local TOC.

The hyperlinks can be put as `<li>` elements inside the `<ul>` or `<ol>` tag. Developers define suitable styles for customized arrangements of the links.

`<main>` Tag

The part of the HTML page marked by the `<main>` tag identifies its primary, unique content. It tells exactly where the central content of the page is located.

```
<main>
  <!--unique content-->
</main>
```

Because its content must be unique, the information that may appear across multiple pages in the website (such as site logos, navigation links, or footers) should not be included within the `<main>` tag. Also, there must be only one occurrence of the `<main>` element in a single page.

Although it behaves like a generic `<div>` element as far as its visual appearance is concerned, it provides crucial structural meaning to screen readers, so users can head directly to the main content and bypass repetitive elements like links and footers.

A typical web page layout may have other elements like `<section>`, `<article>`, forms, and figures inside the `<main>` element. Repetitive elements such as the header, footer, and sidebar fall outside it. See Figure [5-1](#).

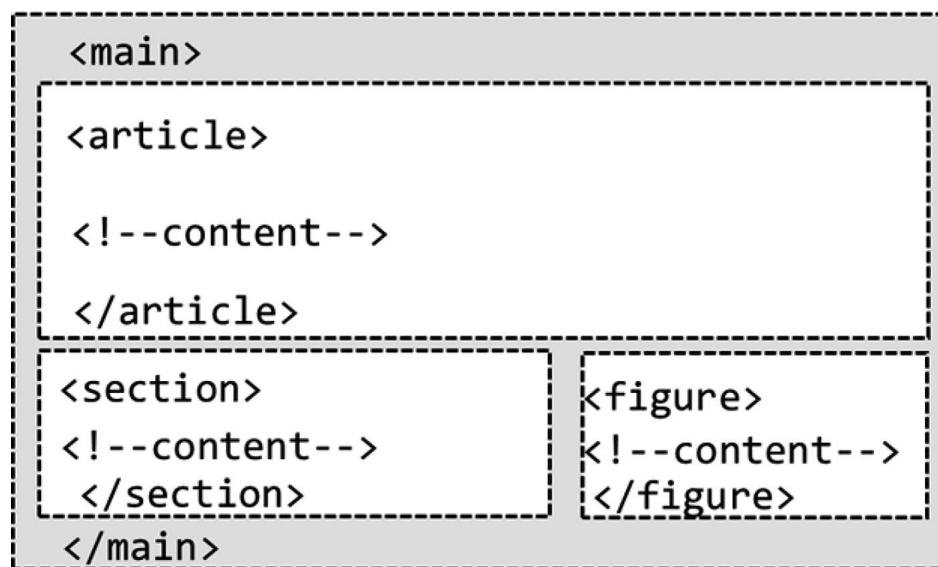


Figure 5-1 `<main>` element

`<section>` Tag

If the content of the main element needs to be distributed into more than one part, you can include as many `<section>` elements as needed. Generally, sections are arranged to maintain continuity. A section does not stand alone; it is contextual to the adjacent sections.

```
<section>
  <!--section content-->
</section>
```

Each section should have a heading element `<h1> ... <h6>`. The heading level of the section should be at a lesser level than the heading of the main element. Sections in the same main element should have headings of the same level.

Think of a chapter of a certain book divided into several topics. The chapter is the main element, whereas each topic can be considered as a section. Just as a topic can have further subtopics, a section element can also have more section elements placed inside it. To indicate that they are subsections, their heading should be at a lower level, as shown in the following example.

[Table](#)

`<article>` Tag

In contrast to a section element, the `<article>` tag marks an independent, self-contained, and reusable element in the main content. The article element has its own context and does not depend on adjacent content.

```
<article>
  <!--independent content of the article-->
</article>
```

One common instance of the article element is a blogging website, where each blog post in the feed is independent content. It is stand-alone, as it can be copied anywhere else and still be as meaningful. Other use cases of the article element are social media posts, comments on a blog, each product card in an e-commerce site, and so on.

An article may have its own navigation links, paragraphs, images, and so forth. Each article element may further have its own section and subsections, with suitable headings to indicate relative importance, as shown in the following example.

[Table](#)

`<footer>` Tag

Normally, the footer element appears at the end of a web page, just before the `</body>` tag.

```
<footer>
  <p>Author: John Doe<br>
  <a href="mailto:doe@example.com">doe@example.com</a><br>
  © 2025 All rights reserved
</p>
</footer>
```

The footer may also be used inside any sectioning content element such as `<article>`, `<section>`, or even `<main>`. Generally, the `<main>` or `<section>` elements often do not require independent footer metadata. However, it is a common practice to have it inside the article element.

A `<footer>` typically contains information such as copyright notices, organization details, legal disclaimers, related navigation links, or contact information. When used with an article, the footer may show publication date, author information, or related links (e.g., LinkedIn profile).

`<aside>` Tag

Sometimes you may want to include certain information that is tangentially related but not essential to the main content. For example, Sidebars, ads, related links, or any other supplementary info. Such content is put inside the `<aside>` element.

```
<aside>
```

```
<!--supplementary content-->
</aside>
```

The `<aside>` element can appear inside `<main>` or next to it. The placement of the `<aside>` element depends on whether it relates to the page or the main content. Certain use cases of the `<aside>` element include related articles, notes, navigation links, ads, and so on.

Disclosure Elements

With the combination of `<details>` and `<summary>` tags in the HTML document, you can add an interactive element that acts as an expandable content block. The text marked by the `<details>` tag creates a disclosure element, while the `<summary>` element acts as its label.

```
<details>
  <summary>Label</summary>
  <!--content of the disclosure element-->
</details>
```

A small triangle symbol is displayed to the left of the label, which, when clicked, toggles the disclosure widget between its expanded and collapsed view, as shown in the following example.

[Table](#)

Put the preceding HTML script in the body section and open the page in the browser. The summary labels with the triangle symbols are on the left. Click any of them to show the details view, and click back to hide it. See Figure 5-2.



Figure 5-2 Disclosure elements

This click-to-expand feature is ideal for features like FAQs, lists, and a glossary of terms on a website.

`<dialog>` Element

HTML 5.2 introduced the dialog element that enables user interaction with the web page. This native HTML element lets you implement pop-up dialogs, alerts, confirmations, or forms. Dialogs are of two types: modal and non-modal. A modal dialog cannot be bypassed; it blocks interaction with the rest of the page until it is closed. A non-modal dialog, on the other hand, is unobtrusive; you can access the page underneath while the dialog is still visible.

```
<dialog>
  <--Dialog content-->
</dialog>
```

By default, the dialog element is hidden. If you use its Boolean open attribute, the dialog will be visible as the page loads. To discard the dialog, you must have an interactive element such as a button or a form. An HTML form with its method attribute set to "dialog" handles the form submission on the client side (as opposed to the GET/POST methods, which submit the form data over the network). The following is an example of HTML and the browser output for the <dialog> element.

[Table](#)

Instead of using a dialog that is visible on page load, using a modal dialog becomes more effective when JavaScript code handles its display and closure by implementing the showModal() and close() methods.

Multimedia Elements

The element has been a part of HTML from its early versions. HTML5 introduced elements like <audio> and <video> to embed audio and video content in the HTML document. Before that, there were no such multimedia playback elements. HTML 4 does have the <object> element that allows certain external resources like audio/video, PDFs, or Java applets, but it requires browser plugins like QuickTime or Flash (which is blocked by most browsers).

The <embed> element, even though still a part of the HTML living standard, is not preferred and is also used to include multimedia content.

 Tag

A web page is predominantly text-based. However, the user finds a document with a continuous flow of text less engaging. Hence, it is desirable to intersperse appropriate images or figures in the text using the element. is a self-closing tag, with its src attribute specifying the path to the image file to be displayed.

```

```

The src attribute is mandatory. It is a string that may refer to a file in the current folder, its absolute or relative path, or the URL of the image to be displayed.

The following are optional attributes.

alt: specifies an alternate text to be displayed if the image cannot be loaded. It proves extremely useful for screen readers.

width and **height:** in pixels, must be integers.

title: an optional subtitle for the alt attribute.

The following is an example of HTML and the browser output for the element.

[Table](#)

The tag supports a variety of image formats. These include PNG, JPG, GIF, and WEBP, among others.

Note that the element is a non-semantic element, even though it is included here in the discussion on semantic tags, as it is part of the semantic image container: <picture>.

<picture> Tag

While the img element lets you embed an image in the web page with its src attribute assigned to a fixed source, the <picture> element renders a container for responsive images, where source selection depends on conditions. Multiple image sources are specified with the srcset attribute, and the browser chooses the most appropriate source.

```
<picture>
  <source srcset="image type1">
  <source srcset="image type2">
  
</picture>
```

The srcset attribute of the <source> tag is mandatory. Other optional attributes are type (MIME type), media (media query), and sizes (responsive sizing). The browser evaluates each <source> in order and uses the first matching source.

Following the multiple sources, you must define an element that acts as a fallback inside the <picture> element. It renders a default image in case the browser is unable to load any of the sources from the list.

In the following HTML script, the <picture> element is defined to choose from multiple image sources. Most modern browsers support the .webp image format; hence, the first image in the list should be rendered. If your browser doesn't support .webp or .png format, it falls back on the .jpg file format.

[Table](#)

<figure> Tag

Most often, you find images embedded in a document as an element. Usually, such an image has an inline reference in the related text. However, is a non-semantic element. The <figure> element is a semantic container that can hold not only an image, but also a code snippet or a quotation. It can optionally have a descriptive caption marked by the <figcaption> tag.

```
<figure>
  <!--figure contents-->
  <figcaption>Caption text</figcaption>
</figure>
```

One distinction between the two is that the element is inline, that is, contextual to the text flow, while the <figure> element is often independent of the main content.

The following is an example of the <figure> element with a code snippet as its content.

[Table](#)

<audio> Tag

One of the semantic additions to the list of HTML elements by HTML5, the <audio> element allows you to embed audio/sound content in the web page. It presents an interactive interface to control the play/pause actions.

The src attribute specifies the audio file to be used for the playback. Using the controls attribute displays the browser's default playback interface, which includes play/pause buttons, a volume control, and a progress bar.

```
<audio src="path/to/audio file" controls></audio>
```

Most modern browsers support MP3, WAV, and OGG audio file formats.

Instead of specifying a single audio source as the src attribute, you can provide multiple audio files for the browser to choose, with <source> elements inside the <audio> and </audio> tags. The following is an example of HTML and the browser output for the <audio> tag.

[Table](#)

The `type` attribute represents the MIME type of the audio file. The browser displays the fallback text if it is unable to play any of the audio file formats in the list.

The interactive control widget exposes events that can be used to control playback with JavaScript event handlers.

play: Fires when the playback has begun.

pause: Occurs when the pause control on the UI is clicked.

ended: Occurs when playback has stopped because the media has ended.

playing: Restarts playback after being paused.

waiting: Occurs during buffering pauses.

error: Fires if an error occurs during the loading or playback process.

<video> Tag

HTML5 also introduced the `<video>` element that embeds a media player in the HTML document. Its usage is similar to the `<audio>` element. The video source can be defined as the `src` attribute of the `<video>` tag as well as more than one `<source>` element to provide alternative video files for the browser to choose from.

```
<video src="path/to/video file" controls>
</video>
```

If you define `<source>` elements, you need to specify the MIME type of the corresponding video format. You may also provide a fallback text if the browser fails to load any of the specified video formats. The following is an example of HTML and the browser output for the `<video>` element.

[Table](#)

Much like the audio control, the interactive video interface also has the start/pause button and the progress bar. It also has a control to toggle between the Fullscreen and normal view. The control also exposes the same events as the audio control and can be programmatically manipulated.

<iframe> Tag

Short for *inline frame*, the `<iframe>` element embeds another HTML document in the current web page. The `src` attribute's value is the URL of the page to be embedded.

```
<iframe src="URL", title, width, height></iframe>
```

The optional width and height attributes are in pixels. Defining the title attribute enhances accessibility, as it proves useful for screen readers.

One typical use case of the `iframe` element is to display a Google map of a desired location. You need to obtain the URL from <https://www.google.com/maps> and use it as the `src` attribute in the `<iframe>` element.

```
<iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d317719.588(
0.43124630246002843!3d51.528179754192685!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f1:
width="100%"
height="300"
title="London UK">
</iframe>
```

Your web page shows the map for London, UK, as shown in Figure [5-7](#).

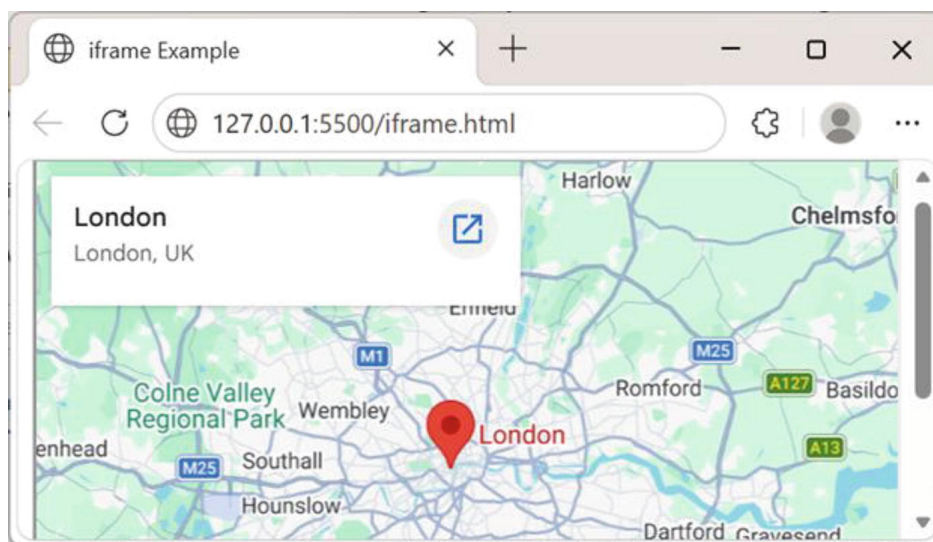


Figure 5-7 Using <iframe>

Text-Level Semantic Tags

In the previous chapter, you learned about a few semantic equivalents of the presentation tags: , , and <ins> for <bold>, <italic>, and <u>. HTML has a few more inline semantic tags. They provide context and meaning to specific text blocks inside the paragraphs or headings. Some of them are explained here in this section. For the other tags, you may refer to MDN Web Docs (<https://developer.mozilla.org/en-US/>).

<code>: Text marked by this tag represents a short fragment of computer code. The following shows an example.

[Table](#)

<var>: This element highlights a variable in a mathematical expression or programming context. The following shows an example.

[Table](#)

<mark>: This tag marks text that should be highlighted because it is relevant in another context. The following shows an example.

[Table](#)

<blockquote>: This element implies that its enclosed text is an extended quotation. It adds an indentation before the text. The cite attribute refers to the URL of the source of the quote. Usually, this is rendered visually by indentation. The URL for the source of the quotation is assigned to the cite attribute of the <blockquote> tag, while the <cite> element renders its text representation. The following shows an example.

[Table](#)

Points to Remember

- Most of the tags in HTML are semantic in nature as they describe the meaning of the content and help search engines and assistive technologies understand the structure of the document.
- HTML5 introduced many new semantic elements.
- Elements related to describing the layout of a document include `<header>`, `<nav>`, `<article>`, among others.
- Using the `<dialog>` element, you can include a dialog box that pops up in the browser, enabling user interactions.
- HTML5 includes disclosure elements that can expand/collapse content with the help of `<details>` and `<summary>` elements.
- One of the important enhancements added by HTML5 is multimedia elements, which you can use to embed audio or video content in a web page.
- In addition, HTML has many text-based semantic elements that format a given piece of inline text as may be desired.

Summary

The concept of semantic elements was introduced in the previous chapter. This chapter elaborates on it by providing a detailed discussion of its various types, such as layout-related, multimedia, and text-related semantic tags.

The next chapter covers some more elements related to forms and tables in an HTML document.

6. Tables and Forms in HTML

Malhar Lathkar¹ 

(¹) Near Rachewad Hospital, Nanded, Maharashtra, India

The previous two chapters focused on various HTML elements for layout and how to embed static content of different types. This chapter discusses tables and forms. Tables are used to display a collection of structured data in an elegant layout. Forms let web applications collect data from the user so it can be sent to the server for further processing.

Tables

In a typical HTML document, you are invariably required to present a collection of structured information, particularly when it can be put in rows and columns. For example, a price list, sales data, or statistical reports. A tabular representation becomes the most meaningful and semantically correct way of presenting them on a web page. More often than not, this data is fetched from the databases on the server and placed in a tabular form. This improves readability and enhances accessibility and maintainability of web content.

Designers also used to employ HTML tables for web page layout. However, with the emphasis on CSS in layout management, tables are now used primarily to represent data that can be neatly expressed in rows and columns.

The `<table>` Element

The entire table construct in an HTML document is placed inside the `<table>` and `</table>` tags, acting as the root element. Just as the web page itself, the HTML table can have one or more of the following sections.

The `<caption>` element represents a suitable title of the table that acts as its accessible name or description.

The `<thead>` element has information about the table's columns, in the form of column headers.

The `<tbody>` element contains the table's main data in the form of a set of table rows.

The `<tfoot>` element acts as the footer of a table, usually a summary of the columns, for example the sum of each column.

```
<table>
  <caption>
    <!--Caption text-->
  </caption>
  <thead>
    <!--header rows-->
  </thead>
  <tbody>
    <!--table rows-->
  </tbody>
  <tfoot>
    <!--footer rows-->
  </tfoot>
</table>
```

Each of these sections consists of one or more row elements represented by `<tr>` and `</tr>` tags. Fundamentally, the table is composed of rows, and each row is a collection of cells (usually the same number of cells in each row). The rows are semantically grouped in named sections as shown in Figure [6-1](#).

	Caption		
thead	header1	header2	header3
↑	Cell1	Cell2	Cell3
tbody	Cell4	Cell5	Cell6
↓	footer1	footer2	footer3
tfoot	footer1	footer2	footer3

Figure 6-1 Components of a table

A row can have one or more data cells marked by `<td>` and `</td>` tags, or header cells marked by `<th>` and `</th>` tags.

The `<tr>` Element

The HTML table uses the row-major approach for constructing the tabular layout. It means that one or more cells are grouped in a row, one or more rows in a section, and sections in a table.

```
<tr>
  <!--row cells-->
</tr>
```

A row may have data cells (`<td>` element) or header cells (`<th>` element).

The `<td>` Element

The content marked by `<td>` and `</td>` tags is a child of the `<tr>` element, representing a cell of a table grid.

```
<td>
  cell data
</td>
```

The content of a table cell may be text, an anchor element with a hyperlink, or an image. Usually, the number of `<td>` elements in each row of a table is the same. However, you can design a complex table

structure with an unequal number of cells per row using the `colspan` and `rowspan` elements.

To create a basic HTML table, only the `<table>`, `<tr>`, and `<td>` elements are necessary. Here is a simple example.

```
<table border="1">
  <tr>
    <td>Name</td>
    <td>Age</td>
    <td>Marks</td>
  </tr>
  <tr>
    <td>Dennis</td>
    <td>23</td>
    <td>45</td>
  </tr>
  <tr>
    <td>Sarah</td>
    <td>29</td>
    <td>78</td>
  </tr>
</table>
```

Your browser renders the table shown in Figure [6-2](#).

Name	Age	Marks
Dennis	23	45
Sarah	29	78

Figure 6-2 HTML table

The <th> Element

Let's highlight the table's column headings more prominently. The <th> element achieve this. It defines a cell as the header of a group of table cells and may be used as a child of the <tr> element.

```
<th>
  <!--Column header-->
</th>
```

The <th> element may appear in any row cell, not necessarily in the top row, to show the column header. The contents of this element are usually displayed in bold and centered and give meaning to the associated column or row.

Let's use the <th> tag to highlight the column headers in the first row of the preceding table.

```
<tr>
  <th>Name</th>
  <th>Age</th>
  <th>Marks</th>
</tr>
```

The scope Attribute

The scope attribute in the <th> element defines which cells the header element relates to. If set to row, the header relates to all cells of the row it belongs to. On the other hand, if it is set to col, the header relates to all cells of the column it belongs to. If the scope attribute is not specified, the applicable set of cells is selected automatically.

Consider the table in Figure [6-2](#). It already shows the column headers. Let's show the names as the row headers by placing the names in a <th> tag with its scope parameter set to row.

```
<table border="1">
  <tr>
    <th scope="col">Name</th>
```

```

        <th scope="col">Age</th>
        <th scope="col">Marks</th>
    </tr>
    <tr>
        <th scope="row">Dennis</th>
        <td>23</td>
        <td>45</td>
    </tr>
    <tr>
        <th scope="row">Sarah</th>
        <td>29</td>
        <td>78</td>
    </tr>
</table>

```

While this code renders a visually correct HTML table, it lacks the semantic details for the benefit of the assistive technologies. Hence, it is always recommended to use the table-related tags such as `<thead>`, `<tbody>`, and `<tfoot>`.

The `<thead>` Element

One or more table rows marked by the `<tr>` and `</tr>` tags are placed inside the `<thead>` element to indicate that they form the head of a table, having information about the table's columns. Usually the cells are column headers (marked by `<th>` elements). In the preceding example, the first row forms the table header; hence it is put inside the `<thead>` element.

```

<thead>
    <tr>
        <th scope="col">Name</th>
        <th scope="col">Age</th>
        <th scope="col">Marks</th>
    </tr>
</thead>

```


Instead of letting the browser infer the table's header information, it always helps to explicitly define the `<thead>` element. One of the major advantages is that when printing a long table, Browsers automatically repeat `<thead>` on every page.

The `<tfoot>` Element

Just as `<thead>`, the `<tfoot>` element adds a semantically important section to the table. The rows, usually at the bottom of a table, are placed inside the `<tfoot>` and `</tfoot>` tags. These rows typically contain certain summary information about the table.

Let's add a footer element to the table in the preceding example and place a row inside it to show the average of the data in the Age and Marks columns.

```
<tfoot>
  <tr>
    <th scope="row">Average</th>
    <td>26</td>
    <td>61.5</td>
  </tr>
</tfoot>
```

The `<tbody>` Element

Browsers treat the rows between the `<thead>` and the `<tfoot>` elements as the body of the table, although it is recommended to explicitly put them inside the `<tbody>` element.

```
<tbody>
  <!--one or more rows-->
</tbody>
```

The `<caption>` Element

You can provide the table an accessible name or description by specifying a caption (or title). If you want to have one, the `<caption>`

element must be the first child of its parent <table> element. By default, the caption appears above the table grid in a center-aligned manner.

```
<table>
  <caption>
    Title of the table
  </caption>
  <thead>
    <!--table rows-->
</table>
```

Let's add "Student List" as the caption to this table.

```
<table border="1">
  <caption>Student List</caption>
  <thead>
    <tr>
      <th scope="col">Name</th>
      <th scope="col">Age</th>
      <th scope="col">Marks</th>
    </tr>
  </thead>
  <tr>
    <th scope="row">Dennis</th>
    <td>23</td>
    <td>45</td>
  </tr>
  <tr>
    <th scope="row">Sarah</th>
    <td>29</td>
    <td>78</td>
  </tr>
  <tfoot>
    <tr>
      <th scope="row">Average</th>
```

```
<td>26</td>
<td>61.5</td>
</tr>
</tfoot>
</table>
```

Your browser now shows a table with row and column headers and the caption, as seen in Figure [6-3](#).

Name	Age	Marks
Dennis	23	45
Sarah	29	78
Average	26	61.5

Figure 6-3 Headers and caption

Spanning

The browser allots one row and one column to each cell in a row (represented by either a `<td>` or `<th>` element). However, you may face a situation where a single cell is required to span multiple adjacent rows or columns. In some tabular layouts, such as a timetable, comparison table, or invoices, you can use the `rowspan` and `colspan` attributes available to the `<td>` and `<th>` elements to merge multiple cells.

colspan: the value of this attribute is also a non-negative integer and indicates how many columns the data cell spans. The default value is 1. It makes the cell stretch horizontally over the specified number of cells to its right.

```
<td colspan=n>data</td>
```

rowspan: the value of this attribute is a non-negative integer and indicates how many rows the data cell spans or extends. The default value is 1. It makes the cell stretch vertically over the specified number of cells below it.

```
<td rowspan=n>data</td>
```

The following HTML code demonstrates the use of rowspan and colspan attributes.

```
<table border="1">
  <caption>Student List</caption>
  <thead>
    <tr>
      <th scope="col">Name</th>
      <th scope="col" colspan="2">Marks</th>
    </tr>
  </thead>
  <tr>
    <th scope="row" rowspan=2>Dennis</th>
    <td>Theory</td>
    <td>45</td>
  </tr>
  <tr>
    <td>Practical</td>
    <td>60</td>
  </tr>
  <tr>
    <th scope="row" rowspan=2>Sarah</th>
    <td>Theory</td>
    <td>78</td>
  </tr>
  <tr>
```

```
<td>Practical</td>
<td>82</td>
</tr>
</table>
```

The browser output is shown in Figure [6-4](#).

Student List		
Name	Marks	
Dennis	Theory	45
	Practical	60
Sarah	Theory	78
	Practical	82

Figure 6-4 Spanning

Forms

In the early days of the World Wide Web, HTML documents mainly consisted of hyperlinked text structured using simple markup elements such as headings, paragraphs, and lists. These pages were largely static and offered limited user interaction. The introduction of web forms in HTML 2.0 enabled websites to collect user input and submit it to server-side programs, commonly implemented using CGI scripts, which processed the data and returned dynamically generated HTML responses.

Over time, the form functionality has evolved through integration with CSS for presentation and JavaScript for dynamic behavior. With HTML5, forms gained richer input types, built-in client-side validation,

and improved accessibility, making them essential to modern web applications.

The <form> Element

An HTML form (also called a *web form*) is represented by the <form> element in the HTML document. It is a structural element that acts as a container for one or more form-associated controls intended to collect the user-specified data, which is then submitted for further processing. The interactive form controls that collect the data from the user are placed inside the <form> and </form> tags.

```
<form>
  <!-- form-associated controls -->
  <input type = "submit">
</form>
```

The form data needs to be submitted either to a server-side script or handled by client-side JavaScript code. Form submission is normally triggered by a submit button, usually placed as the last control in the form. The URL and the HTTP method used for submission are provided as the values of the action and method attributes of the form element.

The action Attribute

Upon submission, the form data is processed by the URL set as the value of the action attribute of a form. The URL may point to a CGI script on the remote server, an API endpoint, or a controller written in server-side technologies such as Java, Ruby, or PHP.

The method Attribute

This attribute refers to the HTTP method used for submitting the form to the specified URL. The default value of the method attribute is "get" (HTTP GET method), causing the form data to be appended to the action URL with a "?" symbol. The form data is assembled as a query string, with the name and value of each control separated by the "&" symbol.

Consider the following HTML code with a form.

```
<form action="/upload" method="get">
  <input type="text" name="name" value="Raman">
  <input type="text" name="age" value=20>
  <input type = "submit">
</form>
```

The browser constructs the query string as follows.

```
name=Raman&age=20
```

This query string is then appended to the action URL.

```
/upload?name=Raman&age=20
```

The browser then sends an HTTP GET request to the action URL.

When the method attribute is set to "post", the form data is placed in the request body, encoded as per the enctype attribute. The browser then sends an HTTP POST request to the action URL.

The GET method exposes the query string in the address bar of your browser; hence, it is not advised to be used when sending sensitive data such as passwords to the server.

The enctype Attribute

This attribute specifies how the form data is encoded before submission. The enctype must be defined when the form's HTTP method is POST. By default, its value is "application/x-www-form-urlencoded", which handles simple name-value pairs of the form's controls.

The other possible value of enctype is "text/plain", used only as a debugging aid, and never in any production environment.

Setting the enctype attribute to "multipart/form-data" is vital, especially when you need to upload files. HTML's input element of file

type lets you choose a file from the local filesystem and send it as part of the form's payload.

```
<form action="/upload" method="post"
enctype="multipart/form-data">
    <!-- other form controls -->
    <input type="file" id="file" name="file">
    <input type = "submit">
</form>
```

When used, the browser creates a request body that is divided into multiple parts, each separated by a boundary string.

HTML has several form-associated controls, which can be suitably classified in the following categories.

- **Text input controls** let the user input data from the keyboard.
- **Selection controls** offer a collection of predefined options for the user to choose from.
- **Button controls** trigger certain actions.
- **Grouping controls** allow the controls to be arranged in semantically significant groups.
- **Output controls** display calculated or dynamic results.

The <label> Element

The label element is a semantic control associated with any other control that displays a non-editable text caption, thereby enhancing usability and accessibility.

```
<label for="control-id">Label text</label>
```

The for attribute is assigned to the id attribute of a control to which you want to associate this label. The caption is marked by the <label> and </label> tags.

```
<label for="email">Email address</label>
<input type="email" id="email" name="email">
```


In this example, the id attribute of the input element is "email". The same is set as the value of the for attribute of the label to establish the association. (The input element and its attributes are discussed later in this chapter).

The label element may also be used with an alternate syntax that shows the association more implicitly.

```
<label>
  Email address
  <input type="email" id="email" name="email">
</label>
```

The <input> Element

Most of the form-associated interactive controls are different types of the input element. It is one of the most important HTML elements. Depending on the input type, the HTML page can accept input in various forms, such as text from the keyboard, choosing from available options, and selecting from interactive widgets like calendars, pickers, and sliders. Input elements of different types expose different attributes that help browsers validate the data without any JavaScript code.

Although the attributes vary depending on the input type, id, name, and value attributes are usually defined in almost every instance of the input element. The following is a typical input element declaration.

```
<input type="..." id="..." name="..." value="...">
```

The id (stands for identifier) attribute of the input element should be unique in the whole document, and is necessary for associating a label to it, as well as for accessing it programmatically. The id attribute is in fact a global attribute, available for all HTML elements (not just the input element). The name and value attributes are essential for form submission.

The name Attribute

Even though not mandatory, the name is treated as a required attribute. It is a string that gives a suitable name to the input element. Its value, along with this name, is submitted when the form data is submitted. Obviously, if it is not specified or is empty, the input's value is not submitted.

The value Attribute

You can optionally define the initial value of the input control, which the user can change at runtime. This value is submitted to the action URL. This value can be set or retrieved at any time using JavaScript.

The type Attribute

The most important attribute of the input element, as it specifies how it accepts the input. The default value of the type attribute is "text".

type="text"

When specified, the browser renders a basic single-line text field. The value attribute is a string that contains the current value entered into the text field. If the input element is of text type, you can define the maxlength and minlength attributes (to check the length of the string entered), which the browser uses for client-side validation.

```
<input type="text" id="name" name="name"
minlength="4" maxlength="8">
```

type="password"

The browser renders a one-line text field in which the text entered is obscured by replacing each character with a symbol such as the asterisk ("*"). Used mainly to let the user input a password, the maxlength and minlength are also generally defined. The required attribute, if specified, makes it a mandatory form field.

```
<input type="password" id="pass" name="password"
minlength="8" required>
```

type="number"

The input element of this type is designed to let the user enter only a number. The browser applies a built-in validation to reject non-numerical entries. The following is an example.

```
<input type="number" id="age" name="age" min="1"
max="100">
```

The browser also provides arrows to the right of the text field to let the user increase and decrease the value, as seen in Figure [6-5](#).



Figure 6-5 Number type

type="date"

The input element of date type allows the user to enter a date. You may specify the starting value.

```
<input type="date" id="date" name="date"
value="2025-12-15">
```

A calendar icon to the right of the control pops up a date picker, which can also be used to select the desired date instead of using the keyboard. See Figure [6-6](#).

12 / 15 / 2025 

December 2025   

Su	Mo	Tu	We	Th	Fr	Sa
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Clear Today

Figure 6-6 Date picker

Apart from these, HTML5 also offers text input controls of email, url, and tel types to accept text and perform respective validations.

Several input types let the user select from a collection of available options.

type="range"

The input element of range type lets the user set its value with the help of a slider control. The default range is 0 to 100, which the min and max attributes can explicitly define.

```
<input type="range" id="volume" name="volume"
min="1" max="10">
```

The default value of the slider is halfway between the specified min and max attributes. See Figure [6-7](#).



Figure 6-7 Range field

type="radio"

The radio type's input element is rendered as a round button that can be clicked by the user. Each radio element must have an associated label to give it a distinct, accessible name.

```
<input type="radio" id="id" name="name"
value="value">
<label for="id">Radio Button text</label>
```

Radio elements are typically used to provide a group of options from which only one can be selected. Multiple radio buttons with the same name attribute form a group of mutually exclusive choices. When one is selected, the others in the group get deselected automatically.

```
<input type="radio" id="id1" name="GroupName"
value="Option1">
<label for="id1">Option 1</label>
```

```
<input type="radio" id="id2" name="GroupName"
value="Option2">
<label for="id2">Option 2</label>
```

You can also include the checked attribute in one of the radio elements in the group, which is displayed in the selected state in the browser. Also, if any radio element in the group has the required attribute, selecting one from the group becomes mandatory.

Generally, the caption appears to the right of the radio symbol. Hence, the label element is defined after the input element. However, proper CSS code can modify the positioning.

Here is an example of three radio buttons to let the user record their preference for receiving an OTP for authentication.

```
<p>How do you want to receive OTP?</p>
<input type="radio" id="sms" name="OTP"
value="SMS" checked>
<label for="sms">By SMS</label>

<input type="radio" id="phone" name="OTP"
value="Phone">
<label for="phone">By Phone</label>

<input type="radio" id="email" name="OTP"
value="Email">
<label for="email">By Email</label>
```

When rendered, the browser displays the three radio buttons, as seen in Figure [6-8](#).

How do you want to receive OTP?

☒ By SMS ☐ By Phone ☐ By Email

Figure 6-8 Radio button

type="checkbox"

The input element of the checkbox type follows almost the same syntax as the radio element. A checkbox is rendered as a tickable square.

```
<input type="checkbox" id="id" name="name"
value="value">
<label for="id">Checkbox text</label>
```

The value attribute is a string sent to the action URL when the form is submitted. If not specified, the default value is "on". Unlike the radio buttons, the checkboxes let the user select multiple options from the available list, even if the name attribute is the same for all.

In the following code snippet, the three checkbox elements have different name attributes.

```
<p>Which Languages Do You Know?</p>
<input type="checkbox" id="java" name="java">
<label for="java">Java</label>

<input type="checkbox" id="python" name="python">
<label for="python">Python</label>

<input type="checkbox" id="js" name="js">
<label for="js">JavaScript</label>
```

The browser renders the checkboxes shown in Figure [6-9](#).

Which Languages Do You Know?

☐ Java ☐ Python ☐ JavaScript

Figure 6-9 Checkbox

If you tick the first two boxes and submit, the browser builds the query string as follows.

```
java=on&python=on
```

However, if you group the three checkboxes by assigning the same name attribute to all of them, you need to specify a unique value to each.

```
<p>Which Languages Do You Know?</p>
<input type="checkbox" id="java" name="language"
value="java">
<label for="java">Java</label>

<input type="checkbox" id="python" name="language"
value="python">
<label for="python">Python</label>

<input type="checkbox" id="js" name="language"
value="javascript">
<label for="js">JavaScript</label>
```

Now, if you submit the form after selecting the first two checkboxes, the query string will be formed as follows.

```
language=java&language=python
```

type="file"

The input element of file type causes the browser to render a button with “Choose File” caption. When clicked, the Open dialog pops up, letting the user choose a file from the computer’s filesystem.

```
<input type="file" id="file" name="file">
```

value attribute: The path of the selected file becomes the value attribute and is displayed alongside the button in the browser window.

multiple attribute: The multiple attribute needs to be included to let the user select multiple files.

accept attribute: The value of this attribute acts as a filter for the files to be displayed (and selected) from the file input dialog box. For example, the following input element lets the user choose only the files of specified types.

```
<input type="file" id="file" name="file"
accept="image/png, image/jpeg">
```

The file type specifier may be any MIME type string or a file name extension. Use of wildcard characters is allowed; for example, .jpg, .pdf, audio/*, video/*, and image/.*

The file type input element only fetches the metadata of the selected file(s). To upload, it must be a form element whose enctype is set to "multipart/form-data". Upon the form submission, the browser includes the file data in a binary form in the request body.

```
<form action="/submit" method="post"
enctype="multipart/form-data">
    <!--Other form fields -->
    <input type="file" id="file" name="file">
</form>
```

type="submit"

Input elements of certain types are rendered as buttons, submit being one of them. When the user clicks the submit button, the browser runs validations on the form elements, collects the validated data, and submits the name-value pairs to the action URL, using the specified method.

```
<input type="submit" value="Submit">
```

The value attribute is optional (the default is "Submit") and is used as the button caption.

You can use a <button> element of "submit" type as an equivalent of a submit-type input element. However, <button> is not a self-closing

element.

```
<button type="submit">Submit</button>
```

type="image"

The input element and the button element of submit type both have a text caption. The input element of image type also renders a button that can display an image as a caption.

```
<input type="image" src="submit.png">
```

type="reset"

Like the submit type, the input element of reset type is also rendered as a button. A click event on this button resets all the controls in the form to their initial values.

```
<input type="reset" value="Reset">
```

The value attribute is optional (the default is “Reset”) and is used as the button caption.

You can use a <button> element of "reset" type as an equivalent of a submit type input element. However, it is not a self-closing element.

```
<button type="reset">Reset</button>
```

type="hidden"

The input element of the hidden type defines an invisible input field. The browser does not render this element, but its value is submitted with the form. Both the name and the value attributes must be defined.

```
<input type="hidden" id="id" name="hidden_field" value="hidden_value">
```

Developers typically employ a hidden field to pass security tokens and session data to the server along with the form’s contents.

<select> Element

In a web form, a drop-down list of predefined options is created using the select element. Each option in the list is marked by <option> and </option> tags. A group of option elements is embedded within the <select> and </select> tags.

```
<select id="id" name="name">
  <option value="Option 1">Option 1</option>
  <option value="Option 2">Option 2</option>
  <option value="Option 3">Option 3</option>
</select>
```

The select element has three key attributes: id, name, and value. The id attribute is used to associate a label. When the form is submitted, the browser includes the name attribute of the select element and the value attribute of the option selected by the user.

The select element can be more effectively used with the following attributes.

multiple: allows users to select more than one option from the drop-down list.

size: the number of visible options in the drop-down list.

The <select> element also supports the <optgroup> element to show that certain options belong to a group. However, it doesn't affect the form submission, as only the values of selected options are included.

The <datalist> Element

The <datalist> element also displays embedded options in a drop-down list, as the select element does. However, the options act only as suggestions and, unlike the select element, do not restrict the user to selecting only from the available options.

The <datalist> element is used in association with an input element of list type. This lets the user enter a string in the input box as well as choose from the suggested options in the data list.

```
<input list="id" name="name">
```

```
<datalist id="id">  
  <option value="Option 1"></option>  
  <option value="Option 2"></option>  
  <option value="Option 3"></option>  
</datalist>
```

For the input element to use the data list as a list of suggestions, its list attribute must be equal to the id of a <datalist> element. The input element, initially displayed as a text field, displays a drop-down when the user clicks in it.

The following example clearly emphasizes the difference between the <select> element and the <input> element with a list attribute associated with the <datalist> element.

```
<!--Datalist with input list-->  
<input list="dl" name="browser">  
  
<datalist id="dl">  
  <option value="Chrome"></option>  
  <option value="Firefox"></option>  
  <option value="Safari"></option>  
  <option value="Edge"></option>  
</datalist>  
  
<!--Select Dropdown-->  
<select id="select" name="browser">  
  <option value="Chrome">Chrome</option>  
  <option value="Firefox">Firefox</option>  
  <option value="Safari">Safari</option>  
  <option value="Edge">Edge</option>  
</select>
```

The input element drops down a list of suggestions as you type, while the select element lets you choose only from the available options. See Figure [6-10](#) and Figure [6-11](#).

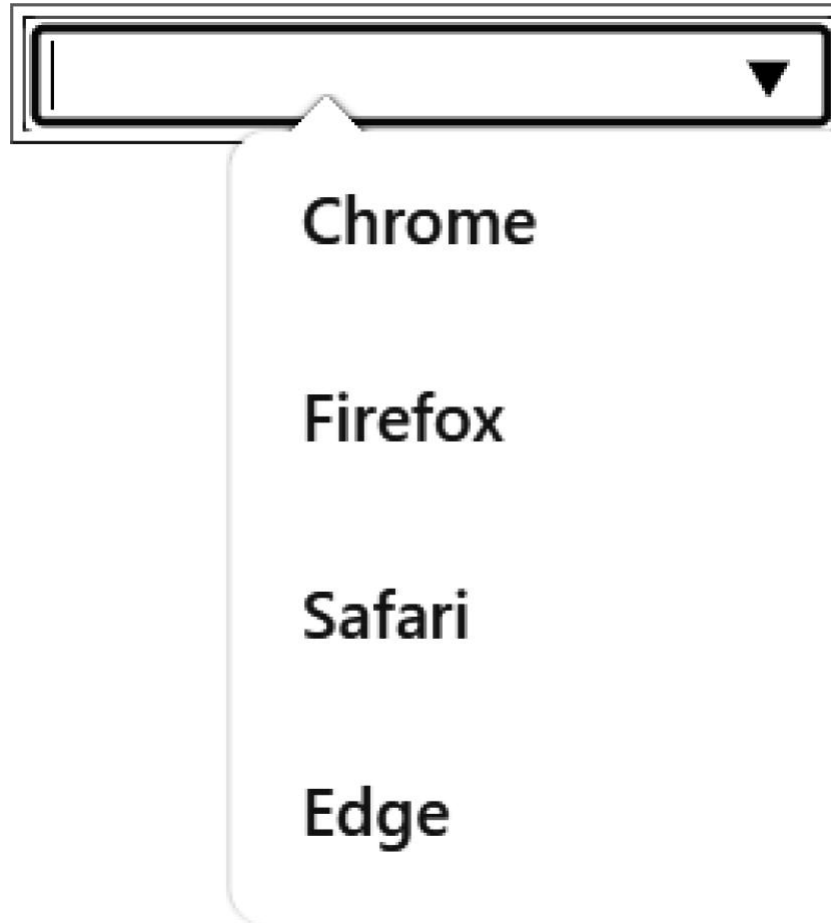


Figure 6-10 The datalist element



Figure 6-11 The select element

<textarea> Element

The input element of text type presents a single-line text field. If the web form needs a control where the user can enter free-form text spanning multiple lines, the textarea element should be used. Typical use cases for a textarea element include entering comments or feedback.

```
<textarea name="name">Initial text</textarea>
```

In addition to the usual attributes such as id and name (the value attribute is not defined to set the initial value), the textarea element is defined with the cols and rows attributes to decide the size of the multiline input field.

The cols Attribute

The cols attribute defines the width of the text control attribute. Its value is a positive integer, with the default being 20. It represents the number of characters visible in a single line of the control.

The rows Attribute

The rows attribute decides how many text lines of the control are visible. The value must be a positive integer; the default value is 2.

Note that the cols and rows attributes define the visible size of the textarea element. If the actual content entered by the user exceeds the columns or rows as specified, the browser automatically provides scrollbars.

The wrap Attribute

The default value of the wrap attribute of the textarea element is soft. If set to hard, each line is no longer than the cols attribute.

Most form-associated elements support the read-only, required, and disabled attributes. The read-only attribute indicates that the user cannot modify the value, and the required attribute specifies that the user must fill in a value for the form to be validated. The disabled attribute means that the user cannot interact with the element.

Points to Remember

- In HTML, <table> is a semantic element, used to display a collection of structured information in the form of rows and columns.
- A table is a collection of rows, each represented by the <tr> element.
- A row, in turn, is a collection of cells, marked by <td> or <th> elements.
- Rows in the table are semantically grouped in thead, tbody, and tfoot elements, and can have a caption.
- HTML 2.0 introduced the <form> element to add interactivity in web pages.
- A web form is a collection of associated controls that let the user enter data or choose from the available options.
- After entering the required data, the form is submitted to the URL mentioned in its action attribute.
- GET and POST methods are used to submit the data to the specified URL.

- The form is constructed with the various types of the `<input>` element. The types that accept text input include text, number, and date.
 - The range, radio, and checkbox types allow the user to select from the available options.
 - You can use the textarea control to enter long, multiline text.
-

Summary

This chapter discusses in detail two of the most often used elements in HTML, namely the table and form elements. Developers often need to display certain data in a structured form, mostly extracted from some databases. HTML tables offer a convenient tool for the purpose. Web forms are a permanent feature in most client-facing web applications. The data entered by the user is collected in various form-associated elements such as text fields, radio and checkbox controls, and drop-down lists. All these control types have been explained in this chapter with appropriate code examples.

This concludes the discussion on HTML. The next chapter addresses another important web technology: CSS.

Knowledge Check: Part II

1. **Which declaration must be the very first line of an HTML5 document to ensure the browser renders it in standards mode?**
 - A. `<html lang="en">`
 - B. `<!DOCTYPE html>`
 - C. `<head>`
 - D. `<?xml version="1.0"?>`
2. **Which tag is the container for metadata (like character encoding and title) that is not displayed directly on the page?**
 - A. `<body>`
 - B. `<header>`
 - C. `<head>`
 - D. `<footer>`
3. **What is the primary purpose of the lang attribute in the `<html>` tag?**
 - A. To set the font family for the document
 - B. To help search engines and screen readers identify the language of the page
 - C. To translate the website automatically

D. To define the programming language of the back end

4.

Which element inside the <head> section defines the text to be displayed in the title bar of a browser tab?

A. <script>

B.

C. <title>

D. <nav>

5.

Which tag defines the main content of the HTML document that is visible to the user?

A. <head>

B. <body>

C. <html>

D. <main>

6. **Which tag is used to specify the character encoding for the document?**

A. <meta charset="UTF-8">

B. <link rel="charset">

C. <style encoding="UTF-8">

D. <base charset="UTF-8">

7. **Which semantic tag should be used to wrap the primary navigation links?**
- A. `<section>`
 - B. `<nav>`
 - C. `<header>`
 - D. `<aside>`
8. **Which heading level represents the most important heading on the page?**
- A. `<h6>`
 - B. `<h10>`
 - C. `<h1>`
 - D. `<header>`
9. **What is the benefit of using semantic tags like `<article>` and `<section>` over generic `<div>` tags?**
- A. They automatically apply CSS styles.
 - B. They reduce the file size of the HTML document.
 - C. They provide meaning to the structure for screen readers and SEO.
 - D. They are required for JavaScript to run.

10. **Which tag is best suited for a sidebar containing advertisements or related links?**
- A. `<main>`
 - B. `<aside>`
 - C. `<article>`
 - D. `<footer>`
11. **Which element should be used for a self-contained composition like a blog post or news story?**
- A. `<section>`
 - B. `<div>`
 - C. `<article>`
 - D. `<nav>`
12. **Which tag is used to represent the footer for its nearest sectioning content?**
- A. `<bottom>`
 - B. `<end>`
 - C. `<footer>`
 - D. `<base>`

13. **Which attribute is used to provide the file path for an image in the tag?**

- A. href
- B. link
- C. src
- D. alt

14. **What does the alt attribute do for an image?**

- A. It changes the size of the image.
- B. It provides a text description if the image cannot be loaded or for visually impaired users.
- C. It creates a link to another page.
- D. It adds a border to the image.

15. **Which tag is used to create a hyperlink?**

- A. <link>
- B. <a>
- C. <href>
- D. <nav>

16. **Which attribute is added to an <a> tag to open a link in a new tab?**

- A. target="_new"
- B. mode="blank"
- C. target="_blank"
- D. rel="external"

17. **Which tag is used to embed a video directly into a page?**

- A. <embed>
- B. <video>
- C. <movie>
- D. <iframe>

18. **How do you link an external CSS stylesheet to an HTML document?**

- A. Using a <script> tag in the body
- B. Using a <link> tag in the head
- C. Using an <a> tag in the head
- D. Using an tag with a .css source

19. **Which tag is used to create an unordered (bulleted) list?**

- A.
- B.

- `<li>`
- C. `<ul>`
- D. `<list>`

20. **Which form attribute determines where the data is sent upon submission?**

- A. `method`
- B. `action`
- C. `target`
- D. `name`

21. **Which input type should be used for sensitive data like a password?**

- A. `type="text"`
- B. `type="hidden"`
- C. `type="password"`
- D. `type="mask"`

22. **What is the purpose of the `<label>` tag?**

- A. To validate the data
- B. To submit the form
- C. To provide a text section for an input and increase the

to provide a text caption for an input and increase the clickable area

- D. To store the data

23. **Which element is used to create a multiline text input?**

- A. `<input type="text">`
- B. `<textarea>`
- C. `<input type="textarea">`
- D. `<textblock>`

24. **If you want a user to pick exactly ONE option from a list, which input type is best?**

- A. `type="checkbox"`
- B. `type="radio"`
- C. `type="submit"`
- D. `type="button"`

25. **Which attribute makes a form field mandatory?**

- A. `validate="true"`
- B. `needed`
- C. `required`
- D. `placeholder`

Answers with Explanations

1. **B**
Explanation: The DOCTYPE declaration is a preamble that tells the browser which version of HTML is being used. (Chapter [4](#))
2. **C**
Explanation: The <head> section contains machine-readable information about the document. (Chapter [4](#))
3. **B**
Explanation: Specifying the language (e.g., lang="en") is crucial for accessibility and SEO. (Chapter [4](#))
4. **C**
Explanation: The <title> element is used to display text in the title bar of the browser's tab. (Chapter [4](#))
5. **B**
Explanation: The <body> tag itself is the primary container for all visible content. (Chapter [4](#))
6. **A**
Explanation: The <meta charset> tag is essential to ensure special characters and different languages display correctly. (Chapter [4](#))
7. **B**
Explanation: The <nav> element is specifically intended for major navigation blocks. (Chapter [5](#))
8. **C**
Explanation: <h1> should be used for the main title of the page; there is usually only one per page. (Chapter [4](#))
9. **C**

Explanation: Semantic HTML tells the browser and search engines what the content actually is, rather than just how it should look. (Chapter [5](#))

10.

B

Explanation: The <aside> element is for content tangentially related to the surrounding content. (Chapter [5](#))

11.

C

Explanation: An <article> should make sense on its own and be distributable independently from the rest of the site. (Chapter [5](#))

12.

C

Explanation: The <footer> typically contains authorship info, copyright data, or contact links. (Chapter [5](#))

13.

C

Explanation: src stands for "source". (Chapter [5](#))

14.

B

Explanation: Alt text is essential for web accessibility. (Chapter [5](#))

15.

B

Explanation: The <a> (anchor) tag, combined with the href attribute, creates links. (Chapter [5](#))

16.

C

Explanation: target="_blank" is the standard attribute for opening links in a new context. (Chapter [5](#))

17.

B

Explanation: The <video> tag allows for native video playback. (Chapter [5](#))

18.

B

Explanation: <link rel="stylesheet" href="style.css"> is the standard method. (Chapter [4](#))

-

19. **C**
Explanation: stands for Unordered List. (Chapter [4](#))
20. **B**
Explanation: The action attribute contains the URL of the processing script. (Chapter [6](#))
21. **C**
Explanation: It masks the input characters as they are typed. (Chapter [6](#))
22. **C**
Explanation: Labels improve accessibility and usability. (Chapter [6](#))
23. **B**
Explanation: textarea is not an input type. Instead, <textarea> is its own tag and supports multiple lines. (Chapter [6](#))
24. **B**
Explanation: Radio buttons allow a single selection from a group. (Chapter [6](#))
25. **C**
Explanation: The required attribute prevents the form from being submitted if the field is empty. (Chapter [6](#))

Part III

CSS for Content Styling

CSS for Content Styling

CSS, along with HTML, handles the design aspect of web application development. The objectives of the LPI Web Essentials exam (Exam Code: 030-100) allocate 8 weightage points (out of 40) to the candidate's knowledge of CSS. **Topic 033: CSS Content Styling** in the objectives covers concepts such as CSS properties, different types of CSS selectors (such as ID selector and class selector), and combinators. It aims to test the candidate's understanding of concepts such as CSS inheritance and specificity.

Topic 033 has the following subtopics.

- 033.1 CSS Basics (weight: 1)
- 033.2 CSS Selectors and Style Application (weight: 3)
- 033.3 CSS Styling (weight: 2)
- 033.4 CSS Box Model and Layout (weight: 2)

Part III of this book corresponds to topic 033, and has three chapters. Here's what each chapter covers:

Chapter [7](#)

Chapter [7](#) emphasizes the advantage of using CSS to stylize the HTML elements. You learn about the composition of a CSS rule. This chapter explains how to use the in-line method of defining styles as well as including stylesheets.

Chapter [8](#)

Each CSS rule is associated with one or more CSS selectors. In Chapter [8](#), you learn about the different types of selectors and how they can be combined to apply CSS styles to a certain category of HTML elements. This chapter also introduces the concept of pseudo-classes. You also learn how CSS inheritance and specificity work.

Chapter [9](#)

The final chapter in this part covers different layout patterns. Starting with the default flow layout, you learn about the box model. This chapter also explains how the flexbox model works. Features such as grid layout and media queries are also part of Chapter [9](#).

This part concludes with a practice set of 25 MCQs (with the explanation of the correct answer for each).

7. CSS Basics

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

CSS (Cascading Style Sheets) is a core technology that powers the World Wide Web. While HTML is responsible for the content and structure of a web page, CSS helps define its styling. CSS is a type of declarative language and can be used to specify the presentation of a document written in any markup language, such as HTML, SVG, MathML, or XHTML. Hence, although originally proposed as cascading HTML style sheets (CHSS) by Håkon Wium Lie, “HTML” was later removed from the name.

Before CSS

Before the adoption of CSS specifications by the W3C, HTML consisted of a small set of tags, primarily focusing on the content and structure of the web document, with no emphasis on presentation, which depended entirely on the browser’s interpretation.

Even as the W3C standardized HTML 2.0 in 1995, it focused on structural and content-oriented elements.

The HTML 3.2 version introduced some elements and attributes to control the appearance of the web page.

The `<font>` Element

The `<font>` element controls the appearance of text within a web page by specifying attributes like size, color, and face.


```
<font color="blue" size="4">Hello, World!</font>
```

The specified font is applied to any text it wraps (apart from heading tags, which have a predefined size); for example, paragraph text, list elements, or other Inline text.

The color attribute defines the text color; it can be a named color or a color value in the hexadecimal #RRGGBB format.

The size attribute is a number between 1 and 7, 3 being the default font size. It may be defined to have a size relative to the default; for example, +1 or -2.

The face attribute is a string of one or more font names separated by commas. The browser renders the most suitable font.

However, this element has been deprecated in favor of CSS properties like font-size, color, and font-family, although the browsers still support it for backward compatibility.

The <center> Element

In pre-HTML5 versions, the center element is a container element that displays its contents horizontally centered. The element doesn't have any attributes to customize its behavior.

```
<center>
  Content to be centered
</center>
```

There may be more than one HTML element wrapped inside the <center> and </center> tags; all appear centered.

Before the introduction of CSS, the <center> element was used for page titles, navigation headers, and logos. CSS offers more flexible options for centering (horizontally as well as vertically—the <center> element takes care of only horizontal centering). Hence, the <center> element has been deprecated in HTML 4 in favor of the CSS text-align property.

The <big> and <small> Elements

Both of these elements were introduced in HTML 3.2. The <big> element has been deprecated since version 4.1. However, <small> continues to be available in the current version of HTML, although with a different meaning.

The text enclosed in the <big> element is rendered at a font size one level larger than the surrounding text. The size is limited by the browser's maximum permitted font size.

Table

CSS provides a more flexible way to control font size (the font-size property), so the <big> element was deprecated.

The <small> element initially behaved just as the <big> element, rendering its text content in a size smaller by one level than the surrounding text. It was deprecated in HTML 4.1 but reintroduced in HTML5 with a new semantic definition. The <small> element is now used to represent side comments and small print, like copyright and legal text.

Before the CSS specifications were formalized, HTML 3.2 had introduced certain attributes to add styling to some HTML elements. Most of them got deprecated with the HTML 4.x versions.

The bgcolor Attribute

The bgcolor attribute is still available for use with some HTML elements like body and table.

```
<body bgcolor="yellow">  
<table bgcolor="#eee">
```

Modern standards recommend using the CSS background-color property.

