JAVASCRIPT

QuickStart Guide®

AUDIOBOOK COMPANION

QuickStart Guides®

TABLE OF CONTENTS

CHAPTER 1 4

CHAPTER 2 13

CHAPTER 3 22

CHAPTER 4 33

CHAPTER 5 40

CHAPTER 6 45

CHAPTER 7 56

CHAPTER 8 61

CHAPTER 9 73

CHAPTER 10 79

CHAPTER 11 90

CHAPTER 12 107

CHAPTER 13 129

CHAPTER 14 134

CHAPTER 15 142

CHAPTER 16 148

CHAPTER 17 161

CHAPTER 18 167

APPENDIX I 170

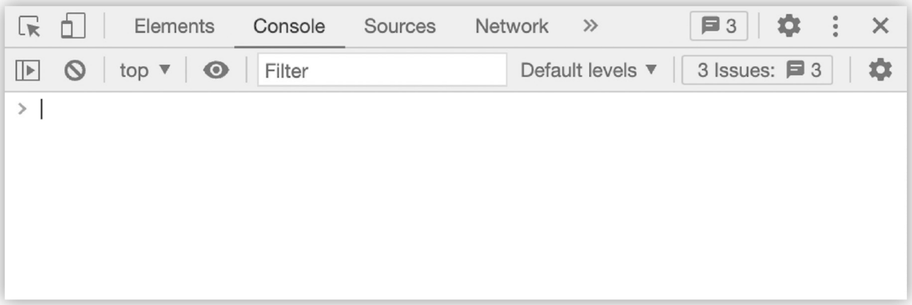
APPENDIX II 180

APPENDIX III 182

APPENDIX IV 188

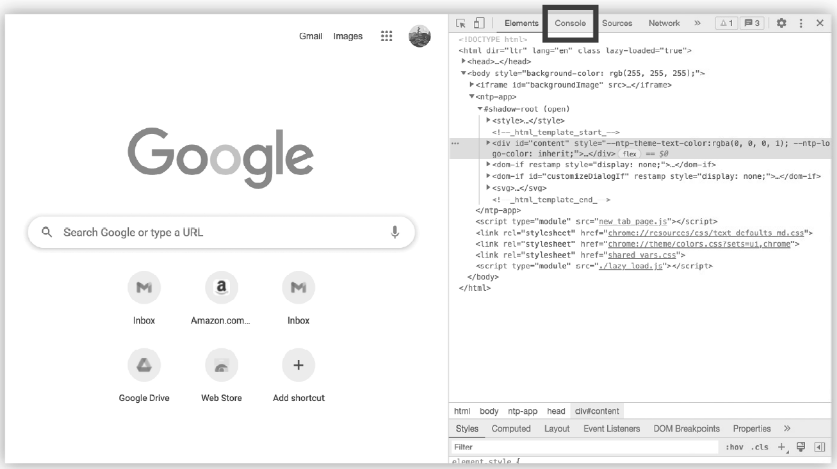
CHAPTER 1
Getting Started with JavaScript

Figure 1



The JavaScript console in Google Chrome.

Figure 2



```
<!DOCTYPE html>
<html dir="ltr" lang="en" class lazy-loaded="true">
 ▶<head>…</head>
 ▼<body style="background-color: rgb(255, 255, 255);">
   ▶<iframe id="backgroundImage" src>…</iframe>
   ▼<ntp-app>
     ▼#shadow-root (open)
       ▶<style>…</style>
        <!--_html_template_start_-->
...    ▶<div id="content" style="--ntp-theme-text-color:rgba(0, 0, 0, 1); --ntp-lo
        go-color: inherit;">…</div> flex == $0
       ▶<dom-if restamp style="display: none;">…</dom-if>
       ▶<dom-if id="customizeDialogIf" restamp style="display: none;">…</dom-if>
       ▶<svg>…</svg>
        <!--_html_template_end_-->
    </ntp-app>
    <script type="module" src="new_tab_page.js"></script>
    <link rel="stylesheet" href="chrome://resources/css/text_defaults_md.css">
    <link rel="stylesheet" href="chrome://theme/colors.css?sets=ui,chrome">
    <link rel="stylesheet" href="shared_vars.css">
    <script type="module" src="./lazy_load.js"></script>
  </body>
</html>
element.style {
```

The Elements tab in the Google Chrome developer tools.
Note the *Console* tab to the right of *Elements* (circled in this figure).

Snippet 01-01.js

```
console.log("Hello, World!")
```

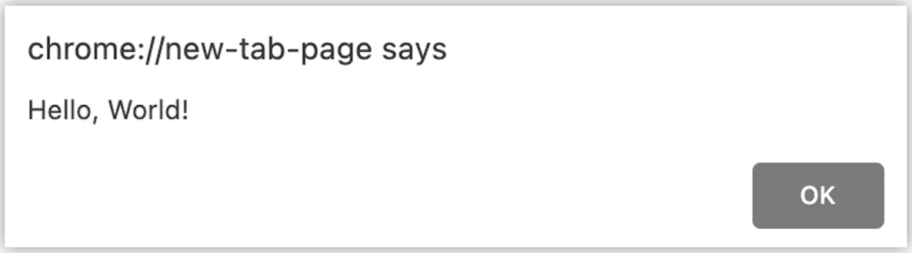
Code Line 1-1:

```
console.log("Hello, Robert!")
```

Code Line 1-2:

```
alert("Hello, World!")
```

Figure 3

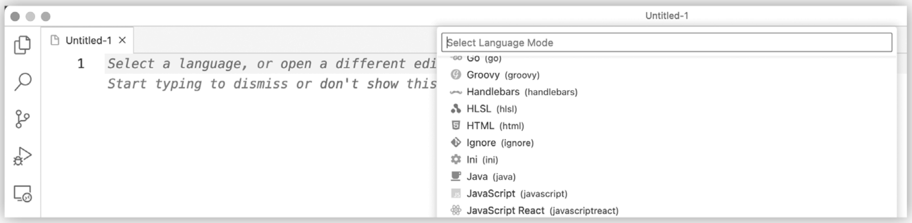


chrome://new-tab-page says

Hello, World!

Alert pop-up window showing "Hello, World!"

Figure 4



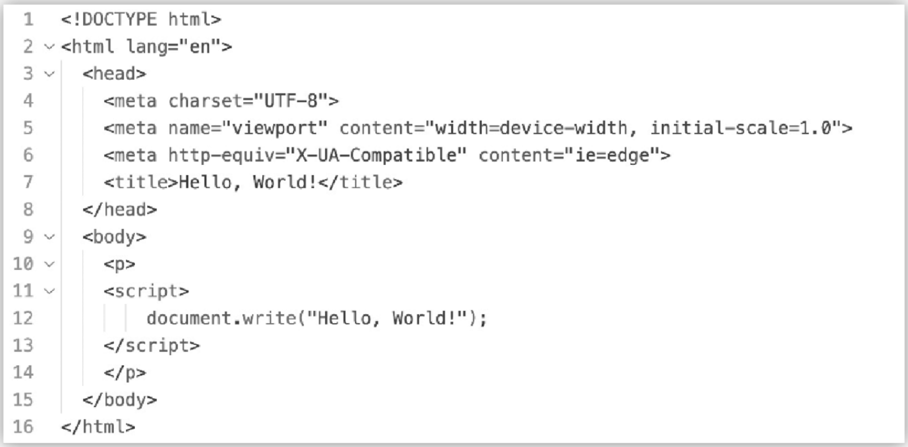
```
Select a language, or open a different edi
Start typing to dismiss or don't show this
```

The file-type drop-down menu in Visual Studio Code.

Snippet 01-02.html

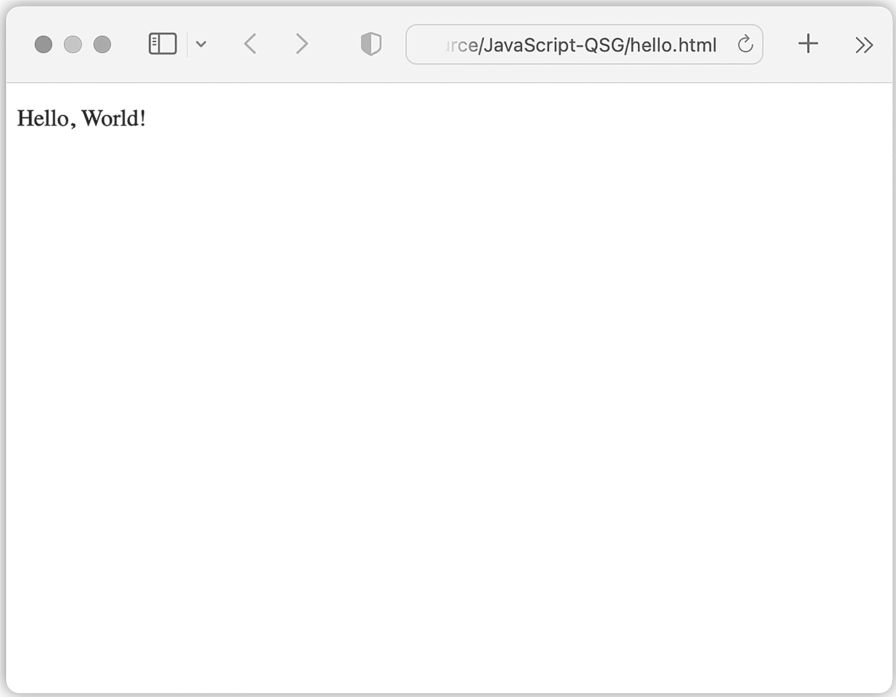
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Hello, World!</title>
    </head>
    <body>
        <p>
        <script>
            document.write("Hello, World!");
        </script>
        </p>
    </body>
</html>
```

Figure 5


```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Hello, World!</title>
  </head>
  <body>
    <p>
    <script>
        document.write("Hello, World!");
    </script>
    </p>
  </body>
</html>
```

The Hello, World! HTML as seen in Visual Studio Code. Note the lines that illustrate the indented portions of code. For example, everything between the start of the head element (`<head>`) and the end of the head element (`</head>`) is indented, and this convention continues throughout the code—indenting additional times if needed.

Figure 6



Hello, World!

The Hello, World! message in the browser.

Snippet 01-03.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Hello, World!</title>
    </head>
    <body>
        <div id="content"></div>
    </body>
</html>
```

Code Line 1-3:

```
document.write("Hello, World!");
```

Code Line 1-4:

```
<body>
    <p>
    <script>
        document.write("Hello, World!");
    </script>
    </p>
</body>
```

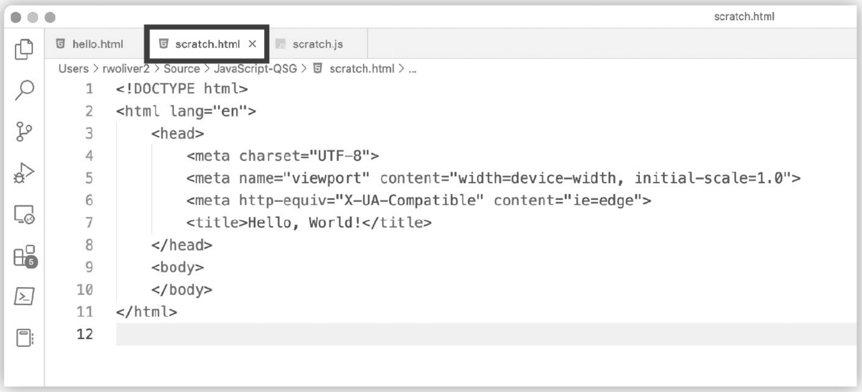
Code Line 1-5:

```
<body>
    <p>
        Hello, World!
    </p>
</body>
```

Code line 1-6:

```
document.getElementById("content").textContent = "Hello, World!"
```

Figure 7


hello.html | scratch.html | scratch.js

Users > rwoliver2 > Source > JavaScript-QSG > scratch.html > ...

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Hello, World!</title>
    </head>
    <body>
    </body>
</html>
```

The `scratch.html` file in Visual Studio Code.
The tab containing the HTML file is circled.

Code Line 1-7:

```
<script src=”scratch.js”></script>
```

Code Line 1-8:

```
<!DOCTYPE html>
<html lang=”en”>
    <head>
        <meta charset=”UTF-8”>
        <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
        <meta http-equiv=”X-UA-Compatible” content=”ie=edge”>
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id=”content”></div>
    </body>
    <script src=”scratch.js”></script>
</html>
```

Code Line 1-9:

```
// A nice greeting
console.log(“Hello, World!”)
```

Code Line 1-10:

```
console.log(“Hello, World!”) // A nice greeting
```

Code Line 1-11:

```
/*
This displays a nice greeting.
Hopefully the user will like it.
*/
console.log(“Hello, World!”)
```

Code Line 1-12:

```
console.log(“This line will run.”)
// console.log(“This line won’t.”)
console.log(“But this one will.”)
```

Code Line 1-13:

```
console.log(“Hello, World!”);
console.log(“Hello, World!”)
```

Code Line 1-14:

```
console.log(“Hello, World!”); console.log(“And Hello Again!”);
```

Code Line 1-15:

```
// For now, display a quick message in the console.
console.log(“ClydeBank Coffee Shop is now open!”)
```

Snippet index.html

```
<!DOCTYPE html>
<html lang=”en”>
<head>
    <meta charset=”UTF-8”>
    <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
    <link rel=”stylesheet” href=”style.css”>
    <title>ClydeBank Coffee Shop</title>
</head>
<body>
    <header>
        <div class=”logo-container”>
            <img src=”coffee-cup.svg” alt=”Coffee Cup” class=”logo”>
        </div>
        <h1>ClydeBank Coffee Shop</h1>
    </header>
    <section class="welcome">
        <h2>Welcome</h2>
        <p>Welcome to ClydeBank Coffee Shop! We serve the best coffee in town.</p>
    </section>
    <section class="shop">
        <h2>Shop</h2>
        <ul class="coffee-list">
            <li>Regular Coffee - $3.00</li>
            <li>Espresso - $3.50</li>
            <li>Cappuccino - $4.00</li>
            <li>Latte - $4.25</li>
        </ul>
    </section>
    <section class="about-us">
        <h2>About Us</h2>
        <p>Our mission is to provide the highest quality coffee to our community.</p>
    </section>
    <section class="contact-us">
        <h2>Contact Us</h2>
        <p>Email us at <a href="mailto:support@clydebankmedia.com">support@clydebankme-
dia.com</a> or call us at (800) 340-3069.</p>
    </section>
    <footer>
        <p>Copyright &copy; 2024 ClydeBank Coffee Shop</p>
    </footer>
    <script src="shop.js"></script>
</body>
</html>
```

Snippet style.css

```
/* Reset some default styles in browsers */
body, h1, h2, p, ul {
    margin: 0;
    padding: 0;
}

/* Add margin below h2 tags */
h2 {
    margin-bottom: 0.4em;
}

/* Apply background color to body */
body {
    background-color: #F4F1DE;
    font-family: Arial, sans-serif;
}
```

```
/* Header styles */
header {
    background-color: #3B1C0B;
    padding: 20px;
    text-align: center;
    color: white;
}

.logo-container {
    display: inline-block;
}

.logo {
    height: 50px;
    margin-right: 20px;
}

/* Section styles */
section {
    padding: 40px 20px;
    text-align: center;
}

.welcome {
    background-color: #C08552;
    color: white;
}

.shop .coffee-list {
    list-style: none;
}

.about-us {
    background-color: #C08552;
    color: white;
}

/* Footer styles */
footer {
    background-color: #3B1C0B;
    padding: 20px;
    text-align: center;
    color: white;
}
```

Snippet shop.js

```
// For now, display a quick message in the console.
console.log("ClydeBank Coffee Shop is now open!")
```

Snippet coffee-cup.svg

```
<svg width="64" height="64" xmlns="http://www.w3.org/2000/svg">
   <rect x="12" y="20" rx="12" ry="12" width="36" height="32" fill="#cad317" />
   <path d="M 46,28 A 10,10 0 0,1 46,40" fill="none" stroke="#cad317" stroke-
width="6"/>
   <ellipse cx="30" cy="32" rx="14" ry="6" fill="#000000" />
</svg>
```

CHAPTER 2
Using Variables

Code Line 2-1:

```
firstName = “Robert”
```

Code Line 2-2:

```
console.log(“Hello, “ + firstName)
```

Code Line 2-3:

```
console.log(“Hello, “ + firstName + “!”)
```

Code Line 2-4:

```
console.log(“Hello, World!”)
```

Code Line 2-5:

```
console.log(“Hello, “ + firstName + “!”)
```

Code Line 2-6:

```
console.log(`Hello, ${firstName}!`)
```

Code Line 2-7:

```
console.log(“Hello, “ + firstName + “ “ + middleName + “ “ + lastName + “!”)
```

Code Line 2-8:

```
console.log(`Hello, ${firstName} ${middleName} ${lastName}!`)
```

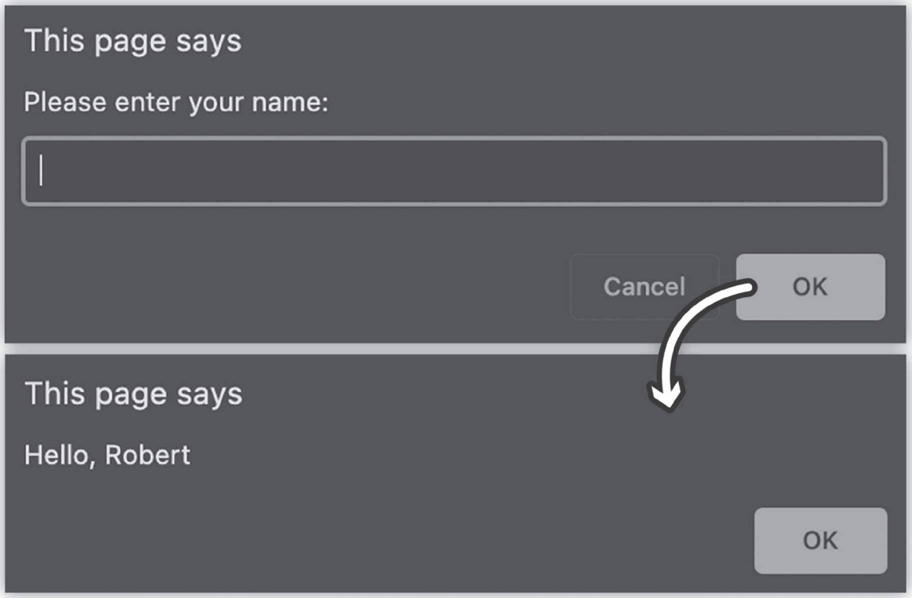
Code Line 2-9:

```
myReallyLongString = `This is a really long string.
So long in fact that it spans multiple lines.
Wow! What a string!`
```

Snippet 02-01.js

```
firstName = prompt("Please enter your name:")
alert(`Hello, ${firstName}`)
```

Figure 8


This page says

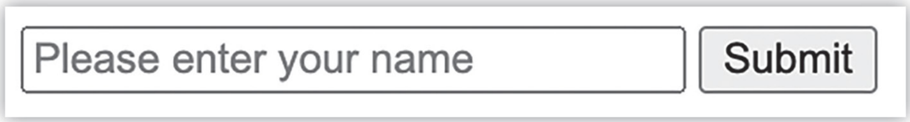
Please enter your name:

This page says

Hello, Robert

The prompt and result after entering "Robert".

Figure 9



A simple input text box in HTML.

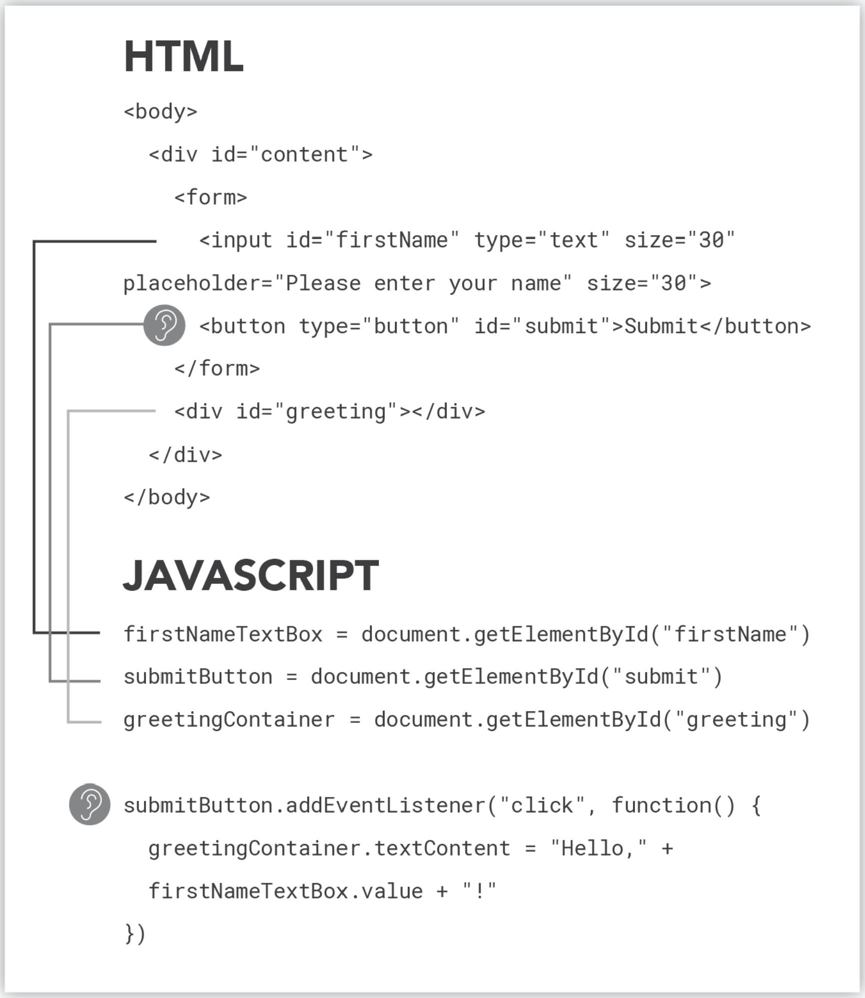
Snippet 02-02.html

```
<!DOCTYPE html>
<html lang=”en”>
    <head>
        <meta charset=”UTF-8”>
        <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
        <meta http-equiv=”X-UA-Compatible” content=”ie=edge”>
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id=”content”>
            <form>
                <input id=”firstName” type=”text” size=”30”
                    placeholder=”Please enter your name” size=”30”>
                <button type=”button” id=”submit”>Submit</button>
            </form>
            <div id=”greeting”></div>
        </div>
    </body>
    <script src=”scratch.js”></script>
</html>
```

Snippet 02-03.js

```
firstNameTextBox = document.getElementById(“firstName”)
submitButton = document.getElementById(“submit”)
greetingContainer = document.getElementById(“greeting”)

submitButton.addEventListener(“click”, function() {
    greetingContainer.textContent = “Hello, “ + firstNameTextBox.value + “!”
})
```

Figure 10



HTML

```
<body>
  <div id="content">
    <form>
      <input id="firstName" type="text" size="30"
placeholder="Please enter your name" size="30">
      <button type="button" id="submit">Submit</button>
    </form>
    <div id="greeting"></div>
  </div>
</body>
```

JAVASCRIPT

```
firstNameTextBox = document.getElementById("firstName")
submitButton = document.getElementById("submit")
greetingContainer = document.getElementById("greeting")

submitButton.addEventListener("click", function() {
  greetingContainer.textContent = "Hello," +
  firstNameTextBox.value + "!"
})
```

Code Line 2-10:

```
greetingContainer.textContent = "Hello, " + firstNameTextBox.value + "!"
```

Code Line 2-11:

```
"Hello, " + firstNameTextBox.value + "!"
```

Code Line 2-12:

```
"Hello, " + "Robert" + "!"
```

Code Line 2-13:

```
"Hello, Robert!"
```

Code Line 2-14:

```
greetingContainer.textContent = "Hello, Robert!"
```

Code Line 2-15:

```
a = 1
b = 3
c = 6
```

Code Line 2-16:

```
d = a + b + c
```

Code Line 2-17:

```
d = 100 + a
```

Code Line 2-18:

```
> a = 1
> b = 3
> c = 6
> d = a + b + c

10

> d = 100 + a

101
```

Code Line 2-19:

```
a = 1
b = 1
c = a + b
console.log("The answer to a + b is " + c.toString())
```

Code Line 2-20:

```
console.log("The answer to a + b is " + c.toString())
console.log("The answer to a + b is " + c)
```

Code Line 2-21:

```
The answer to a + b is 2
The answer to a + b is 2
```

Code Line 2-22:

```
a = "1"
b = parseInt(a, 10)
```

Code Line 2-23:

```
console.log('The value of pi is somewhat close to ${355/113}.')
```

Code Line 2-24:

```
The value of pi is somewhat close to 3.1415929203539825.
```

Code Line 2-25:

```
piString = `The value of pi is somewhat close to ${355/113}.`
console.log(piString)
```

Code Line 2-26:

```
a = 1
b = "Hello!"
```

Code Line 2-27:

```
var a = 1
var b = "Hello"
```

Code Line 2-28:

```
const a = 1
const b = "Hello"
```

Code Line 2-29:

```
const b = "Hello"
b = "Hello again!"
```

Code Line 2-30:

```
let a = 1
let b = "Hello"
```

Code Line 2-31:

```
let shoppingList = ["Milk", "Eggs", "Cheese", "Bread"]
let lotteryNumbers = [1, 4, 17, 19, 24, 7]
```

Code Line 2-32:

```
let numbersAndStrings = ["Los Angeles", 364, "Houston", 942]
```

Code Line 2-33:

```
let shoppingList = ["Milk", "Eggs", "Cheese", "Bread"]
console.log("The second item on the shopping list is " + shoppingList[1])
```

Code Line 2-34:

```
let shoppingList = ["Milk", "Eggs", "Cheese", "Bread"]
console.log("The first item on the shopping list is " + shoppingList[0])
```

Code Line 2-35:

```
let shoppingList = ["Milk", "Eggs", "Cheese", "Bread"]
console.log("The third item on the shopping list is " + shoppingList[2])
```

Code Line 2-36:

```
let shoppingList = [“Milk”, “Eggs”, “Cheese”, “Bread”]
console.log(“Shopping List: “ + shoppingList.join(“, “))
```

Code Line 2-37:

```
Shopping List: Milk, Eggs, Cheese, Bread
```

Code Line 2-38:

```
let shoppingList = [“Milk”, “Eggs”, “Cheese”, “Bread”]
console.log(“Shopping List: “ + shoppingList.join(“ “))
```

Code Line 2-39:

```
Shopping List: Milk Eggs Cheese Bread
```

Code Line 2-40:

```
let shoppingList = [“Milk”, “Eggs”, “Cheese”, “Bread”]
console.log(“Shopping List: “ + shoppingList.join(“ “))
console.log(“Oops, we forgot the oranges!”)
shoppingList.push(“Oranges”)
console.log(“Our New Shopping List: “ + shoppingList.join(“ “))
```

Code Line 2-41:

