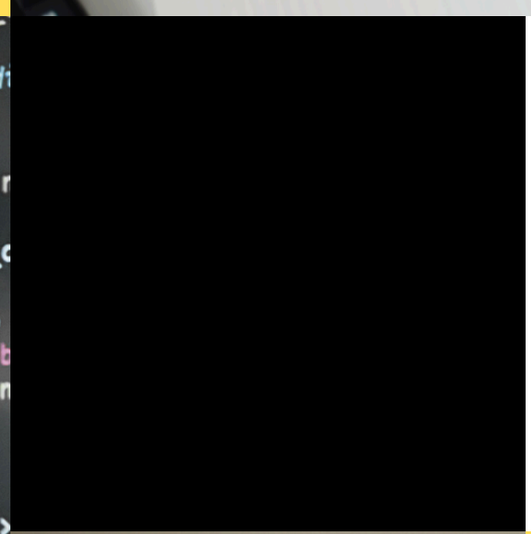
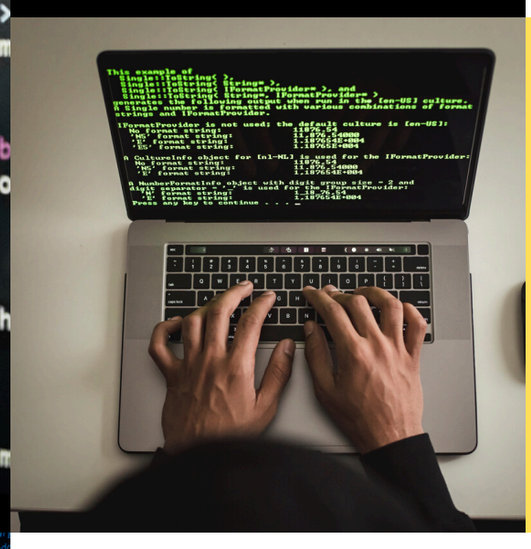
MENTEC FOUNDATION

STUDENT TEXTBOOK





INTRODUCTION TO

WEBSITE DEVELOPMENT

Chapter 1: Introduction to Web Development

Learning Objectives

By the end of this chapter, you will:

- Understand what web development is and the difference between frontend and backend development
- Learn about the tools and platforms you will use: SoloLearn and Replit
- Create your very first simple web page using HTML
- Become familiar with basic web page structure and essential HTML tags

1.1 What Is Web Development?

Concept: Web development is the process of building websites and web applications that you see on the internet. It involves writing code that tells browsers how to display content and how websites behave.

Real-World Analogy

Think of a website like a restaurant:

- The **frontend** is the dining area where customers interact — it needs to look nice and be easy to navigate.
- The **backend** is the kitchen where the food is prepared — it manages orders, ingredients, and cooking.

As a web developer, you might work on the frontend, backend, or both (called full-stack).

1.2 Tools of the Trade: SoloLearn and Replit

- **SoloLearn:** A mobile-friendly platform where you can learn coding concepts step-by-step and practice with small quizzes and challenges.
- **Replit:** An online code editor that runs your code instantly in the browser, no setup required. It's like your virtual coding workspace.

Tip Box

Use SoloLearn to learn and review theory, then switch to Replit to write and run your own code projects.

1.3 Your First Web Page: The Building Blocks of HTML

HTML (HyperText Markup Language) is the skeleton of any webpage. It tells browsers what content to show and how it is organized.

Step-by-Step Guide: Create a Simple HTML Page

1. Open Replit and create a new HTML project.
2. Copy and paste the following code:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My First Webpage</title>
 </head>
 <body>
  <h1>Welcome to My Website</h1>
  <p>This is my first web page made with HTML!</p>
 </body>
</html>
```

3. Click **Run** and observe your web page in the preview pane.

Expected Output:

A web page with a big heading "Welcome to My Website" and a paragraph below it.

1.4 Understanding the Code

Tag	Purpose
<!DOCTYPE html>	Declares this document as HTML5
<html>	Root element of the web page
<head>	Contains meta info like the page title
<title>	Sets the title shown on the browser tab
<body>	Contains the visible content of the page
<h1>	Main heading
	Paragraph of text

Common Errors to Avoid

- Forgetting to close tags (e.g., missing) can break your page layout.
- Case sensitivity matters: tags are lowercase in HTML5.

1.5 Check Your Understanding

- What is the difference between the <head> and <body> sections?
- Why do we use the <!DOCTYPE html> declaration?
- What tag would you use for a smaller heading below <h1>?

1.6 Practice Exercises

1. Modify the web page to add a second paragraph about your hobbies.

2. Add a <h2> heading below the <h1> with the text "About Me".
3. Change the page title to your name.

Chapter Summary

- Web development is split into frontend (user interface) and backend (server-side) work.
- HTML is the fundamental language used to structure webpages.
- SoloLearn and Replit are great tools for learning and practicing web development.
- Your first HTML page contains a document type, root elements, a head, and a body with headings and paragraphs.

Mini Project: Create a Personal Introduction Page

Goal: Build a simple web page introducing yourself.

Requirements:

- Include your name as the main heading
- Add at least two paragraphs about your interests or background
- Use at least one <h2> or <h3> subheading
- Test your page on Replit and save your work

Reflection Prompt

Think about a website or web app you use daily. How do you think the frontend and backend work together to provide the experience you enjoy? Write a short paragraph describing your thoughts.

Chapter 2: HTML Basics

Learning Objectives

By the end of this chapter, you will:

- Understand the detailed structure of an HTML document
- Learn to use common HTML tags such as headings, paragraphs, links, images, and lists
- Discover the importance of semantic HTML for accessibility and SEO
- Practice creating a richer web page with multiple content elements

2.1 Anatomy of an HTML Document

Every HTML file follows a basic structure that helps browsers render the content properly.

Code Walkthrough: Basic HTML Template

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>My Sample Page</title>
</head>
<body>
 <!-- Content goes here -->
</body>
</html>
```

Explanation:

- <!DOCTYPE html> tells the browser to use HTML5 standards.
- <html lang="en"> starts the document and sets the language to English.