The background Attribute

Similarly, the background attribute can be defined with the <body> and <table> elements to specify the image to be used as the background.

```
<body background="image.jpg">
```

Although still supported, the CSS background-image property should be preferred instead.

The width and height Attributes

Both these attributes are defined in many HTML elements, such as body, table, img, and others. Both must be specified as an integer number of pixels.

```
<table width="600">  
<img width="200" height="100">
```

Except for the media-related elements (for example <video>, <canvas>, etc.), the respective CSS properties, width and height, are prescribed.

The border Attribute

Elements like table and img have the border attribute that creates a frame surrounding the element with the given width in pixels.

```
<table border="1">  
<img border="0">
```

Modern web development standards recommend using the CSS property of the same name (border) instead, as it provides more flexibility.

In general, most of the presentational attributes (hspace and vspace in the img element, padding attributes in the table element, etc.) have been deprecated (and indeed marked as obsolete in HTML5).

Inline Style

As a first step toward separating structure from presentation, W3C introduced style as a global attribute. The global attributes are common to all HTML elements. The value of the style attribute is a string that includes a semicolon-separated list of CSS properties. A colon symbol (:) separates each CSS property and its value.

```
<element style="prop1: value1; prop2: value2;
">content</element>
```

For example, the following HTML statement displays the paragraph text in white and centered in a div element with a gray background and a border of 2 pixels.

[*Table*](#)

The style attribute replaced the presentational elements like font, center, and attributes like bgcolor. However, the style attribute needs to be defined for each HTML element; hence, it becomes repetitive and difficult to maintain consistency.

The <style> Element

The style element is placed inside the <head> section of the web document to define the CSS styling rules available for the entire document. You can define multiple CSS rules in a style element. The rules are wrapped inside a pair of opening and closing curly braces.

```
<head>
  <style> {
    rule 1
    rule 2
  }
</style>
</head>
```

Each rule defines the behavior of a CSS selector in the form of one or more CSS property: value pairs separated by semicolons and placed inside curly braces.

```
selector {  
    prop1: value;  
    prop2: value  
}
```

The browser finds the HTML elements that match the patterns defined in the rules and applies the styles. The selector may be an element type, an attribute, a pseudo-class, or a combination of all.

The following HTML snippet renders all the div elements in the document with a gray background. The text content of all paragraph elements has a white color and center alignment.

```
<head>  
<style>  
    div {  
        background: gray;  
        border: 2px;  
    }  
    p {  
        color: white;  
        text-align: center;  
    }  
</style>  
</head>
```

When the styling rules are defined in the head section of a page, there should be no text content, nor any HTML; only CSS syntax is allowed. You can put comments though, enclosed inside the `/*` and `*/` symbols.

Note that while the style element may be defined inside the body section, it is discouraged as it affects the rendering order. The `<style>`

element in the head section is standard practice and acts as the internal stylesheet for the current HTML page.

External Stylesheets

This is another method for defining the styling rules. All the CSS rules are stored in a separate text file as a stylesheet with the .css extension. Any HTML document can include this stylesheet with the help of the <link> element. Set its rel attribute to stylesheet and use the path of the required stylesheet file as the value of the href attribute.

Let's rewrite the CSS rules from the previous example to show how the external stylesheet works.

Remove the <style> element from the head section, and save its CSS code as styles.css (make sure that the <style> and </style> tags are excluded).

```
div {  
    background: gray;  
    border: 2px;  
}  
p {  
    color: white;  
    text-align: center;  
}
```

Add a <link> element in the head section of the HTML code.

```
<head>  
<link rel="stylesheet" href="styles.css">  
</head>
```

Now, when you open the web page in your browser, it shows the same result: the paragraph with a gray background and the text in white.

Compared to the other two methods (using the style attribute for each element and defining a style element with CSS rules for various HTML elements), linking an external stylesheet to an HTML page is considered best practice because it is reusable and easily scalable.

CSS Syntax

CSS as a styling language has a simple yet well-defined syntax for its basic building blocks. A stylesheet (a text file with the .css extension or the <style> element inside the head section) is a collection of CSS rules. Each rule has a declaration block with one or more declarations put inside the left and right curly braces. Each declaration is a pair of a CSS property and value separated by a colon. The declarations in a block are separated by a semicolon, and the declaration block in turn is preceded by one or more selectors (separated by a comma if more than one).

```
sel1, sel2 {  
    prop1 : value;  
    prop2 : value;  
}
```

Putting a semicolon at the end of the last declaration in the block isn't necessary, but is considered good practice.

In CSS, a property is a human-readable name of a specific presentational aspect of a certain HTML element. Each property may be assigned a numeric value or from a predefined set of constants.

```
/*property with numeric value*/  
width: 200px;
```

```
/*property with value as a predefined constant*/  
text-align: center;
```

Some properties may be available for multiple elements. For example, the text-based elements (such as paragraphs, headings, and

anchors) share properties like color, font-size, and text-align.

CSS Properties

As per the latest CSS specifications, there are more than 500 properties. They are grouped in various modules. Each module defines properties belonging to a particular functional category.

All the font-related properties are defined in the CSS fonts module. These properties control the face (family), size, weight, style, and so forth.

The properties in the display module deal with the formatting of the box tree that is generated from the document element tree.

The color and opacity properties that help you define the colors, blending, and the opacity effects of an HTML element are defined in the color module.

The properties in the images module control features such as orientation and positioning of an image element.

The CSS text module defines properties that control how the browser performs text manipulation, such as line breaking, justification, and alignment. The text decoration module provides properties about underlining, generating shadows, and emphasis. These properties can help improve the accessibility, functionality, and aesthetics of the text.

To transform HTML elements in two- or three-dimensional space and control the transition of elements between two CSS values, you have the transform and transition modules, respectively.

The next section discusses the most commonly used properties.

Color

One of the most frequently used properties, the color property can be used with many HTML elements. It lets you change the color of the text, the background color of a container, the border color, and so on. The value of this property may be any of the predefined color keywords, the

value returned by the `rgb()` function, or a hexadecimal representation of the color value.

For example, the CSS declaration that sets the paragraph text to blue is as follows.

```
p {  
    color: blue;  
}
```

This uses a predefined color name as the value. All the common color names (black, white, red, purple, green, yellow, and blue), as predefined in the CSS specifications, are accepted as color keywords. CSS has more than a hundred predefined color keywords, including the legacy colors (blue, green, red, etc.), various shades of gray (gray, darkgray, lightgrey, etc.), different variants of primary colors such as lightblue, darkgreen, violet, darkred, and many more.

CSS defines a very useful `rgb()` function. It uses three comma-separated parameters, each specifying the numeric value (between 0 and 255) of red, green, and blue color respectively. The `rgba()` function accepts a fourth parameter corresponding to the alpha (transparency) value, which ranges from 0—standing for full transparency to 1 indicating fully opaque. It may also be given as a percentage of transparency between 0% and 100%.

The blue color in the preceding CSS declaration may be specified with the `rgb()` function as follows.

```
p {  
    color: rgb(0,0,255); /* Blue*/  
}
```

You can also use the RGB values in percentages: 0% for 0 and 100% for 255.

```
p {  
    color: rgb(0%,0%,100%); /* Blue*/
```

```
}
```

The modern CSS syntax for the `rgb()` function allows you to specify RGB values separated by spaces instead of commas.

```
p {  
    color: rgb(0 0 255); /* Blue*/  
}
```

A third alternative to specify a color is using its hexadecimal value. A color is a combination of red, green, and blue components, with values ranging from 0 to 255, represented in hexadecimal as 00 and FF, respectively. To express an RGB value in hexadecimal, concatenate the two-digit hex value of each component and prefix it with the # symbol. As a result, the blue color is represented by #0000FF as its hex value. If you want to display the paragraph text in dark blue (with a decimal value of 139 and a hex value of 8B), the corresponding CSS declaration would be as follows.

```
p {  
    color: #00008b; /* Dark Blue */  
}
```

The opacity property (the opposite of transparency) in CSS defines the degree to which content behind an element is hidden. Its value lies in the range 0.0 to 1.0, or expressed as a percentage in the range 0% to 100% to represent the value of its alpha channel. This property is usually defined for container elements.

```
div {  
    opacity: 0.6;  
}
```

Background

In CSS, background is one of the shorthand properties. Although you can set different background style properties (such as color, image, origin, size, or repeat) at once, you can also configure them individually.

The background-color property is often used to set the background color of an HTML element, especially body, div, and table; although technically it can be defined for any. Just like the color property, its value can be set as a predefined color name, with the rgb() or rgba() function, or as a hexadecimal string.

```
div {  
    background-color: lightgray;  
}
```

You can replace lightgray with #d3d3d3 or rgb(211, 211, 211).

The background-image property configures an element to have one or more images as its background. Images of all the standard image formats (jpg, png, bmp, etc.) are accepted. The url() designator, which has a string parameter representing the path of the image to be used as a background, is the value of this property.

```
body {  
    background-image: url("logo.jpg");  
}
```

If the image dimensions are smaller than the element, it is repeatedly rendered as a background, both horizontally and vertically. The background-repeat property controls this behavior. The following are accepted values for the background-repeat property.

- **repeat:** the default behavior, the image repeats in both directions, filling the entire element.
- **no-repeat:** The image is displayed only once.
- **repeat-x:** The image repeats only horizontally.
- **repeat-y:** The image repeats only vertically.

- **space:** The image repeats as many times as possible without clipping.
- **round:** The image repeats and is automatically stretched or squished to fill the available space.

To place the background image at a specific position, use the `background-position` property. The `background-position-x` property sets the horizontal position with possible values `left`, `center`, and `right`. To set the vertical position, use `top`, `center`, and `bottom` values. You can use a two-value syntax for the `background-position` to set both positions at once. Let's place the `gradient.png` as a background for the body section at the center, both vertically and horizontally.

```
body {  
    background-image: url('gradient.png');  
    background-position: center center;  
}
```

Assuming the HTML body has an `h1` element with the text `Hello World`, it will be rendered as shown in Figure [7-1](#).

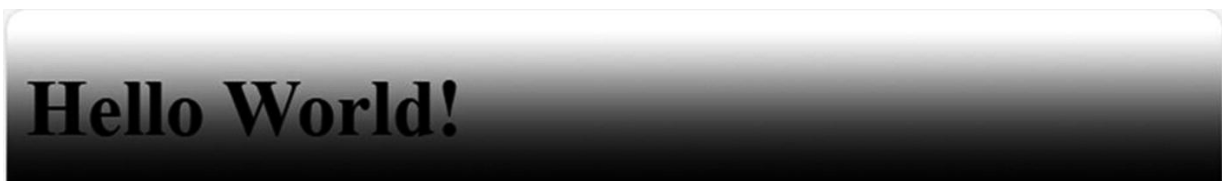


Figure 7-1 The background property

To set all the background-related properties at once, you can use the `background` property, a shorthand CSS property.

```
body {  
    background: gray url('img.png') no-repeat  
bottom center;  
}
```

Border

Every HTML element is rendered as a rectangular box. Its content is surrounded by (by default invisible) a line that forms the border. In CSS, border is a shorthand property that deals with several elements of the border, such as the width, the style (whether dotted or dashed, etc.), and the color. In fact, each of them is a shorthand property itself.

The border-width property is a shorthand for border-top-width, border-right-width, border-bottom-width, and border-left-width properties. The value of each of them can be one of thin, medium, or thick (default is medium). Note that the border-style property must be defined to see the effect of the width property.

```
div {  
    border-style: solid;  
    border-top-width: thick;  
    border-bottom-width: thin;  
}
```

```
/*width shorthand*/  
div {  
    border-style: solid;  
    border-width: thick;  
}
```

The border-style property is a group of four: border-bottom-style, border-left-style, border-right-style, and border-top-style. The following are accepted values.

- **none** displays no border.
- **hidden** displays no border.
- **dotted** displays a series of rounded dots.
- **dashed** displays a series of short square-ended dashes or line segments.
- **solid** displays a single, straight, solid line.
- **double** displays two straight lines that add up to the pixel size defined by border-width.

- **groove** displays a border with a carved appearance.
- **ridge** opposite of groove, displays a border with an extruded appearance.
- **inset** displays a border that makes the element appear embedded.
- **outset** displays an embossed border, and is the opposite of inset.

Let's format a div element to have double borders on the top and bottom, with a thick width, whereas the left and right borders are thin and use dashed and dotted styles.

```
div {
  border-style: double;
  border-left-style: dashed;
  border-right-width: dotted;
  border-width: thick;
  border-left-width: thin;
  border-right-width: thin;
}
```

The div element simply renders a heading (h1) element. Figure [7-2](#) shows how it is rendered.



Figure 7-2 The border property

Likewise, the border-color is a shorthand property for setting the color of each side of the border (i.e., border-bottom-color, border-left-color, border-right-color, and border-top-color properties). The value of each property can be set as a predefined color name, with the rgb() function, or a corresponding hex value.

```
/*color of top border of div element
```

```
div {  
    border-style: solid;  
    border-top-color: blue;  
}  
  
/*using shorthand border-color*/  
div {  
    border-style: solid;  
    border-color: red yellow green blue;  
}
```

In shorthand, colors are applied in the order of top, right, bottom, and left border.

The top-level background-color shorthand CSS property allows you to configure different aspects of the border at once.

```
div {  
    border: thick solid red;  
}
```

The div element will now have a thick solid and red border.
Let's look at another example.

```
div {  
    border: thick solid red;  
    border-right: thin solid blue;  
    border-left: thin solid blue;  
}
```

This results in top and bottom borders rendered as thick and solid, with red color; and the right and left sides are thin, solid, and blue.

Font

CSS has another shorthand property: the font property. Like the other shorthands, the font property is useful for setting various font-related

properties at once; you can configure each of them individually. The font property has another use: it sets a given element's font to a specified system font.

One of the constituents of this shorthand is the font-family property. Its value is a string containing a comma-separated list of font names; the browser uses the first available font in the list, which generally ends with a generic family name (such as serif, sans-serif, monospace) as a fallback option.

```
h1 {  
    font-family: "Times New Roman", Times, serif;  
}  
h2 {  
    font-family: Arial, Helvetica, sans-serif;  
}  
h3 {  
    font-family: "Comic Sans MS", "Brush Script MT",  
    cursive;  
}  
h4 {  
    font-family: "Courier New", Courier, monospace;  
}
```

Various heading levels are displayed with different fonts.

Hello World!

Hello World!

Hello World!

Hello World!

As the name suggests, the font-size property sets the size of the font to be applied. Its value can be specified as predefined constants (accepted values are xx-small, x-small, small, medium, large, x-large, xx-large, and xxx-large), or relative to the parent element (smaller or larger), or as an absolute value in px or pt, or in relative units (em and rem). The following are a couple of examples of the font-size property.

```
h1 {  
    font-size: 32px;  
}  
  
h2 {  
    font-size: xx-large;  
}  
  
p {  
    font-size: 16px;  
}  
  
li {  
    font-size: 12pt;  
}
```

You can choose how the text should appear in style (normal, italic, or oblique) by setting the font-style property of the element. If set to italic, the browser selects an italic version of the font if one is available; if no italic version of the face is available, the browser selects a font that is classified as oblique. If neither is available, the style is artificially simulated. Conversely, if set to oblique and not available, an italic font is chosen. You can also specify the angle for the text slanting.

```
p {  
    font-style: italic;  
}
```

The font-weight property controls the thickness of the text. You can use normal or bold keywords, as well as lighter and bolder for font weight relative to the parent element. If you need to set an absolute weight, use a numeric value between 1 and 1000, in multiples of 100.

The shortform font property lets you define all the preceding properties in one declaration.

```
p {  
  font: italic bold large Arial;  
}
```

This renders the paragraph text in Arial, with italic style, bold weight, and a large font size.

Text

Some other CSS properties that manipulate how text is rendered are denoted as text-* properties, as their names have a text- prefix. A frequently used text-align property aligns the text embedded in elements (heading, paragraph, table, cell, etc.). The accepted values are start, end, left, right, center, justify, and match parent.

```
h1 {  
  text-align: center;  
}
```

The text-indent property specifies the horizontal space before the first line of text in a block-level element, such as a paragraph. It has a numeric value, expressed in em, px, or percentage.

```
p {  
  text-indent: 20px;  
}
```

The text-wrap property controls how the text inside an element (such as a paragraph) is wrapped (or not wrapped at all). Its default

value is wrap, preventing the text from overflowing the container. Predictably, nowrap causes the overflow. The other values are balance, pretty, and stable.

```
h1 {  
  text-wrap: balance;  
}
```

Margin

The browser treats every HTML element as a rectangular box. The margin property sets the margin area on all the sides of the box. The margin area, bounded by the margin edge, extends the border area to include an empty area used to separate the element from its neighbors. Its dimensions are the margin box width and height.

CSS has the margin shorthand property that lets you define the margin area in px, em, or in percentage terms. When set to auto, the browser selects a suitable margin value for each component of this shorthand (margin-top, margin-bottom, margin-left, and margin-right).

```
div {  
  margin: 10px 20px;  
}
```

This sets the left and right margins as 20 px, while the top and bottom margins are 10 px.

height and width

In CSS, these two properties determine the dimensions of an element's content area. The specified values apply to height and width as long as they remain within the minimum and maximum values (min-width and max-width for width, and min-height and max-height for height).

By default, the browser sets min-height and max-height to auto. The px unit is used to define height in absolute terms. For relative dimensions, these properties are expressed in percentages, relative to

the parent element's dimensions or in viewport units, which are relative to the browser window size (vw for width, vh for height).

```
div {  
  width: 50%;  
  height: 50%;  
}
```

Units

The value of a CSS property can be set as a predefined constant or as a numeric value expressed in various units. The following is an example of a CSS property with a predefined constant.

```
font-size: large;
```

The following is an example of the same property expressed in numeric form.

```
font-size: 32px;
```

There are certain length-type properties such as

- width, height
- margin, padding
- font-size
- border-width

Some are angular properties (such as gradient—expressed in deg or rad) or time properties (duration and delay defined in seconds or milliseconds).

Numeric values can be expressed in absolute or relative units.

Absolute Units

These units represent fixed physical lengths. The following are some examples.

px: In CSS, one pixel represents one dot on a computer monitor or mobile display. One CSS pixel is 1/96th of an inch.

pt: It stands for point. 1pt is equivalent to 1/72th of an inch. This unit is generally used in printing technology.

Traditional length units in (inch: 1 in is equivalent to 2.54 cm or 96 px), cm (centimeter: 1 cm is equivalent to 96 px/2.54), and mm (millimeter: 1 mm is equivalent to 1 cm/10) may also be used.

Physical units (cm, in) are mainly useful for print stylesheets. Absolute units are not responsive, as pixel density may vary across different monitors or mobile displays. They do not depend on parent or viewport size.

Relative Units

Relative units are based on some other object's size (such as font, viewport, or its parent element). Hence, the same relative measurement might be a different physical measurement depending on the value that the unit is related to.

em: The size of the current element's font.

rem: Expressed as a multiple of the root element size. The default root element is <html>. It remains the same throughout the document.

vw: 1% of the viewport's width (i.e., the width of the browser).

vh: 1% of the viewport's height (i.e., the height of the browser).

You can also set a length-related value (such as width or height) of an element as a percentage of the same property of its parent element. Examples are width and height, margins, and background position.

Points to Remember

- W3C adopted the CSS standards to separate the structure of the web document from its presentation.
- Before CSS, developers had to rely on presentational elements and properties like font and bgcolor.

- A universal style attribute further enhanced the styling capabilities.
 - The introduction of style elements is a major step toward separating structure and styling.
 - If the stylesheet is defined inside the head element, it becomes the internal stylesheet of the current document.
 - A text file with the .css extension included in the head section of the web page with the link element becomes the external stylesheet.
 - The CSS declaration assigns a value to the CSS property. A CSS declaration is a collection of CSS declarations applicable to one or more CSS selectors.
 - Over 500 CSS properties are defined in different modules.
 - The properties related to color, background, and border are the most frequently used.
 - A CSS property's value may be a predefined constant or a numeric value.
 - The numeric value is either expressed in absolute or relative units.
-

Summary

CSS plays an important part in web page design that is great-looking and responsive. This chapter covers the basic features of the CSS language. Starting with the CSS syntax, this chapter explains some of the most frequently used CSS properties. It also explains the absolute and relative units used to define the values of various CSS properties.

The next chapter discusses various selectors in CSS as well as how features like inheritance and specificity work in CSS.

8. CSS Selectors

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

In the previous chapter, you learned that a style sheet has one or more CSS rules and that each rule is a collection of CSS declarations that decide the styling of certain HTML elements. The browser's CSS engine chooses the elements based on the CSS selectors (one or multiple) associated with each rule. The examples in the previous chapter used various HTML elements (such as `div`, `p`, or `h1`, etc.) as selectors to show how a CSS rule is formed and how it is applied. This chapter explores the various other types of selectors so that it becomes possible to precisely select and style the desired elements based on their type, attributes, class, and so on.

Universal Selector

The asterisk (*) symbol—a widely accepted wild card character—is used to define a universal CSS selector. As the name suggests, the browser applies the styles defined in this selector to all the elements in the document.

```
* {  
    color: green;  
}
```

This causes the text color of the entire document to be green. However, the universal selector has the lowest possible specificity. Almost any other selector will override it, and hence it is not advisable to use it for general styling. Developers use a universal selector mainly for box-sizing, very small resets, and debugging and generally don't retain it in the production version of website design.

```
* {  
  box-sizing: border-box;  
}
```

Element Selector

This is the most commonly used type of selector. It helps you ensure that all the elements of the same type are styled uniformly. Generally, developers use this type of selector to define uniform styling for elements like paragraphs, ordered and unordered lists, and headers in different documents on a website.

The previous chapter primarily used the element selectors to demonstrate common CSS properties. Let's define different element selectors to design a simple blogging website.

First, let's choose the font for all the text-based elements in the page by defining the font-family property in the body element.

```
body {  
  font-family: Arial, sans-serif;  
  margin: 0;  
  padding: 10px;  
  line-height: 1.6;  
}
```

The header element would contain a nav bar, the blog title as an h1 element, and a div element to display a short description of the blog. The nav element is styled with the following rule.


```
nav {  
    background-color: rgb(25, 25, 25);  
    text-align: right;  
}
```

The anchor element displays the navigation links in white and has a 15px margin on the right.

```
a {  
    color: white;  
    text-decoration: none;  
    margin: 0 15px;  
    font-size: 16px;  
}
```

Let's format the blog title to be displayed in h1 tags and center-aligned.

```
h1 {  
    color: #333;  
    text-align: center;  
}
```

The <div> element for the blog description is center-aligned.

```
div {  
    font-size: 1.1em;  
    font-style: italic;  
    text-align: center;  
    margin-bottom: 15px;  
}
```

Finally, the page footer is designed to be shown in a smaller font, white color, and center-aligned.

```
footer {
```

```
text-align: center;
background-color: #191919;
color: white;
font-size: small;
margin-top: 10px;
}
```

Put all the preceding CSS rules in styles.css and use the <link> tag in the head element of the page to include it in the HTML code of index.html. The following is the body element code.

```
<body>
  <header>
    <nav>
      <a href="#">Home</a>
      <a href="#">About</a>
      <a href="#">Contact</a>
    </nav>

    <h1>My Tech Blog</h1>
    <div>
      <p>Welcome to my Tech Blog. Here I share
tips on web development.<br>
      From HTML basics to advanced CSS tricks.
</p>
    </div>
  </header>

  <article>
    <h2>Introduction to CSS Selectors</h2>

    <p>In the previous post, I discussed the
basics of CSS. In this post, I'll explain how
powerful CSS selectors are.</p>
```

```
<p>Stay tuned for more updates!</p>
</article>

<footer>
  <p>&copy; 2026 My Tech Blog. All rights
reserved.</p>
</footer>

</body>
```

Ensure that both these files (index.html and styles.css) are in the same folder. When opened in your browser, the index page would look like Figure [8-1](#).



Figure 8-1 Element selector

ID Selector

Although optional, every HTML element can have an ID as a unique global attribute. You can form a CSS rule to target an element of a specific ID and format it as required.

The selector in this rule must be the ID prefixed by the # symbol. Hence, the CSS rule that searches for a given ID should be declared as follows.

```
#idname {  
    /* CSS declarations */  
}
```

There are three paragraph elements in the HTML code used in the previous section. Let's assign an ID to the one containing a short description of the blog.

```
<p id="intro">Welcome to my Tech Blog. Here I  
share tips on web development.<br>  
From HTML basics to advanced CSS tricks.</p>
```

Let's add a CSS rule corresponding to this ID so that the text is displayed against a light gray background and adds a shadow.

```
#intro {  
    text-shadow: 1px 1px;  
    background-color: lightgray;  
}
```

Similarly, the third <p> element is modified to define an ID.

```
<p id="end">Stay tuned for more updates!</p>
```

The post-ending paragraph will be displayed in bold by the following rule.

```
#end {  
    font-weight: bold;
```

}

As a result, the home page is rendered as shown in Figure [8-2](#).



Figure 8-2 ID selector

Class Selector

HTML 4.0 added certain global attributes, class being one of them (id, style, lang being among the others). The class attribute groups elements logically and allows you to apply common styles to various HTML elements. The value of the class attribute is a string that consists of a user-specified class name (if there is more than one class, they are separated by whitespace).

```
<element class="classname">contents</element>
```

To define a corresponding CSS rule, its selector is the class name prefixed by the period (.) symbol.

```
.classname {  
    /* CSS declarations */  
}
```

Let's take the example used in the previous section, where the first `<p>` element had the ID attribute set to "intro". Replace the ID attribute with the class attribute.

```
<p class="intro">Welcome to my Tech Blog. Here I  
share tips on web development.<br>  
From HTML basics to advanced CSS tricks.</p>
```

Similarly, add `class="end"` instead of the ID attribute of the third `<p>` element as follows.

```
<p class="end">Stay tuned for more updates!</p>
```

Obviously, we need to change the stylesheet rules. Instead of the ID selectors for intro and end IDs, define the class selectors as follows.

```
.intro {  
    text-shadow: 1px 1px black;  
    background-color: lightgray;  
}  
  
.end {  
    font-weight: bold;  
}
```

This won't change the appearance of the page. It's just that the IDs are replaced by class attributes.

Class selectors are extremely powerful when you need to apply similar styling to multiple elements of different types. In our example,

there are two heading elements (h1 and h2) for the blog title and the post title, respectively. Let's display their text content in a different font than the one set as default by the CSS rule of the parent body element.

For this purpose, add the class="heading" attribute in the h1 and h2 element code.

```
<h1 class="heading">My Tech Blog</h1>
```

```
<h2 class="heading">Introduction to CSS  
Selectors</h2>
```

Then, add a class selector in the styles.css file.

```
.heading {  
    color: #333;  
    font-family: Georgia;  
}
```

You'll find that the blog and post headings are displayed in the Georgia font. See Figure [8-3](#).



Figure 8-3 Class selector

Attribute Selector

The attribute selectors are best suited when you need to select and apply styles to HTML elements with a specific attribute or an attribute with a given value.

To define an attribute selector, the valid CSS attribute is placed inside the square brackets. Here is the general syntax.

```
[attribute] {  
    /*CSS declarations*/  
}
```

This applies the styling as per the CSS declarations to any element having the given attribute. If the square bracket is preceded by an

HTML element, the browser's CSS engine selects only the specified element with the given attribute.

This makes the CSS rule apply only to elements that have a specific attribute with an exact value.

```
[attribute="value"] {  
    /*CSS declarations*/  
}
```

Let's say you want to open a page that shows certain detailed information in different languages when the user clicks on a link around the text such as "Click here for more information". Further, you want this message in each language to be displayed in a different font.

So, let's put this text along with its translations in French, Chinese, and Hindi inside <p> elements, each with the lang attribute set to the language codes "en", "fr", "zh", and "hi", respectively.

```
<p lang="en">  
    <a href="#">Click here</a> for more  
information in English  
</p>  
<p lang="fr">  
    <a href="#">Cliquez ici</a> pour plus  
d'informations en français  
</p>  
<p lang="zh">  
    <a href="#">点击此处获取更多中文信息</a>  
</p>  
<p lang="hi">  
    अधिक जानकारी के लिए <a href="#">यहाँ क्लिक  
करें</a>  
</p>
```

The href attributes are set to a dummy hyperlink here, but can be replaced by actual pages. Define CSS attribute selector rules for each

language code in the stylesheet.

```
[lang="en"] {  
    font-family: Arial, sans-serif;  
}  
  
[lang="fr"] {  
    font-family: "Georgia", serif;  
}  
  
[lang="zh"] {  
    font-family: "Noto Sans SC", sans-serif;  
}  
  
[lang="hi"] {  
    font-family: "Noto Sans Devanagari", sans-serif;  
}
```

You should now get the same message in different languages, each displayed in a different font.

[Click here](#) for more information in English

[Cliquez ici](#) pour plus d'informations en français

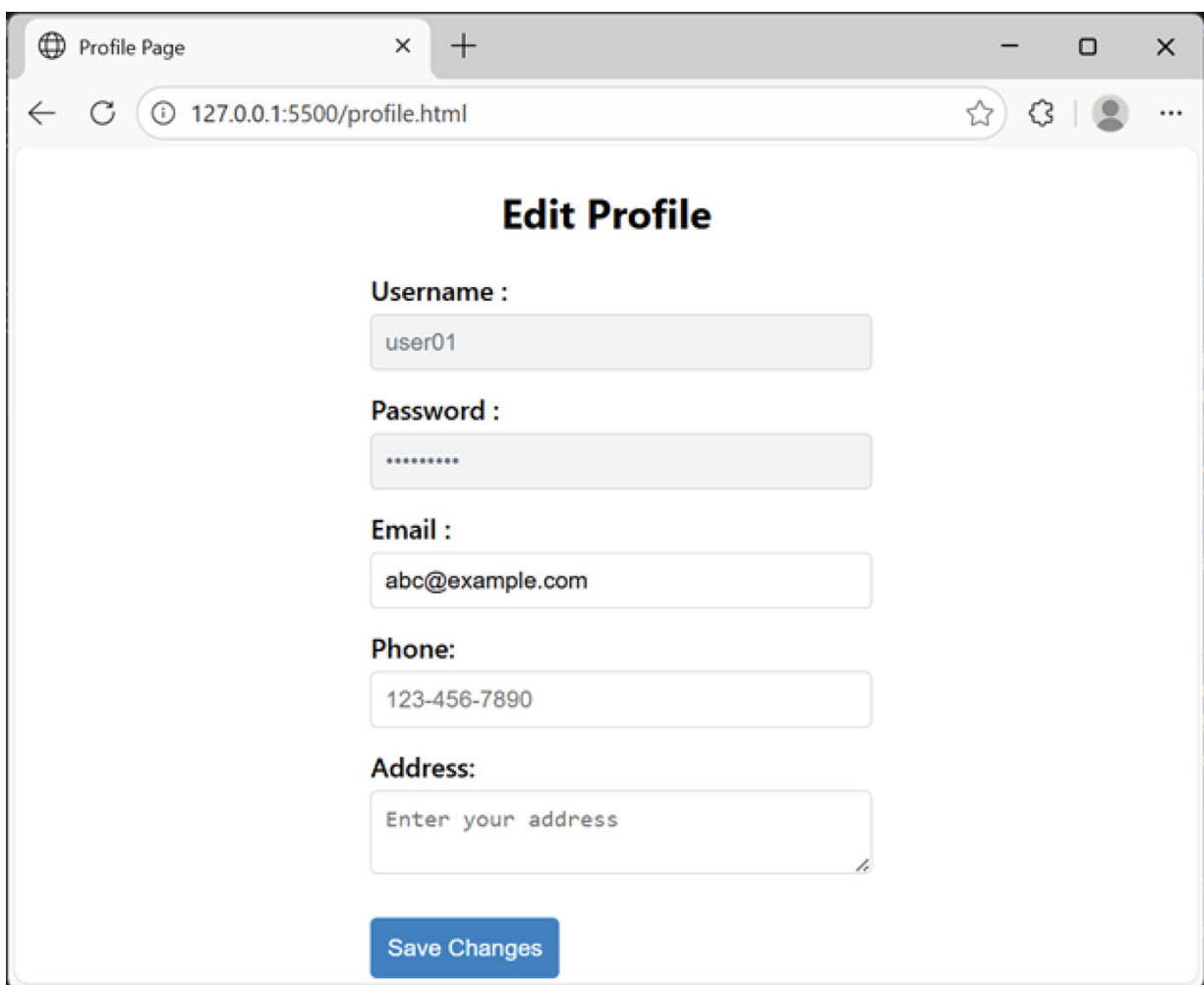
[点击此处获取更多中文信息](#)

[अधिक जानकारी के लिए यहाँ क्लिक करें](#)

Another use case is rendering editable text boxes in an HTML form in a different style than the ones that are read-only. In a typical form that lets the user edit his personal data, fields like the username and password are read-only. In contrast, fields like email, phone, and address are usually editable. To distinguish read-only text fields from the others, define an attribute selector for the readonly attribute as follows.

```
[readonly] {  
  background-color: #f1f3f5;  
  color: #6c757d;  
  border: 2px dashed #adb5bd;  
  cursor: not-allowed;  
}
```

The different styling of the two types of text input fields renders as shown in Figure [8-4](#).



The screenshot shows a web browser window titled 'Profile Page' with the address bar displaying '127.0.0.1:5500/profile.html'. The main content area is titled 'Edit Profile' and contains a form with the following fields:

- Username :** A text input field with a solid border and the value 'user01'.
- Password :** A password input field with a solid border and masked characters '.....'.
- Email :** A text input field with a solid border and the value 'abc@example.com'.
- Phone:** A text input field with a solid border and the value '123-456-7890'.
- Address:** A text input field with a dashed border and the placeholder text 'Enter your address'.

A blue 'Save Changes' button is located at the bottom of the form.

Figure 8-4 Attribute selector

Apart from the preceding two, CSS allows you to define an attribute selector in many different ways.

```
[attribute~=value] {  
    /*CSS declarations*/  
}
```

When you add the ~symbol, it tells the CSS engine to select elements with the specified attribute's value containing the given word.

```
[attribute|=value] {  
    /*CSS declarations*/  
}
```

The pipe symbol (|) indicates that elements with the given attribute's value match exactly the given value or can begin with the value immediately followed by a hyphen symbol (-).

```
[attribute^=value] {  
    /*CSS declarations*/  
}
```

All the elements whose specified attribute starts with the value assigned in the selector.

```
[attribute$=value] {  
    /*CSS declarations*/  
}
```

This CSS rule applies to elements whose specified attribute ends with the value assigned in the selector.

```
[attribute*=value] {  
    /*CSS declarations*/  
}
```

Finally, this rule lets the browser select elements whose attribute contains a specific value.

Combinators

A CSS rule can have more than one selector. A combinator has (only) two selectors, separated by a specific symbol that defines the relationship between the two. Based on their relationship, the browser can uniquely identify the elements in the document tree and apply the styling declared in the rule.

For each type of combinator, CSS uses a certain symbol as a separator between the two selectors. These symbols are " " (one or more whitespaces), ">", "+" and "~" for descendant, child, adjacent or next sibling, and subsequent or general sibling combinators, respectively.

Consider the following HTML code.

```
<article>
  <h1>>My Tech Blog</h1>
  <p>Welcome to my Tech Blog. Here I share tips on
web development.</p>
  <div>
    <h2>CSS Combinators</h2>
    <p>In the previous post, I explained the
element selectors.
      In this post, today you will learn about the
CSS combinators.
    </p>
    <p>CSS combinators allow you to select
elements based on the
      relationship between selectors.</p>
  </div>
  <div>
    <h3>Types of CSS Combinators</h3>
    <p>This section lists the different types of
CSS combinators</p>
    <ul>
      <li>Descendant combinator (A B)</li>
```

```

    <li>Child combinator (A > B)</li>
    <li>Adjacent sibling (A + B)</li>
    <li>General sibling (A ~ B)</li>
  </ul>
</div>
</article>

```

The browser constructs the DOM tree while rendering the document, as shown in Figure [8-5](#).

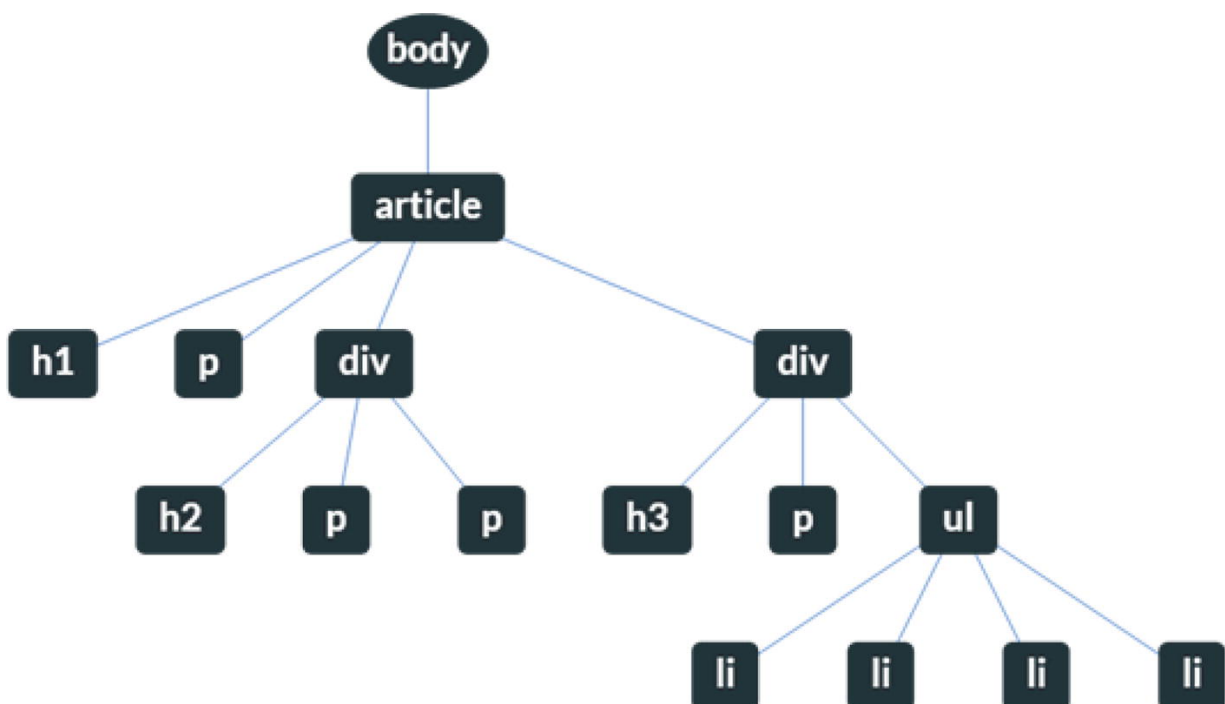


Figure 8-5 DOM tree

Let's find out how the different combinators select the HTML elements from the tree and apply the styling.

Descendant Combinator (" ")

To form a descendant combinator, you need to separate two selectors with one or more spaces.

```

sel1 sel2 {
/*CSS declarations*/

```

```
}
```

The browser selects nodes that are descendants of the first element. The following is an example.

```
article p {  
  /*CSS declarations*/  
}
```

This matches all <p> elements that are inside a <article> element. See Figure [8-6](#).

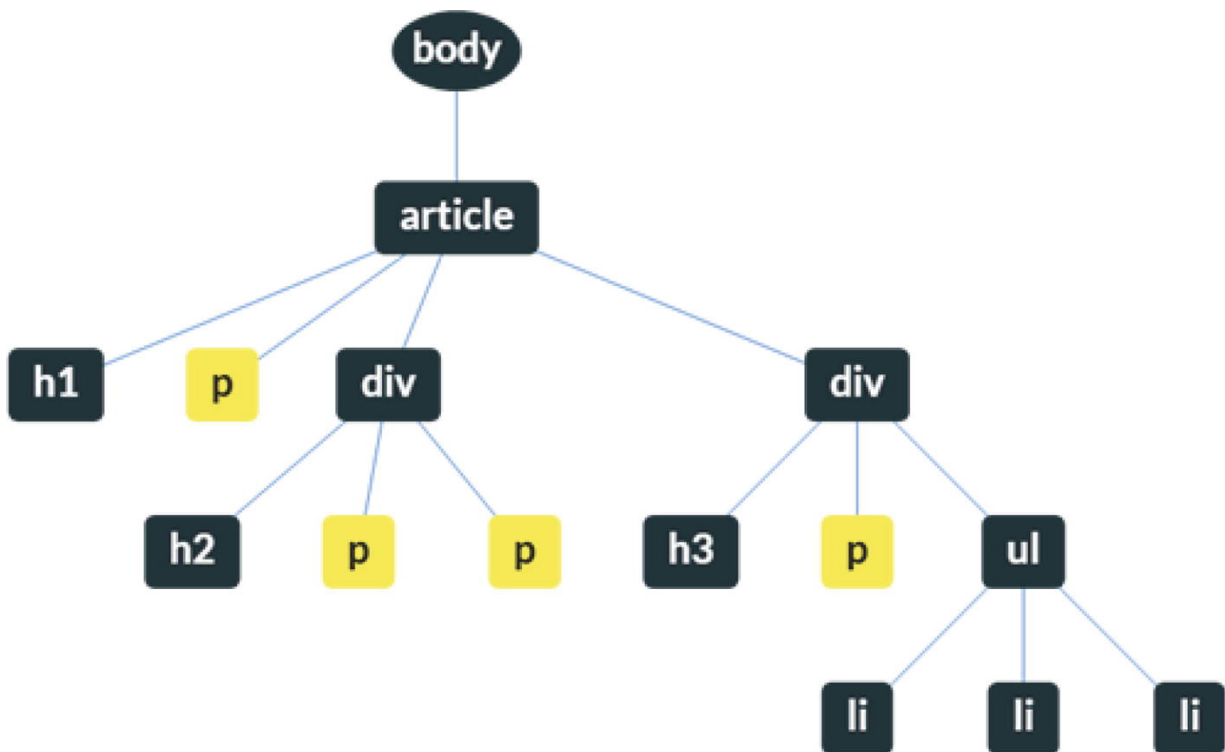


Figure 8-6 Descendant combinator

Child Combinator (>)

Two selectors separated by the greater than character (>) form a child combinator. Being more specific than the descendant combinator, the child combinator selects all the direct children nodes of the first element.

```
div > p {  
  /*CSS declarations*/  
}
```

This matches only the <p> elements that are direct children of a <div> element. See Figure [8-7](#).

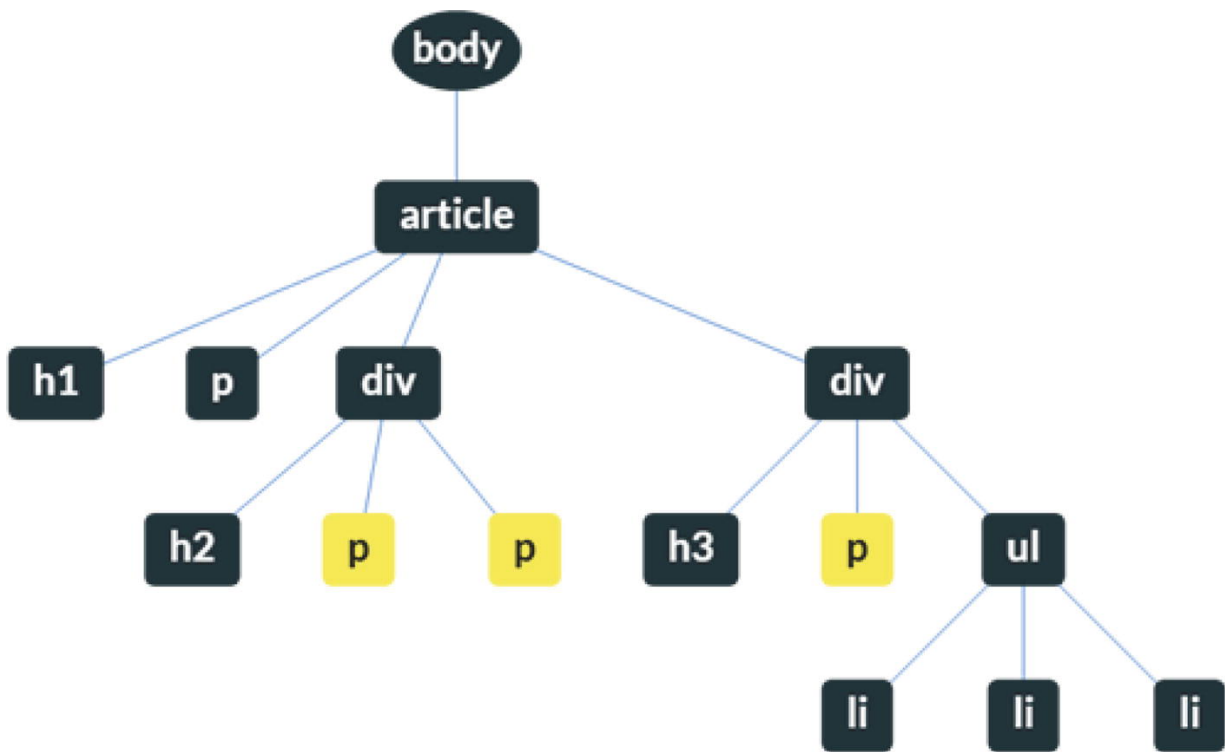


Figure 8-7 Child combinator

Subsequent Sibling Combinator (~)

The subsequent sibling (a.k.a. general sibling) combinator is denoted with a tilde (~) symbol. All elements matching the second selector will be selected if they are preceded by the first, provided both share the same parent.

```
sel1 ~ sel2 {  
  /*CSS declarations*/  
}
```


Here, sel1 represents the element after which any element represented by the second selector is to be selected.

```
h1 ~ div {  
  /*CSS declarations*/  
}
```

This matches all <div> elements that follow an h1, irrespective of whether immediately or not. See Figure [8-8](#).

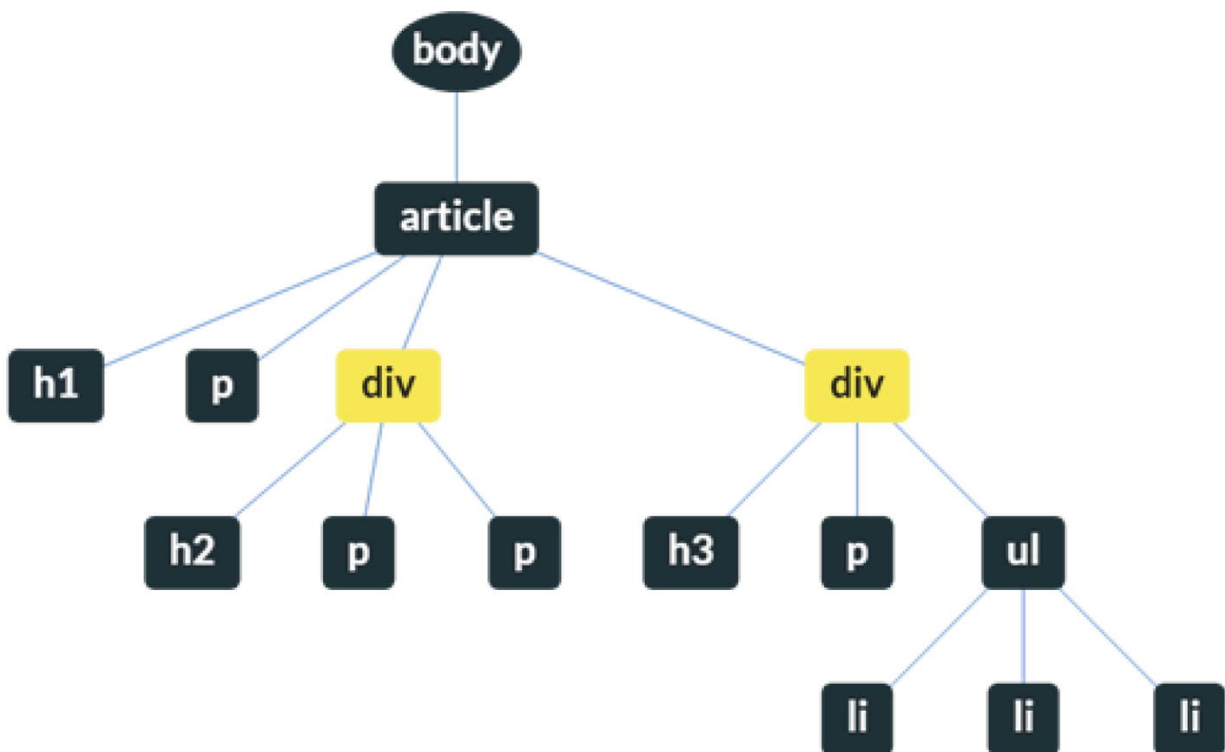


Figure 8-8 Subsequent siblings

Adjacent Sibling Combinator (+)

The next sibling combinator (or the adjacent sibling combinator) uses the plus symbol (+). It is used to select all the immediate siblings of an element in a web page.

```
sel1 + sel2 {  
  /*CSS declarations*/  
}
```

If an element represented by sel1 is immediately followed by the one represented by sel2, the latter is selected provided both the elements are children of the same parent. The following is an example.

```
h2 + p {  
  /*CSS declarations*/  
}
```

This rule matches only the single <p> element that immediately follows an <h2> element. See Figure [8-9](#).

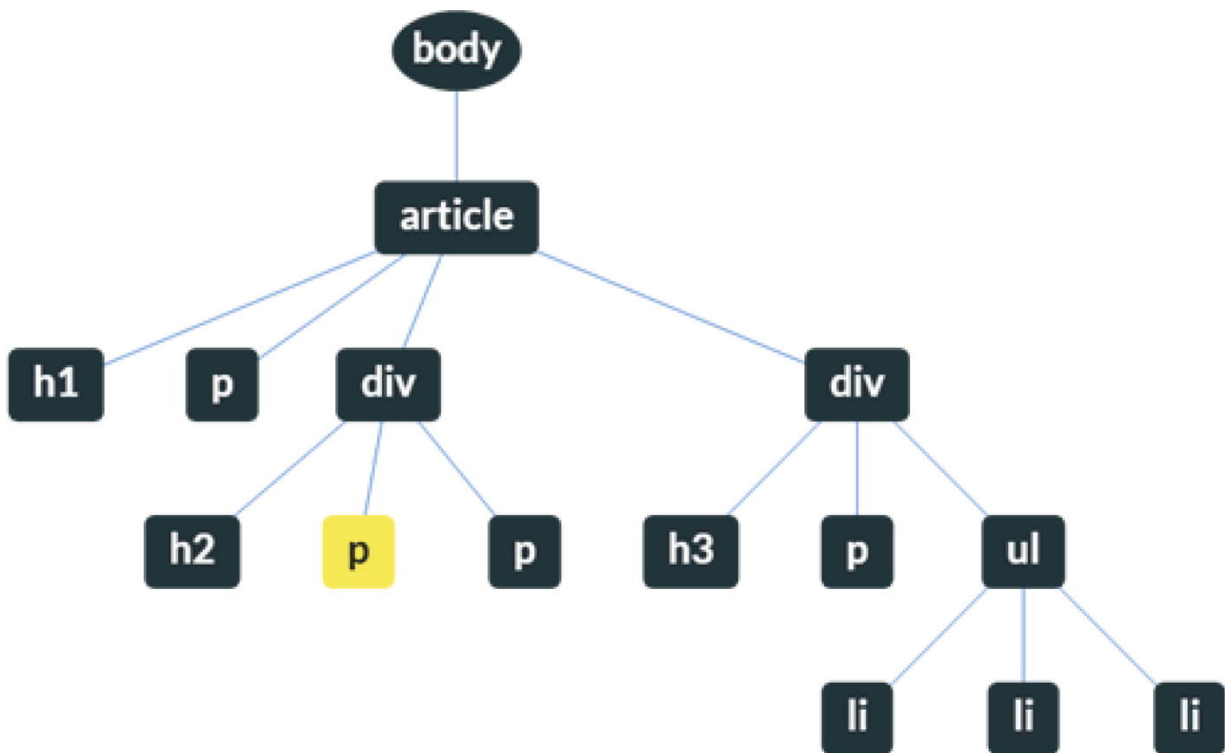


Figure 8-9 Adjacent siblings

Pseudo-Classes

CSS selectors help the browser to locate and stylize HTML elements based on tags, IDs, attributes, or classes. The capabilities of selectors are extended by adding pseudo-class keywords (about 60 of them). They allow styles to react to element states as determined by the browser, based on user interaction, and document structure. CSS

pseudo-classes can create basic interactivity and add dynamic visual behavior even without JavaScript.

A pseudo-class keyword is appended to a CSS selector. Here's the general syntax.

```
selector:pseudo-class {  
    /*CSS declarations*/  
}
```

One of the most commonly used pseudo-classes is `:hover`. The following CSS rule causes the button's color to turn blue whenever the mouse hovers over it.

```
button:hover {  
    color: blue;  
}
```

Note that pseudo-classes do not modify HTML document structure. They let CSS react to browser-detected conditions such as state changes, user interaction, or an element's position in the document tree.

CSS pseudo-classes can be put in the following broad categories.

User Interaction

These pseudo-classes respond to how a user interacts with elements. The following are some examples in this category.

:hover

The rule that uses this pseudo-class applies when the mouse pointer hovers over an element, and is very common for UI feedback.

```
button:hover {  
    background-color: blue;  
}
```

:active

This pseudo-class applies while an element is being clicked. For example, the following rule creates a “pressed button” feel.

```
button:active {  
    transform: scale(0.98);  
}
```

:focus

A CSS rule with this pseudo-class is triggered when an element receives keyboard or programmatic focus; for example, the following is used for accessibility and keyboard navigation.

```
input:focus {  
    border-color: green;  
}
```

Structural

This type of pseudo-class relates to the element’s position in the DOM tree. They are useful when you need to perform selection that otherwise is not possible with other simple selectors or combinators. The following are some frequently used structural pseudo-classes.

:first-child

A CSS rule that uses this pseudo-class selects an element that is the first child of its parent. For example, the following rule applies bold font-weight to the first element in an ordered/unordered list.

```
li:first-child {  
    font-weight: bold;  
}
```

:last-child

The `:last-child` pseudo-class targets the final child.

```
p:last-child {  
    margin-bottom: 0;  
}
```

:nth-child()

This is one of the most powerful pseudo-classes, used to select elements based on their position among their siblings. It uses a formula of the form $an+b$, or the keywords `odd` or `even` for making the selection.

The formula $an+b$ represents the cycle size (an integer). n is a counter that starts at 0 and increments for each element, while b is an offset value (an integer). The browser calculates $a * n + b$ and applies the styles to elements corresponding to the resulting positive index values.

The selector with the `:nth-child(even)` pseudo-class selects all even-numbered children, whereas `:nth-child(odd)` selects all odd-numbered children. To select every third child, define a rule with the `:nth-child(3n)` pseudo-class. Using `:nth-child(-n+3)` selects the first three children.

```
tr:nth-child(even) {  
    background: #f2f2f2;  
}
```

This rule sets the background of all even-numbered rows in an HTML table.

Form/Input State

This category of pseudo-classes is related to form elements. They enable selecting elements based on HTML attributes and the state of the input field before and after interaction. You have learned about the `readonly` attribute of form elements (text-type input, textarea, etc.). A CSS rule with a selector appended with the `:read-only` pseudo-class targets those elements for which the `readonly` attribute has been defined.

The following are other notable pseudo-classes in this category.

:checked

This pseudo-class helps CSS target those checkboxes or radio buttons which have been selected. The following rule sets their accent color.

```
input:checked {  
    accent-color: green;  
}
```

:disabled

A very useful pseudo-class that, when used with a selector, matches all disabled controls.

```
input:disabled {  
    background: #eee;  
}
```

Location

These pseudo-classes are specially defined for working with hyperlinks. You can apply styling to elements based on information related to the document's URL and navigation history, particularly links.