```
let shoppingList = [“Milk”, “Eggs”, “Cheese”, “Bread”]
shoppingList[0] = “Almond Milk”
console.log(“Shopping List: “ + shoppingList.join(“ “))
```

Code Line 2-42:

```
Shopping List: Almond Milk Eggs Cheese Bread
```

Code Line 2-43:

```
let shoppingList = [“Milk”, “Eggs”, “Cheese”, “Bread”]
shoppingList[0] = “Almond Milk”
console.log(“Shopping List: “ + shoppingList.join(“, “))
```

Code Line 2-44:

```
Shopping List: Almond Milk, Eggs, Cheese, Bread
```

CHAPTER 3
Controlling Program Flow in JavaScript

Snippet 03-01.js:

```
let price = 4.95
if (price == 4.95) {
    console.log("The price is 4.95.")
} else {
    console.log("The price is something else.")
}
```

Code Line 3-1:

```
let price = 3.95
if (price == 4.95) {
    console.log("The price is 4.95.")
}
```

Table 1

OPERATOR	EXAMPLE	DESCRIPTION
==	(a == 1)	Check if both values are equal.
===	(a === 1)	Check if both values are equal and the same type.
!=	(a != 1)	Check if values are not equal.
!==	(a !== 1)	Check if values or types are not equal.
<	(a < 1)	Check if the first value is less than the second.
>	(a > 1)	Check if the first value is greater than the second.
<=	(a <= 1)	Check if the first value is less than or equal to the second.
>=	(a >= 1)	Check if the first value is greater than or equal to the second.

Snippet 03-02.js

```
let a = 1
let b = “1”

if (a === b) {
    console.log(“a and b are both equal and the same type”)
}

if (a == b) {
    console.log(“a and b are both 1”)
}
```

Code Line 3-2:

```
a and b are both 1
```

Code Line 3-3:

```
let a = 1
let b = “2”

if (a !== b) {
    console.log(“a and b are not equal or are of different types”)
}

if (a != b) {
    console.log(“a and b are not equal even if we ignore their types”)
}
```

Code Line 3-4:

```
a and b are not equal or are of different types
a and b are not equal even if we ignore their types
```

Snippet 03-03.js

```
let a = 5
let b = 10
let c = 100

if (b < c) {
    console.log(`${b} is less than ${c}`)
}
```

```
if (c > b) {
    console.log(`${c} is greater than ${b}`)
}

if (a <= 5) {
    console.log(`${a} is less than or equal to 5`)
}
```

Snippet 03-04.js

```
let a = 1
let b = 2

if (a == 1 && b == 2) {
    console.log("a is 1 and b is 2")
}

if (a == 1 && b == 2 && a != b) {
    console.log("a is 1, b is 2, and they are not equal")
}
```

Code Line 3-5:

```
let doorPosition = "shut"
let trunkPosition = "closed"
let seatBeltStatus = "on"

if (doorPosition == "shut"
    && trunkPosition == "closed"
    && seatBeltStatus == "on") {
        console.log("Let's roll!")
}
```

Snippet 03-05.js

```
let a = 1
let b = 2

if (a == 1 || b == 3) {
    console.log("At least one of these conditions is true.")
}

if (a == 1 || b == 2 || a == b) {
    console.log("At least one of the three conditions is true.")
}
```

Code Line 3-6:

```
let a = 1

if (a == 1 || a == 12 || a == 32) {
    console.log("This message will be displayed.")
}
```

Snippet 03-06.js

```
let done = false

if (done) {
    console.log("All done here.")
}
```

Code Line 3-7:

```
let done = false

// Do some things

done = true

if (done) {
    console.log("All done here.")
}
```

Code Line 3-8:

```
let done = false

// Do some things

if (! done) {
    console.log("Still not done.")
}

// Do some more things

done = true

if (done) {
    console.log("All done.")
}
```

Table 2

OPERATOR	NAME	EXAMPLE	DESCRIPTION
&&	**AND**	`( a == 1 && a != 2 )`	Check if all comparisons evaluate to true.
||	**OR**	`( a == 1 || a == 2 )`	Check if at least one comparison evaluates to true.
!	**NOT**	`( ! a )`	Check if false.

Snippet 03-07.js

```
let color = "blue"

switch (color) {
    case "red":
        console.log("The color is red.")
        break
    case "orange":
        console.log("The color is orange.")
        break
    case "purple":
        console.log("The color is purple.")
        break
    default:
        console.log("I'm not sure what color it is.")
}
```

Snippet switch.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <link rel="stylesheet" href="switch.css">
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id="content">
            <form>
                <select id="colorSelector" name="colorSelector">
                    <option value="red">Red</option>
                    <option value=”orange”>Orange</option>
                    <option value=”yellow”>Yellow</option>
                    <option value=”green”>Green</option>
                    <option value=”blue”>Blue</option>
                    <option value=”indigo”>Indigo</option>
                    <option value=”violet”>Violet</option>
                </select>
            </form>
            <div id=”colorSquare”></div>
        </div>
    </body>
    <script src=”switch.js”></script>
</html>
```

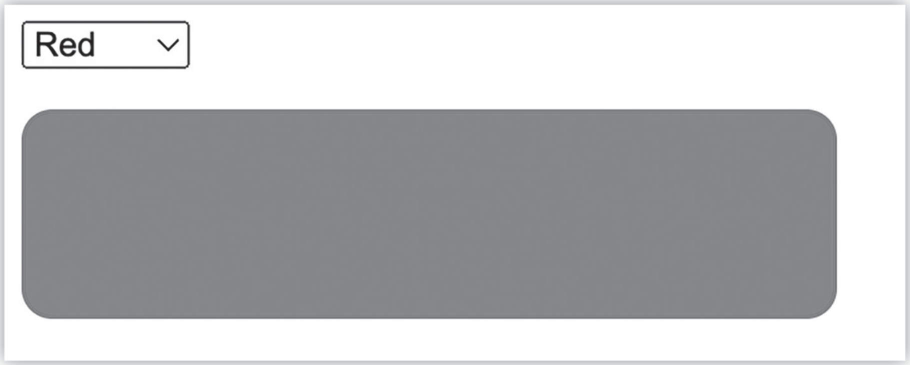
Snippet switch.css

```
#colorSquare {
    width: 20em;
    height: 5em;
    background-color: red;
    border: 0.1em solid gray;
    border-radius: 0.75em;
    margin-top: 1em;
}
```

Snippet switch.js

```
colorSelector = document.getElementById(“colorSelector”)
colorSquare = document.getElementById(“colorSquare”)

colorSelector.addEventListener(“change”, function() {
    let color = colorSelector.value
    colorSquare.style.backgroundColor = color
    console.log(color)
})
```

Figure 11



The result of our color selector code as shown in the browser.

Code Line 3-9:

```
colorSelector = document.getElementById("colorSelector")
colorSquare = document.getElementById("colorSquare")

colorSelector.addEventListener("change", function() {
    let color = colorSelector.value
    colorSquare.style.backgroundColor = color
    console.log(color)
})
```

Code Line 3-10:

```
colorSelector = document.getElementById("colorSelector")
colorSquare = document.getElementById("colorSquare")

colorSelector.addEventListener("change", function() {
    colorSquare.style.backgroundColor = colorSelector.value
})
```

Code Line 3-11:

```
<select id="colorSelector" name="colorSelector">
    <option value="#FF0000">Red</option>
    <option value="#FFA500">Orange</option>
    <option value="#FFFF00">Yellow</option>
    <option value="#008000">Green</option>
    <option value="#0000FF">Blue</option>
    <option value="#4B0082">Indigo</option>
    <option value="#EE82EE">Violet</option>
</select>
```

Code Line 3-12:

```
console.log(“Hello, World!”)
console.log(“Hello, World!”)
console.log(“Hello, World!”)
console.log(“Hello, World!”)
console.log(“Hello, World!”)
console.log(“Hello, World!”)
console.log(“Hello, World!”)
console.log(“Hello, World!”)
console.log(“Hello, World!”)
console.log(“Hello, World!”)
```

Code Line 3-13:

```
for (let i = 0; i <= 10; i++) {
    console.log(“Hello, World!”)
}
```

Code Line 3-14:

```
( 11 ) Hello, World!
```

Code Line 3-15:

```
for (let i = 0; i <= 10; i++) {
```

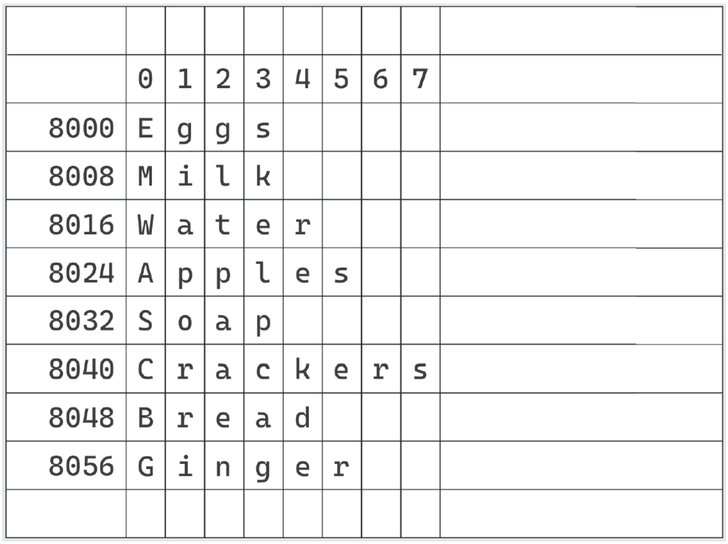
Code Line 3-16:

```
for (let i = 0; i <= 10; i++) {
    console.log(“Hello, World!”)
}
```

Code Line 3-17:

```
for (let i = 0; i < 10; i++) {
    console.log(“Hello, World!”)
}
```

Figure 12



	0	1	2	3	4	5	6	7	
8000	E	g	g	s					
8008	M	i	l	k					
8016	W	a	t	e	r				
8024	A	p	p	l	e	s			
8032	S	o	a	p					
8040	C	r	a	c	k	e	r	s	
8048	B	r	e	a	d				
8056	G	i	n	g	e	r			

Code Line 3-18:

```
for (let i = 100; i > 0; i--) {
    console.log(`i = ${i}`)
}
```

Code Line 3-19:

```
i = 100
i = 99
i = 98
…
i = 3
i = 2
i = 1
```

Snippet 03-08.js

```
for (let x = 0; x < 10; x++) {
    for (let y = 0; y < 10; y++) {
        console.log(`x, y = ${x}, ${y}`)
    }
}
```

Code Line 3-20:

```
x, y = 0, 0
x, y = 0, 1
x, y = 0, 2
…
x, y = 9, 7
x, y = 9, 8
x, y = 9, 9
```

Code Line 3-21:

```
let condition = true

while (condition) {
    // Run code until condition is false
}
```

Snippet 03-09.js

```
let v = ""

while (!v) {
    v = prompt("Please enter a value: ")
}

console.log(`You entered: ${v}`)
```

Code Line 3-22:

```
let v = ""

do {
    v = prompt("Please enter a value: ")
}
while (!v)

console.log(`You entered: ${v}`)
```

Snippet 03-10.js

```
let groceryList = [“Milk”, “Eggs”, “Cheese”, “Bread”]
for (let i in groceryList) {
    console.log(groceryList[i])
}
```

Code Line 3-23:

```
Milk
Eggs
Cheese
Bread
```

CHAPTER 4
Functions

Snippet 04-01.js

```
function greeting() {
    console.log("Hello, World!")
}
```

Code Line 4-1:

```
greeting()
```

Code Line 4-2:

```
Hello, World!
```

Code Line 4-3:

```
function greeting() {
    alert("Hello, World!")
}
```

Snippet 04-02.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id="content">
            <div id="greeting"></div>
            <button id="sayHello">Say Hello</button>
        </div>
    </body>
    <script src="scratch.js"></script>
</html>
```

Snippet 04-03.js

```
greetingContainer = document.getElementById(“greeting”)
sayHello = document.getElementById(“sayHello”)

function greeting() {
    greetingContainer.textContent = “Hello, World!”
}

sayHello.addEventListener(“click”, greeting())
```

Code Line 4-4:

```
sayHello.addEventListener(“click”, greeting())
```

Code Line 4-5:

```
sayHello.addEventListener(“click”, greeting())
```

Code Line 4-6:

```
sayHello.addEventListener(“click”, greeting)
```

Snippet 04-04.js

```
function greeting(firstName) {
    console.log(`Hello, ${firstName}!`)
}
```

Code Line 4-7:

```
greeting(“Robert”)
```

Code Line 4-8:

```
Hello, Robert!
```

Code Line 4-9:

```
greeting()
```

Code Line 4-10:

```
Hello, undefined!
```

Code Line 4-11:

```
greeting(“”)
```

Code Line 4-12:

```
Hello, !
```

Code Line 4-13:

```
function greeting(firstName = “World”) {
    console.log(`Hello, ${firstName}!`)
}
```

Code Line 4-14:

```
greeting()
```

Code Line 4-15:

```
Hello, World!
```

Code Line 4-16:

```
function greeting(firstName = “World”, lastName = “”) {
    console.log(`Hello, ${firstName} ${lastName}!`)
}
```

Code Line 4-17:

```
greeting(“Robert”, “Oliver”)
```

Code Line 4-18:

```
Hello, Robert Oliver!
```

Code Line 4-19:

```
greeting()
```

Code Line 4-20:

```
Hello, World !
```

Code Line 4-21:

```
function greeting(firstName = “World”, lastName = “”) {
    let message = “Hello, “

    if (lastName) {
        message = message + firstName + “ “ + lastName + “!”
    } else {
        message = message + firstName + “!”
    }

    console.log(message)
}
```

Code Line 4-22:

```
greeting()
```

Code Line 4-23:

```
Hello, World!
```

Code Line 4-24:

```
greeting(“Robert”)
```

Code Line 4-25:

```
Hello, Robert!
```

Code Line 4-26:

```
greeting(“Robert”, “Oliver”)
```

Code Line 4-27:

```
Hello, Robert Oliver!
```

Code Line 4-28:

```
colorSelector = document.getElementById(“colorSelector”)
```

Code Line 4-29:

```
calculateSalesTax(42.94)
```

Code Line 4-30:

```
let salesTax = calculateSalesTax(42.94)
```

Code Line 4-31:

```
function calculateSalesTax(subtotal, taxRate = 0.08) {
    return subtotal * taxRate
}

let subtotal = 42.94
let salesTax = calculateSalesTax(subtotal)
let total = subtotal + salesTax
```

Code Line 4-32:

```
console.log(total)
```

Code Line 4-33:

```
total.toFixed(2)
```

Code Line 4-34:

```
let realTotal = parseFloat(total.toFixed(2))
```

Code Line 4-35:

```
let realTotal = Math.round(total * 100) / 100
```

Code Line 4-36:

```
total.toLocaleString("en-US", { maximumFractionDigits: 2 })
```

Snippet 04-06.js

```
function getFormattedCurrency(amount, region = "en-US") {
    return amount.toLocaleString(region, { maximumFractionDigits: 2 })
}

function calculateSalesTax(subtotal, taxRate = 0.08) {
    return subtotal * taxRate
}

let subtotal = 42.94
let salesTax = calculateSalesTax(subtotal)
let total = parseFloat((subtotal + salesTax).toFixed(2))
let stringTotal = getFormattedCurrency(total)
```

Code Line 4-37:

```
> total
46.38
> stringTotal
'46.38'
```

Snippet 04-07.js

```
function calculateSalesTax(subtotal, taxRate = 0.08) {
    return subtotal * taxRate
}
```

Code Line 4-38:

```
calculateSalesTax = function(subtotal, taxRate = 0.08) {
    return subtotal * taxRate
}
```

Snippet shop.js

```
// Display a quick message in the console.
console.log("ClydeBank Coffee Shop is now open!")

// Our inventory
let inventory = ["Regular Coffee", "Espresso", "Cappuccino", "Latte"]
let inventoryPrices = [3.00, 3.50, 4.00, 4.25]

// Obtain reference to the menu list by ID
let menuList = document.getElementById("coffee-menu")

// Loop through the inventory and display each item in the menu list
function populateMenu(container) {
    for (let i = 0; i < inventory.length; i++) {
        container.innerHTML += "<li>" + inventory[i] + " - $" + inventoryPrices[i].to-
Fixed(2) + "</li>"
    }
}

populateMenu(menuList)
```

CHAPTER 5
Understanding Closure

Snippet 05-01.js

```
let productPrice = 8.95
let productQuantity = 3

function calculateLineCost() {
    return productPrice * productQuantity
}
```

Code Line 5-1:

```
calculateLineCost()
```

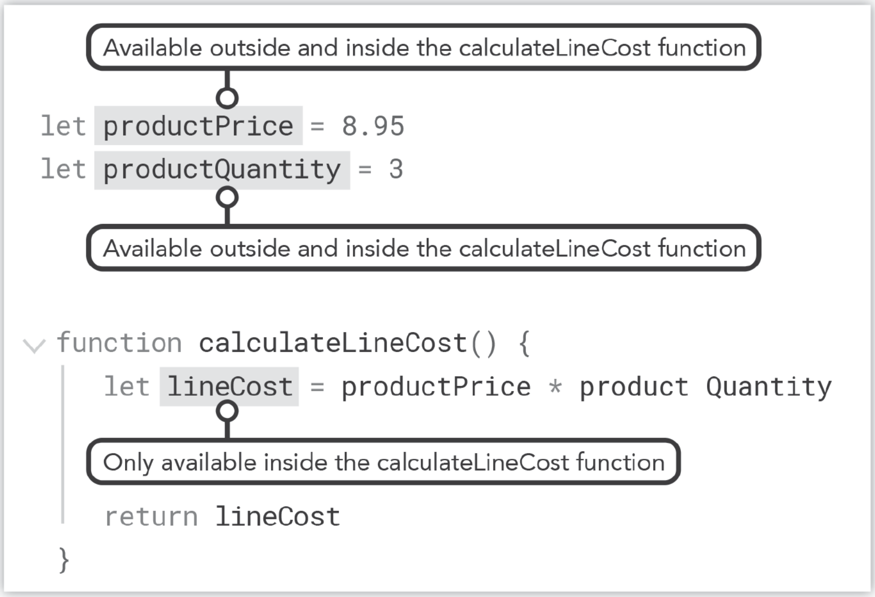
Code Line 5-2:

```
let productPrice = 8.95
let productQuantity = 3

function calculateLineCost() {
    let lineCost = productPrice * productQuantity
    return lineCost
}
```

Code Line 5-3:

```
console.log(lineCost)

Uncaught ReferenceError: lineCost is not defined
```

Figure 13


Available outside and inside the calculateLineCost function
Available outside and inside the calculateLineCost function
Only available inside the calculateLineCost function

```
let productPrice = 8.95
let productQuantity = 3

function calculateLineCost() {
    let lineCost = productPrice * product Quantity
    return lineCost
}
```

A detailed look at the scope of the `lineCost` variable.

Code Line 5-4:

```
let productPrice = 8.95
let productQuantity = 3
```

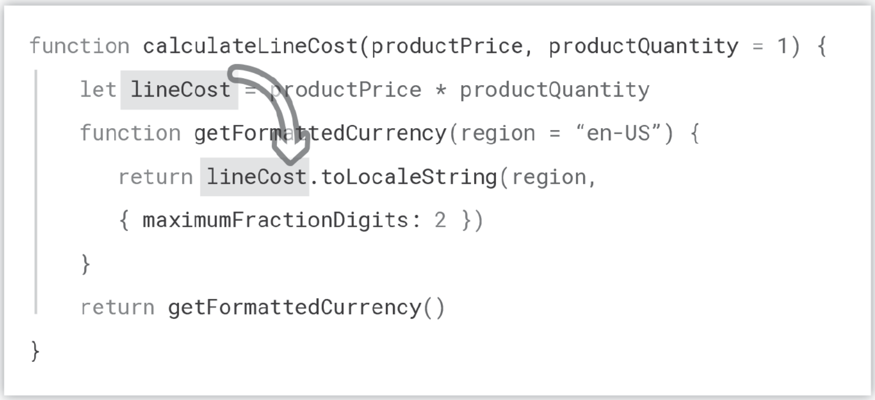
Code Line 5-5:

```
function calculateLineCost() {
    let lineCost = productPrice * productQuantity
    return lineCost
}
```

Snippet 05-02.js

```
function calculateLineCost(productPrice, productQuantity = 1) {
    let lineCost = productPrice * productQuantity
    function getFormattedCurrency(region = "en-US") {
        return lineCost.toLocaleString(region, { maximumFractionDigits: 2 })
    }
    return getFormattedCurrency()
}
```

Figure 14



```
function calculateLineCost(productPrice, productQuantity = 1) {
    let lineCost = productPrice * productQuantity
    function getFormattedCurrency(region = “en-US”) {
        return lineCost.toLocaleString(region,
        { maximumFractionDigits: 2 })
    }
    return getFormattedCurrency()
}
```

The `lineCost` variable is accessible inside `getFormattedCurrency` because the `getFormattedCurrency` function is defined within the `calculateLineCost` function, which is where the `lineCost` variable is defined.

Code Line 5-6:

```
calculateLineCost(4.95, 5)
```

Code Line 5-7:

```
‘24.75’
```

Snippet 05-03.js

```
function cartItem(price, quantity) {
    let lineCost = 0.00
    let formattedLineCost = “”

    function calculateLineCost() {
        lineCost = price * quantity
    }

    function applyQuantityDiscount() {
        if (quantity > 1) {
            lineCost -= quantity * (price * 0.01)
        }
    }

    function formatCurrency(region = “en-US”) {
        formattedLineCost =
            lineCost.toLocaleString(region, { maximumFractionDigits: 2 })
    }
    calculateLineCost()
    applyQuantityDiscount()
    formatCurrency()

    return formattedLineCost
}
```

Code Line 5-8:

```
item = cartItem(4.95, 10)
```

Code Line 5-9:

```
const cartItem = (function(price, quantity = 1) {
    let totalQuantity = 0
    let lineCost = 0.00

    function calculateLineCost() {
        lineCost = price * totalQuantity
    }

    function getFormattedCurrency(region = "en-US") {
        return lineCost.toLocaleString(region, { maximumFractionDigits: 2 })
    }

    return function () {
        totalQuantity += quantity
        calculateLineCost()
        return getFormattedCurrency()
    }
})
```

Code Line 5-10:

```
let item = cartItem(4.95)
```

Code Line 5-11:

```
item()
```

Code Line 5-12:

```
let item1 = cartItem(4.95)
let item2 = cartItem(16.24)
let item3 = cartItem(8.92)
```

CHAPTER 6
Organizing Data and Logic with Objects

Figure 15

Classes are like cookie cutters, and the cookies are like objects. But cookies don't necessarily have to be made by cookie cutters (classes).

Code Line 6-1:

```
class Greeting {
    constructor(firstName) {
        console.log(`Hello, ${firstName}!`)
    }
}
```

Code Line 6-2:

```
let a = new Greeting("Robert")
```

Code Line 6-3:

```
Hello, Robert!
```

Code Line 6-4:

```
class Greeting {
    constructor(firstName) {
        console.log(`Hello, ${firstName}!`)
    }
}
```

Code Line 6-5:

```
class Greeting {
    constructor(firstName) {
        this.firstName = firstName
    }
    sayHello() {
        console.log(`Hello, ${this.firstName}!`)
    }
}
```

Code Line 6-6:

```
let a = new Greeting("Robert")
a.sayHello()
```

Code Line 6-7:

```
Hello, Robert!
```

Code Line 6-8:

```
class Greeting {
    constructor(firstName) {
        this.firstName = firstName
    }
    sayHello(greeting = "Hello, ") {
        console.log('${greeting}${this.firstName}!')
    }
}
```

Code Line 6-9:

```
let a = new Greeting("Robert")
a.sayHello()
```

Code Line 6-10:

```
Hello, Robert!
```

Code Line 6-11:

```
let a = new Greeting("Robert")
a.sayHello("Greetings, ")
```

Code Line 6-12:

```
Greetings, Robert!
```

Code Line 6-13:

```
class Greeting {
    constructor(firstName) {
        this.firstName = firstName
    }
    sayHello(greeting = "Hello, ") {
        console.log(`${greeting}${this.firstName}!`)
    }
}
```

Code Line 6-14:

```
let a = new Greeting("Robert")
```

Code Line 6-15:

```
console.log(a.firstName)
```

Code Line 6-16:

```
"Robert"
```

Code Line 6-17:

```
console.log(a["firstName"])
```

Snippet 06-01.js

```
class Address {
    constructor(firstName, lastName) {
        this.firstName = firstName
        this.lastName = lastName
    }
}

let customer = new Address("Robert", "Oliver")

for (let detail in customer) {
    console.log(detail)
}
```

Code Line 6-18:

```
firstName
lastName
```

Code Line 6-19:

```
for (let detail in customer) {
    console.log(customer[detail])
}
```

Code Line 6-20:

```
Robert
Oliver
```

Code Line 6-21:

```
for (let detail in customer) {
    console.log(detail)
}
```

Code Line 6-22:

```
for (let detail in customer) {
    console.log(customer[detail])
}
```

Code Line 6-23:

```
class Address {
    constructor(firstName, lastName) {
        this.firstName = firstName
        this.lastName = lastName
    }
}
```

Code Line 6-24:

```
const robert = {
    firstName: “Robert”,
    lastName: “Oliver”,
    zipCode: 12345,
    zodiacSign: “Aquarius”
}
```

Code Line 6-25:

```
class Address {
    constructor(firstName, lastName, zipCode, zodiacSign) {
        this.firstName = firstName
        this.lastName = lastName
        this.zipCode = zipCode
        this.zodiacSign = zodiacSign
    }
}
```

Code Line 6-26:

```
const robert = {
    firstName: “Robert”,
    lastName: “Oliver”,
    zipCode: 12345,
    zodiacSign: “Aquarius”,
    greeting: function() {
        console.log(`Hello, ${this.firstName}!`)
    }
}
```

Code Line 6-27:

```
robert.greeting()
```

Snippet 06-02.js

```
const robert = {
    firstName: “Robert”,
    lastName: “Oliver”,
    zipCode: 12345,
    zodiacSign: “Aquarius”,
    greeting: function() {
        console.log(`Hello, ${this.firstName}!`)
    }
}
```

Code Line 6-28:

```
robert.phoneNumber = “555-555-1212”
```

Code Line 6-29:

```
const robert = {
    firstName: “Robert”,
    lastName: “Oliver”,
    zipCode: 12345,
    zodiacSign: “Aquarius”,
    phoneNumber: “(555) 555-1212”,
    greeting: function() {
        console.log(`Hello, ${this.firstName}!`)
    }
}
```

Code Line 6-30:

```
robert.sayPhoneNumber = function() {
    console.log(this.phoneNumber)
}
```

Code Line 6-31:

```
const robert = {
    firstName: “Robert”,
    lastName: “Oliver”,
    zipCode: 12345,
    zodiacSign: “Aquarius”,
    phoneNumber: “(555) 555-1212”,
    greeting: function() {
        console.log(`Hello, ${this.firstName}!`)
    }
    sayPhoneNumber: function() {
        console.log(this.phoneNumber)
    }
}
```

Code Line 6-32:

```
robert.greeting = function() {
    console.log(`Hello there, ${this.firstName}!`)
}
```

Snippet 06-03.js

```
evenNumbers = {}

evenNumbers[Symbol.iterator] = function() {
    let n = 0
    return {
        next() {
            n += 2
            return {value: n, done: false}
        }
    }
}

for (let n of evenNumbers) {
    console.log(n)
    if (n >= 10) break
}
```

Code Line 6-33:

```
2
4
6
8
10
```

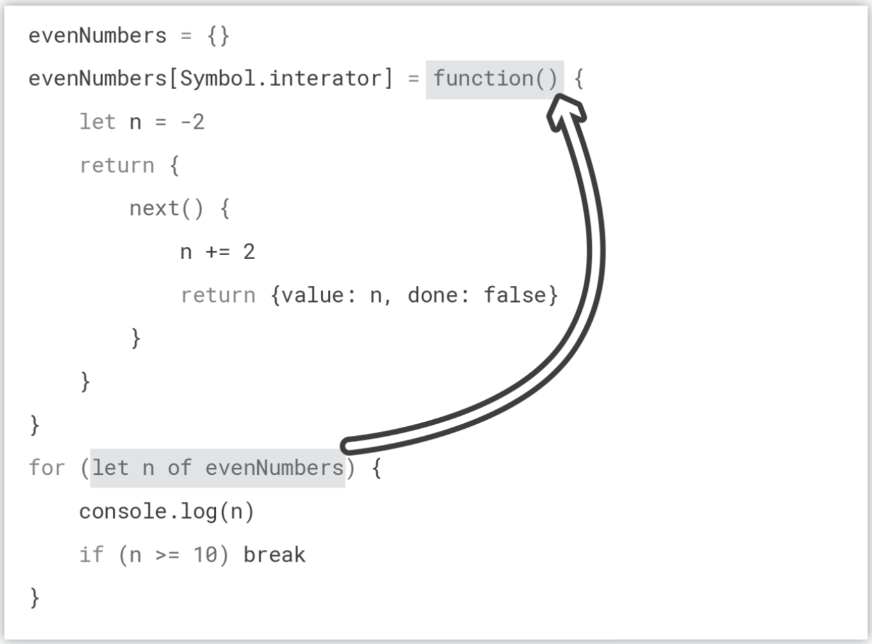
Code Line 6-34:

```
evenNumbers = {}

evenNumbers[Symbol.iterator] = function() {
    let n = -2
    return {
        next() {
            n += 2
            return {value: n, done: false}
        }
    }
}

for (let n of evenNumbers) {
    console.log(n)
    if (n >= 10) break
}
```

Code Line 6-35:

```
evenNumbers is not defined
```

Figure 16



```
evenNumbers = {}
evenNumbers[Symbol.interator] = function() {
    let n = -2
    return {
        next() {
            n += 2
            return {value: n, done: false}
        }
    }
}
for (let n of evenNumbers) {
    console.log(n)
    if (n >= 10) break
}
```

Each time we iterate through `evenNumbers`, the value of `n` is derived by executing the function in the iterator.

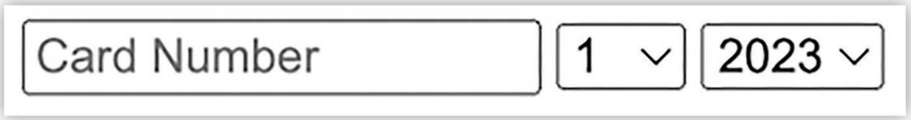
Snippet iterator.html

```
<!DOCTYPE html>
<html lang=”en”>
    <head>
        <meta charset=”UTF-8”>
        <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
        <meta http-equiv=”X-UA-Compatible” content=”ie=edge”>
        <title>Iterator</title>
    </head>
    <body>
        <div id=”content”>
            <form>
                <input type=”text” name=”cardNumber” placeholder=”Card Number”>
                <select id=”expMonth” name=”expMonth”>
                </select>
                <select id=”expYear” name=”expYear”>
                </select>
            </form>
        </div>
    <body>
    <script src=”iterator.js”></script>
</html>
```

Snippet iterator.js

```
// Define iterable objects
expMonths = {}
expYears = {}

// Iterator object for expiration months
expMonths[Symbol.iterator] = function() {
    let m = 0
    return {
        next() {
            m += 1
            theValue = “<option value=’” + m + “’>” + m + “</option>”
            if (m > 12) {
                return {value: theValue, done: true}
            } else {
                return {value: theValue, done: false}
            }
        }
    }
}

// Iterator object for expiration years
expYears[Symbol.iterator] = function() {
    let y = new Date().getFullYear() - 1
    return {
        next() {
            y += 1
            theValue = {html: "<option value='" + y + "'>" + y + "</option>", year: y}
            return {value:theValue, done:false}
        }
    }
}

// Construct the expiration month dropdown options
for (let month of expMonths) {
    document.getElementById("expMonth").innerHTML += month
}

// Construct the expiration year dropdown options
maxYear = new Date().getFullYear() + 10
for (let year of expYears) {
    document.getElementById("expYear").innerHTML += year.html
    if (year.year >= maxYear) break
}
```

Figure 17



Our pre-populated credit card expiration month and year fields.

Code Line 6-36:

```
maxYear = new Date().getFullYear() + 10
```

Code Line 6-37:

```
document.getElementById("expYear").innerHTML += year
```

Code Line 6-38:

```
document.getElementById("expYear").innerHTML += year.html
```

Snippet shop-06.js

```
// Display a quick message in the console.
console.log(“ClydeBank Coffee Shop is now open!”)

// The inventory object
let menu = {
    inventory: {
        “Regular Coffee”: 3.00,
        “Espresso”: 3.50,
        “Cappuccino”: 4.00,
        “Latte”: 4.25
    },
    populate: function(container) {
        for (let item in this.inventory) {
            let price = this.inventory[item]
            container.innerHTML += “<li>” + item + “ - $” + price.toFixed(2) + “</li>”
        }
    }
}

// Obtain reference to the menu list by ID
let menuList = document.getElementById(“coffee-menu”)

// Populate the menu
menu.populate(menuList)
```

CHAPTER 7
Using JSON

Snippet 07-01.json

```
{
    “coffeeSales”: [
        {
            “blendName”: “Arabica”,
            “cupsSold”: “120”
        },
        {
            “blendName”: “Robusta”,
            “cupsSold”: “242”
        }
    ]
}
```

Snippet 07-02.js

```
const salesReport = {
    coffeeSales: [
        {
            blendName: “Arabica”,
            cupsSold: “120”
        },
        {
            blendName: “Robusta”,
            cupsSold: “242”
        }
    ]
}
```

Code Line 7-1:

```
let jsonString =
‘{“coffeeSales”:[{“blendName”:”Arabica”,”cupsSold”:”120”},
{“blendName”:”Robusta”,”cupsSold”:”242”}]}’
```

Code Line 7-2:

```
let salesReport = JSON.parse(jsonString)
```

Code Line 7-3:

```
let newJsonString = JSON.stringify(salesReport)
```

Code Line 7-4:

```
console.log(jsonString == newJsonString)
```

Code Line 7-5:

```
{“Greeting”: “Hello, World!”}
```

Code Line 7-6:

```
{
    “Greeting”: “Hello, World!”
}
```

Code Line 7-7:

```
const hello = {
    Greeting: “Hello, World!”
}
```

Code Line 7-8:

```
console.log(hello.Greeting)
```

Code Line 7-9:

```
Hello, World!
```

Code Line 7-10:

```
{
    “coffeeSales”: [
        {
            “blendName”: “Arabica”,
            “cupsSold”: “120”
        },
        {
            "blendName": "Robusta",
            "cupsSold": "242"
        }
    ]
}
```

Snippet 07-03.json

```
const salesReport = {
    coffeeSales: [
        {
            blendName: "Arabica",
            cupsSold: "120"
        },
        {
            blendName: "Robusta",
            cupsSold: "242"
        }
    ]
}
```

Snippet 07-04.js:

```
dayNumber = 1
for (let day in salesReport.coffeeSales) {
        console.log("Day " + dayNumber + ":")
        console.log("Blend: " + salesReport.coffeeSales[day].blendName +
            ", Cups Sold: " + salesReport.coffeeSales[day].cupsSold)
        dayNumber++
}
```

Code Line 7-11:

```
{
    "loggedIn": true
}
```

Code Line 7-12:

```
const userStatus = {
    loggedIn: true
}
```

Code Line 7-13:

```
let menu = {
    inventory: {
        "Regular Coffee": 3.00,
        "Espresso": 3.50,
        "Cappuccino": 4.00,
        "Latte": 4.25
    },
    populate: function(container) {
        for (let item in this.inventory) {
            let price = this.inventory[item]
            container.innerHTML += "<li>" + item + " - $" + price.toFixed(2) + "</li>"
        }
    }
}
```

Snippet 07-05.js

```
{
    "inventory": {
            "Regular Coffee": 3.00,
            "Espresso": 3.50,
            "Cappuccino": 4.00,
            "Latte": 4.25
    }
}
```

Code Line 7-14:

```
let inflatedPrices = {
    "inventory": {
        "Regular Coffee": 4.00,
        "Espresso": 4.50,
        "Cappuccino": 4.75,
        "Latte": 5.00
    }
}
```

Code Line 7-15:

```
menu.inventory = inflatedPrices.inventory
```

Code Line 7-16:

```
console.log(menu.inventory)
```

Code Line 7-17:

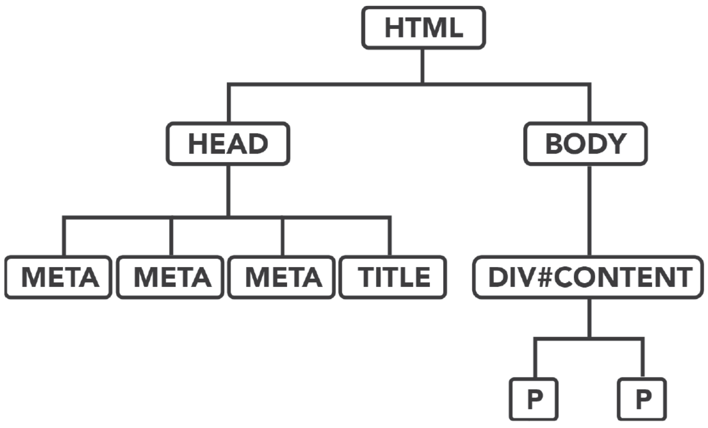
```
menuList.innerHTML = “”
menu.populate(menuList)
```

CHAPTER 8
Using the Document Object Model (DOM)

Code Line 8-1:

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <meta http-equiv="X-UA-Compatible" content="ie=edge">
      <title>Scratch Pad</title>
   </head>
   <body>
      <div id="content">
         <p>Hello, World!</p>
         <p>And hello again!</p>
      </div>
   </body>
</html>
```

Figure 18



Here is a visual representation of the HTML code.

Snippet 08-01.html

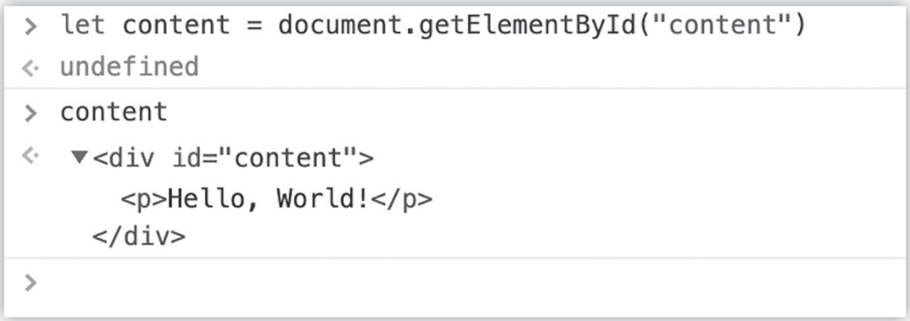
```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="UTF-8">
```

```
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id="content">
            <p>Hello, World!</p>
        </div>
    </body>
</html>
```

Code Line 8-2:

```
let content = document.getElementById("content")
```

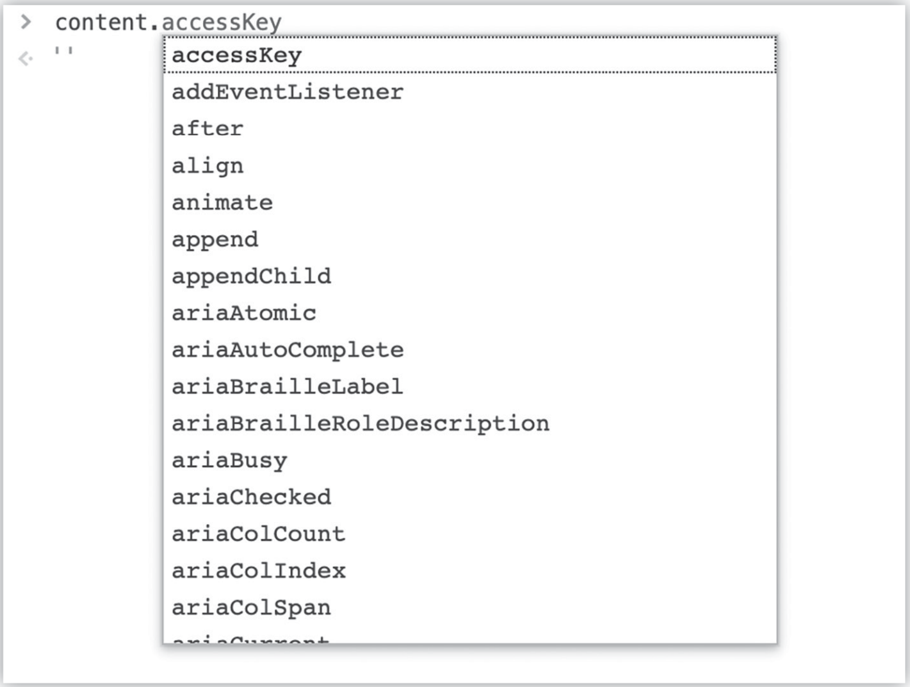
Figure 19



```
> let content = document.getElementById("content")
<· undefined
> content
<·  ▼<div id="content">
       <p>Hello, World!</p>
     </div>
>
```

The `content` object expanded to show the HTML within the object.

Figure 20



```
> content.accessKey
<· ''
```

Google Chrome showing a list of methods and properties of the `content` object.

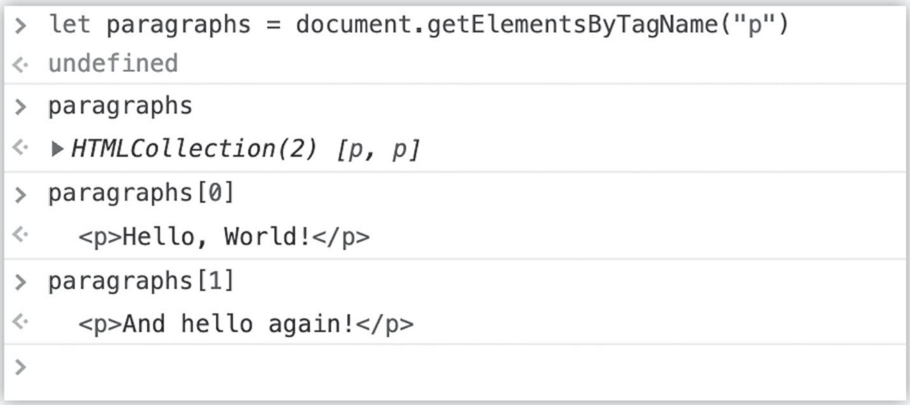
Snippet 08-02.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id="content">
            <p>Hello, World!</p>
            <p>And hello again!</p>
        </div>
    </body>
</html>
```

Code Line 8-3:

```
let paragraphs = document.getElementsByTagName("p")
```

Figure 21



```
> let paragraphs = document.getElementsByTagName("p")
<· undefined
> paragraphs
<· ▶HTMLCollection(2) [p, p]
> paragraphs[0]
<·   <p>Hello, World!</p>
> paragraphs[1]
<·   <p>And hello again!</p>
>
```

Illustrating the `document.getElementsByTagName` method and its results.

Code Line 8-4:

```
let firstParagraph = paragraphs[0]
let secondParagraph = paragraphs[1]
```

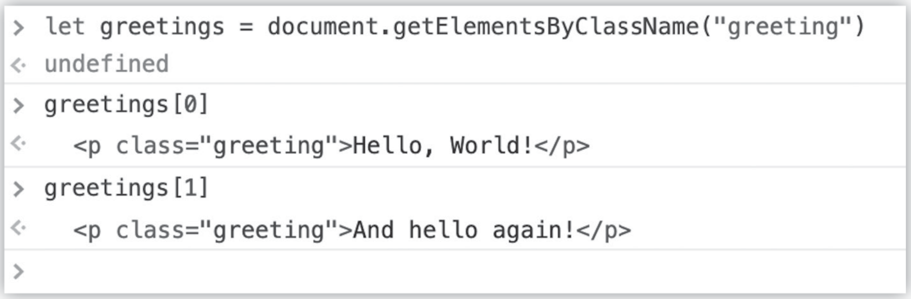
Snippet 08-03.html

```
<!DOCTYPE html>
<html lang=”en”>
    <head>
        <meta charset=”UTF-8”>
        <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
        <meta http-equiv=”X-UA-Compatible” content=”ie=edge”>
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id=”content”>
            <p class=”greeting”>Hello, World!</p>
            <p class=”greeting”>And hello again!</p>
        </div>
    </body>
</html>
```

Code Line 8-5:

```
let greetings = document.getElementsByClassName(“greeting”)
```

Figure 22



```
> let greetings = document.getElementsByClassName("greeting")
<· undefined
> greetings[0]
<·   <p class="greeting">Hello, World!</p>
> greetings[1]
<·   <p class="greeting">And hello again!</p>
>
```

Illustrating the `document.getElementsByClassName` method and its results.

Code Line 8-6:

```
let content = document.querySelector(“#content”)
```

Code Line 8-7:

```
let firstGreeting = document.querySelector(“.greeting”)
```

Code Line 8-8:

```
let allGreetings = document.querySelectorAll(“.greeting”)
```

Code Line 8-9:

```
for (let greeting of allGreetings) {
    console.log(greeting)
}
```

Code Line 8-10:

```
let allParagraphs = document.querySelectorAll(“p”)
```

Code Line 8-11:

```
<!DOCTYPE html>
<html lang=”en”>
    <head>
        <meta charset=”UTF-8”>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id="content">
            <p class="greeting">Hello, World!</p>
            <p class="greeting">And hello again!</p>
        </div>
    </body>
</html>
```

Code Line 8-12:

```
let content = document.getElementById("content")
content.style.fontWeight = "bold"
```

Code Line 8-13:

```
let paragraphs = document.getElementsByTagName("p")
```

Code Line 8-14:

```
for (let p of paragraphs) {
    p.style.fontWeight = "bold"
}
```

Snippet 08-04.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0"
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id="content">
            <p class="greeting">Hello, World!</p>
            <p class="greeting">And hello again!</p>
        </div>
        <div id="moreContent">
            <p class="greeting">This should not become bold.</p>
        </div>
    </body>
</html>
```

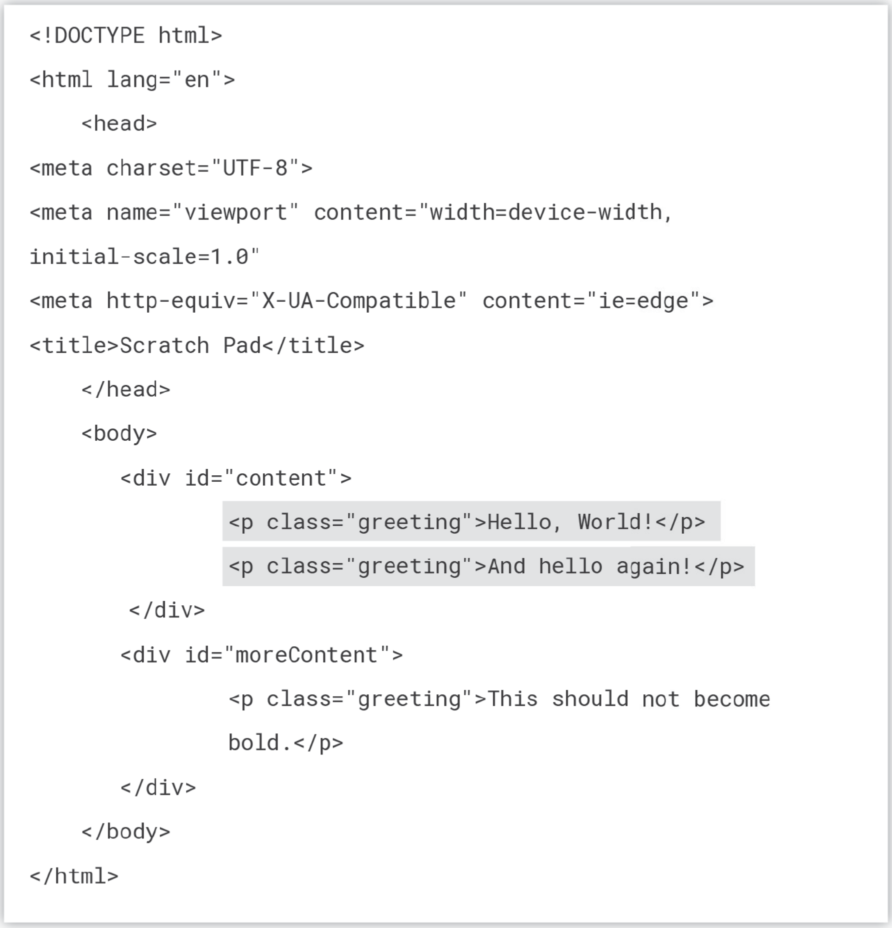
Code Line 8-15:

```
let content = document.getElementById("content")
```

Code Line 8-16:

```
let paragraphs = content.getElementsByTagName("p")
```

Figure 23



```
<!DOCTYPE html>
<html lang="en">
    <head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width,
initial-scale=1.0"
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Scratch Pad</title>
    </head>
    <body>
       <div id="content">
               <p class="greeting">Hello, World!</p>
               <p class="greeting">And hello again!</p>
        </div>
       <div id="moreContent">
               <p class="greeting">This should not become
               bold.</p>
       </div>
    </body>
</html>
```

Code Line 8-17:

```
for (let p of paragraphs) {
    p.style.fontWeight = "bold"
}
```

Code Line 8-18:

```
content.style.
```

Snippet 08-05.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0"
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id="content">
            <p class="greeting">Hello, World!</p>
            <p class="greeting">And hello again!</p>
        </div>
        <div id="moreContent">
            <p class="differentText">This should not become bold.</p>
        </div>
    </body>
</html>
```

Code Line 8-19:

```
let differentTexts = document.getElementsByClassName("differentText")
```

Code Line 8-20:

```
differentTexts[0].classList.add("greeting")
differentTexts[0].classList.remove("differentText")
```

Code Line 8-21:

```
for (let d of differentTexts) {
    d.classList.add(“greeting”)
    d.classList.remove(“differentText”)
}
```

Code Line 8-22:

```
<!DOCTYPE html>
<html lang=”en”>
    <head>
        <meta charset=”UTF-8”>
        <meta name=”viewport” content=”width=device-width, initial-scale=1.0”
        <meta http-equiv=”X-UA-Compatible” content=”ie=edge”>
        <title>Scratch Pad</title>
    </head>
    <body>
        <div id=”content”>
            <p class=”greeting”>Hello, World!</p>
            <p class=”greeting”>And hello again!</p>
        </div>
        <div id=”moreContent”>
            <p class=”differentText”>This should not become bold.</p>
        </div>
    </body>
</html>
```

Code Line 8-23:

```
let moreContent = document.getElementById(“moreContent”)
```

Code Line 8-24:

```
moreContent.remove()
```

Code Line 8-25:

```
let content = document.getElementById(“content”)
content.innerHTML = ‘<p class=”greeting”>Hello, World!</p>’
```

Code Line 8-26:

```
document.getElementById(“content”).innerHTML = ‘<p class=”greeting”>Hello, World!</p>’
```

Code Line 8-27:

```
document.getElementById(“content”).innerHTML = ‘<p class=”greeting”>
Hello, p.1!</p>’
document.getElementById(“content”).innerHTML = ‘<p class=”greeting”>
Hello, 2!</p>’
document.getElementById(“content”).innerHTML = ‘<p class=”greeting”>
Hello, 3!</p>’
document.getElementById(“content”).innerHTML = ‘<p class=”greeting”>
Hello, 4!</p>’
document.getElementById(“content”).innerHTML = ‘<p class=”greeting”>
Hello, 5!</p>’
```

Code Line 8-28:

```
let content = document.getElementById(“content”)

content.innerHTML = ‘<p class=”greeting”>Hello, 1!</p>’
content.innerHTML = ‘<p class=”greeting”>Hello, 2!</p>’
content.innerHTML = ‘<p class=”greeting”>Hello, 3!</p>’
content.innerHTML = ‘<p class=”greeting”>Hello, 4!</p>’
content.innerHTML = ‘<p class=”greeting”>Hello, 5!</p>’
```

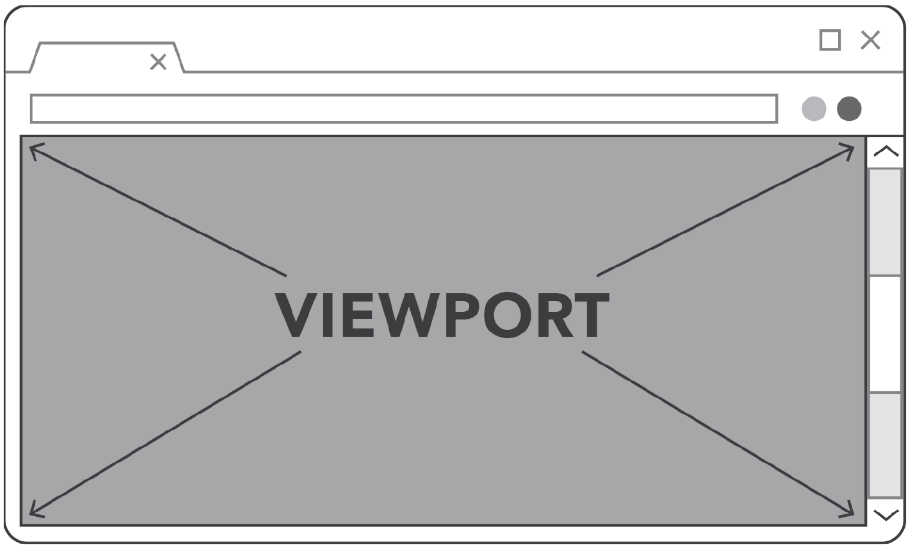
Code Line 8-29:

```
let content = document.getElementById(“content”)
```

Code Line 8-30:

```
let content = window.document.getElementById(“content”)
```

Figure 24



The viewport of the browser window.

Snippet resize.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Resize</title>
    </head>
    <body>
        <div id="content">
            <p>
                Window width: <span id="width"></span><br>
                Window height: <span id="height"></span>
            </p>
        </div>
    </body>
    <script src="resize.js"></script>
</html>
```

Snippet resize.js

```
let widthValue = document.getElementById("width")
let heightValue = document.getElementById("height")

function displayWindowSize() {
    widthValue.innerHTML = window.innerWidth
    heightValue.innerHTML = window.innerHeight
}

displayWindowSize()
window.addEventListener("resize", displayWindowSize)
```

Snippet shop.js

```
// Display a quick message in the console.
console.log("ClydeBank Coffee Shop is now open!")

// The inventory object
let menu = {
    inventory: {
        "Regular Coffee": 3.00,
        "Espresso": 3.50,
        "Cappuccino": 4.00,
        "Latte": 4.25
    },
    populate: function(container) {
        for (let item in this.inventory) {
            let price = this.inventory[item].toFixed(2)

            // Create a new li element
            let li = document.createElement("li")

            // Add text content to the li element
            li.textContent = `${item} - $${price}`

            // Append the li element to the container
            container.appendChild(li)
        }
    }
}

// Obtain reference to the menu list by ID
let menuList = document.getElementById("coffee-menu")

// Populate the menu
menu.populate(menuList)
```

CHAPTER 9
Asynchronous JavaScript

Code Line 9-1:

```
<!-- This code will be executed in the order the files are listed -->
<script src=”scratch1.js”></script>
<script src=”scratch2.js”></script>

<!-- This code will be executed right away in the background -->
<script src=”scratch.js” async></script>

<!-- This code will be executed after the page has fully loaded -->
<script src=”scratch.js” defer></script>
```

Snippet 09-01.js

```
function reliesOnPageLoaded() {
    // This code needs to be executed after the page is loaded
}

document.addEventListener(“DOMContentLoaded”, reliesOnPageLoaded)
```

Table 3

EVENT NAME	EVENT DESCRIPTION
`DOMContentLoaded`	This event is fired when the page is fully loaded.
`click`	This event is fired when a button is clicked.
`submit`	This event is fired when a form is submitted.
`change`	This event is fired when an input element is changed.
`keydown`	This event is fired when the user presses a key.
`mouseover`	This event is fired when the user moves their mouse over a particular element.
`mouseout`	This event is fired when the user moves their mouse away from an element where they had previously moused over.
`ontouchstart`	This event is fired on a device with a touch screen when the user begins touching an element.
`ontouchend`	This event is fired on a device with a touch screen when the user lifts their finger from an element.

