

CSS3 Workshop

Exercises and Exams for Students

CSS



Abdelfattah Ragab

CSS3 Workshop

Exercises and Exams for Students

Abdelfattah Ragab

© 2023 Abdelfattah Ragab.

All rights reserved.

[Introduction](#)

[Chapter 1: Selectors and Specificity](#)

[1.1 CSS3 selectors and their usage](#)

[Type Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Universal Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Class Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[ID Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Attribute Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Pseudo-Class Selectors](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Pseudo-Element Selectors](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Descendant Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Child Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Adjacent Sibling Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[General Sibling Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Enabled Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Disabled Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Checked Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Empty Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Root Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Lang Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[UI-State Pseudo-Classes](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[1.2 Specificity rules and calculations](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[1.3 Advanced selector techniques in CSS3](#)

[Descendant Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Child Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Adjacent Sibling Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[General Sibling Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Attribute Selectors](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[:nth-child\(\) and :nth-of-type\(\)](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[:not\(\) Selector](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[:focus and :hover Selectors](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Chapter 2: CSS3 Box Model](#)

[2.1 Box Sizing](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[2.2 Margin and Padding Properties](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[2.3 Width and Height Properties](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Chapter 3: CSS Positioning](#)

[3.1 The position property](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[3.2 The display property](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[3.3 The z-index property.](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Chapter 4: Modern CSS layouts](#)

[4.1 Flexbox Layout](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[4.2 Grid Layout](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[4.3 Multi-column Layout](#)

Exercises

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

4.4 Shapes

shape-outside

Exercises

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

shape-inside

Exercises

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

shape-margin

Exercises

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[clip-path](#)

[Exercises](#)

[Exercise 1](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[4.5 Scroll Snap](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Chapter 5: CSS3 Backgrounds and Borders](#)

[5.1 Styling Backgrounds with CSS3](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[5.2 Gradient Backgrounds and Image Effects](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

5.3 Border Properties in CSS3

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

5.4 Rounded Corners and Box Shadows

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

Chapter 6: CSS3 Typography and Web Fonts

6.1 Web Fonts and @font-face Rule

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

6.2 Customizing Typography with CSS3

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[6.3 Text Shadows and Text Effects](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[6.4 Creating Responsive Typography](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Chapter 7: CSS3 Transitions and Animations](#)

[7.1 Transition Properties and Timing Functions](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[7.2 Creating Animations with CSS3 Keyframes](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[7.3 Advanced Animation Techniques](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Chapter 8: CSS3 Transformations and 3D Effects](#)

[8.1 2D Transforms](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[8.2 3D Transforms and Perspective](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[8.3 Creating 3D Effects and Animations](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Chapter 9: CSS3 Media Queries and Responsive Design](#)

[9.1 Media Queries and Breakpoints](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[9.2 Responsive Layouts with CSS3](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[9.3 Optimizing Images for Different Devices](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Chapter 10: Advanced CSS3 Techniques](#)

[10.1 CSS Variables and Custom Properties](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[10.2 CSS Calc\(\) Function](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[10.3 CSS Filters and Blend Modes](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[10.4 Working with SVG in CSS3](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Chapter 11: CSS Best Practices and Optimization](#)

[11.1 Writing Efficient and Maintainable CSS](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[11.2 CSS Performance Optimization Techniques](#)

[Exercises](#)

[Exercise 1](#)

[Solution](#)

[Exercise 2](#)

[Solution](#)

[Exercise 3](#)

[Solution](#)

[Exams](#)

[Exam 1](#)

[Exam 2](#)

[Exam 3](#)

[Solutions](#)

[Exam 1](#)

[Exam 2](#)

[Exam 3](#)

[Conclusion](#)

[About the Author](#)

Introduction

Welcome to the CSS3 Workshop: Exercises and Exams for Students! This comprehensive resource has been designed to provide students with a hands-on learning experience in mastering CSS3, the latest version of Cascading Style Sheets.

CSS3 plays a crucial role in web development, allowing designers and developers to enhance the visual appeal and interactivity of websites. With its advanced features and capabilities, CSS3 enables the creation of stunning and responsive user interfaces.

This workshop is structured to guide students through a series of exercises and exams that cover the fundamental concepts and practical applications of CSS3. Whether you are a beginner taking your first steps into web development or an experienced student looking to expand your skills, this workshop will equip you with the knowledge and techniques needed to excel in the world of CSS3 styling.

Throughout the workshop, you will embark on a progressive learning journey. We will start with an introduction to the basics of CSS, including selectors, properties, and values. As we dive deeper, we will explore the powerful features of CSS3, such as transitions, transformations, animations, and responsive design principles.

Each exercise presents a real-world scenario, challenging you to apply your CSS3 knowledge to solve specific design problems.