:link

Use this pseudo-class to apply styles (show the link in blue, as shown in the following rule) to links the user has not yet visited.

```
a:link {  
    color: blue;  
}
```

:visited

:visited lets you target the previously visited links.

```
a:visited {  
    color: purple;  
}
```

:any-link

A CSS rule involving this pseudo-class is useful for applying general link styles. It matches an element if it would match either `:link` or `:visited` pseudo-class.

```
a:any-link {  
    text-decoration: underline;  
}
```

Pseudo-Elements

In CSS, pseudo-elements are keywords added to a selector that enable you to style a specific part of the selected element. Pseudo-element keywords are appended to the selector after the double colon (`::`) symbol instead of a single colon as in pseudo-classes.

Here is an example of a pseudo-element.

```
p::first-line {  
    color: blue;  
    text-transform: uppercase;  
}
```

This rule targets the first line of every `<p>` element. A detailed discussion on the different pseudo-elements is beyond the scope of this book. To learn more about them, you can refer to the MDN Web Docs.

Inheritance

If you have any programming background, especially object-oriented languages such as Java, you have most probably come across the concept of inheritance. However, the notion of inheritance in CSS is a little different. The HTML elements in a document form a document tree (DOM), and there is an inherent parent-child relationship. For instance, all the visual elements in the document are the children of the

root element (i.e., the body). Similarly, elements are children of their top-level or element.

CSS defines hundreds of properties associated with different HTML elements. Inheritance in CSS is the mechanism by which certain CSS properties are passed down from a parent element to its descendants in the document tree. However, not all the properties of a parent element are inherited by its descendants. Generally, the properties related to text and typography are inheritable. For example, the parent properties like color, font-family, font-size, font-weight, line-height, text-align, and visibility are automatically inherited by its child elements.

Consider the following CSS rule.

```
div {  
  text-align: center;  
  font-weight: bold;  
}
```

Here's the HTML snippet.

```
<div>  
<p>Hello World inside div</p>  
</div>  
<p>Hello World outside div</p>
```

The properties defined in the <div> element are inherited by the <p> element inside, but not by the one outside it. Hence, the inner paragraph will follow center alignment and boldface.

Properties related to the box model, layout, visual backgrounds, and so forth are non-inheritable. Examples include border, margin, padding, width, height, background, display, and position.

So, if your <div> element is styled as follows, the border property will not be available to the inner paragraph since it is not inherited by default.


```
div {  
    border: solid;  
}
```

When a non-inheritable property isn't set, the browser uses its initial value instead of checking its parent. The `inherit` keyword forces a child element to take the parent's value of the corresponding property.

To force the paragraph inside the `<div>` element to acquire the border property of the latter, use the `inherit` keyword.

```
p {  
    border: inherit;  
}
```

Specificity

How does the browser react when multiple CSS rules target the same HTML element? Which of the style declarations will be ultimately applied? The specificity algorithm helps the browser to resolve this issue.

The CSS engine gives the highest priority to inline style. The ID selectors follow it. Class and attribute selectors as well as pseudo-classes are next in the priority list. Selectors with pseudo-elements have low priority, and universal selectors have no priority at all.

Consider the following stylesheet.

```
#p1 {  
    color: blue;  
    font-size: large;  
}  
  
.test {  
    color: green;  
    font-size: small;  
}
```

If your HTML code has a paragraph with its ID as well as the class attribute matching those in the preceding CSS rules.

```
<p id="p1" class="test">CSS Specificity rule</p>
```

You should get the text rendered in a large font and in blue color.

!important

To accord highest priority to a specific property and value, add the **!important** keyword to its declaration. This effectively overrides all the specificity considerations.

For example the CSS rule

```
p {  
  color: red !important;  
}
```

This displays all the `<p>` elements in the document in red, even if the stylesheet has other selectors (or inline styles) that target the `<p>` element to style differently.

Points to Remember

- CSS selectors are the patterns used by the browser to select HTML elements and style them with CSS rules.
- The rule with the universal selector applies to all elements in a document.
- You can use any HTML element as a selector, which is a common practice.
- ID selector targets an element with a specific ID.
- Using attribute selectors, you can target all the elements defined with a specific attribute.
- Class selectors are very powerful. The class attribute is a global attribute used to group elements logically and allows you to apply

common styles to various HTML elements.

- CSS allows you to define CSS rules with more than one selector. The combinator symbols combine two selectors.
 - Whitespace between the two forms a descendant combinator.
 - Two selectors separated by the > symbol form a child combinator.
 - The ~ symbol is defined to form a subsequent sibling combinator.
 - The adjacent sibling combinator is formed by putting the + symbol between the two.
 - CSS defines pseudo-class keywords to target elements based on their state, user interaction, and the DOM structure. These keywords are appended to any selector after the : symbol.
 - Most typographical CSS properties, like color and font-family, are inheritable, while layout and background properties are non-inheritable. To make a property explicitly inheritable, use the inherit keyword.
 - Specificity is the priority order in which the CSS engine applies competing selector rules. The inline style has the highest priority, whereas a universal selector has no priority.
 - Appending the CSS declaration with the !important keyword overrides all the rules of specificity.
-

Summary

The concept of CSS selectors was introduced in the previous chapter. In this chapter, different types of selectors such as ID, attribute, and class selectors have been explained with useful examples. You also learned how the CSS combinators help the browser to target specific elements in the DOM tree. Pseudo-classes are a powerful feature in CSS. You learned how they enhance interactivity and dynamic visual behavior of the web page. This chapter briefly touches upon the concepts of inheritance and specificity in CSS.

The next chapter covers CSS layouts in detail.

9. CSS Layouts

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

By now, you know that a web page has various HTML elements, and each of them is styled by defining appropriate CSS rules. This chapter addresses an important issue: how the various elements in a document are laid out to enhance its visual appeal, usability, and functionality. Different types of layouts help control the relative positioning and spacing of elements on the web page.

This chapter explains how the elements are arranged in the default flow layout and how their positioning can be controlled. It also discusses modern layout techniques such as flexbox and grid.

Box Model

Recall that a browser renders each HTML element as a rectangular box. In HTML, elements are either block-level or inline. Elements like `<div>`, `<p>`, and `<section>` are block-level elements. In contrast, `<img>`, `<input>`, and `<select>` are some examples of inline elements. Accordingly, the boxes that bound HTML elements are also of two types: block boxes and inline boxes.

Each HTML element is wrapped inside four concentric rectangular boxes. The innermost box is the one that actually holds its contents. A transparent region around the content is known as the padding. It separates the content box from the border box around it. The outermost box is also transparent and controls the space between the

adjacent elements: those above, below, to its left and right. See Figure [9-1](#).

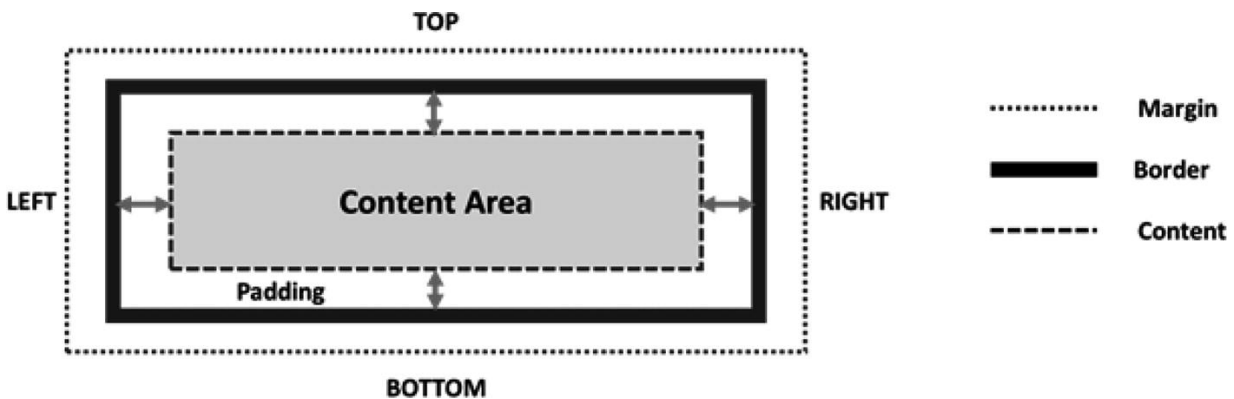


Figure 9-1 Box model

The margin CSS shorthand property decides the margin area on all four sides of an element. To set the margin separately on four sides, use `margin-top`, `margin-bottom`, `margin-left`, and `margin-right` properties. The border property defines the size, style, and color of the border around the element. Padding separates the content box from the border. The extent of the padding depends on the value of the padding shortcut property (or `padding-top`, `padding-bottom`, `padding-left`, and `padding-right` properties). Each of these properties obviously has a default value, which the CSS rules can override.

Flow Layout

By default, the browser starts placing all the HTML as they appear in the document, from left to right and from top to bottom. A block-level element always starts on a new line. It takes up the available width fully (subject to its width and height properties), adding some margin before and after the element. An inline element, on the other hand, does not start on a new line. It only occupies as much width as necessary.

Consider the following HTML script.

```
<body>
  <h2> Flow Layout </h2>
```

```
<h3> Web Development Essentials </h3>
<div>
<h4>ClientSide technologies:</h4>
<span> HTML </span>
<span> CSS </span>
<span> JavaScript </span>
</div>

<div>
<h4>Server side technologies:</h4>
<span> Node JS</span>
<span> Express </span>
<span> SQLite </span>
</div>

<b>HTML
Logo</b>

</body>
```

To understand how the flow layout works, define a universal selector that applies border, margin, and padding to all the elements.

```
* {
  border: 2px solid;
  margin: 5px;
  padding: 5px;
}
```

Figure [9-2](#) shows the browser output.

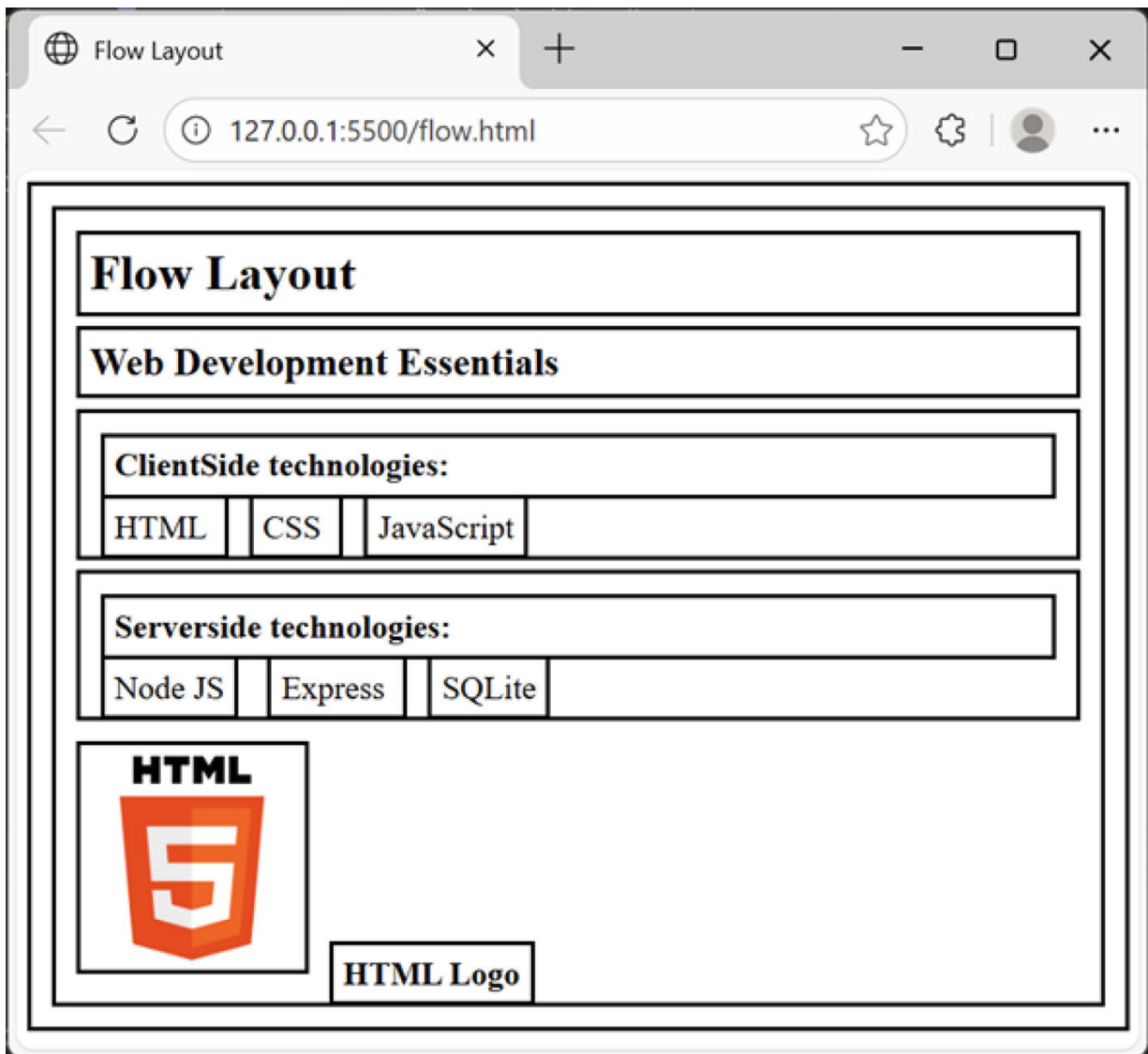


Figure 9-2 Flow layout

It is evident that the block boxes occupy the available width and the inline boxes are placed adjacent to the others. However, you can override the default sequential flow of an HTML document by using the float and position properties appropriately.

The float Property

The float CSS property does exactly as it suggests—it causes the element to float to the left or right side of its container, thus allowing

text and inline elements to wrap around it. The element is removed from the normal flow of the page, although it remains a part of the flow.

The float property is most commonly used to wrap textual content around images or similar objects, as is normally seen in newspaper or magazine layouts. The following are its possible values.

left: The element floats on the left side of the container.

right: The element floats on the right side of the container.

none: The element does not float on either side.

inherit: The element inherits the float value of its parent.

If set to inline-start or inline-end, the element floats on the start side or the end side of its containing box. The property is available for all elements. However, it has no effect if an element's display property is set to none.

To understand how the float property works, consider a document with an image (an inline element), immediately followed by a div element containing text. To make the image block float to the left of the text, define a CSS rule for the img element as follows.

```
img {  
  float: left;  
  margin: 10px;  
  width: 100px;  
  height: 100px;  
}
```

When opened in the browser, the page shows that the logo floats to the left. The adjacent text is wrapped around it. See Figure [9-3](#).

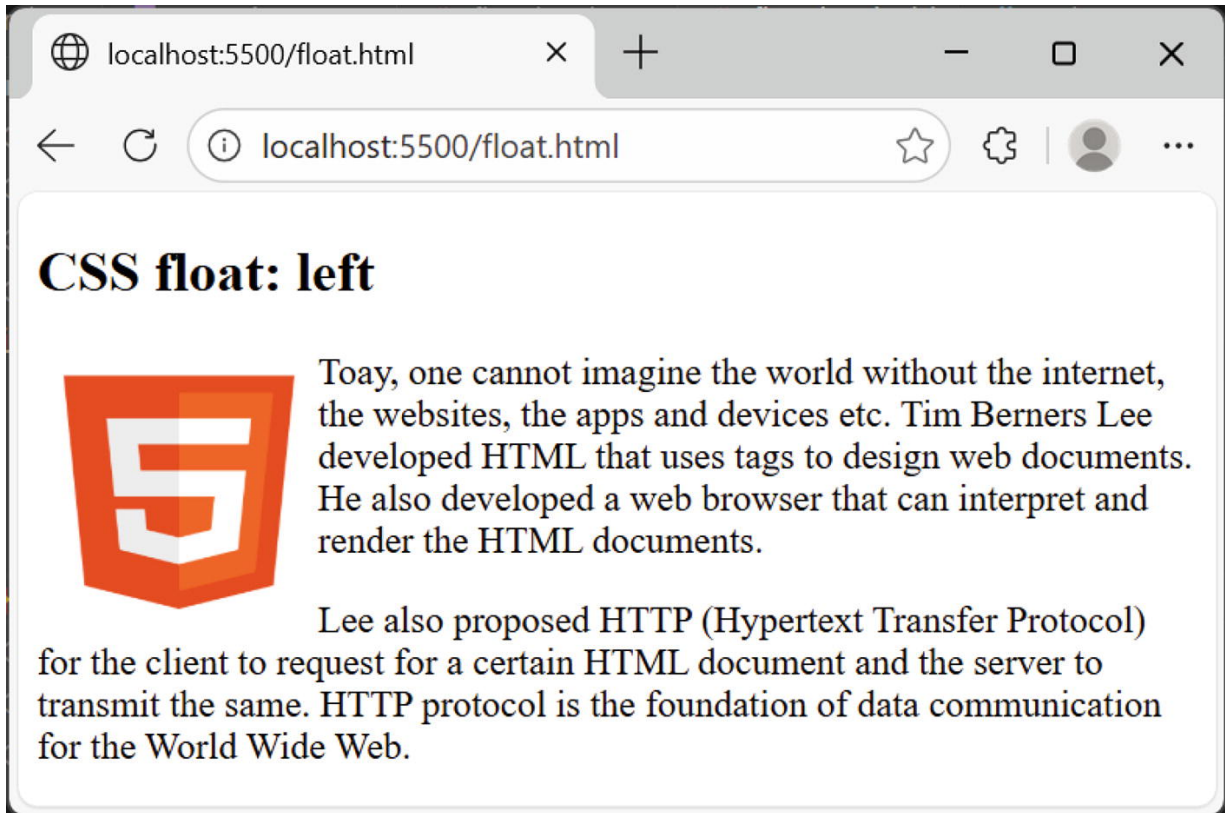


Figure 9-3 The left float property

Just change the float property to right, and it results in the other way round. See Figure [9-4](#).

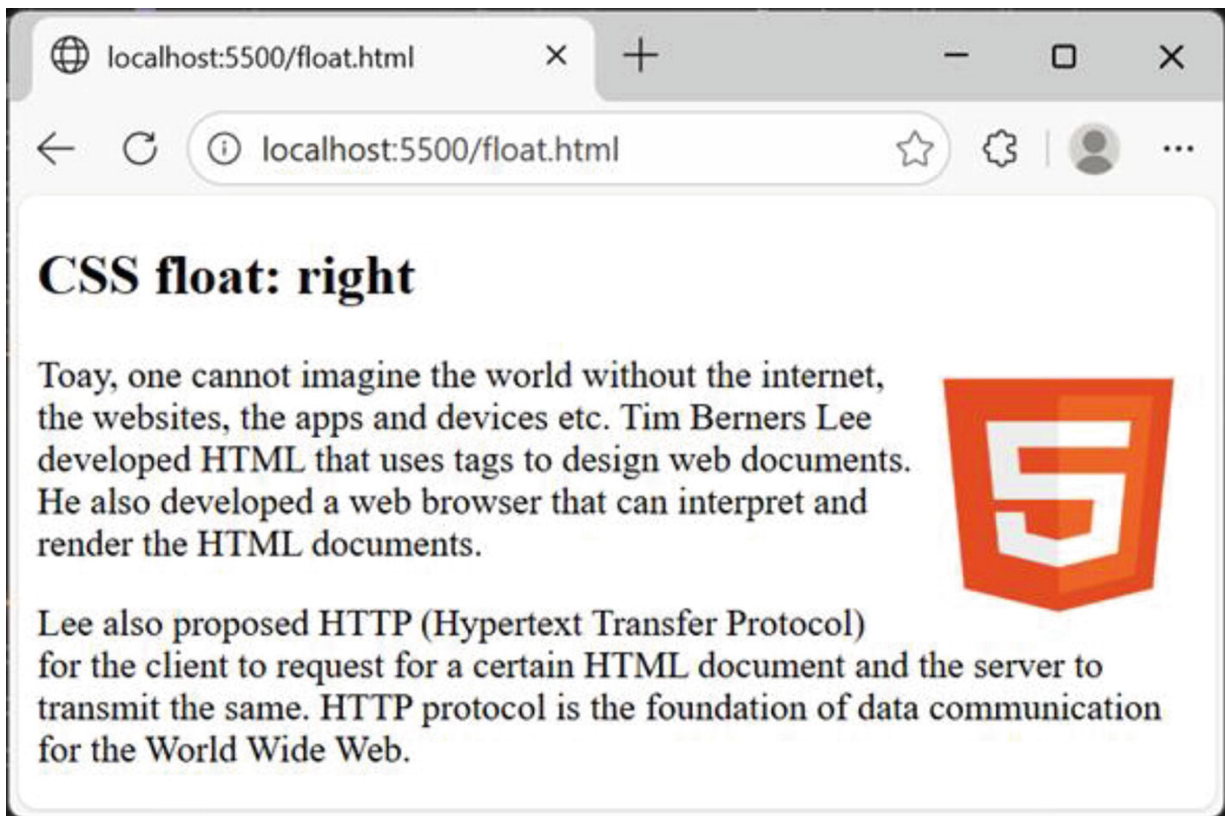


Figure 9-4 The right float property

The position Property

Compared to the float property, which is used for flowing content (like text wrapping around an image), the position property offers more precise control over the placement of an element, taking elements completely out of the flow.

The position property takes one of the predefined values: static, relative, absolute, fixed, and sticky. The actual positioning of an element is decided by the top, right, bottom, and left physical properties.

static

This is the default value; hence, the element is positioned according to the normal flow of the document. The subsequent top, right, bottom, and left properties have no effect.

relative

Although the element is placed according to the normal flow of the document, its position is offset relative to itself based on the values of top, right, bottom, and left without affecting the position of other elements. The top and bottom values specify the vertical offset, and the horizontal offset depends on the values of the left and right properties.

Here is an example. The following CSS rule defines as a relative positional element alongside a <div> container with a few paragraphs.

```
img {  
    position: relative;  
    top: 40px;  
    left: 40px;  
    width: 100px;  
    height: 100px;  
}
```

The logo follows the preceding relative positioning, as shown in Figure [9-5](#).

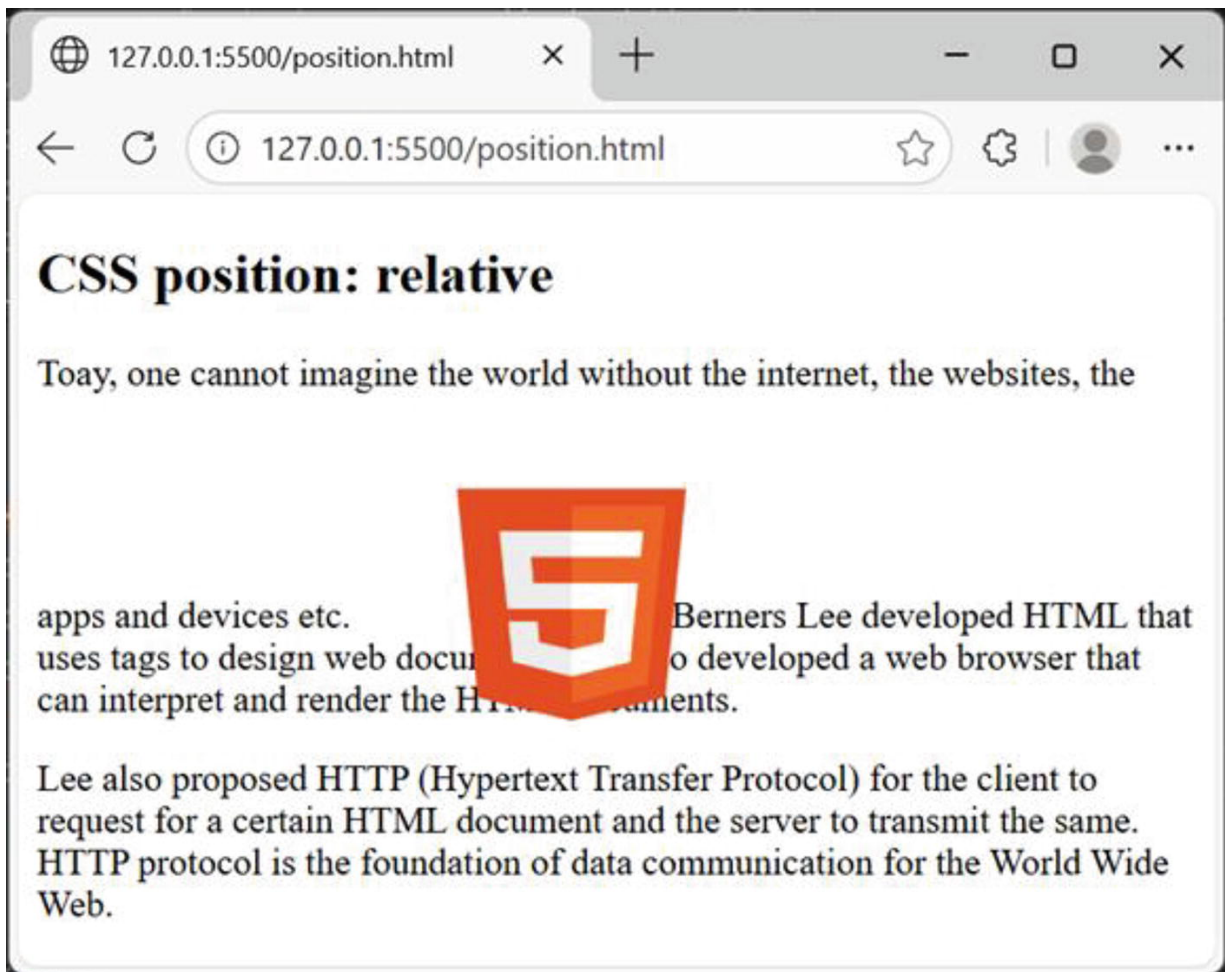


Figure 9-5 The position property

absolute

If the position property is set to absolute, its computed position value is absolute or fixed. It is removed from the normal document flow, with no space created for the element in the page layout. It is positioned relative to its closest ancestor or to the initial containing block. The top, right, bottom, and left properties specify offsets from the edges of the element's containing block. If the element has margins, they are added to the offset.

Let's change the preceding CSS rule to make the `<img>` element absolute.

```
img {  
    position: absolute;
```

```
top: 0px;  
left: 300px;  
width: 75px;  
height: 75px;  
}
```

Since the top value is 0, the logo is placed very close to the top of the page, as seen in Figure [9-6](#).

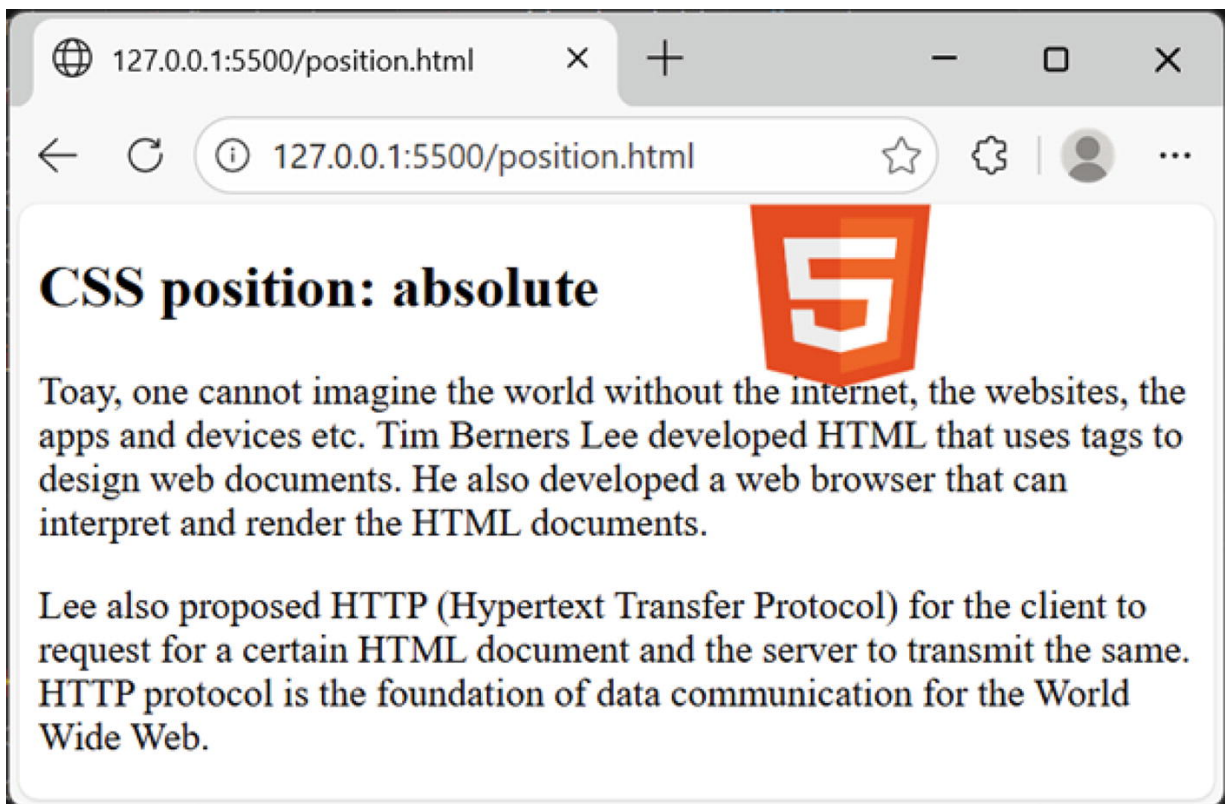


Figure 9-6 absolute positioning

fixed

The main difference between fixed and absolute positioning is their reference point. While fixed positioning is relative to the viewport (browser window), elements having their position property set to absolute are positioned relative to their nearest positioned ancestor.

sticky

An element whose position property is set to sticky actually works as a relatively positioned element until the scroll threshold is reached. At the threshold point, it “sticks” to the viewport.

To make an element sticky, you must define at least one threshold property (like `top: 0`). Additionally, you must also define the z-index property.

The z-order property specifies the stack order of positioned elements, controlling which element should be placed on top of other elements.

```
.header {  
  position: sticky;  
  top: 0;  
  z-index: 1000;  
}
```

Common use cases of sticky positioning include navigation bars sticking to the top, sidebars that remain visible alongside a scrolling main content pane, and table headers.

The following HTML code has a container with a header followed by lengthy text.

```
<div class="container">  
  <div class="header">  
    <h2>CSS position: sticky</h2>  
    <h3>Scroll down to see the header stay at the  
top.</h3>  
  </div>  
  <div class="content">  
  
    <!-- Add more text here to enable scrolling --  
>  
  
    <p>Lorem ipsum dolor sit amet, consectetur  
adipiscing elit. Aenean commodo ligula eget dolor.  
Aenean massa.. . .</p>
```

```
</div>  
</div>  
</div>
```

To ensure scrolling, the container must have enough height. The CSS rule for the header class is made sticky, with `top:0` acting as the threshold value.

```
.container {  
    height: 1000px;  
    background-color: #f4f4f4;  
}  
  
/* The sticky element */  
.header {  
    position: sticky;  
    top: 0; /* Threshold: sticks to the top of the  
viewport */  
    background-color: #007bff;  
    color: white;  
    z-index: 1000;  
}
```

Make sure the content has sufficient text to experience the sticky behavior of the header as the user scrolls past the threshold. See [Figure 9-7](#).

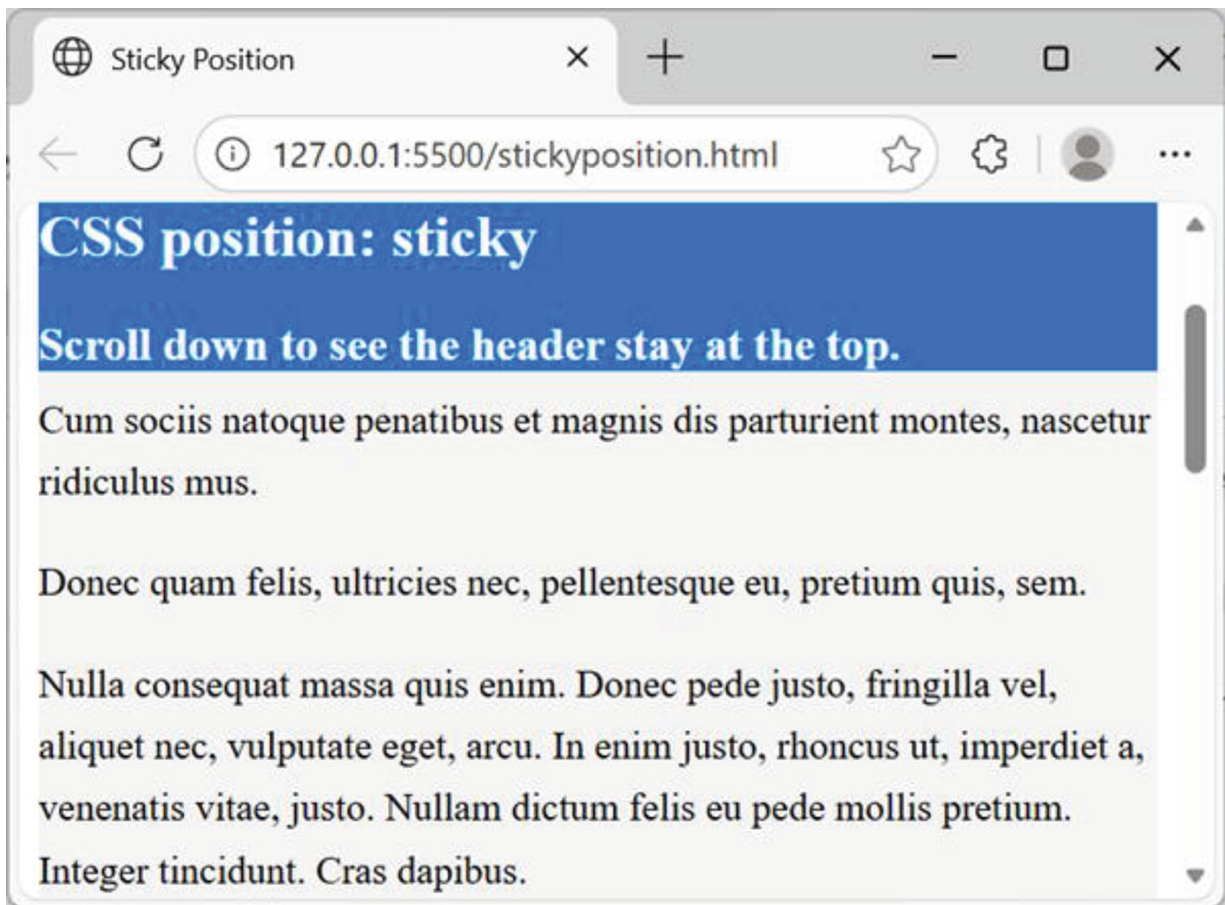


Figure 9-7 The sticky position property

The display Property

An HTML element is either a block-level element or an inline element. This element type is reflected in the default value of its display property, which is either block or inline, respectively. A block-level element always starts at a new line and occupies the entire width available in the container. In contrast, the inline element uses the width in the same line, only as much as is necessary.

However, this default behavior can be overridden. To render a block-level element as an inline element, change its display property to inline (and vice versa).

The `<li>` element is a block-level element. Hence, the elements in an ordered or unordered list appear one below the other by default. To

display the list component horizontally (as you would normally find in a navbar), set the display property of the element to inline.

```
<html>
<head>
<style>

li {
  display: inline;
}
</style>
</head>
<body>

<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">Login</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</nav>

</body>
</html>
```

This produces a navbar at the top of the page, with the hyperlinked anchors placed horizontally.

The other acceptable values of the display property are flex and grid, used respectively when you implement the flexbox and grid layouts.

Flexbox

Modern CSS introduced flexbox as a one-dimensional layout model. You can control the distribution of spacing between the items included in a

container whose display property is set to flex.

The flexbox layout handles one dimension at a time—either row or column. The CSS grid layout, on the other hand, is a two-dimensional model, which deals with columns and rows together.

When the display property of a container element (such as `div`) is set to flex (or inline-flex), it is turned into a flex box. It contains one or more flex items. In other words, the direct children of the flex container become flex items.

Next, you need to define the flex-direction property of the container. By default, it is set to row, which means the flex-items will be arranged horizontally. To ensure that the items are stacked vertically, set the flex-direction property to column.

The flex-direction property has four possible values.

- row
- row-reverse
- column
- column-reverse

The flex-wrap property of the container decides whether the flex items should wrap. By default, it is nowrap, but you can set it to wrap or wrap-reverse.

Similarly, the alignment of the flex items depends on the justify-content property. The following are possible values.

- center
- flex-start (default)
- flex-end
- space-around
- space-between
- space-evenly

The align-items property aligns the flex items vertically. Its default value is normal. If set to flex-start, the items are aligned at the top of the container. Set it to flex-end so the items align at the bottom.

To understand how the flex-direction property controls the placement of flex items, consider the following <div> elements with three paragraphs.

```
<div class="rowcontainer">
  <p class="flexitem">HTML</p>
  <p class="flexitem">CSS</p>
  <p class="flexitem">JavaScript</p>
</div>
```

The CSS rule for the rowcontainer class turns this div into a flex box with the direction of its flex elements set to row.

```
.rowcontainer {
  display: flex;
  flex-direction: row;
  background-color: darkgrey;
}
```

Let's render each paragraph with a white background against the dark gray background of the container.

```
.flexitem {
  background-color: white;
  width: 100px;
  margin: 10px;
  padding: 10px;
  text-align: center;
```

The result can be seen in the browser screenshot shown in Figure [9-8](#).

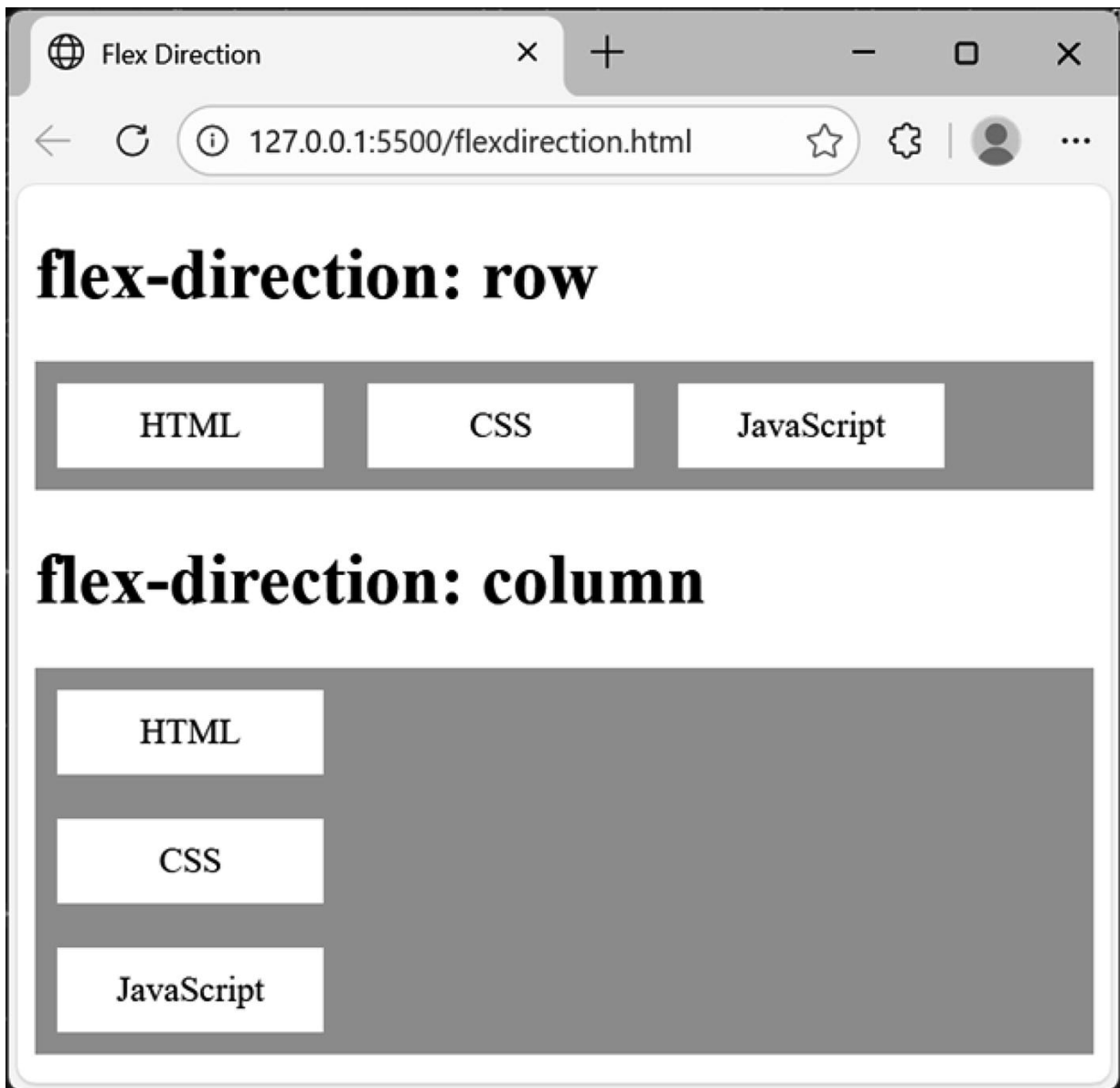


Figure 9-8 A flexbox

Now, change the flex-direction property to column, keeping the others the same.

```
.colcontainer {  
  display: flex;  
  flex-direction: column;  
  background-color: darkgrey;  
}
```

Make sure that the paragraph elements are now placed inside a vertical flexbox.

```
<div class="colcontainer">
  <p class="flexitem">HTML</p>
  <p class="flexitem">CSS</p>
  <p class="flexitem">JavaScript</p>
</div>
```

The browser shows that the flex items are stacked vertically. See [Figure 9-8](#).

Let's build a simple but responsive user profile form to demonstrate more comprehensive use of flexbox.

The form element itself is defined as a flexbox, with its items to be arranged vertically.

```
form {
  display: flex;
  flex-direction: column;
  gap: 15px;
}
```

Each flex item is itself a flexbox with the flex-direction property set to row. It is designed to hold a form control and its corresponding label.

```
.form-row {
  display: flex;
  flex-direction: row;
  align-items: center;
  gap: 10px;
}
```

The form has a few text-type inputs and a textarea control. To let them occupy the entire width following the label, set their flex property to 1.

```
.form-row input,
.form-row textarea {
  flex: 1;
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
}
```

Finally, the bottom row (note that it is a flex-item of the form flexbox) has two buttons: submit and cancel. Its justify-content property is set to flex-end to ensure they are placed right-aligned.

```
.form-actions {
  display: flex;
  justify-content: flex-end;
  gap: 10px;
}
```

Here is the HTML code for the user profile form.

```
<form>

  <div class="form-row">
    <label>Full Name</label>
    <input type="text" placeholder="Enter your
name">
  </div>

  <div class="form-row">
    <label>Email</label>
    <input type="email" placeholder="Your
email">
  </div>

  <div class="form-row">
```

```
        <label>Phone</label>
        <input type="tel" placeholder="Enter phone
number">
    </div>

    <div class="form-row">
        <label>About you</label>
        <textarea placeholder="Describe yourself">
</textarea>
    </div>

    <div class="form-actions">
        <button type="button">Cancel</button>
        <button type="submit">Save</button>
    </div>

</form>
```

You should display the user profile form as shown in Figure [9-9](#).

The image shows a web browser window with a single tab titled 'Flexbox example'. The address bar displays '127.0.0.1:5500/flexform.html'. The main content area has a large heading 'Flexbox Example'. Below it is a rounded rectangle containing a form titled 'User Profile'. The form has four labels on the left: 'Full Name', 'Email', 'Phone', and 'About you'. Each label is paired with a text input field on the right. The input fields contain placeholder text: 'Enter your name', 'Your email', 'Enter phone number', and 'Describe yourself'. At the bottom right of the form are two buttons: 'Cancel' and 'Save'.

Figure 9-9 HTML form with flexbox

Grid Layout

The grid layout turns the container into a grid, marking it with the required number of rows and columns. A container element such as

`<div>` with its `display` property set to `grid` or `inline-grid` acts as a grid container. You need to use the `grid-template-columns` property to specify the number of columns you want the container to have. The direct children of the grid container are treated as grid elements, and they are placed in columns from left to right and rows from top to bottom.

The following CSS rule uses the `grid-template-columns` property to convert the container into a three-column grid.

```
.container {  
  display: grid;  
  grid-template-columns: auto auto auto;  
}
```

You can set the column width in pixels or the `fr` unit. Similarly, the `grid-template-rows` property specifies the height of the rows in a grid layout. The space between any two adjacent lines on the grid is called a *grid track*.

To understand how the grid layout works, consider the following HTML snippet with a `div` containing some `<p>` elements.

```
<div class="threecol">  
  <p class="griditem">HTML</p>  
  <p class="griditem">CSS</p>  
  <p class="griditem">JavaScript</p>  
  <p class="griditem">NodeJS</p>  
  <p class="griditem">ExpressJS</p>  
</div>
```

First, turn the `<div>` element into a grid container with three columns, with widths set to `auto`.

```
.threecol {  
  display: grid;  
  grid-template-columns: auto auto auto;
```

```
background-color: darkgrey;
padding: 5px;
gap: 5px;
}
```

The direct children of the grid container are the grid items (the <p> elements in this case) and are placed according to the grid layout algorithm. All the paragraphs are styled with the following CSS rule.

```
.griditem {
background-color: white;
width: 100px;
margin: 10px;
padding: 10px;
text-align: center;
}
```

The grid layout algorithm automatically calculates row heights. However, you can explicitly mention it for each row. The following CSS rule applies a different row height to each row.

```
.threerow {
display: grid;
grid-template-columns: auto auto;
grid-template-rows: 50px 75px auto;
background-color: darkgrey;
padding: 5px;
gap: 5px;
}
```

Make sure that the paragraph elements are now placed in this new grid container.

```
<div class="threerow">
  <p class="griditem">HTML</p>
```

```
<p class="griditem">CSS</p>
<p class="griditem">JavaScript</p>
<p class="griditem">NodeJS</p>
<p class="griditem">ExpressJS</p>
</div>
```

The browser screenshot in Figure [9-10](#) shows the result of both grid layout examples.

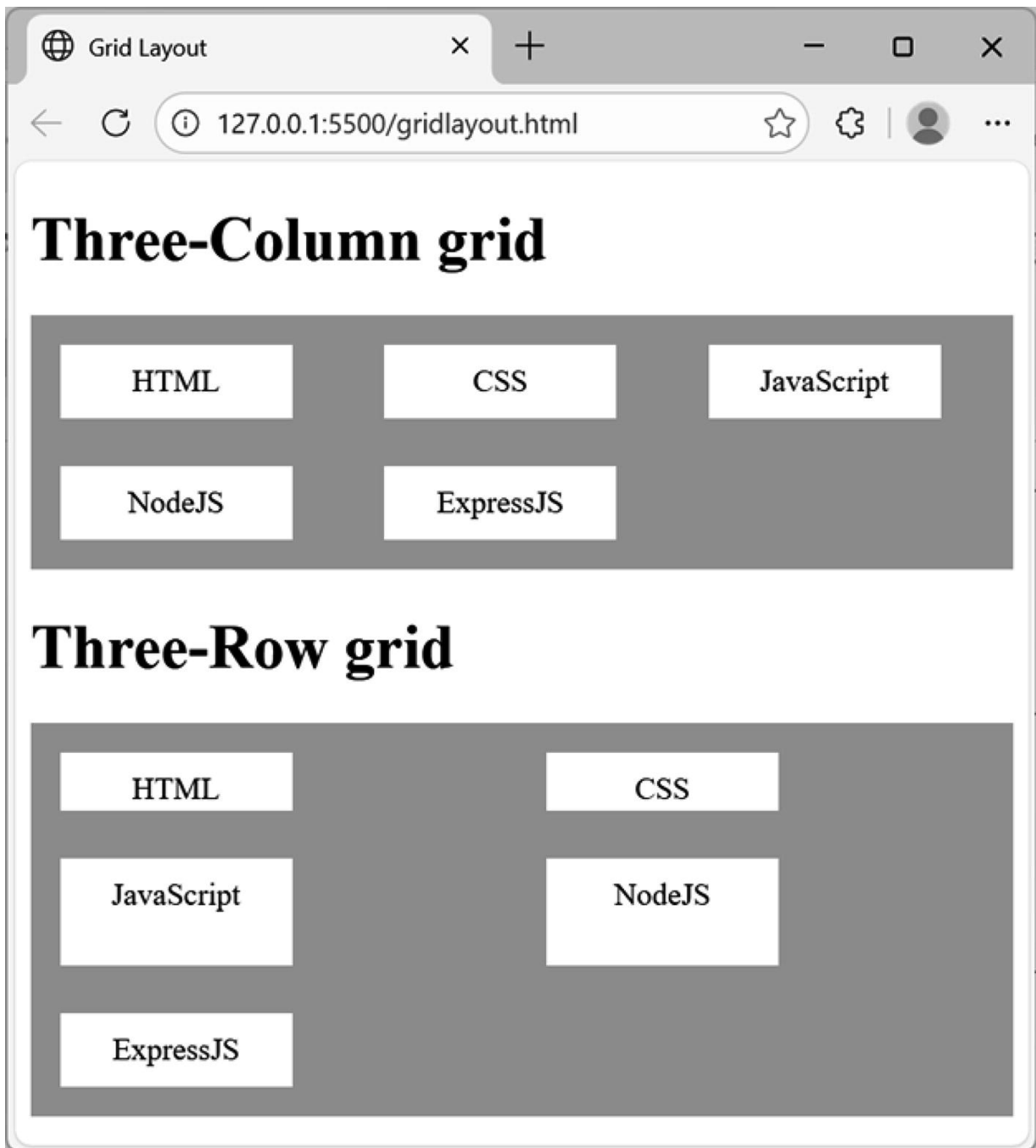


Figure 9-10 Grid layout

Spanning

You used the `colspan` and `rowspan` properties of the HTML `<td>` element with which a table cell may be spanned across multiple adjacent columns or rows. The CSS grid layout also allows you to create a grid cell occupying multiple columns or rows.

The `grid-column-start` property sets the column-line on which the item will start. Along with it, define the `grid-column-end` property to tell the CSS engine how many columns an item will grow horizontally.

```
.colspanitem {  
    background-color: white;  
    grid-column-start: 1;  
    grid-column-end: span 3;  
    width: auto;  
    margin: 10px;  
    padding: 10px;  
    text-align: center;  
}
```

Similarly, the `grid-row-start` and `grid-row-end` properties control how a grid cell spans rows.

```
.rowspanitem {  
    background-color: white;  
    grid-row-start: 1;  
    grid-row-end: span 3;  
    height: auto;  
    margin: 10px;  
    padding: 10px;  
    text-align: center;  
}
```

Use these CSS rules in the previous examples to show the column and row spanning as shown in Figure [9-11](#).

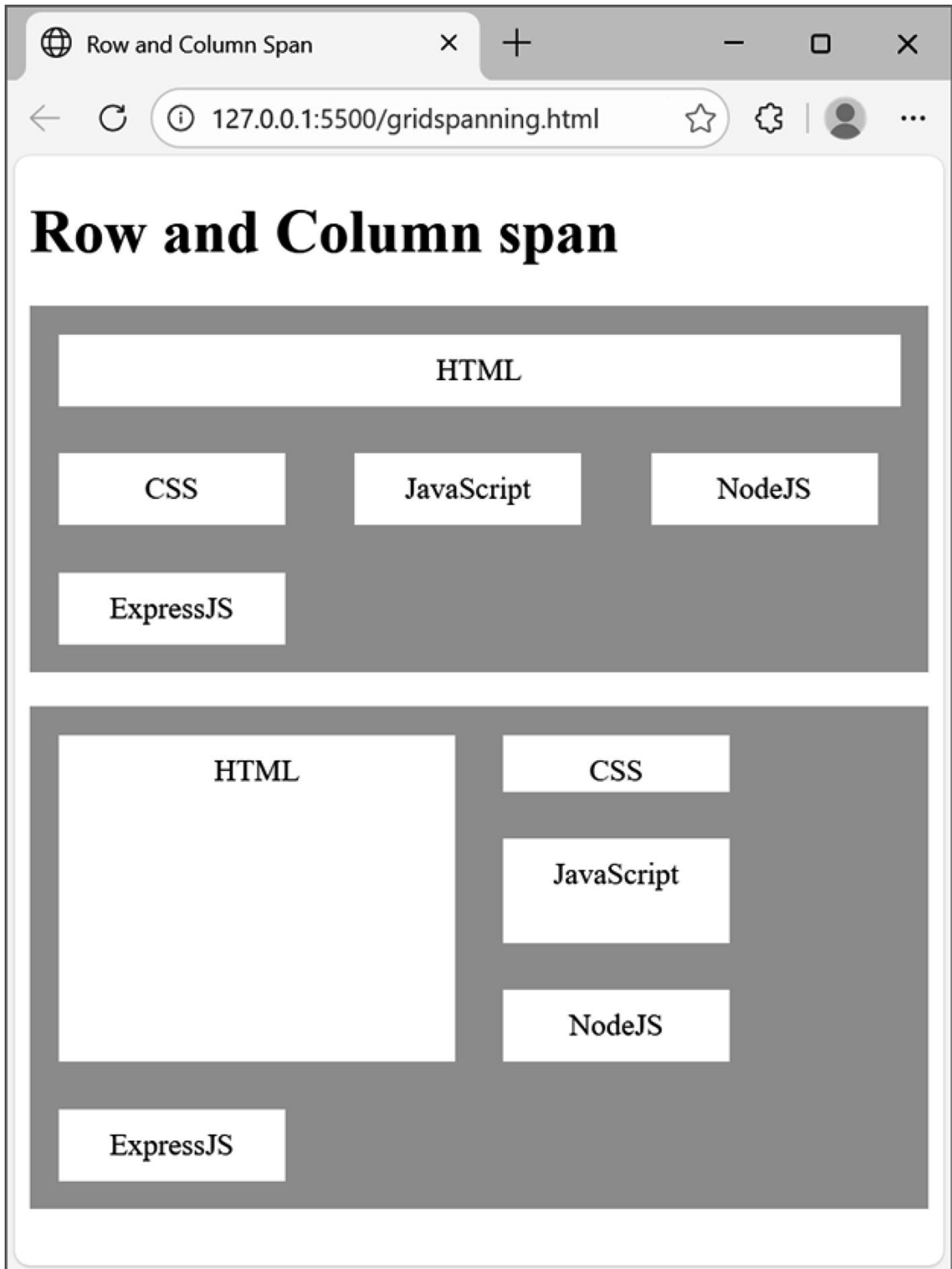


Figure 9-11 Spanning in grid layout

The `grid-column` property works as the shortcut for `grid-column-start` and `grid-column-end` properties.

```
.colspanitem {  
  grid-column: 1 / span 3;  
}
```

Similarly, the `grid-row` property is the shortcut for `grid-row-start` and `grid-row-end` properties.

```
.rowspanitem {  
  grid-row: 1 / span 3;  
}
```

The grid layout gives you absolute control over the web page design with various other properties, which you can explore in the official MDN documentation.

Note In the previous section, you used flexbox to design a user profile form. Try to use the grid layout to build its exact replica. You can find the solution in this book's code repository ([gridform.html](#)).

Responsive Web Design

Modern trends in web development focus on responsive design. This design approach ensures that web pages automatically adapt to changes in screen size and rearrange content, so you don't have to build multiple fixed layouts designed for different screen widths, such as desktop monitors, tablets, and mobile phones.

The following are some of the important considerations in the responsive design approach.

Viewport

The user agent's visible area of a web page is called the *viewport*. Obviously, it changes from device to device (desktop monitor, tablet, or

mobile). You must define a viewport meta tag in the head section of the web page.

```
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
```

This sets the width of the web page to the screen-width of the device. The initial-scale is set to 1.0, which means that there is no zoom on the page load. This setting ensures better readability and helps media queries run correctly.

Fluid Layouts

To make the page responsive, you must set the main container element as a flexbox or a grid. This is done by setting its display property to flex or grid.

Another point to note is not to set dimensions (font size, width, height, etc.) in fixed units such as pixels, but always use relative units: %, vw, vh, em, and rem.

Media Queries

CSS media queries play a crucial part in creating responsive web pages. A media query applies the specific CSS style only when the browser and device settings match a given rule. This helps you create different layouts depending on factors like the viewport size and other characteristics such as resolution and orientation.