Code Line 9-2:

```
document.addEventListener(“DOMContentLoaded”, reliesOnPageLoaded)
```

Code Line 9-3:

```
submitButton = document.getElementById(“submit”)
```

Code Line 9-4:

```
submitButton.addEventListener(“click”, runWhenClicked)
```

Code Line 9-5:

```
submitButton.addEventListener(“click”, function() {
    alert(“You clicked me!”)
})
```

Code Line 9-6:

```
<button onclick=”alert(‘You clicked me’)”>Click me!</button>
```

Code Line 9-7:

```
<button onclick=”showAlertMessage()”>Click me!</button>
```

Code Line 9-8:

```
submitButton.addEventListener(“click”, function() {
    alert(“You clicked me!”)
})
```

Code Line 9-9:

```
submitButton.addEventListener(“click”, function() {
    this.textContent = “You clicked me!”
})
```

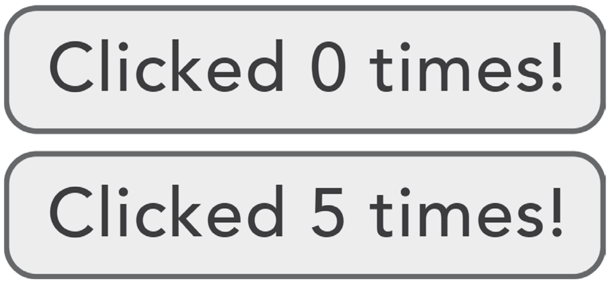
Snippet clicked.html

```
<!DOCTYPE html>
<html lang=”en”>
    <head>
        <meta charset=”UTF-8”>
        <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
        <meta http-equiv=”X-UA-Compatible” content=”ie=edge”>
        <title>Click This Button</title>
    </head>
    <body>
        <div id=”content”>
            <button id=”theButton”>Clicked 0 times!</button>
        </div>
    </body>
    <script src=”clicked.js”></script>
</html>
```

Snippet clicked.js

```
let theButton = document.getElementById(“theButton”);

theButton.addEventListener(“click”, (function() {
    // Initially set the clickCount to 0
    let clickCount = 0

    return function() {
    // Increment the clickCount
        clickCount += 1
        // Update the button’s textContent
        this.textContent = `Clicked ${clickCount} times!`
    }
})())
```

Figure 25



Button on initial page load and after clicking five times.

Code Line 9-10:

```
function timeMachine() {
    console.log("Hello from the past!")
}
```

Code Line 9-11:

```
setTimeout(timeMachine, 5000)
```

Snippet timeout.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>You Will Watch This Video</title>
    </head>
    <body>
        <div id="content">
            <div id="current">
                <a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ"
                    target="_blank">Watch the Video</a>
            </div>
            <br><br>
            <div id="next" style="display: none;">
                <a href=https://www.youtube.com/watch?v=XqZsoesa55w
                    target="_blank">Watch the Next Video</a>
            </div>
        </div>
    </body>
    <script src="timeout.js"></script>
</html>
```

Snippet timeout.js

```
let nextLink = document.getElementById("next")

function showNextLink() {
    nextLink.style.display = "block"
}

setTimeout(showNextLink, 5000)
```

Snippet 09-02.js

```
let tickCount = 0

function incrementTicks() {
    tickCount += 1
    console.log(`There have been ${tickCount} ticks so far.`)
}

const interval = setInterval(incrementTicks, 1000)
```

Code Line 9-12:

```
There have been 1 ticks so far.
There have been 2 ticks so far.
```

Code Line 9-13:

```
clearInterval(interval)
```

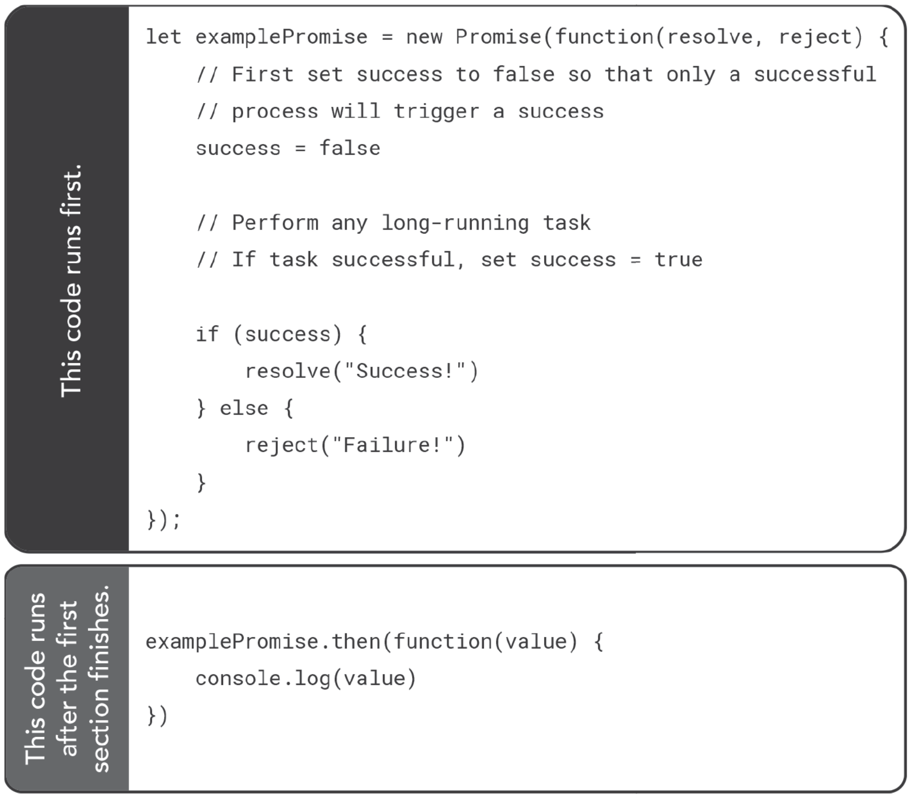
Code Line 9-14:

```
let tickCount = 0

function incrementTicks() {
    tickCount += 1
    console.log(`There have been ${tickCount} ticks so far.`)
    if (tickCount >= 5) {
        clearInterval(interval)
    }
}

const interval = setInterval(incrementTicks, 1000)
```

Snippet 09-03.js

```
let examplePromise = new Promise(function(resolve, reject) {
    // First set success to false so that only a successful
    // process will trigger a success
    success = false

    // Perform any long-running task
    // If task successful, set success = true
    if (success) {
        resolve("Success!")
    } else {
        reject("Failure!")
    }
});

examplePromise.then(function(value) {
    console.log(value)
})
```

Figure 26



```
let examplePromise = new Promise(function(resolve, reject) {
    // First set success to false so that only a successful
    // process will trigger a success
    success = false

    // Perform any long-running task
    // If task successful, set success = true

    if (success) {
        resolve("Success!")
    } else {
        reject("Failure!")
    }
});
```

```
examplePromise.then(function(value) {
    console.log(value)
})
```

CHAPTER 10
JavaScript Design Patterns

Code Line 10-1:

```
<select id="colorSelector" name="colorSelector">
    <option value="red">Red</option>
    <option value="orange">Orange</option>
    <option value="yellow">Yellow</option>
    <option value="green">Green</option>
    <option value="blue">Blue</option>
    <option value="indigo">Indigo</option>
    <option value="violet">Violet</option>
</select>
```

Snippet switch2.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <link rel="stylesheet" href="switch2.css">
        <title>Switch 2</title>
    </head>
    <body>
        <div id="content">
            <form>
                <div id="colorSelection">
                </div>
            </form>
        <div id="colorSwatch"></div>
        </div>
    </body>
    <script src="switch2.js"></script>
</html>
```

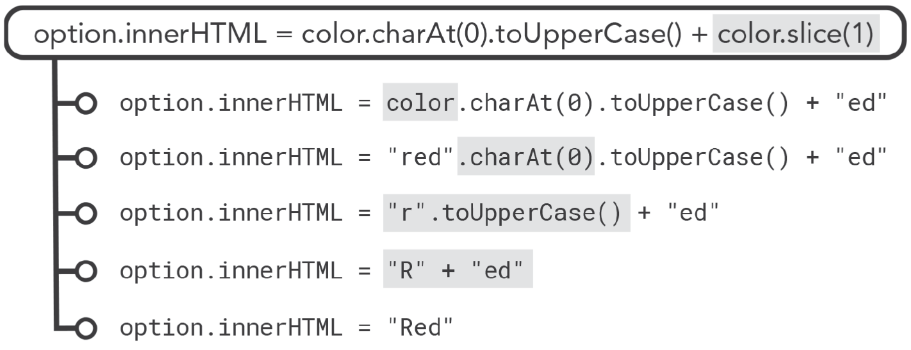
Snippet switch2.css

```
#colorSwatch {
    width: 20em;
    height: 5em;
    background-color: red;
    border: 0.1em solid gray;
    border-radius: 0.75em;
    margin-top: 1em;
}
```

Snippet switch2.js

```
const colorSelector = {
    colors: {
        "red": "#FF0000",
        "orange": "#FFA500",
        "yellow": "#FFFF00",
        "green": "#008000",
        "blue": "#0000FF",
        "indigo": "#4B0082",
        "violet": "#EE82EE"
    },
    render: (parent, id, swatch, className = "") => {
        let select = document.createElement("select")
        select.id = id
        if (className) {
            select.className = className
        }
        for (let color in colorSelector.colors) {
            let option = document.createElement("option")
            option.value = colorSelector.colors[color]
            option.innerHTML = color.charAt(0).toUpperCase() + color.slice(1)
            select.appendChild(option)
        }

        select.addEventListener("change", function(event) {
            swatch.style.backgroundColor = event.target.value
        })

        parent.appendChild(select)
    }
}

let colorSwatch = document.getElementById("colorSwatch")
let colorSelection = document.getElementById("colorSelection")
colorSelector.render(colorSelection, "colorSelect", colorSwatch)
```

Figure 27


option.innerHTML = color.charAt(0).toUpperCase() + color.slice(1)
option.innerHTML = color.charAt(0).toUpperCase() + "ed"
option.innerHTML = "red".charAt(0).toUpperCase() + "ed"
option.innerHTML = "r".toUpperCase() + "ed"
option.innerHTML = "R" + "ed"
option.innerHTML = "Red"

In much the same way as equations are simplified in algebra or fractions are reduced, dot notation performs an operation on the value and then returns that value inline. This is then chained to the next operation if there's more than one, until a single value can be assigned (in this example, to `option.innerHTML`).

Code Line 10-2:

```
<option value=”#FF0000”>Red</option>
```

Code Line 10-3:

```
let colorSwatch = document.getElementById(“colorSwatch”)
let colorSelection = document.getElementById(“colorSelection”)
colorSelector.render(colorSelection, “colorSelect”, colorSwatch)
```

Code Line 10-4:

```
option.innerHTML = color.charAt(0).toUpperCase() + color.slice(1)
```

Code Line 10-5:

```
function capFirstLetter(string) {
    return string.charAt(0).toUpperCase() + string.slice(1)
}
```

Code Line 10-6:

```
function capFirstLetter(string) {
    return string.charAt(0).toUpperCase() + string.slice(1)
}
```

```
const colorSelector = {
    colors: {
        “red”: “#FF0000”,
        “orange”: “#FFA500”,
        “yellow”: “#FFFF00”,
        “green”: “#008000”,
        “blue”: “#0000FF”,
        “indigo”: “#4B0082”,
        “violet”: “#EE82EE”
    },
    render: (parent, id, swatch, className = “”) => {
        let select = document.createElement(“select”)
        select.id = id
        if (className) {
            select.className = className
        }
        for (let color in colorSelector.colors) {
            let option = document.createElement(“option”)
            option.value = colorSelector.colors[color]
            option.innerHTML = capFirstLetter(color)
            select.appendChild(option)
        }

        select.addEventListener(“change”, function(event) {
            swatch.style.backgroundColor = event.target.value
        })

        parent.appendChild(select)
    }
}
```

Code Line 10-7:

```
function changeBgColor(element, color) {
    element.style.backgroundColor = color
}
```

Snippet 10-01.js

```
function capFirstLetter(string) {
    return string.charAt(0).toUpperCase() + string.slice(1)
}

function changeBgColor(element, color) {
    element.style.backgroundColor = color
}
```

```
const colorSelector = {
    colors: {
        "red": "#FF0000",
        "orange": "#FFA500",
        "yellow": "#FFFF00",
        "green": "#008000",
        "blue": "#0000FF",
        "indigo": "#4B0082",
        "violet": "#EE82EE"
    },
    render: (parent, id, swatch, className = "") => {
        let select = document.createElement("select")
        select.id = id
        if (className) {
            select.className = className
        }
        for (let color in colorSelector.colors) {
            let option = document.createElement("option")
            option.value = colorSelector.colors[color]
            option.innerHTML = capFirstLetter(color)
            select.appendChild(option)
        }

        select.addEventListener("change", function(event) {
            changeBgColor(swatch, event.target.value)
        })

        parent.appendChild(select)
    }
}
```

Code Line 10-8:

```
let db = new sqlite3.Database("mydata.db", (error) => {
    if (error) {
        // Handle the error
    }
})
```

Snippet 10-02.js

```
class MySingletonClass {
    constructor() {
        if (MySingletonClass._instance) {
            throw new Error("You can only create one instance of this class.")
        }
        MySingletonClass._instance = this
    }

    // Rest of class code would go here
}
```

Code Line 10-9:

```
let mySingletonObject = new MySingletonClass()
```

Code Line 10-10:

```
let mySecondSingletonObject = new MySingletonClass()
```

Code Line 10-11:

```
Uncaught Error: You can only create one instance of this class.
```

Code Line 10-12:

```
throw new Error("You can only create one instance of this class.")
```

Snippet switch2.js

```
// Utility functions

function capFirstLetter(string) {
    return string.charAt(0).toUpperCase() + string.slice(1)
}

function changeBgColor(element, color) {
    element.style.backgroundColor = color
}

// Data for selector

const colors = {
    "red": "#FF0000",
    "orange": "#FFA500",
    "yellow": "#FFFF00",
    "green": "#008000",
    "blue": "#0000FF",
    "indigo": "#4B0082",
    "violet": "#EE82EE"
}

// Define function to be executed on the change event

function changeColor(swatch, event) {
    changeBgColor(swatch, event.target.value)
}

const selectorPrototype = {
    render: (values, parent, id, onChange, swatch = "", className = "") => {
        let select = document.createElement("select")
        select.id = id
        if (className) {
            select.className = className
        }
        for (let value in values) {
            let option = document.createElement("option")
            option.value = values[value]
            option.innerHTML = capFirstLetter(value)
            select.appendChild(option)
        }

        select.addEventListener("change", (event) => {
            onChange(swatch, event)
        })

        parent.appendChild(select)
    }
}

let colorSwatch = document.getElementById("colorSwatch")
let colorSelection = document.getElementById("colorSelection")

let colorSelector = Object.create(selectorPrototype)

colorSelector.render(colors, colorSelection,
    "colorSelect", changeColor, colorSwatch)
```

Code Line 10-13:

```
const selectorPrototype = {
    render: (values, parent, id, onChange, swatch = "", className = "") => {
        let select = document.createElement("select")
        select.id = id
        if (className) {
            select.className = className
        }
        for (let value in values) {
            let option = document.createElement("option")
            option.value = values[value]
            option.innerHTML = capFirstLetter(value)
            select.appendChild(option)
        }

        select.addEventListener("change", (event) => {
            onChange(swatch, event)
        })

        parent.appendChild(select)
    }
}
```

Code Line 10-14:

```
let colorSelector = Object.create(selectorPrototype)

colorSelector.render(colors, colorSelection,
    "colorSelect", changeColor, colorSwatch
```

Code Line 10-15:

```
select.addEventListener("change", onChange(swatch, event))
```

Code Line 10-16:

```
select.addEventListener("change", (event) => {
    onChange(swatch, event)
})
```

Code Line 10-17:

```
const sizes = {
    "s": "Small",
    "m": "Medium",
    "l": "Large",
}
```

Code Line 10-18:

```
let sizeSelector = Object.create(selectorPrototype)

sizeSelector.render(sizes, sizeSelection, "sizeSelect", changeSize)
```

Code Line 10-19:

```
let menu = {
    inventory: {
        "Regular Coffee": 3.00,
        "Espresso": 3.50,
        "Cappuccino": 4.00,
        "Latte": 4.25
    },
    populate: function(container) {
        for (let item in this.inventory) {
            let price = this.inventory[item]

            // Create a new li element
            let li = document.createElement("li")

            // Add text content to the li element
            li.textContent = item + " - $" + price.toFixed(2)

            // Append the li element to the container
            container.appendChild(li)
        }
    }
}
```

Snippet index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>ClydeBank Coffee Shop</title>
</head>
<body>
    <header>
        <div class="logo-container">
            <img src="coffee-cup.svg" alt="Coffee Cup" class="logo">
        </div>
        <h1>ClydeBank Coffee Shop</h1>
    </header>
    <section class="welcome">
        <h2>Welcome</h2>
        <p>Welcome to ClydeBank Coffee Shop! We serve the best coffee in town.</p>
    </section>
    <section class="shop">
        <h2>Shop</h2>
        <div id="coffee-menu-container">
        </div>
    </section>
    <section class="about-us">
        <h2>About Us</h2>
        <p>Our mission is to provide the highest quality coffee to our community.</p>
    </section>
    <section class="contact-us">
        <h2>Contact Us</h2>
        <p>Email us at <a href="mailto:support@clydebankmedia.com">support@clydebankme-
dia.com</a> or call us at (800) 340-3069.</p>
    </section>
    <footer>
        <p>Copyright &copy; 2024 ClydeBank Coffee Shop</p>
    </footer>
    <script src="shop.js"></script>
</body>
</html>
```

Snippet shop.js

```
// Display a quick message in the console.
console.log("ClydeBank Coffee Shop is now open!")

// The list prototype
const listPrototype = {
    render: (values, parent, id, separator = " - $", className = "") => {

        // Create the unordered list element
        let ul = document.createElement("ul")

        // Assign the HTML element ID
        ul.id = id

        // If className is set, assign it, otherwise ignore
        if (className) {
            ul.className = className
        }

        // Loop through the values array and create a new li element
        // for each
        for (let key in values) {
            // Create a key/value pair in the variables k and v
            let k = key
            let v = values[key]

            // Create the li element
            let li = document.createElement("li")
            // Add the text content to the li element with the separator
            li.textContent = k + separator + v.toFixed(2)

            // Append the li element to the unordered list
            ul.appendChild(li)
        }

        // Append the unordered list to the parent element
        parent.appendChild(ul)
    }
}

let inventory = {
    "Regular Coffee": 3.00,
    "Espresso": 3.50,
    "Cappuccino": 4.00,
    "Latte": 4.25
}

// Obtain reference to the menu list by ID
let menuList = document.getElementById("coffee-menu-container")

// Create the menu object from the prototype
let menu = Object.create(listPrototype)

// Render the menu
// We supply the separator because we need to specify
// the CSS class and you must provide all optional variables
// to the left of the optional variable you are specifying
menu.render(inventory, menuList, "coffee-menu", " - $", "coffee-list")
```

CHAPTER 11
Building a JavaScript-Driven Website

Snippet index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Load Google's Open Sans font -->
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700" rel="-
stylesheet">
    <link rel="stylesheet" href="style.css">
    <title>ClydeBank Coffee Shop</title>
</head>
<body>
<div id="main">
    <header>
        <div class="logo-container">
            <img src="coffee-cup.svg" alt="Coffee Cup" class="logo">
        </div>
        <h1>ClydeBank Coffee Shop</h1>
    </header>
    <section class="welcome">
        <h2>Welcome</h2>
        <p>Welcome to ClydeBank Coffee Shop! We serve the best coffee in town.</p>
    </section>
    <section class="shop">
        <h2>Shop</h2>
        <div id="coffee-menu-container">
        </div>
    </section>
    <section class="about-us">
        <h2>About Us</h2>
        <p>Our mission is to provide the highest quality coffee to our community.</p>
    </section>
    <section class="contact-us">
        <h2>Contact Us</h2>
        <p>Email us at <a href="mailto:support@clydebankmedia.com">support@clydebankme-
dia.com</a> or call us at (800) 340-3069.</p>
    </section>
    <footer>
    <p>Copyright &copy; 2024 ClydeBank Coffee Shop</p>
        </footer>
</div>
```

```
<script src="shop.js"></script>
</body>
</html>
```

Snippet style.css

```
/* Reset some default styles in browsers */
body, h1, h2, p, ul {
    margin: 0;
    padding: 0;
}

/* Add margin below h2 tags */
h2 {
    margin-bottom: 0.4em;
}

/* Apply background color and font to body */
body {
    background-color: #3B1C0B;
    font-family: "Open Sans", sans-serif;
}

/* Header styles */
header {
    background-color: #3B1C0B;
    padding: 20px;
    text-align: center;
    color: white;
}

/* Move the hamburger a bit to the left and down */
.hamburger {
    margin-left: -20px;
    margin-top: 7px;
}

.logo-container {
    display: inline-block;
}

.logo {
    height: 50px;
    margin-right: 20px;
}

/* Section styles */
section {
    padding: 40px 20px;
    text-align: center;
}

.welcome {
    background-color: #C08552;
    color: white;
}

.shop {
    background-color: #F4F1DE;
}

.shop .coffee-list {
    list-style: none;
}

.about-us {
    background-color: #C08552;
    color: white;
}

.contact-us, .contact-us a {
    color: white;
}

/* Footer styles */
footer {
    background-color: #3B1C0B;
    padding: 20px;
    text-align: center;
    color: white;
}

footer.h2 {
     color: white;
}
```

Snippet shop.js

```
// ClydeBank Coffee Shop

// Define hamburger menu object
const hamburgerMenu = {
    render: (parent, id = "hamburger", width = 40, height = 40) => {

        // Construct the SVG
        let svg = document.createElementNS("http://www.w3.org/2000/svg", "svg")
        let rect1 = document.createElementNS("http://www.w3.org/2000/svg", "rect")
        let rect2 = document.createElementNS("http://www.w3.org/2000/svg", "rect")
        let rect3 = document.createElementNS("http://www.w3.org/2000/svg", "rect")

        // Set the attributes for the SVG
        svg.id = id
        svg.classList.add("hamburger")
        svg.setAttribute("width", width)
        svg.setAttribute("height", height)
        svg.setAttribute("viewBox", "0 0 100 80")
        svg.setAttribute("fill", "#cad317")

        // Set the attributes for the rectangles
        rect1.setAttribute("width", "100")
        rect1.setAttribute("height", "20")

        rect2.setAttribute("y", "30")
        rect2.setAttribute("width", "100")
        rect2.setAttribute("height", "20")

        rect3.setAttribute("y", "60")
        rect3.setAttribute("width", "100")
        rect3.setAttribute("height", "20")

        // Add the rectangles to the SVG
        svg.appendChild(rect1)
        svg.appendChild(rect2)
        svg.appendChild(rect3)

        parent.appendChild(svg)
    }
}

// The list prototype
const listPrototype = {
    render: (values, parent, id, separator = " - $", className = "") => {

        // Create the unordered list element
        let ul = document.createElement("ul")

        // Assign the HTML element ID
        ul.id = id

        // If className is set, assign it, otherwise ignore
        if (className) {
            ul.className = className
    }

        // Loop through the values array and create a new li element
        // for each
        for (let key in values) {
            // Create a key/value pair in the variables k and v
            let k = key
            let v = values[key]

            // Create the li element
            let li = document.createElement("li")

            // Add the text content to the li element with the separator
            li.textContent = k + separator + v.toFixed(2)

            // Append the li element to the unordered list
            ul.appendChild(li)
        }

        // Append the unordered list to the parent element
        parent.appendChild(ul)
    }
}

let inventory = {
    "Regular Coffee": 3.00,
    "Espresso": 3.50,
    "Cappuccino": 4.00,
    "Latte": 4.25
}

// Run only when the document is loaded and ready to go
document.addEventListener("DOMContentLoaded", function(event) {

    // The main div
    const header = document.getElementsByTagName("header")[0]

    // Obtain reference to the menu list by ID
    let menuList = document.getElementById("coffee-menu-container")

    // Create the menu object from the prototype
    let menu = Object.create(listPrototype)

    // Render the inventory menu
    menu.render(inventory, menuList, "coffee-menu", " - $", "coffee-list")

    // Render the hamburger menu
    hamburgerMenu.render(header)

})
```