- <head> contains metadata such as the character set and page title.
- <meta name="viewport" ...> makes your page mobile-friendly by controlling scaling.
- <body> is where visible content is placed.

Tip Box

Always include the lang attribute in your <html> tag for better accessibility and SEO.

2.2 Headings and Paragraphs

Headings organize your content hierarchically from <h1> (most important) to <h6> (least important).

html

```
<h1>Welcome to My Website</h1>
<h2>About Me</h2>
<p>Hello! I'm a beginner web developer learning HTML basics.</p>
<p>I enjoy coding and building simple websites.</p>
```

Real-World Analogy:
Think of headings like chapter titles and subtitles in a book—they guide the reader through the content.

2.3 Adding Links

Links allow users to navigate to other web pages or resources.

html

```
<a href="https://www.sololearn.com">Visit SoloLearn</a>
```

Expected Output:
A clickable link labeled "Visit SoloLearn" that opens the SoloLearn website.

Common Mistake to Avoid

- Forgetting the href attribute or leaving it empty causes links to not work.

2.4 Images on Your Page

Images make your site engaging and visually appealing.

html

- src is the image URL or path
- alt provides descriptive text for screen readers and when the image fails to load

Tip Box

Always include meaningful alt text for accessibility.

2.5 Lists: Ordered and Unordered

Lists help organize items clearly.

Unordered List:

html

 HTML Basics
 CSS Styling
 JavaScript Interactivity

```
</ul>
```

Ordered List:

html

```
<ol>
  <li>Learn HTML</li>
  <li>Practice CSS</li>
  <li>Explore JavaScript</li>
</ol>
```

2.6 Semantic HTML

Semantic tags like <header>, <nav>, <article>, and <footer> give meaning to page sections, improving SEO and accessibility.

```
<header>
  <h1>My Website</h1>
  <nav>
    <a href="#about">About</a> | <a href="#contact">Contact</a>
  </nav>
</header>
<article>
  <h2 id="about">About Me</h2>
  <p>This is a paragraph inside an article section.</p>
</article>
<footer>
  <p>© 2025 My Website</p>
</footer>
```

Why It Matters

Screen readers and search engines rely on semantic structure to better understand content.

2.7 Check Your Understanding

- What is the difference between <ul> and ?
- Why should you include the alt attribute on images?
- How do semantic tags improve a website's accessibility?
- Which tag defines the main title of a webpage?

2.8 Practice Exercises

1. Create an HTML page with:
    a. A <header> containing your site title and navigation links
    b. An <article> with two paragraphs about your learning goals
    c. An unordered list of your favorite websites
    d. An image with appropriate alt text
    e. A <footer> with your name and year
2. Modify the page title in the <head> section to reflect your name.
3. Add links that open in a new tab (hint: use target="_blank").

Chapter Summary

- HTML documents have a defined structure including <!DOCTYPE>, <html>, <head>, and <body>.
- Headings and paragraphs organize content logically.
- Links and images enhance navigation and visual appeal.
- Lists organize information clearly.
- Semantic tags improve accessibility and SEO.

Mini Project: Build a Personal Profile Page

Goal: Create a multi-section webpage introducing yourself.

Requirements:

- Use semantic tags to structure the page
- Include headings, paragraphs, links, images, and lists
- Use proper alt text and accessible attributes
- Test your page on Replit and save your work

Reflection Prompt

How can well-structured HTML improve user experience for diverse audiences, including people with disabilities? Reflect on how semantic HTML and accessibility attributes help achieve this.

Chapter 3: CSS Fundamentals

Learning Objectives

By the end of this chapter, you will:

- Understand what CSS (Cascading Style Sheets) is and why it's essential for web design
- Learn different ways to add CSS to your HTML pages
- Explore common CSS selectors and properties for styling text, colors, and layout
- Practice writing CSS rules and applying styles to your web pages

3.1 What is CSS?

CSS is the language used to style HTML elements. While HTML structures the content, CSS controls *how* it looks — colors, fonts, spacing, layout, and more.

Real-World Analogy

If HTML is the structure of a house, CSS is the paint, wallpaper, furniture, and decorations that make it beautiful and comfortable.

3.2 Ways to Add CSS to Your Web Page

There are three main methods to apply CSS:

1. Inline CSS

Style applied directly to an HTML element using the style attribute.

html

```
<p style="color: blue; font-size: 18px;">This text is blue and bigger.</p>
```

2. Internal CSS

CSS written inside a <style> tag within the <head> section.

html

```
<head>
 <style>
  p {
   color: green;
   font-size: 16px;
  }
 </style>
```

```
</head>
```

3. External CSS

CSS placed in a separate .css file linked to the HTML.

html

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

3.3 Basic CSS Syntax and Selectors

A CSS rule has three parts:

css

```
selector {
  property: value;
}
```

- **Selector**: the HTML element you want to style (e.g., p, .class, #id)
- **Property**: what aspect you want to change (e.g., color, font-size)
- **Value**: the setting for that property (e.g., red, 24px)

Examples:

css

```
/* Style all paragraphs */
p {
  color: navy;
}
```

```
/* Style element with id="header" */
#header {
  background-color: lightgray;
}

/* Style elements with class="highlight" */
.highlight {
  font-weight: bold;
  background-color: yellow;
}
```

3.4 Common CSS Properties

Property	Description	Example
color	Text color	color: red;
background-color	Background color	background-color: #f0f0f0;
font-size	Size of text	font-size: 16px;
font-family	Font style	font-family: Arial, sans-serif;
margin	Space outside an element	margin: 20px;
padding	Space inside an element	padding: 10px;
border	Border style	border: 1px solid black;

3.5 Hands-On: Adding Style to Your Page

Step-by-step:

1. Open your HTML page on Replit.
2. Add a <style> block inside the <head> tag with the following CSS:

css

```
body {
  font-family: 'Arial', sans-serif;
  background-color: #fafafa;
  margin: 20px;
}

h1 {
  color: #2c3e50;
}

p {
  color: #34495e;
  font-size: 18px;
}
```

3. Run your page and observe the styled output.

3.6 Check Your Understanding

- What are the differences between inline, internal, and external CSS?
- How do you select an element with a specific class in CSS?
- Which property changes the text color?
- What is the purpose of the margin property?