A media query starts with @media followed by media type, a condition (media feature), and a block of CSS rules.

```
@media media-type and (media-feature-rule) {
  /* CSS rules to apply when the condition is true
  */
}
```

For example, the following media query sets the background to light blue only for desktop-sized screens (at least 768px wide).


```
/* Tablet and larger */
@media screen and (min-width: 768px) {
  body {
    background-color: lightblue;
  }
}
```

Media queries can be classified in the following categories.

screen: To account for various computer screens, tablets, and smartphones.

print: For printers and print preview mode.

speech: Used for screen readers.

all: Suitable for all devices.

Points to Remember

- Browsers follow a model of four concentric rectangular boxes to render any HTML element. A padding area, border, and margins surround the actual content.
- By default, the browser places HTML elements in a normal flow from left to right and top to bottom order.
- The CSS float property removes an element from the normal flow and allows content to wrap around it.
- The position property offers more precise control, either by positioning an element by specifying absolute or relative parameters. A sticky-positioned element remains stuck to the viewport when it reaches a scroll threshold.
- The display property determines whether an element is treated as a block-level or inline element.
- Flexbox is a one-dimensional layout. The flexbox model requires the container's display property to be set to flex. All the direct children of a flex container are its flex items. Their position depends on the flex-direction, which is either row or column.

- By setting the display property to grid, a container element acts as a two-dimensional layout. Its grid-item children are placed by specifying row and column positions. The column width and row height can be set as required. You can also span an element across columns or rows with the grid-column and grid-row properties.
 - The modern mobile-first approach in web design uses media queries in defining CSS rules. With media queries, the web page automatically adapts to different screen dimensions of user agents and rearranges the element positioning accordingly.
-

Summary

This chapter starts by describing the box model and the default flow layout used by browsers to place the HTML elements in a web page. The float and position properties that override the normal flow are explained with examples. You also learned about the flexbox and grid layouts and how to manage the complex positioning of HTML elements in modern web design practices.

The last three chapters covered CSS, one of three important front-end web technologies. The next chapter dives into another one: JavaScript.

Knowledge Check: Part III

1. **What does the “C” in CSS stand for?**
 - A. Creative
 - B. Cascading
 - C. Computer
 - D. Colorful

2. **Which of the following is the correct syntax for a CSS comment?**
 - A. `// this is a comment`
 - B. `# this is a comment`
 - C. `/* this is a comment */`
 - D. `' this is a comment`

3. **Which property is used to inherit all properties from a parent element in CSS?**
 - A. `all: inherit;`
 - B. `inherit-all: true;`
 - C. `apply: parent;`
 - D. `set: default;`

4. **How do you select an element with the id "header" in a CSS file?**
- A. `.header`
 - B. `header`
 - C. `#header`
 - D. `*header`
5. **Which selector targets all <p> elements that are direct children of a <div>?**
- A. `div p`
 - B. `div > p`
 - C. `div + p`
 - D. `div ~ p`
6. **What is the correct way to apply a style to multiple selectors (e.g., h1 and p) at once?**
- A. `h1 p { ... }`
 - B. `h1; p { ... }`
 - C. `h1, p { ... }`
 - D. `h1 + p { ... }`
7. **Which CSS selector has the highest specificity?**

- A. p (Type selector)
- B. .example (Class selector)
- C. #example (ID selector)
- D. div p (Combined type selectors)

8. **What does the selector a:hover represent?**

- A. A link that has already been clicked.
- B. A link that is currently being clicked.
- C. A link when the user's mouse pointer is over it.
- D. A link that has not been visited yet.

9. **Where is the most appropriate place to put an external CSS link?**

- A. At the very end of the <body>
- B. Inside the <title> tag
- C. Inside the <head> section
- D. Inside the <footer> section

10. **Which attribute is used to define inline styles directly on an HTML tag?**

- A. class

- B. style
- C. css
- D. font

11. **What is the purpose of the !important rule in CSS?**

- A. To make the code easier to read.
- B. To override all previous styling rules, regardless of specificity.
- C. To highlight code for other developers.
- D. To ensure the style only works on mobile devices.

12. **Which selector would target an <input> element only when it has focus?**

- A. input:active
- B. input:visited
- C. input:focus
- D. input:target

13. **Which property is used to change the background color of an element?**

- A. color
- B. bgcolor
- C. background-color

background-color

- D. surface-color

14.

How do you make the text bold in CSS?

- A. font-weight: bold;
- B. text-decoration: bold;
- C. font-style: bold;
- D. font-face: bold;

15.

Which property controls the size of text?

- A. text-style
- B. font-size
- C. text-size
- D. font-height

16. **Which value for the text-align property centers text within its container?**

- A. middle
- B. center
- C. justify
- D. inside

17. **What is the correct CSS to remove the underline from all hyperlinks?**
- A. `a { text-decoration: none; }`
 - B. `a { underline: none; }`
 - C. `a { text-style: no-underline; }`
 - D. `a { decoration: no-bottom-line; }`
18. **Which property is used to change the font of an element?**
- A. `font-style`
 - B. `font-weight`
 - C. `font-family`
 - D. `font-type`
19. **In the CSS box model, which layer is between the padding and the margin?**
- A. Content
 - B. Border
 - C. Outline
 - D. Ghost layer
20. **Which property is used to create space inside an element, between the content and the border?**

between the content and the border?

- A. margin
- B. spacing
- C. padding
- D. gap

21.

If an element has width: 200px, padding: 20px, and border: 5px, what is its total calculated width under the default box-sizing: content-box?

- A. 200px
- B. 225px
- C. 250px
- D. 240px

22. **Which property allows you to change the box model calculation so that padding and borders are included in the width?**

- A. box-layout: border-box;
- B. box-sizing: border-box;
- C. size-calculation: fixed;
- D. display: border-box;

23. **Which display value allows an element to sit on the same line as other elements but still respect width and height properties?**

- A. inline
- B. block
- C. inline-block
- D. flex-item

24. **Which CSS property is used to position an element relative to the browser window, so it stays in the same place even when scrolling?**

- A. position: relative;
- B. position: absolute;
- C. position: fixed;
- D. position: static;

25. **Which property is the primary container property used to start a Flexbox layout?**

- A. layout: flex;
- B. display: flex;
- C. flex-direction: row;

- D.
align-items: center;
-

Answers with Explanations

1. **B**
Explanation: CSS is the acronym for Cascading Style Sheets, referring to the way priority is determined when multiple rules apply to the same element. (Chapter [7](#))
2. **C**
Explanation: CSS uses forward-slash and asterisk / .. */ for multiline and single-line comments. // is for JavaScript. (Chapter [7](#))*
3. **A**
Explanation: The all shorthand property, when set to inherit, forces all properties of an element to match those of its parent. (Chapter [7](#))
4. **C**
Explanation: The hash (#) symbol is the ID selector in CSS. (Chapter [8](#))
5. **B**
Explanation: The > combinator is the child selector, targeting only immediate children. (Chapter [8](#))
6. **C**
Explanation: Selectors are grouped using a comma to apply the same declaration block to multiple elements. (Chapter [8](#))
7. **C**
Explanation: ID selectors have a higher specificity weight than class or type selectors. (Chapter [8](#))
8. **C**
Explanation: The !important declaration is used to override other declarations. (Chapter [8](#))

Explanation: :hover is a pseudo-class used to select elements when the user gestures with a pointing device. (Chapter [8](#))

9.

C

Explanation: Linking stylesheets in the <head> ensures the styles are loaded before the page content is rendered. (Chapter [7](#))

10.

B

Explanation: The style attribute allows you to apply CSS rules directly to a specific HTML element. (Chapter [7](#))

11.

B

Explanation: !important gives a declaration the highest precedence, overriding other declarations even with higher specificity. (Chapter [8](#))

12.

C

Explanation: The :focus pseudo-class applies when an element (like a text input) is selected or clicked into. (Chapter [8](#))

13.

C

Explanation: background-color sets the background; color sets the text color. (Chapter [7](#))

14.

A

Explanation: font-weight controls the thickness of text characters. (Chapter [7](#))

15.

B

Explanation: font-size is the standard property for defining text dimensions. (Chapter [7](#))

16.

B

Explanation: text-align: center; horizontally aligns the inline content (like text) to the middle. (Chapter [7](#))

17. **A**

Explanation: text-decoration is used to add or remove decorations

...like underlines or line-throughs. (Chapter [8](#))

18.

C

Explanation: font-family specifies a prioritized list of one or more font family names. (Chapter [7](#))

19.

B

Explanation: The order from inside to outside is: Content -> Padding -> Border -> Margin. (Chapter [9](#))

20.

C

Explanation: Padding is the inner space of an element. (Chapter [9](#))

21.

C

Explanation: Width (200) + Left Padding (20) + Right Padding (20) + Left Border (5) + Right Border (5) = 250px. (Chapter [9](#))

22.

B

Explanation: box-sizing: border-box; ensures the specified width/height includes the padding and border. (Chapter [9](#))

23.

C

Explanation: inline-block combines features of both: it flows with text (inline) but accepts dimensions (block). (Chapter [9](#))

24.

C

Explanation: fixed positioning removes an element from the normal document flow and positions it relative to the viewport. (Chapter [9](#))

25.

B

Explanation: display: flex; defines a flex container and enables a flex context for all its direct children. (Chapter [9](#))

Part IV

JavaScript Programming

JavaScript Programming

This part corresponds to **topic 034: JavaScript Programming** in the objectives of the Web Development Essentials Exam (Exam Code: 030-100). Although JavaScript is a general-purpose programming language, its ability to access and manipulate the webpage DOM plays a crucial role in web application development.

LPI allocates 10 weightage points to the topic 034. Its four subtopics and their respective distribution of weights are:

- 034 JavaScript Programming
- 034.1 JavaScript Execution and Syntax (weight: 1)
- 034.2 JavaScript Data Structures (weight: 3)
- 034.3 JavaScript Control Structures and Functions (weight: 4)
- 034.4 JavaScript Manipulation of Website Content and Styling (weight: 2)

These subtopics are covered in three chapters as follows:

Chapter [10](#)

Chapter [10](#) builds the fundamentals of JavaScript as a programming language. The basic building blocks of JavaScript, such as keywords, data types, variables, and operators, are explained. These concepts are essential for effective use of JavaScript as a front-end technology.

Chapter [11](#)

Like any programming language, JavaScript also has conditional and looping constructs. This chapter explains how to use if-else statements and how various loop types work in JavaScript. Chapter [11](#) also describes modular programming, with specific context for different types of functions in JavaScript.

Chapter [12](#)

Chapter [12](#) has special significance in the context of JavaScript as a front-end technology, and its ability to manipulate the webpage DOM. In this chapter, you learn about the properties and methods of the document object, how to traverse the DOM tree, and how to handle various DOM events.

To help you test and reinforce your learning from this part, a set of 25 MCQs has been included at the end.

10. JavaScript Fundamentals

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

Various polls and estimates consistently rank JavaScript as one of the top programming languages over the last decade. Not only is it one of the main front-end web technologies (used in an estimated 98% of websites), but it has also found its way into back-end development with the advent of Node.js. The emergence of JavaScript frameworks like React, Vue, and Express has also boosted its popularity.

A brief historical context of JavaScript appears in Chapter [1](#). This chapter covers essential features of JavaScript as a programming language, such as its execution and syntax, data types and variables, and operators.

Enabling JavaScript

JavaScript is primarily used in web applications. Hence, all modern browsers have a built-in JavaScript engine, enabled by default. Google Chrome and other Chromium-based browsers use the V8 engine. Firefox, on the other hand, uses the SpiderMonkey engine.

You can manually enable/disable JavaScript by accessing your browser's settings. In Chrome, for example, open the privacy and security settings and choose whether you want to allow/disallow using JavaScript. See Figure [10-1](#).

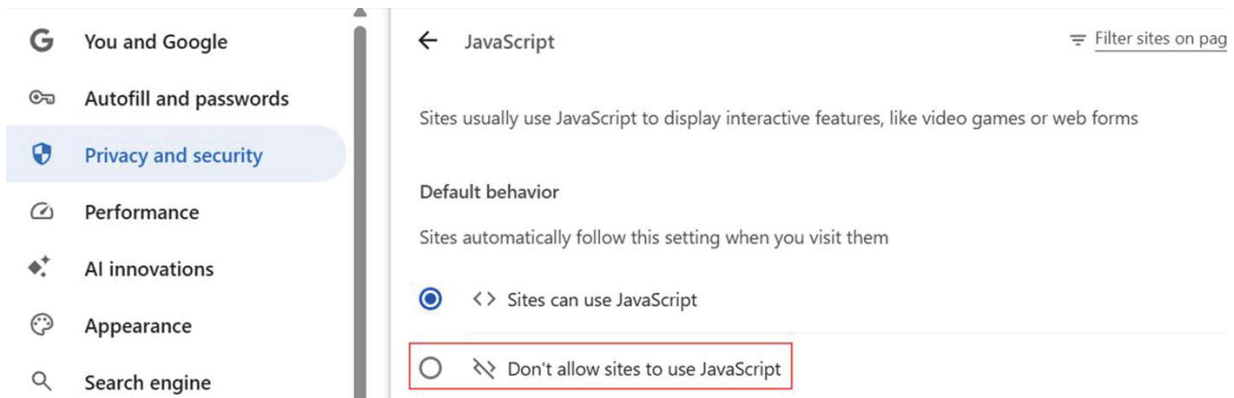


Figure 10-1 JavaScript settings

You can also disable JavaScript using the browser's developer tools. In Microsoft Edge, open the developer tools by pressing Ctrl+Shift+I, select Run Command from the ellipsis (...) menu, and choose the Disable JavaScript option. See Figure [10-2](#).

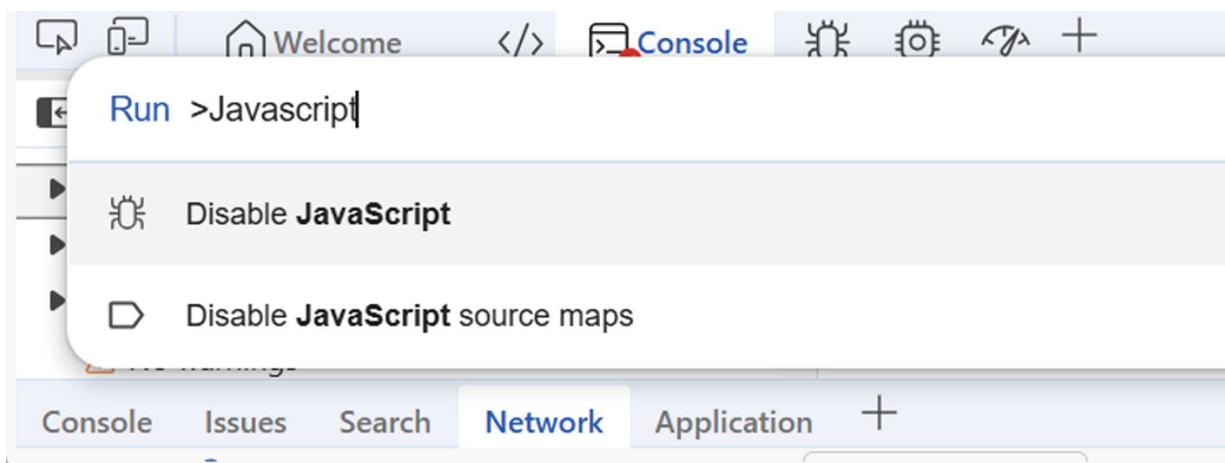


Figure 10-2 Developer tools run command

Once enabled, access the console tab. You can now execute JavaScript statements from the prompt. See Figure [10-3](#).

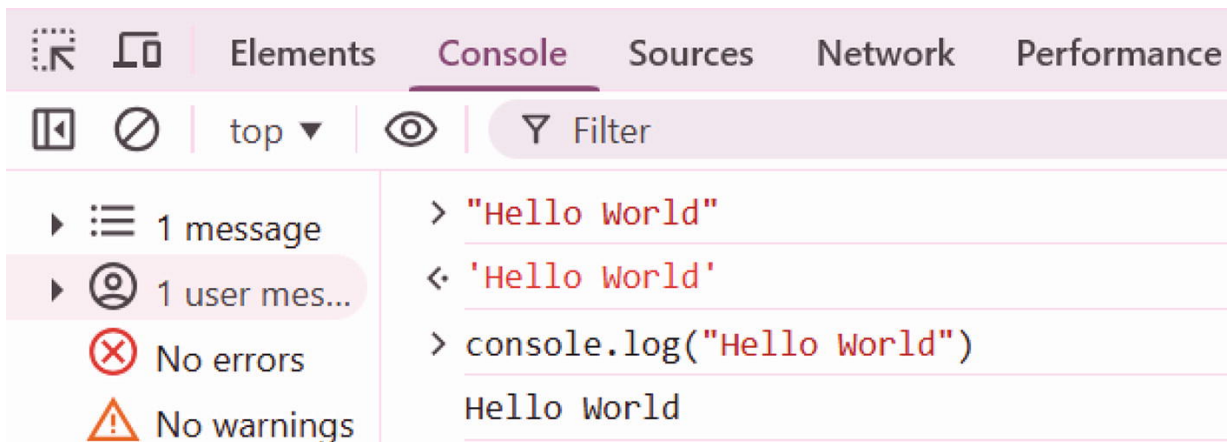


Figure 10-3 JavaScript in console

JavaScript Inside HTML

To use JavaScript in a web application, you need to place the JavaScript code inside HTML. One way to do it is to put the code inside the `<script>` element, which may be present either in the `<head>` section, the body section, or even after the body. The other, and more recommended approach is to include a `.js` file (containing the JavaScript code) with the `src` attribute of the `<script>` element.

Normally, the inline code in the `<script>` element or in a `.js` file contains function/class definitions, which are invoked in response to certain DOM events such as a button click. However, you can also put any executable code inside the `<script>` element.

If you put the `<script>` tag with a call to the `document.write()` function in it inside the `<head>` section.

```
<head>
  <script>
    document.write("Hello World")
  </script>
</head>
```

The Hello World message displays before the body of the page is loaded. However, if you put it inside the `<body>` section, it appears after the contents of the page preceding it are rendered.

```
<body>
<h2>JavaScript in Body</h2>
  <script>
    document.write("Hello World");
  </script>
<h2>JavaScript in Body</h2>
</body>
```

Here, the Hello World message appears between the two `<h2>` elements.

To link JavaScript code from a file (with the .js extension) to the current HTML document, set the `src` attribute to the file path.

```
<script src="myscript.js"></script>
```

This can be in the head or the body section. You can also place the JS script just before the closing `</body>` tag, which ensures that the DOM is loaded and available for your script.

Note the use of `document.write()` and `console.log()` statements in the preceding code snippets. Here, both `document` and `console` are the global objects in JavaScript. The `document` object is created when the HTML document is loaded in the browser. Its `write()` method renders content directly into the document's output stream. The `console` object accesses the browser's console. Its `log()` method echoes the input to the console, usually helpful for debugging the script. The `console` object is also available in Node.js.

Basic Syntax

JavaScript is a cross-platform object-oriented programming language. Most of its syntax is similar to C/C++ and Java.

Keywords

As in any human language, a computer language has a set of predefined words (called *keywords*). The set of keywords is usually very small.

Each keyword has well-defined rules of usage (syntax). As per the modern ECMAScript standards, there are 36 keywords.

Table

In addition, there are literal reserved words (true, false, null) and 22 future reserved words (a total of 61), all of which cannot be used as identifiers to name user-defined variables, functions, or classes. (*Note that the future reserved words may be used as identifiers in non-strict mode*). Keywords are lowercase and case-sensitive.

Statements

Each instruction in the JavaScript code is a statement. Normally, each statement should end with a semicolon (;) as in C/C++ or Java.

```
document.write("Hello World");
```

However, if a line break occurs, the JavaScript engine automatically inserts the semicolon, even if it is not explicitly present.

Hence, the following code is valid.

```
let a = 10
let b = 20
console.log(a + b)
```

However, putting the semicolon is considered best practice. If you need to treat a series of statements as a block (e.g., in a function definition or a loop), you need to put them in a pair of braces.

```
for (let i = 1; i <= 5; i++) {
    let square=i*i;
    document.write(`number: ${i} Square: ${square}
<br>`);
}
```

Comments

Any text not intended to be executed by the JavaScript engine is a comment. JavaScript uses the C++ style of single line comment (text starting with `//`) and multiline comments (text inside `/*` and `*/` markers).

```
// this is a single line comment
```

```
/*  
this text is treated as a  
multiline comment  
*/
```

JavaScript also supports the hashbang comment syntax, which is a single line that begins with `#!` characters. This is valid at the start of a script or module.

```
#!/usr/bin/env node
```

The hashbang-style comment is mainly found in command-line environments like Node.js and is not really useful in client-side JavaScript code embedded in HTML.

Identifiers

There are various elements in a computer program, such as variables, functions, and classes. Each of them needs to be assigned a unique label or identifier, so that they can be easily referred to in the program.

The following are some examples of JavaScript identifiers.

```
const MAXMARKS=100;
```

```
var 姓名="王";
```

```
let _count=0;
```

```
#year = 2000;
```

```
const capital_cities = ["London", "Paris", "New  
Delhi"];

function sayHello() {  
  console.log("Hello");  
}

class Employee {  
  /*class body*/  
}
```

The identifiers consist of alphanumeric characters, underscores (`_`), `#` to define a private variable, and dollar signs (`$`). An identifier cannot start with a digit. However, JavaScript allows using non-ASCII characters in an identifier.

Data Types

Computer programs process data. Every programming language has its own nomenclature for the types of data it processes. JavaScript has two data types: primitive types and object types.

Primitive Types

Primitives are immutable and stored by value. There are seven primitive types in JavaScript.

`boolean` represents the logical state of an expression, either `true` or `false`.

`null` indicates the intentional absence of a value.

`undefined` represents any property that has not been assigned any value.

`number` is a positive or negative number with or without a floating point; for example, 42, -5.75, 3.14159, 1E04, 3.2e-3. (E or e is a scientific notation that means 10 raised to.)

`bigint` is an integer number with arbitrary precision. The letter `n` appended to an integer makes it a `bigint`. For example: `101n`, `9007199254740992n`.

`string` is a sequence of characters enclosed between single or double quotes. For example: `"Hello"` or `'hello'`. You can also use backticks to form template literals (such as ``Hello``) that are especially used for multiline strings.

`symbol` is a JavaScript built-in `Symbol()` function that returns a unique and immutable identifier. For example, `Symbol("id")`.

Object Types

Unlike primitives, object types are mutable and stored by reference. An object is a collection of properties, each property being a key-value pair. The key must be a string. A literal representation of a JavaScript object includes one or more properties within the curly braces.

```
let employee = {  
  name: "John Doe",  
  email: "john@example.com"  
};
```

JavaScript offers built-in object types such as array, map, set, and function. The dot operator lets you access a property of the object. For example, `employee.name`.

Array: An array is a collection of data values, not necessarily of the same type. A list of comma-separated data items/elements placed inside square brackets (`[` and `]`) forms an array object.

```
const cities = ["London", "Paris", "New Delhi"];
```

Each item in the array is identified by an incrementing positional index starting with 0. Hence, `cities[2]` returns the third element in the array: `New Delhi`.

The array object's `length` property returns the number of elements in it. JavaScript arrays are dynamic. You can insert or remove elements

from an array. The `pop()` method removes the element at the last index, and the `push()` method adds a new element, while the `pop()` method extracts an item from the end.

```
let cities = ["London", "Paris", "New Delhi"];
cities.pop();
cities.push("New York");
document.write(cities);
```

In JavaScript, a map is similar to the standard object; it is also a collection of key-value properties. However, unlike in an object, the key in the map property may be of any type.

A set is a collection of unique values (no keys). The value can be of any type (primitive or object type).

The typeof Operator

JavaScript's `typeof` operator returns a string that corresponds to the data type of the operand expression. Include the following code inside the `<script>` and `</script>` tags and place it in the body section.

```
console.log(typeof("John Doe"));
console.log(typeof(100));
console.log(typeof({ name: "John Doe", email:
"john@example.com"}));
```

The console tab of your browser's developer tools shows the corresponding data types: string, number, and object.

Variables

Variables play an important role in programming. Data values of any of the preceding types need to be stored at a certain address in the computer memory for processing. Instead of the binary (or hex) address, high-level programming languages (including JavaScript) assign it a convenient label. Hence, variables are the named memory

locations that make data storage and retrieval easier for the programmer.

In JavaScript, the = symbol is the assignment operator. It is used to assign a value to the variable.

```
var_name = value;
```

All the rules for naming an identifier apply to the variable name. The following are all valid assignment statements.

```
str = 'Hello';  
var age = 25;  
let user = 'John';  
const pi = 3.14159;
```

As you can see, the variables age, user, and pi are preceded by three keywords: var, let, and const, respectively. Moreover, the str variable doesn't have any such preceding keyword.

The declaration without any keyword is historically the oldest approach. As such, a variable becomes a global variable; this practice is discouraged in modern JavaScript. Later on, the keyword var was introduced. It is generally used to declare a variable inside a function, as it ceases to exist outside the function's scope.

Modern JavaScript (ECMAScript 2015, or ES6 standard) prescribes the use of let and const keywords.

A variable declared with the let keyword also has block scope, but it must be declared before use. You may combine the declaration and assignment in a single statement. (This is also the case with the var keyword).

```
let user;  
user= 'John';  
//or  
let user= 'John';
```

Moreover, a variable once declared with `let` cannot be redeclared in the same scope. (Note that if declared with `var`, the variable can be redeclared.)

```
let user= 'John';  
let user="Doe";
```

The console tab of your browser’s developer tools shows “SyntaxError: Identifier 'user' has already been declared”.

As it suggests, the `const` keyword creates a variable whose value cannot be changed. In other words, you can assign it a value only once. You also cannot split the declaration and initialization into two statements. If you do (as in the following snippet), the console shows “SyntaxError: Missing initializer in const declaration”.

```
const name;  
name= 'John';
```

Strict Mode

Although early versions of JavaScript expected a variable to be declared with the `var` keyword, it also allowed variables to be declared without it; thus populating the global namespace and leading to potentially unsafe code. The introduction of strict mode in the ES5 standard (in 2009) is aimed at preventing unwanted behavior such as creating global variables and accidentally deleting variables.

JavaScript code (especially code embedded in an HTML script) runs in non-strict, or “sloppy,” mode. To enable strict mode, you should have the “use strict” directive at the top of the script. Once enabled, declaring a variable without any of the keywords (`var`, `let`, or `const`) will not be allowed, as strict mode requires explicit declaration.

```
"use strict";
```

```
x = 10; //this throws ReferenceError: x is not  
defined
```

Modern JavaScript is automatically strict. With the new features like modules and classes, JavaScript is already strict by default, so explicit use of "use strict" is not mandatory.

Dynamic Typing

JavaScript is a dynamically typed language. The type of the variable depends on the value assigned to it, and not the other way round as in statically typed languages like C/C++ and Java.

In C/C++/Java, you need to specify the type in the variable declaration.

```
int x=10;
```

This ties the variable x with the int type for the entire scope, and you cannot assign a value of any other type.

In JavaScript, you just declare the variable. The type of the variable (returned by the typeof operator) changes as per the value assigned to it.

```
let x;  
x=10;  
console.log(typeof(x));  
x="Hello World";  
console.log(typeof(x));
```

The console output echoes number as the type initially and string subsequently, thus showing the dynamic nature of typing in JavaScript.

Operators

Like keywords and variables, the operators are also one of the fundamental concepts in the theory of programming. Operators are non-alphanumeric characters predefined in the language to carry out a certain operation on their operands. (Although some operators can be mistaken as keywords. For example, typeof, new, and instanceof are

operators and neither keywords nor functions). For example, the + symbol in JavaScript (as in most other languages) is defined to perform addition of the two numeric operands on either side.

JavaScript has more than 60 operators, classified in different categories. The ones that are most commonly used are the familiar arithmetic operators (+, -, * and / representing addition, subtraction, multiplication and division respectively), the comparison operators (the <, >, <=, >=, == and != respectively representing the less than, greater than, less than or equal to, greater than or equal to, is equal to and is not equal to operations) and the logical operators (logical AND, logical OR and logical NOT represented by &&, || and ! symbols respectively).

Arithmetic Operators

The subtraction, multiplication and division operators need two numeric operands. However, the + operator works a bit differently. For two numeric operands, it performs their addition, but if one of them is a string, it acts as a concatenation operator.

```
let x,y,z;  
x=10;  
y=20;  
z=x+y; //z is 30 here  
x="Hello";  
y="World";  
z=x+y; //z is HelloWorld
```

The other frequently used arithmetic operators are ++ as an increment and -- as a decrement operator.

```
let x,y;  
x=10;  
y=20;  
x++; // x is 11  
y--; // y is 19
```

Increment and decrement operators may be put before (prefix) or after (postfix) the variable.

The % symbol is reserved as the modulus operator, and it returns the remainder of the division of its two numeric operands.

```
let x, y;  
x=10;  
y=3;  
z=x%y; // z is 1
```

The exponentiation operation is performed with the ** symbol. If the first operand is x and the second is y, it returns the yth power of x (or x raised to y).

```
let x, y;  
x=10;  
y=3;  
z=x**y; // z is 1000
```

Comparison and Logical Operators

The conditional and looping constructs in the code are controlled by the comparison operators. These operators always return a Boolean value, true or false. An expression with a comparison operator is called a Boolean expression.

```
let x, y, z;  
x=5, y=7;  
z = x<y; // true  
z = x>y; // false  
z = x==y; // false  
z = x!=y; // true
```

You can also use <= (less than or equal to) and >= (greater than or equal to) operators in JavaScript. There are two more operators in this

category: strict equality (===) and strict non-equality (!===) operators. You will learn about them in the next section.

Logical operators join two Boolean expressions. You can perform AND and OR operations.

The && symbol stands for logical AND. It returns true only if both of its operand expressions are true.

```
let x, y, z;
x=5, y=7;
z = x<10 && y>5; // z is true
z = x<10 && y>10; // z is false
z = x>10 && y>10; // z is false
z = x>10 && y<5; // z is false
```

Since both expressions (x<10 and y>5) are true, their AND operation results in true. In all other cases, at least one operand is false.

The OR operator is represented by the || (two pipe characters) symbol in JavaScript. This operator returns true if any of its operand expressions is true. The result will be false only if both are false.

```
let x, y, z;
x=5, y=7;
z = x<10 || y>5; // true
z = x>10 || y>5; // true
z = x>10 || y<5; // false
z = x<10 || y<5; // true
```

Other operators in JavaScript include bitwise operators, conditional operators, and more. Their detailed explanation is not within the scope of this book.

Compound Assignment Operators

JavaScript uses the = key as an assignment operator. The value of the expression on its right is stored in the variable on its left.


```
var_name = expression;
```

Note that the statement $x=y$ is not the same as $y=x$. There must be only one variable on the left of the $=$ sign. The operand on its right may be any expression (arithmetic, Boolean, etc.)

Look at the following statement.

```
a=a+5;
```

While in Mathematics, this is invalid (how can x be equal to $x+5$?) because it's an equation, but in JS (and in other languages) it is perfectly valid because it is a statement. The expression on the right adds 5 to a , and the result is assigned back to a . The same is true with the other operators.

The two operators in the preceding statement ($+$ and $=$) have a shortcut representation in JavaScript. You can very well write it as follows.

```
a+=5; // x increments by 5
```

You may call $+=$ as the add-and-assign operator. It is a compound assignment operator. The following are others in this category.

```
a += b // Add and Assign operator. same as a = a + b
```

```
a -= b // Subtract and Assign operator. same as a = a - b
```

```
a *= b // Multiply and Assign operator. same as a = a * b
```

```
a /= b // Divide and Assign operator. same as a = a / b
```

```
a %= b // Modulus and Assign operator. same as a = a % b
```

```
a **= b // Raise to and Assign operator. same as a = a ** b
```

Type Conversion

Not only is JavaScript dynamically typed, but it is also a loosely typed language. Take the addition operation for example. Typically, the + operator requires two numeric operands. However, if one of them is a string, the loose typing nature of JavaScript coerces the other operand to string type and performs concatenation.

```
let x, y, z;  
x=5, y=7;  
z = x+y; // z is 12  
  
x=5, y='7';  
z = x+y; // z is '57'
```

In the first case, both x and y are number types, so the + operator can perform addition. However, in the second case, y is a string. Here, JavaScript internally coerces x to a string as well and performs concatenation.

In JavaScript, coercion (type conversion) is either implicit or explicit.

Implicit Conversion

The second scenario in the preceding code is implicit coercion, where JavaScript converts one type to another. However, the choice of operator determines which operand is coerced to match the data types.

```
x=5, y='7';  
z = x*y; // z is 35
```

With the multiplication operator, the string operand is coerced to become a number and not the other way around as with the + operator. The same is true with -, /, and % operators.

Explicit Coercion

Instead of letting the JavaScript engine perform automatic conversion, you can employ JavaScript's built-in functions to convert an operand of one type to another.

Conversion to String

The built-in `String()` function returns a string representation of its number type argument.

```
let x, y;  
x=5;  
y=String(x);  
console.log(typeof (y));
```

The JavaScript console confirms that the `String()` function explicitly converts a number to a string.

Most of the data types have a built-in `toString()` method that returns a corresponding string representation.

```
let x, y;  
x=5;  
y=x.toString(); // number to string. y is '5'  
  
x=true;  
y=x.toString(); // boolean to string. y is 'true'
```

Conversion to Number

For explicit conversion to a number type, use the `Number()` function.

```
let x, y;  
x='5';  
y=Number(x);  
console.log(typeof (y));
```

Converting a Boolean to number type returns 1 as true and false values are equivalent to 1 and 0, respectively.

```
x=true;
y=Number(x); // boolean to number. y is 1
```

If the string holds a valid representation of a number type literal, the `parseInt()` and `parseFloat()` functions return its integer and floating-point equivalent.

```
let x, y;
x='123.45';
y=parseInt(x);
console.log(y);

y=parseFloat(x);
console.log(y);
```

The first console output is 123 (an integer) and the second 123.45 (a floating point number).

Strict Equality Operator

JavaScript has `===` and `!==` operators. They are called *strict equality* and *strict not equality* operators, respectively. You would recall that the `==` and `!=` operators are also there. Let's now call them non-strict or loose equality and non-equality operators.

The script versions of operators are necessitated because of the implicit coercion feature in JavaScript, because of which 5 and '5' pass the equality test.

```
let x, y, z;
x=5, y='5';
z = x==y;
console.log(z); // z is true
```

But change to the strict equality operator, and you will see that it returns false.

```
let x, y, z;
```

```
x=5, y='5';
```

```
z = x===y;  
console.log(z); // z is false
```

Similar behavior is seen with the strict non-equality operator. `x!=y` should return false and `!==` should return true.

Points to Remember

- JavaScript is one of the most popular computer languages, used as the front-end web technology. It adds interactivity to the HTML elements in a web page.
- Most modern browsers have JavaScript enabled by default. Your browser's developer tools are useful for debugging the JavaScript code embedded inside HTML.
- JavaScript code can be placed inside the `<script>` element, either in the head section or the body section.
- To include the JavaScript code in a .js file, assign its path as the value of the `src` attribute of the `<script>` element.
- JavaScript has 61 words, including the core keywords (36) and other future reserved words. They cannot be used as identifiers.
- An identifier is a user-specified name for different programming elements such as variables and functions.
- JavaScript defines seven primitive data types, including number, string, Boolean, and others.
- Object data types are collections of properties, each being a key-value pair. An array is one of the built-in object types, in which the comma-separated list of values is placed inside the square brackets. Each element in the array has a positional index starting with 0.
- JavaScript is a dynamically typed language. You don't need to specify the type in the variable declaration. The type depends on the value assigned.

- JavaScript has three keywords (var, let, and const) for variable declaration.
 - The strict mode prevents a variable from being in the global namespace and from accidentally deleting it.
 - JavaScript has different operators. The familiar arithmetic, comparison, and logical operators are available.
 - JavaScript is a loosely typed language. It tries to implicitly coerce an operand of one type to another type. In addition, you can explicitly convert one type to another using appropriate built-in functions.
-

Summary

This chapter is an overview of the fundamental building blocks of JavaScript. Starting with how to enable JavaScript in your browser and including the JavaScript code in HTML, you learned about the primitive and object data types, the concept of variables and operators. This chapter also explains why JavaScript is dynamic and loosely typed.

The next chapter covers an important aspect of JavaScript programming: conditionals, loops, and functions.

11. JavaScript Control Structures

Malhar Lathkar¹ 

(¹) Near Rachewad Hospital, Nanded, Maharashtra, India

Out of the three core web technologies, HTML and CSS are declarative languages used to describe the structure and presentation of a web page, whereas JavaScript is a full-fledged programming language. It is often said to be a Turing-complete language because of its flow control statements that allow developers to implement algorithms and perform arbitrary computation.

This chapter focuses on the programming elements in JavaScript: conditionals, looping constructs, and functions.

Conditional Statements

A computer program is a series of instructions, executed sequentially (from the first statement to the last) by the interpreter/compiler (in this case, the JavaScript runtime engine in the browser). However, you can make it execute a statement or a group of statements if a specified Boolean expression is true or false.

In the previous chapter, you learned about the comparison and logical operators that always return a Boolean value (true or false). They play an important role in putting the conditional logic in code. JavaScript provides if-else and switch-case statements for this purpose.

The if Statement

The if statement lends the basic decision-making ability to a computer program. JavaScript prescribes a simple syntax for the if statement.

```
if (boolean expression) {  
    //statements to be executed when true  
}
```

The if keyword is always followed by parentheses, which include a Boolean expression (or condition). As you know, the Boolean expression is either true or false. The code block of statements (inside the curly braces) after the parenthesis is executed if the condition evaluates to true.

```
let age;  
  
age = 20;  
  
if (age>=18) {  
    document.write ("You are eligible to vote!");  
}
```

Set age to any value less than 18, and you won't get the message rendered in the browser as the condition turns false.

Note that the statements after the conditional block are executed anyway—after the message if the condition is true, or without the message if it is false.

You can add an optional else clause after the conditional block. It is also followed by a code block, meant to be executed if the condition (in front of the if keyword) is not true.

```
if (boolean expression) {  
    //statements to be executed when true  
} else {  
    //statements to be executed when not true  
}
```


Note The curly braces after the parenthesis or else keyword are necessary if the code block has more than one statement. You may choose not to use them if there is only one statement under the if or else clause.

Let's implement the if-else logic in a simple yet practical use case, where JavaScript code renders a fail/pass message depending on whether the marks scored are less than 50.

```
let marks;

marks = prompt("enter the marks obtained");

if (marks<50) {
    document.write ("Your score is below the passing
criteria");
} else {
    document.write("Your score fulfills the passing
criteria");
}
```

JavaScript's built-in `prompt()` function adds realistic interactivity here. When the web page is loaded, a dialog box pops up in the browser, asking for input from the user. See Figure [11-1](#).

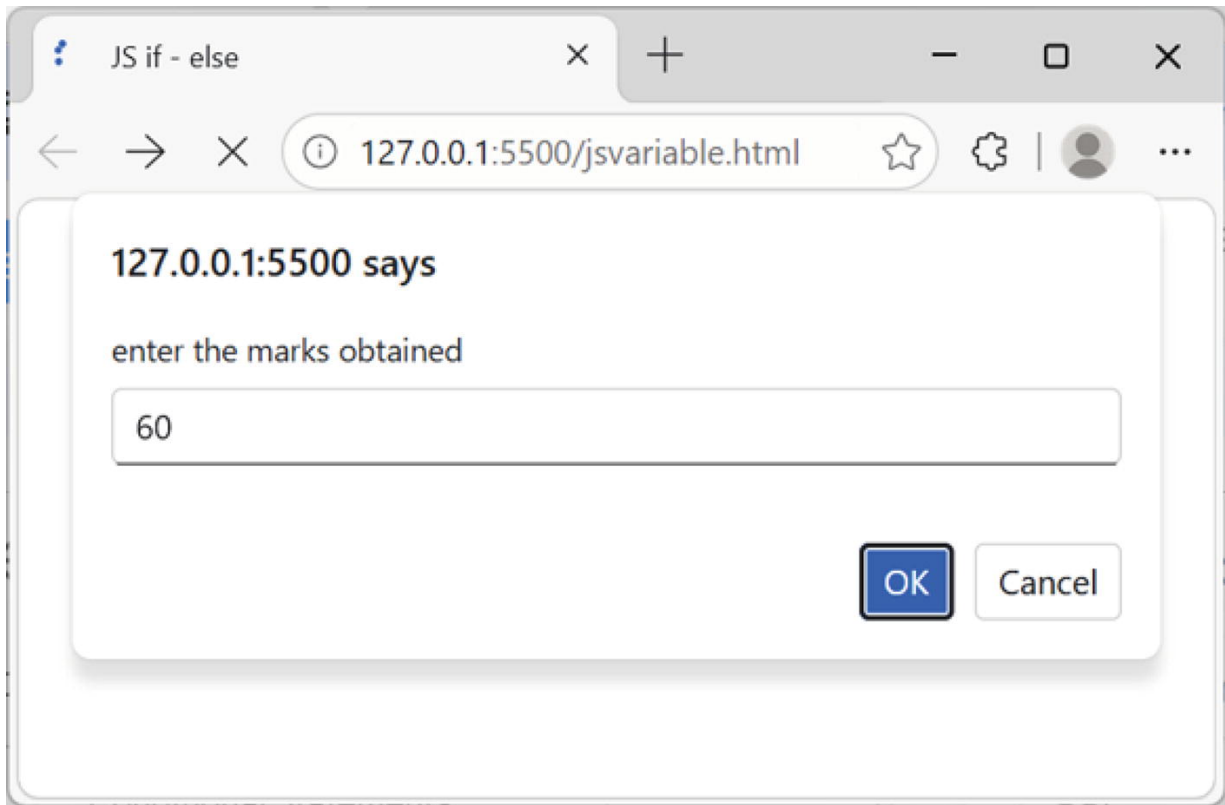


Figure 11-1 prompt() function

Depending on the number entered, the appropriate message string is rendered.

Nested if

You may need to put another if-else construct inside an existing if or else block. Such usage is commonly known as nesting if statements.

```
if (expr1) {  
    if (expr2) {  
        //statements to be executed when expr2 is  
true  
    } else {  
        //statements to be executed when expr2 is not  
true  
    }  
} else {
```

```

    if (expr3) {
        //statements to be executed when expr3 is
true
    } else {
        //statements to be executed when expr3 is not
true
    }
}

```

expr2 is evaluated only if expr1 is true. Similarly, expr3 is evaluated if expr1 is not true. Such a nested use of if-else statements can be extended to any number of levels.

The following example employs nested if statements to decide whether a given number is divisible by two, three, both, or neither.

```

let num;

num = prompt("enter a number");

if (num%2==0) {
    if (num%3==0) {
        document.write (`${num} is divisible by 3 and
2`);
    } else {
        document.write (`${num} is divisible by 2 but
not by 3`);
    }
} else {
    if (num%3==0) {
        document.write (`${num} is divisible by 3 but
not by 2`);
    } else {
        document.write (`${num} is neither divisible by
2 nor by 3`);
    }
}

```

```
}
```

The result of executing the preceding code is straightforward. It starts with checking whether the given number is divisible by two. In either case, the output depends on whether the number is divisible by three. Try refreshing the browser and input different numbers.

```
10 is divisible by 2 but not by 3
55 is neither divisible by 2 nor by 3
24 is divisible by 3 and 2
75 is divisible by 3 but not by 2
```

String Interpolation

Note that the strings in the `document.write()` statements in the preceding code snippets use the backticks. As you learned in the previous chapter, the strings formed with backticks are called template literals. In addition to constructing multiline strings, the template literals have an important feature called string interpolation.

String interpolation is the process of creating dynamic strings by substituting the values of variables/expressions inside the backticks at runtime. The variables or expressions are put inside the curly braces and prefixed by the `$` symbol. The `${expr}` notation acts as the placeholder within the template literal.

Have a look at the following script.

```
let x=10, y=20;
document.write(`z=${x}+${y}`);
```

The JavaScript engine computes the expression in the placeholders and concatenates their result with the static parts of the literal to construct the string dynamically as follows.

```
z=10+20
```

The else if Statement

Instead of the nested if constructs, the use of “else if” syntax proves to be more effective, especially if the conditions are mutually exclusive. An additional if statement appears immediately after each else clause. If all the conditions turn out to be false, the program flow enters the code block of the last else clause. Moreover, as soon as one of the conditional blocks gets executed, the program control exits the condition chain.

```
if (expr1) {  
    //Statements to be executed if expr1 is true  
} else if (expr2) {  
    //Statements to be executed if expr2 is true  
} else if (expr3) {  
    //Statements to be executed if expr3 is true  
} else {  
    //Statements to be executed if none of the  
    expressions is true  
}
```

To demonstrate this logic, let's extend the first use case in this section and decide the student's result based on the marks scored.

Assume that different grades are determined by the mark ranges, as shown in the following example.

- grade A+ for marks ≥ 90 ,
- grade A for marks ≥ 75 to 89,
- grade B+ for marks ≥ 60 to 74,
- grade B for marks ≥ 50 to 59, and
- grade F for marks < 50

The following code uses the condition-chaining algorithm for this purpose.

```
let marks, grade;  
marks = prompt("enter your marks");  
if (marks < 50)  
    grade = "F";
```

```

else if (marks<60)
    grade = "B";
else if (marks<75)
    grade = "B+";
else if (marks<90)
    grade = "A";
else
    grade = "A+";

document.write (`Marks: ${marks}. Your grade is
${grade}`);

```

It can be seen that with the “else if” syntax, the code becomes more readable, especially when only one of several conditions is true.

switch and case

Along with if and else, JavaScript also defines the switch and case keywords, which are also used to express conditional logic. However, their usage is different. With if-else, the branching depends on a Boolean expression; switch-case is suitable when you need the program to compare a single expression against its different discrete values. While you can still use the if statement with the equality operator (==), using switch-case is more compact and convenient.

Have a look at the following JavaScript code. It accepts two numbers and an operation code (1 to 4 for arithmetic operations: add/subtract/multiply/divide, respectively) and shows the result.

```

let num1, num2, code, result;

num1=Number(prompt("enter a number"));
num2=Number(prompt("enter another number"));
code=Number(prompt("enter operation code 1-
4")));

if (code==1) {

```

```

        result=num1+num2;
    } else if (code==2) {
        result=num1-num2;
    } else if (code==3) {
        result=num1*num2;
    } else if (code==4) {
        result=num1/num2;
    }

    document.write(`Result=${result}`);

```

Although the processing logic in the preceding program is fairly straightforward, it can become fairly lengthy. If there are many such equality conditions to be compared, multiple pairs of curly braces can be confusing and cluttered. The switch-case syntax offers to simplify things.

As the keywords suggest, the idea is to switch the program flow to different cases of an expression. The expression itself is the parameter to the switch keyword. Inside the ensuing pair of braces, you need to specify each value as a case and the statements to be executed.

```

switch (expr) {
    case val1:
        //statements to be executed when expr equals
val1
    case val2:
        //statements to be executed when expr equals
val2
    ...
    case valN:
        //statements to be executed when expr equals
valN
    default:
        //statements to be executed when expr equals
any other value

```

```
}
```

You can see that the code has already become more compact, there being only one pair of braces. Note the use of the default keyword; it acts as the case to be evaluated for all other values of the expression.

There's one problem with the preceding approach, though. Assuming that `expr` equals `val2`, the statements after the `:` symbol will be executed, but the execution will continue through the rest of the statements until it reaches the closing brace. To prevent this fall-through behavior, you need to put a `break` statement at the end of each case block. This takes control immediately out of the switch construct. The following is a more accurate representation of switch-case syntax.

```
switch (expr) {  
    case val1:  
        //statements to be executed when expr equals  
val1  
        break;  
    case val2:  
        //statements to be executed when expr equals  
val2  
        break;  
    ...  
    case valN:  
        //statements to be executed when expr equals  
valN  
        break;  
    default:  
        //statements to be executed when expr equals  
any other value  
}
```

This calculator program becomes quite clutter-free when the if-else construct is replaced with switch-case syntax, as in the following snippet.


```
let num1, num2, code, result;

num1=Number(prompt("enter a number"));
num2=Number(prompt("enter another number"));
code=Number(prompt("enter operation code 1-
4")));

switch (code) {
case 1:
result=num1+num2;
break;
case 2:
result=num1-num2;
break;
case 3:
result=num1*num2;
break;
case 4:
result=num1/num2;
}

document.write(`Result=${result}`);
```

Loops

One of the greatest strengths of a computer program is its ability to repeat a task reliably. The looping (repetition) feature available in programming languages (including JavaScript) makes it possible to execute the same set of instructions repeatedly for a desired number of times.

For example, the calculator program may need to repeatedly read numbers from the user and display the result until the user decides to stop. JavaScript provides several looping constructs (such as while, do...while, and variants of the for statement). One can choose the most appropriate one depending on the requirement.

The usage of these keywords has three requirements: a Boolean expression or condition that controls the loop, a code block to be repeatedly run, and some mechanism to terminate the repetition (in the absence of which it turns into an undesired infinite loop).

The while Loop

As with the if statement, the while keyword also has a pair of parentheses containing a Boolean expression, followed by a code block. However, unlike the if statement, the code block executes repeatedly as long as the condition remains true. The iterative execution of the block stops as soon as the condition is false, and the program flow continues to execute the subsequent statements after the block.

```
while (expr) {  
    //Statements to be executed as long as expr is  
    true  
}
```

Any looping construct must have a provision that makes the expr false so that it terminates. The easiest and most common approach is to keep a count of repetitions until it reaches the desired number. This type of loop is usually called the counted loop.

```
let count=0;  
  
while ( count<5 ) {  
    document.write( "Hello, World!<br>" );  
    count++;  
}
```

The preceding code prints the message “Hello World!” five times. How? Let’s go over how the while loop works.

- The program reaches the while statement with count set to 0.
- The JavaScript engine checks whether the expression `count < 5` is true.

- Since it is true, the first iteration takes place. The message is displayed, and the count is incremented.
- The control returns to the beginning of the block and checks the condition.
- If true, the next iteration takes place.
- If false, the loop terminates, and the flow jumps to the statement (if any) after the closing brace.

The looping block may have any number of statements. Note that for the iteration to start, the condition before the block must be true. In other words, the loop may not run at all if the condition is false initially.

Recall that if a loop is controlled by an expression that always remains true, it is an infinite loop.

```
while (true) {  
    //Statements  
}
```

Instead of forming a counted loop, the looping can be controlled by an expression whose value depends on a variable/object inside the block itself. Think of the calculator program in the previous section. Can you make it repeatedly perform calculations till the user wants to stop?

The do while Loop

The “do . . while” looping construct is an upside-down version of the while loop.

```
do {  
    //Statements to be executed as long as expr is  
    true  
} while (expr);
```

The code block to be repeated sits between the two keywords: do and while. (Note that while is after the block, not before). The program flow enters the looping body unconditionally. The test check (whether to repeat again or not) is placed after the closing brace.

```
let count=0;

do {
    document.write( "Hello, World!<br>" );
    count++;
} while ( count<5 );
```

The preceding code also prints the message “Hello World!” five times, as the while loop version. However, the do-while loop works a little differently. The following explains how.

1. The program reaches the do keyword with count set to 0.
2. The program flow enters the looping block.
3. The first iteration takes place. The message is displayed, and the count is incremented.
4. When it reaches the closing brace, the condition count<5 is checked.
5. If true, the control goes back to the starting brace, and the next iteration takes place.
6. If false, the loop terminates, and the subsequent statement (if any) is executed.

The main difference between the while and do-while loop is where the condition check happens. In the former case, the condition is evaluated before entering the looping body, whereas in the latter (do-while), it happens after the looping body (i.e., the closing brace). As a result, the do-while loop guarantees at least one execution of the block no matter the condition. On the other hand, the loop won't run at all if the condition is false before entering the block.

To verify this behavior, set count to a value more than 10 (like count=50) in both the code snippets (of while and do-while examples)

and check the result. In the first case, you won't get anything displayed in the browser. In the second code, you should get to see the "Hello World!" message rendered once.

There's one more subtle difference between the syntax of while and do-while. There is a semicolon at the end of the parentheses in the latter case.

The for Loop

Developers generally prefer using a for loop over while or do...while, especially when the number of repetitions is known. The compact syntax of the for loop makes it more convenient to use in such cases.

```
for (initialization; condition; step) {  
    // code block to be repeated  
}
```

The semicolon-separated expressions inside the parentheses following the for keyword define the three components of the loop: initialization of the loop variable, a condition that checks whether the loop variable has reached its limit, and an update step that modifies the variable after each iteration.

```
for (let count = 1; count <= 5; count++) {  
    document.write("Hello World!<br/>");  
}
```

This also prints the Hello World message five times, but the execution algorithm is different.

- When the program reaches the for statement, the count variable is initialized to 1.
- The condition (count <= 5) is evaluated.
- If false, the loop is not executed; control goes to the next statement after the closing brace.
- If true, the program enters the loop body, and the message is displayed.

- When the end of the block is reached, the count is updated.
- Control returns to the condition check before the next iteration.

The semicolon-separated parameters in the for loop syntax are optional, although semicolons are mandatory.

```
for ( ; ; ) {  
  // statements  
}
```

In such a case, it is an infinite loop, and you may have to terminate it with a break statement inside. The break statement is explained later in this chapter.

In addition to the classical for loop, JavaScript offers two more variants that work with objects, arrays, and strings.

for...in

You may recall that in JavaScript, the Object data type is a collection of enumerated properties, each property being a key-value pair.

```
let employee = {  
  name: "John Doe",  
  age: 30,  
  email: "john@example.com"  
};
```

You can retrieve the value associated with a key. The expression `employee["age"]` returns 30. The for...in loop iterates over all the properties available to the object.

```
for (let key in employee) {  
  document.write(`Key: ${key} Value:  
${employee[key]}<br>`);  
}
```

The browser output displays the key-value pairs.

Key: name Value: John Doe
Key: age Value: 30
Key: email Value: john@example.com

for...of

ES6 introduced the “for...of” variant of the for loop, especially to iterate over iterable object types such as arrays and strings. Each element in an array has an index starting with 0.

```
for (variable of iterable) {  
    //Statements  
}
```

Here, the variable attains the successive values from the iterable object on each iteration until the last element is reached.

The following JavaScript code returns the sum of all numbers in a given array.

```
let marks = [60, 50, 80, 70, 40];  
let sum=0;  
for (let mark of marks) {  
    sum += mark;  
}  
document.write("Sum: " + sum);
```

A JavaScript string is also an iterable of characters. You can use the for...of loop to traverse one character of the string at a time.

```
let message = "Hello World";  
  
for (let char of message) {  
    document.write(char + "<br>");  
}
```

break and continue

The while and for loops repeat as long as the value of a Boolean expression is true. Sometimes, however, it may be necessary to exit the loop before it becomes false. In other situations, you may want to abandon the current iteration and proceed to the next one. JavaScript provides the break and continue statements for these purposes.

You have already come across the break keyword used along with the switch-case construct, where it prevents the flow from falling through the cascade of cases. Here, it is about employing the break keyword inside the loop body. When encountered, the program flow jumps out of the loop to the statement immediately after the closing brace.

```
while (loop condition) {  
    . .  
    . .  
    if (expr)  
        break;  
    // remaining statements in loop  
}  
// statements after loop
```

The break statement can appear anywhere in the loop body, in any type of loop (while, do... while, and all types of for loops).

The following code example has a loop designed for 10 repetitions but terminates when it detects that the count variable has exceeded 5.

```
let count=0;  
  
while (count<10) {  
    count++;  
  
    if (count>5)  
        break;  
    document.write("count:"+count+"<br>");  
}
```


You frequently find the break statement being employed to terminate an infinite loop, especially when specifying the loop condition is not possible.

The continue statement's behavior is almost the opposite. If the program flow encounters it anywhere inside the loop body, the subsequent statements in the current iteration are skipped, and the control is transferred back to the beginning of the loop.

```
while (loop condition) {  
    . .  
    . .  
    if (expr)  
        continue;  
    // remaining statements in loop  
}
```

Similar to break, the continue statement may also be employed in all types of loops. Have a look at the following for loop. The continue statement skips the print statement whenever the count is divisible by 2, and only the odd numbers are displayed.

```
for (let count=1; count<=10; count++) {  
    if (count%2==0)  
        continue;  
    document.write("count:"+count+"<br>");  
}
```

The contrasting behavior of break and continue statements can be better understood in the flowcharts shown in Figure [11-2](#).