Snippet shop.js

```
// ClydeBank Coffee Shop

// Define hamburger menu object
const hamburgerMenu = {
    render: (parent, id = "hamburger", mainMenuId = "mainMenu", width = 40, height =
40) => {

        // Construct the SVG
        let svg = document.createElementNS("http://www.w3.org/2000/svg", "svg")
        let rect1 = document.createElementNS("http://www.w3.org/2000/svg", "rect")
        let rect2 = document.createElementNS("http://www.w3.org/2000/svg", "rect")
        let rect3 = document.createElementNS("http://www.w3.org/2000/svg", "rect")

        // Set the attributes for the SVG
        svg.id = id
        svg.classList.add("hamburger")
        svg.setAttribute("width", width)
        svg.setAttribute("height", height)
        svg.setAttribute("viewBox", "0 0 100 80")
        svg.setAttribute("fill", "#cad317")

        // Set the attributes for the rectangles
        rect1.setAttribute("width", "100")
        rect1.setAttribute("height", "20")

        rect2.setAttribute("y", "30")
        rect2.setAttribute("width", "100")
        rect2.setAttribute("height", "20")

        rect3.setAttribute("y", "60")
        rect3.setAttribute("width", "100")
        rect3.setAttribute("height", "20")

        // Add the rectangles to the SVG
        svg.appendChild(rect1)
        svg.appendChild(rect2)
        svg.appendChild(rect3)

        // When the hamburger is clicked, show the menu
        // But first, get the menu
        let menu = document.getElementById(mainMenuId)
        svg.addEventListener("click", (event) => {
            // Show the menu
            menu.style.display = "block"
        })

        parent.appendChild(svg)
    }
```

```
}

// Define a menu
const mainMenu = {
    render: (parent, links, id = "mainMenu", hamburgerId = "hamburger",
        minWidth = "300px", maxWidth = "800px",
        backgroundColor = "#F4F1DE", borderColor = "#3B1C0B",
        linkColor = "#3B1C0B") => {

        // Create the div element
        let div = document.createElement("div")

        // Set attributes
        div.id = id
        div.style.minWidth = minWidth
        div.style.maxWidth = maxWidth
        div.style.top = "-1px"
        div.style.left = "-1px"
        div.style.position = "absolute"
        div.style.height = "70%"
        div.style.backgroundColor = backgroundColor
        div.style.border = "1px solid " + borderColor

        // We want the menu to be hidden by default
        div.style.display = "none"

        // Construct the menu
        let ul = document.createElement("ul")

        // Add some padding to the ul element and make the text bigger
        ul.style.padding = "1em"
        ul.style.fontSize = "1.25em"

        // Build the menu
        for (let link in links) {

            // Create the needed elements
            let li = document.createElement("li")
            let a = document.createElement("a")

            // Make sure the link is readable
            a.style.color = linkColor

            // Hide the dot
            li.style.listStyle = "none"

            // Create the link
            a.href = links[link]
            a.innerHTML = link
            // Add the completed HTML of the link
            // (i.e., <a href=”...”>...</a>) to the list item
            li.innerHTML = a.outerHTML

            // Add the list item (li) to the unordered list (ul)
            ul.appendChild(li)
        }

        // When the parent of the menu (i.e., the main div)
        // is clicked, hide the menu
        div.addEventListener(“click”, (event) => {
            // Hide the menu
            div.style.display = “none”
        })

        // Add the list’s completed HTML to the div
        div.innerHTML = ul.outerHTML

        parent.appendChild(div)
    }
}

// The list prototype
const listPrototype = {
    render: (values, parent, id, separator = “ - $”, className = “”) => {

        // Create the unordered list element
        let ul = document.createElement(“ul”)

        // Assign the HTML element ID
        ul.id = id

        // If className is set, assign it, otherwise ignore
        if (className) {
            ul.className = className
        }

        // Loop through the values array and create a new li element
        // for each
        for (let key in values) {
            // Create a key/value pair in the variables k and v
            let k = key
            let v = values[key]

            // Create the li element
            let li = document.createElement(“li”)

            // Add the text content to the li element with the separator
            li.textContent = k + separator + v.toFixed(2)
            // Append the li element to the unordered list
            ul.appendChild(li)
        }

        // Append the unordered list to the parent element
        parent.appendChild(ul)
    }
}

// Menu structure
// (for now, these are all links to the home page)
const menuLinks = {
    "Home": "index.html",
    "About": "index.html",
    "Contact": "index.html"
}

// Our inventory
let inventory = {
    "Regular Coffee": 3.00,
    "Espresso": 3.50,
    "Cappuccino": 4.00,
    "Latte": 4.25
}

// Run only when the document is loaded and ready to go
document.addEventListener("DOMContentLoaded", function(event) {

    // The main div
    const header = document.getElementsByTagName("header")[0]

    // Obtain reference to the menu list by ID
    let menuList = document.getElementById("coffee-menu-container")

    // Create the menu object from the prototype
    let menu = Object.create(listPrototype)

    // Render the inventory menu
    menu.render(inventory, menuList, "coffee-menu", " - $", "coffee-list")

    // Render the menu (hidden by default)
    mainMenu.render(main, menuLinks)

    // Render the hamburger menu
    hamburgerMenu.render(header)

})
```

Figure 28

innerHTML

```
<li>First Item</li>
<li>Second Item</li>
```

outerHTML

```
<ul>
<li>First Item</li>
<li>Second Item</li>
```

With `outerHTML`, we can reference the entire element, including its enclosing tags, `innerHTML`, `outerHTML`, `</ul>`.

Code Line 11-1:

```
let menu = document.getElementById(mainMenuId)
```

Code Line 11-2:

```
render: (parent, id = “hamburger”, mainMenuId = “mainMenu”, width = 40, height = 40) =>
{
```

Code Line 11-3:

```
render: (args) => {
```

Code Line 11-4:

```
args = {
    “parent”: header,
    “id”: “hamburger”,
    “mainMenuId”: “mainMenu”,
    “width”: 40,
    “height”: 40,
}

hamburgerMenu.render(args)
```

Code Line 11-5:

```
svg.setAttribute(“width”, args.width)
svg.setAttribute(“height”, args.height)
```

Code Line 11-6:

```
<a href=”another-page.html”>Another page</a>
```

Code Line 11-7:

```
<div id=”main”>
</div>

<div id=”about”>
</div>

<div id=”contact”>
</div>
```

Code Line 11-8:

```
<div id=”main”>
</div>

<div id=”about” style=”display: none;”>
</div>

<div id=”contact” style=”display: none;”>
</div>
```

Code Line 11-9:

```
<section id=”about” class=”about-us” style=”display:none;”>
   <h2>About Us</h2>
   <p>Our mission is to provide the highest quality coffee to our community.</p>
</section>
<section id=”contact” class=”contact-us” style=”display: none;”>
   <h2>Contact Us</h2>
   <p>Email us at <a href=”mailto:support@clydebankmedia.com”>support@clydebankmedia.
com</a> or call us at (800) 340-3069.</p>
</section>
```

Code Line 11-10:

```
const menuLinks = {
    "Home": "index.html",
    "About": "index.html",
    "Contact": "index.html"
}
```

Code Line 11-11:

```
const menuLinks = {
    "Home": "#",
    "About": "#",
    "Contact": "#"
}
```

Code Line 11-12:

```
// ClydeBank Coffee Shop

var currentPage = "main"

// Define hamburger menu object
const hamburgerMenu = {
```

Snippet shop.js

```
// ClydeBank Coffee Shop

var currentPage = "main"

// Define hamburger menu object
const hamburgerMenu = {
    render: (parent, id = "hamburger", mainMenuId = "mainMenu", width = 40, height =
40) => {

        // Construct the SVG
        let svg = document.createElementNS("http://www.w3.org/2000/svg", "svg")
        let rect1 = document.createElementNS("http://www.w3.org/2000/svg", "rect")
        let rect2 = document.createElementNS("http://www.w3.org/2000/svg", "rect")
        let rect3 = document.createElementNS("http://www.w3.org/2000/svg", "rect")

        // Set the attributes for the SVG
        svg.id = id
        svg.classList.add("hamburger")
        svg.setAttribute("width", width)
        svg.setAttribute("height", height)
        svg.setAttribute("viewBox", "0 0 100 80")
        svg.setAttribute("fill", "#cad317")

        // Set the attributes for the rectangles
        rect1.setAttribute("width", "100")
        rect1.setAttribute("height", "20")

        rect2.setAttribute("y", "30")
        rect2.setAttribute("width", "100")
        rect2.setAttribute("height", "20")

        rect3.setAttribute("y", "60")
        rect3.setAttribute("width", "100")
        rect3.setAttribute("height", "20")

        // Add the rectangles to the SVG
        svg.appendChild(rect1)
        svg.appendChild(rect2)
        svg.appendChild(rect3)

        // When the hamburger is clicked, show the menu
        // But first, get the menu
        let menu = document.getElementById(mainMenuId)
    svg.addEventListener("click", (event) => {
            // Show the menu
            menu.style.display = "block"
        })

        parent.appendChild(svg)
    }
}

// Define a menu
const mainMenu = {
    render: (parent, links, id = "mainMenu", hamburgerId = "hamburger",
        minWidth = "300px", maxWidth = "800px",
        backgroundColor = "#F4F1DE", borderColor = "#3B1C0B",
        linkColor = "#3B1C0B") => {

        // Create the div element
        let div = document.createElement("div")

        // Set attributes
        div.id = id
        div.style.minWidth = minWidth
        div.style.maxWidth = maxWidth
        div.style.top = "-1px"
        div.style.left = "-1px"
        div.style.position = "absolute"
div.style.height = “70%”
div.style.backgroundColor = backgroundColor
div.style.border = “1px solid ” + borderColor

// We want the menu to be hidden by default
div.style.display = “none”

// Construct the menu
let ul = document.createElement(“ul”)

// Add some padding to the ul element and make the text bigger
ul.style.padding = “1em”
ul.style.fontSize = “1.25em”

// Build the menu
for (let link in links) {

    // Create the needed elements
    let li = document.createElement(“li”)
    let a = document.createElement(“a”)

    // Make sure the link is readable
    a.style.color = linkColor

    // Hide the dot
    li.style.listStyle = “none”

    // Create the link
    a.href = links[link]
    a.innerHTML = link

    // If the link is a pound sign, treat differently
    if (links[link] == “#”) {
        // When the link is clicked, hide the menu
        a.addEventListener(“click”, (event) => {
            // Prevent the default action (i.e., following the
            // link)
            event.preventDefault()
            // If “Home” is link title, make newPage “main”
            let newPage
            if (link == “Home”) {
                newPage = “main”
            } else {
                // Otherwise, set lowercase version of the new
                // page name
                newPage = link.toLocaleLowerCase()
            }

            // Hide the current page if not main, since it
            // contains everything else
```

```
                    if (currentPage != “main”) {
                        document.getElementById(currentPage).style.display = “none”
                    }

                    // Show the new page
                    document.getElementById(newPage).style.display = “block”

                    // Set the currentPage variable to the new page
                    currentPage = newPage
                })
            }

            // Add the list item (li) to the unordered list (ul) with
            // the link
            li.appendChild(a)
            ul.appendChild(li)
        }

        // When the parent of the menu (i.e., the main div)
        // is clicked, hide the menu
        div.addEventListener(“click”, (event) => {
            // Hide the menu
            div.style.display = “none”
        })

        // Add the list’s completed HTML to the div
        div.appendChild(ul)

        // Now add the main div to the parent
        parent.appendChild(div)
    }
}

// The list prototype
const listPrototype = {
    render: (values, parent, id, separator = “ - $”, className = “”) => {

        // Create the unordered list element
        let ul = document.createElement(“ul”)

        // Assign the HTML element ID
        ul.id = id

        // If className is set, assign it, otherwise ignore
        if (className) {
            ul.className = className
        }

        // Loop through the values array and create a new li element
        // for each
        for (let key in values) {
            // Create a key/value pair in the variables k and v
            let k = key
            let v = values[key]

            // Create the li element
            let li = document.createElement("li")

            // Add the text content to the li element with the separator
            li.textContent = k + separator + v.toFixed(2)

            // Append the li element to the unordered list
            ul.appendChild(li)
        }

        // Append the unordered list to the parent element
        parent.appendChild(ul)
    }
}

// Menu structure
// (for now, these are all links to the home page)

const menuLinks = {
    "Home": "#",
    "About": "#",
    "Contact": "#"
}

// Our inventory
let inventory = {
    "Regular Coffee": 3.00,
    "Espresso": 3.50,
    "Cappuccino": 4.00,
    "Latte": 4.25
}

// Run only when the document is loaded and ready to go
document.addEventListener("DOMContentLoaded", function(event) {

    // The main div
    const header = document.getElementsByTagName("header")[0]

    // Obtain reference to the menu list by ID
    let menuList = document.getElementById("coffee-menu-container")

    // Create the menu object from the prototype
    let menu = Object.create(listPrototype)

    // Render the inventory menu
    menu.render(inventory, menuList, “coffee-menu”, “ - $”, “coffee-list”)

    // Render the menu (hidden by default)
    mainMenu.render(main, menuLinks)

    // Render the hamburger menu
    hamburgerMenu.render(header)

})
```

Code Line 11-13:

```
// Set attributes
    div.id = id
    div.style.minWidth = minWidth
    div.style.maxWidth = maxWidth
    div.style.top = “-1px”
    div.style.left = “-1px”
    div.style.position = “absolute”
    div.style.height = “80%”
    div.style.backgroundColor = backgroundColor
    div.style.border = “1px solid “ + borderColor
```

Code Line 11-14:

```
#mainMenu {
    top: -1px;
    left: -1px;
}
```

CHAPTER 12
Advanced JavaScript Techniques

Snippet 12-01.js

```
let greeting = "Hello, World!"
let subject = greeting.slice(7, 12)
console.log(subject)
```

Code Line 12-1:

```
let shoppingList = ["milk", "eggs", "apples", "grapes", "hamburger"]
let justFruit = shoppingList.slice(2, 4)
console.log(justFruit)
```

Code Line 12-2:

```
[ 'apples', 'grapes' ]
```

Code Line 12-3:

```
option.innerHTML = color.charAt(0).toUpperCase() + color.slice(1)
```

Code Line 12-4:

```
var theString = "Hello, World!"
console.log(theString.length)
```

Code Line 12-5:

```
let firstName = document.getElementById("firstName")
if (firstName.value.length > 0) {
    console.log("You have entered a value for first name")
}
```

Code Line 12-6:

```
const theURL = window.location.href
```

Code Line 12-7:

```
const urlParts = theURL.split("/")
```

Code Line 12-8:

```
const urlParts = window.location.href.split("/")
```

Code Line 12-9:

```
console.log(urlParts)
```

Code Line 12-10:

```
(5) ['https:', '', 'example.com', 'about', 'our-team']
```

Code Line 12-11:

```
if (urlParts.includes("our-team")) {
   // Show div for about and our-team content
}
```

Code Line 12-12:

```
let comment = "A not nice thing about someone."
```

Code Line 12-13:

```
let filteredComment = comment.replace("not nice", "nice")
console.log(filteredComment)
```

Code Line 12-14:

```
"A nice thing about someone."
```

Code Line 12-15:

```
let comment = "A not nice thing about someone."
let filteredComment = comment.replace("not", "")
console.log(filteredComment)
```

Code Line 12-16:

```
let filteredComment = comment.replace("not ", "")
```

Code Line 12-17:

```
let filteredComment = "A not nice thing about someone.".replace("not nice", "nice")
```

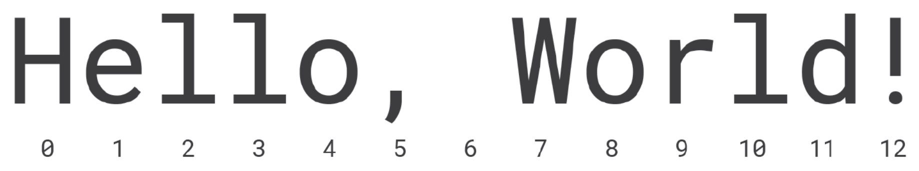
Code Line 12-18:

```
let greeting = "Hello, World!"
console.log(greeting.substring(6, 13))
```

Code Line 12-19:

```
 World!
```

Figure 29



With substring and other operations on strings, indexing starts at zero.

Code Line 12-20:

```
console.log(greeting.substring(7))
```

Code Line 12-21:

```
World!
```

Code Line 12-22:

```
console.log(greeting.substring(greeting.length - 6))
```

Code Line 12-23:

```
let greeting = ‶    Robert Oliver  ‶
console.log(greeting.trim())
```

Table 4

ORDER OF OPERATIONS					
P (Parentheses)	**E** (Exponent)	**M** (Multiply)	**D** (Divide)	**A** (Add)	**S** (Subtract)
B (Bracket)	**O** (Order)	**D** (Divide)	**M** (Multiply)	**A** (Add)	**S** (Subtract)
()	**$\sqrt{x}x^2$**	**÷ or x**		**+ or –**	

Code Line 12-24:

```
let result = (3 * 2) / 4 - (2 + 7) + 4 ** 3
console.log(result)
```

Code Line 12-25:

```
console.log(Math.PI)
```

Table 5

CODE	DESCRIPTION	VALUE
Math.E	Euler's number, the base of the natural logarithm.	2.718281828459045
Math.LN10	The natural logarithm of 10.	2.302585092994046
Math.LN2	The natural logarithm of 2.	0.6931471805599453
Math.LOG10E	The base 10 logarithm of e.	0.4342944819032518
Math.LOG2E	The base 2 logarithm of e.	1.4426950408889634
Math.PI	The ratio of the circumference of a circle to its diameter.	3.141592653589793
Math.SQRT1_2	The square root of ½.	0.7071067811865476
Math.SQRT2	The square root of 2.	1.4142135623730951

Code Line 12-26:

```
// roundedNumber will be set to 1
let roundedNumber = Math.round(1.2)
```

Code Line 12-27:

```
// roundedNumber will be set to 1
let roundedNumber = Math.floor(1.6)
```

Code Line 12-28:

```
// roundedNumber will be set to 2
let roundedNumber = Math.ceil(1.3)
```

Code Line 12-29:

```
console.log(Math.random())
```

Code Line 12-30:

```
0.06813061476164983
```

Code Line 12-31:

```
let randomNumber = Math.floor(Math.random() * 100) + 1
console.log(randomNumber)
```

Table 6

METHOD	DESCRIPTION	EXAMPLE
`Math.abs`	Return the absolute value (i.e., make the number positive)	`Math.abs(4.1)`
`Math.cos`	Return the cosine of an angle	`Math.cos(1)`
`Math.log`	Return the natural logarithm (base e)	`Math.log(5)`
`Math.sin`	Return the sine of an angle	`Math.sin(1)`
`Math.tan`	Return the tangent of an angle	`Math.tan(1)`

Code Line 12-32:

```
const now = new Date()
console.log(now)
```

Code Line 12-33:

```
Thu Apr 06 2023 20:38:26 GMT-0500 (Central Daylight Time)
```

Code Line 12-34:

```
const now = new Date()
const month = now.getMonth() + 1
const day = now.getDate()
const year = now.getFullYear()
const hours = now.getHours()
const minutes = now.getMinutes()
const seconds = now.getSeconds()
```

Code Line 12-35:

```
console.log(`${month}-${day}-${year} ${hours}:${minutes}:${seconds}`)
```

Code Line 12-36:

```
const now = new Date()
console.log(now.toDateString())
```

Code Line 12-37:

```
Thu Apr 06 2023
```

Code Line 12-38:

```
const now = new Date()
console.log(now.toTimeString())
```

Code Line 12-39:

```
21:14:22 GMT-0500 (Central Daylight Time)
```

Code Line 12-40:

```
try {
    // This code might cause an error
} catch (error) {
    // This code handles the error
    // The error object contains details about the error
} finally {
    // This code always runs, regardless of the error
}
```

Code Line 12-41:

```
function greeting(firstName) {
    if (firstName.length === 0) {
        // The string firstName is empty
        throw new Error("The firstName argument is empty.")
    }
    console.log("Hello, " + name + "!")
}
```

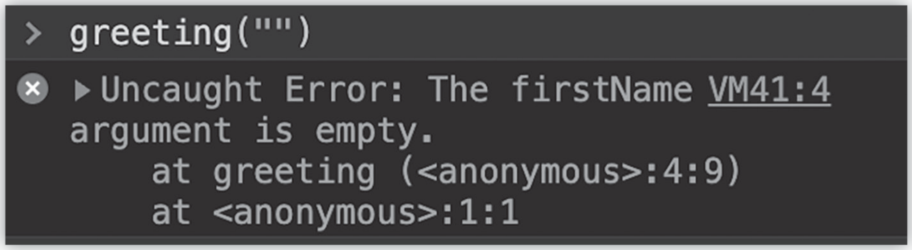
Code Line 12-42:

```
try {
    greeting(“”)
} catch (error) {
    // This code would run because an error was thrown
    console.log(‘An error occurred: ${error.message}’)
}
```

Code Line 12-43:

```
An error occurred: The firstName argument is empty.
```

Figure 30



```
> greeting("")
✖ ▶Uncaught Error: The firstName VM41:4
argument is empty.
    at greeting (<anonymous>:4:9)
    at <anonymous>:1:1
```

An uncaught error. The `error.message` is still shown.

Code Line 12-44:

```
function greeting() {
    console.log(“Hello, World!”)
}
```

Code Line 12-45:

```
greeting = function() {
    console.log(“Hello, World!”)
}
```

Code Line 12-46:

```
greeting = () => {
console.log(“Hello, World”)
}
```

Code Line 12-47:

```
greeting = function() {
    return “Hello, World!”
}
```

Code Line 12-48:

```
greeting = () => “Hello, World!”
```

Code Line 12-49:

```
document.cookie = “firstName=Robert”
```

Snippet 12-02.js

```
function getCookie(name) {
    // Add the equal sign to the name (name=)
    let cookieName = name + “=”

    // Decode cookie string to handle cookies with special characters
    let decodedCookie = decodeURIComponent(document.cookie)

    // Check if there are any cookies to process
    if (!decodedCookie) {
        return null
    }

    // Split document.cookie on semicolons into an array
    let cookieArray = decodedCookie.split(“;”)

    // Iterate through the name=value pairs in the cookieArray
    for (let i = 0; i < cookieArray.length; i++) {
        let cookie = cookieArray[i]

        // Remove any leading spaces using a regular expression
        cookie = cookie.replace(/^\s+/, “”)

        // If cookie is found, return its value
        if (cookie.indexOf(cookieName) === 0) {
            return cookie.substring(cookieName.length, cookie.length)
        }
    }
    // If cookie name was not found, return null
    return null
}
```

Code Line 12-50:

```
let firstName = getCookie("firstName")
```

Code Line 12-51:

```
"myName=Robert%20Oliver"
```

Code Line 12-52:

```
decodeURIComponent("myName=Robert%20Oliver")
```

Code Line 12-53:

```
myName=Robert Oliver
```

Code Line 12-54:

```
document.cookie = "firstName=Marsha"
```

Code Line 12-55:

```
function createCookieExpiration(hours) {
    // Create a new Date object
    let expirationDate = new Date()
    // Add the correct number of milliseconds to the existing time
        expirationDate.setTime(expirationDate.getTime() + (hours * 60 * 60 * 1000))
    // Return the time in a format expected by the browser
    return expirationDate.toUTCString()
}
```

Code Line 12-56:

```
expTime = createCookieExpiration(168)
document.cookie = "firstName=Robert; expires=" + expTime
```

Code Line 12-57:

```
function removeCookie(cookieName) {
    document.cookie = cookieName + “=; expires=Thu, 01 Jan 1970 00:00:00 UTC;”
}
```

Code Line 12-58:

```
document.cookie = “firstName=Robert”
```

Code Line 12-59:

```
document.cookie = “firstName=Robert; path=/”
```

Code Line 12-60:

```
document.cookie = “total=24.95; path=/store”
```

Snippet style.css

```
/* Cart styles */
.add-to-cart-button {
    margin-left: 1em;
}
```

Snippet index.html

```
<!DOCTYPE html>
<html lang=”en”>
<head>
    <meta charset=”UTF-8”>
    <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
    <!-- Load Google’s Open Sans font -->
    <link href=”https://fonts.googleapis.com/css?family=Open+Sans:400,700” rel=”-
stylesheet”>
    <link rel=”stylesheet” href=”style.css”>
    <title>ClydeBank Coffee Shop</title>
</head>
<body>
<div id=”main”>
    <header>
        <div class=”logo-container”>
            <img src=”coffee-cup.svg” alt=”Coffee Cup” class=”logo”>
        </div>
        <h1>ClydeBank Coffee Shop</h1>
    </header>
    <section class="welcome">
        <h2>Welcome</h2>
        <p>Welcome to ClydeBank Coffee Shop! We serve the best coffee in town.</p>
    </section>
    <section class="shop">
        <h2>Shop</h2>
        <div id="coffee-menu-container">
        </div>
    </section>
    <section class="about-us">
        <h2>About Us</h2>
        <p>Our mission is to provide the highest quality coffee to our community.</p>
    </section>
    <section class="contact-us">
        <h2>Contact Us</h2>
        <p>Email us at <a href="mailto:support@clydebankmedia.com">support@clydebankme-
dia.com</a> or call us at (800) 340-3069.</p>
    </section>
    <footer>
        <p>Copyright &copy; 2024 ClydeBank Coffee Shop</p>
    </footer>
</div>
<script src="shop.js"></script>
</body>
</html>
```