3.7 Practice Exercises

1. Change the background color of your page to light blue.
2. Make all <h2> headings bold and colored dark green.
3. Add a class .highlight that makes text yellow with a black background, and apply it to one paragraph.
4. Create an external CSS file named styles.css and move your internal styles there. Link it to your HTML.

Chapter Summary

- CSS styles your web pages by controlling colors, fonts, spacing, and layout.
- You can add CSS inline, internally within HTML, or externally via separate files.
- Selectors target HTML elements by tag, class, or ID to apply styles.
- Common properties include color, background-color, font-size, margin, and padding.

Mini Project: Style Your Personal Profile Page

Goal: Use CSS to make your profile page visually appealing.

Requirements:

- Apply colors and fonts to headings and paragraphs
- Use margins and padding to space elements neatly
- Create and use a .highlight class
- Use an external CSS stylesheet linked to your HTML file

Reflection Prompt

How can CSS improve not only the look but also the usability of a website? Think about how spacing, colors, and fonts affect user experience.

Chapter 4: JavaScript Basics

Learning Objectives

By the end of this chapter, you will:

- Understand what JavaScript (JS) is and its role in web development

- Learn basic JavaScript syntax including variables, data types, and operators
- Write and run simple JS scripts using Replit
- Use console.log() and alert() for output and debugging
- Practice with coding exercises to build confidence

4.1 What is JavaScript?

JavaScript is the programming language that makes websites interactive. While HTML creates structure and CSS styles it, JS adds dynamic behavior — like responding to clicks, validating forms, updating content without refreshing, and more.

Real-World Analogy

Think of HTML as the stage, CSS as the set design, and JavaScript as the actors that bring the story to life.

4.2 Running JavaScript in Replit

In Replit, you can create a new **JavaScript** project to write and run JS code instantly.

Example:

javascript

```
console.log("Hello, world!");
```

This prints “Hello, world!” to the console.

4.3 Variables and Data Types

Variables store information you want to use later.

javascript

```
let name = "Alice";     // String
let age = 25;           // Number
let isStudent = true;   // Boolean
```

Data Types:

- **String:** Text inside quotes
- **Number:** Numeric values
- **Boolean:** True or false values

4.4 Operators

Operators perform actions on values:

Operator	Meaning	Example	Result
+	Addition	2 + 3	5
-	Subtraction	5 - 1	4
*	Multiplication	4 * 2	8
/	Division	10 / 2	5
=	Assignment	x = 10	Assigns 10

4.5 Console Output and Alerts

- console.log() shows messages in the console for debugging.
- alert() pops up a message box in the browser.

Example:

javascript

```
console.log("Welcome to JS!");
alert("Hello, world!");
```

4.6 Writing Your First Script

Try this script in Replit's JavaScript environment:

javascript

```
let userName = prompt("What is your name?");
alert("Hello, " + userName + "! Welcome to JavaScript.");
console.log("User " + userName + " has started the program.");
```

4.7 Check Your Understanding

- What does let do in JavaScript?
- What data types have you learned?
- How do you display a message in the console?
- What is the difference between alert() and console.log()?

4.8 Practice Exercises

1. Declare variables for your favorite food, number of pets, and if you like coding (true/false).
2. Write a script that asks the user for their age and logs a message to the console.
3. Create a simple calculator that adds two numbers and alerts the result.
4. Experiment with changing variable values and logging them.

Chapter Summary

- JavaScript adds interactivity to websites and runs in browsers.
- Variables store data, with types like strings, numbers, and booleans.
- Operators perform calculations and assignments.
- console.log() and alert() are basic ways to output messages.
- You can write and run JS scripts directly in Replit.

Mini Project: Interactive Greeting Page

Goal: Build a simple web page that interacts with the user.

Requirements:

- Use prompt() to ask the user's name
- Use alert() to greet them personally
- Log messages to the console for debugging
- Add comments explaining your code

Reflection Prompt

Think about websites you visit daily. How does interactivity improve your experience? Write a paragraph about one interactive feature you like and why.

Chapter 5: Conditional Statements

Learning Objectives

By the end of this chapter, you will:

- Understand what conditional statements are and how they control program flow
- Learn the syntax for if, else if, and else statements in JavaScript
- Apply conditionals to real-world decision-making scenarios
- Practice writing conditional code with guided exercises

5.1 What Are Conditional Statements?

Conditional statements let your program make decisions and execute code only when certain conditions are met. This is how you add logic and interactivity to your programs.

Real-World Analogy

Imagine you're deciding what to wear:

- If it's raining, you take an umbrella.
- Else if it's sunny, you wear sunglasses.
- Otherwise, you dress normally.

This decision-making process is like conditional statements in programming.

5.2 Basic if Statement Syntax

```
if (condition) {
  // code to run if condition is true
```

```
}
```

Example:

javascript

```
let temperature = 30;

if (temperature > 25) {
  console.log("It's hot outside!");
}
```

If temperature is greater than 25, the message will be logged.

5.3 Using else and else if

These allow you to add more choices:

```
if (temperature > 30) {
  console.log("It's very hot!");
} else if (temperature > 20) {
  console.log("The weather is nice.");
} else {
  console.log("It's a bit cold.");
}
```

The program checks conditions in order and runs the first true branch.

5.4 Comparison Operators

Common operators used in conditions:

Operator	Meaning	Example
==	Equal to	x == 10
===	Equal value & type	x === 10
!=	Not equal	x != 5
<	Less than	x < 10
>	Greater than	x > 5
<=	Less than or equal	x <= 7
>=	Greater than or equal	x >= 3

5.5 Logical Operators

Combine multiple conditions:

Operator	Meaning	Example
&&	AND	x > 5 && x < 10
`		`
!	NOT	!(x == 10)

5.6 Hands-On: Temperature Checker

```
let temp = prompt("Enter the current temperature:");

if (temp > 30) {
  alert("It's very hot today!");
} else if (temp > 20) {
  alert("The weather is nice.");
} else {
  alert("It's a bit chilly.");
}
```

Run this in Replit and test different temperatures.