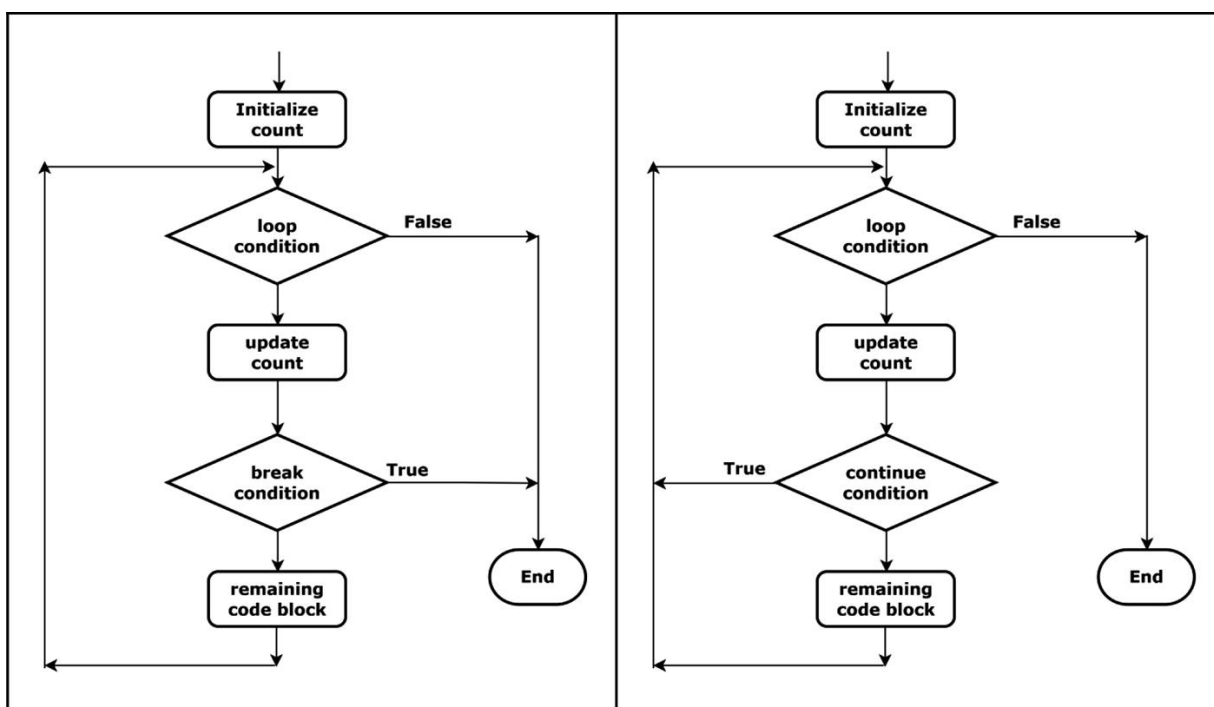


Figure 11-2 break and continue

Functions

When solving complex problems, a program may contain hundreds or thousands of statements. Such programs are difficult to debug and maintain. To address these problems, developers break the process into its logical components. Each subtask that performs a single task is developed as a separate, reusable subprogram called by various terms, most commonly as a function in many programming languages, including JavaScript.

Built-in Functions

JavaScript has predefined global functions, built-in functions, and built-in methods that perform a certain predefined task.

The JavaScript interpreter comes with many global functions (they are not keywords). Some of them are

```
const x = Number("123");  
const y = parseInt("123");
```

```
const z = eval("x+y");
```

JavaScript has many functions of built-in objects, like Number, String, and Array.

```
const num = Number.parseInt("42")
const arr = Array.from("foo");
const val = Math.sqrt(16);
```

JavaScript's built-in methods operate on values of data types.

```
const upper = "Hello".toUpperCase();
const num = (123).toString();
const length = [1,2,3].push(4);
```

Note Functions like `alert()`, `prompt()`, and `confirm()` are global functions when used in a browser environment. They are not defined for use in other JS environments such as Node.js.

In addition, programmers can define user-defined functions to perform tasks specific to their applications.

User-Defined Functions

To use a custom function inside the current script, put a sequence of statements that perform its intended task in a pair of braces, assign it a suitable identifier, and tell the JavaScript engine that it is a function. A function is defined as follows.

```
function name(parameters) {
    // statements
}
```

JavaScript has the `function` keyword for this purpose. Parameters in the parentheses are optional. There can be any number of parameters.

Here is a simple JavaScript function.

```
function sayHello() {  
    document.write("Hello World!");  
}
```

Note that the parentheses in front of the function's name are necessary (even if empty), as they help the JavaScript engine differentiate it from other identifiers like variables.

The statements in the function body are executed only when the function is invoked (or called). Because of its reusability, a function once defined can be called any number of times. Calling a function is simply using it as an expression.

```
function sayHello() { // function definition  
    document.write("Hello World!");  
}
```

```
sayHello(); // function call
```

Calling a function transfers the execution flow to the function body. After executing the statements inside the body, control returns and continues with the next statement.

```
function sayHello() { // function definition  
    document.write("Hello World!<br>");  
}
```

```
document.write("Calling sayHello()<br>");  
sayHello(); // function call  
document.write("Back from sayHello()")
```

The browser should show the following output.

```
Calling sayHello()  
Hello World!
```

Back from sayHello()

You can also declare an anonymous function, assign it to a variable, and call it by the name of the variable.

```
let hello =function () { // function definition
    document.write("Hello World!<br>");
}

hello(); // function call
```

A Function with Parameters

Usually a function is called from the main context of a program, although a function can be called by any other function (including itself, which is called recursion). How do functions interact with each other?

When you define a function, you assume that a set of input data is available to it. The function parameters are variables in its definition representing expected inputs.

```
function myFunction(param1, param2, ..) {
//statements
}
```

To call a function, you must pass the required data to it.

```
myFunction(val1, val2. ..);
```

The actual values passed are the function arguments.

In the following example, the addition function is defined to have two parameters: num1 and num2. It is called by passing two arguments.

```
function addition(num1, num2) { //parameters
    let add = num1+num2;
    document.write(`num1: ${num1} num2: ${num2}
addition: ${add}`);
```

```
}  
addition(10, 20); //arguments
```

Ideally, the number and type of parameters and arguments should match. Although JavaScript does not strictly enforce data types, mismatches can lead to unexpected results.

A Function with a Return

When a function is called (from the main context or another function), the program's control goes to the function body. After executing its statements, the control returns to where it was called. Often, you want the function's result to be consumed in the subsequent flow. So, along with the control, the function should also return its result to the calling context. This is achieved by having a return as the last statement in the function body.

```
function myFunction(param) {  
    //statements to compute the result  
    return result;  
}
```

To consume the returned data, it can be used in an expression (displaying or storing in a variable)

```
let val = myFunction(arg);
```

Let's modify the addition() function and add a return statement. The return value is then utilized in subsequent processing.

```
function addition(num1, num2) {  
    let add = num1+num2;  
    return add;  
}
```

```
let first = addition(10, 20);  
let second = addition(30, 40);
```

```
let result = addition(first, second);
document.write(`Result: ${result}`);
```

The `addition()` function returns the sum of its two parameters. This function is called thrice. The variables `first` and `second` store the return values of two separate calls. They are further used as arguments for a third call. The program should show “Result: 100”.

Arrow Functions

The ES6 standard introduced arrow functions. This new feature helps you create an anonymous function with a shorter syntax.

You already learned how to use a function expression, where a function is assigned to a variable and then called.

```
let addition = function (num1, num2) {
  let add = num1+num2;
  return add;
}
```

The syntax of an arrow function makes the definition quite compact.

```
let var =(parameters) => {
  // function body
};
```

Note that the `function` keyword itself is not used; instead, an arrow (`=>`) appears before the function body. Our addition function is now written as follows.

```
let addition = (num1, num2) => {
  let add = num1+num2;
  return add;
}
```

The syntax becomes even shorter if the function body has a single expression whose value is implicitly returned to the caller.

```
let addition = (num1, num2) => num1+num2;
```

Arrow functions are well-suited for callbacks in DOM manipulation as they provide a concise and readable way to write small, inline functions.

Recursive Functions

Recursion is a powerful yet often enigmatic programming technique to solve a problem that is defined in terms of itself.

```
myFunction {  
  . .  
  myFunction();  
}
```

Obviously, calling the function unconditionally from inside its body causes an infinite loop. Naturally, there must be a way to prevent it.

Calculation of the factorial of a number is often cited as the most popular example of a recursive process. Mathematically, factorial is defined as follows.

$$n! = n \times (n-1)!$$

Note that a factorial is used to define a factorial. This repeated call continues until it reaches 1, which is the base condition.

The definition of a recursive function looks like the following.

```
function recursiveFunction() {  
  if(base condition) {  
    // stop calling itself  
  } else {  
    recursiveFunction();  
  }  
}
```


Every subsequent recursive call must ensure that it is progressing incrementally closer to the base condition, eventually stopping the repetition.

The recursive factorial function should repeatedly call itself by decreasing the value of its parameter till it reaches one.

```
function factorial(n) {  
  if (n==1) //base condition  
    return 1;  
  else  
    return n* factorial (n-1);  
}
```

You can test the preceding function by passing it any number.

```
let x=5;  
f=factorial(x);  
document.write(`${x}! = ${f}`);
```

The browser should display the following.

5! = 120

Binary tree and filesystem traversal, sorting, and binary search algorithms are some of the well-known use cases of recursive functions.

Most of the time, recursion can be replaced by loop constructs. However, iterative solutions may become lengthy and tedious. Recursion often provides a more compact and intuitive solution, although identifying the recursive pattern is not always straightforward.

Points to Remember

- Conditionals, loops, and functions are the core programming constructs of JavaScript.

- Comparison and logical operators play an important role in implementing conditional logic in a program.
- JavaScript provides two main constructs for decision-making: if-else and switch-case.
- A block of statements following an if statement is executed when its Boolean expression evaluates to true. Optionally, an else block can be provided to execute when the condition is false.
- The if-else if-else structure allows multiple mutually exclusive conditions to be evaluated.
- The switch-case construct is useful when different actions are required for discrete values of an expression.
- JavaScript supports various looping constructs. Using while, do-while, and for loops, a set of instructions can be executed repeatedly until a specified condition is met.
- Every loop involves a controlling condition, a block of code to execute, and a mechanism to terminate the loop.
- The while and do-while loops are similar, except that in the while loop the condition is evaluated before executing the loop body. However, in the do-while loop, the condition is evaluated after execution. As a result, the do-while loop guarantees at least one execution of the loop body.
- The for loop provides a compact syntax, as initialization, condition, and update are specified together.
- The for...in loop is used to iterate over the enumerable properties of an object, while the for...of loop (introduced in ES6) is designed for iterating over iterable objects such as arrays and strings.
- The concept of a function is central to the modular approach in programming. A function is a reusable block of code that performs a specific task. A complex problem can be broken down into smaller functional components.
- Once defined, a function can be invoked from other parts of the program. Data is passed between functions using parameters and arguments, and results are returned using the return statement.

- Arrow functions, also introduced in ES6, provide a concise way to write functions and are particularly useful for short, inline callbacks, such as those used in DOM manipulation.
 - Recursion is a powerful yet tricky programming technique in which a function calls itself.
 - A recursive function must include a base condition to terminate the recursion, and each recursive call should move the computation closer to this condition.
-

Summary

This chapter lays a strong foundation of the core programming concepts required for the development of the front as well as the back end of a web application. Starting with the conditional statements (if-else and switch-case), this chapter discusses the various looping constructs with the help of simple yet suitable code snippets. You also learned about the subtle nuances of each looping statement. The modern notation of arrow functions has also been introduced in this chapter. The chapter concluded with the concept of recursive functions.

The next chapter applies these programming concepts to handle events on HTML and CSS elements in a web page with JavaScript functions as callbacks.

12. JavaScript and DOM

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

One of the biggest strengths of JavaScript, especially in the browser environment, is its ability to interact with the elements in a web page. As a document is loaded, the browser constructs an object tree that represents the hierarchy of various HTML elements in it. This tree, called the Document Object Model (DOM), acts as the interface for JavaScript to programmatically read an element's content, even add, modify, or remove HTML elements, and respond to various events occurring in the browser's event system. JavaScript thus makes web applications truly dynamic and interactive.

This chapter explains how to interact with the elements in the DOM with various querying methods and dynamically manipulate their CSS properties. You will also learn to attach event listeners to an element and trigger a JavaScript function in response to user interactions such as clicks, key presses, and other browser events.

What Is the DOM?

The HTML parser performs the important task of constructing the DOM tree. It involves the following steps.

- a. The browser receives raw HTML code from the server.
- b. The parser converts the markup into tokens, such as starting and

ending tags, attributes and values, and the data between them.

- c. Using these tokens, it builds the DOM tree—an in-memory structure of hierarchically arranged node objects.

Consider the following HTML script.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>DOM Tree</title>
</head>
<body>
  <h1>DOM Tree Parser</h1>
  <p>Hello World!</p>
</body>
</html>
```

The browser builds a DOM tree that resembles Figure [12-1](#).

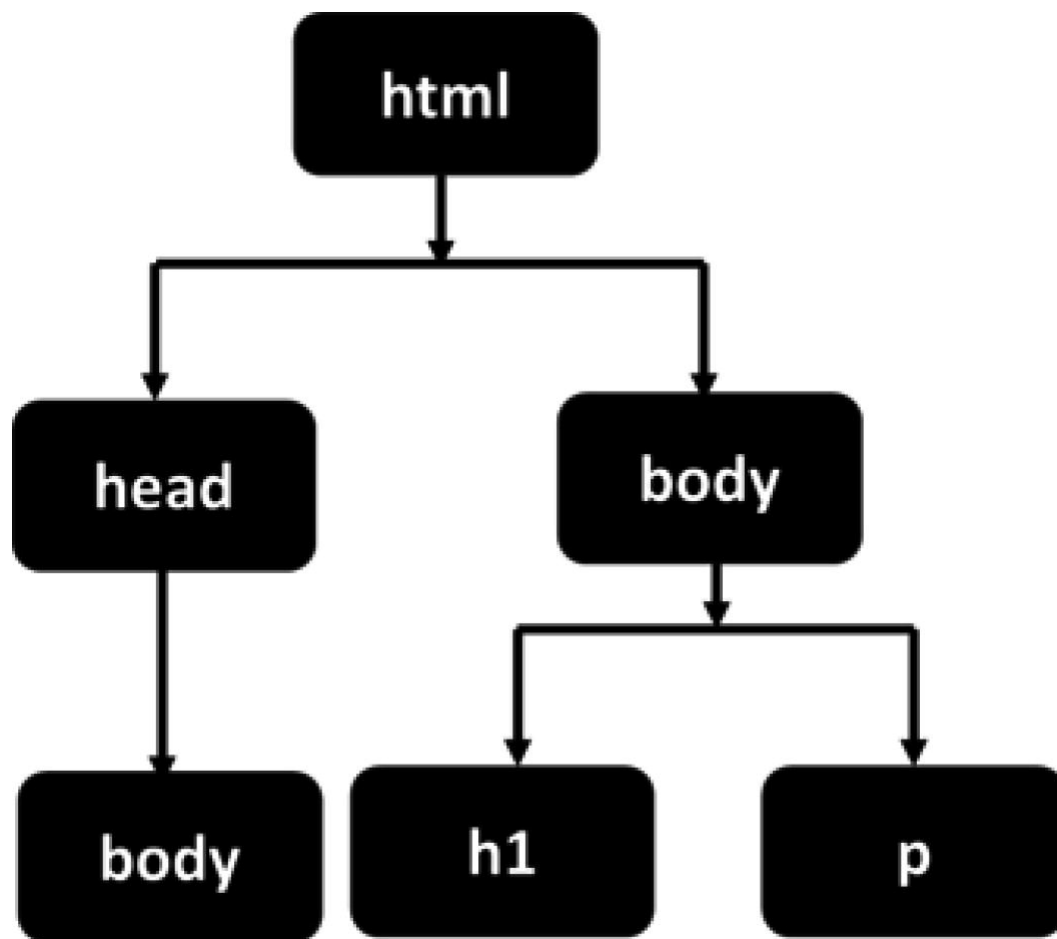


Figure 12-1 DOM tree

Many tools are available to get a live preview of the DOM tree. Modern browsers have various tree visualization extensions (many of them free to use) to choose from. The DOM Tree Visualizer extension available for Google Chrome is widely used. See Figure [12-2](#).

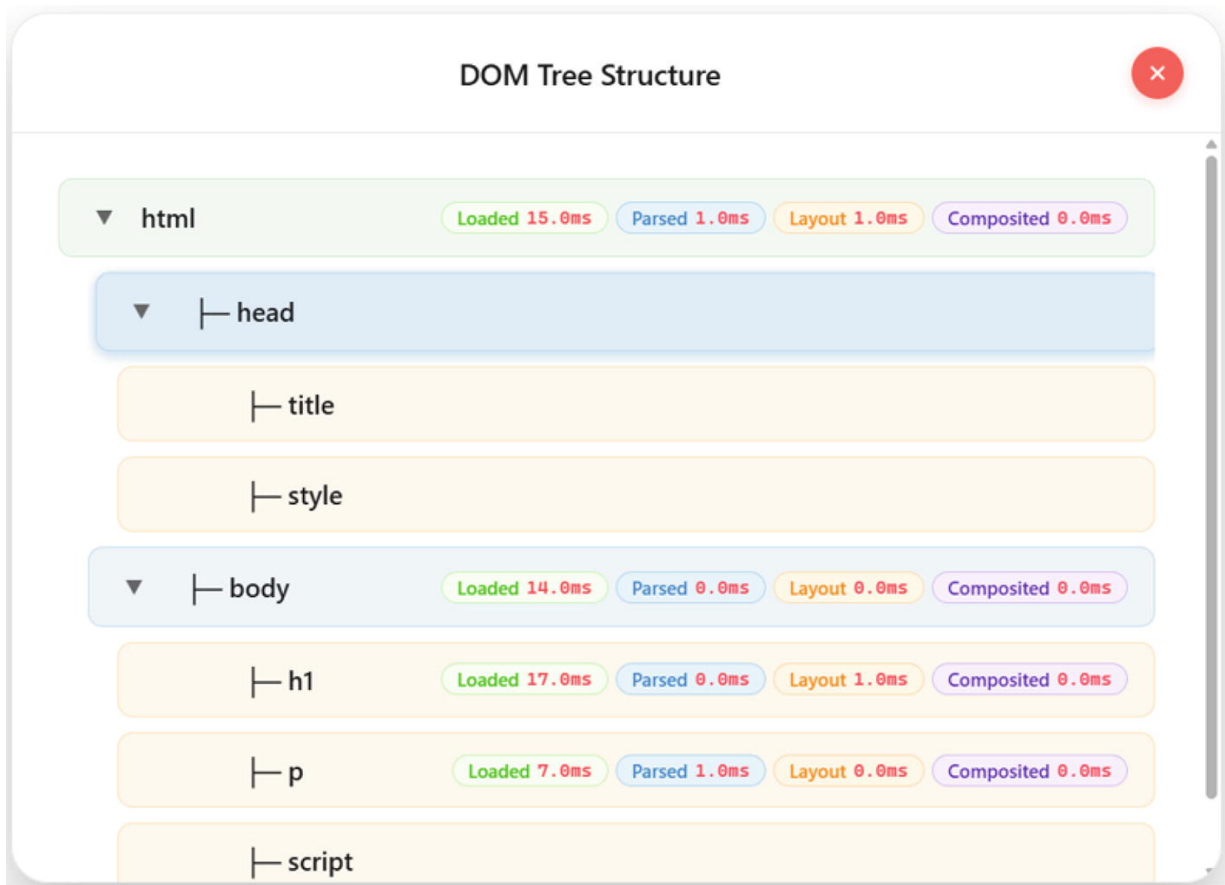


Figure 12-2 DOM Tree Visualizer

You can also inspect the DOM tree from the Elements tab of your browser's developer tools. See Figure [12-3](#).

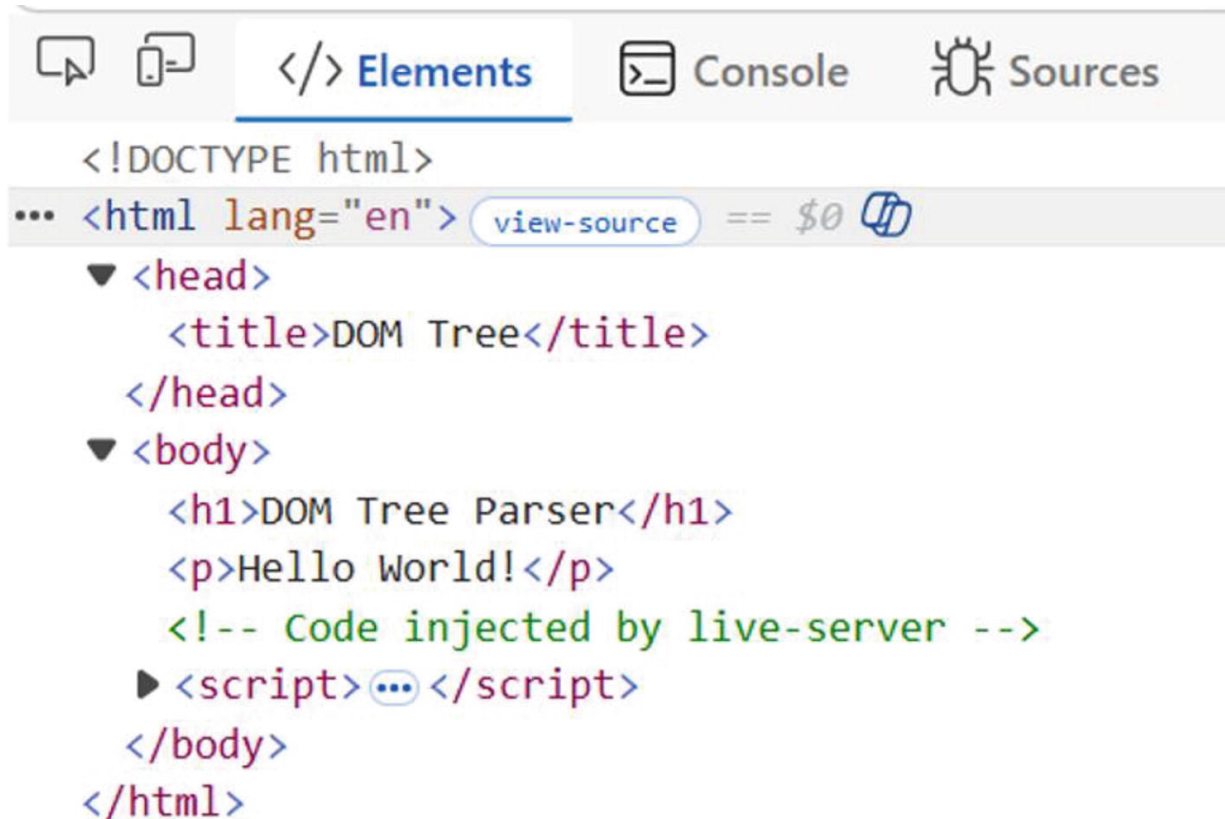


Figure 12-3 DOM tree in developer tools

Document Object

For a browser, the entry point into the web page's content is the Document object. It is the root of the DOM tree. It contains multiple node objects representing HTML elements. The `<html>` element corresponds to the root node of the DOM tree. Nodes having a common parent are called siblings. `<head>` and `<body>` are siblings. A node without any child node is called a *leaf*. The Document object is then handled programmatically to perform actions like find, create, and remove elements as well as handle events.

Document Properties

Several properties and methods characterize the Document object. Many of the properties are read-only, while others can be manipulated

through JavaScript. Let's look at some common properties of the Document object.

document.body

The document.body property represents the <body> node of the current document. It allows you to set attributes of the body element during runtime.

```
const myBody = document.body;
myBody.id="id1";
document.write(myBody.id);
```

document.head

The document.head is a read-only property. It returns the <head> element of the current document. The following statement returns HEAD as the name of the <head> tag.

```
let tag = document.head.tagName;
document.write(tag);
```

document.documentURI

Also a read-only property, document.documentURI returns the document location as a string.

```
document.write(document.documentURI); //
http://127.0.0.1:5500/index.html
```

document.images

The document.images property returns a collection of the images in the current document. Each image represents an element. The property is read-only.

```
<body>


```

```
<script>
for (const image of document.images) {
    console.log(image.src);
}
</script>
</body>
```

The browser's console shows the following output.

```
http://127.0.0.1:5500/bg.png
http://127.0.0.1:5500/logo.jpg
```

document.scripts

A collection of script objects, each representing a `<script>` element in the current HTML document, is returned by the `document.scripts` property. It is a read-only property. You can retrieve data such as the number of script blocks (`length`) and the content of a script element.

This HTML document has two script elements.

```
<script>
let message="Hello World";
</script>
<script>
console.log(document.scripts[0].text);
</script>
```

The browser console shows the contents of the first script element.

```
let message="Hello World";
```

document.styleSheets

The `document.styleSheets` property returns a collection of external `StyleSheet` objects, included in the HTML document with the `<link>` tag. You can retrieve the text of a CSS rule, or add/delete rules from the stylesheet.

```
<head>
  <link rel="stylesheet" href="style.css">
</head>

<body>

<script>
for (const sheet of document.styleSheets) {
  console.log(sheet.cssRules[0].cssText);
}
</script>

</body>
```

You would get the contents of the first rule in the embedded stylesheet.

Document Methods

Out of the extensive list of methods available for the document object, some of the most often used methods help you select the required elements from the DOM tree. These are called querying methods (examples are `getElementById()`, `getElementsByName()`, `querySelector()`, etc.). You also have methods that can dynamically create and remove a new element, a new text node, an attribute, and so forth.

getElementById()

The `getElementById()` method is a very powerful querying method. The global attribute `id` is unique in the given HTML document. Using a string parameter, the browser can find a matching element in the tree. You can then read or manipulate the properties of the selected element, such as fetching the paragraph text, traversing its child nodes, appending or removing one of its children, and more.

```
<body>
  <p id="p1">Hello World </p>
```

```
<script>
let element = document.getElementById("p1");
console.log(element.innerText); // displays Hello
World
</script>
</body>
```

getElementsByTagName()

Note that there may be more than one element of the same tag in a document; for example, many <div> elements. The `getElementsByTagName()` method returns a collection of elements of the given type.

```
document.getElementsByTagName("element")
```

Since this method returns a collection, you can traverse it using any of the looping constructs and process each paragraph element. The following example retrieves the text inside <p> and </p> tags of each p node.

```
<body>
<p>Finding HTML Elements by Tag Name.</p>
<p>This example demonstrates the
<b>getElementsByTagName</b> method.</p>

<script>
const paras = document.getElementsByTagName("p");
for (let i = 0; i < paras.length; i++) {
    console.log(`Paragraph ${i + 1}:
    ${paras[i].textContent}`);
}

</script>
</body>
```

Here's the console output.

Paragraph 1: Finding HTML Elements by Tag Name.

Paragraph 2: This example demonstrates the `getElementsByTagName` method.

getElementsByName()

A document may have multiple elements with a common name attribute. This is especially found in checkbox and radio buttons. The `getElementsByName()` method returns a collection of such elements. You can then iterate over the collection for various purposes, like applying common styling.

In the following example code, there are three checkbox buttons with `sub` as a common name attribute. With the `getElementsByName()` method, the states of each element are retrieved.

```
<body>
  <input type="checkbox" id="phy" name="sub"
value="Physics" checked />
  <label for="phy">Physics</label>

  <input type="checkbox" id="che" name="sub"
value="Chemistry" />
  <label for="che">Chemistry</label>

  <input type="checkbox" id="math" name="sub"
value="Maths" />
  <label for="math">Maths</label>

<script>
btns = document.getElementsByName("sub");
for(i=0;i<btns.length;i++){
  console.log(btns[i].value + " IS " +
(btns[i].checked));
}
```

```
</script>  
</body>
```

Output in the developer tools console.

```
Physics IS true  
Chemistry IS false  
Maths IS false
```

getElementsByClassName()

By defining the class attribute of the same value in multiple related elements, it becomes easier to apply identical styling to all. The `getElementsByClassName()` method of the Document object returns a collection of elements with the given class name as the argument.

```
document.getElementsByClassName (name)
```

You can also specify multiple classes, separated by whitespace, as the argument.

```
document.getElementsByClassName (name1 name2)
```

In this case, the collection of elements whose class attribute is either name or name2.

You might remember that a class-based CSS selector is defined as follows.

```
.name {  
  //CSS rules  
}
```

To manipulate any rules belonging to a given class selector, you need to fetch the elements with this class and then process each element in the collection.

In the following HTML script, some elements have the class attribute of the same value. The `getElementsByClassName()` method

retrieves them in an HTMLCollection. The JavaScript code merely lists each element and its respective text component.

```
<body>

<h1 class="testclass">The Document Heading</h1>

<p>This is the first paragraph.</p>
<div class="testclass">
  <p>This is a div element.</p>
</div>
<br>
<div>
  <p>This is another div element.</p>
</div>
<br>
<section class="testclass">
  <p>This is a section.</p>
</section>

<script>
const collection =
document.getElementsByClassName("testclass");
for (let i = 0; i < collection.length; i++) {
  console.log(collection[i].tagName + ": " +
collection[i].innerText);
}
</script>

</body>
```

The following is the console output.

```
H1: The Document Heading
DIV: This is a div element.
```

SECTION: This is a section.

In addition to DOM API methods like `getElementById()` and `getElementsByClassName()`, you also have `querySelector()` and `querySelectorAll()`, which are more flexible and convenient because they can be used to query the DOM by id, element, or class.

querySelector()

As the name suggests, the `querySelector()` method uses any valid CSS selector (or multiple comma-separated selectors) as the argument, and returns the first matching element from the DOM tree.

```
document.querySelector(CSS-Selector)
```

The matching process starts from the root node and follows a depth-first pre-order traversal. The method returns null if no matching element can be located.

Change the `<script>` element in the HTML code from the previous example.

```
<script>
let element = document.querySelector('p');
console.log(element.tagName + ": " +
element.innerText);
element = document.querySelector('.testclass');
console.log(element.tagName + ": " +
element.innerText);
</script>
```

The call to `querySelector('p')` fetches the first paragraph element; the tree search stops as soon as the first occurrence is met. Similarly, `querySelector('.testclass')` returns the first element with the given class attribute.

querySelectorAll()

To select all the elements that match the given selector (or selectors), you need to use the `querySelectorAll()` method. This method returns the collection of matching elements.

```
<script>
const elements = document.querySelectorAll('p');
let i=0;
for (let element of elements) {
  console.log("Paragraph " + (i+1) + ": " +
    element.innerText);
  i++;
}
</script>
```

The `<script>` element displays the text component of each paragraph in the document.

The difference between this and the `querySelector()` method is obvious— selecting all matching elements against the first.

In addition to the preceding querying methods, some others let you manipulate the DOM tree; for example, you can add/remove elements, comments, and attributes.

createElement()

The `createElement()` method dynamically creates a new HTML element node. You need to give a valid string representation of an HTML tag as an argument. For example, the following statement creates a new paragraph element.

```
const para = document.createElement("p");
```

Note that the element thus created is in memory only, not visible on the web page until it is explicitly appended to the DOM. To add it, use the `appendChild()` or `append()` methods.

```
<script>
```

```
const para = document.createElement("p");
para.innerText = "This is a dynamically created
paragraph.";
document.body.append(para);
</script>
```

remove()

You can also remove an element from the DOM tree. The `remove()` method is called on the element to be removed. Usually, you search the DOM tree with any of the querying methods and run the `remove()` method on the element(s) returned.

```
const elem = document.getElementById('id1');
elem.remove();
```

createAttribute()

`createAttribute()` is another useful document method, with which you can dynamically add an attribute to an element. It returns an instance of the `Attr` object, which can be used as an argument to the required element's `setAttributeNode()` method.

Assuming that the HTML document has a paragraph element as follows.

```
<p id="id1">This is the first paragraph.</p>
```

Let's employ the `createAttribute()` method to add a class attribute and further add it to the `<p>` element.

```
<script>
const atr = document.createAttribute('class');
atr.value = 'myclass';
const elem = document.getElementById('id1');
elem.setAttributeNode(atr);
console.log(elem.outerHTML);
</script>
```

The `outerHTML` property returns the HTML markup of the element.

```
<p id="id1" class="myclass">This is the first  
paragraph.</p>
```

Note that you can also use the `setAttribute()` method. You need to give two arguments (class name and its value) instead of the `Attr` instance.

```
elem.setAttribute('class', 'myclass');
```

createTextNode()

A text node is the actual text inside the opening and closing tags. It actually is treated as the child node of the enclosing element.

```
<p>Hello World</p>
```

Here, `<p>` is the element node and “Hello World” is the text node. The `createTextNode()` method creates a new text node.

```
const text = document.createTextNode("Hello  
World")
```

Obviously, this instance of the `Text` object lives in memory. You need to add it to some element node (existing or new) for it to appear in the browser window.

```
const p = document.createElement("p");  
p.appendChild(text);
```

You can also append it directly to the document body.

```
document.body.appendChild(text);
```

However, text nodes as direct children of the body are not recommended.

HTMLElement Object

The base of the DOM is a node. It includes text nodes, comment nodes, and element nodes. The element may be HTML, SVG, or XML. All HTML elements are instances of HTMLElement. Each HTML tag has a corresponding subclass of HTMLElement.

For example, HTMLDivElement represents the <div> element, HTMLParagraphElement represents the <p> element, and HTMLHeadingElement represents <h1>. Others are HTMLFormElement for <form>, HTMLInputElement for <input>, and so on.

Properties and methods of HTMLElement are available to its subclasses. Additionally, each subclass provides tag-specific properties and methods.

As a result of the hierarchy, all HTML element objects inherit properties from Node, Element, and HTMLElement, along with their specific subclass.

HTMLElement Properties

HTMLElement is the base class for all HTML elements. This section discusses some of its common properties.

id

The global attribute id of any HTML element is the id property. You can programmatically set or retrieve the id of an element.

```
const para = document.querySelector("p");
console.log(para.id);      // retrieve
para.id = "p1"; //set
```

innerText

The innerText property represents the text content of a node and its descendants. You can read its value and assign it a new value.

```
<body>
<p id="id1">This is the first paragraph.</p>
```

```
<script>
let node = document.getElementById("id1");
let text = node.innerText;
console.log(text);
node.innerText = "This is the modified
paragraph.";
console.log(node.innerText);
</script>

</body>
```

innerHTML

The `innerHTML` property gets or sets the HTML markup contained within the element. Instead of a pure text string, a string with HTML tags can be set as the value. Consider the following example.

```
<body>
<p id="id1">This is the first paragraph.</p>

<script>
let node = document.getElementById("id1");
node.innerHTML = "This is the
<strong>modified</strong> paragraph.";
</script>

</body>
```

When the script runs, it actually renders the paragraph element as follows.

This is the **modified** paragraph.

textContent

The `textContent` property sets or gets the raw content of an element, including that of its descendants.

```
<div id="div1">
<p id="id1" style="display:none;">This is a
hidden paragraph.</p>
<p>This is the first paragraph</p>
</div>
```

Here, the `<div>` element has two paragraphs, one of them with the `display` attribute set to `none`. Hence, only the second paragraph is visible in the browser window. But the `textContent` property returns both.

```
<script>
let node = document.getElementById("div1");
console.log(node.textContent)
</script>
```

The console shows the text of both paragraphs.

```
This is a hidden paragraph.
This is the first paragraph
```

attributes

The `attributes` property returns a collection of all attributes defined in a given element. Each object in the collection is an instance of the `Attr` object. You can access the collection with an index or traverse with a `for-of` loop.

```
<body>
<p id="id1" style="color:blue;">This is a
hidden paragraph.</p>

<script>
let node = document.getElementById("id1");
let list = node.attributes;
```

```
for (let atr of list) {  
    console.log(atr.name + ": " + atr.value);  
}  
  
</script>  
  
</body>
```

There are two attributes in the `<p>` element in the preceding code. The attributes collection contains both.

```
id: id1  
style: color:blue;
```

outerHTML

The `outerHTML` property allows you to retrieve and set the HTML markup of the element and its descendants (along with attributes, start and end tags).

Consider the following `<div>` with a paragraph and a JS script element.

```
<div id="div1">  
<p>Hello World</p>  
</div>  
<script>  
const div = document.getElementById("div1");  
console.log(div.outerHTML);  
</script>
```

The `outerHTML` property returns the complete HTML representation of this tag.

```
<div id="div1">  
<p>Hello World</p>  
</div>
```

Now, replace the `<p>` element with a `<h1>` element.

```
<script>
const element = document.getElementById("div1");
element.outerHTML = "<h1 id='h1'>Hello
World</h1>";
const elem = document.getElementById("h1");
console.log(elem.outerHTML);
</script>
```

In the browser window, the text Hello World immediately changes to `<h1>` style, and it also reflects in the console output.

classList

In HTML, class is a global attribute that allows you to apply common styles to various HTML elements. Its value is a string of classes separated by spaces.

Assuming that a document has the following CSS.

```
<style>
.myClass {
  background-color: blue;
}
</style>
```

You can always declare a `<div>` element with its class attribute as follows.

```
<div class="myClass newClass">
<p>Hello World</p>
</div>
```

Note that the individual class in the list may or may not have been defined in the CSS stylesheet.

The `classList` property of the element returns the collection representing the class names included in the class attribute. It is an instance of the `DOMTokenList` object, which you can use to manipulate the class list.

To begin with, the `div` doesn't have a class attribute.

```
<div id="div1">
<p>Hello World</p>
</div>
```

Using JavaScript, we can add a class to this `div`. To do so, fetch its `classList` property and call the `add()` method.

```
const div = document.getElementById("div1");
let list = div.classList
list.add("myClass");
console.log(div.outerHTML);
```

The console shows the updated HTML of the `<div>` element.

```
<div id="div1" class="myClass">
<p>Hello World</p>
</div>
```

Immediately, its background color changes since the `myClass` rule has already been declared (see Figure [12-4](#)).



Hello World

Figure 12-4 Use `classList` property

However, if you add another class not declared (as follows), the browser ignores it silently, although the `div` HTML will be updated.

```
list.add("newClass");
```

The `toggle()` method adds the class if it is not included in the list and removes it if it is present.

```
list.toggle("newClass");
```

You can also include multiple classes at a time by putting a comma-separated list of class strings as arguments to the `add()` method.

```
list.add("foo", "bar");
```

You can replace one class with the other. Use the `replace()` method.

```
list.replace("foo", "baz");
```

Finally, the `remove()` method removes a class from the list.

```
list.remove("baz");
```

DOM Traversal

The next set of properties facilitates the traversal of the DOM structure of an HTML document. After locating a desired node with an appropriate querying method, you can find its parent node, its siblings, and children.

children

The `children` property returns a collection of all such elements whose direct parent is the given element. In other words, the descendant nodes of the next level are not included in the collection.

In this HTML code, there's a `<div>` element with a paragraph and another `<div>` which in turn has another paragraph.

```
<div id="div1">
<p>The main paragraph</p>
  <div>
    <p>The paragraph in the child div</p>
  </div>
```

</div>

The children collection of the upper <div> won't include the second paragraph.

```
<script>
const collection =
document.getElementById("div1").children;
let text = "";
for (let i = 0; i < collection.length; i++) {
    document.write(`Child tag:
${collection[i].tagName} <br>`);
}
</script>
```

The browser output confirms this behavior.

```
Child tag: P
Child tag: DIV
```

firstElementChild

The firstElementChild read-only property returns the first child of an element, the same as the 0th element in the children collection.

lastElementChild

Similarly, the lastElementChild property is also read-only and returns the last child element. It corresponds to the element at the position with its index as (length-1).

parentElement

The parentElement property returns its immediate parent (not any of the grandparent elements), that is, at one level above the specified element.

nextElementSibling

The `nextElementSibling` property is read-only, and returns the next element of its parent.

previousElementSibling

Similarly, the `previousElementSibling` property is also read-only, and returns the previous element of its parent.

The following example demonstrates all these properties. An ordered list of subjects is included in the following script.

```
<ul id="Subjects">
  <li id="s1">Physics</li>
  <li id="s2">Chemistry</li>
  <li id="s3">Maths</li>
</ul>
```

The children collection of the `<ul>` element at the top of the hierarchy is obtained. Its first and last elements are then retrieved. To show the sibling properties, we get the `<li>` element with `id="s2"`.

```
<script>
const subjects =
document.getElementById("Subjects");

//children
console.log(`No. of child nodes:
${subjects.children.length}`);

//firstElementChild
console.log(`First:
${subjects.firstElementChild.innerText}`);

//lastElementChild
console.log(`Last:
${subjects.lastElementChild.innerText}`);

const second = document.getElementById("s2");
```

```
// parentElement
console.log(`Parent of second:
${second.parentElement}`);

// nextElementSibling
console.log(`Next sibling:
${second.nextElementSibling.innerText}`);

// previousElementSibling
console.log(`Previous sibling:
${second.previousElementSibling.innerText}`);
</script>
```

The output is as follows.

```
No. of child nodes: 3
First: Physics
Last: Maths
Parent of second: [object HTMLUListElement]
Next sibling: Maths
Previous sibling: Physics
```

DOM Events

As you have learned, JavaScript has all the features of a powerful multipurpose programming language. However, its true strength, particularly in the context of web development, lies in its ability to interact with the DOM through event handling.

In a browser environment, an event is any action, typically triggered by the user—such as clicking a button, hovering over an element, selecting an item from a drop-down, or entering text into an input field. It could also be a browser-induced event, like page load, window resize, or so forth.

Each such action is said to trigger an event. In response to these actions, the browser creates the corresponding Event object

(mouseEvent when the mouse is clicked, keyboardEvent when a key is pressed or released, etc.) and makes it available to the JavaScript engine for processing. You can ask the JavaScript engine to listen to a specific event and execute certain instructions in response.

How does JavaScript handle events? There are three primary ways to handle DOM events.

- Using event handler properties: These are properties like onclick, oninput, assigned directly to DOM elements.
- Using addEventListener(): This method attaches one or more event handler functions to an element.
- Using inline event handlers

Event Handler Properties

Since they are inherited from a common DOM interface, DOM objects such as document or HTML elements like button and div are defined with event handler properties. These properties (named “on” followed by the event type, such as click, submit, etc.) invoke the JavaScript function assigned to them when the corresponding event occurs on the specified object.

```
<element onevent="function"> .. </element>
```

The following is an example.

```
<button onclick="alert('Hello  
World') ">Click</button>
```

Some of the common event handler properties are onclick, onmouseover, onmouseout, onchange, and onsubmit. Typically, the functionality of an element determines which properties to define. The following are examples.

- onclick is useful with buttons and other clickable elements.
- oninput and onfocus are useful with text-type input fields.

- onmouseover and onmouseout are generally used with container elements.
- onsubmit is useful with forms.
- onload and onresize are useful with document and window elements.

The onclick Property

The most frequent mode of user interaction with a GUI is clicking a button. The onclick property enables an element to capture the click event. Its value is typically a JavaScript function (either built-in or user-defined) to be executed in response.

The preceding usage of the onclick property employed the built-in alert() function. You can also assign a user-defined function as its value.

```
<button onclick="myFunction()">Click</button>
```

```
<script>
function myFunction() {
    alert("Hello World");
}
</script>
```

Instead of having the onclick property inside the HTML, you can define it as the property of the corresponding element. To change the preceding code accordingly, add an id property to the button, obtain the HTMLButtonElement object, and set its onclick property to myFunction.

```
<button id='btn'>Click</button>
```

```
<script>
function myFunction() {
    alert("Hello World");
}
```

```
const btn=document.getElementById("btn");
```

```
btn.onclick = myFunction;  
</script>
```

Or, you can even employ an arrow function.

```
const btn=document.getElementById("btn");  
btn.onclick = () => { alert("Hello World"); }
```

There are many other use cases where the onclick property is employed, such as hiding/showing content on click or dynamically adding/removing items in a menu.

Mouse Movement Properties

While the click event is generated when the user actually “clicks” an element with a mouse, the movement-related events are triggered when the mouse pointer enters/leaves the bounding rectangle of an element. These are mouseout, mouseover, mouseenter, and mouseleave events.

The mouseenter/mouseleave events are triggered when the pointer enters or leaves an element’s boundaries.

The mouseover/mouseout events, though similar to the two previous events, support event bubbling—a mechanism in which an event triggered on a target element propagates upward through its parent elements.

The corresponding event handler properties are onmouseenter, onmouseleave, onmouseover, and onmouseout.

Here’s a simple example that demonstrates the onmouseover and onmouseout properties. The following HTML+CSS code renders two buttons.

```
.btn {  
width:200px;  
height:100px;  
background:lightblue;  
}
```



```
<button id="b1" class="btn">
  Move mouse here
</button>
<button id="b2" class="btn">
  Move mouse here
</button>
```

A JavaScript function changes the background color of the first button as the mouse pointer enters its boundary and restores it when the mouse pointer goes out.

```
const btns =
document.getElementsByClassName("btn");

btns[0].onmouseover = function () {
  btns[0].style.background = "orange";
};

btns[0].onmouseout = function () {
  btns[0].style.background = "lightblue";
};
```

The browser screenshot in Figure [12-5](#) shows the changed color of the first button.

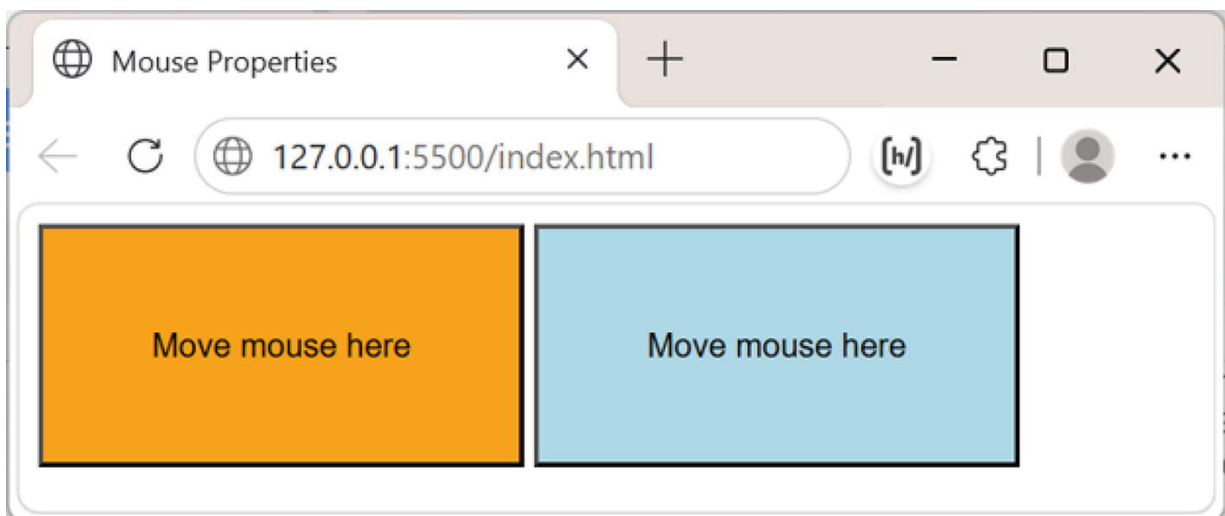


Figure 12-5 Mouse events

The other event related to mouse movement is `mousemove`, which occurs when the pointer moves over an element, whether or not any mouse buttons are pressed. The corresponding handler property therefore is `onmousemove`.

Keyboard Event Properties

Any user interaction with the keyboard device triggers the `KeyboardEvent`. Specifically, the `keydown` event is fired when a key is pressed, and the `keyup` event when released. However, these events can occur on any element (not just the input fields) that has keyboard focus, as well as on the document or window object.

The input event is more suitable for working with the fields whose value changes by the user. It occurs when the value of an `<input>`, `<textarea>`, or `<select>` element is changed. The `oninput` event occurs when an element gets input. The `oninput` event handler property allows you to process the input event.

The following HTML has a password field and an empty `<div>` element.

```
<input type="password" id="txt" placeholder="Type
here...">
<div id="msg"></div>
```

To ensure that the password consists of alphabets and digits (the allowed set of characters), we use the `oninput` property of the input element.

```
const allowed =
"abcdefghijklmnopqrstuvwxyz0123456789";

const txt = document.getElementById("txt");
txt.oninput = myfunction;
```

The event handler function checks whether each character in the password field is present in the allowed set. If not, the initially empty `<div>` element is set to a warning message.

```
function myfunction () {  
    const value = txt.value;  
    const msg = document.getElementById("msg");  
  
    for (let i = 0; i < value.length; i++) {  
        const ch = value[i].toLowerCase();  
  
        if (!allowed.includes(ch)) {  
            msg.innerHTML = "Only alphabets and digits  
allowed";  
            return;  
        }  
    }  
  
    msg.innerHTML = "";  
};
```

Figure [12-6](#) shows the sample output.



Figure 12-6 The oninput event

The onsubmit Property

Along with the other form controls, an HTML form has a submit button. When clicked, the browser collects the form data and submits it to the action URL. The submit event is triggered just before the data is submitted. It usually acts as a custom validator. If you want to stop the data from being submitted, you need to call the `preventDefault()` method.

```
form.onsubmit = function (e) {  
  if (condition) {  
    //validation check fails  
    e.preventDefault();  
  }  
}
```

The following is a simple example to help you understand how an `onsubmit` event works. The form has a single text input field.

```
<form id="form">  
<label for="name">Enter name: </label>  
  <input id="name" type="text" name="name">  
  <input type="submit" value="Submit">  
</form>  
  
<script>  
const form = document.getElementById("form");  
const name=document.getElementById("name");  
  
form.onsubmit = function (e) {  
  if (name.value=="") {  
    alert("name is empty");  
    e.preventDefault();  
  }  
}  
</script>
```

The JavaScript code assigns a function to the `onsubmit` property of the form object. When the submit button is clicked, the submission is prevented if the text field is empty, making an alert box pop up.

Event Listener Methods

Using the `onevent` properties is a fairly convenient way to respond to events triggered by HTML elements. However, modern JavaScript recommends employing event listeners for greater flexibility. The `addEventListener()` method registers a handler function to handle an event. You can attach multiple handlers to an event, which handler properties are unable to do. The `removeEventListener()` method de-registers the handler.

addEventListener

The `addEventListener()` method adds a function to the list of listeners attached to the given element. Hence, when the event is fired, all the handler functions in the list are called.

The method follows the following syntax.

```
element.addEventListener(event, function,
useCapture);
```

The first parameter is a DOM event, such as `click`, `mouseover`, or `keydown`. (Note that it is not `onclick` or `onmouseover` because these are the event handler properties.) The second parameter is the handler function you want called in response to the event. It can be any JavaScript function: built-in or user-defined. The third parameter, `useCapture`, is optional. Its value is `true/false`. If `true`, the event capture will take place.

Earlier in this chapter, you used the `onclick` property to call an `alert()` function in response to the click event of a button. Let's rewrite the same example by using the `addEventListener()` method.

The button HTML is simple.

```
<button id="b1">Click</button>
```

To add a function as the handler, obtain the `HTMLButtonElement` object and use its `addEventListener()` method.

```
const btn1=document.getElementById("b1");
btn1.addEventListener("click", myFunction);

function myFunction() {
    alert("Hello World");
}
```

What happens if there are multiple handlers? Let's verify it. Define two functions and add both as the listeners. Each function concatenates a string to the existing innerHTML of a `<div>` element.

```
<button id="b1">Click</button>
<div id="div"></div>

<script>
const btn1=document.getElementById("b1");
btn1.addEventListener("click", myFunction);
const div=document.getElementById("div");

function myFunction() {
    div.innerHTML+="Hello World<br>";
}

btn1.addEventListener("click", newFunction);
function newFunction() {
    div.innerHTML+="Hello World!<br>";
}

</script>
```

removeEventListener

As the name suggests, this method removes a function from the list of handlers attached to an event.

This method takes the same parameters as in the `addEventListener()` method.

```
element.removeEventListener(event, function,  
useCapture);
```

To check out how `removeEventListener` works, let's add another button to the document.

```
<button id="b2">Click</button>
```

Now there are two buttons on the page; one has two handlers attached.

Next, register another function that handles the click event on `b2`. Inside this function, let's call `removeEventListener` on the `btn1` object.