Snippet shop.js

```
// ClydeBank Coffee Shop
// Copyright (C) 2024 ClydeBank Media, All Rights Reserved.
// Written by Robert W. Oliver II, Author of JavaScript QuickStart Guide

// The locale and currency to use for formatting
// Alter these to suit your needs
const locale = "en-US"
const currency = "USD"

// Menu structure
const menuLinks = {
    "Home": "index.html",
    "About": "index.html",
    "Contact": "index.html"
}

// Our inventory
const inventory = {
    "Regular Coffee": 3.00,
    "Espresso": 3.50,
    "Cappuccino": 4.00,
    "Latte": 4.25
}

// Random number to add to the end of IDs that aren't
// specified to prevent collisions. Only one per page load is needed
const randomIdPostfix = Math.floor(Math.random() * 100000)

// This object is used here to format currency
// (see addToCart() and listPrototype for use)
const intlFormat = new Intl.NumberFormat(locale, {
    style: "currency",
    currency: currency,
})

// Retrieve a cookie by name
function getCookie(name) {
    // Add the equal sign to the name (name=)
    let cookieName = name + "="

    // Decode cookie string to handle cookies with special characters
    let decodedCookie = decodeURIComponent(document.cookie)

    // Check if there are any cookies to process
    if (!decodedCookie) {
        return null
    }

    // Split document.cookie on semicolons into an array
    let cookieArray = decodedCookie.split(";")

    // Iterate through the name=value pairs in the cookieArray
    for (let i = 0; i < cookieArray.length; i++) {
        let cookie = cookieArray[i]
        // Remove any leading spaces using a regular expression
        cookie = cookie.replace(/^\s+/, "")

        // If cookie is found, return its value
        if (cookie.indexOf(cookieName) === 0) {
            return cookie.substring(cookieName.length, cookie.length)
        }
    }

    // If cookie name was not found, return null
    return null
}

function replaceNonAlphanumericWithDashes(input) {
    // Regular expression to match spaces and non-alphanumeric
    // characters
    let regex = /[^a-zA-Z0-9]+/g
    // Replace matches with dashes
    let output = input.replace(regex, "-")
    return output
}

// Obtain the cart and return it as JSON
function getCart() {
    // Retrieve the "cart" cookie
    let cartCookie = getCookie("cart")

    // If it exists, parse contents and return the result
    // Otherwise return an empty object
    if (cartCookie) {
        return JSON.parse(cartCookie)
    } else {
        return {}
    }
}

function saveCart(cart) {
    // Save the updated cart back to the cookie
    document.cookie = "cart=" + encodeURIComponent(JSON.stringify(cart))
}

// Add an item to the cart
function addToCart(itemName, itemPrice, itemQnty) {
    // Load the existing cart from the cookie (if it exists)
    // If it doesn't exist, cart will be empty
    let cart = {}
    if (document.cookie) {
        cart = JSON.parse(document.cookie.replace("cart=", ""))
        // This strips "cart=stringifiedJSON" into "stringifiedJSON",
        // then parses that and assigns it to the cart object
    }

    // Create an object with the name, price, and quantity to add
    const item = {
        name: itemName,
        price: itemPrice,
        quantity: itemQnty
    }

    // If a cart item with the same name already exists,
    // increment quantity. If not, add item to cart
    if (cart[itemName]) {
        cart[itemName].quantity += itemQnty
    } else {
        cart[itemName] = item
    }

    // Log to console for diagnostic purposes
    console.log("Added " + JSON.stringify(item) + " to cart!")

    // Save the cart back to the cookie
    saveCart(cart)
}

// Remove item from cart
function removeFromCart(itemName) {
    // Retrieve the "cart" cookie
    let cartCookie = getCookie("cart")

    // If it exists, parse contents
    if (cartCookie) {
        let cart = JSON.parse(cartCookie)

        // If the item exists in the cart, remove it
        if (cart[itemName]) {
            delete cart[itemName]

            // Save the update cart back to the cookie
            saveCart(cart)
        }
    }
}

// Define a button prototype
const buttonPrototype = {
    // HTML element id (defaults to hamburger)
    // Plus random postfix to prevent collisions
    id: "button" + randomIdPostfix,
    // The text to be displayed on the button
    text: "Click Me",
    // The color of the button
    color: "#CAD317",
    // The border style of the button
    borderStyle: "1px solid #3B1C0B",
    // The click event handler
    clickEvent: false,
    // The CSS class name (defaults to "")
    className: "",
    // The render method (with the parent element as an argument)
    render: function(parent) {
    // Set the variables from properties of this object with the same
    // name
    let {id, text, color, borderStyle, clickEvent, className} = this

    // Construct the button
    let button = document.createElement("button")

        // Set the attributes
        button.id = id
        button.textContent = text
        button.style.border = borderStyle

        // Assign the clickEvent argument (passed as function) to the
        // onclick event
        button.onclick = clickEvent

        // If className is set, assign it, otherwise ignore
        if (className) {
            button.className = className
        }

        // Set the background color
        button.style.backgroundColor = color

        // Add the button to the parent element
        parent.appendChild(button)
    }
}

// Define hamburger menu prototype
const hamburgerMenuPrototype = {
    // HTML element id (defaults to hamburger)
    // Plus random postfix to prevent collisions
    id: "hamburger" + randomIdPostfix,
    // The ID of the mainMenu div (defaults to mainMenu)
    mainMenuId: "mainMenu",
    // The width and height of the hamburgerMenu (by default, 40)
    width: 40,
    height: 40,
    // The color of the hamburgerMenu
    color: "#CAD317",
    // The render method (with the parent element as an argument)
    render: function(parent) {
        // Set the variables from properties of this object with the
        // same name
        let {id, mainMenuId, width, height, color} = this

        // Construct the SVG
        const svgUrl = "http://www.w3.org/2000/svg"
        let svg = document.createElementNS(svgUrl, "svg")
        let rect1 = document.createElementNS(svgUrl, "rect")
        let rect2 = document.createElementNS(svgUrl, "rect")
        let rect3 = document.createElementNS(svgUrl, "rect")

        // Set the attributes for the SVG
        svg.id = id
        svg.classList.add("hamburger")
        svg.setAttribute("width", width)
        svg.setAttribute("height", height)
        svg.setAttribute("viewBox", "0 0 100 80")
        svg.setAttribute("fill", color)

        // Set the attributes for the rectangles
        rect1.setAttribute("width", "100")
        rect1.setAttribute("height", "20")

        rect2.setAttribute("y", "30")
        rect2.setAttribute("width", "100")
        rect2.setAttribute("height", "20")

        rect3.setAttribute("y", "60")
        rect3.setAttribute("width", "100")
        rect3.setAttribute("height", "20")

        // Add the rectangles to the SVG
        svg.appendChild(rect1)
        svg.appendChild(rect2)
        svg.appendChild(rect3)

        // When the hamburger is clicked, show the menu
        // But first, get the menu
        let menu = document.getElementById(mainMenuId)
        svg.addEventListener("click", (event) => {
            // Show the menu
            menu.style.display = "block"
        })

        parent.appendChild(svg)
    }
}

// Define a menu
const mainMenuPrototype = {
    // The links to be displayed in the menu
    links: {},
    // The ID of the mainMenu div (defaults to mainMenu)
    // Plus a random postfix to prevent collisions
    id: "mainMenu" + randomIdPostfix,
    // The ID of the hamburger menu (defaults to hamburger)
    hamburgerId: "hamburger",
    // The minimum width and minimum height of the menu (defaults to 300px, 800px)
    minWidth: "300px",
    maxWidth: "800px",
    // The background color of the menu (defaults to #F4F1DE)
    backgroundColor: "#F4F1DE",
    // The border color of the menu (defaults to #3B1C0B)
    borderColor: "#3B1C0B",
    // The link color of the menu (defaults to #3B1C0B)
    linkColor: "#3B1C0B",
    // The render method (with the parent element as an argument)
    render: function(parent) {
        // Set the variables from properties of this object with the
        // same name
        let {links, id, minWidth, maxWidth, backgroundColor, borderColor, linkColor} =
this

        // Create the div element
        let div = document.createElement("div")

        // Set attributes
        div.id = id
        div.style.minWidth = minWidth
        div.style.maxWidth = maxWidth
        div.style.top = "-1px"
        div.style.left = "-1px"
        div.style.position = "absolute"
        div.style.height = "70%"
        div.style.backgroundColor = backgroundColor
        div.style.border = "1px solid " + borderColor

        // We want the menu to be hidden by default
        div.style.display = "none"

        // Construct the menu
        let ul = document.createElement("ul")

        // Add some padding to the ul element and make the text bigger
        ul.style.padding = "1em"
        ul.style.fontSize = "1.25em"

        // Build the menu
        for (let link in links) {

            // Create the needed elements
            let li = document.createElement("li")
            let a = document.createElement("a")

            // Make sure the link is readable
            a.style.color = linkColor

            // Hide the dot
            li.style.listStyle = "none"

            // Create the link
            a.href = links[link]
            a.innerHTML = link
            // Add the completed HTML of the link
            // (i.e., <a href=”...”>...</a>) to the list item
            li.innerHTML = a.outerHTML

            // Add the list item (li) to the unordered list (ul)
            ul.appendChild(li)
        }

        // When the parent of the menu (i.e., the main div)
        // is clicked, hide the menu
        div.addEventListener(“click”, (event) => {
            // Hide the menu
            div.style.display = “none”
        })

        // Add the list’s completed HTML to the div
        div.innerHTML = ul.outerHTML

        parent.appendChild(div)
‘}
}

// The list prototype
const listPrototype = {
    // Values to be displayed in the list
    values: {},
    // The HTML element id (defaults to list + random number)
    // The random number is used to prevent duplicate IDs in the case
    // of multiple lists on the same page
    id: “list” + randomIdPostfix,
    // The separator between the key and value (defaults to “ - $”)
    separator: “ - “,
    // The class name of the list (defaults to “”)
    className: “”,
    // Should we format the values as currency? (defaults to true)
    formatCurrency: true,
    // The render method, with the parent element as an argument
    render: function(parent) {
        // Set the variables from properties of this object with the
        // same name
        let {values, id, separator, className, formatCurrency} = this


        if (!parent) {
            throw new Error(“listPrototype: parent property not defined”)
        }
        // Create the unordered list element
        let ul = document.createElement(“ul”)

        // Assign the HTML element ID
        ul.id = id
        // If className is set, assign it, otherwise ignore
        if (className) {
            ul.className = className
        }

        // Loop through the values array and create a new li element
        // for each
        for (let key in values) {
            // Create a key/value pair in the variables k and v
            let k = key
            let v = values[key]

            // Create the li element
            let li = document.createElement("li")

            // Add an id to the li element based on the name
            li.id = "item-" + replaceNonAlphanumericWithDashes(k)

            // Add the text content to the li element with the separator
            // If formatCurrency is true, format the value as currency
            if (formatCurrency) {
                li.textContent = k + separator + intlFormat.format(v)
            } else {
                li.textContent = k + separator + v
            }

            // Append the li element to the unordered list
            ul.appendChild(li)
        }

        // Append the unordered list to the parent element
        parent.appendChild(ul)
    }
}

// Cart view prototype
const cartViewPrototype = {
    // The background color of the cart (defaults to #F4F1DE)
    backgroundColor: "#F4F1DE",
    // The border of the cart (defaults to #3B1C0B)
    borderStyle: "1px solid #3B1C0B",
    // The padding of the cart (defaults to 1em)
    padding: "1em",
    // The title of the cart (defaults to "Your Cart")
    cartTitle: "Your Cart",
    // The contents of the cart (defaults to {})
    cartContents: {},
    render: () => {
        // Set the variables from properties of this object with the
        // same name
        let {backgroundColor, borderStyle, padding, cartTitle, cartContents} = this

        // Create a container div in the middle of the screen
        let div = document.createElement("div")
        div.style.position = "absolute"
        div.style.top = "50%"
        div.style.left = "50%"
        div.style.transform = "translate(-50%, -50%)"
        div.style.backgroundColor = backgroundColor
        div.style.border = borderStyle
        div.style.padding = padding
        div.style.minWidth = "300px"
        div.style.maxWidth = "600px"

        // Add a title to the cart
        let h2 = document.createElement("h1")
        h2.textContent = cartTitle
        div.appendChild(h2)

        // Create the unordered list element
        let ul = document.createElement("ul")

        // Loop through the cart and create a new li element for each
        for (let item in cartContents) {
            // Create a key/value pair in the variables k and v
            let price = cartContents[item]

            // Create the li element
            let li = document.createElement("li")

            // Add the text content to the li element with the separator
            li.textContent = k + separator + v

            // Append the li element to the unordered list
            ul.appendChild(li)
        }

        // Append the unordered list to the parent element
        div.appendChild(ul)
    }
}

// Run only when the document is loaded and ready to go
document.addEventListener("DOMContentLoaded", function(event) {

    // Important divs to reference
    // Technically, body is not a div, but it's a useful shortcut
    const body = document.body
    const header = document.getElementsByTagName("header")[0]
    const coffeeMenu = document.getElementById("coffee-menu-container")

    // Create the menu object from the prototype
    let menu = Object.create(listPrototype)
    menu.values = inventory
    menu.className = "coffee-list"
    menu.render(coffeeMenu)

    // Create add to cart buttons for each menu item
    for (let item in inventory) {
        // Create the button
        let button = Object.create(buttonPrototype)

        // Give the button a unique name
        button.id = "button" + "-" + replaceNonAlphanumericWithDashes(item)

        // Add the text and style to the button
        button.text = "Add to Cart"
        button.className = "add-to-cart-button"

        // Handle the button click
        button.clickEvent = function() {
            // Add the item to the cart
            // Use 1 quantity for now, but this could be added
            // as a property of the button later
            addToCart(item, inventory[item], 1)
            alert("Item added to cart!")
        }

        // Get the li element to add the add to cart button to
        const itemId = "item-" + replaceNonAlphanumericWithDashes(item)
        const li = document.getElementById(itemId)

        // Render the button
        button.render(li)
    }

    // Render the menu (hidden by default)
    let mainMenu = Object.create(mainMenuPrototype)
    mainMenu.id = "mainMenu"
    mainMenu.links = menuLinks
    mainMenu.render(body)

    // Render the hamburger menu
    let hamburgerMenu = Object.create(hamburgerMenuPrototype)
    hamburgerMenu.id = "hamburger"
    hamburgerMenu.render(header)

})
```

CHAPTER 13
Animating with JavaScript

Snippet animate1.html

```
<!DOCTYPE html>
<html lang=”en”>
<head>
    <meta charset=”UTF-8”>
    <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
    <meta http-equiv=”X-UA-Compatible” content=”ie=edge”>
    <title>Animation Demo 1</title>
</head>
<body>
    <button id=”theButton” style=”position: relative; left: 0;”>
        Click Me!
    </button>
    <script src=”animate1.js”></script>
</body>
</html>
```

Snippet animate1.js

```
// Bind event to button click
let element = document.getElementById(“theButton”)

// Pass the function to the click event
element.addEventListener(“click”, animateButton)

// Toggle direction of animation (1 for right, -1 for left)
let direction = 1
// Set the speed of the animation
let speed = 1
// Declare the interval variable
let id

function animateButton() {
    // If an existing interval exists, clear it
    clearInterval(id)

    // Get current position of the element
    let position = parseInt(window.getComputedStyle(element).left, 10)

    // Create the interval
    id = setInterval(frame, 10)
    // This is called every 10 milliseconds
    function frame() {
        // Check to see if position is greater than
        // or equal to 500 and direction is right
        if (position >= 500 && direction == 1) {
            // If so, change to left
            direction = -1
        } else if (position <= 0 && direction == -1) {
            // Stop animation when we reach the leftmost end
            // and reset direction
            clearInterval(id)
            direction = 1
        } else {
            // Increment or decrement position
            position += direction * speed
            // Set style to match new position in pixels
            element.style.left = position + "px"
        }
    }
}
```

Snippet animate2.html

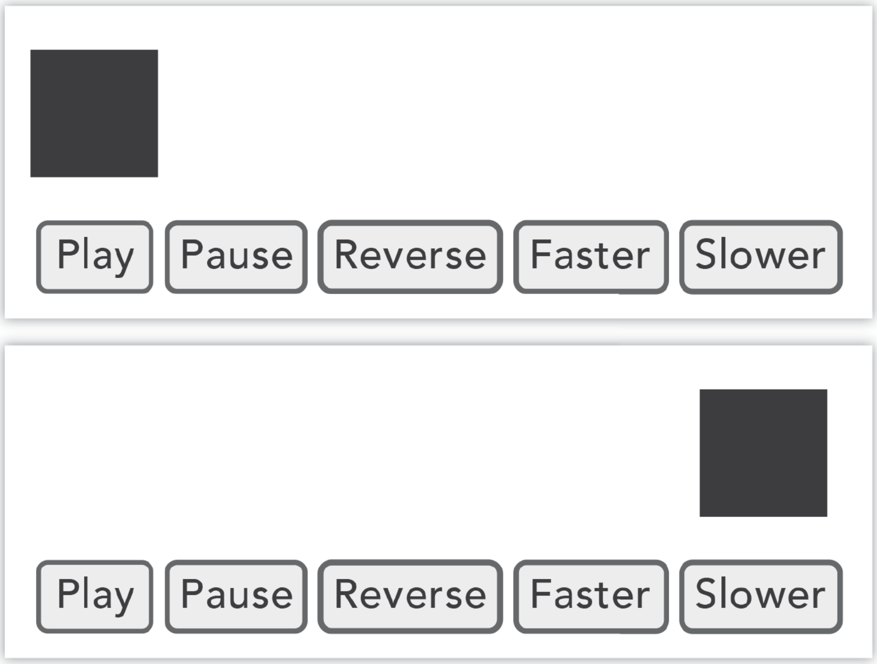
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Web Animations API</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <style>
        #box {
            width: 50px;
            height: 50px;
            background-color: red;
            position: relative;
        }
    </style>
</head>
<body>

<div id="box"></div>

<br>

<button id="play">Play</button>
<button id="pause">Pause</button>
<button id="reverse">Reverse</button>
<button id="incRate">Faster</button>
```

```
<button id="decRate">Slower</button>

<script src="animate2.js"></script>

</body>
</html>
```

Snippet animate2.js

```
// Query the element
let element = document.getElementById('box')

// Define animation
let animation = element.animate([
    { transform: 'translateX(0px)' },
    { transform: 'translateX(240px)' }
],
    { duration: 2500,
      iterations: Infinity,
      direction: 'alternate'
    }
)

// Bind event buttons
document.getElementById("play").addEventListener("click", playAnimation)
document.getElementById("pause").addEventListener("click", pauseAnimation)
document.getElementById("reverse").addEventListener("click", reverseAnimation)
document.getElementById("incRate").addEventListener("click", incPlaybackRate)
document.getElementById("decRate").addEventListener("click", decPlaybackRate)

// Pause the animation at the beginning
animation.pause()

// Play the animation
function playAnimation() {
    console.log("Playing animation.")
    animation.play()
}

// Pause the animation
function pauseAnimation() {
    console.log("Animation paused.")
    animation.pause()
}

// Reverse the animation
function reverseAnimation() {
    console.log("Reversing animation.")
    animation.reverse()
}

// Increase playback rate by 0.5x
function incPlaybackRate() {
    animation.playbackRate += 0.5
    console.log(`Playback rate increased to ${animation.playbackRate}`)
}

// Decrease playback rate by 0.5x
function decPlaybackRate() {
    if (animation.playbackRate > 0) {
        animation.playbackRate -= 0.5
        console.log(`Playback rate decreased to ${animation.playbackRate}`)
    }
}
```

Figure 31



The box at the start and furthest point of the animation.

Code Line 13-1:

```
// Define animation
let animation = element.animate([
    { transform: 'translateX(0px)' },
    { transform: 'translateX(240px)' }
],
    { duration: 2500,
        iterations: Infinity,
        direction: 'alternate'
    }
)
```

Table 7

ARGUMENT	DESCRIPTION	EXAMPLE
`duration`	Set duration of an animation cycle (in milliseconds).	`{ duration: 1500 }`
`delay`	Specify delay before animation starts (in milliseconds).	`{ delay: 2500 }`
`direction`	The direction of the animation. Common values include `normal`, `reverse`, `alternate`, `and alternate-reverse`.	`{ direction: "reverse" }`
`easing`	Define the timing of the animation. For example, the value `ease-in-out` eases into, and out of, the animation by starting slow, moving faster, then slowing down at the end. Other common values: `ease-in`, `ease-out`, `linear`.	`{ easing: "ease-in" }`
`endDelay`	Delay after animation ends (in milliseconds).	`{ endDelay: 750 }`
`fill`	Specify how values are applied before and after animation. Can be `none`, `forwards`, `backwards`, or `both`.	`{ fill: "both" }`
`iterations`	Number of times the animation will play before finishing. The `Infinity` value can also be specified.	`{ iterations: 100 }`
`keyframes`	Defines states at specified points throughout the animation. This could be start and end states (in the example, opacity), or you can define multiple states.	`}, { opacity: 1 }]`

Code Line 13-2:

```
@media (prefers-reduced-motion: reduce) {
    /* reduce or, ideally, eliminate animations */
}
```

CHAPTER 14
JavaScript Junk Drawer

Code Line 14-1:

```
<button onclick=”history.back()”>Back</button>
```

Table 8

PROPERTY	DESCRIPTION	RETURN VALUE FROM THE URL https://www.clydebankmedia.com/javascript/?q=1
`href`	Returns the full URL, including query string.	`https://www.clydebankmedia. com/javascript/?q=1`
This is useful when you need to display the full link on the page, or provide the link to an external service, like a social media widget or something similar.		
`host`	Returns the server hostname portion of the URL, including the port number, if specified.	`www.clydebankmedia.com`
If you need to provide just the domain name to an external service, this property comes in handy.		
`hostname`	Returns the server hostname portion of the URL.	`www.clydebankmedia.com`
This is usually useful in providing the full hostname to an external service, or for providing a way to set a bookmark for the whole site.		
`pathname`	The path/file portion of the URL without the query string.	`/javascript/`
Using this relative pathname makes it easy to preserve a link across sites that share the same URL structure but have different domains. For example, a development or test website will have a different host (e.g., dev.example.com), but the path to the content will be the same.		
`protocol`	Returns the protocol used (e.g., `http`, `https`, etc.)	`https:`
This is handy when determining whether the page is being served over SSL.		
`search`	Returns the query string, or GET variable string (after the path/file)	`?q=1`
This is useful when applying custom logic to parsing through the parameters supplied to the page in the URL.		

Code Line 14-2:

```
console.log(navigator.userAgent)
```

Code Line 14-3:

```
'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/112.0.0.0 Safari/537.36'
```

Code Line 14-4:

```
if (navigator.userAgent.includes("Edg")) {
    // Run this code if using Microsoft Edge
}
```

Table 9

PROPERTY	DESCRIPTION	RETURN VALUE (MacBook M1 Pro)	RETURN VALUE (AMD Windows PC)
`language`	Returns the current system language.	`en-US`	`en-US`
`platform`	Returns the hardware platform.	`MacIntel`	`Win32`
`product`	Returns the browser engine (see caution below).	`Gecko`	`Gecko`
`userAgent`	Returns the full user agent string.	`Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/ 537.36`	`Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/ 537.36`

Code Line 14-5:

```
let a = 31415926535897932384626433n
let b = BigInt(31415926535897932384626433)
```

Code Line 14-6:

```
console.log(a * b)
```

Code Line 14-7:

```
986960440108935807202011224049440862260895693144064n
```

Code Line 14-8:

```
let exampleSet = new Set()
```

Code Line 14-9:

```
let exampleSet = new Set([“Apples”, “Oranges”, “Pears”])
```

Code Line 14-10:

```
console.log(exampleSet)
```

Code Line 14-11:

```
Set(3) {‘Apples’, ‘Oranges’, ‘Pears’}
```

Code Line 14-12:

```
let exampleSet = new Set([“Apples”, “Oranges”, “Pears”])
exampleSet.add(“Grapes”)
console.log(exampleSet)
```

Code Line 14-13:

```
Set(4) {‘Apples’, ‘Oranges’, ‘Pears’, ‘Grapes’}
```

Code Line 14-14:

```
let exampleSet = new Set([“Apples”, “Oranges”, “Pears”])
exampleSet.delete(“Apples”)
console.log(exampleSet)
```

Code Line 14-15:

```
Set(2) {‘Oranges’, ‘Pears’}
```

Code Line 14-16:

```
if (exampleSet.has(“Pears”)) {
    // Do something if Pears exists in the set
}
```

Code Line 14-17:

```
let exampleSet = new Set([“Apples”, “Oranges”, “Pears”])
for (const item of exampleSet) {
    console.log(item)
}
```

Code Line 14-18:

```
Apples
Oranges
Pears
```

Code Line 14-19:

```
let exampleSet = new Set([“Apples”, “Oranges”, “Pears”])
console.log(exampleSet.size)
```

Code Line 14-20:

```
let numbers = [2, 4, 8, 16, 32, 64]

let doubles = numbers.map(number => number * 2)

console.log(doubles)
```

Code Line 14-21:

```
[4, 8, 16, 32, 64, 128]
```

Code Line 14-22:

```
let doubles = numbers.map(function(number) {
    return number * 2
})
```

Code Line 14-23:

```
let numbers = [2, 4, 8, 16, 32, 64]

function doubler(number) {
    return number * 2
}

let doubles = numbers.map(doubler)

console.log(doubles)
```

Code Line 14-24:

```
let emailList = new Map()

emailList.set("John Smith", "jsmith@example.com")
emailList.set("April Smith", "asmith@example.com")
emailList.set("Robert Smith", "rsmith@example.com")
```

Code Line 14-25:

```
console.log(emailList.get("John Smith"))
```

Code Line 14-26:

```
jsmith@example.com
```

Code Line 14-27:

```
console.log(emailList.size)
```

Code Line 14-28:

```
emailList.delete(“John Smith”)
```

Code Line 14-29:

```
for (let [key, value] of emailList.entries()) {
    console.log(key, value)
}
```

Code Line 14-30:

```
April Smith asmith@domain.com
Robert Smith rsmith@domain.com
```

Code Line 14-31:

```
emailList.forEach((value, key) => {
    console.log(key, value)
})
```

Code Line 14-32:

```
emailList.clear()
```

Code Line 14-33:

```
‘use strict’
```

Code Line 14-34:

```
const myCode = “console.log(‘Hello, World!’)”
eval(myCode)
```

Snippet 14-01.js

```
// Define our regular expression
// This will match text that contains the word gray or grey
// and ignore case
const searchPattern = /(gray|grey)/i
```

```
// The string to search
let myString = “The quick gray fox jumped over the lazy red dog.”
// Perform the search
if (searchPattern.test(myString)) {
// This code runs if myString contains gray or grey,
// regardless of case
}
```

Code Line 14-35:

```
const searchPattern = /^(.{5})/
let myString = “Hello, World!”

console.log(searchPattern.exec(myString)[0])
```

Code Line 14-36:

```
Hello
```

Code Line 14-37:

```
const searchPattern = /(.{6})$/
let myString = “Hello, World!”

console.log(searchPattern.exec(myString)[0])
```

Code Line 14-38:

```
World!
```

Code Line 14-39:

```
const validCardPattern = /^[3456]\d{14,15}$/
```

Snippet 14-02.js

```
const validCardPattern = /^[3456]\d{14,15}$/

let cards = [
    “4123456789012345”,
    “9630123484921205”,
    “5155625609411234”,
    “345141116111987”,
    “6161515141413”
]

cards.forEach(function (card) {
    if (validCardPattern.test(card)) {
        console.log(card + “ is valid.”)
    }
})
```

CHAPTER 15
Saving Time with jQuery

Snippet jquery-01.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
        <script src="jquery.js"></script>
        <title>Learning About jQuery</title>
    </head>
    <body>
        <div id="content">
            <p id="greeting">Hello, World!</p>
        </div>
    </body>
</html>
```

Code Line 15-1:

```
<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
```

Code Line 15-2:

```
<script src="jquery-3.6.4.min.js"></script>
```

Code Line 15-3:

```
$("#content").hide()
```

Code Line 15-4:

```
// The JavaScript way
document.getElementById("content").style.display = "none"

// The jQuery way
$("#content").hide()
```

Code Line 15-5:

```
$(“div”).hide()
```

Code Line 15-6:

```
function reliesOnPageLoaded() {
    // Code to run when DOM is loaded
}

document.addEventListener(“DOMContentLoaded”, reliesOnPageLoaded)
```

Code Line 15-7:

```
$(document).ready(function() {
    // Code to run when DOM is loaded
}
```