5.7 Check Your Understanding

- What does the else clause do?
- How does else if differ from if?
- What operator would you use to check if two values are exactly equal?
- How do logical AND (&&) and OR (||) differ?

5.8 Practice Exercises

1. Write a program that asks the user for their age and alerts whether they are a child (under 12), a teenager (12-18), or an adult (over 18).
2. Create a password checker that alerts “Access granted” if the password equals "letmein" and “Access denied” otherwise.
3. Combine conditions using && and || to check if a number is within or outside a range.

Chapter Summary

- Conditional statements control which code runs based on conditions.
- Use if, else if, and else to add multiple branches.
- Comparison and logical operators help build complex conditions.
- Conditionals make your programs responsive and dynamic.

Mini Project: Simple Quiz Game

Goal: Build a quiz that asks a question and responds differently based on the user's answer.

Requirements:

- Use prompts to ask a question
- Use conditionals to check answers

- Provide feedback with alerts or console messages
- Comment your code clearly

Reflection Prompt

Think about how conditional logic helps make websites interactive and personalized. Write a short paragraph on a website feature that uses decision-making behind the scenes.

Chapter 6: Loops and Iteration

Learning Objectives

By the end of this chapter, you will:

- Understand what loops are and why they are essential in programming
- Learn how to use for and while loops in JavaScript
- Write loops to automate repetitive tasks
- Practice coding exercises involving loops

6.1 What Are Loops?

Loops allow you to run the same block of code multiple times without rewriting it.

Real-World Analogy

Imagine you need to send a birthday card to every member of your family. Instead of writing each card separately, you prepare one card and send it repeatedly — that's what loops do for code.

6.2 The for Loop

The for loop runs a block of code a specific number of times.

Syntax:

javascript
CopyEdit

```
for (let i = 0; i < 5; i++) {
  console.log("Iteration number " + i);
}
```

- **Initialization:** let i = 0 starts the counter
- **Condition:** i < 5 runs the loop while true
- **Increment:** i++ increases i by 1 after each loop

Output:

typescript
CopyEdit

```
Iteration number 0
Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
```

6.3 The while Loop

The while loop continues as long as a condition remains true.

Syntax:

javascript
CopyEdit

```
let count = 0;

while (count < 5) {
  console.log("Count is " + count);
  count++;
}
```

6.4 Loop Control Statements

- **break**: exits the loop early
- **continue**: skips the current iteration and moves to the next

6.5 Hands-On: Print Numbers

Write a loop to print numbers from 1 to 10.

```
javascript
CopyEdit
for (let num = 1; num <= 10; num++) {
  console.log(num);
}
```

6.6 Check Your Understanding

- What are the three parts of a for loop?
- How does a while loop differ from a for loop?
- What does the break statement do?
- When would you use continue?

6.7 Practice Exercises

1. Use a for loop to print all even numbers between 2 and 20.
2. Use a while loop to count down from 10 to 1.
3. Write a loop that prints the first 10 Fibonacci numbers.
4. Create a loop that skips printing the number 5 using continue.

Chapter Summary

- Loops automate repetitive tasks by running code multiple times.
- for loops run a known number of times; while loops run while a condition is true.
- Use break and continue to control loop execution.

Mini Project: Number Guessing Game

Goal: Build a simple game where the user guesses a number between 1 and 10.

Requirements:

- Use a loop to allow multiple guesses
- Give feedback if the guess is too high, too low, or correct
- Exit the loop when the user guesses correctly or after 5 tries

Reflection Prompt

How can loops make programming more efficient and less error-prone? Reflect on how you might use loops in everyday tasks or projects.

Chapter 7: JavaScript Functions

Learning Objectives

By the end of this chapter, you will:

- Understand what functions are in JavaScript and why they are important
- Learn how to define and call functions
- Use parameters and return values in your code
- Apply functions to build more modular, reusable code
- Practice with hands-on coding in Replit or SoloLearn

7.1 What Are Functions?

A **function** is a reusable block of code that performs a specific task. You can define it once and call it many times.

💡 Real-World Analogy

Think of a function like a blender:

- You put in fruits (inputs)
- Press a button (call the function)
- It gives you a smoothie (output)

7.2 Defining a Function

javascript
CopyEdit

```
function greet() {
  console.log("Hello, welcome to JavaScript!");
}
```

Calling the Function

javascript
CopyEdit

```
greet();  // Output: Hello, welcome to JavaScript!
```

You can call a function anywhere in your code once it's defined.

7.3 Functions with Parameters

Functions can accept **parameters**, which are variables passed into the function.

javascript
CopyEdit

```
function greetUser(name) {
  console.log("Hello, " + name + "!");
}

greetUser("Thabo");  // Output: Hello, Thabo!
```

You can pass multiple parameters:

javascript
CopyEdit

```
function add(a, b) {
  console.log(a + b);
}

add(4, 5);  // Output: 9
```

7.4 Return Values

Functions can **return** a result using the return keyword.

```
javascript
CopyEdit
function square(number) {
  return number * number;
}

let result = square(6);
console.log(result);  // Output: 36
```

The return statement ends the function and sends a value back to where it was called.

7.5 Function Expressions and Arrow Functions

In JavaScript, functions can be stored in variables.

```
javascript
CopyEdit
const sayHi = function() {
  console.log("Hi there!");
};

sayHi();  // Output: Hi there!
```

Arrow functions (modern syntax):

```
javascript
CopyEdit
const sayHello = () => {
  console.log("Hello!");
};

sayHello();  // Output: Hello!
```

Use Case: Input Validation on a Web Page

You can use functions to:

- Validate form data
- Respond to user clicks
- Calculate totals dynamically

Example:

javascript
CopyEdit

```
function validateEmail(email) {
  return email.includes("@");
}
```

Check Your Understanding

1. What's the difference between a function and a variable?
2. How do you return a value from a function?
3. Can you have multiple parameters?
4. What are arrow functions used for?

Practice Exercises

1. Write a function multiply() that takes two numbers and returns their product.
2. Create a function isEven() that returns true if a number is even, otherwise false.
3. Build a function that calculates the area of a triangle using base and height.
4. Write a function greetUser(name, timeOfDay) that greets someone based on time.