```
const btn2 = document.getElementById("b2");  
btn2.addEventListener("click", anotherFunction);  
function anotherFunction() {  
    btn1.removeEventListener("click", myFunction);  
    div.innerHTML="";  
}
```

As a result, when a user clicks the first button, the `<div>` element shows two strings as before. Afterward, when the second button is clicked, one handler from the list is removed, and the `div` content is reset as well. Now, when the first button is clicked again, only one string will be rendered.

Points to Remember

- The browser's HTML parser constructs a tree of various HTML elements in the web page.
- The ability to interact with the DOM tree is the biggest strength of JavaScript.

- The Document object is the root of the DOM tree. The browser exposes this object to JavaScript.
- Of the different properties of the Document object, some are read-only, while others can be manipulated by JavaScript code.
- The querying methods of the Document object help you in selecting one or more elements for further processing.
- To select an element by its id attribute, use the `getElementById()` method.
- Other `getElement...` methods are available for selection based on `tagName`, `className`, and so on.
- Modern JavaScript introduced `querySelector()` and `querySelectorAll()` methods. In these methods, the selection is done by CSS selectors.
- Other document methods like `createElement` and `createAttribute` are also very useful.
- Different HTML elements are instances of corresponding subclasses of `HTMLElement` objects.
- For example, the `<div>` element is the instance of `HTMLDivElement`, `<h1>` of `HTMLHeadingElement`, and so on.
- You can use properties belonging to `HTMLElement` to dynamically manipulate the content of a web page.
- The `id` property is extremely important as querying the DOM with `id` is most common.
- Similarly, the `innerHTML` property is also frequently used to programmatically modify the content of an element.
- The `Element` object provides special properties that make the DOM tree traversal possible.
- The `parentElement` property of an element points to the element inside which it is placed. Similarly, you can navigate to the `nextElementSibling` and the `previousElementSibling`.
- The real power of JavaScript in web development lies in its event handling ability.
- In JavaScript, an event is triggered by an action performed by the user on an HTML element. For example, a button click, entering data

in input elements, and so on.

- Event handler properties are named “on” followed by the name of a DOM event; for example, onclick, onmouseover, onsubmit.
 - These properties can be used as the element’s attributes. The value of an event handler property is a JavaScript function. It is called whenever the respective event is triggered.
 - Instead of handling events with these properties, the preferred approach is to attach a handler function to an event using the `addEventListener()` method.
 - The `removeEventListener()` method de-registers a function from the list of listeners.
-

Summary

This chapter not only has the highest weighting in your LPI Web Development Essentials certification exam; it will certainly play an important role in your journey as a web developer. It covers the all-important concept of event handling. In this chapter, you learned the DOM tree structure, the document object and its properties as well as methods. Querying the DOM tree is central to event handling. We explored different querying methods. Event handler properties and event listeners were discussed in detail, with simple examples.

With this, we reach an important milestone. So far, the chapters have covered the front-end technologies: HTML, CSS, and JavaScript. The last section of the book—starting with the next chapter—is about server-side programming with Node.js.

Knowledge Check: Part IV

1. **Which HTML tag is used to embed or reference executable JavaScript code within a web page?**
 - A. `<javascript>`
 - B. `<script>`
 - C. `<js>`
 - D. `<code src="...">`
2. **Which attribute of the script tag is used to specify the path to an external JavaScript file?**
 - A. `href`
 - B. `link`
 - C. `src`
 - D. `path`
3. **In JavaScript, which of the following is true regarding case sensitivity?**
 - A. Only variable names are case-sensitive.
 - B. JavaScript is entirely case-insensitive.
 - C. Variable names, keywords, and function names are all case-sensitive.
 - D. Only strings are case-sensitive.

Only strings are case-sensitive.

4. **What will the following code output to the console:
`console.log(typeof true);?`**
- A. "string"
 - B. "boolean"
 - C. "undefined"
 - D. "object"
5. **Which keyword should be used to declare a variable that is intended to be reassigned later in the program?**
- A. const
 - B. fixed
 - C. let
 - D. static
6. **How do you access the third element in an array named `fruits`?**
- A. `fruits[2]`
 - B. `fruits[3]`
 - C. `fruits.3`
 - D. `fruits(2)`

7. **Which of the following correctly defines a JavaScript object?**

- A. `let user = ["name", "Alice"];`
- B. `let user = {name: "Alice", age: 25};`
- C. `let user = (name: "Alice");`
- D. `let user = "name": "Alice";`

8. **What is the result of the expression `10 + 20 + "5"`?**

- A. `"305"`
- B. `35`
- C. `"10205"`
- D. `NaN`

9. **What is the value of a variable that has been declared but not yet assigned a value?**

- A. `null`
- B. `0`
- C. `undefined`
- D. `NaN`

Which array method removes the last element from an array?

10.

- A. `shift()`
- B. `push()`
- C. `pop()`
- D. `splice()`

11.

What is the difference between the `==` and `===` operators?

- A. `==` checks for value and type; `===` only checks for value.
- B. `==` only checks for value; `===` checks for both value and type.
- C. `==` is for assignment; `===` is for comparison.
- D. There is no difference.

12.

Which logical operator returns true if **at least one of the operands is true?**

- A. `&&`
- B. `!`
- C. `||`
- D. `??`

13. **What is the correct way to write an else if statement?**

- A. `elseif (x == 10)`

- B. else if (x == 10)
- C. if else (x == 10)
- D. else (x == 10)

14. **Which part of a for loop is used to update the counter variable?**

- A. Initialization
- B. Condition
- C. Increment/Decrement expression
- D. Statement block

15. **How many times will a while (false) { ... } loop execute?**

- A. Infinite times
- B. 1 time
- C. 0 times
- D. Until the browser crashes

16. **In a function definition, what are the names listed in the parentheses called?**

- A. Arguments
- B. Parameters

- C. Variables
- D. Attributes

17. **What does the return keyword do in a function?**

- A. Restarts the function from the beginning
- B. Prints a value to the console
- C. Stops function execution and specifies the value to be sent back to the caller
- D. Deletes the function from memory

18. **Which statement is used to handle different conditions based on the value of a single expression?**

- A. switch
- B. select
- C. case
- D. for

19. **What will be the result of Boolean("") (an empty string)?**

- A. true
- B. false
- C. undefined

D.
NaN

20. **How do you call (invoke) a function named displayMessage?**

A.
call displayMessage();

B.
displayMessage;

C.
displayMessage();

D.
run displayMessage;

21. **Which method would you use to select all <p> elements in a document?**

A.
document.getElementById("p")

B.
document.getElementsByTagName("p")

C.
document.getP()

D.
document.select("p")

22. **How can you change the HTML content (including tags) inside a div element?**

A.
.textContent

B.
.innerText

C.
.innerHTML

D.
.content

23. **Which property is used to change the visibility of an element through the style object?**
- A. `element.style.display`
 - B. `element.style.view`
 - C. `element.style.hidden`
 - D. `element.style.show`
24. **What is the correct syntax to add a click event listener to a button variable named btn?**
- A. `btn.listen("click", myFunction);`
 - B. `btn.addEventListener("click", myFunction);`
 - C. `btn.on("click", myFunction);`
 - D. `btn.setEvent("click", myFunction);`
25. **Which JavaScript method is used to remove an existing element from the Document Object Model (DOM)?**
- A. `element.delete()`
 - B. `element.remove()`
 - C. `document.removeChild(element)`
 - D. `element.hide()`

Answers with Explanations

1. **B**
Explanation: The <script> tag is the standard HTML element used to include JavaScript, either directly between the tags or by referencing an external file using the src attribute. (Chapter [10](#))
2. **C**
Explanation: Similar to the tag, the <script> tag uses the src (source) attribute to point to the location of an external .js file. (Chapter [10](#))
3. **C**
Explanation: JavaScript is a case-sensitive language. This means that myVariable, MyVariable, and MYVARIABLE are treated as three distinct identifiers. (Chapter [10](#))
4. **B**
Explanation: The value true is a primitive Boolean literal. The typeof operator returns the string "boolean" for this type. (Chapter [10](#))
5. **C**
Explanation: let allows for the declaration of variables that can be updated or reassigned. const is used for values that should remain constant. (Chapter [10](#))
6. **A**
Explanation: JavaScript arrays are zero-indexed, meaning the first element is at index 0, the second at index 1, and the third at index 2. (Chapter [10](#))
7. **B**
Explanation: Objects are defined using curly braces {} containing key-value pairs separated by colons. (Chapter [10](#))

8.

A

Explanation: JavaScript evaluates from left to right. First, $10 + 20$ results in 30 (number). Then, $30 + "5"$ triggers string concatenation, resulting in the string "305". (Chapter [10](#))

9.

C

Explanation: When a variable is declared (e.g., `let x;`) but not initialized, JavaScript automatically assigns it the value `undefined`. (Chapter [10](#))

10.

C

Explanation: The `pop()` method removes the last element and returns it. `shift()` removes the first element, and `push()` adds to the end. (Chapter [10](#))

11.

B

Explanation: `==` performs type coercion (converting types to match), while `===` (strict equality) requires both the value and the data type to be identical. (Chapter [10](#))

12.

C

Explanation: The `||` (OR) operator returns true if either the left side, the right side, or both are true. (Chapter [10](#))

13.

B

Explanation: JavaScript uses two separate words: `else` followed by `if` and the condition in parentheses. (Chapter [10](#))

14.

C

Explanation: A standard for loop looks like `for(init; condition; increment)`. The third part updates the variable after each loop iteration. (Chapter [11](#))

15. **C**

*Explanation: A while loop checks the condition *before* executing the block. Since the condition is false, the code block is never*

the block. Since the condition is false, the code block is never entered. (Chapter [11](#))

16.

B

Explanation: Names listed in the function definition are parameters. The actual values passed to the function when it is called are arguments. (Chapter [11](#))

17.

C

Explanation: return exits the function immediately and provides a resulting value to the code that called the function. (Chapter [11](#))

18.

A

Explanation: The switch statement evaluates an expression and matches its value against various case clauses. (Chapter [11](#))

19.

B

Explanation: In JavaScript, an empty string "" is considered "falsey," so converting it to a Boolean results in false. (Chapter [10](#))

20.

C

Explanation: A function is invoked by using its name followed by parentheses (). (Chapter [11](#))

21.

B

Explanation: getElementByTagName returns a collection of all elements with the specified tag name (in this case, all paragraph tags). (Chapter [12](#))

22.

C

Explanation: innerHTML allows you to get or set the HTML markup contained within an element. textContent only handles plain text. (Chapter [12](#))

23. **A**

Explanation: Setting element.style.display = "none"; hides an

element, while `element.style.display = "block";` (or other values) makes it visible. (Chapter [12](#))

24.

B

Explanation: `addEventListener` is the modern and preferred way to attach event handlers to DOM elements. (Chapter [12](#))

25.

B

Explanation: The `remove()` method removes the element it is called on directly from the DOM tree. The `removeChild()` method is called on the parent node of the element you wish to delete. (Chapter [12](#))

Part V

Server Programming with Node.js

Server Programming with Node.js

The preceding chapters in this book (especially Part II, Part III, and Part IV) discussed the important frontend web technologies—namely HTML, CSS, and JavaScript. This part, on the other hand, is dedicated to developing the back end of a web application. **Topic 035 NodeJS Server Programming** in the objectives of the Web Development Essentials Exam (Exam Code: 030-100) outlines three major server-side technologies: Node.js, the Express.js framework, and the SQLite database.

Topic 035 is worth 8 weightage points in the examination scheme, further distributed in its subtopics:

- 035.1 NodeJS Basics (weight: 1)
- 035.2 NodeJS Express Basics (weight: 4)
- 035.3 SQL Basics (weight: 3)

The three chapters in this part correspond to these subtopics.

Chapter [13](#)

Chapter [13](#) introduces Node.js, Google's V8 JavaScript runtime environment, as a server-side technology. Starting with installation, this chapter discusses the concepts of callbacks in the asynchronous processing model of Node.js. You also learn the functionality of its http module and build a basic server-side Node-powered web application.

Chapter [14](#)

Express is a minimal web application framework. In Chapter [14](#), you learn the basic functionality of Express, including how the routing mechanism works and the way it serves the EJS templates. This chapter also explains the concept of middleware and static assets.

Chapter [15](#)

The final chapter in this book covers an extremely important aspect: using databases (particularly SQLite) as a means of persistent data storage. You learn the RDBMS and SQL basics, and then move to running SQL commands in the SQLite command-line interface. Next, you learn to install and use the `sqlite3` module to perform the CRUD operations in the Node.js code and build persistent REST routes for an Express app.

As with the other parts, a set of 25 MCQs is included to help you better prepare for your examination.

13. Node.js Basics

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

The arrival of Node.js has changed the landscape of server-side programming for web applications. Because it uses the V8 JavaScript engine, Node.js lets you use a single programming language for both server-side and client-side development. This chapter explains the core concepts of Node.js, such as asynchronous I/O and its event-driven architecture. You will also learn how to run a web application on a local server.

In 2008, Google developed V8, a JavaScript engine for its Chrome browser. It can be used as a JavaScript runtime environment outside of a web browser. Not only Chrome, but many other modern browsers (such as Edge and Opera) also employ the V8 engine for its high-performance JavaScript execution.

In 2009, Ryan Dahl used V8 to develop Node.js, which can perform non-blocking, event-driven processing. Node.js is cross-platform and open source. Its latest stable release is 25.6.1 (February 2026), though the 24.15.0 LTS (long-term support) release is recommended.

You know how to enable JavaScript in your browser's settings for handling the client-side processing. However, for the back end, you must install Node.js on the server. You can download pre-built binaries for your operating system (Windows, macOS, or Linux) and architecture (X64, x86, or ARM) from the official downloads page (<https://nodejs.org/en/download>).

Installation

To install Node.js on Windows, download and run node-v24.15.0-x64.msi. The wizard-based installation is very straightforward. On Ubuntu Linux, the easiest way to install is using the Ubuntu package manager.

```
sudo apt update  
sudo apt install nodejs
```

The pre-built binaries of Node.js on its official website come with npm (Node Package Manager), so it doesn't need to be installed separately.

To check if Node.js and npm have been properly installed, use the following commands from the terminal.

```
C:\Users\user>node -v  
C:\Users\user>npm -v
```

Node.js vs. JavaScript in a Browser

Though Node.js is a server-side technology, it is not a separate programming language in the traditional sense. At its core, it is basically the same JavaScript that runs inside a browser, but with some distinct differences. The basic building blocks of JavaScript, such as objects, arrays, and functions, are the same for both. However, inside a browser, JavaScript has access to the DOM and the window object, which Node.js doesn't have. On the other hand, Node.js can process the local filesystem, access the OS functionality, and network API; client-side JavaScript can't. The main differentiating features of Node.js are non-blocking I/O and an event-driven architecture. These concepts are discussed later in this chapter.

With Node.js, you can execute the JavaScript code in two ways: using an interactive REPL shell and running the JavaScript code in a file with the .js extension.

REPL Terminal

Many programming languages, database products, and operating systems provide an interactive environment, such as PowerShell in Windows, the Linux terminal, and the Python shell. Node.js has an built-in feature that works on the principle of REPL (Read, Evaluate, Print, Loop). When invoked, the Node.js REPL terminal shows a prompt, where you can issue a single JavaScript instruction. The shell reads the instruction, evaluates it, prints the response, and renders the prompt again. These steps are performed in a loop.

Just enter `node` in your OS terminal to start the Node.js REPL. The `>` prompt appears; type a simple JS arithmetic expression.

```
$ node
Welcome to Node.js v24.15.0.
Type ".help" for more information.
> 10+20
30
```

You can try the `console.log()` statement you used earlier.

```
> console.log("Hello World");
Hello World
```

You can even enter multiline code such as a function definition.

```
> function hello() {
... console.log("Hello World");
... }
```

If you need to write longer code, you can invoke `.editor` mode, enter as many statements, and exit editor mode with `Ctrl+D` to finish, or `Ctrl+C` to cancel.

```
> .editor
```

```
// Entering editor mode (Ctrl+D to finish, Ctrl+C to cancel)
function hello() {
  console.log("Hello World");
}
```

To abort the REPL mode, press Ctrl+D or enter the `.exit` command.

Script Mode

While the REPL terminal is a useful learning tool in the beginning stage, as you can quickly test short snippets of code, the more conventional way is to use the scripting mode. Use any text editor available in your operating system (such as Notepad or Vim; an IDE such as VS Code is recommended), enter your Node.js code, save it with the `.js` extension, and run it from the command line.

Open a new text file and save it as `addition.js` after entering the following code.

```
let num1=10;
let num2=20;

let addition=num1+num2;

console.log(`num1: ${num1} num2: ${num2} addition:
${addition}`);
```

Then run it from the command line as follows.

```
$ node addition.js
num1: 10 num2: 20 addition: 30
```

You can also pass arguments to the script from the command line. The `process.argv` property stores the arguments as an array. (Note the use of the `parseInt()` function!).

```
let num1=parseInt(process.argv[2]);
let num2=parseInt(process.argv[3]);

let addition=num1+num2;

console.log(`num1: ${num1} num2: ${num2} addition:
${addition}`);
```

The script now reads the values from the command line arguments.

```
$ node addition.js 10 20
num1: 10 num2: 20 addition: 30
```

Node.js Modules

Programming elements defined in one .js file can be called in another. Such a reusable script is called a *module*. A Node.js module is a library of functions, classes, and other reusable code, stored in one or more .js files. The pre-built Node.js binary is bundled with many core modules, which you can use in your code. The `require()` function at the top loads a given module.

The following are the core modules that are often used.

- **fs module** defines functions that can access the file system.
- **os module** defines functions for performing platform-specific actions.
- **url module** is used for parsing and formatting URLs.
- **events module** is at the center of the event-driven architecture of Node.js.
- **http module** is important in web application development.

Use `require()` to load a module and then call the functions defined in it. Here, you load the `fs` module and use its `readFile()` function to fetch the contents of a file.

```
const fs = require('fs');
```

```
fs.readFile('myfile.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log('File content:', data);
});
```

Here, the read operation is a callback-based asynchronous operation. Modern JavaScript provides a more efficient promise-based alternative.

In general, any .js file can be imported provided it is qualified with the appropriate path. For example, to import hello.js in index.js (both in the same folder).

```
const hello=require('./hello');
```

If hello.js has a function defined in it, it must export it for it to be used.

```
/*Hello.js*/
function hello() {
  console.log("Hello World");
}

module.exports = {
  hello
};
```

You can now call hello() from inside the index.js code.

```
hello.hello();
```

Apart from the core modules and local .js files, you can install and use any required package from the npm Registry, which hosts thousands of JavaScript and Node.js packages. You need the npm utility, which is bundled with the pre-built binaries of Node.js.

Node Package Manager (npm)

The npm Registry (<https://registry.npmjs.org/>) is a central public repository, the largest collection of software packages in the JavaScript ecosystem. The npm utility is a command-line tool that performs version management and dependency management of Node.js packages. Package installation is an important role of npm, but it also initializes the project, executes actions such as start, and manages dependencies. npm thus is the control center of your Node.js project.

npm install

Installation of a package from the central registry is the main role of the npm utility.

```
npm install package
```

By default, npm installs the given package in the local scope. It creates a `node_modules` subfolder inside the current project folder and places the package dependencies in it. It also updates the `package.json` file, which is the central configuration file of your Node.js project.

On the other hand, if the `-g` option is added to the command as follows.

```
npm install package -g
```

the package is installed globally and does not reflect in your project's `package.json` file, which is the central configuration file of a Node.js project.

npm init

To set up a new Node.js project in the current folder, use the `npm init` command. By default, it prompts the user to enter project details such as the project name, main entry point, and author. Based on the inputs,

the package.json file is created inside the project folder. Adding -y to the command creates it with all default values.

```
npm init -y
```

Wrote to D:\workspace\helloapp\package.json:

```
{
  "name": "helloapp",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" &&
exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

The scripts field has one or more user-defined terminal commands. The default is a test command echoing an error message. You often find the start command that executes your project's entry point script.

```
"scripts": {
  "start": "node index.js"
}
```

As a result, use

```
npm start
```

command to run your project.

Note that when you install a new package in this project folder (with npm install), its package.json file is updated.

Callbacks

The single-threaded, non-blocking, asynchronous nature of Node.js largely depends on the callback mechanism. In traditional server-side technologies, the code is executed synchronously. If an instruction involves a time-consuming process, the execution thread gets blocked. The multithreading approach solves this issue but is very expensive in terms of CPU and memory usage.

In contrast, the asynchronous execution model uses a single thread with a long-running process yielding to subsequent instructions, thereby employing a cooperative multitasking approach.

Any function that is passed as an argument to another function is a callback. It is called after the first function (the one to which it is an argument) has completed its task. Here is a simple example of how a callback works. There are two functions: `sayHello()` and `sayThanks()`, with the latter used as a callback in the former.

```
/*callback.js*/
function sayHello(name, callback) {
    console.log("Hello, " + name + "!");
    callback(); // Executing the callback function
    console.log("How are you doing today?");
}

function sayThanks() {
    console.log("Thanks for calling!");
}

// Pass sayThanks as the callback
sayHello("Alice", sayThanks);
```

When executed, the following output is obtained in the console.

```
Hello, Alice!
Thanks for calling!
How are you doing today?
```

This example does demonstrate a callback, but the execution is synchronous. To test asynchronous behavior, let's use a library function in JavaScript: `setTimeout()`.

`setTimeout()` is a typical asynchronous function. It returns immediately after scheduling the callback function or code to run. It does not block the execution of any code after `setTimeout()` until the scheduled code has run.

```
setTimeout(function, delay);
```

The following is an example.

```
setTimeout(() => {  
    console.log("Hello World");  
}, 5000);
```

This logs the given message after a delay of 5 seconds. Let's wrap the call to `setTimeout()` inside the preceding `sayThanks()` function. The following is the updated code.

```
/*callback.js*/  
function sayHello(name, callback) {  
    console.log("Hello, " + name + "!");  
    callback(); // Executing the callback function  
    console.log("How are you doing today?");  
}  
  
function sayThanks() {  
    setTimeout(() => {  
        console.log("Thanks for calling!");  
    }, 5000);  
}  
  
// Pass sayThanks as the callback  
sayHello("John", sayThanks);
```

As you run the `sayHello()` function, the first Hello message displays. It is followed by a call to `sayThanks()`, but it is scheduled to introduce a 5-second delay. Hence, without blocking the execution, the next message (How are you doing today?) will be displayed. Then, after the delay elapses, the thanks message prints.

```
Hello, John!  
How are you doing today?  
Thanks for calling!
```

Error-First Callback

Asynchronous operations usually deal with the environment outside the program, such as file I/O, networks, or databases, and hence are either likely to cause runtime errors or likely to succeed. The callback mechanism of Node.js is designed to handle both the error and the data returned by the successful completion of the operation. Hence, Node.js follows a convention for defining the callback function: the first argument is always the error object, and the second argument is the data. The following is the structure of a typical callback definition.

```
function callback(err, data) {  
    if (err) {  
        // handle error  
    } else {  
        // use data  
    }  
}
```

So, when an async operation fails, the callback function is called by passing the error object and the data object set to null. On success, it is called with null as the error object and the data returned by the operation.

The `readFile()` function in the `fs` module is a classic example of how the error-first callback works.

```
fs.readFile(filepath, [encoding], callback);
```

The first mandatory argument is the file to be read. The second argument, the encoding, is optional; the default is utf8. The third argument is the callback function, which receives two arguments: the error that may arise in file reading, and the file contents on success. The function either displays the error message or the file contents.

```
const contents=function(err, data) {  
    if (err) {  
        console.error("Error reading file:", err);  
        return;  
    }  
    console.log("File content:", data);  
};
```

Try reading a file with the statement.

```
const fs = require('fs');  
fs.readFile('file.txt', 'utf8', contents);
```

If the file is indeed present, its contents will be displayed. If not, the error log will be as follows.

```
Error reading file: [Error: ENOENT: no such file  
or directory, open 'C:\users\user\file.txt'] {  
  errno: -4058,  
  code: 'ENOENT',  
  syscall: 'open',  
  path: 'C:\\users\\user\\file.txt'  
}
```

Instead of a named callback, Node.js developers prefer anonymous arrow functions. Here is the arrow function variation of the `readFile()` call.

```
const fs = require('fs');

fs.readFile('file.txt', 'utf8', (err, data) => {
  if (err) {
    console.error("Error reading file:", err);
    return;
  }
  console.log("File content:", data);
});
```

The callback mechanism is the original method of handling asynchronous operations in JavaScript and, by extension, Node.js. The ES6 revision of the JavaScript standard introduced a more structured Promises-based approach. The modern ES2017 standard recommends using `async/await` for asynchronous processing.

http Module

This chapter focuses on Node.js as a server-side technology in web application development. In this context, its `http` module is the foundation of server-side web programming, as it defines the important functionality that helps you create an HTTP server and handle HTTP requests and responses.

In the third chapter of this book (ANATOMY OF PROTOCOL), you learned how the HTTP request-response cycle works, the HTTP methods, the headers, and the status codes. This section explains its various objects and methods that implement these concepts.

The `createServer()` Function

A web application needs an HTTP server object that listens to the incoming HTTP requests and sends the appropriate HTTP response. The `createServer()` function in the `http` module launches the server object. It uses a listener function as its argument. The listener function, in turn, has two arguments: request and response objects.

```
listener = function (request, response) {  
  //request processing and response generation logic  
};
```

The Node.js runtime obtains the request and response objects from the incoming HTTP request as instances of `IncomingMessage` and `ServerResponse`. This listener function is the argument for the `createServer()` function.

```
const http = require('node:http');  
const server = http.createServer(listener);
```

Once the server object is ready, it can be used to listen to incoming requests at a specified port.

```
server.listen(3000);
```

Instead of defining a separate listener function and then using it in the `createServer()`, you can use an arrow function as a callback.

```
const http = require('http');  
  
const server = http.createServer((request,  
  response) => {  
  //listener code  
});
```

The server object's `listen()` function may also use a callback function as the argument. For example, to display a message on the console.

```
server.listen(3000, () => {  
  console.log('Server listening on  
http://localhost:3000/ ...');  
});
```

The response Object

The listener function receives the response object as an instance of `ServerResponse`. The server sends data in the response stream, using various methods.

The `response.write()` function pushes the given string in the response stream.

```
response.write("Hello World");
```

The `response.writeHead()` function writes the status code and the headers to the stream.

```
response.writeHead(200, { 'Content-Type':  
'text/html' });
```

`response.end()` is called to signal that there is no more data to be pushed in the response stream. A string argument to this function is optional.

```
response.end();
```

Hello World

Let's put all these statements together in the `index.js` script as our Node.js server application that renders the Hello World message as its response when a request is received at port 3000.

```
const http = require('http');  
  
const server = http.createServer((request,  
response) => {  
  response.writeHead(200, { 'Content-Type':  
'text/html' });  
  response.write('Hello, World!');  
  response.end();  
});  
  
server.listen(3000, () => {
```



```
    console.log('Server running at  
http://localhost:3000/');  
});
```

Run this application with the `npm start` command, open your browser, and enter `http://localhost:3000/` as the URL. The browser says Hello, World!

The request Object

The request object, as stated, is an instance of `IncomingMessage`. Some of its properties are frequently needed when you process the request data in a server application.

request.url: This property returns a string that trails after the domain:host name. It starts from the first `/` character, includes the `?` character and query string appended after it. You need to use the `url.parse()` function here. (Don't forget to import the `url` module). For example, the URL in the client is `http://localhost:3000/users?name=XYZ&age=25`, then the `request.url` property returns [/users?name=XYZ&age=25](#). You can further extract the querystring part by using the `url.query` property.

```
const http = require('http');  
const url = require('url');  
  
const urlParsed = url.parse(request.url, true);  
const queryString = urlParsed.query;
```

The value of a given key may also be fetched.

```
const name = queryString.name || 'Unknown';
```

Here, `name` becomes `'Unknown'` if the `'name'` key is not found in the query string.

request.headers: The value of this property is a JavaScript object, a collection of key-value pairs where the key is the HTTP header (for

example, host, connection, user-agent, etc.)included in the request and its value. The headers object may also include the cookies header if the server has set cookies in an earlier response.

request.method: This property represents the HTTP method used by the client to send the request. Expectedly, it returns a string equaling GET, POST, PUT, or DELETE.

The GET Method

To process the data received with a GET request from the client, the listener function body checks whether request.method matches GET', parses the request data, and formulates an appropriate response back to the client. Here, we just send back the received data.

```
const server = http.createServer((request,
response) => {
  if (request.method==='GET')
  {
    const urlParsed = url.parse(request.url,
true);
    const queryString = urlParsed.query;
    const name = queryString.name || 'Unknown';
    const age = queryString.age || 'Unknown';
    response.write('GET request data\n');
    response.write(`name: ${name}\n`);
    response.write(`age: ${age}\n`);
    response.end();
  }
});
```

Assuming the browser's URL is `http://localhost:3000/users?name=XYZ&age=20`, you should see the following data rendered.

```
GET request data
name: XYZ
age: 20
```

The POST Method

The browser, as an HTTP client, can only send a GET request. The POST method sends the data as the request body. Hence, you either need to collect the data in an HTML form and submit the data to the server with its method attribute set to POST. Or, you can use an HTTP client such as HTTPie, Postman, or Insomnia.

The data from the request body is received as a stream. Moreover, the data is received in small intermittent chunks. Hence, the request object emits events such as data (every time it receives a chunk) or error (whenever something goes wrong during streaming and ends), which are fired once when all data has been received. The `request.on()` method is an event listener that executes a callback in response to one of these events.

```
request.on(eventName, callback);
```

Inside the `createServer()` function, put an if block that checks whether the `request.method` is POST.

Whenever the server receives the data event, concatenate the request body chunks.

```
let postData = '';

request.on('data', (chunk) => {
  postData += chunk.toString();
});
```

When the stream terminates, send the JSON representation of the request body as the response.

```
request.on('end', () => {
  console.log('Received Body:',
postData);
  const parsedData =
JSON.parse(postData);
```

```
        response.writeHead(200, { 'Content-  
Type': application/json' });  
        response.end(JSON.stringify({  
            data: parsedData  
        }));  
    });
```

Start the Node.js server before you send a POST request. Use the HTTPie client.

Select the POST request type and enter `http://localhost:3000/-` as the URL. From the Headers menu, set Content-Type to `application/json`, choose JSON as the body type, and enter the body as follows.

```
{  
  "name": "XYZ",  
  "age": 20  
}
```

Send the POST request. The server sends the JSON response in the bottom panel. See Figure [13-1](#).

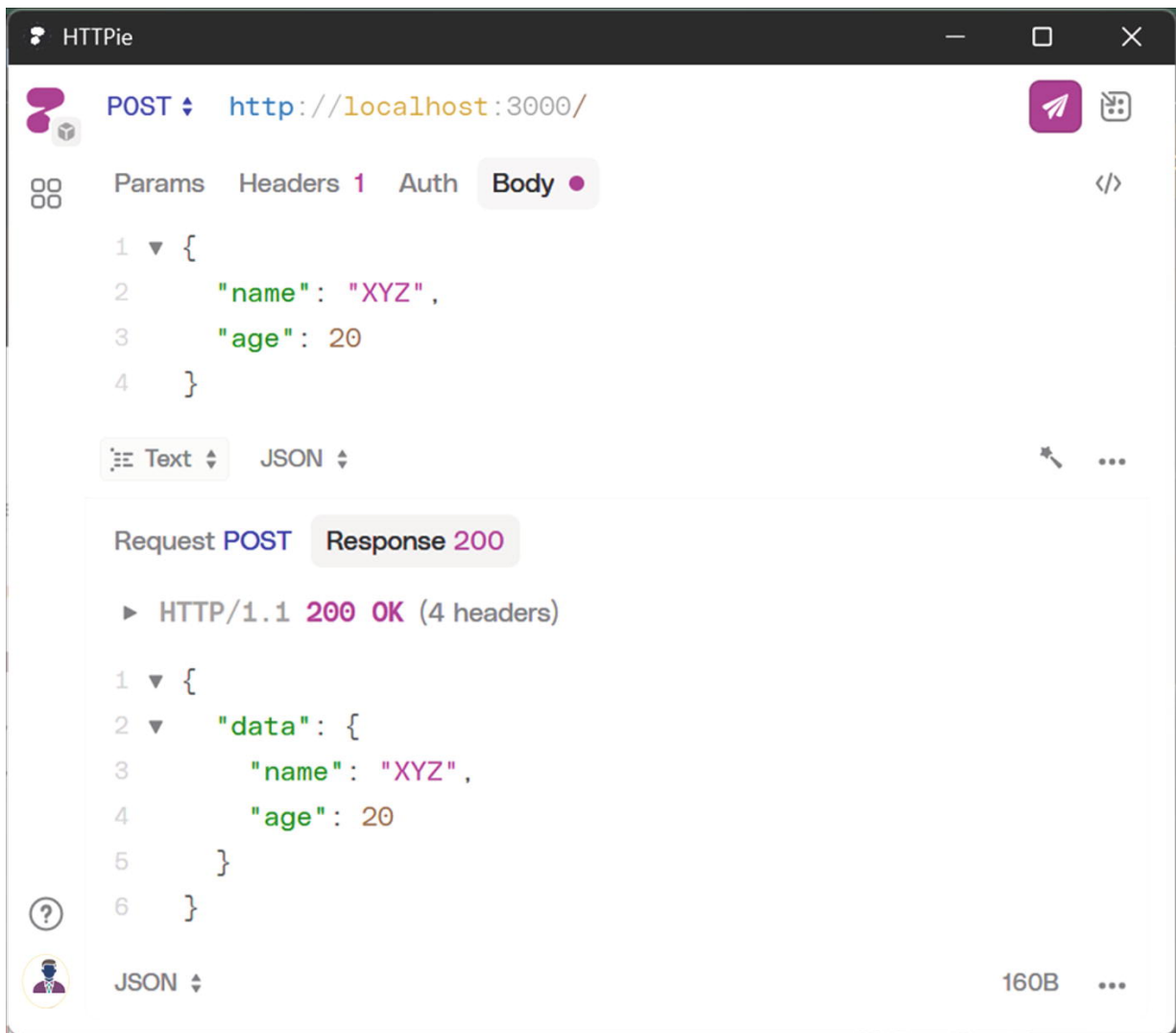


Figure 13-1 POST request in HTTPie Client

Node.js uses the core `http` module to launch the server and involves manual handling of HTTP requests and responses. You also need to parse URLs, conditionally process request methods, and construct appropriate responses.

While this approach is suitable for simple use cases, it has many limitations as it involves more complex routing logic.

The Express.js framework addresses these limitations by providing a robust and declarative routing system, easy integration with template engines such as EJS, and a powerful middleware architecture.

Points to Remember

- Node.js uses Google's V8 JavaScript engine. It is a non-blocking, event-driven runtime used to build server-side applications in JavaScript.
- JavaScript runs inside the browser if it is enabled. Node.js must be installed to run on a server. Pre-built binaries for Windows, Linux, and macOS are available on <https://nodejs.org>.
- You can use the Node.js REPL terminal to test small snippets of JS code, or run JavaScript files from the command line.
- Pre-built binaries are bundled with modules that can be imported by the `require()` function.
- To install a third-party package/library of modules from the central public repository (<https://registry.npmjs.org/>), you need to install the npm utility.
- Use the `npm install` command to install a Node.js package.
- The `npm init` command initializes a typical Node.js project by creating a `package.json` configuration file.
- The callback mechanism is at the heart of the asynchronous and event-driven nature of Node.js.
- A callback is a function that is used as an argument for another function.
- Even if the callback is a long-running process, it doesn't block the current thread.
- Node.js uses an error-first pattern in the definition of a callback. The first argument of the callback is an error object, and the second argument is the data returned by the calling function.
- The server-side HTTP operations are handled by the `http` module in Node.js.
- The `createServer()` function returns the server object.
- You need to call its `listen()` function, which binds the server to a given port and starts accepting incoming HTTP requests.
- The callback to the `createServer()` function processes the HTTP requests and writes responses in the `ServerResponse` stream.

- By default, the browser sends a GET request. To send a POST request, you either use an HTML form or use an HTTP client such as HTTPie or Postman.
-

Summary

This chapter prepares a solid foundation for you as a web application back-end developer. It introduces the basic concepts of Node.js as a server-side programming technology. The callback function mechanism has been explained with suitable examples. You also learned how Node.js handles the GET and POST requests and formulates the server response.

This chapter concludes by pointing out the limitations of pure Node.js for building server applications, and mentions that Express.js addresses these issues. Hence, the next chapter explains the important features of the Express framework, the EJS templates, and how it serves static files, among other topics.

14. Express.js Framework

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

The previous chapter highlighted various advantages of using Node.js to develop the server-side logic of a web application. It allows you to use JavaScript for both the front end and the back end. Add to it its asynchronous, non-blocking I/O approach and the event-driven architecture.

However, if you use pure Node.js, you will hit certain limitations sooner rather than later. You need to manually handle the routing mechanism, which becomes tedious, particularly if it involves complex routing logic. The same is true for important aspects of a server application, such as authentication and error handling.

The Express.js framework offers a layer of abstraction over the core functionality of the http module in Node.js. It reduces the lengthy boilerplate code and improves code organization. Additionally, it has important features such as middleware and supports template engines.

This chapter discusses the fundamentals of the Node Express framework and its built-in routing system. This chapter also explains how to serve static assets and Embedded JavaScript (EJS) templates. This will give you a solid foundation for building a dynamic web application.

Express: Important Features

In general, a web framework (or web application framework) is a software environment designed to facilitate rapid development of web applications, including services and APIs, without having to handle low-level operations such as the HTTP request-response cycle. Express.js framework (or simply Express) is the preferred framework for web application development on the Node.js platform.

Developed by T. J. Holowaychuk in 2010, Express is free and open source software. <https://expressjs.com/> is its official website. Express is often described as a minimal and unopinionated framework.

It is **minimal** because it includes the most essential features (such as routing, a middleware system, and helpers to handle request and response objects) for web application development.

It is **unopinionated** because it doesn't force the developer to use a particular design pattern nor is it tied to any specific templating library or an ORM.

Let's briefly overview the important features of the Express framework.

Routing

Before the rise of web frameworks, websites used to map the URL directly to a physical file on the server. The routing mechanism matches the client request to a specific piece of code (method or function) based on the HTTP method, the parameters, and headers that come along with the request.

For example, <http://example.com/products.php?id=10> is a file-based URL. The server locates the file products.php, the parameter id=10 is parsed and processed by the server code, and the result is rendered as its response. In the modern approach, the server's routing mechanism reads the URL

<http://example.com/products/10> and maps it to a function.

The method or function thus identified parses the path parameters, processes the request, and formulates the response. The routing mechanism, along with the template engine, generates dynamic responses.

Later in this chapter, you will learn how the routing works in the Express framework.

Middleware

The term middleware refers to a software component that intercepts the incoming HTTP request before it is processed by the router. It can also be employed to run after the request handler. The middleware component is responsible for operations like authentication, logging, error-handling, and security enhancement.

Static Assets

Even for a dynamic web application, it needs to deliver content to the client that doesn't change with respect to user input or server-side logic. For example, the media files, the stylesheets, and so forth. The built-in middleware `express.static` makes it easy to serve the static assets.

Request and Response Helpers

If you use pure or vanilla Node.js to build the server-side logic, you are forced to take a laborious and verbose approach to processing request data and formulating the response. Express provides a useful collection of helper functions that simplify the request and response handling.

Template Engine Support

In the web application context, a web template is an HTML document that embeds placeholders for dynamic data and statements in a certain templating language inside the standard HTML script. The server uses a templating engine that substitutes placeholders with real-time data, dynamically builds the HTML content, and renders the response.

Express integrates various template engines, such as EJS and Pug. This is an important feature that enables the development of dynamic web applications using server-side JavaScript.

Installation

Express.js is available on the npm Registry. You can install it in your Node.js project with the npm install command. However, you need to make sure the project has a configuration file (package.json).

Start by creating a new folder and making it current. Then initialize the Node.js project as follows.

```
$ mkdir HelloExpress  
$ cd HelloExpress  
$ npm init
```

Then include the following installation command.

```
$ npm install express -save
```

This command saves Express in the current project environment. Figure [14-1](#) shows it inside the node_modules folder.



```
node_modules  
├── express  
│   └── lib  
│       ├── application.js  
│       ├── express.js  
│       ├── request.js  
│       ├── response.js  
│       ├── utils.js  
│       └── view.js
```

Figure 14-1 node_modules folder

The project's package.json is also updated, adding express (its latest version) in the dependencies section.

```
"dependencies": {  
  "express": "^5.2.1"  
}
```

Hello Express

To have a basic Express application that renders a Hello Express message, first you need to obtain the application object by calling the `express()` function (you need to import the express module).

```
const express = require('express');  
const app = express();
```

This application object lets you call various methods that listen to incoming requests, handle routing HTTP requests, register templating engines, render HTML views, and so forth. So, in a way, the app object is the central hub of your Express application.

To make your app listen to the incoming requests on port 3000, use the `app.listen()` function.

```
const server = app.listen(3000, function () {  
  console.log("Express App running at  
http://127.0.0.1:3000/");  
})
```

Note that this method call is very similar to `http.Server.listen()` when using the raw Node.js server application.

The app object can handle different HTTP requests by calling a method of the corresponding name. For now, we'll use the `app.get()` method to render the Hello Express string as the server's response to a GET request at the `'/'` endpoint.

Save the following code as `index.js` in the project folder.

```
const express = require('express');
const app = express();

app.get('/', function (req, res) {
  res.send('Hello Express');
});

const server = app.listen(3000, function () {
  console.log("Express App running at
http://127.0.0.1:3000/");
});
```

As the preceding script is run, the server starts listening to incoming requests.

```
$ node index.js
Express App running at http://127.0.0.1:3000/
```

To test, open the browser and visit the preceding URL; you should find the Hello Express message rendered.

express-generator

Express.js comes with a useful command-line tool called `express-generator`. It scaffolds a typical folder structure of a ready-to-use Express project. It auto-generates a lot of boilerplate code, the starter and configuration files, along with certain sample routes. With the basic application ready, you can then expand on it by adding other functionality.

To use the express generator utility, use the **`npx express-generator`** command while inside the application folder.

```
$ mkdir my-project
$ cd my-project
```

```
$ npx express-generator
$ npm install
```

This results in the folder structure for your app as shown in [Figure 14-2](#).

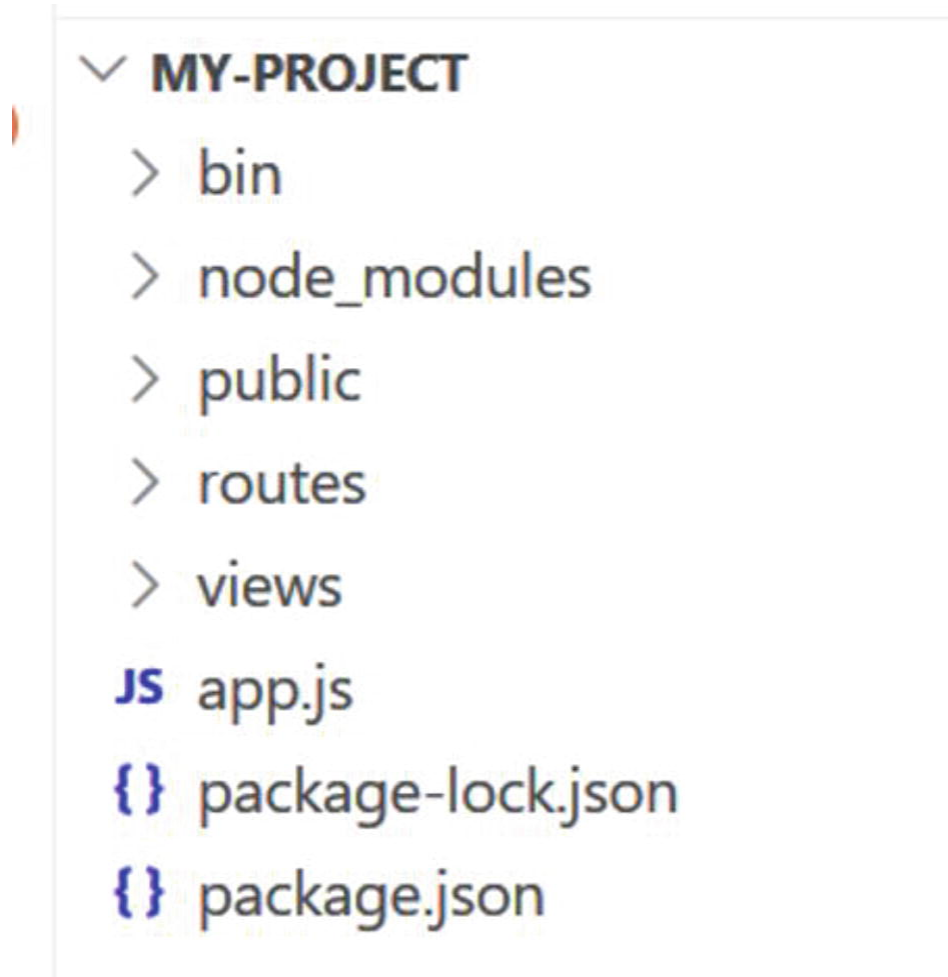


Figure 14-2 Express app structure

Run `npm start` from the terminal and visit `http://localhost:3000/` in your browser. The following display confirms the successful creation of an Express app with `express-generator`. See [Figure 14-3](#).

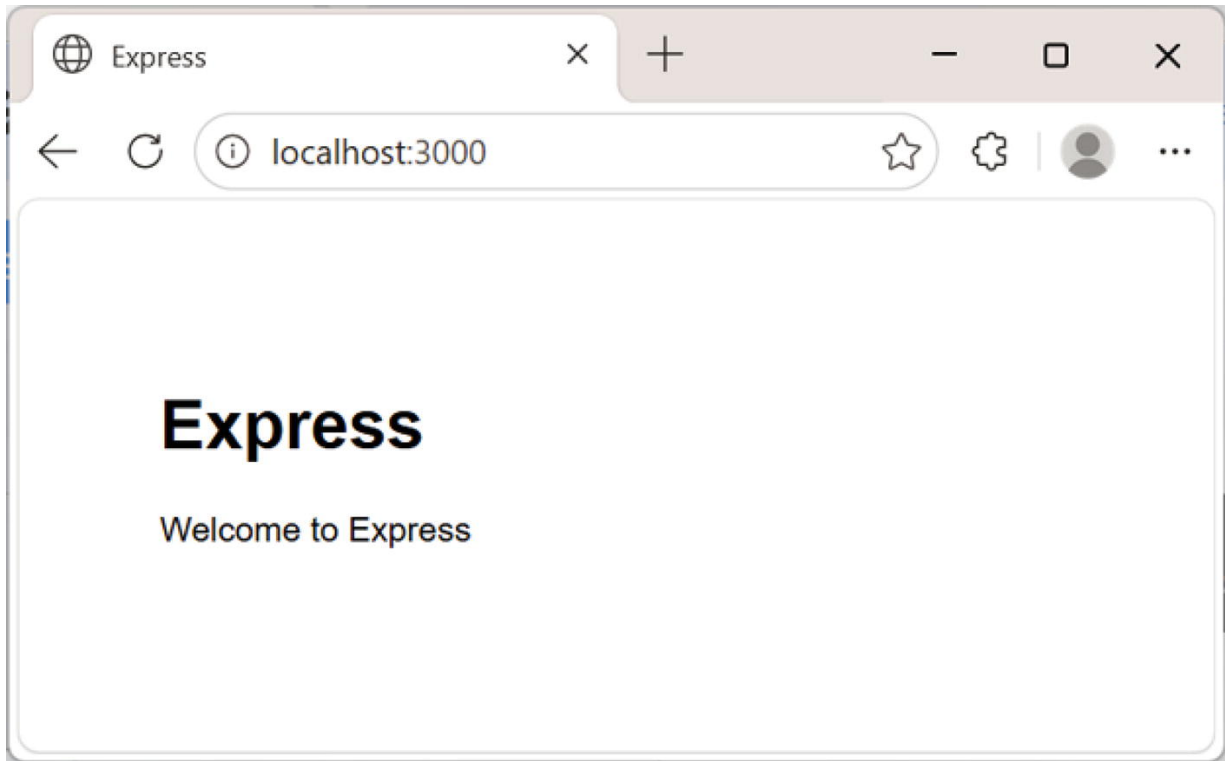


Figure 14-3 Express app by Express generator

Routing Mechanism

Web application frameworks like Express.js employ a routing mechanism that identifies the incoming HTTP request, its method and parameters, and directs the application flow to a certain handler function.

The Express application object (app in the example) is responsible for handling routing. A route maps an incoming HTTP request—defined by its method (such as GET, POST, PUT, or DELETE) and URL path—to a specific handler function that processes the request and generates an appropriate response.

The following is the general syntax that defines a route.

```
app.method(path, callback);
```

Here, app is the Express application object and method stands for any of the HTTP methods (names are always in lowercase). The path

argument is a string representing the URL endpoint, which, when matched, executes the callback—the second argument.

You have seen the following route in our Hello Express example.

```
app.get('/', function (req, res) {  
  res.send('Hello Express');  
});
```

This definition simply means that when the app object receives a GET request for the '/' endpoint (which is the URL `http://localhost:3000/`), the callback function sends a Hello Express string in the response stream.

Path Parameters

Web frameworks also don't adhere to the classical approach of passing parameters in a query string attached to the URL (as in `/users?name=XYZ&age=20`). Instead, the modern way of forming the URL endpoint is `/users/XYZ/20`. Note that in the former case, the query string includes the keys, while the route-based endpoint has only the values separated by the '/' symbol. Hence, the parameters need to be specified in the route definition. To parse a value, the path must have the parameter placeholder preceded by the ':' symbol.

```
app.get('/users/:name/:id', function (req, res) {  
  const id = req.params.id;  
  const name = req.params.name;  
  console.log(req.params);  
  res.send(`

# User: ${name}, ID: ${id}</h1>`); });


```

The route parses the values from the endpoint into the named parameters in the path and constructs a `req.params` object, as the browser's console shows.


```
{ name: 'XYZ', id: '20' }
```

The server sends its HTML response to the browser, rendering the username and age in an h1 heading.

Middleware

In addition to routing, middleware is also an important feature of the Express framework. Middleware is any function that works between the request and response cycle. It can be employed after the server receives the request and before the response is sent to the client. Middleware functions are used to perform tasks such as logging, authentication, and error handling.

Middleware has access to the request and response objects. Hence, it can change their composition. You can have a sequence of middleware functions executed one after the other. The first one must call `next()` to pass control to the next.

The callback function that we used as one of the arguments of the route definition is itself a middleware.

```
app.get(path, (req, res) => {  
  //route handler;  
});
```

The anonymous callback function in this route definition is a middleware; more precisely, a route-level middleware.

However, you can have middleware as another argument in the route definition.

```
const middleware = function (req, resp, next) {  
  //middleware code;  
  next();  
};  
app.get(path, middleware, (req, res) => {  
  //route handler;
```

```
});
```

In the following example, `function1` is used as a middleware argument in the `app.get()` route.

```
const function1 = function (req, res, next) {  
  console.log("before executing the route  
handler");  
  next();  
}  
app.get('/home', function1, function (req, res) {  
  console.log("Hello Express");  
});
```

As a result, when the endpoint `'/home'` is visited, the middleware code is executed first. The call to `next()` passes control to the subsequent callback.

To mark a middleware on receipt of each client request, use the `app.use()` method.

```
app.use(middleware)
```

With `app.use()`, Express applies middleware to all subsequent routes or a specific path. Whereas with `app.get(middleware, handler)`, Express applies middleware only to that specific route.

Built-in Middleware

Various ready-to-use functions (without having to install any packages) are bundled with Express. Since they are always available, you only need to call them by having the required middleware as the argument of the `app.use()` statement.

```
app.use(built-in middleware);
```

The following is an example.

```
app.use(express.json());
```

Next, let's go over some of the most frequently used built-in middleware functions.

express.json()

This middleware function parses incoming requests with JSON payloads and populates the request body with a JavaScript object. Usually, a GET request doesn't contain a request body; therefore, this middleware is typically not used with the `app.get()` route. However, in an `app.post()` or `app.put()` route, the request does contain a body component (usually the HTML form submitted to its URL endpoint), so you generally find `express.json` middleware used.

In the following example, the `express.json()` middleware is used to parse the incoming POST request into a JSON response.

```
app.use (express.json());

app.post('/', function (req, res) {
  console.log(req.body);
  res.send(req.body);
});
```

We need to send a POST request to the `/` endpoint. In the previous chapter, we used the HTTPie tool as the HTTP client. This time, let's use `cURL`, a command-line tool to send HTTP requests to a server and obtain its response. Open the command terminal of your operating system and enter the following command.

```
$ curl -i -v POST http://127.0.0.1:3000/ -H
"Content-Type: application/json" -d
'{"name": "John", "age": 20}'
```

In this command, the `-H` option sets the Content-Type header to JSON. The `-d` option specifies the POST data that forms the request

body. If your Express server is running, the preceding cURL command responds as follows.

```
> POST / HTTP/1.1
> Host: 127.0.0.1:3000
> User-Agent: curl/7.85.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 23
>
} [23 bytes data]
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: application/json; charset=utf-8
< Content-Length: 23
< ETag: W/"17-uLRF2uAZxXCyaVfUx2BXJMYfcQM"
< Date: Sun, 03 May 2026 18:36:25 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
<
{ [23 bytes data]
100      46 100      23 100      23   3183   3183 --:--
--:-- --:--:-- --:--:--  9200HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 23
ETag: W/"17-uLRF2uAZxXCyaVfUx2BXJMYfcQM"
Date: Sun, 03 May 2026 18:36:25 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"name":"John","age":20}
* Connection #1 to host 127.0.0.1 left intact
```

express.urlencoded()

This built-in middleware function parses incoming requests with urlencoded payloads. When an HTML form sends the data as a POST request, its enctype attribute is set to “application/x-www-form-urlencoded”. This middleware also populates the request body with key-value pairs.

Call the following to enable parsing of the form data.

```
app.use(express.urlencoded({ extended: true }));
```

express.static()

This middleware is required when you want to serve static files (HTML, CSS, JS, images).

express.Router()

This middleware creates a modular routing system.

Template Engine

In general, the term *template* refers to a pre-designed structure or pattern of an artifact. For example, a mold used in manufacturing plastic or metal products acts as a template. Similarly, we often use pre-built templates in applications like Microsoft Word or Microsoft PowerPoint to create documents or presentations quickly.

A web template is a document that provides the basic structure or skeleton of a web page, but contains placeholders where variable or dynamic data can be inserted. See Figure [14-4](#).

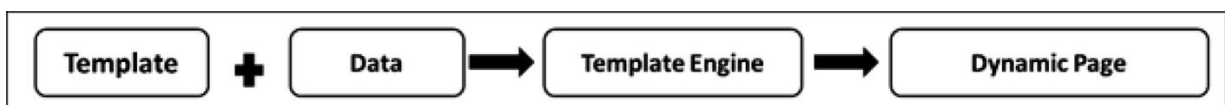


Figure 14-4 Web template engine

Web application frameworks such as Express.js use a template engine that

- reads data from a source,
- inserts the data into appropriate placeholders,
- and dynamically generates the final web page.

Express can generate dynamic web content by interacting with template engines such as EJS, Pug, and Handlebars.

EJS Templates

The EJS template engine (stands for Embedded JavaScript; Express calls it a view engine) is intended for use with any web application powered by the Node.js runtime. With EJS, you can intersperse JavaScript logic inside the HTML script. Released in 2007 as a Node.js package, its latest version can be installed from the npm Registry.

```
$ mkdir EJSapp
$ cd EJSapp
$ npm init -y
$ npm install express ejs
```

The EJS library is installed in the `node_modules` subdirectory, and the `package.json` settings are updated to include `ejs` in the dependencies list.

```
{
  "dependencies": {
    "ejs": "^5.0.2",
    "express": "^5.2.1"
  }
}
```

You need to explicitly set EJS as the template engine.

```
const express = require('express');
const app = express();
```

```
app.set('view engine', 'ejs');
```

Express expects the view template to be put inside the views subfolder (inside the project folder).

```
$ mkdir views
```

A normal HTML script, with or without embedded JavaScript code, becomes a view if saved with the .ejs extension.

A basic web page that displays the Hello EJS message is saved as index.ejs.

```
<!DOCTYPE html>
<html>
<head>
  <title>EJS Template</title>
</head>
<body>
  <h1>Hello EJS</h1>
</body>
</html>
```

For now, there's no JavaScript code embedded in this HTML script.

Previously, we used the `response.send()` method that adds either plain text, an HTML string, or JSON data in the response stream. Here, we use the `render()` method of the response object, which reads the HTML code from the view template (.ejs file), invokes the template engine, inserts the dynamic contents into the placeholders (if present) in the HTML, and then sends it in the response stream.

```
res.render(view, data)
```

If you don't pass any data to the view template, you can drop the second argument.

```
res.render(view)
```

Put the preceding statement inside the `app.get()` route that displays the template in response to the `'/'` endpoint in your browser.

```
app.get('/', (req, res) => {  
  res.render('index');  
});
```

The actual strength of EJS lies in inserting dynamic data into HTML. Assuming that the URL is entered with the username trailing the forward slash (such as `'/Admin'`). The view should accordingly display Hello Admin. Hence, we need to modify the template (`views/index.ejs`) as follows.

```
<body>  
  <h1>Hello <%= name %></h1>  
</body>
```

In the previous section, you learned how to parse the path parameters from the URL. Let the path parameter here be `'name'`. Modify the `app.get()` route accordingly.

```
app.get('/:name', (req, res) => {  
  const name = req.params.name;  
  res.render('index', { name: name });  
});
```

The template engine substitutes the value of the `name` parameter in the placeholder. As a result, you get a Hello Admin message in the browser window in response to the URL `http://localhost:3000/Admin`.

EJS Syntax

To embed JavaScript code in the template, EJS uses special tags: `<%` and `%>`.

`<%= expr %>`

Any expression inside `<%=` and `%>` is rendered as part of the browser output, as in the preceding example.

```
Hello <%= name %>
```

The template engine substitutes the value of the `name` property (Admin as in the example) at runtime and displays Hello Admin.

`<% code %>`

If you need to place JS code with conditional or looping constructs, you can put it inside the `<%` and `%>` tags. This doesn't add any output to the browser.

Let's rewrite the `app.get()` route that reads two path parameters, `name` and `marks` of a student.

```
app.get('/:name/:marks', (req, res) => {  
  const name = req.params.name;  
  const marks = req.params.marks;  
  res.render('index', { student: { name: name,  
marks: marks } });  
});
```

The student object is passed to the view template as the data.

You can include conditional logic in the EJS template code to display the result as pass or fail.

```
<% if (student.marks >= 50) { %>  
  <p><%= student.name %> passed with <%=  
student.marks %> marks.</p>  
  <% } else { %>  
    <p><%= student.name %> failed with <%=  
student.marks %> marks.</p>  
  <% } %>
```

As a result, if the parameters are passed as `/John/60`, the browser shows the response as John passed with 60 marks.

Similarly, you can also insert a JS loop and iterate over the array-like object that a template might receive from the server.

`<%# text %>`

The `<%# text %>` tag allows you to insert a comment in the template.

Static Assets

Although the main focus of web application development is rendering dynamic output in response to the user's request and sending variable data to the server, it still needs to use many static resources, such as stylesheets, images, and JavaScript code modules. These are called *static assets* because they remain the same regardless of the variable data.

The `express.static()` middleware creates a route for handling such types of files, so that you don't need to manually write `app.get()` routes for each. The static middleware uses the following syntax.

```
express.static(root);
```

Here, the `root` argument denotes a folder in your project from where the static assets are served. As a convention, it is set to `public`—the folder in which all the static files are placed.

The static middleware is mounted with the `app.use()` function.

```
app.use(express.static('public'));
```

As a result, if a file (`logo.png`) is present in the `public` folder, the URL `http://localhost:3000/logo.png` can fetch it.