Code Line 15-8:

```
$(function() {
    // Code to run when DOM is loaded
}
```

Code Line 15-9:

```
$(“#greeting”).on(“click”, function() {
    $(this).hide()
})
```

Code Line 15-10:

```
$(“p”).first()
```

Code Line 15-11:

```
$(“div p:first-child”)
```

Code Line 15-12:

```
$(“tr:even”).addClass(“rowWhiteBg”)
$(“tr:odd”).addClass(“rowGrayBg”)
```

Snippet jquery.html

```
<!DOCTYPE html>
<html lang=”en”>
    <head>
        <meta charset=”UTF-8”>
        <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
        <meta http-equiv=”X-UA-Compatible” content=”ie=edge”>
        <script src=”https://code.jquery.com/jquery-3.6.4.min.js”></script>
        <script src=”jquery.js”></script>
        <title>Learning About jQuery</title>
    </head>
    <body>
        <div id=”content”>
            <p id=”greeting”>Hello, World!</p>
        </div>
        <div>
            <input type=”text” id=”firstName” name=”firstName”
                placeholder=”Please enter your first name” size=”35”>
            <button id=”editGreeting”>Say Hello!</button>
        </div>
        </body>
</html>
```

Snippet jquery.js

```
// Only bind after element exists
$(function() {
    // Bind to the click event of the editGreeting button
    $(“#editGreeting”).on(“click”, function() {
        // Get the firstName value from the input box
        let firstName = $(“#firstName”).val()
        // Adjust the greeting text in the p tag with the name
        $(“#greeting”).text(`Hello, ${firstName}!`)
    })
})
```

Code Line 15-13:

```
$(“#content”).text()
$(“#content”).html()
```

Code Line 15-14:

```
'\n             Hello, World!\n         '
'\n             <p id="greeting">Hello, World!</p>\n         '
```

Code Line 15-15:

```
$("#greeting").text()
```

Code Line 15-16:

```
'Hello, World!'
```

Code Line 15-17:

```
// Enlarge the firstName input box to 100
$("#firstName").attr("size", 100)
```

Code Line 15-18:

```
$("#greeting").addClass("boldText")
```

Code Line 15-19:

```
$("#greeting").removeClass("boldText")
```

Code Line 15-20:

```
$("#greeting").css("font-weight", "bold")
```

Code Line 15-21:

```
$("#editGreeting").width(300)
$("#editGreeting").height(100)
```

Code Line 15-22:

```
$("#greeting").fadeOut()
$("#greeting").fadeIn()
```

Code Line 15-23:

```
$("#greeting").fadeOut(5000)
$("#greeting").fadeIn(5000)
```

Snippet jquery-animate.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
        <script src="jquery-animate.js"></script>
        <title>Learning About jQuery</title>
        <style type="text/css">
            #purpleBox {
                position: relative;
                width: 100px;
                height: 100px;
                background-color: purple;
            }
        </style>
    </head>
    <body>
        <div id="content">
            <p id="greeting">Hello, World!</p>
        </div>
        <div>
            <input type="text" id="firstName" name="firstName"
                placeholder="Please enter your first name" size="35">
            <button id="editGreeting">Say Hello and Move the Box!</button>
        </div>
        <br>
        <div id="purpleBox">
        </div>
    </body>
</html>
```

Snippet jquery-animate.js

```
// Only bind after element exists
$(function() {
    // Bind to the click event of the editGreeting button
    $("#editGreeting").on("click", function() {
        // Get the firstName value from the input box
        let firstName = $("#firstName").val()
        // Adjust the greeting text in the p tag with the name
        $("#greeting").text(`Hello, ${firstName}!`)
        // Move the purple box to the right by adding 340px to
        // its left position over a period of 3 seconds
        $("#purpleBox").animate({left: "+=340px"}, 3000)
    })
})
```

Snippet jquery-animate.js

```
// Only bind after element exists
$(function() {
    // Bind to the click event of the editGreeting button
    $("#editGreeting").on("click", function() {
        // Get the firstName value from the input box
        let firstName = $("#firstName").val()
        // Adjust the greeting text in the p tag with the name
        $("#greeting").text(`Hello, ${firstName}!`)
        // Move the purple box to the right by:
        //   1) setting its opacity to 0.25 (25%)
        //   2) adding 340px to its left position
        //   3) setting the final width to 0px
        //   4) setting the final height to 0px
        //   5) animating the change over 3 seconds
        $("#purpleBox").animate({
            opacity: "0.0",
            left: "+=340px",
            height: "0px",
            width: "0px"
        }, 3000)
    })
})
```

CHAPTER 16
Node.js and AJAX

Snippet 16-01.js

```
// The URL to request
const url = "https://javascript.clydebankmedia.com/hello"

fetch(url).then(response => {
    if (response.ok) {
        // Request was successful, now the response object
        // is populated. The promises response.json() and
        // response.text() are available.
        const textPromise = response.text()
        const jsonPromise = response.json()
        return Promise.all([textPromise, jsonPromise])
    } else {
        // The request failed.
        throw new Error("The request encountered an error.")
    }
}).then(([text, json]) => {
    // text and json now have actual data and are ready for use
    console.log("Text response: ", text)
    console.log("JSON response: ", json)
}).catch(error => console.error("An error occurred: ", error))
```

Snippet 16-02.js

```
// The URL to POST to
const url = "https://www.example.com"
const requestMethod = "POST"

// The JSON object to POST to the web server
const postData = {
    firstName: "Robert",
    email: "support@clydebankmedia.com"
}

// Make the POST request and return a promise
fetch(url, {
    method: requestMethod,
    headers: {
        "Content-Type": "application/json"
    },
    body: JSON.stringify(postData)
}).then(response => response.json())
```

```
.then(result => console.log('POST was a success: ${result}'))
.catch(error => console.error('An error occurred: ${error}'))
```

Snippet 16-03.js

```
// Define the URL
const url = "https://www.example.com"

// Make the request, passing the URL
// and function to process the resulting data
$.get(url, function(data) {
    // Successful, display data retrieved
    console.log(data)
}).fail(function(jqXHR, textStatus, errorThrown) {
    // An error occurred
    console.log(`An error occurred: ${errorThrown}`)
})
```

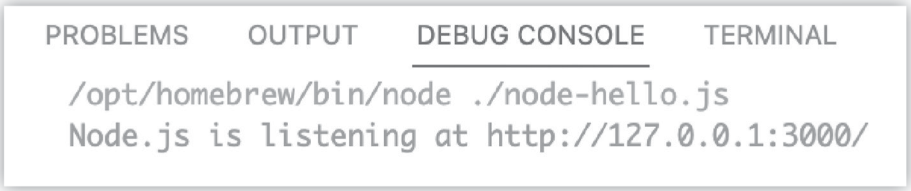
Snippet 16-04.js

```
// Define the URL
const url = "https://www.myexamplewebserver.com"

// Example data to pass
const postData = {
    firstName: "Robert",
    email: "support@clydebankmedia.com"
}

// Make the request, passing the URL
// and function to process the resulting data
$.post(url, postData, function(data) {
    // Successful, display response retrieved
    console.log(data)
}).fail(function(jqXHR, status, error) {
    // An error occurred
    console.log(`An error occurred: ${errorThrown}`)
})
```

Code Line 16-1:

```
const http = require("http")

const address = "127.0.0.1"
const port = 3000
const server = http.createServer((req, res) =>
    res.statusCode = 200
    res.setHeader("Content-Type", "text/plain")
    res.end("Hello, World!")
})

server.listen(port, address, () => {
    console.log(`Node.js is listening at http://${address}:${port}/`)
})
```

Figure 32



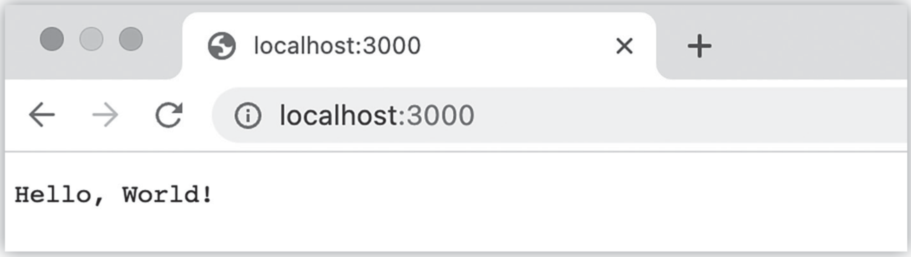
```
/opt/homebrew/bin/node ./node-hello.js
Node.js is listening at http://127.0.0.1:3000/
```

The debug console showing "Node.js is listening."

Code Line 16-2:

```
http://127.0.0.1:3000
```

Figure 33



```
Hello, World!
```

Our "Hello, World!" message served from Node.js to our web browser.

Snippet node-json1.js

```
const http = require("http")

const hostname = "127.0.0.1"
const port = 3000
const server = http.createServer((req, res) => {
    // Create a sample JSON object
    const exampleJSON = {
        "firstName": "Robert",
    }

    // Set the status code to 200, signifying success to the browser
    res.statusCode = 200
    // Set the mimetype to application/json, so the browser
    // knows how to handle the response
    res.setHeader("Content-Type", "application/json")

    // Set CORS headers to allow any page to send requests
    // This is only for development, don't do this in production!
    res.setHeader("Access-Control-Allow-Origin", "*")
    res.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE")
    res.setHeader("Access-Control-Allow-Headers", "Content-Type")

    // Send the JSON object as a string
    res.end(JSON.stringify(exampleJSON))
})

server.listen(port, hostname, () => {
    console.log(`Node.js is listening at http://${hostname}:${port}/`)
})
```

Snippet hello-json1.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Hello!</title>
    </head>
    <body>
        <div id="content">
            <p id="message">Hello </p>
        </div>
        <script>
            const url = "http://localhost:3000"
            fetch(url).then(response => response.json())
            .then(data => {
                const message = document.getElementById("message")
                message.innerHTML += data.firstName + "!"
            }
            )
        </script>
    </body>
</html>
```

Code Line 16-3:

```
Hello Robert
```

Snippet node-get.js

```
const http = require("http")
const url = require("url")

const hostname = "127.0.0.1"
const port = 3000

const server = http.createServer((req, res) => {
    // Parse the URL and obtain the GET vars, also
    // known as the query string
    const parsedUrl = url.parse(req.url, true)
    const getVars = parsedUrl.query

    res.statusCode = 200
    res.setHeader("Content-Type", "application/json")

    // Display the getVars in the browser
    res.end(JSON.stringify(getVars))
})

// Start the server
server.listen(port, hostname, () => {
    console.log(`Node.js is listening at http://${hostname}:${port}/`)
})
```

Code Line 16-4:

```
http://127.0.0.1:3000/?firstName=Robert&lastName=Oliver
```

Code Line 16-5:

```
http://127.0.0.1:3000/?firstName=Robert&lastName=Oliver
```

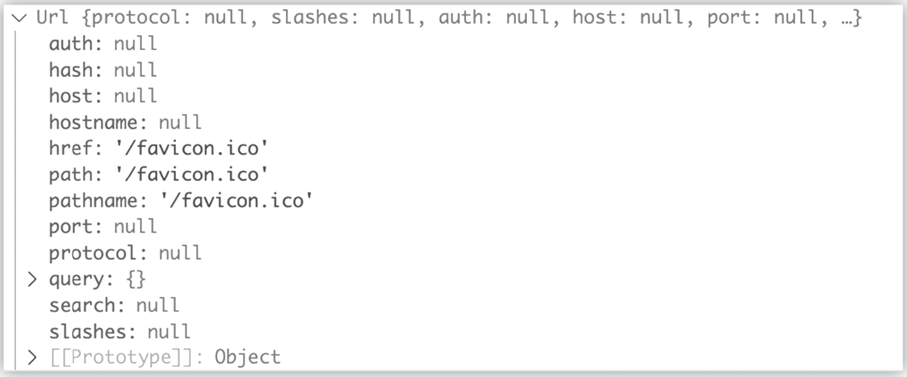
Code Line 16-6:

```
{"firstName":"Robert","lastName":"Oliver"}
```

Code Line 16-7:

```
console.log(parsedUrl)
```

Figure 34



```
⌄ Url {protocol: null, slashes: null, auth: null, host: null, port: null, …}
    auth: null
    hash: null
    host: null
    hostname: null
    href: '/favicon.ico'
    path: '/favicon.ico'
    pathname: '/favicon.ico'
    port: null
    protocol: null
  > query: {}
    search: null
    slashes: null
  > [[Prototype]]: Object
```

Properties in the `parsedUrl` object, displayed in the Visual Studio Code console.

Snippet 16-05.js:

```
const http = require("http")
const hostname = "127.0.0.1"
const port = 3000

const server = http.createServer((req, res) => {
    if (req.method === "POST") {
        let body = ""
        // Collect data chunks, adding each to the body string
        req.on("data", chunk => {
            body += chunk.toString()
        })

        // When end event fires, log the string and end the request
        req.on("end", () => {
            console.log(body)
res.end("OK")
        })
    }
})
```

```
// Start the server
server.listen(port, hostname, () => {
    console.log(`Node.js is listening at http://${hostname}:${port}/`)
})
```

Snippet 16-06.js:

```
const http = require("http")
const url = require("url")

const hostname = "127.0.0.1"
const port = 3000

const server = http.createServer((req, res) => {
    const parsedUrl = url.parse(req.url, true)

    switch(parsedUrl.pathname) {
        case "/":
            // Display the Home page
            res.writeHead(200, {"Content-Type": "text/html"})
            res.end("Home page contents.")
            break
        case "/about":
            // Display the About Us page
            res.writeHead(200, {"Content-Type": "text/html"})
            res.end("About Us page contents.")
            break
        case "/contact":
            // Display the Contact page
            res.writeHead(200, {"Content-Type": "text/html"})
            res.end("Contact Us page contents.")
            break
        default:
            // Display a 404 page (Page Not Found)
            res.writeHead(404, {"Content-Type": "text/html"})
            res.end("404 - Page Not Found.")
    }
})

// Start the server
server.listen(port, hostname, () => {
    console.log(`Node.js is listening at http://${hostname}:${port}/`)
})
```

Code Line 16-8:

```
npm install sqlite3
```

Code Line 16-9:

```
npm install sqlite3
```

Code Line 16-10:

```
sqlite3
```

Code Line 16-11:

```
apt install sqlite3
```

Code Line 16-12:

```
yum install sqlite
```

Code Line 16-13:

```
pacman -S sqlite
```

Code Line 16-14:

```
npm install sqlite3
```

Snippet stunes.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>sTunes Tracks</title>
    </head>
    <body>
        <div id="content">
            <table id="tracks">
                <th>
                    <td>ID</td>
                    <td>Name</td>
                    <td>Composer</td>
                    <td>Price</td>
                </th>
            </table>
        </div>
        <script>
            const url = "http://localhost:3000"
            fetch(url).then(response => response.json())
            .then(data => {
                const tracksTable = document.getElementById("tracks")
                // Iterate over the results
                data.forEach(row => {
                    console.log(row)
                    // Create the needed DOM elements
                    let resultRow = document.createElement("tr")
                    let cellId = document.createElement("td")
                    let cellName = document.createElement("td")
                    let cellPrice = document.createElement("td")

                    // Put the data in the columns
                    cellId.innerText = row.TrackId
                    cellName.innerText = row.Name
                    cellPrice.innerText = row.UnitPrice

                    // Append columns to the row
                    resultRow.appendChild(cellId)
                    resultRow.appendChild(cellName)
                    resultRow.appendChild(cellPrice)

                    // Append row to the table
                    tracksTable.appendChild(resultRow)
                })
            })
        </script>
    </body>
</html>
```

Snippet node-stunes.js

```
const http = require("http")
const sqlite3 = require("sqlite3")

const address = "127.0.0.1"
const port = 3000

// SQL query to use
const sqlQuery = "SELECT * FROM tracks;"

const server = http.createServer((req, res) => {
    res.statusCode = 200
    res.setHeader("Content-Type", "application/json")

// Set CORS headers to allow any page to send requests
// This is only for development, don't do this in production!
res.setHeader("Access-Control-Allow-Origin", "*")
res.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE")
res.setHeader("Access-Control-Allow-Headers", "Content-Type")


    // Connect to the sTunes database
    let db = new sqlite3.Database("./sTunes.db", sqlite3.OPEN_READONLY, (err) => {
        if (err) {
            console.error('Database connection error: ${err.message}')
        }
        console.log("Connection established.")
    })

    // Create an array to accumulate rows of results
    let results = []

    // Query the database
    db.serialize(() => {
        // Iterate through each row in the table
        db.each(sqlQuery, (err, row) => {
            if (err) {
                // If an error occurs, log to console
                console.error(err.message)
                } else {
                // Add the row (track) to the array
                    results.push(row)
                }
            }, (err, count) => {
            // This code is executed when all rows have been retrieved
            // We don't use it, but the count variable has the total
            // number of rows retrieved from the result.
            if (err) {
                // If an error occurred, log it in the console
                console.error(err.message)
            } else {
                // Send results to the web browser
                res.end(JSON.stringify(results))
            }
        })
    })
})

server.listen(port, address, () => {
    console.log(`Node.js is listening at http://${address}:${port}/`)
})
```

Code Line 16-15:

```
SELECT * FROM tracks;
```

Snippet 16-07.js

```
// Add item to cart
function addToCart(itemName, itemPrice, itemQnty) {
    // Create an object named item with details of product
    const item = {
        name: itemName,
        price: itemPrice,
        quantity: itemQnty
    }

    // Make POST request to the server
    fetch("/cart/add", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(item)
    })
    .then(response => response.json())
    .then(data => console.log(data))
    .catch((error) => {
        console.error(`An error occurred: ${error}`)
    })
}
```

Snippet 16-08.js

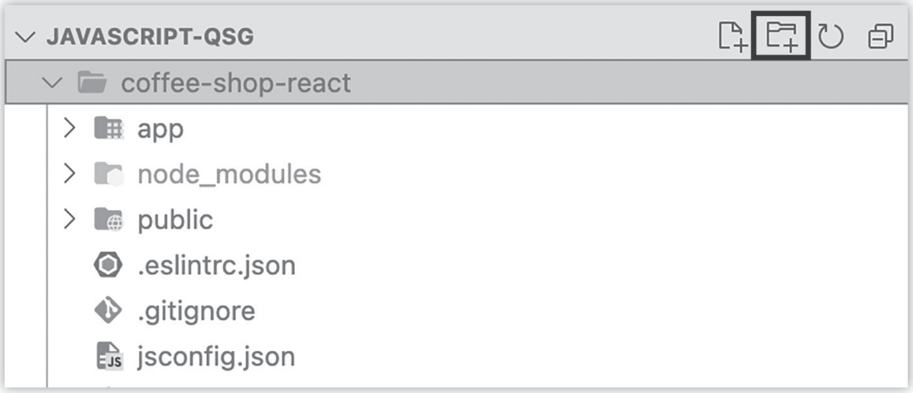
```
function getCart() {
    // Make the HTTP request to the server via fetch
    fetch("/cart", { method: "GET" })
        .then(response => response.json())
        .then(data => console.log(data))
        .catch((error) => {
            console.error("An error occurred: ", error)
        })
}
```

Snippet 16-09.js

```
const http = require("http")
const url = require("url")

const hostname = "127.0.0.1"
const port = 3000

// Initialize the cart
let cart = {}

const server = http.createServer((req, res) => {
    const parsedUrl = url.parse(req.url, true)

    switch(parsedUrl.pathname) {
        case "/":
            // Display the Home page
            res.writeHead(200, {"Content-Type": "text/html"})
            res.end("Home page contents.")
            break
        case "/about":
            // Display the About Us page
            res.writeHead(200, {"Content-Type": "text/html"})
            res.end("About Us page contents.")
            break
        case "/contact":
            // Display the Contact page
            res.writeHead(200, {"Content-Type": "text/html"})
            res.end("Contact Us page contents.")
            break
        case "/cart":
            // GET will return the cart contents
            if (req.method === "GET") {
                res.writeHead(200, {"Content-Type": "application/json"})
                res.end(JSON.stringify(cart))
            // POST will add an item to the cart
            } else if (req.method === "POST") {
                let body = ""
                req.on("data", chunk => {
                    body += chunk.toString()
                })
                req.on("end", () => {
                    const item = JSON.parse(body)

                    // If a cart item with the same name already exists,
                    // increment quantity. If not, add item to cart
                    if (cart[item.name]) {
                        cart[item.name].quantity += item.quantity
                    } else {
                        cart[item.name] = item
                    }

                    res.writeHead(200, {"Content-Type": "application/json"})
                    res.end(JSON.stringify({success: true}))
                })
            } else {
                res.writeHead(405, {"Content-Type": "text/plain"})
                res.end("Method not allowed")
            }
            break
            default:
            // Display a 404 page (Page Not Found)
            res.writeHead(404, {"Content-Type": "text/html"})
            res.end("404 - Page Not Found.")
        }
})

// Start the server
server.listen(port, hostname, () => {
    console.log(`Node.js is listening at http://${hostname}:${port}/`)
})
```

CHAPTER 17
Building Web Pages with React

Figure 35

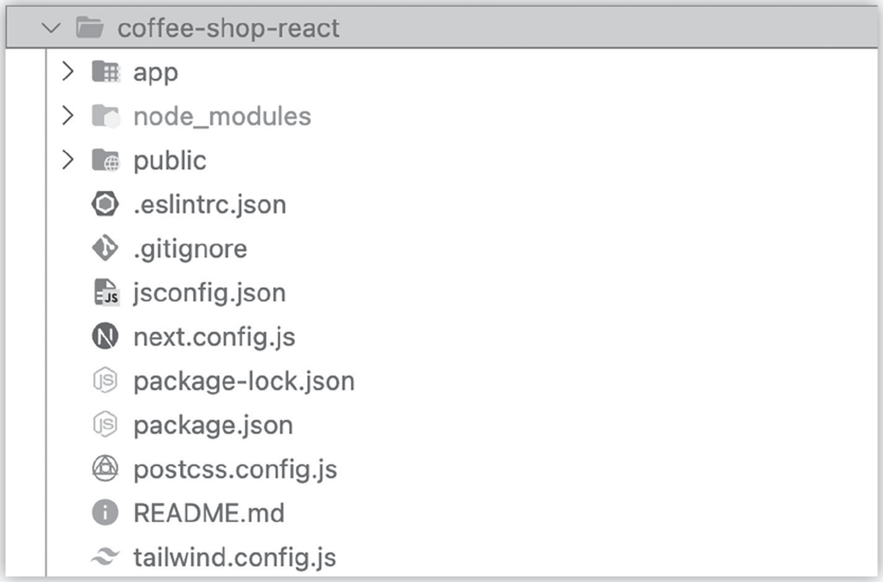


The new folder icon circled in Visual Studio Code's Explorer pane.
The *New Folder...* tooltip is shown when you hover over the icon.

Code Line 17-1:

```
npx create-next-app@latest .
```

Figure 36



The resulting files from the create-next-app wizard.

Table 10

FILE / FOLDER	PURPOSE
`.eslintrc.json`	Configuration file for ESLint, which is an analysis tool that helps find bugs and design issues within JavaScript.
`.gitignore`	A file created to instruct Git, the source code version control system, which files should not be included in a repository and should remain only on your computer.
`next.config.js`	The configuration file for the Next.js web framework.
`package-lock.json`	A file automatically generated by npm that sets versions of dependencies to ensure the package will install and run correctly on another computer.
`package.json`	A configuration file that details the dependencies for the project.
`postcss.config.js`	Configuration file for PostCSS, which facilitates the transformation of styles with various JavaScript plugins, preventing conflicts.
`README.md`	A Markdown file that contains information about the project. This is especially helpful for other developers.
`tailwind.config.js`	The Tailwind CSS configuration file, a CSS framework that makes it easy to quickly add helpful styles to your HTML.
`app`	The folder containing the React JavaScript files.
`node_modules`	The dependency files for the project are stored within this folder.
`public`	This folder contains files that will be served directly to the browser rather than be processed as part of the React app.

Table 11

FILE / FOLDER	PURPOSE
`favicon.ico`	The icon displayed for the site in bookmarks. You can replace this with your website's logo in `ico` format.
`globals.css`	This CSS file is meant to house CSS styles that will apply to your entire React site. Tailwind CSS will also include some boilerplate code here to import its styles and utilities.
`layout.js`	JavaScript code here generally applies to the entire site, providing navigation elements, header/footer components, and other site-wide features.
`page.js`	The primary landing page of the site.

Code Line 17-2:

```
import React from “react”

function Greeting() {
    return (
        <div>
            <p>Hello, World!</p>
        </div>
    )
}
```

Code Line 17-3:

```
import React from “react”

function Greeting(props) {
    return (
        <div>
            <p>Hello, {props.firstName}!</p>
        </div>
    )
}
```

Code Line 17-4:

```
<Greeting />
```

Code Line 17-5:

```
<Greeting firstName=”Robert” />
```

Snippet 17-01.js

```
import React from “react”

const AlertButton = ({ alertMessage }) => {
    const handleClick = () => {
        alert(alertMessage)
    }

    return (
        <button onClick={handleClick}>
            Click Me
        </button>
    )
}

const Greeting = () => {
    return (
        <div>
            <AlertButton alertMessage="Hello, World!" />
        </div>
    )
}
```

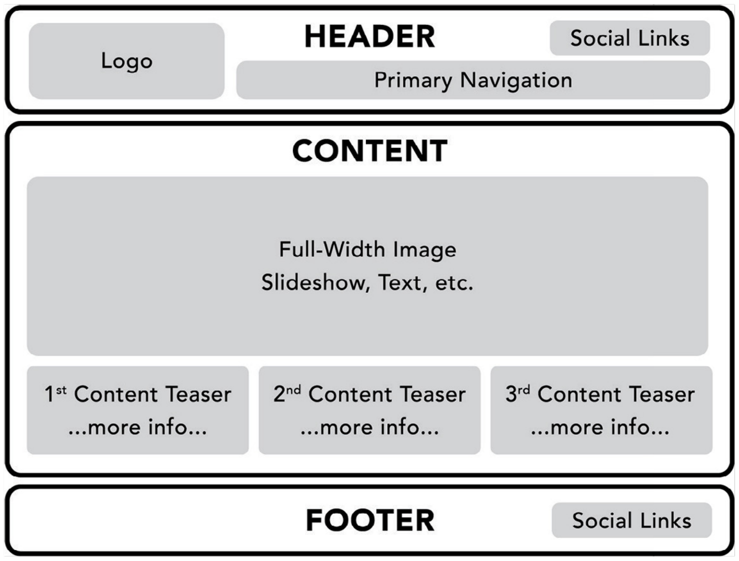
Code Line 17-6:

```
const Greeting = () => {
    return (
        <div>
            <p>Hello, World!</p>
        </div>
    )
}

export default Greeting
```

Code Line 17-7:

```
import Greeting from "../components/Greeting.js"
```

Figure 37


HEADER
Logo
Social Links
Primary Navigation
CONTENT
Full-Width Image
Slideshow, Text, etc.
1st Content Teaser ...more info...
2nd Content Teaser ...more info...
3rd Content Teaser ...more info...
FOOTER
Social Links

A sample web page layout.

Snippet 17-02.js