Chapter Summary

- Functions help organize and reuse code

- Use function to define and call reusable tasks
- Parameters allow input, return allows output
- Arrow functions are a modern, concise way to write functions

Mini Project: Interactive Tip Calculator

Goal: Create a simple tip calculator for a restaurant bill.

Requirements:

- Ask the user for bill total and tip percentage (use prompt())
- Use a function to calculate the tip
- Display the total amount including tip using alert() or console.log()

Sample Starter Code:

javascript
CopyEdit

```
function calculateTip(bill, tipPercent) {
  return bill * (tipPercent / 100);
}
```

Reflection Prompt

Why is using functions better than copying and pasting code blocks? Think about a website with a contact form — what could be handled by a function?

Chapter 8: JavaScript Arrays

Learning Objectives

By the end of this chapter, you will:

- Understand what arrays are and why they're useful in JavaScript
- Learn how to create, access, update, and loop through arrays
- Use built-in array methods to modify and organize data
- Practice with hands-on examples and exercises in Replit or SoloLearn

8.1 What Is an Array?

An **array** is a list-like data structure that stores multiple values in one variable. You can access each item by its position (called an index).

Real-World Analogy

Think of an array like a row of mailboxes:

- Each mailbox (index) holds one item (value)
- You access them by their number (position)

8.2 Creating an Array

javascript
CopyEdit
let colors = ["red", "green", "blue"];

You can store:

- Strings: ["HTML", "CSS", "JavaScript"]
- Numbers: [10, 20, 30]
- Mixed types: ["Hello", 5, true]

8.3 Accessing and Modifying Items

JavaScript arrays use **zero-based indexing**.

```
javascript
CopyEdit
console.log(colors[0]);  // Output: red
colors[1] = "yellow";    // Change green to yellow
```

8.4 Adding and Removing Items

Method	What It Does	Example
.push()	Adds to the **end**	colors.push("purple")
.unshift()	Adds to the **beginning**	colors.unshift("orange")
.pop()	Removes the **last** item	colors.pop()
.shift()	Removes the **first** item	colors.shift()
.splice()	Adds/removes items anywhere	colors.splice(1, 1, "pink")

8.5 Looping Through Arrays

for loop:

```
javascript
CopyEdit
for (let i = 0; i < colors.length; i++) {
  console.log(colors[i]);
}
```

for...of loop (simpler syntax):

```
javascript
CopyEdit
for (let color of colors) {
  console.log(color);
}
```

8.6 Useful Array Methods

Method	Purpose	Example
.length	Returns the number of items	colors.length
.includes()	Checks if item exists	colors.includes("blue")
.indexOf()	Gets the position of an item	colors.indexOf("green")
.join()	Converts array to string	colors.join(", ")
.reverse()	Reverses order	colors.reverse()
.sort()	Sorts alphabetically	colors.sort()

Check Your Understanding

1. How do you add an item to the end of an array?
2. What does .splice() allow you to do?
3. What is the difference between for and for...of loops?
4. How would you check if “HTML” is in a list of technologies?

Practice Exercises

1. Create an array of your 5 favorite websites.
2. Use .push() to add one more, then print the full list.
3. Remove the first item using .shift()
4. Loop through your updated array and print each website in uppercase.

Chapter Summary

- Arrays store multiple values in one variable
- You access values using their index
- JavaScript provides powerful array methods for editing and looping
- Arrays are essential for working with lists of data in websites (like menus, form values, API responses)

Mini Project: To-Do List

Goal: Create a simple command-line to-do list app using arrays.

Steps:

1. Create an empty array called tasks
2. Use prompt() to ask the user to enter a task
3. Add it to the array using .push()
4. Ask if they want to enter another
5. After adding tasks, display all tasks with a loop

Bonus: Let users remove tasks by index using .splice().

Reflection Prompt

Think about the different ways websites use lists: navigation menus, product listings, FAQs, etc. How would you store and manage that data using arrays?

Chapter 9: JavaScript Objects and Maps

Learning Objectives

By the end of this chapter, you will:

- Understand what objects are in JavaScript and how to create them
- Learn how to store and access key-value pairs
- Update, delete, and loop through object properties
- Get introduced to Map and when to use it over objects
- Apply objects and maps to real web development use cases

9.1 What Is an Object?

An **object** is a collection of key-value pairs. It's used to group related data and behavior together.

💡 Real-Life Analogy

Think of an object like a **contact card**:

- Name: "Alice"
- Phone: "0781234567"
- Email: "alice@example.com"

In JavaScript:

javascript
CopyEdit
```
let contact = {
  name: "Alice",
  phone: "0781234567",
  email: "alice@example.com"
};
```

9.2 Accessing Object Data

Use dot notation:

javascript
CopyEdit
```
console.log(contact.name);  // Output: Alice
```

Or bracket notation:

javascript
CopyEdit

```
console.log(contact["phone"]);  // Output: 0781234567
```

9.3 Modifying Objects

Update a value:

```
javascript
CopyEdit
contact.phone = "0712345678";
```

Add a new property:

```
javascript
CopyEdit
contact.address = "123 Main Street";
```

Delete a property:

```
javascript
CopyEdit
delete contact.email;
```

9.4 Looping Through Objects

Use for...in loop:

```
javascript
CopyEdit
for (let key in contact) {
  console.log(key + ": " + contact[key]);
}
```

9.5 Object Methods

Objects can also hold functions as values (called **methods**):

javascript
CopyEdit

```
let user = {
  name: "Zama",
  greet: function () {
    console.log("Hello, " + this.name);
  }
};

user.greet();  // Output: Hello, Zama
```

9.6 Introduction to Map

A Map is a newer JavaScript data type that also stores key-value pairs, but with more flexibility:

- Keys can be **any** type (not just strings)
- Keeps order of insertion
- Has special methods like .set(), .get(), .has(), .delete()

Example:

javascript
CopyEdit

```
let settings = new Map();
settings.set("theme", "dark");
settings.set("fontSize", 14);

console.log(settings.get("theme"));  // Output: dark
```

When to Use Object vs. Map?