Let's modify the index view to display the logo as a static asset. You need to create a new folder (`public`) inside your project and ensure that `logo.png` resides in it.

Call the `express.static()` middleware in the Express application code (`app.js`).

```
const express = require('express');
const app = express();
app.use(express.static('public'));

app.set('view engine', 'ejs');

app.get('/', (req, res) => {
  res.render('index');
});

app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

The `app.get('/')` route is defined to render the index view, which uses the `<img>` tag to render the logo.

```
<!DOCTYPE html>
<html>
<body>

</body>
</html>
```

The folder structure of your project should resemble what's shown in Figure [14-5](#).

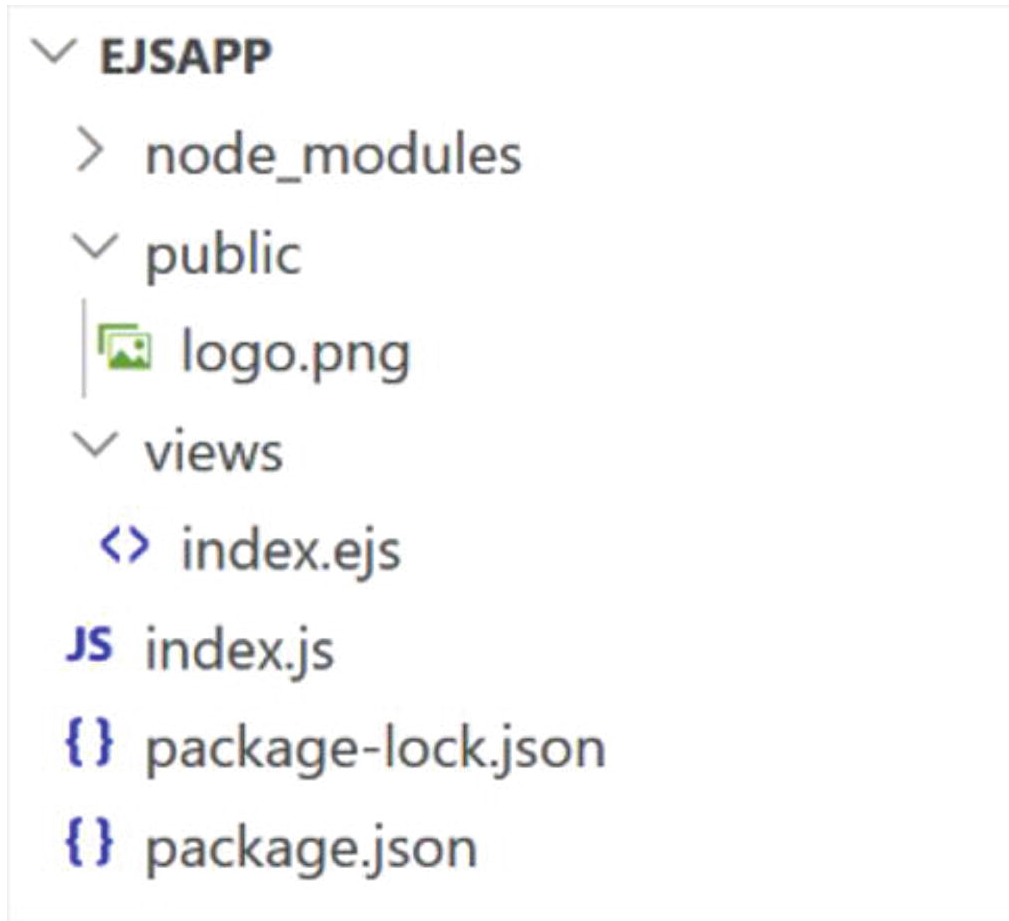


Figure 14-5 Express project with views folder

Run the Express server, visit the `/` endpoint, and you should see the logo displayed.

Error Handling

Web applications often encounter runtime errors (such as invalid URLs, database connection failure, file not found, etc.), causing them to crash. Express framework provides a structured approach to handle errors gracefully.

By default, the error handler usually sends the 500 Internal Server Error status code. In the following code, every time the path parameter equals `'err'`, it throws an Error object.

```
app.get('/:id', (req, res) => {
```

```
const id = req.params.id;
if (id === 'err') {
  throw new Error('Invalid ID');
}
res.render('index');
});
```

The response is the error string followed by a lengthy debug log.

```
Error: Invalid ID
```

You can define a custom error-handler middleware. Unlike the other middleware functions, it needs four parameters; the first one is the error object.

```
app.use((err, req, res, next) => {
  console.log(err);
  res.json({ error: err.message });
});
```

You don't get the debug log; instead, you get a JSON response, as follows.

```
{"error": "Invalid ID"}
```

You can also write asynchronous error handlers. In Express 5, the route handlers that return an async function call `next(err)` automatically when an error is encountered.

Points to Remember

- Express.js is a minimal and unopinionated web application framework that adds an abstraction layer over the http module functionality of Node.js.

- Routing is an important feature of the Express framework. The Express application object maps a given URL endpoint to a JavaScript function. The route handler functions of Express (`get()`, `post()`, `put()` and `delete()`) register a specified callback to a certain URL endpoint.
- The route handler can also parse the path parameters from the URL endpoint. The parameter name in the path is prefixed with a `:` symbol and separated by the `/` character.
- Middleware in Express is a function that works between the request and response cycle. It has access to the request and response objects. Middleware can modify the request/response, call the next middleware in the chain, or end the response stream.
- The built-in middleware functions that are often used include `express.json()`, `express.urlencoded()`, and `express.static()`.
- Templating is an important aspect of any web application framework, including Express. A template intersperses JavaScript code placeholders in the HTML script. The template engine reads data from the server, uses data provided by the user or fetches it from a database, and populates the placeholders appropriately, thus generating a dynamic web document.
- Express framework can work with EJS, Pug, and Handlebars template engines.
- To use EJS, you need to install it from the npm Registry and set it as the view engine.
- HTML with EJS code is saved as an `.ejs` file and put in the views folder.
- EJS uses special tag notations `<%` and `%>` to embed JavaScript expressions inside HTML.
- The `express.static()` middleware automatically defines a route from which all the static assets, such as images and stylesheets, can be served.
- Express provides a default error handler, to which you can add a custom error-handling middleware.

Summary

This chapter explored the essential features of Express.js, the preferred web application framework for developing server-side applications on the Node.js platform. Concepts such as routing, path parameters, and the EJS template engine have been explained with useful examples. It also touches upon the concept of middleware and serving static assets. Finally, it introduces how Express addresses the error-handling mechanism.

The next chapter is the last in this book. It is dedicated to explaining the basics of SQL databases and using the SQLite database as a back end for Node.js applications.

15. Node.js and SQLite

Malhar Lathkar¹ 

(1) Near Rachewad Hospital, Nanded, Maharashtra, India

Any web application needs to perform read/write operations on persistent storage. While Node.js provides access to the local filesystem and can store/retrieve data using JSON or text files, such unstructured file-based storage suffers from limitations such as data redundancy, poor data integrity, and inefficient searching. Web applications typically use relational databases such as SQLite, MySQL, and PostgreSQL to perform CRUD (create, read, update, delete) operations efficiently and reliably.

A database is a structured and logical organization of data used to overcome these drawbacks. Specifically in a relational database, data is stored in entities called tables, where each table consists of rows and columns representing records and attributes, respectively. This enables establishing relationships between tables using common attributes (keys), leading to efficient and organized data management. Relational database systems such as SQLite use SQL (Structured Query Language) to create tables and manage data stored in them.

Each of these relational database management system (RDBMS) products has its own GUI to perform the CRUD actions. They also provide a command-line shell that allows executing raw SQL queries. However, Node.js developers can interact with databases programmatically using JavaScript through suitable npm packages. For example, the `sqlite3` package available in the npm Registry provides a

comprehensive API for performing database operations by executing SQL queries programmatically from Node.js code.

This chapter discusses the fundamentals of relational databases, explains using the SQLite database with a GUI as well as the SQLite shell. The main focus of this chapter is the `sqlite3` module and how it enables using the SQLite database as the back end for a NodeJS and Express app.

RDBMS Concepts

The concept of relational databases was proposed by Edgar F. Codd in 1970. The core idea of the relational model is to organize data into entity tables. Each table has a pre-defined schema that specifies the column names and their data types.

Each row in a table represents an instance of the entity, while each column represents a distinct attribute of that entity. In RDBMS terminology, a row is also called a *tuple*. For example, each row in a Customer table represents one customer record, characterized by attributes such as Customer ID, Name, Address, and Phone Number. See Table [15-1](#).

Table 15-1 Customer Table

To uniquely identify each record, one attribute (or a combination of attributes) must contain unique values. Such an attribute is called the *primary key*. For example, the Customer ID uniquely identifies each customer in the Customer table.

One of the most important features of the relational model is the ability to establish relationships between tables using common attributes. An attribute in one table that references the primary key of another table is called a *foreign key*. Foreign keys help maintain referential integrity between related tables.

For example, in an Invoices table, Invoice Number may serve as the primary key, while CustomerID acts as a foreign key because it

references the primary key of the Customer table. See Table [15-2](#).

Table 15-2 Invoice Table

Based on this relational model, Donald D. Chamberlin and Raymond F. Boyce developed SQL for manipulating relational data. Oracle is the first SQL-based RDBMS. Since then, several other RDBMS products such as MySQL, PostgreSQL, and SQLite have emerged. All these systems use SQL as the standard language for creating, querying, and manipulating data organized in related tables.

SQL

SQL is predominantly a declarative language. Although it has been periodically standardized (initially by ANSI and later internationally by ISO/IEC, with the latest revision in 2023), various RDBMS vendors implement their own extensions and variations over the standard specification.

SQL instructions are written as statements consisting of keywords and clauses. A statement may contain one or more clauses. SQL statements are broadly classified into DDL, DML, DCL, and TCL.

Data Definition Language (DDL): Commands used to define or modify database objects such as tables and indexes.

Examples: CREATE, ALTER, DROP

Data Manipulation Language (DML): Commands used to manipulate data stored in tables.

Examples: INSERT, UPDATE, DELETE, SELECT

Data Control Language (DCL): Commands related to permissions and access control.

Examples: GRANT, REVOKE

Transaction Control Language (TCL): Commands used to manage transactions.

Examples: COMMIT, ROLLBACK

The implementation of SQL by a particular vendor is called a SQL dialect. Each SQL engine may introduce proprietary keywords, functions, and procedural extensions beyond the standard specification. The SQLite dialect is comparatively lightweight when compared to systems such as MySQL, PostgreSQL, and databases from Oracle Corporation.

SQL Data Types

The SQL standard broadly defines the following categories of data types.

Numeric types: INTEGER, SMALLINT, DECIMAL

Character types: CHAR, VARCHAR

Date/Time types: DATE, TIME, TIMESTAMP

Boolean types: BOOLEAN

Binary types: BLOB

SQLite broadly conforms to standard SQL, although it does not provide a dedicated BOOLEAN type. It internally represents Boolean values using integers (0 and 1). It also doesn't support access control statements: GRANT and REVOKE.

SQLite

SQLite is a lightweight relational database engine in which the entire database is stored as a single disk file. Unlike other server-based RDBMS products such as Oracle or MySQL, SQLite does not require a separate server process. It can be embedded directly into an application without requiring complex configuration. However, despite being lightweight and highly portable, it supports most standard SQL features and transactional operations.

D. Richard Hipp developed SQLite in 2000, as a free and open source library written in C. The latest stable version (SQLite 3.x), both as a precompiled binary bundle as well as source code, is available for download from <https://sqlite.org/download.html>.

Installation

The easiest way is to use the precompiled binaries. For Ubuntu Linux, use the apt package manager.

```
$ sudo apt update
$ sudo apt install sqlite3
```

To use SQLite on Windows, download the latest version of SQLite command-line tools (named as `sqlite-tools-win-x64-xxxxxxx.zip`), unzip it in a suitable folder (such as `c:\sqlite3`) and check the version from the terminal.

```
sqlite3 --version
```

Along with the library, the precompiled bundle also includes a SQLite command-line shell that you can use to run SQL statements interactively against an SQLite database. Additionally, many GUI-based utilities such as DB Browser for SQLite and SQLiteStudio are built on top of the SQLite engine to provide graphical interfaces for database management. You can also install SQLite extensions available for all popular IDEs such as VS Code and JetBrains.

Many programming languages provide bindings or APIs to interact with the SQLite library. The npm Registry hosts packages such as `sqlite3` and `better-sqlite3`. Later in this chapter, you learn how to handle an SQLite database from Node.js code.

DB Browser for SQLite

One of the most popular GUI front ends for the SQLite library, DB Browser for SQLite is an open source, cross-platform tool. You can perform all SQL operations like creating, modifying, and deleting databases/tables/indexes, adding/updating/removing records in a table; and importing data from CSV files as well as exporting the database to a SQL dump—all with conveniently laid-out menus, buttons, and toolbars. Additionally, you can execute pure SQL queries and check the log of SQL commands executed. See Figure [15-1](#).

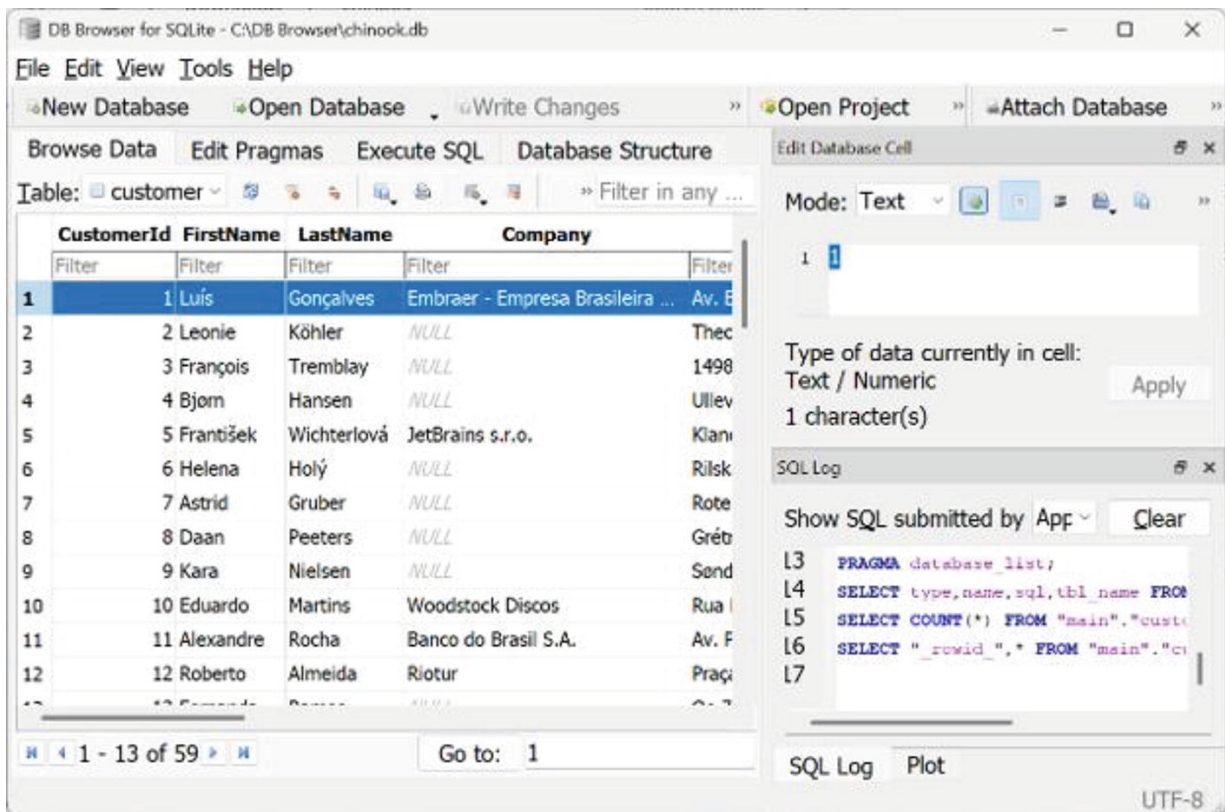


Figure 15-1 DB Browser

SQLite Extension for VS Code

VS Code is a lightweight and multipurpose source code editor, whose functionality can be extended by including different add-ons from its extension marketplace. You will find many extensions there to work with SQLite inside the VS Code environment. One of the most used is the `vscode-sqlite` extension, which allows you to query databases, view results, and export query results to JSON, CSV, and HTML format. The VS Code screenshot in Figure 15-2 shows a Chinook database, a popular open source database in the SQLite Explorer.

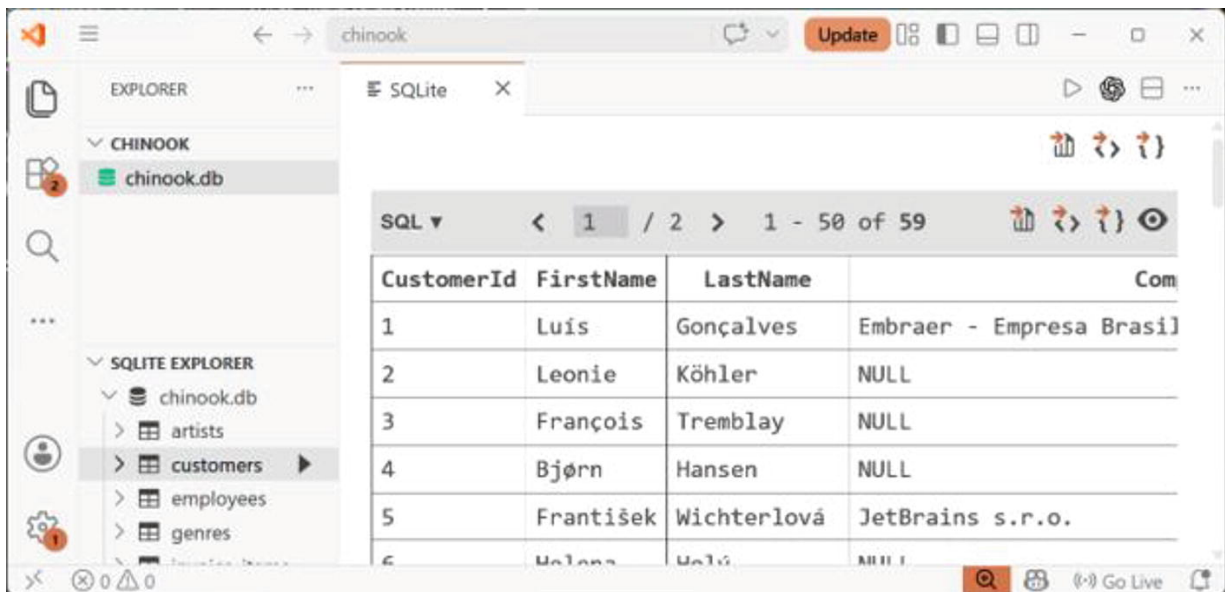


Figure 15-2 SQLite extension for VS Code

You may also experiment with the SQLite viewer extension.

SQLite3 Fiddle

The official website of the SQLite library

(<https://sqlite.org/fiddle/>) has a very useful online tool called SQLite3 Fiddle. It runs inside your browser, so you don't need to download and install the library locally. See Figure [15-3](#).

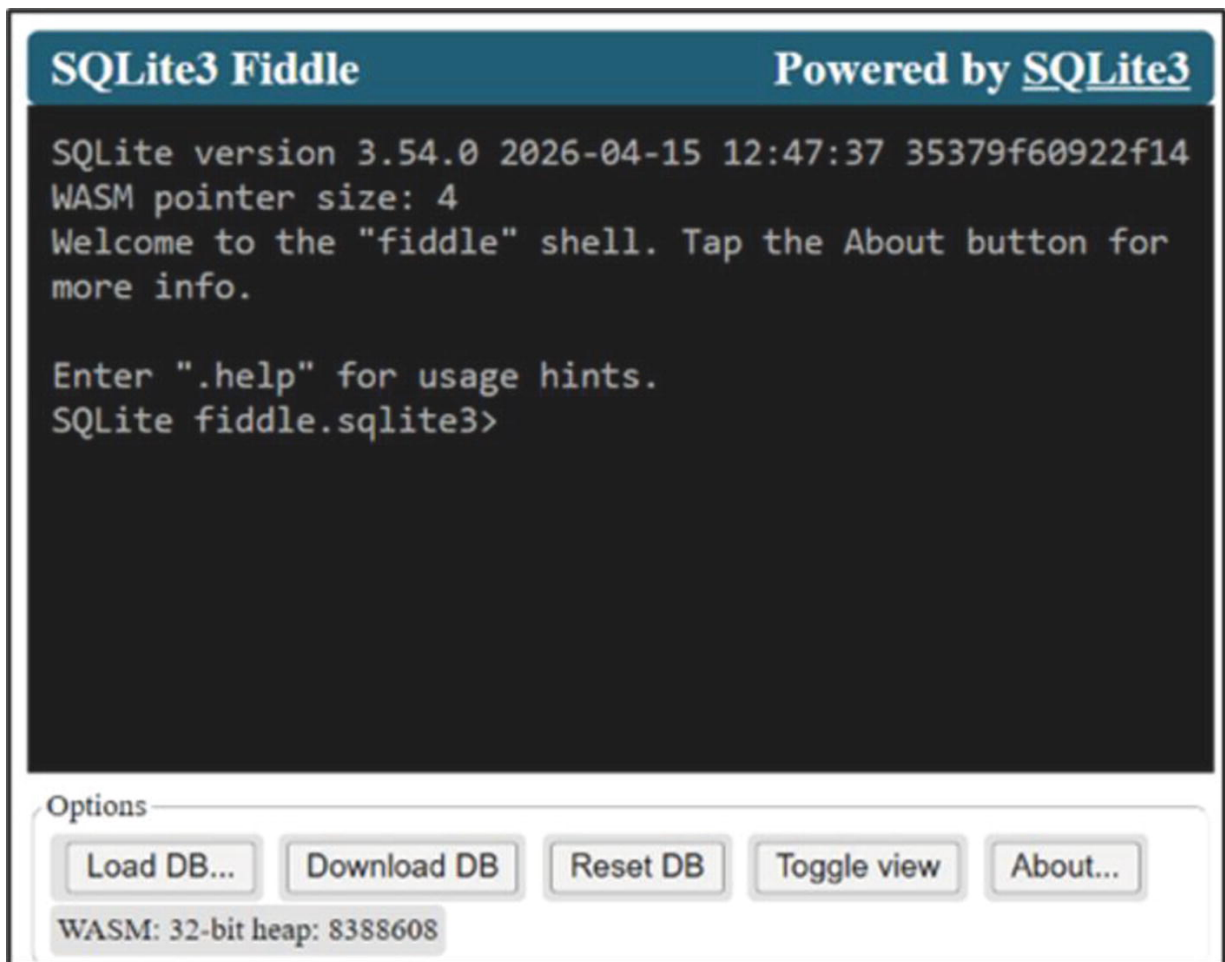


Figure 15-3 SQLite3 Fiddle

SQLite Command-Line Shell

The `sqlite3` application included in the bundle is a command-line interface to the SQLite library. You can perform all the SQL operations from inside the command terminal of your operating system.

Create Database

To open a new database, use its name (as a convention, use `.sqlite` as the file extension) in the command.

```
$ sqlite3 mydb.sqlite
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite>
```

If the database with the given name is already present, it will be opened. If not, a new database is created.

Your terminal now enters the SQLite shell with the `sqlite>` prompt, before which you can enter and execute any valid SQL statement. Additionally, SQLite provides its own commands that are prefixed with “.” (dot). For example, to open an already existing database from within the shell, use the `.open` command. The `.tables` command lists the tables in the current database. To exit the shell, use the `.quit` or `.exit` command.

Create a New Table

CREATE TABLE is an important DDL statement in SQL. You need to specify the name of the table, followed by a comma-separated list of column names, respective data types, and constraints, if any, put inside parentheses. Don't forget to put a semicolon at the end of the statement.

```
CREATE TABLE table_name (  
    column1 datatype [width] [default]  
    [constraint],  
    column2 ...,  
    column3 ...,  
    ....  
);
```

Let's create a table named Employees, with the column attributes as FirstName, LastName, Title, and Phone of VARCHAR type and Salary a Numeric type column. EmployeeId is the primary key for the Employees table.

```
sqlite> CREATE TABLE Employees (  
    EmployeeId INTEGER PRIMARY KEY,  
    FirstName   VARCHAR (20) NOT NULL,  
    LastName   VARCHAR (20) NOT NULL,  
    Title      VARCHAR (30),  
    Phone      VARCHAR (24),
```



```
Salary    NUMERIC (8, 2)
);
```

The .tables command now shows this newly created table in the list. The preceding SQL statement uses two constraint clauses.

PRIMARY KEY: The SQLite engine ensures that the value of this column for each record is unique. You can also use the **AUTOINCREMENT** constraint (only if the data type is **INTEGER**) so that the value will be filled automatically, incrementing from 1 onward.

NOT NULL: If this constraint is specified, the column cannot be empty.

You can modify the structure of an existing table with the **ALTER TABLE** statement, and remove the table forever from the database with the **DROP TABLE** statement.

Insert a Record

Each row in the Employees table has the details of an employee. You need to execute the **INSERT INTO** statement to add a new row, as per the prescribed syntax.

```
INSERT INTO table_name (col1, col2, ...) VALUES
(val1, val2, ...);
```

The comma-separated list of columns must be enclosed in parentheses after the table name. The **VALUES** clause also uses a set of parentheses. The number of value items in it must match the number of columns. Also, the column's type and the corresponding value's type must match (especially for some RDBMS products like Oracle).

The following SQL statement adds a new employee record.

```
sqlite> INSERT INTO Employees (
        EmployeeId,
        FirstName,
```

```
LastName,  
Title,  
Phone,  
Salary)  
VALUES (  
1,  
"Luis",  
"Goncalves",  
"Manager",  
"555-1234",  
75000);
```

SELECT Statement

SQL helps you retrieve data from a table with the SELECT statement. The SELECT query fetches values in the specified columns (from all records or only those that meet certain conditions) of the given table.

Here's the general syntax for executing the SELECT query.

```
SELECT column1, column2, ... FROM table_name;
```

Several other optional clauses are also used with the SELECT statement. Most commonly used clauses are WHERE (filters retrieved records based on specific conditions), ORDER BY (sorts the result set), and GROUP BY (arranges the rows on a given expression and adds a summary row).

If you need to fetch the values from all the fields in a table, you can simply use * instead of the list of columns.

```
sqlite> SELECT * from Employees;
```

Another variation of the SELECT statement retrieves the FirstName and Title of employees with salaries above 50000.

```
sqlite> SELECT FirstName, Title, Salary from  
Employees
```

```
...> WHERE Salary>50000;
```

You can construct complex filters with the AND/OR operators in the WHERE clause.

UPDATE Statement

Frequently, you may need to change the value of one or more columns of certain rows in a table. This is done with the UPDATE statement. Identify the row(s) to be updated with the WHERE clause and assign a new value to the desired field, as per the prescribed syntax.

```
UPDATE table_name SET col1=val1, col2=val2,...  
WHERE [filter];
```

Enter the following SELECT query in the SQLite shell to increase the salary of managers (Title="Manager") by 10 percent.

```
sqlite> UPDATE Employees SET  
Salary=Salary+Salary*0.1 WHERE Title="Manager";
```

In the UPDATE statement, the WHERE clause is not mandatory, but if it is not present, the update action is done for all records in the table.

Delete a Record

The DELETE statement must be used with caution, as it permanently removes the affected record from the table. Again, the precise WHERE filter should be included; if it is not, all the records in the table will be removed. The syntax for a DELETE query is very straightforward.

```
DELETE FROM table_name WHERE [filter];
```

The following statement, when executed in the SQLite shell, removes those employee records having Manager in the Title column.

```
sqlite> DELETE FROM Employees WHERE  
Title="Manager";
```

Getting acquainted with basic SQL queries is crucial as you use a SQLite database in a Node.js application with the `sqlite3` module.

sqlite3 Module

SQLite is a library of functions in C. The SQLite shell is the interface that calls these functions when you execute the SQL queries. Similarly, the `sqlite3` module hosted on the npm Registry acts as a binding/wrapper around the original SQLite C library, allowing JavaScript applications to access SQLite functionality through an object-oriented API.

Start a new [Node.js](#) project and install the `sqlite3` library.

```
$ mkdir NodeDBApp
$ cd NodeDBApp
$ npm init -y
$ npm install sqlite3
```

The project's configuration file (`package.json`) adds `sqlite3` as its dependency.

```
{
  "name": "nodedbapp",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" &&
exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "sqlite3": "^6.0.1"
  }
}
```

```
}
```

Connecting to a Database

To create a new (or open an existing) database and perform CRUD operations, you need to obtain the database connection object by invoking its constructor exposed by the `sqlite3` module.

```
const sqlite3 = require("sqlite3");  
  
const db = new sqlite3.Database("mydb.sqlite");
```

If the entry point of your Node project is [index.js](#), add these statements and run. You will find the database created in the project folder. Note that you may also create an in-memory database, which is active in memory (RAM) only when the application is running.

```
const db = new sqlite3.Database(":memory:");
```

Database Connection Object

The connection object acts as a handle to the database it is referring to. Using its various methods, you can programmatically perform the CRUD operations.

db.run()

This method needs a mandatory string parameter, which is a SQL query (CREATE, INSERT, UPDATE, or DELETE) to be executed. It's an asynchronous method, so it also accepts an optional callback parameter. This method doesn't have any return value.

```
db.run(SQL_string, callback);
```

Create a Table

Let's pass a string that represents the SQL statement for creating the Employees table (with the same structure as before). Add the following

code in the project's index.js file.

```
db.run(`CREATE TABLE IF NOT EXISTS Employees (  
  EmployeeId INTEGER PRIMARY KEY,  
  FirstName   VARCHAR (20) NOT NULL,  
  LastName   VARCHAR (20) NOT NULL,  
  Title      VARCHAR (30),  
  Phone      VARCHAR (24),  
  Salary     NUMERIC (8, 2)  
)`, (err) => {  
  if (err) {  
    console.error("Error creating table:",  
err.message);  
  } else {  
    console.log("Table created successfully");  
  }  
});
```

Run the code after adding this snippet, and check if the Employees table has been created. You can use any of the tools.

Insert a Record

The same `db.run()` method performs the insert operation if the string parameter is the representation of the INSERT query.

```
db.run(`INSERT INTO Employees (  
  EmployeeId, FirstName, LastName, Title,  
  Phone, Salary)  
  VALUES (  
    1, "Luis", "Goncalves", "Manager", "555-  
1234", 75000)`,  
  (err) => {  
    if (err) {  
      console.error("Error inserting record:",  
err.message);  
    }  
  });
```

```
    } else {  
        console.log("Record inserted successfully");  
    }  
});
```

The insert operation can be made more flexible with a variation of `db.run()` syntax that lets you pass the query parameters to the SQL query statement.

```
db.run(sql, params, callback)
```

Here, `params` is an array object. The SQL query string uses '?' placeholders in the VALUES clause. These placeholders are substituted at runtime by the respective items in the `params` array.

```
const insertQry = `INSERT INTO Employees  
(FirstName, LastName, Title, Phone, Salary)  
VALUES (?, ?, ?, ?, ?)`;  
  
const data = ["Luis", "Goncalves", "Manager",  
"555-1234", 75000];  
  
db.run(insertQry, data, function (err) {  
    if (err) {  
        console.error("Error inserting record:",  
err.message);  
    } else {  
        console.log("Record inserted successfully");  
    }  
});  
  
db.close();
```

Note that after the insert operation is done, the `db.close()` method must be called.

Modify a Record

You need to execute the UPDATE query to modify the values of one or more columns of those records that satisfy the given filter condition. Since the new values and those in the filter expression may vary, it is best to use a parameterized query.

The following code snippet runs the UPDATE query to assign a new value to the Salary column of all manager employees.

```
const updateQry="UPDATE Employees SET Salary=?  
WHERE Title=?";  
const data = [80000, "Manager"];  
  
db.run(updateQry, data, function (err) {  
  if (err) {  
    console.error("Error updating record:",  
err.message);  
  } else {  
    console.log(`Updated the record`);  
  }  
});  
  
db.close();
```

Remove a Record

As you know, the DELETE query permanently removes records from the SQLite table that satisfy the filter criteria specified in the WHERE clause. The query string is passed as a parameter to the db.run() method defined in the sqlite3 module. If it's a parameterized query, pass the values for the filter expression as the params array. The following Node.js code removes rows from the Employees table that have "Manager" in the Title column.

```
const deleteQry="DELETE FROM Employees WHERE  
Title=?";  
const data = ["Manager"];
```



```
db.run(deleteQry, data, function (err) {
  if (err) {
    console.error("Error deleting record:",
err.message);
  } else {
    console.log(`Deleted the record`);
  }
});

db.close();
```

The `db.run()` method is used to run queries that perform certain write operations, such as creating a table or inserting/updating/deleting records. To read data from a SQL table, the database connection object provides three methods: `db.get()`, `db.all()`, and `db.each()`

db.close()

It is highly desirable that the database connection be explicitly closed to release resources such as file handles, internal buffers, and other native resources. Closing the connection also ensures that all pending asynchronous database operations are completed in an orderly manner. This helps avoid resource leaks and ensures proper cleanup of the database connection.

The `close()` method is called on a database instance and can optionally accept a callback function that executes when the connection is successfully closed or if an error occurs.

```
db.close((err) => {
  if (err) {
    return console.error(err.message);
  }

  console.log(
```

```
        "Closed the database connection."
    );
});
```

db.get()

The first parameter to the `db.get()` method is a string representing a `SELECT` query. You may optionally pass the `params` array and a callback function. The `db.get()` method returns a JavaScript object corresponding to the first row of the query result set (even if the `SELECT` statement might fetch multiple rows).

The `SELECT` query in the following code returns all the employee records whose `Title` column equals `Manager`. However, the `db.get()` method returns a JavaScript object with the columns as keys and contents as values.

```
const selectQry = "SELECT * FROM Employees WHERE
Title=?";
const data = ["Manager"];

db.get(selectQry, data, function (err, row) {
    if (err) {
        console.error("Error fetching record:",
err.message);
    } else {
        console.log("Fetched record:", row);
    }
});

db.close();
```

The JavaScript object would be as follows.

```
Fetched record: {
  EmployeeId: 1,
```

```
    FirstName: 'Luis',  
    LastName: 'Goncalves',  
    Title: 'Manager',  
    Phone: '555-1234',  
    Salary: 75000  
}
```

Note that the call to `db.get()` always expects a single row to be returned. This is especially useful when you fetch a record with its primary key.

db.all()

Compared to the `get()` method, a call to `db.all()` returns the set of all records that satisfy the given `SELECT` query. Hence, the two parameters passed to the callback functions are `err` and an array of JS objects; each object represents one row in the table.

Let's keep the `SELECT` query string the same and pass it to the `db.all()` method.

```
const selectQry = "SELECT * FROM Employees WHERE  
Title=?";  
const data = ["Manager"];  
db.all(selectQry, data, (err, rows) => {  
    if (err) {  
        console.error(err.message);  
    } else {  
        console.log("Fetched records:", rows);  
    }  
});  
db.close();
```

Assuming there are multiple employees with the title of Manager, you might see an array of records displayed in the console.

```

Fetched records: [
  {
    EmployeeId: 1,
    FirstName: 'Luis',
    LastName: 'Goncalves',
    Title: 'Manager',
    Phone: '555-1234',
    Salary: 75000
  },
  {
    EmployeeId: 3,
    FirstName: 'Andrew',
    LastName: 'Adams',
    Title: 'Manager',
    Phone: '455-2143',
    Salary: 70000
  }
]

```

db.each()

The `db.each()` method fetches one row at a time from the result set of the SQL query string passed to it. Unlike `db.run()`, `db.get()`, and `db.all()`, `db.each()` can optionally accept a fourth argument: a completion callback that executes after all rows in the result set have been processed.

```
db.each( sql, params, rowCallback,
completionCallback );
```

The `rowCallback` function is called as the next row is fetched from the database. `completionCallback` is called only when all the rows are processed. It receives the count of rows in the result set.

```
db.each (
```

```

    query,
    params,
    (err, row) => {
        //process each row;
    },

    (err, count) => {
        //process end of rows;
    }
);

```

For small or moderate datasets, using `db.all()` to retrieve the complete query result as an array of JavaScript objects before processing may be convenient and efficient. However, it will consume significant memory for large result sets. This is where `db.each()` is preferred, as it fetches and processes one row at a time through callbacks, avoiding the need to load the entire result into memory. This method is more memory-efficient and better suited for large datasets.

The following is the `db.each()` call in action. The parameterized SQL query is the same as before.

```

console.log("Employees with Title as Manager...");

db.each(
    selectQry,
    data,
    (err, row) => {
        if (err) {
            console.log("Error fetching row");
        } else {
            console.log(`First Name:
${row.FirstName}, Salary: ${row.Salary}`);
        }
    },

```

```
(err, count) => {
    if (err) {
        console.log("Error fetching rows");
    } else {
        console.log(`Total rows processed:
${count}`);
    }
}
);

db.close();
```

The output may be something similar.

```
Employees with Title as Manager...
First Name: Luis, Salary: 75000
First Name: Andrew, Salary: 70000
Total rows processed: 2
```

Transaction Control

Often, a database operation may involve multiple SQL queries executed one after another, where a subsequent query depends on the success of a previous query. Such related operations should be treated as a single transaction.

Databases follow the ACID properties.

A – Atomicity: A transaction should either complete fully or not happen at all.

C – Consistency: Transactions prevent the database from entering an invalid state.

I – Isolation: One transaction should not see incomplete work of another transaction.

D – Durability: Once committed, changes survive failures such as power loss or system crashes.

SQLite supports transaction control statements.

- BEGIN TRANSACTION
- COMMIT
- ROLLBACK

These statements ensure that multiple database operations can be treated as a single logical unit. If all operations execute successfully, COMMIT permanently saves the changes. If any operation fails, ROLLBACK cancels the transaction and restores the database to its previous state.

Other SQLite Modules

Besides the popular `sqlite3` module available on the npm Registry, Node.js offers other ways to work with SQLite databases. One commonly used alternative is `better-sqlite3`, which is also available on the npm Registry and provides a synchronous API. It is often considered simpler and faster because it avoids callback overhead and offers a more straightforward programming model.

Recent versions of Node.js (22.5.0 onward) include the built-in `node:sqlite` module, so there's no need to install the npm package. It provides native SQLite support, integrating the commonly used functionality into the runtime itself. Like `better-sqlite3`, it primarily offers a synchronous programming model.

These libraries are direct SQLite bindings that provide access to the underlying SQLite database engine. Alternatively, ORM-based solutions such as `Sequelize` and `Prisma` can be used. Their object-oriented API provides a higher level of abstraction by generating SQL queries internally.

SQL Injection

SQL queries often contain variable data. For example, data values in the VALUES clause of an INSERT statement or in conditions of the WHERE clause in SELECT, UPDATE, or DELETE statements.

Consider the following SELECT query.

```
SELECT * FROM users WHERE USER='admin' AND  
PASSWORD='secret';
```

Assuming that if this query returns a row, the user is allowed to log into the application. In practice, these values often come from user input. In the Node code, they are put in variables.

```
const name = "admin";  
const pwd = "secret";
```

The SQL query is constructed using variables.

```
`SELECT * FROM users WHERE USER='${name}' AND  
PASSWORD='${pwd}'`;
```

The term *SQL injection* refers to the possibility that user input may contain SQL syntax or SQL logic. When such input is concatenated directly into an SQL query string, it effectively “injects” SQL code through user input and alters the intended meaning of the query. This is especially possible when data comes from HTML forms (or other sources of user input).

For example, in the password field of a login form, a user may enter the following.

```
' OR '1'='1
```

The concatenated query effectively becomes the following.

```
SELECT * FROM users WHERE USER='admin' AND  
PASSWORD='' OR '1'='1'
```

Note that the database engine cannot distinguish between the SQL written by the programmer and the SQL logic injected by the attacker. Look at the following.


```
OR '1'='1'
```

This is a SQL condition fragment inserted through user input that is always true. As a result, the login process may succeed irrespective of the password.

SQL injection can prove to be extremely harmful. Imagine the SQL statement “DROP TABLE users;” being injected by the user input.

Using parameterized queries is the safest way to prevent such attacks. Never build SQL queries using string concatenation with user input. Use parameterized queries instead. Use placeholders such as ? in the SQL string and pass parameter values separately. (This technique has already been used in the code snippets explaining the database object methods: `db.run()`, `db.get()`, `db.all()`, etc.).

```
db.get (
    "SELECT * FROM users WHERE USER=? AND
    PASSWORD=?",
    [name, pwd]
);
```

How do placeholders prevent SQL injection? The database engine first parses and learns the SQL query structure, and then binds the parameter values separately. It does not reinterpret user input as SQL syntax. Even if the input looks like SQL code, once passed as a parameter, it is treated as data and cannot change the meaning of the query.

SQLite in Express App

Most applications—desktop as well as web applications—need the data generated during user interaction and processing (such as users, employees, products, blog posts, and so on) to be stored persistently in a back-end database. In a web application, users typically interact with it through a browser. REST APIs, on the other hand, are designed for programmatic interaction between software systems.

SQLite is a “lightweight” database. Developers often use it as a back-end choice, typically to build a prototype (POC, or proof of concept), eventually to be substituted by certain enterprise database products such as MySQL or PostgreSQL.

While responding to incoming HTTP requests, conventional web applications usually render an HTML response, whereas REST APIs generally return data in JSON format. You have already learned that the Express framework employs a routing mechanism that directs incoming requests to a specific handler function depending upon the type of HTTP request (GET, POST, etc.) and the path parameters passed in the request URL.

The four principal HTTP methods map naturally to the CRUD operations on a SQL table.

POST: Create (INSERT query)

GET: Read (SELECT query)

PUT: Update (UPDATE query)

DELETE: Delete (DELETE query)

In an Express application, routes are defined using methods of the application object, where the method name itself indicates the type of HTTP request.

```
app.method(path, callback);
```

Accordingly, `app.get()` defines a route that processes GET requests, `app.post()` processes POST requests, and similarly for the other HTTP methods.

A REST application usually follows a consistent route-design approach. For an Employees table, the following routes may be defined in a typical Express-based REST API.

```
app.get('/employees', callback);
```

This route retrieves all rows from the Employees table. The callback internally executes the `db.all()` method on the underlying SQLite

database.

```
app.get('/employees/:id', callback)
```

This route retrieves the record corresponding to the EmployeeID passed as a path parameter. The callback typically uses the `db.get()` method.

```
app.post('/employees', callback)
```

The callback registered with this route processes the request body obtained from the client's POST method. It internally executes the INSERT query with the `db.run()` method.

```
app.put('/employees/:id', callback)
```

The PUT request containing modified column values for a specific employee record is routed to the callback that performs an UPDATE query.

```
app.delete('/employees/:id', callback)
```

The corresponding callback removes the corresponding employee row by executing a DELETE query.

Thus, in a persistent REST API, Express routes act as a bridge between HTTP requests and database operations. Incoming requests are translated into SQL operations, the database performs the required processing, and the results are returned to the client as JSON responses.

Points to Remember

- Web applications use relational databases for performing persistent CRUD operations.
- A database is a structured and logical organization of data that overcomes redundancy and lack of data integrity.

- A relational database organizes data in entity tables. Each table has a pre-defined schema of columns, their data type, and length. Each row is an instance of the entity, and the column represents an attribute.
- SQLite, MySQL, and PostgreSQL are some examples of RDBMS products. SQLite is a lightweight, serverless database stored in a single portable file.
- SQL is designed to create databases and tables and perform insert/update/delete operations on them.
- In a SQL table schema, one column that stores a unique value for each row is called a primary key.
- Tables can be related based on common attributes. An attribute of a table that refers to the primary key of another table is called the foreign key.
- SQL has its own nomenclature of numeric, character, date/time, and binary types.
- SQLite is a free and open source library written in C. Its precompiled binary distribution offers a command-line interface to execute SQL queries. Different GUI utilities like DB Browser are built on top of this library.
- The following are important SQL statements.
 - `CREATE TABLE table_name (column1 datatype [width] [constraint], column2, ...,);`
 - `INSERT INTO table_name (col1, col2, ...) VALUES (val1, val2, ...);`
 - `SELECT column1, column2, ... FROM table_name;`
 - `UPDATE table_name SET col1=val1, col2=val2,.. WHERE [filter];`
 - `DELETE FROM table_name WHERE [filter];`
- The `sqlite3` module on the npm Registry is a binding around the original C library that helps JavaScript applications programmatically access the SQLite functionality.
- Once installed in a Node.js project, you need to obtain a database connection object referring to a local SQLite database or an in-memory database.

- The `run()` method of the connection object, `db.run()`, executes the given query string and optionally executes a callback. It is used to programmatically perform CREATE, INSERT, UPDATE, and DELETE queries.
 - For a SELECT query, the `sqlite3` module has `db.get()`, `db.all()` and `db.each()` methods.
 - To retrieve a unique row that satisfies the filter condition, use `db.get()`, which returns a JS object.
 - To retrieve all rows in a given table as an array, use `db.all()`.
 - `db.each()` method processes each row in the result set of a given SQL query.
 - Other substitutes for the `sqlite3` module are `better-sqlite` and a built-in `node:sqlite` module found in newer versions of the Node.js runtime. Both are synchronous in nature.
 - While designing the SQL queries, one has to be careful that a vulnerability called SQL injection doesn't creep into the application. It can occur if variable data to be included in the SQL query string is valid SQL logic.
 - The query string constructed by concatenation may cause damage to the database.
 - Use of parameterized queries prevents the possibility of SQL injection.
 - Express-powered web apps and REST APIs use routing. The routing methods corresponding to the HTTP methods are mapped to SQL queries that perform CRUD operations.
-

Summary

This is the last chapter of this book and is intended to be a study companion when preparing for Web Development Essentials certification conducted by LPI. This chapter aligns with the last topic in the exam's objectives: SQL basics.

Starting with RDBMS concepts, it explains installing SQLite and using its command-line interface to execute basic SQL queries. The

main focus of this chapter is how to use the sqlite3 module to programmatically perform CRUD operations on a SQLite database.

A very crucial concept of SQL injection and how to prevent the same with parameterized queries has been explained in this chapter.

Although not mentioned explicitly in the LPI objectives, the discussion on transaction control and developing a REST API will be helpful for the journey ahead of a prospective developer.

Knowledge Check: Part V

1. **What is Node.js?**
 - A. A client-side JavaScript library used to manipulate the DOM
 - B. A server-side runtime environment built on Google Chrome's V8 JavaScript engine
 - C. A RDBMS designed for asynchronous queries
 - D. A CSS preprocessor used to optimize web page delivery layout templates
2. **Which core module in Node.js provides a way of handling file system operations?**
 - A. fs
 - B. url
 - C. path
 - D. http
3. **How do you import a built-in module like http into a standard CommonJS Node.js script?**
 - A. `import http from 'http';`
 - B. `const http = require('http');`
 - C. `const http = include('http');`
 - D. `fetch('http');`

`return req;`

4. **Which command is used to install Express and save it as a dependency in your local package.json file?**
 - A. `npm install express`
 - B. `npm download express`
 - C. `node get express`
 - D. `npm initiate express`

5. **In an Express application, which function call is used to bind and listen for connections on a specified host and port?**
 - A. `app.start(3000, 'localhost')`
 - B. `app.connect(3000, '127.0.0.1')`
 - C. `app.listen(3000, 'localhost')`
 - D. `app.serve(3000, '127.0.0.1')`

6. **What do the parameters `req` and `res` typically represent in an Express route handler callback function?**
 - A. Requirement and Resolution
 - B. HTTP Request and Response objects
 - C. Regular Expression and Result String
 - D. Redirect and Restart

7. **If an incoming GET request is sent to /users?sort=asc&limit=10, how can you extract the value of limit inside your Express route handler?**
- A. req.params.limit
 - B. req.body.limit
 - C. req.query.limit
 - D. req.url.limit
8. **Consider the route path definition: app.get('/products/:id', ...). How would you access the :id parameter passed by the client's URL?**
- A. req.query.id
 - B. req.params.id
 - C. req.body.id
 - D. req.headers.id
9. **Which of the following is a valid EJS tag used to render an expression as part of browser output?**
- A. <% expr %>
 - B. <%= expr %>
 - C. <# expr %>
 - D. <#= expr %>

10. **What is the fundamental role of middleware functions in Express?**
- A. To store persistent rows inside relational database systems
 - B. To compile front-end JavaScript files into raw machine code instructions
 - C. To access and manipulate the request and response objects before passing control to the next handler
 - D. To replace standard CSS stylesheets with layout engine variables
11. **Which built-in middleware function in Express parses incoming requests with JSON payloads?**
- A. `app.use(express.parseJSON())`
 - B. `app.use(express.json())`
 - C. `app.use(express.bodyParser())`
 - D. `app.use(express.static())`
12. **How does an Express application pass control to the next middleware function in line?**
- A. By returning true at the end of the current function block
 - B. By calling the `res.end()` method
 - C. By invoking the `next()` function parameter
 - D. By throwing a custom system exception error

13. **Which middleware is used to serve static assets like HTML files, images, and CSS files from a directory named “public”?**
- A. `app.use(express.static('public'))`
 - B. `app.use(express.assets('public'))`
 - C. `app.get(express.directory('public'))`
 - D. `res.sendFile('public')`
14. **Which HTTP request method most securely creates a new resource record in a web API?**
- A. GET
 - B. PUT
 - C. POST
 - D. DELETE
15. **Which of the following commands creates an Express application skeleton using the express-generator tool?**
- A. `npm init express`
 - B. `express <app-name>`
 - C. `node express-generator <app-name>`
 - D. `npx express-generator <app-name>`

16. **In Express, how can you intercept and handle an error thrown inside a route?**
- A. By defining a middleware function that takes four arguments: (err, req, res, next)
 - B. By wrapping the entire server.js file script with an external system-level try...catch
 - C. By running the command-line tool npm fix error
 - D. Express handles all custom programmatic runtime errors by default by deleting the database data
17. **Which clause is used in a SQL statement to filter records based on a specific condition?**
- A. HAVING
 - B. WHERE
 - C. GROUP BY
 - D. FILTER
18. **What does the INSERT INTO command do in an SQL database?**
- A. Modifies columns inside an existing table row structural framework
 - B. Creates an empty new table schema within a project
 - C. Adds one or more new record rows to an existing database table

table

- D. Permanently cleans out all columns from a data store instance

19. **Which SQL constraint ensures that a particular field has a unique value for each row in a database table?**
- A. UNIQUE
 - B. FOREIGN KEY
 - C. PRIMARY KEY
 - D. NOT NULL
20. **When using the Node.js sqlite3 library, which method is best suited for executing an SQL query where you do not expect to fetch rows back?**
- A. db.all()
 - B. db.get()
 - C. db.run()
 - D. db.each()
21. **Which of the following connection object methods retrieves only the first matching row as a JavaScript object?**
- A. db.all()
 - B. db.get()
 - C. db.first()

D.
db.queryRow()

22. **How do you prevent SQL injection?**

- A. By using parameterized queries and prepared statements
- B. By encrypting the database password
- C. By creating an in-memory database
- D. By using node:sqlite module

23. **Look at the following expression:**

**db.run("UPDATE users SET email = ? WHERE id = ?",
[newEmail, userId]).**

What purpose do the ? characters serve?

- A. Wildcard query parameters that match any value within text rows
- B. Non-standard optional JavaScript syntax markers
- C. Placeholders for parameterized data strings that prevent SQL injection
- D. Conditional true/false ternary operational logic parameters

24. **Which of the following is not a clause used in a SQL statement?**

- A. ORDER BY
- B.

- GROUP BY
- C. SORT BY
- D. WHERE
25. **Which statement removes an entire table along with its data records permanently from the database?**
- A. DELETE TABLE users;
- B. DROP TABLE users;
- C. TRUNCATE TABLE users;
- D. REMOVE TABLE users;
-

Answers with Explanations

1. **B**
Explanation: Node.js is an open source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a web browser, running primarily on the server-side via the V8 engine. (Chapter [13](#))
2. **A**
Explanation: The fs module defines functions that can access the file system; for example, the readFile() function fetches the contents of a file. (Chapter [13](#))
3. **B**
Explanation: In CommonJS, modules are imported synchronously using the built-in require() function. (Chapter [13](#))

4. **A**

Explanation: Running `npm install express` downloads the Express package from the npm Registry, saves it inside the `node_modules` folder, and adds it to your `package.json` dependency manifest. (Chapter [14](#))

5.

C

Explanation: The `app.listen()` method spins up an HTTP server and begins accepting client connections at the specified port and host. (Chapter [14](#))

6.

B

Explanation: `req` stands for the incoming HTTP Request object, and `res` stands for the outgoing HTTP Response object that the server sends back to the client. (Chapter [14](#))

7.

C

Explanation: URL query parameters (the key-value pairs following the `?` character) are automatically parsed by Express and populated inside the `req.query` object. (Chapter [14](#))

8.

B

Explanation: Route parameters (tokens named with a leading colon `:` symbol) are stored inside the `req.params` object. (Chapter [14](#))

9.

B

Explanation: Any expression inside the `<%=` and `%>` is rendered as part of the browser output. (Chapter [14](#))

10.

C

Explanation: Middleware functions have access to the request object (`req`), the response object (`res`), and the next middleware function in the application's request-response cycle. (Chapter [14](#))

11. **B**

Explanation: `express.json()` is a built-in middleware function that

parses incoming requests containing JSON payloads and places the parsed object data directly onto req.body. (Chapter [14](#))

12.

C

Explanation: If the current middleware function does not end the request-response cycle, it must call next() to hand over execution control to the subsequent middleware function. (Chapter [14](#))

13.

A

Explanation: express.static is a built-in middleware function in Express that serves static configuration assets directly from a target folder. (Chapter [14](#))

14.

C

Explanation: In standard RESTful architecture, the data submitted by an HTTP POST request is utilized to create a new resource, typically a record in a SQL table. (Chapter [15](#))

15.

D

Explanation: express-generator is a useful command-line tool. It scaffolds a typical folder structure of a ready-to-use Express project. (Chapter [14](#))

16.

A

Explanation: Error-handling middleware functions are defined precisely like normal middleware functions, except they accept four arguments instead of three: err, req, res, next. (Chapter [14](#))

17.

B

Explanation: The WHERE clause specifies that an SQL statement should only select, modify, or delete matching rows that satisfy a declared condition. (Chapter [15](#))

18. **C**

Explanation: INSERT INTO is a DML statement that appends new records (rows) into a designated database table target. (Chapter [15](#))

19. **C**
Explanation: A PRIMARY KEY uniquely identifies every row in a table. It must contain unique values, and it cannot contain NULL values. (Chapter [15](#))
20. **C**
Explanation: db.run() executes an SQL query and does not return any data rows. It is ideal for operational tasks like schema configuration changes, INSERTs, UPDATEs, and DELETEs. (Chapter [15](#))
21. **B**
Explanation: db.get(query, params, callback) executes the SQL query and passes the first matching row to the callback function as a single JavaScript object. If no rows match, it returns undefined. (Chapter [15](#))
22. **A**
Explanation: With parameterized queries, the database engine treats user input strictly as a literal value, never as executable code. They are the primary defense against SQL injection. (Chapter [15](#))
23. **C**
Explanation: The ? symbols act as parameterized positional placeholders. The driver safely passes separate variables into the query layout engine, neutralizing SQL injection attempts. (Chapter [15](#))
24. **C**
Explanation: SQL uses ORDER BY for sorting. GROUP BY groups rows by a given expression and adds a summary row. WHERE specifies filter criteria. (Chapter [15](#))
- B**

25. *Explanation: DROP TABLE deletes the entire table structure, including its data, permanently from the schema. (Chapter [15](#))*

Model Test Paper I

- Questions: 40
- Duration: 60 minutes
- Single-response questions require selecting exactly one correct choice (A, B, C, or D).
- Multiple-response questions explicitly indicate requirements (e.g., TWO correct options).
- No partial credit is awarded.

Q1.

Which HTML element is used to link an external CSS stylesheet to an HTML document?

- A. `<style>`
- B. `<script>`
- C. `<link>`
- D. `<css>`

Q2.

Which SQL statement is used to retrieve all columns from a table named users where the age column is greater than 18?

- A. `FETCH * FROM users WHERE age > 18`
- B. `GET * FROM users WHERE age > 18`
- C. `SELECT * FROM users WHERE age > 18`
- D. `READ * FROM users WHERE age > 18`

Q3.

Which <form> attribute specifies the URL to which the form data is sent on submission?

- A. method
- B. enctype
- C. action
- D. name

Q4.

What does the JavaScript expression `typeof null` evaluate to?

- A. "null"
- B. "undefined"
- C. "object"
- D. "symbol"

Q5.

Which of the following are valid values for the type attribute of a <button> element? (TWO correct options)

- A. image
- B. submit
- C. check
- D. reset

Q6.

Which CSS pseudo-class applies styles to an element when the user moves their pointer over it?

- A. :focus
- B. :active
- C. :visited
- D. :hover

Q7.

A developer wants to expose application data to third-party developers over the web using a standardized, stateless interface. Which technology is most appropriate?

- A. WebAssembly
- B. REST API
- C. a content management system
- D. a virtual host

Q8.

Which keyword declares a block-scoped variable in JavaScript that cannot be reassigned?

- A. var
- B. let
- C. const

D. static

Q9.

Which EJS tag is used to output the value of a variable into the HTML response while escaping special characters?

- A. `<% variable %>`
- B. `<%- variable %>`
- C. `<%# variable %>`
- D. `<%= variable %>`

Q10.

Which meta tag attribute is used to declare the character encoding of an HTML document?

- A. content
- B. name
- C. charset
- D. encoding

Q11.

Which loop structure is most appropriate when the number of iterations is known in advance?

- A. while
- B. for

C. switch

D.
if...else

Q12.

Which of the following are examples of relational (SQL) database systems? (TWO correct options)

A.
PostgreSQL

B.
MongoDB

C.
Oracle

D.
Redis

Q13.

Which HTML attribute is used to provide alternative text for an image when the image cannot be displayed?

A.
title

B.
src

C.
alt

D.
href

Q14.

A web server responds with HTTP status code 301. What does this indicate?

A.
The request was successful.

- B. The client is not authorized to access the resource.
- C. The resource has been permanently moved to a new URL.
- D. The server encountered an internal error.

Q15.

Which JavaScript method returns the first element in the document that matches a given CSS selector?

- A. `document.getElementsByTagName()`
- B. `document.getElementsByClassName()`
- C. `document.getElementById()`
- D. `document.querySelector()`

Q16.

Which tag is used to create an unordered (bulleted) list?

- A. `<ol>`
- B. `<ul>`
- C. `<li>`
- D. `<list>`

Q17.

How does an Express route handler access data submitted via an HTML form using the POST method?

- A. `req.query`

- B. req.params
- C. req.body
- D. req.headers

Q18.

In a JavaScript switch statement, which keyword prevents execution from continuing into the next case block?

- A. exit
- B. stop
- C. return
- D. break

Q19.

Which CSS selector targets only <p> elements that have the class note?

- A. p #note
- B. p, .note
- C. p.note
- D. .note p

Q20.

Which command installs a Node.js package named express and records it as a dependency in package.json?

- A. node install express
- B. npm install express
- C. npm init express
- D. node get express

Q21.

What does the following JavaScript function return?