```
import Image from “next/image”

const Logo = () => (
    <Image src=”logo.png” alt=”Our Logo” />
)

const PrimaryNavigation = ({ links }) => (
    <div>
        {links.map((link, index) => (
            <a href={link.page}>{link.name}</a>
        ))}
    </div>
)

const SocialLinks = ({ socialAccounts }) => (
    <div>
        {socialAccounts.map((account, index) => (
            <a href={account.link}>{account.site}</a>
        ))}
    </div>
)

const FullWidthSlider = ({ slides }) => (
    <div>
        {slides.map((slide, index) => (
            <a href={slide.link}><img src={slide.image} alt={`Slide ${index}`} /></a>
        ))}
    </div>
)

const Teaser = ({ thumbnail, text }) => (
    <div>
        <img src={thumbnail} alt=”thumbnail” />
        <div>
            {text}
        </div>
    </div>
)

const Header = ({ links, socialAccounts }) => (
    <div>
        <Logo />
        <PrimaryNavigation links={links} />
        <SocialLinks socialAccounts={socialAccounts} />
    </div>
)

const Content = ({ slides, teaserThumbnails, teaserTexts }) => (
    <div>
        <FullWidthSlider slides={slides} />
        <Teaser
            thumbnail={teaserThumbnails[0]}
            text={teaserTexts[0]}
        />
    </div>
)

const Footer = ({ socialAccounts }) => (
    <SocialLinks socialAccounts={socialAccounts} />
)
```

Code Line 17-8:

```
const inventory = [
    { productName: "Coffee", price: "3.95", inStock: true },
    { category: "Espresso", price: "4.49", inStock: true },
    { category: "Cappuccino", price: "4.95", inStock: false }
]
```

Code Line 17-9:

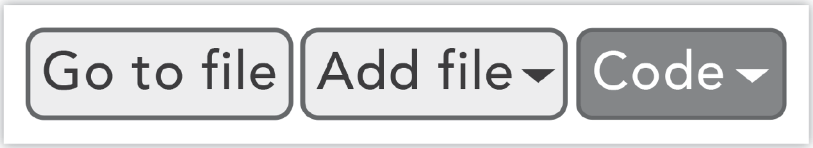
```
const InventoryList = ({ inventory }) => {
    return (
        <div>
            {inventory.map((item, index) => (
                <div id={index}>
                    <h2>{item.productName}</h2>
                    <p>Price: ${item.price}</p>
                    <p>In Stock: {item.inStock ? 'Yes' : 'No'}</p>
                </div>))}
        </div>
    )
}
```

CHAPTER 18
Managing Your Code with Git

Code Line 18-1:

```
git version
```

Figure 38



The Code sync button in the GitHub interface.

Code Line 18-2:

```
cd ~/Source
```

Code Line 18-3:

```
cd %HOMEPATH%\Source
```

Code Line 18-4:

```
git clone https://github.com/clydebankmedia/JavaScript-CoffeeShopWebsite
```

Code Line 18-5:

```
git add filename.js
```

Code Line 18-6:

```
git add .
```

Code Line 18-7:

```
git status
```

Code Line 18-8:

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Code Line 18-9:

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
    (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
    modified:   main.js

no changes added to commit (use "git add" and/or "git commit -a")
```

Code Line 18-10:

```
git commit -m "Short message describing your changes."
```

Code Line 18-11:

```
git commit -a -m "A note about your changes."
```

Code Line 18-12:

```
git push origin main
```

Code Line 18-13:

```
git checkout -b branch_name
```

Code Line 18-14:

```
git push origin branch_name
```

Code Line 18-15:

```
git checkout branch_name
```

Code Line 18-16:

```
git diff
```

Code Line 18-17:

```
git log
```

Code Line 18-18:

```
commit 6dd258b5273bc4a13eee201fd6304efc61baef5a (HEAD -> main, origin/main, origin/
HEAD)
Author: Robert W. Oliver II <118407+rwoliver2@users.noreply.github.com>
Date:   Sun Jul 3 23:11:07 2022 -0500

    Update README.md

commit a2af786f19023bd2ac2d023ce8b2f2ed664f733b
Author: Robert W. Oliver II <118407+rwoliver2@users.noreply.github.com>
Date:   Sun Jul 3 23:07:07 2022 -0500

    Initial commit.
```

Appendix I

HTML/CSS Quick Reference

IMPORTANT

This appendix is meant to serve as a refresher and quick reference for HTML and CSS. As such, it will read more as a reference and less as a guide. If you aren't familiar with HTML/CSS, we recommend you check out *HTML & CSS QuickStart Guide* by David DuRocher, available at www.quickstartguides.com

Understanding HTML Structure

HyperText Markup Language (HTML) is a structured markup language. While you can use some HTML code outside of a full HTML document (e.g., web page), it's important to understand the overall layout of an HTML file if you're going to design a web page.

Elements and Tags

Before we get into the structure, let's look at the basic building block of HTML: the element. Elements usually have an opening and closing tag, and content within them.

Let's consider the paragraph element:

```
<p>A paragraph of text would go here.</p>
```

The `<p>` portion is the opening tag, and the `</p>` portion is the closing tag. Between the two is the actual content that (usually) appears on the screen.

Elements have attributes. The attributes available vary, but `id` and `class` are perhaps the most important attributes.

```
<p id="theId" class="theCssClass">
    Some text here.
</p>
```

The `id` attribute is used to give the element a referenceable value to be used in JavaScript and Cascading Style Sheets (CSS). The `class` attribute assigns CSS class names to an element.

Head

The `<head>` element is a special element with an open and closing tag (`<head>` and `</head>`) that contains special elements like `<title> </title>` and `<meta>` (meta has no closing tag) that define attributes about the web page for your browser and search engines. These elements aren't visible to the user.

Body

The `<body>` element, which begins and ends with `<body>` and `</body>`, is where the bulk of the page lies, and makes up the visual portion of the page.

Comments

Any HTML code within comments is not shown on the page.

```
<!-- This is an HTML comment. -->
```

HTML comments can span multiple lines. They start with `<!--` and end with `-->`.

Using Divs and Semantic Elements

The most common element to use to group other elements together is `<div>`, which has an opening and a closing tag:

```
<div id="footer">
    <p>Copyright &copy; 2024 Your Company, All Rights Reserved.</p>
    <img src="images/logo.png" alt="Your Logo">
</div>
```

This is a very useful technique, but semantic elements provide a cleaner way to do this:

```
<footer>
    <p>Copyright &copy; 2024 Your Company, All Rights Reserved.</p>
    <img src="images/logo.png" alt="Your Logo">
</footer>
```

Now you can apply styles via CSS directly to the `footer` element, rather than having to specify the `id`.

Here's a list of semantic elements in HTML (table 12):

GRAPHIC

table 12

ELEMENT	PURPOSE
`<article></article>`	Best used for an article, blog post, comment, or similar.
`<aside></aside>`	Great for call-outs, quotes, etc.
`<details></details>`	A toggle-like element that can show and hide additional details. Used with the `<summary></summary>` element.
`<figcaption></figcaption>`	Used to caption a figure.
`<figure></figure>`	Used to insert a figure into content. The `<figcaption></figcaption>` element can optionally be used inside the `<figure></figure>` element to caption the figure.
`<footer></footer>`	Typically used to display information at the bottom of the page.
`<header></header>`	Typically used to display information at the top of the page.
`<main></main>`	Helps group the dominant content portion of the `<body></body>` element. There should be only one `<main></main>` element on a page.
`<mark></mark>`	Highlights text within a larger collection of text (e.g., a `<p></p>` element).
`<nav></nav>`	Used to contain the navigation elements of the page (e.g., a menu or breadcrumb navigation).
`<section></section>`	A generic grouping of other elements. Should contain a heading `(<h1></h1>`, `<h2></h2>`, etc., element).
`<summary></summary>`	Within a `<details></details>` element, specifies the text that is clickable. The text outside the `summary` element, but within the `details` element, is hidden, displayed when the `summary` text is clicked.
`<time></time>`	Used to display a date/time value. The `datetime` attribute of this element allows for additional formatting.

MY TAKE

I prefer to use semantic elements whenever possible. However, sometimes you need to use `div` elements when a semantic element to describe the grouping doesn't exist or isn't practical.

Frequently Used Elements

Here is a list of some of the most frequently used HTML elements (table 13):

GRAPHIC

table 13

ELEMENT	PURPOSE

`<h1></h1>`
`<h2></h2>`
`<h3></h3>`
`<h4></h4>`
`<h5></h5>`
`<h6></h6>`
	These heading elements allow for text within the tags to be displayed in larger, bolder fonts and serve as text structure of the page. H1 is the largest and H6 is the smallest.

`<p></p>`	A paragraph of text.
`<a></a>`	A link to another page. The text within the opening and closing tags is the text displayed for the link, and the `href` attribute (e.g., `<a href=”page.html”>` denotes the URL to navigate to.
`<img>`	The image element is used to display an image on the page. It does not have a closing tag.

`<ul>`
  `<li></li>`
`</ul>`
	An unordered list is a bullet list with `<li></li>` elements, each of which contains the text for one of the list items.

`<ol>`
  `<li></li>`
`</ol>`
	An ordered list (starting with 1, and incrementing with each item) with `<li></li>` elements, each of which contains the text for one of the list items. You can use the value I in the `type` attribute to display Roman numerals instead of numbers.

`<div></div>`	A generic but incredibly useful element to group other elements together.
`<span></span>`	Similar to `<div></div>`; however, it is meant to work on smaller groups inline within divs.

`<table>`
  `<tr>`
  `<td></td>`
  `</tr>`
`</table>`
	Displays a table. Each `<tr></tr>` is a row, and each `<td></td>` is a cell.

Creating Forms

Forms allow you to provide a space on your web page to collect data from visitors:

```
<form method="POST" action="url">
    <input type="email" name="email" value="">
    <input type="submit" name="submit" value="Submit">
</form>
```

In this example, the `method` attribute defines that the form will be sent via the HTTP verb POST, and the `action` attribute specifies the URL (in this case, I put "url" as a placeholder, but normally a real page or web address would go here).

Two input elements exist within this example. The first is the attribute `type="email"`, which denotes a text input box that accepts and validates an email address. The name of the input box is also `email`. The second `input` element has `type="submit"`, meaning it's a submit button—the `value` of which sets the text to show on the button, in this case "Submit".

Designing with Cascading Style Sheets (CSS)

HTML defines the structure and content of a document, while Cascading Style Sheets (CSS) applies style to the HTML elements in a cascading fashion (hence the name). By cascading, I mean that hierarchies of elements are considered when applying these styles.

A stylesheet is (usually) a separate file and has the extension of `.css`. You can include stylesheets in an HTML page within the `<head></head>` element like this:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>An Example HTML Page</title>
        <link rel="stylesheet" href="css/style.css">
    </head>
    <body>
```

```
        <main>
            <p>Page content goes here!</p>
        </main>
    </body>
</html>
```

The line that adds the stylesheet in the above example is in bold type. Here it is separately…

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

This would include a file called `style.css` in the `css` folder.
CSS files contain rules like this:

```
body {
    font-family: "Open Sans";
}

.boldText { font-weight: bold; }
```

Rules contain selectors, which, in the example above, would be `body` and `.boldText`. The `body` selector will apply the rule within the curly braces (in this case, setting the font to Open Sans) to all HTML elements within the body (hence the cascading part of the acronym CSS).

The second rule shown is a CSS class, which can be applied with the `class` attribute on any HTML element. So, if these two CSS rules are active in our file, we can apply the class `.boldText` to a span within a paragraph to bold that particular portion.

```
<p>Learning JavaScript is <span class="boldText">FUN</span>!</p>
```

You can apply a rule to any HTML element, custom class, or a particular ID (defined by the `id` attribute on an HTML element).

CSS File

```
#specialPart { font-weight: bold; }
```

HTML File

```
<div id="specialPart">
    <p>Something really special!</p>
</div>
```

CSS classes can be applied to multiple HTML elements, and each HTML element class attribute can contain multiple CSS classes, separated by a space. But IDs belong to only one element on the page. An element may have both an ID and a class.

Comments can be inserted into CSS files like this:

```
/* This line will not be processed as a CSS rule */
```

The Box Model

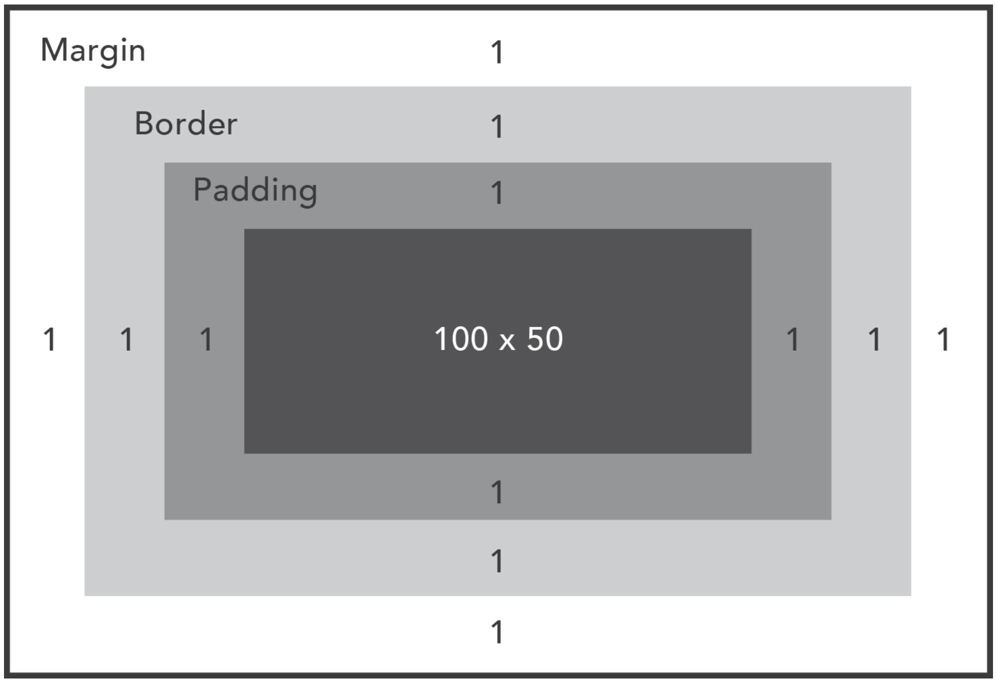
The box model is a paradigm in web development that envisions rectangular boxes around HTML elements. The box in the center contains the element's content. This content is sized via the `width` and `height` CSS properties. But the element consumes more space than the content's width and height. It is surrounded by a layer called padding, or the padding box.

The next layer outward is the border. In many cases, this isn't defined, but you can set a border around an element and whatever size the border is becomes the additional space taken up by the main element. It's important to note that the padding lies between the border and the element, so if you add padding, the border will appear to be separate from the element. Finally, the margin box is the amount of space between the element in question and other elements.

If a rectangular element has a width of 100 pixels and a height of 50 pixels, and a padding, border, and margin all set to 1 pixel for all sides, then the actual space the element will consume on the page will be at least 106 pixels wide and 56 pixels tall. To calculate this, consider that the width of an element includes the padding, border, and margin values of both left and right sides. Likewise, the height of an element includes the padding, border, and margin values of the top and bottom sides (figure 39).

GRAPHIC

fig. 39



NOTE

The margins aren't technically considered part of the element, but rather part of the space around it.

Colors

You can specify colors in CSS in several ways. The simplest is to use one of the CSS color names that are supported in all browsers. There are 140 colors in the collection—too many to list here (see https://developer.mozilla.org/en-US/docs/Web/CSS/named-color for a full list). Let's work with some examples:

```
.redText { color: red; }
.blueText { color: blue; }
.orangeText { color: orange; }
```

You can also specify colors in hexadecimal format:

```
.redText { color: #FF0000; }
```

Hexadecimal colors use hex values for red, green, and blue. You'll note that FF is in the first spot, meaning FF (256 in decimal) units of red, 00 (0 in decimal) units of green, and 00 (0 in decimal) units in blue. This will yield a very bright red.

There are many tools on the web (and in web design programs) that can give you hex colors via a color picker. Here's one I use quite often: https://htmlcheatsheet.com. It's also worth noting that Visual Studio Code puts a little character-sized square of the color inline as a handy reference as you type.

MY TAKE

If the color I want to use is in the list of 140 browser-supported color names, I use it, if possible. This works well and is easy to remember. I'd rather memorize lemonchiffon than #FFFACD, or royalblue instead of #4169E1.

Fonts

You can specify fonts in CSS, as we did in a previous example:

```
body {
    font-family: "Open Sans";
}
```

There are quite a few fonts that are available in most browsers and systems, like Arial, Helvetica, Verdana, etc., and generic names like sans-serif, serif, cursive, and monospace.

Google Fonts gives you a wide assortment of very nice-looking fonts and even gives you the code to use. Visit https://fonts.google.com/ to browse the fonts, click one, then select your desired weight (400 is considered regular), then click the plus icon to view the `<link>` tag to import the font.

NOTE

In the years I've been using Google Fonts, the interface has changed quite a few times, so things might look different for you than I described. However, the general paradigm of picking your font, selecting the weight (or style, like italic), then clicking a button to see the CSS code has remained.

Responsive Design

The topic of responsive design—that is, web design that looks good on all devices, from computers through tablets to smart phones—could fill its own book. This topic is covered in depth in *HTML & CSS QuickStart Guide* by David DuRocher. To summarize the concept here, you can specify media queries within your CSS file to tailor CSS rules to particular device sizes.

Here’s a simple example:

```
/* This part applies to the entire file */

    body {
        font-family: "Open Sans";
    }

        .redText { color: red; }

    #logo {
        width: 100px;
        height: 80px;

    /* This part applies to devices with a maximum width of 500
pixels */
    @media screen and (max-width: 500px) {
        #logo {
            width: 50px;
            height: 40px;
        }
    }
```

In this CSS example, the HTML element with the ID of `logo` will have a width of 100 pixels and a height of 80 pixels, unless you’re on a device with a maximum width of 500 pixels (like a small cell phone). In that case, the CSS rule(s) within the curly braces will apply. Here, that would make the width of the logo 50 pixels and the height 40 pixels.

Appendix II

Using the http-server Node.js Package

As we've discussed throughout the book, you can browse HTML files with Google Chrome locally—that is, directly—by loading them in the web browser. This works fine for most cases, but some features, like cookies, don't work in this mode.

To test cookies, your web pages need to be hosted on a web server. A simple solution for this is to install the `http-server` Node.js package. Fortunately, this process is easy no matter which operating system you're running.

Simply launch your terminal (or command prompt) and run the command:

```
npm install --global http-server
```

Once this is done, the `http-server` command is available. To use, change to any directory in the terminal with the `cd` command, like this…

Windows

```
cd %USERPROFILE%\Source\coffee-shop
```

macOS and Linux

```
cd ~/Source/coffee-shop
```

…then run the `http-server` command, like this…

```
http-server
```

You can also specify the path at the end of the `http-server` command and omit the `cd` step. This is helpful if you want to put this command in a script or batch file. To do this, run…

Windows

```
http-server %USERPROFILE%\Source\coffee-shop
```

macOS and Linux

```
http-server ~/Source/coffee-shop
```

In either example, change `Source/coffee-shop` to the path of your source code folder with the `index.html` file you wish to view.

If all goes well, you should see something like this in the terminal window:

```
Starting up http-server, serving ./

http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
    http://127.0.0.1:8080
    http://10.0.0.63:8080
Hit CTRL-C to stop the server
```

If `http-server` doesn't start (i.e., you get a "command not found" or similar error message), try closing the terminal and restarting it.

Now you can copy one of the links displayed in the "Available on:" section (I recommend choosing the `127.0.0.1` link) and paste this into your browser. Going to http://127.0.0.1:8080/ will show you the `index.html` and other pages upon request from the current directory.

If you're using this to test the coffee shop, you can now open the JavaScript console and watch as you add items to the cart. You can run `getCart()` in the console to see the cart stored in the cookies.

Appendix III

Using Common Third-Party Libraries

Sometimes we all need a little help from our friends. With JavaScript, our friends include some helpful third-party libraries that smooth over the rough edges in JavaScript. Here are some of my favorites, along with quick overviews of how to use them.

currency.js

Formatting currency and working with floating-point numbers can be tricky in JavaScript, but fortunately, the `currency.js` library can help. The currency.js library is extremely lightweight and simple to use.

Installation

There are multiple ways to install `currency.js`, but the simplest is to download the JavaScript source file to your local website files and include it via a `<script>` tag. You can visit https://currency.js.org to obtain the link. As of publication, the *Download currency.js* button at the top of the page is a direct link to the JavaScript source of the library. You can save this to your source code folder by clicking the three-dot menu in Chrome and selecting *Save and share* then *Save page as…*

You can also use npm to download the file to your project automatically if you'd like. However, the instructions here will focus on simply including the downloaded file in your project.

Assuming you save the `currency.js` file to your source code folder, and the HTML file that uses it is in the same folder, the HTML file would look like this:

currency.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Using currency.js</title>
    </head>
    <body>
        <script src="currency.min.js"></script>
    </body>
</html>
```

Usage

The principal workflow of `currency.js` involves creating currency objects with the values you need. These objects contain the value and the necessary methods and internal logic to properly handle currency and floating-point numbers.

NOTE

To run this code, either place the code samples below in a `<script></script>` tag on the `currency.html` page or load the page in the browser and use the JavaScript console.

IMPORTANT

The filename you download from the official website is named currency.min.js, a minified version of the currency.js script. This is why we load it with `<script src="currency.min.js"></script>`. The min in the filename isn't what makes it minified, as any .js file can be minified, but this generally is an indication it is minified. In any case, you'll have to make sure the filename in the `src` attribute of the script tag matches the actual filename.

For example, if we wanted to create a currency object with a price, it would look something like this:

```
let price = currency(4.95)
```

We can now discount this price without fear of any floating-point variable issues:

```
// Price with 20% discount
let discountedPrice = price.multiply(0.8)
```

To display the `discountedPrice` variable, we can now use…

```
console.log(discountedPrice.format())
```

…which displays…

```
$3.96
```

If you want to display the currency symbol for your locale, you can assign it when you create the currency object:

```
// Use Euro currency
let price = currency(4.95, { symbol: "€" })
```

Using methods like `add`, `subtract`, `multiply`, and `divide` on the resulting currency object allows for correct mathematical calculations.

You can even chain calculations together in a single function:

```
// Start with 4.95
let price = currency(4.95)

// Add 2.25 to the price then subtract 3.16
let newPrice = price.add(2.25).subtract(3.16)

// Display the new price
console.log(newPrice.format())
```

For more information about the capabilities of `currency.js`, visit currency.js.org.

Moment.js

The `moment.js` JavaScript library makes it easy to work with time, dates, and time zones.

Installation

As with `currency.js`, there are several ways to install `moment.js`. However, to keep things simple, we'll download the JavaScript source file and include it via a `script` tag. You can download the file from https://momentjs.com. As of publication, there are four files you can download, but essentially two versions—one with locales and one without. If you don't need support for internationalization features (sometimes abbreviated as i18n in programming), I'd recommend downloading the `moment.js` version.

When I said there were four files, both the locale and non-locale versions have a regular and minified version. For local use, I'd recommend the regular version (`moment.js` and `moment-with-locales.js`). However, for website use, it's preferable to use the minified version (with the `min.js` extension) to reduce download time.

Assuming you save the `moment.js` file to your source code folder, and the HTML file that uses it is in the same folder, the HTML file would look like this:

moment.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Using moment.js</title>
    </head>
    <body>
        <script src="moment.js"></script>
    </body>
</html>
```

NOTE

The non-minified version of `moment.js` is easily readable in the browser, or in Visual Studio Code. Browsing other people's code can be very informative, and you'll probably learn some new techniques, so I'd recommend checking it out.

Usage

NOTE

To run this code, either place the code samples below in a `<script></script>` tag on the `moment.html` page or load the page in the browser and use the JavaScript console.

First, let's get the current time and date with moment, but in a format of our choosing.

```
// Set the now variable to the current time and date
let now = moment().format("MMMM Do, YYYY @ h:mm:ss a")

// Display it
console.log(now)
```

When I run this code in my JavaScript console (with `moment.html` loaded), I see:

```
September 30th, 2024 @ 6:17:11pm
```

This is the current time at which I'm writing this appendix. Yours will be different, of course. And, if you like, you can format it a bit differently than I did. You can remove the `@` symbol between the time and date, or remove the commas. But if you'd like to change the formatting codes (e.g., `MMMM`, `Do`, `h:mm:ss`, etc.), you can refer to the formatting codes at the momentjs.com/ website, which, as of publication, are found at momentjs.com/docs/#/displaying/format.

But `moment.js` does far more than simply format dates. One of my favorite features is relative time. Adding minutes, hours, or dates to dates can be annoying, but `moment.js` makes it easy:

```
// Set the now variable to the current time and date
let now = moment()
```

```
// Add 1 hour, then display it
let inOneHour = now.add(1, "hours")
console.log(inOneHour.format("MMMM Do, YYYY @ h:mm:ss a"))
```

In this example, I don't add the `format` method until I'm ready to use it. By default, setting a variable to the result of `moment()` returns the current time and date. I can perform any calculations to that time I want without being concerned about formatting until I am ready to display it on a page (or, in this case, the JavaScript console).

We can also find out the difference between two dates:

```
// Obtain the current time
let now = moment()

// Obtain a time in the past
let past = moment("2000-01-01")

// Display the difference between the two times
console.log('The start of the new millennium was about ${past.
from(now)}.')
```

When I run this code, I see:

```
The start of the new millennium was about 25 years ago.
```

I feel old reading that sentence!

To learn more about `moment.js` and its features, including locale and time zone support, which we don't have space to cover here, go to momentjs.com.

Appendix IV

On Your Own Answers

Chapter 1: Getting Started with JavaScript

Logging to the Console

In this example, rather than use Robert, I changed the name to Marsha. Use whatever name you wish.

```
console.log("Hello, Marsha!")
```

Displaying on the Web Page

In this example, I changed “Hello, World!” to “Hello, Robert!”.

```
<body>
    <p>
    <script>
        document.write("Hello, Robert!");
     </script>
    </p>
 </body>
```

Chapter 2: Using Variables

Template Literals

```
greetingContainer.textContent = `Hello, $
{firstNameTextBox.value}!`
```

Arrays

In this example, I made an array containing notable cities in Tennessee.

```
let tennesseeCities =
    ["Nashville", "Memphis", "Chattanooga", "Knoxville"]
```

Chapter 6: Organizing Data and Logic with Objects

Using Properties

```
class Address {
    constructor(firstName, lastName, country) {
        this.firstName = firstName
        this.lastName = lastName
        this.country = country
    }
}

let customer = new Address("Robert", "Oliver", "United States")

for (let detail in customer) {
    console.log(detail + ": " + customer[detail])
}
```

Chapter 8: Using the Document Object Model (DOM)

Changing an Element's CSS

```
for (let p of paragraphs) {
    p.style.color = "red"
}
```