Use Case	Use Object	Use Map
Fixed structure (e.g., user)	✅	
Dynamic key-value store		✅
Keys are strings	✅	✅
Keys are other types		✅
Order matters		✅

Check Your Understanding

1. What is the difference between an object and an array?
2. How do you add a new key-value pair to an object?
3. What does the for...in loop do?
4. When might you use a Map instead of a regular object?

Practice Exercises

1. Create an object called student with properties name, age, and grade.
2. Add a new property email, then update the grade.
3. Loop through all the keys in the student object and print their values.
4. Create a Map called productSettings and store two key-value pairs: "currency" → "ZAR" and "inStock" → true.

Chapter Summary

- JavaScript **objects** store data as key-value pairs
- Use dot or bracket notation to access or modify properties
- Use for...in to loop through object keys

- Map is a flexible alternative for dynamic, ordered key-value pairs

Mini Project: User Profile Builder

Goal: Create a web-based user profile object and display user details.

Steps:

1. Use a prompt() to collect user info: name, email, and age
2. Store the info in an object
3. Create a method displayProfile() that prints a formatted user summary
4. Use console.log() or alert() to display the profile

Reflection Prompt

Objects are used all over the web: from storing user data to representing elements in the DOM. Think of a website you use often. What types of objects might be working behind the scenes?

Chapter 10: Error Handling and Debugging in JavaScript

Learning Objectives

By the end of this chapter, you will:

- Understand common types of errors in JavaScript
- Learn how to use try, catch, and finally to handle errors gracefully
- Know how to use debugging tools like the browser console and breakpoints
- Practice identifying and fixing errors in code
- Build confidence in troubleshooting code issues

10.1 Types of Errors

- **Syntax Errors:** Mistakes in the code structure that prevent it from running.
  Example: Missing a closing bracket.
- **Runtime Errors:** Errors that occur while the code is running, like trying to use an undefined variable.
- **Logical Errors:** Code runs but produces incorrect results due to a mistake in logic.

10.2 Using try...catch to Handle Errors

try lets you test code for errors, while catch handles them without breaking your program.

javascript
CopyEdit

```
try {
  let user = JSON.parse('{"name": "Ava"');  // Missing closing brace
} catch (error) {
  console.log("There was an error: " + error.message);
}
```

Output:
There was an error: Unexpected end of JSON input

The program continues running instead of crashing.

10.3 The finally Block

Code inside finally runs whether or not an error occurs:

javascript
CopyEdit

```
try {
  // risky code
} catch (error) {
  // handle error
} finally {
  console.log("This always runs.");
}
```

10.4 Debugging Tools

Browser Console

- Open with F12 or right-click → Inspect → Console
- Use console.log() to print variables and check code flow

javascript
CopyEdit
console.log("Current value: ", value);

Breakpoints

- Pause execution to inspect variables line-by-line
- Set breakpoints in browser developer tools under "Sources"

10.5 Best Practices for Debugging

- Read error messages carefully
- Test code in small parts
- Use comments to isolate issues
- Keep your code organized and clean
- Use descriptive variable and function names

Check Your Understanding

1. What's the difference between syntax and runtime errors?
2. How does try...catch improve code stability?
3. When would you use finally?
4. Why is console.log() useful in debugging?

Practice Exercises

1. Write code with a syntax error, then fix it using try...catch.
2. Create a function that divides two numbers and handles division by zero gracefully.
3. Use console.log() to debug a loop that doesn't behave as expected.
4. Set breakpoints in your browser's developer tools and step through a simple function.

Chapter Summary

- Errors can stop your code; handling them keeps your app running
- Use try, catch, and finally to manage errors in JavaScript
- Debugging tools like the console and breakpoints help identify issues
- Effective debugging skills are essential for any web developer

Mini Project: Safe JSON Parser

Goal: Build a function to safely parse JSON strings and handle errors without crashing.

Example:

javascript
CopyEdit

```
function safeParse(jsonString) {
  try {
```

```
    return JSON.parse(jsonString);
  } catch (error) {
    return { error: "Invalid JSON" };
  }
}

console.log(safeParse('{"name": "John"}'));  // Outputs object
console.log(safeParse('{"name": "John"'));   // Outputs error object
```

Reflection Prompt

Think about a time when a small typo broke your code. How did debugging help you solve the problem? How can you use error handling in your future projects?

Chapter 11: Object-Oriented Programming (OOP) in JavaScript

Learning Objectives

By the end of this chapter, you will:

- Understand the basics of OOP concepts like classes and objects
- Learn how to create and use classes in JavaScript
- Explore properties, methods, and constructors
- Understand inheritance and how to extend classes
- Practice writing simple OOP code to model real-world objects

11.1 What is Object-Oriented Programming?

OOP is a programming style that organizes code around **objects**, which represent real-world entities with properties (data) and methods (functions).

💡 Real-Life Analogy

Think of a **Car** as an object:

- Properties: color, make, model, year
- Methods: start(), stop(), honk()

11.2 Classes and Objects

Creating a Class

javascript
CopyEdit

```
class Car {
  constructor(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year;
  }

  start() {
    console.log(` ${this.make} ${this.model} is starting. `);
  }
}
```

Creating an Object (Instance)

javascript
CopyEdit

```
let myCar = new Car("Toyota", "Corolla", 2020);
myCar.start();  // Output: Toyota Corolla is starting.
```

11.3 Properties and Methods

- **Properties** store data about the object.
- **Methods** are functions that describe actions the object can do.

11.4 The Constructor Method

The constructor is a special method that runs when you create a new object, initializing its properties.

11.5 Inheritance

Inheritance lets a class inherit properties and methods from another class.

```
javascript
CopyEdit
class ElectricCar extends Car {
  constructor(make, model, year, batteryLife) {
    super(make, model, year);
    this.batteryLife = batteryLife;
  }

  charge() {
    console.log(` ${this.make} ${this.model} is charging. `);
  }
}

let myElectricCar = new ElectricCar("Tesla", "Model 3", 2021, "100%");
myElectricCar.start();   // From Car class
```

```
myElectricCar.charge();  // From ElectricCar class
```

Check Your Understanding

1. What is the purpose of a class?
2. How do you create an object from a class?
3. What does super() do in a subclass?
4. How do methods differ from properties?