```
function add(a, b) {  
    return a + b;  
}  
add(3, 4);
```

- A. undefined
- B. "34"
- C. 7
- D. NaN

Q22.

Which sqlite3 NPM module method executes an SQL query and passes ALL resulting rows to a callback function?

- A. db.get()
- B. db.run()

C. `db.each()`

D. `db.all()`

Q23.

Which declaration must appear at the very beginning of an HTML5 document?

A. `<html lang="en">`

B. `<meta charset="UTF-8">`

C. `<!DOCTYPE html>`

D. `<head>`

Q24.

Two of the following are valid JavaScript primitive data types. Which are they? (TWO correct options)

A. array

B. Boolean

C. object

D. symbol

Q25.

A developer wants an element's font size to scale relative to the root element's font size. Which CSS unit should be used?

A. `em`

- B. px
- C. %
- D. rem

Q26.

Which HTTP method is most appropriate when submitting a login form that contains a password?

- A. GET
- B. POST
- C. PUT
- D. HEAD

Q27.

In an Express application, which method is used to define a route that handles HTTP GET requests to the path /home?

- A. `app.post('/home', ...)`
- B. `app.use('/home', ...)`
- C. `app.get('/home', ...)`
- D. `app.route('/home', ...)`

Q28.

Which security vulnerabilities should a developer be aware of when processing user-submitted data in an Express application? (TWO correct options)

- A. SQL normalization
- B. Cross-site scripting (XSS)
- C. TCP flooding
- D. Cross-site request forgery (CSRF)

Q29.

Which HTML element is used to embed an external JavaScript file in a web page?

- A. <javascript>
- B. <js>
- C. <script>
- D. <link>

Q30.

What is the output of the following code?

```
let x = 5;
if (x == "5") {
  console.log("loose");
} else {
  console.log("strict");
}
```

- A. strict
- B. loose

- C. undefined
- D.
An error is thrown

Q31.

Which of the following best describes the role of a compiler?

- A.
It executes source code line by line at runtime
- B.
It translates the entire source code into machine code before execution
- C.
It manages memory allocation during program execution
- D.
It provides a graphical interface for writing source code

Q32.

Which CSS display value is used to arrange child elements in a flexible one-dimensional layout?

- A.
block
- B.
inline
- C.
none
- D.
flex

Q33.

Which CSS property is used to set the background color of an HTML element?

- A.
color
- B. fill

- C. background-color
- D. bg

Q34.

A developer wants to mark a section of text as strongly important so that screen readers and browsers convey emphasis. Which element is most semantically appropriate?

- A.
- B. <i>
- C. <u>
- D.

Q35.

Which of the following statements about HTTP cookies are correct? (TWO correct options)

- A. Cookies are stored permanently on the server
- B. Cookies can be used to maintain session state across multiple requests
- C. Cookies are transmitted inside the HTTP body by default
- D. Cookies can have an expiry date set by the server

Q36.

Which statement is correct syntax to add a CSS class active to a DOM element stored in the variable el?

- A. `el.style.add("newClass")`
- B. `el.setAttribute("newClass")`
- C. `el.classList.add("newClass")`
- D. `el.class = "newClass"`

Q37.

Consider the following CSS.

```
p { color: blue; }  
p { color: red !important; }  
p { color: green; }
```

What color will the paragraph text appear?

- A. blue
- B. red
- C. green
- D. the browser default

Q38.

A hyperlink should open its target URL in a new browser tab. Which value of the target attribute achieves this?

- A. `_self`
- B. `_parent`

- C. `_top`
- D. `_blank`

Q39.

If you want an element to remain in place relative to the viewport even when the page is scrolled. Which position value should you use to achieve this?

- A. `relative`
- B. `absolute`
- C. `static`
- D. `fixed`

Q40.

In Node.js, when using the `sqlite3` module, which technique should be used to prevent SQL injection when including user-supplied values in a query?

- A. Wrapping values in double quotes within the SQL string
- B. Using parameterized queries with `'?'` placeholders
- C. Converting all values to integers before use
- D. Storing all queries in a separate configuration file

Model Test Paper II

- Questions: 40
- Duration: 60 minutes
- Single-response questions require selecting exactly one correct choice (A, B, C, or D).
- Multiple-response questions explicitly indicate requirements (e.g., TWO correct options).
- No partial credit is awarded.

Q1.

Which of the following expressions refers to the third element in the items array?

- A. `items(2)`
- B. `items[3]`
- C. `items.3`
- D. `items[2]`

Q2.

Which of the following statements is true regarding the variable names and identifiers in JavaScript?

- A. Identifiers are not case-sensitive.
- B. Variable names and language keywords are case-sensitive in JavaScript.
- C. Only global variables are case-sensitive, not the local variables.
- D. A JavaScript identifier must start with an uppercase letter.

Q3.

Which DOM method retrieves all paragraph node elements from an active web document?

- A. `document.getElementById("p")`
- B. `document.getElementsByTagName("p")`
- C. `document.selectNodes("p")`
- D. `document.getParagraphs()`

Q4.

Which of the following statements are true about the middleware functions in an Express-powered application? (TWO correct options)

- A. They can execute code blocks, modify request or response objects, and terminate the processing sequence.
- B. They can invoke a subsequent middleware with the `next()` function pointer.
- C. They help render EJS templates.
- D. They parse the payload contents derived from `req.body`.

Q5.

How many times will a code block inside the following while loop be executed?

```
while (false) {  
  //code block;  
}
```

- A. An infinite number of executions.
- B. Exactly once.
- C. It will never be executed.
- D. A syntax error will be encountered.

Q6.

Which programming practice is recommended to prevent SQL injection attacks when using the sqlite3 module with Node.js?

- A. Interpolating the user-supplied data into the query string using template literals.
- B. Utilizing parameterized queries with '?' as placeholders and passing inputs in an argument array.
- C. By translating all characters manually into lowercase hex matrices.
- D. Enforcing server routers to accept database tasks solely from client admin tokens.

Q7.

What does the server's response with the HTTP status code 403 indicate?

- A. The resource has permanently moved to an unmapped path.
- B. The server met an unhandled runtime error.
- C. The server refused to process the request as the client lacks the necessary permissions.
- D. The network gateway timed out.

Q8.

Which of the following is the correct syntax to define a JavaScript object?

- A. `let userRecord = [name: "Alex", role: "Admin"];`
- B. `let userRecord = {name: "Alex", role: "Admin"};`
- C. `let userRecord = (name = "Alex", role = "Admin");`
- D. `let userRecord = "name": "Alex", "role": "Admin";`

Q9.

Which HTML element matches best to generate a multiline, scrollable alphanumeric text input block?

- A. `<input type="text" multi="true">`
- B. `<textarea>`
- C. `<input type="paragraph">`
- D. `<textblock>`

Q10.

What is the role of the client interface within a client-server architecture?

- A. To execute SQL queries that perform CRUD operations on the back-end database.
- B. To submit resource requests to the server and render its responses as a DOM tree for user interaction.
- C. To enforce firewall policies on data layers.

- D. To balance workloads among available computing instances.

Q11.

What type of HTML elements does the CSS combinator rule 'div > p' capture?

- A. Every paragraph nested anywhere inside an ancestor div element of the DOM tree.
- B. Only the direct paragraph child node elements of a div parent element.
- C. The very first paragraph element that is adjacent alongside a div wrapper component.
- D. Any paragraph that shares the identical hierarchy layer with an alternate div structure layout.

Q12.

Which CSS property configuration explicitly sets the standardized characters to be displayed in a bold face?

- A. font-style: bold;
- B. text-decoration: bold;
- C. font-weight: bold;
- D. text-transform: uppercase;

Q13.

Which two of the following approaches are used to include CSS style definitions into an HTML script? (TWO correct options)

- A. Including the stylesheets (.css files) via a <link> element inside the head section.
- B. Embedding CSS rule blocks inside a <style> tag in the body section.
- C. Dynamically fetching the styling parameters from relational database tables.
- D. Writing procedural CSS instructions inside binary JSON schema files.

Q14.

Consider the following URL.

<https://example.com/users?type=active>

Which of the following statements are true? (TWO correct options)

- A. The query parameter type is mapped to the string value active.
- B. It sends a POST request to the server.
- C. The query parameter is a part of the request body.
- D. The ? character acts as a separator between the path and the query string.

Q15.

A JavaScript code initializes the variable as

```
let data;
```

If the subsequent code checks its type without assigning any value, what does it return?

- A. null

- B. 0
- C. undefined
- D. NaN

Q16.

What is the semantic purpose of the 'alt' attribute defined inside the img tag?

- A. It specifies the aspect ratio of the image to be displayed.
- B. It provides a descriptive context for accessibility screen readers if the image cannot be fetched.
- C. It transforms graphical vectors into interactive vector paths during browser execution.
- D. It instructs the browser to prioritize local storage over remote queries.

Q17.

Which semantic HTML element is used as a container block that wraps the principal navigation components of a website?

- A. `<div class="nav">`
- B. `<section>`
- C. `<nav>`
- D. `<header>`

Q18.

Which attribute of the HTML form element specifies the URL endpoint where the form is to be submitted?

- A. method
- B. enctype
- C. action
- D. target

Q19.

Consider the following route in an Express application.

```
app.get("/employees/:id", (req, res) => {  
  ...  
});
```

How should the callback handler pull the value of the path parameter named 'id'?

- A. req.body.id
- B. req.query.id
- C. req.params.id
- D. req.argument.id

Q20.

Which of the following statements would you use to attach an external stylesheet named 'theme.css' into an HTML document?

- A. <import src="theme.css"> statement within the body section.

- B. `<link rel="stylesheet" href="theme.css">` instruction inside the head section.
- C. `<anchor href="theme.css" type="style">` element inside the body section.
- D. `<embed file="theme.css">` element before the final closing document tag.

Q21.

Assuming that an HTML document has an element with its ID attribute as 'sidebar', which of the following is the correct ID selector that informs the browser to apply the CSS properties declared in the associated rule to it?

- A. `.sidebar`
- B. `$sidebar`
- C. `#sidebar`
- D. `*sidebar`

Q22.

An element has the following CSS.

```
width: 300px;  
padding: 20px;  
border: 10px solid black;  
box-sizing: content-box;
```

What is the element's total rendered width (including padding and border)?

- A. 300px

B. 340px

C.
360px

D.
320px

Q23.

Which of the following code snippets are valid function declarations in JavaScript? (TWO correct options)

A.
`function computeTotal(sub, tax) { return sub + tax; }`

B.
`const computeTotal = (sub, tax) => { return sub + tax; };`

C.
`function computeTotal(sub, tax) [ return sub + tax; ]`

D.
`let computeTotal = function primitive computeTotal(sub, tax) select sub + tax;`

Q24.

Which statement correctly starts an Express server listening on port 3000 at host localhost?

A.
`app.listen('localhost', 3000);`

B.
`app.listen(3000, 'localhost');`

C.
`app.connect(3000, 'localhost');`

D.
`app.serve('localhost');`

Q25.

A developer installs the official Express scaffolding utility globally on a Linux server. Which command must be executed to initialize a

standard directory skeleton structure named my-application?

- A. npm init express
- B. express-bootstrap project
- C. express my-application
- D. node create-express

Q26.

Which two statements about HTML heading elements (<h1> to <h6>) are correct? (TWO correct options)

- A. An <h1> element usually represents the main heading of a page.
- B. A heading such as <h3> represents a subsection under a higher-level heading like <h2>.
- C. An <h6> element changes block content into inline content.
- D. Heading elements require a custom level attribute to be valid.

Q27.

Which HTTP method fetches the meta-information headers from a requested resource without retrieving its body payload?

- A. POST
- B. OPTIONS
- C. HEAD
- D. PATCH

Q28.

Which is the core design principle of a standard Single-Page Application (SPA)?

- A. It requires unique physical resource files on the host for every layout path navigation.
- B. It eliminates client-side runtime script parsing steps to lower client device overhead.
- C. It dynamically updates the required elements of the DOM tree without forcing a full page reload.
- D. It forces the browser to establish a secure peer-to-peer connection for resource updates.

Q29.

Which CSS property completely removes the decoration lines rendered on link anchors?

- A. text-style: none;
- B. underline: hidden;
- C. text-decoration: none;
- D. border-style: blank;

Q30.

Which of the following is an accurate definition of software 'source code'?

- A. The raw binary structures executed by the computer hardware.
- B. Unambiguous set of human-readable instructions written in a programming language.

- C. An encrypted and compressed production package containing compiled media artifacts.
- D. A server-side data structure mapping relational database schemas to storage sectors.

Q31.

Which two statements are true about the database connection object methods defined in the sqlite3 module for Node.js applications? (TWO correct options)

- A. The `db.all()` method fetches all matching result rows and returns a JavaScript array.
- B. The `db.get()` method yields the first matching row strictly as a single JavaScript object.
- C. The `db.each()` method must be called inside a while loop.
- D. The `db.run()` method processes only the SELECT statements and returns a result set.

Q32.

Which JavaScript keyword terminates the execution of the code block in a function while returning the control to the caller context, optionally along with an expression?

- A. `break`
- B. `return`
- C. `exit`
- D. `terminate`

Q33.

Which of the following statements correctly attaches an event handler callback to an existing button named 'submitButton'?

- A. `submitButton.attachListener("onclick", callBack);`
- B. `submitButton.addEventListener("click", callBack);`
- C. `submitButton.bind("onchange", callBack);`
- D. `submitButton.addEventListener("onclick", callBack);`

Q34.

What differentiates the equality operator (==) from the strict equality operator (===) in JavaScript?

- A. The == operator compares only text formatting values.
- B. The === operator compares both value and data type without automatic type conversion.
- C. The == operator is used to create constants, while the === operator updates variables.
- D. The === operator ignores decimal values to make loops faster.

Q35.

Which is the correct syntax to insert comment strings inside an external CSS stylesheet file?

- A. `// This is a style block comment`
- B. `/* This is a style block comment */`
- C. `<!-- This is a style block comment -->`

D. # This is a style block comment

Q36.

Why is it essential to start an HTML document with the '<!DOCTYPE html>' expression?

- A. To activate secure socket layer validations inside browser contexts.
- B. To instruct user browser rendering engines to parse the HTML code using modern standards mode.
- C. To import compressed character encoding from official CDNs.
- D. To declare that the markup structure relies on server-side pre-processing tools.

Q37.

Which SQL statement correctly creates a table called 'employees' and enforces an auto-incrementing integer primary key?

- A. CREATE TABLE employees (id PRIMARY KEY, name TEXT);
- B. CREATE TABLE employees (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT);
- C. MAKE TABLE employees [id INT KEY, name VARCHAR];
- D. NEW STRUCTURE employees (id INT AUTO, name STRING);

Q38.

Which statement correctly describes Node.js?

- A. A multithreaded system used only for synchronous database operations.

- B. An asynchronous, event-driven JavaScript runtime built on Chrome's V8 engine.
- C. A browser-based framework used to create graphical user interfaces.
- D. A server platform that can display HTML but cannot run scripts.

Q39.

In the CSS box model, which of the following is positioned outside the padding bounds but inside the margin boundaries?

- A. content box
- B. border box
- C. outline ring
- D. margin edge

Q40.

Which HTML element contains metadata and linked resources that are not shown directly in the webpage content?

- A. <head>
- B. <body>
- C. <main>
- D. <section>

Epilogue: Beyond Essentials

As you prepare for the LPI Web Development Essentials certification exam, you are developing a strong foundation for your profession as a web developer. The official curriculum for this exam covered the core concepts: front-end web technologies including HTML, CSS, and JavaScript; and server-side development with Node.js and the Express framework with SQLite as a back-end database. These technologies are the bedrock upon which the modern web is built.

However, you should treat this certification as the first step. The web ecosystem evolves rapidly. To thrive as a professional developer, you should use this foundation as a springboard to explore industry-standard technologies.

In addition to vanilla JavaScript, you should also be proficient in using component-based libraries like React or Angular, paired with TypeScript for robust, type-safe development.

You should also look to upgrade your styling capabilities by adding utility-first frameworks like Tailwind CSS into your toolkit.

The LPI curriculum introduced you to the principles of REST. Build on this with robust authentication techniques like JWT and OAuth. For real-time data handling, modern web applications employ advanced protocols like WebSocket and GraphQL, which you should be adept at.

While SQLite is an excellent, lightweight database generally used for building prototypes or proof-of-concept (POC) applications, production applications typically replace it with enterprise-grade relational systems like PostgreSQL or NoSQL options like MongoDB. Hence, a sound understanding of these technologies is highly desired.

Modern software delivery relies heavily on deployment infrastructure. As a professional web developer, you should strive for proficiency in containerization tools like Docker, orchestration platforms like Kubernetes, and major cloud architectures.

As a signing-off note, I hope this book serves you as a handy companion—not only for your ensuing LPI certification exam but also

as a reliable reference point throughout your professional journey as a web developer.

Here's wishing you all the best!

Answer Keys

Model Test Paper I

Q	Answer	Q	Answer	Q	Answer	Q	Answer
1	C	11	B	21	C	31	B
2	C	12	A, C	22	D	32	D
3	C	13	C	23	C	33	C
4	C	14	C	24	B, D	34	D
5	B, D	15	D	25	D	35	B, D
6	D	16	B	26	B	36	C
7	B	17	C	27	C	37	B
8	C	18	D	28	B, D	38	D
9	D	19	C	29	C	39	D
10	C	20	B	30	B	40	B

Model Test Paper II

Q	Answer	Q	Answer	Q	Answer	Q	Answer
1	D	11	B	21	C	31	A, B
2	B	12	C	22	C	32	B
3	B	13	A, B	23	A, B	33	B
4	A, B	14	A, D	24	B	34	B
5	C	15	C	25	C	35	B
6	B	16	B	26	A, B	36	B
7	C	17	C	27	C	37	B
8	B	18	C	28	C	38	B
9	B	19	C	29	C	39	B
10	B	20	B	30	B	40	A

Index

A

Absolute units [219](#)

Accept attribute [174](#)

Action attribute [163](#)

addEventListener() method [388](#), [389](#)

addition() function [348](#)

Adjacent sibling combinator (+) [242](#), [243](#)

Align-items property [270](#)

Anchor element

- absolute address [125](#)

- href attribute [124-126](#)

- HTML code [125](#)

- internal hyperlinks [127](#), [128](#)

- URL schemes [127](#)

- values [126](#)

- web pages [125](#)

Angular [43](#)

Apache HTTP server [76](#)

Apple Safari [38](#)

Application programming interfaces (APIs) [41](#), [46-48](#)

Application software [7](#)

Arithmetic operators [312](#), [313](#)

Array [307](#)

Arrow functions [348](#), [349](#)

Artificial intelligence [21](#)

Assembly language [4](#)

Asynchronous execution model [415](#)

Asynchronous I/O [407](#)

Atomicity, consistency, isolation and durability (ACID) [481](#)

Attribute selector [232-237](#)

B

Block boxes [255](#)
Boolean expression [313](#), [324](#)
Border-style property [211](#)
Border-width property [211](#)
Box model [255](#), [256](#)
Break statement [341–343](#)
Browser-induced event [380](#)
Built-in functions [343](#), [344](#)
Button controls [165](#)

C

C++ [20](#)
Cacheability [80](#)
Callbacks
 asynchronous execution model [415](#)
 error-first [417–419](#)
 functions [415](#), [440](#)
 multithreading approach [415](#)
 server-side technologies [415](#)
 setTimeout() [416](#)
Cascading HTML style sheets (CHSS) [197](#)
Cascading style sheets (CSS) [24](#)
 background attribute [200](#)
 bgcolor attribute [199](#)
 <big> and <small> element [199](#)
 border attribute [200](#)
 <center> element [198](#)
 definition [197](#)
 external stylesheets [203](#), [204](#)
 element [197](#)
 inline style [201](#)
 properties
 background [208–210](#)
 border [210–214](#)
 color [206–208](#)

- font [214–216](#)
- height and width [218](#)
- margin [217](#), [218](#)
- modules [205](#), [206](#)
- text [216](#), [217](#)
- structural and content-oriented elements [197](#)
- <style> element [202](#), [203](#)
- syntax [204](#), [205](#)
- units
 - absolute [219](#)
 - defined [218](#)
 - length-type properties [219](#)
 - relative [220](#)
 - width and height attribute [200](#)
- Cassandra Query Language (CQL) [52](#)
- Centralized version control system (CVCS) [28](#)
- Character entities [112](#), [113](#)
- Child combinator (>) [240](#)
- Chrome [37](#)
- Class [16](#)
- Class selector [229–232](#)
- Client-server model [33](#), [34](#), [58](#), [80](#)
- Client-side programming [24](#)
- close() method [476](#)
- COBOL [19](#)
- Code on demand [80](#)
- Cols attribute [180](#)
- Column-oriented databases [52](#)
- Combinators
 - adjacent sibling [242](#), [243](#)
 - child [240](#)
 - definition [237](#)
 - descendant [239](#)
 - DOM tree [238](#)

- HTML code [237](#)
- selectors [237](#)
- subsequent sibling [240](#), [241](#)
- Comments [100](#), [304](#)
- Comparison and logical operators [313–315](#)
- Compiler [8](#), [9](#)
- Compound assignment operators [315](#), [316](#)
- Computer architectures [5](#)
- Conditional statements
 - definition [323](#)
 - else if statement [329](#), [330](#)
 - if statement [324–326](#)
 - nested if [326–328](#)
 - string interpolation [328](#)
 - switch and case [330](#), [331](#), [333](#)
- Condition-chaining algorithm [330](#)
- Constraints [48](#)
- Continue statement [341–343](#)
- Cookies [74](#), [75](#)
- Counted loop [334](#)
- C program [6](#), [19](#)
- createAttribute() method [369](#)
- createElement() method [368](#)
- Create, read, update, delete (CRUD) [65](#)
- createServer() function [420](#), [421](#)
- createTextNode() method [370](#)
- Cross-origin resource sharing (CORS) [65](#)
- Cross-site request forgery (CSRF) [75](#)
- CSS layouts
 - box model [255](#), [256](#)
 - display property [267](#), [268](#)
 - flexbox [269–274](#)
 - float property [259–266](#)
 - flow layout [256–258](#)

- grid layout [275-281](#)
- position property [261-266](#)
- responsive web design [281-283](#)
- types [255](#)

CSS selectors

- attribute [232-237](#)
- class [229-232](#)
- combinators [237-242](#)
- element [224-227](#)
- ID [228](#), [229](#)
- pseudo-classes [243-248](#)
- pseudo-elements [248-251](#)
- universal [223](#), [224](#)

Customer table [458](#)

D

Database connection object

- db.all() [478](#), [479](#)
- db.close() [476](#)
- db.each() [479-481](#)
- db.get() [476](#), [477](#)
- db.run() [472-475](#)

Databases [50-52](#)

Data control language (DCL) [23](#), [460](#)

Data definition language (DDL) [23](#), [460](#)

Data manipulation language (DML) [23](#), [460](#)

Data query language (DQL) [23](#)

Date picker [169](#)

- db.all() [478](#), [479](#)

- db.close() [476](#)

- db.each() [479-481](#)

- db.get() [476](#), [477](#)

- db.run()

 - asynchronous method [472](#)

 - creating table [472](#), [473](#)

- insert record [473](#), [474](#)
- modify record [474](#), [475](#)
- remove record [475](#)
- Declarative paradigm [18](#), [19](#)
- DELETE method [65](#)
- Descendant combinator (" ") [239](#)
- Dialog element [138](#), [139](#)
- Directives [43](#)
- Disclosure elements [136](#), [137](#)
- Display property [267](#), [268](#)
- Distributed version control system (DVCS) [28](#)
- DNS server [35](#), [36](#)
- document.body property [359](#)
- document.documentURI property [359](#)
- document.head property [359](#)
- document.images property [360](#)
- Document object model (DOM) [24](#), [36](#)
 - definition [355](#)
 - events [380–390](#)
 - HTMLElement object [370–380](#)
 - HTML script [356](#)
 - leaf [358](#)
 - methods [361–370](#)
 - properties [359–361](#)
 - steps [355](#)
 - traversal
 - children property [377](#), [378](#)
 - firstElementChild [378](#)
 - lastElementChild [378](#)
 - nextElementSibling [378](#)
 - parentElement [378](#)
 - previousElementSibling [378](#), [380](#)
- tree [356](#), [357](#)
 - developer tools [357](#), [358](#)

- visualizer [357](#)
- Document root [77](#)
- document.scripts property [360](#)
- document.styleSheets property [361](#)
- document.write() function [301](#)
- document.write() statements [328](#)
- Domain name system (DNS) [34](#)
- do while looping [336](#), [337](#)
- Dynamic Host Configuration Protocol (DHCP) [57](#)
- Dynamic typing [311](#)

E

- ECMAScript [25](#), [39](#)
- Edge [37](#)
- Element selector [224–227](#)
- else if statement [329](#), [330](#)
- Embedded JavaScript (EJS) [431](#)
 - definition [445](#)
 - dependencies [446](#)
 - dynamic data [447](#)
 - Node.js package [445](#)
 - path parameters [448](#)
 - syntax [448](#), [449](#)
 - web page [446](#)
- Employee class [17](#)
- Encapsulation [17](#)
- Enctype attribute [165](#)
- Enigmatic programming technique [349](#)
- Error-first callback [417–419](#)
- Error handling [452](#), [453](#)
- ES6 (ES2015) [41](#)
- ES8 (ES2017) [41](#)
- ES15 (ES2024) [41](#)
- ES16 (ES2025) [41](#)
- ES Standards [41](#)

- Event-driven architecture [407](#)
- Event handler properties
 - definition [381](#)
 - examples [381](#)
 - keyboardEvent property [385](#)
 - mouse movement property [383–385](#)
 - onclick property [382](#), [383](#)
 - oninput event [386](#)
 - onsubmit property [386](#), [387](#)
- Event Listener methods [388–390](#)
- Explicit coercion [317](#)
- Express app structure [437](#)
- Express-generator [436–438](#)
- Express.js framework [44](#)
 - built-in middleware [442–444](#)
 - definition [431](#)
 - EJS template [445–449](#)
 - express-generator [436](#), [437](#)
 - Hello Express [435](#), [436](#)
 - installation [434](#), [435](#)
 - middleware [433](#), [440](#), [441](#)
 - minimal and unopinionated framework [432](#)
 - node_modules folder [434](#)
 - path parameters [439](#), [440](#)
 - request and response helpers [433](#)
 - routing mechanism [432](#), [438](#), [439](#)
 - static assets [433](#)
 - template engine support [433](#)
 - web applications [431](#)
- express.json() [442–444](#)
- express.Router() [444](#)
- express.static() [444](#), [450](#)
- express.urlencoded() [444](#)
- Extensible HyperText Markup Language (XHTML) [97](#)

Extensible Markup Language (XML) [39–41](#)

External stylesheets [203](#), [204](#)

F

Factorial, defined [349](#)

File Transfer Protocol (FTP) [57](#)

Flexbox [269–275](#)

Flex-wrap property [269](#)

Float property [259–261](#)

Flow layout [256–258](#)

Fluid layouts [282](#)

Foreign keys [51](#), [459](#)

for...in loop [339](#)

for looping [337](#), [339](#)

Forms

CGI scripts [162](#)

<datalist> element [177–179](#)

<form> element

action attribute [163](#)

defined [163](#)

enctype attribute [165](#)

method attribute [164](#)

<input> element

declaration [167](#)

form-associated interactive controls [166](#)

name attribute [167](#)

type attribute [167–176](#)

types [167](#)

value attribute [167](#)

<label> element [166](#)

markup elements [162](#)

<select> element [176](#), [177](#), 179

tags [98](#)

<textarea> element

cols attribute [180](#)

- comments [180](#)
- rows attribute [180](#)
- wrap attribute [180](#)
- for...of loop [340](#)
- FORTRAN [19](#)
- Functional programming [19](#)
- Functions
 - arrow functions [348](#), [349](#)
 - built-in functions [343](#), [344](#)
 - function with parameters [346](#), [347](#)
 - function with return [347](#), [348](#)
 - recursive functions [349-351](#)
 - reusable subprogram [343](#)
 - user-defined functions [344](#), [346](#)

G

- getElementById() method [362](#)
- getElementsByClassName() method [364-366](#)
- getElementsByName() method [363](#), [364](#)
- getElementsByTagName() method [362](#), [363](#)
- GET method [62](#), [164](#), [424](#)
- Git [29](#)
- Global attributes [106](#), [201](#)
- Graph query language (GraphQL) [49](#), [50](#)
- Grid layout [275-281](#)
- Grid track [276](#)
- Grouping controls [165](#)

H

- Heading tags [108-110](#)
- High-level language [5](#), [6](#)
- High-level programming languages
 - C [19](#)
 - C++ [20](#)
 - CSS [24](#)

- HTML [23](#)
- Java [21](#)
- JavaScript [24](#), [25](#)
- Python [21](#), [22](#)
- SQL [22](#), [23](#)
- Hotel booking apps [48](#)
- HTMLDivElement [370](#)
- HTMLElement [100](#)
 - properties
 - attributes [373](#), [374](#)
 - classList [375–377](#)
 - id [371](#)
 - innerHTML [372](#)
 - innerText [371](#)
 - outerHTML [374](#), [375](#)
 - textContent [372](#)
 - subclasses [370](#)
- HTML Living Standard [41](#), [42](#)
- HTML tags [98–100](#)
- http module
 - createServer() function [420](#), [421](#)
 - definition [420](#)
 - request object [422–427](#)
 - response object [421](#), [422](#)
- HTTP protocol [128](#)
- Hypertext [124](#)
- HyperText Markup Language (HTML) [23](#), [97](#)
- HyperText Transfer Protocol (HTTP) [23](#), [57](#)
 - cookies [74](#), [75](#)
 - HTTPS and TLS [78](#), [79](#)
 - principles [58](#), [59](#)
 - request methods [61–68](#)
 - request-response cycle [60](#), [61](#)
 - response

- components [68](#)
- headers [72](#), [73](#)
- status code [68-72](#)
- REST constraints [79-81](#)
- servers [76](#)
- specifications [58](#)
- URLs [76](#), [77](#)
- virtual host [77](#), [78](#)
- HyperText Transfer Protocol Secure (HTTPS) [78](#), [79](#)

I

- Identifiers [305](#), [306](#)
- ID selector [228](#), [229](#)
- if statement [324](#), [326](#)
- Imperative paradigm [15](#), [16](#)
- Implicit coercion [317](#)
- Infinite loop [335](#)
- Inheritance [17](#), [249](#), [250](#)
- Inline boxes [255](#)
- Integrated development environments (IDEs) [7](#), [12-15](#)
- IntelliJ IDEA [13](#)
- Interactive brokers (IBKR) [48](#)
- Internal hyperlinks [127](#), [128](#)
- Internet Engineering Task Force (IETF) [58](#)
- Internet Explorer [37](#)
- Internet Information Services (IIS) [76](#)
- Internet Protocol (IP) address [35](#)
- Internet Protocol version 6 (IPv6) [35](#)
- Interpreter [10-12](#)
- Invoice table [459](#)

J

- Java Development Kit (JDK) [21](#)
- Java language [21](#)
- Java Runtime (JRE) [12](#), [21](#)

- JavaScript (JS) [24](#), [25](#)
 - console [300](#), [301](#)
 - data types
 - object [307](#), [308](#)
 - primitives [306](#), [307](#)
 - typeof operator [308](#)
 - developer tools run command [300](#)
 - frameworks [42–45](#), [299](#)
 - front-end web technologies [299](#)
 - inside HTML [301](#), [302](#)
 - operators [312–316](#)
 - settings [299](#), [300](#)
 - syntax
 - comments [304](#)
 - identifiers [305](#), [306](#)
 - keyword [303](#)
 - statement [304](#)
 - variables [309–311](#)
 - web applications [299](#)
- JavaScript Object Notation (JSON) [39](#)
- Java Virtual Machine (JVM) [11](#), [21](#)

K

- Keywords [303](#)

L

- LaTeX [97](#)
- Layered system [80](#)
- Layout elements
 - <article> tag [133](#)
 - <aside> tag [135](#)
 - <div> elements [128](#)
 - <footer> tag [135](#)
 - <header> tag [128](#)
 - <main> tag [130](#), [131](#)

- <nav> tag [129](#), [130](#)
 - <section> tag [131](#), [132](#)
- Libraries [25–27](#)
- Link tags [99](#)
- List tags [98](#)
 - <div> element [118](#), [119](#)
 - nested list [116](#), [117](#)
 - ordered list [114](#), [115](#)
 - unordered list [115](#), [116](#)
- LiteSpeed [76](#)
- Local repository [28](#)
- Loops
 - break and continue [341–343](#)
 - calculator program [334](#)
 - do while keyword [336](#), [337](#)
 - for...in loop [339](#)
 - for keyword [337](#), [339](#)
 - for...of loop [340](#)
 - repetition feature [333](#)
 - requirements [334](#)
 - while keyword [334](#), [335](#)

M

- Machine code [6](#)
- Machine language [4](#)
- Machine learning [21](#)
- Media-independent protocol [59](#)
- Media queries [282](#), [283](#)
- Method attribute [164](#)
- Microsoft Visual C++ compiler (MSVC) [20](#)
- Middleware [433](#), [440](#), [441](#)
- Modal dialog [138](#)
- MongoDB Query Language (MQL) [52](#)
- Mouse events [384](#)
- mouseenter/mouseleave events [383](#)

mouseover/mouseout events [383](#)
Mozilla Developer Network (MDN) [98](#)
Mozilla Firefox [37](#)
Mozilla Web Docs [98](#)
Multimedia elements
 <audio> tag [143](#), [144](#)
 <figure> tag [142](#)
 <iframe> tag [145](#), [146](#)
 tag [139](#), [140](#)
 <picture> tag [140](#), [141](#)
 playback [139](#)
 <video> tag [144](#), [145](#)
Multimedia tags [99](#)
Multiple attribute [174](#)
Multipurpose Internet Mail Extensions (MIME) [59](#)

N

Name attribute [167](#)
Nesting if statements [326–328](#)
NestJS [44](#)
Next.js [44](#)
Nginx [76](#)
Node.js [44](#), [482](#)
 callbacks [415–419](#)
 concepts [407](#)
 http module [420–427](#)
 installation [408](#)
 vs. JavaScript [408](#)
 modules [411–413](#)
 npm [413–415](#)
 REPL terminal [409](#), [410](#)
 script mode [410](#), [411](#)
 server-side programming [407](#)
Node Package Manager (npm) [408](#)
 init [414](#), [415](#)

- installation [413](#), [414](#)
- registry [413](#)
- Non-modal dialog [138](#)
- NoSQL database [52](#)
- Notepad [12](#)
- npx express-generator command [436](#)
- Number() function [318](#), [319](#)

O

- Object code [6](#)
- Object-oriented paradigm [16–18](#)
- Object types [307](#), [308](#)
- onmouseover/onmouseout properties [383](#)
- Opcode [4](#)
- OpenJDK [21](#)
- Opera [38](#)
- Operand [4](#)
- Operating system [8](#)
- Operators
 - arithmetic [312](#), [313](#)
 - categories [312](#)
 - comparison and logical [313–315](#)
 - compound assignment [315](#), [316](#)
 - non-alphanumeric characters [312](#)
 - type conversion [316–319](#)
- Oracle [21](#)
- Oracle Corp. [22](#)
- OR operator [314](#)
- Output controls [165](#)

P

- Packaging web applications [52](#)
- Padding [255](#)
- Paired tags [99](#)
- Paragraph element [110–112](#)

- Path parameters [439](#), [440](#)
- Polymorphism [17](#)
- pop() method [308](#)
- Position property [263](#)
 - absolute [263](#), [264](#)
 - fixed [264](#)
 - predefined values [261](#)
 - relative [262](#)
 - static [262](#)
 - sticky [265-267](#)
- POST method [63](#), [64](#), [424](#), [426](#), [427](#)
- Presentation tags [98](#), [107](#), [108](#)
- preventDefault() method [387](#)
- Primary keys [51](#), [458](#), [467](#)
- Primitive types [306](#), [307](#)
- Procedural programming [16](#)
- Programming paradigms
 - concepts [15](#)
 - declarative [18](#), [19](#)
 - imperative [15](#), [16](#)
 - methodology [15](#)
 - object-oriented [16-18](#)
- prompt() function [325](#), [326](#)
- Proof of concept (POC) [485](#)
- Pseudo-classes
 - form/input state
 - :checked [247](#)
 - :disabled [247](#)
 - elements [246](#)
 - :hover [244](#)
 - keywords [243](#)
 - location
 - :any-link [248](#)
 - :link [247](#)

- Relational databases [457](#)
- Relational Software Inc. [22](#)
- Relations [51](#)
- Relative units [220](#)
- Remote Procedure Call (RPC) [48](#)
- removeEventListener() method [388](#), [390](#)
- remove() method [368](#)
- render() method [447](#)
- REPL shell [10](#)
- Representational state transfer (REST) [41](#), [48](#), [79-81](#)
- Request headers [66-68](#)
- Request object [422-427](#)
- Request-response cycle [58](#), [60](#), [61](#)
- Reserved characters [113](#)
- response.end() function [422](#)
- Response object [421](#), [422](#)
- response.send() method [447](#)
- response.write() function [421](#)
- response.writeHead() function [421](#)
- Responsive web design
 - fluid layouts [282](#)
 - media queries [282](#), [283](#)
 - screen widths [281](#)
 - viewport [282](#)
 - web development [281](#)
- REST constraints [79-81](#)
- rgb() function [207](#)
- rgba() function [208](#)
- Routing mechanism [431](#), [432](#), [438](#), [439](#)
- RowCallback function [479](#)
- Rows attribute [180](#)
- Rows/tuples [51](#), [458](#)

S

- sayHello() [415](#), [417](#)

- sayThanks() [415-417](#)
- Scope attribute [156](#)
- Script mode [410](#), [411](#)
- Secure sockets layer (SSL) [78](#)
- Selection controls [165](#)
- Self-closing tags [99](#)
- Semantics [123](#)
- Semantic tags [99](#)
 - anchor element [124-128](#)
 - <dialog> elements [138](#), [139](#)
 - disclosure elements [136](#), [137](#)
 - elements [124](#)
 - layout elements [128-136](#)
 - multimedia elements [139-146](#)
 - presentation tags [124](#)
 - text-level [147-148](#)
- Server-side rendering (SSR) [44](#)
- setAttributeNode() method [369](#)
- setTimeout() [416](#)
- Simple Mail Transfer Protocol (SMTP) [57](#)
- Simple Object Access Protocol (SOAP) [48](#)
- Simplicity [22](#)
- Single-page applications [45](#), [46](#)
- Software [6-8](#)
- Software development life cycle (SDLC) [30](#)
- Software testing [30](#), [31](#)
- Source code [5](#)
- Spanning [160-162](#)
- Spanning in grid layout [279-281](#)
- Specificity [250](#), [251](#)
- SQL injection [482-484](#)
- SQLite
 - command-line shell
 - create database [465](#), [466](#)

- create new table [466](#), [467](#)
- DELETE statement [470](#)
- insert record [467](#), [468](#)
- SELECT statement [468](#), [469](#)
- UPDATE statement [469](#)
- DB browser [462](#), [463](#)
- definition [461](#)
- evolution [461](#)
- Express app [484–486](#)
- installation [461](#), [462](#)
- modules [482](#)
- SQLite3 Fiddle [464](#), [465](#)
- VS Code [463](#), [464](#)
- SQLite3 Fiddle [464](#), [465](#)
- sqlite3 module
 - connecting database [471](#)
 - definition [470](#)
 - project's configuration file [470](#)
- Standard Generalized Markup Language (SGML) [97](#)
- Statelessness [59](#), [80](#)
- Statements [304](#)
- Static assets [433](#), [450–452](#)
- Static Site Generation (SSG) [44](#)
- Status code
 - 1xx informational response [69](#)
 - 2xx successful response [69](#), [70](#)
 - 3xx redirection response [70](#)
 - 4xx client error response [71](#)
 - 5xx server error response [71](#), [72](#)
 - definition [69](#)
- Stock trading apps [48](#)
- Strict equality operators [319](#)
- Strict mode [310](#), [311](#)
- Strict not equality operators [319](#)

- String() function [317](#), [318](#)
- String interpolation [328](#)
- Structural tags [98](#), [109](#)
 - body section [106](#), [107](#)
 - DOCTYPE [101](#)
 - head section
 - external resources [102](#)
 - <link> element [102](#)
 - <meta> element [105](#), [106](#)
 - <script> element [104](#)
 - <style> element [103](#)
 - <title> element [102](#)
 - root element [101](#)
- Structured programming [16](#)
- Structured query language (SQL) [22](#), [23](#), [51](#), [457](#)
 - classification [460](#)
 - data types [460](#), [461](#)
 - definition [459](#)
 - dialect [460](#)
 - instructions [460](#)
- Subsequent sibling combinator (~) [240](#), [241](#)
- Switch/case keywords [330](#), [331](#), [333](#)
- Synchronous programming model [482](#)
- Syntax [5](#), [123](#)
- System R [22](#)
- System software [7](#), [8](#)

T

Tables

- <caption> element [158–160](#)
- components [153](#)
- headers and caption [160](#)
- spanning [160–162](#)
- <table> element [152](#), [153](#)
- tags [98](#)

- `<tbody>` element [158](#)
- `<td>` element [154](#), [155](#)
- `<tfoot>` element [157](#), [158](#)
- `<th>` element [155](#), [156](#)
- `<thead>` element [157](#)
- `<tr>` element [153](#)
- web page layout [151](#)

Tags [97](#)

Template literals [328](#)

Testing methodologies [31](#)

Text formatting [106](#)

Text input controls [165](#)

Text-level semantic tags

- `<blockquote>` [148](#)

- `<code>` [147](#)

- `<mark>` [147](#)

- `<var>` [147](#)

`toggle()` method [376](#)

Transaction control

- ACID properties [481](#)

- statements [481](#)

Transaction control language (TCL) [460](#)

Transport layer security (TLS) [78](#), [79](#)

Type attribute

- `type="checkbox"` [172](#), [173](#)

- `type="date"` [169](#), [170](#)

- `type="file"` [173](#), [174](#)

- `type="hidden"` [176](#)

- `type="image"` [175](#)

- `type="number"` [168](#)

- `type="password"` [168](#)

- `type="radio"` [170](#), [171](#)

- `type="range"` [170](#)

- `type="reset"` [175](#)

type="submit" [174](#), [175](#)

type="text" [168](#)

Type conversion

conversion to number [318](#), [319](#)

conversion to string [317](#), [318](#)

explicit [317](#)

implicit [317](#)

operands [316](#)

strict equality [319](#)

typeof operator [308](#)

U

Uniform interface [79](#)

Uniform Resource Identifier (URI) [49](#)

Uniform Resource Locators (URLs) [34](#), [76](#)

Universal selector [223](#), [224](#)

url() designator [209](#)

User agent [60](#)

User-defined functions [344](#), [346](#)

V

Value attribute [167](#), [173](#)

Variables [309-311](#)

Vector database [52](#)

Version control [27-29](#)

Version control system (VCS) [28](#)

Viewport [282](#)

Virtual host [77](#), [78](#)

Visual Studio Code (VS Code) [14](#)

Void tags [99](#)

Vue.js [43](#)

W, X, Y

Weather apps [47](#)

Web Application Archive (WAR) [53](#)

WebAssembly (wasm) [53](#)

Web browsers [36–38](#), [60](#)

Web form [163](#)

Web Hypertext Application Technology Working Group (WHATWG) [41](#),
[98](#)

WebSocket [81](#)

Web standards [38](#), [39](#)

Web template [445](#)

While loop [334](#), [335](#)

window.addEventListener() [107](#)

Word processor [6](#)

World Wide Web (WWW) [34](#), [35](#), [124](#)

World Wide Web Consortium (W3C) [23](#), [24](#), [38](#), [58](#), [97](#)

Wrap attribute [180](#)

Write once, run anywhere (WORA) [21](#)

Z

z-order property [265](#)

| HTML | Browser Output |
|--|--|
| <pre><h3>Examples of operating systems</h3> Windows Unix Linux macOS Android </pre> | <p>Examples of operating systems</p> <ul style="list-style-type: none">• Windows• Unix• Linux• macOS• Android |

Table 15-1 Customer Table

CustomerID (Primary Key)	Name	Email	Phone
101	Alice Smith	alice@example.com	9876543210
102	Bob Johnson	bon@example.com	9123456780
103	Charlie Brown	charlie@example.com	9988776555
104	Diana Prince	diana@example.com	9090110901

HTML	Browser Output
<pre> <h3>Steps to create a web page</h3> <ol type="a"> Start your computer Open a text editor Type HTML script Save as .html file Open in a browser </pre>	Steps to create a web page a. Start your computer b. Open a text editor c. Type HTML script d. Save as .html file e. Open in a browser

await	break	case	catch
class	const	continue	debugger
default	delete	do	else
enum	export	extends	finally
for	function	if	import
in	instanceof	let	new
return	super	switch	this
throw	try	typeof	var
void	while	with	yield

HTML	Browser Output
<pre><div style="background: gray; border: 2px"> <p style="color:white; text-align: center"> Hello, World! </p> </div></pre>	Hello World!

HTML	Browser Output
<pre><h3>Tennis Grand Slam Winners</h3> Novak Djokovic Rafael Nadal Roger Federer Pete Sampras Rod Laver </pre>	Tennis Grand Slam Winners 1. Novak Djokovic 2. Rafael Nadal 3. Roger Federer 4. Pete Sampras 5. Rod Laver

HTML	Browser Output
<pre><pre> #include <iostream> int main() { std::cout << "Hello, World!" << std::endl; return 0; } </pre></pre>	<pre>#include <iostream> int main() { std::cout << "Hello, World!" << std::endl; return 0; }</pre>

HTML	Browser Output
<pre><dialog open> <h2>A Modal Dialog</h2> <p>Click to Close</p> <form method="dialog"> <input type="submit"> </form> </dialog></pre>	 <i>Figure 5-3</i> <dialog> element

HTML	Browser Output
<pre data-bbox="203 279 1221 367"></pre>	HTML  <i>Figure 5-4</i> element

HTML

```
<picture>
  <source srcset="sample.webp" type="image/webp">
  <source srcset="sample.png" type="image/png">
  <source srcset="sample.jpg" type="image/jpeg">
  
</picture>
```


HTML	Browser Output
<pre> <h3>Continents and Countries</h3> Europe France Germany Spain Asia China India Japan Africa Egypt Zimbabwe Nigeria </pre>	Continents and Countries - Europe - France - Germany - Spain - Asia - China - India - Japan - Africa - Egypt - Zimbabwe - Nigeria

Table 4-1 Font Sizes of Heading Tags in em, px, and pt Units

Heading tag	em Size	px Size	pt Size
<h1>	2em	32px	24pt
<h2>	1.5em	24px	18pt
<h3>	1.17em	18.72px	14.04pt
<h4>	1em	16px	12pt
<h5>	0.83em	13.28px	9.96pt
<h6>	0.67em	10.72px	8.04pt

HTML	Browser Output
<pre><p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam porta vitae felis non ullamcorper.
 Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed eget nisi at ipsum mattis congue at non lorem. Vivamus tristique pulvinar eros, a malesuada metus dignissim sit amet. Donec a tellus massa.</p></pre>	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam porta vitae felis non ullamcorper. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed eget nisi at ipsum mattis congue at non lorem. Vivamus tristique pulvinar eros, a malesuada metus dignissim sit amet. Donec a tellus massa.

HTML	Browser Output
<pre> <p> The area of a rectangle is <var>length</var> × <var>width</var> </p> </pre>	The area of a rectangle is <i>length × width</i>

HTML	Browser Output
In HTML, the <code>img</code> tag is <code><mark>non-semantic</mark></code>	In HTML, the <code>img</code> tag is non-semantic

HTML	Browser Output
<code><p>In HTML, the <code><HTML></code> tag defines the root element</p></code>	In HTML, the <code><HTML></code> tag defines the root element

HTML	Browser Output
<pre><header> <h1>Web Development Essentials</h1> <p>HTML, CSS and JavaScript</p> </header></pre>	Web Development Essentials HTML, CSS and JavaScript

HTML	Browser Output
<pre><header> <nav> Home Register Login </nav> </header></pre>	Home Register Login My Website

HTML	Browser Output
<pre> <blockquote cite="https://webfoundation.org/about/community/knight-2008-tbl-speech/"> <p> The Web does not just connect machines, it connects people. </p> </blockquote> <p>—Tim Berners-Lee, <cite>World Wide Web Foundation</cite></p> </pre>	The Web does not just connect machines, it connects people. —Tim Berners-Lee, <i>World Wide Web Foundation</i>

HTML	Browser Output
<pre> <main> <h1>My Tech Blog</h1> <article id="blog1"> <h2>HTML</h2> <section class="section1"> <h3>Introduction</h3> <p>Introduction to HTML</p> </section> <section class="section2"> <h3>Tags</h3> <p>Basic Tags in HTML</p> </section> </article> <hr> <article id="blog2"> <h2>CSS</h2> <section class="section1"> <h3>Introduction</h3> <p>Introduction to CSS</p> </section> <section class="section2"> <h3>Selectors</h3> <p>Basic Selectors in CSS</p> </section> </article> </main> </pre>	My Tech Blog HTML Introduction Introduction to HTML Tags Basic Tags in HTML CSS Introduction Introduction to CSS Selectors Basic Selectors in CSS

HTML	Browser Output
<pre><main> <h1>Web Technologies</h1> <section id="section1"> <h2>HTML</h2> <p>Markup language for creating web pages.</p> </section> <section id="section2"> <h2>CSS</h2> <p>Styling of the content.</p> </section> <section id="section3"> <h2>JavaScript</h2> <p>Enables interactivity in page elements.</p> </section> </main></pre>	Web Technologies HTML Markup language for creating web pages. CSS Styling of the content. JavaScript Enables interactivity in page elements.

HTML	Browser Output
<pre> <figure> <pre> def hello(): print ("Hello World") return </pre> <figcaption>Listing 1: Python function</figcaption> </figure> </pre>	<pre> def hello(): print ("Hello World") return </pre> <i>Listing 5-1: Python function</i>

HTML

```
<details>
```

```
  <summary>HTML</summary>
```

```
  <p>Markup language for creating web pages.</p>
```

```
</details>
```

```
<details>
```

```
  <summary>CSS</summary>
```

```
  <p>Styling of the content.</p>
```

```
</details>
```

```
<details>
```

```
  <summary>JavaScript</summary>
```

```
  <p>Enables interactivity in page elements.</p>
```

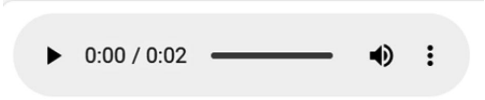
```
</details>
```

<code><p>Default text size. </p></code>	Default text size.
<code><p><big>Text in bigger size.</big></p></code>	Text in bigger size.

Table 4-2 Reserved Characters and Named and Decimal Entry Equivalents

Reserved Character	Description	Named HTML Entity	Decimal HTML Entity
"	Double quotation mark	"	"
'	Single quotation mark / Apostrophe	'	'
&	Ampersand	&	&
<	Less-than sign	<	<
>	Greater-than sign	>	>

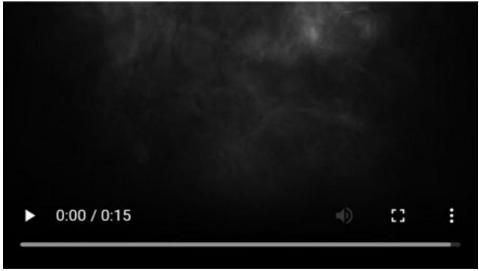
HTML	Browser Output
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam porta vitae felis non ullamcorper. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed eget nisi at ipsum mattis congue at non lorem. Vivamus tristique pulvinar eros, a malesuada metus dignissim sit amet. Donec a tellus massa.</p>	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam porta vitae felis non ullamcorper. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed eget nisi at ipsum mattis congue at non lorem. Vivamus tristique pulvinar eros, a malesuada metus dignissim sit amet. Donec a tellus massa.

HTML	Browser Output
<pre><audio controls> <source src="sample.mp3" type="audio/mpeg" /> <source src="sample.ogg" type="audio/ogg" /> <!-- This is the fallback content --> <p>Your browser does not support the audio element.</p> </audio></pre>	 Figure 5-5 <audio> tag

<pre><!--Linked page in the current folder--> Web Technologies
 <!--Linked page with absolute path--> HTTP Protocol
 <!--Linked page with relative path--> HTML tags</pre>	Web Technologies HTTP Protocol HTML tags
---	--

Table 15-2 Invoice Table

InvoiceNo (Primary Key)	Date	CustomerID (Foreign Key)	Amount
001	2026-01-10	102	1500.00
002	2026-01-31	101	2100.00
003	2026-03-01	104	1795.00
004	2026-03-15	102	2001.00

HTML	Browser Output
<pre><video controls> <source src="sample.mp4" type="video/mp4" /> <source src="sample.webm" type="video/webm" /> <!-- This is the fallback content --> <p>Your browser does not support the video element.</p> </video></pre>	 <i>Figure 5-6</i> <video> element