Practice Exercises

1. Create a class Person with properties name and age, and a method introduce() that prints a greeting.
2. Create two objects from Person with different data and call introduce() on both.
3. Create a subclass Student that inherits from Person and adds a property grade. Add a method study().
4. Instantiate a Student object and demonstrate calling inherited and new methods.

Chapter Summary

- OOP helps model real-world things as objects with properties and methods
- Classes are blueprints; objects are instances of classes
- Constructors initialize objects when created
- Inheritance allows classes to extend and reuse code

Mini Project: Simple Library System

Goal: Create a class-based system to manage books.

Steps:

1. Create a Book class with properties: title, author, and checkedOut (boolean)
2. Add methods: checkOut() and returnBook() to update the status
3. Create instances of books and simulate checking them in and out
4. Use console.log() to show the current status of each book

Reflection Prompt

How can organizing code with classes and objects help when building complex websites? Can you think of a feature on a website that might be represented as an object?

Chapter 12: Working with Modules and Libraries in JavaScript

Learning Objectives

By the end of this chapter, you will:

- Understand what modules are and why they help organize code
- Learn how to create and import/export modules in JavaScript
- Explore how to use third-party libraries to add features quickly
- Practice integrating libraries in simple web projects

12.1 What Are Modules?

Modules are separate files or pieces of code that contain specific functionality. They help keep your code clean, organized, and reusable.

Real-Life Analogy

Think of modules like **building blocks**: each block has a job, and you combine blocks to build a full website.

12.2 Creating and Exporting Modules

You can create a module by putting code in a separate .js file and export parts of it.

mathUtils.js:

javascript
CopyEdit

```
export function add(a, b) {
  return a + b;
}

export const PI = 3.1416;
```

12.3 Importing Modules

Import functions or variables from modules to use in your main code.

javascript
CopyEdit

```
import { add, PI } from './mathUtils.js';

console.log(add(2, 3));  // Output: 5
console.log(PI);        // Output: 3.1416
```

12.4 Default Exports

Modules can export one default item.

logger.js:

javascript

CopyEdit

```
export default function logMessage(message) {
  console.log(message);
}
```

Import with any name:

javascript
CopyEdit

```
import log from './logger.js';
log("Hello from logger!");
```

12.5 Using Third-Party Libraries

Libraries are collections of code others have written to make development faster.

How to include libraries:

- **CDN (Content Delivery Network):** Link libraries directly in your HTML
  Example: Using jQuery

html
CopyEdit

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

- **npm (Node Package Manager):** For larger projects (requires Node.js)

12.6 Example: Using Moment.js for Date Formatting

Add Moment.js via CDN:

html
CopyEdit

<script src="https://cdn.jsdelivr.net/npm/moment@2.29.4/moment.min.js"></script>

Use it in your JavaScript:

javascript
CopyEdit
console.log(moment().format('MMMM Do YYYY, h:mm:ss a'));

Check Your Understanding

1. What are the benefits of using modules in your code?
2. How do you export multiple items from a module?
3. What's the difference between named exports and default exports?
4. How do you add a library to your website using a CDN?

Practice Exercises

1. Create a module stringUtils.js that exports two functions: capitalize() and lowercase().
2. Import and use these functions in your main script.
3. Add a third-party library (like Lodash or Axios) to a simple HTML page using CDN.
4. Use a function from the library to complete a small task (e.g., debounce a button click).

Chapter Summary

- Modules help organize and reuse code efficiently
- Use export and import to share code between files
- Libraries let you add powerful features without reinventing the wheel
- CDNs are an easy way to include libraries in web projects

Mini Project: Modular Calculator

Goal: Build a simple calculator using modules.

Steps:

1. Create a module calculator.js with functions add(), subtract(), multiply(), and divide()
2. Export these functions
3. Import the module into your main script
4. Build a simple HTML interface and use the module functions to perform calculations based on user input

Reflection Prompt

How do you think using modules and libraries can speed up your website development process? Can you imagine working on a large website without modular code?

Chapter 13: Introduction to APIs and JSON

Learning Objectives

By the end of this chapter, you will:

- Understand what an API is and why developers use them

- Learn the basics of JSON (JavaScript Object Notation)
- Make a simple request to an API using JavaScript
- Use API data to display dynamic content on a web page

13.1 What Is an API?

API stands for **Application Programming Interface**. It allows different software systems to talk to each other.

💡 Real-Life Analogy

Think of an API like a **waiter in a restaurant**. You place your order, the waiter brings it to the kitchen (the server), and then returns with your food (the data).

13.2 What Is JSON?

JSON is a format used to send and receive data. It's easy for humans to read and for machines to understand.

Example JSON Data:

json
CopyEdit

```
{
  "name": "Laptop",
  "price": 14999,
  "inStock": true
}
```

- JSON uses key–value pairs
- Keys are always in quotes
- Values can be strings, numbers, booleans, arrays, or even nested objects

13.3 Fetching Data Using JavaScript

JavaScript provides a built-in function called fetch() to get data from APIs.

Example:

```
javascript
CopyEdit
fetch('https://api.example.com/products')
  .then(response => response.json())
  .then(data => {
    console.log(data);
  })
  .catch(error => console.error('Error:', error));
```

13.4 Displaying API Data on a Web Page

You can take data from an API and display it using the DOM.

Example:

```
html
CopyEdit
<ul id="product-list"></ul>

<script>
  fetch('https://fakestoreapi.com/products')
    .then(res => res.json())
    .then(data => {
      const list = document.getElementById('product-list');
      data.forEach(product => {
        const item = document.createElement('li');
        item.textContent = product.title;
        list.appendChild(item);
```

```
    });
  });
</script>
```

Check Your Understanding

1. What does API stand for and what does it do?
2. What does JSON stand for?
3. How does the fetch() function work in JavaScript?
4. What types of data can JSON hold?

Practice Exercises

1. Visit https://jsonplaceholder.typicode.com/posts in your browser. What kind of data do you see?
2. Use fetch() to get the list of posts from that API and display only the titles on a webpage.
3. Format and style the list using basic CSS.
4. Add error handling to your fetch function.

Chapter Summary

- APIs let your website get data from other websites or services
- JSON is the format in which most APIs send their data
- fetch() is a built-in JavaScript method to request data
- You can use JavaScript to dynamically update your page with the API data

Mini Project: Build a Weather Widget

Goal: Create a weather widget that shows the current temperature of a selected city.

Steps:

1. Use the OpenWeatherMap API
2. Ask the user to enter a city name
3. Fetch the weather data for that city
4. Display the temperature, weather condition, and an icon
5. Style the widget using CSS

Reflection Prompt

Have you ever used a site that showed live updates (like sports scores, weather, or news)? Now that you know about APIs, how do you think those updates are powered?

Chapter 14: Intro to Data Visualization

Learning Objectives

- Understand what data visualization means and why it's useful
- Learn about basic data visualization tools for the web
- Create a simple chart using Chart.js

14.1 What is Data Visualization?

Data visualization is the representation of information in the form of a chart, graph, or map. It helps make complex data easier to understand.

Real-Life Analogy

Think of a **bar chart** showing your monthly expenses. It gives you a clear picture of where your money goes—better than just reading numbers!

14.2 Introducing Chart.js

Chart.js is a simple and powerful JavaScript library for creating charts.

How to Include Chart.js

Use a CDN to include Chart.js in your HTML file:

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

14.3 Creating a Simple Chart

Add a canvas element to your HTML:

```
<canvas id="myChart" width="400" height="200"></canvas>
```

Create a chart using JavaScript:

```
const ctx = document.getElementById('myChart').getContext('2d');
const myChart = new Chart(ctx, {
  type: 'bar',
  data: {
    labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'],
    datasets: [{
      label: '# of Votes',
```

```
        data: [12, 19, 3, 5, 2, 3],
        backgroundColor: 'rgba(75, 192, 192, 0.2)',
        borderColor: 'rgba(75, 192, 192, 1)',
        borderWidth: 1
      }]
    },
    options: {
      scales: {
        y: {
          beginAtZero: true
        }
      }
    }
});
```

Chapter Summary

- Data visualization is useful for understanding large sets of data
- Chart.js is a great tool for creating charts with JavaScript
- You can quickly add it to your website using a CDN and a few lines of code

Mini Project: Create a Feedback Chart

Goal: Use Chart.js to display student feedback on your course

Steps:

1. Create an HTML page with a canvas element
2. Use Chart.js to show satisfaction ratings (e.g., Excellent, Good, Average, Poor)
3. Style the chart and page

Reflection Prompt

Have you ever looked at a dashboard or chart and immediately understood something complex? How might you use data visualization in your own web projects?

Chapter 15: Final Project and Portfolio Launch

Learning Objectives

By the end of this chapter, you will:

- Plan, build, and deploy a complete personal website
- Apply all the skills learned: HTML, CSS, JavaScript, forms, APIs, modules, and libraries
- Publish your website using GitHub Pages or Replit
- Create a portfolio to showcase your skills

15.1 Project Overview

You will create a fully functional, responsive personal website that demonstrates everything you've learned in this course.

Example Project Ideas:

- A portfolio/resume site
- A product landing page with form submissions
- A weather or quote app using an API
- A small blog or digital journal

15.2 Planning Your Site

Use the following checklist to plan your final project:

- What is the **purpose** of your site?
- Who is your **audience**?
- What pages will you include (Home, About, Contact, etc.)?
- What **features** will you build in?
    - Forms?
    - API integration?
    - Image gallery?
    - Light/dark mode toggle?

15.3 Tools You Can Use

- **Editor**: Replit or Visual Studio Code
- **Libraries**: Chart.js, jQuery, Bootstrap (optional)
- **Deployment**: GitHub Pages or Replit hosting

15.4 Project Requirements

Your final project must include:

- Valid HTML and CSS
- At least 1 form
- JavaScript for interactivity
- At least 1 external library or module
- Responsive design
- Clear navigation
- Commented code

Chapter Summary

- The final project ties all the course concepts together
- You are now capable of building and launching real websites
- A portfolio site helps you showcase your skills to potential employers or clients

Reflection Prompt

How has your understanding of website development changed since Chapter 1? What will you build next now that you have these skills?

Glossary of Web Development Terms

This glossary includes simple, beginner-friendly definitions of key terms used throughout this textbook.

API (Application Programming Interface) – A set of rules that allows different programs to communicate with each other.

Attribute – Extra information in an HTML tag, such as src in <img src="logo.png">.

Boolean – A data type that can be either true or false.

CDN (Content Delivery Network) – A way to load resources like scripts from external servers.

Class (CSS) – A reusable styling identifier used in HTML elements. For example: .

CSS (Cascading Style Sheets) – A language that describes how HTML elements should be styled.

DOM (Document Object Model) – The structure of a web page, which JavaScript can interact with and change.

Function (JavaScript) – A reusable block of code that performs a task.

HTML (HyperText Markup Language) – The basic language used to structure web content.

JavaScript – A programming language used to make websites interactive.

Library – A collection of pre-written code that performs common tasks.

Loop – A programming structure that repeats a block of code.

Module – A separate file that contains code you can reuse by importing it.

Responsive Design – A way of designing websites to look good on all screen sizes.

Selector (CSS) – A pattern used to target HTML elements for styling.

String – Text in programming, usually written inside quotes, like "Hello".

Tag (HTML) – The basic building block of HTML, like <p> for paragraphs or <a> for links.

Variable – A container for storing data in JavaScript.

Viewport – The visible area of a webpage on a user's screen.

Wireframe – A sketch or plan of a website's layout.

Appendix A: Helpful Tools and Resources

Practice Platforms

- SoloLearn – Interactive coding tutorials
- Replit – Online code editor for building and testing websites
- CodePen – Great for experimenting with HTML, CSS, and JS

Web Dev Tools

- Visual Studio Code
- GitHub – For version control and portfolio hosting
- Canva – Design graphics for your websites

Further Learning

- freeCodeCamp
- MDN Web Docs
- CSS-Tricks

Index

(Numbers refer to chapter.section format)