These exercises are designed to reinforce your understanding of CSS3 concepts and sharpen your problem-solving skills. Within each

exercise, you will find step-by-step instructions and code snippets to guide you through the process.

Additionally, the workshop includes exams that assess your comprehension and mastery of CSS3 concepts. These exams will test your ability to apply CSS3 techniques to various scenarios and evaluate your proficiency in creating visually compelling and responsive web interfaces.

As you progress through this workshop, it is crucial to practice and experiment with the concepts covered. CSS3 is a dynamic and evolving technology, and it is essential to stay updated with the latest trends and best practices. By immersing yourself in hands-on exercises and exams, you will build a solid foundation in CSS3 and develop the skills necessary to create modern and impactful web designs.

I encourage you to approach this workshop with enthusiasm and a willingness to explore and experiment. Remember, the best way to learn CSS3 is to dive in headfirst and apply your knowledge to real-world scenarios.

So, let's embark on this exciting journey of mastering CSS3 together! By the end of this workshop, you will have the skills and confidence to create visually stunning and responsive web interfaces that captivate users and leave a lasting impression.

Happy styling!

Abdelfattah Ragab

Chapter 1: Selectors and Specificity

1.1 CSS3 selectors and their usage

Here's an example of CSS3 selectors and their usage:

index.html

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css"
        href="styles.css">
</head>
<body>
    <h1>Welcome to CSS Selectors Example</h1>
    <div id="container">
        <p class="highlight">This paragraph has a class
        of "highlight".</p>
        <p>This paragraph does not have any special
        class or id.</p>
        <p id="special">This paragraph has an id of
        "special".</p>
    </div>
</body>
</html>
```

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <link rel="stylesheet" type="text/css" href="styles.css" />
5    </head>
6    <body>
7      <h1>Welcome to CSS Selectors Example</h1>
8      <div id="container">
9        <p class="highlight">This paragraph has a class of "highlight".</p>
10       <p>This paragraph does not have any special class or id.</p>
11       <p id="special">This paragraph has an id of "special".</p>
12     </div>
13   </body>
14 </html>
15
```

styles.css

```
/* Selecting elements by tag name */
h1 {
  color: blue;
}

/* Selecting elements by class name */
.highlight {
  font-weight: bold;
}

/* Selecting elements by id */
#special {
  text-decoration: underline;
}

/* Selecting elements by descendant selector */
#container p {
  margin: 10px;
}

/* Selecting elements by child selector */
#container > p {
  background-color: yellow;
}
```

```
/* Selecting elements by attribute */
p[class="highlight"] {
    color: red;
}
```

```
1 /* Selecting elements by tag name */
2 h1 {
3     color: blue;
4 }
5
6 /* Selecting elements by class name */
7 .highlight {
8     font-weight: bold;
9 }
10
11 /* Selecting elements by id */
12 #special {
13     text-decoration: underline;
14 }
15
16 /* Selecting elements by descendant selector */
17 #container p {
18     margin: 10px;
19 }
20
21 /* Selecting elements by child selector */
22 #container > p {
23     background-color: yellow;
24 }
25
26 /* Selecting elements by attribute */
27 p[class="highlight"] {
28     color: red;
29 }
```

In this example, we have various CSS3 selectors used to style different elements. Here's a breakdown of the selectors and their usage:

- Selecting elements by tag name: The `h1` selector selects the `h1` element and applies a blue color to it.
- Selecting elements by class name: The `.highlight` selector selects the element with the class "highlight" (`<p class="highlight">`) and applies a bold font weight to it.
- Selecting elements by id: The `#special` selector selects the element with the id "special" (`<p id="special">`) and applies an

underline text decoration to it.

- Selecting elements by descendant selector: The `#container p` selector selects all `p` elements that are descendants of the element with the id "container" (`<div id="container">`). It applies a margin of 10 pixels to these paragraphs.
- Selecting elements by child selector: The `#container > p` selector selects the `p` elements that are immediate children of the element with the id "container" and applies a yellow background color to them.
- Selecting elements by attribute: The `p[class="highlight"]` selector selects the `p` element with the class "highlight" and applies a red color to it.

These are just a few examples of CSS3 selectors and their usage.

CSS3 provides many more selectors that can be used to target specific elements and apply styles to them.

Type Selector

Targets elements of a specific type.

Syntax: `elementName`

Examples: `a`, `p`, `div`

Exercises

Exercise 1

Create a CSS rule that selects all paragraphs (`<p>` elements) on a webpage and sets their font size to 16 pixels.

Solution

```
p {  
    font-size: 16px;  
}
```

```
1 p {  
2     font-size: 16px;  
3 }  
4
```

This CSS rule selects all paragraphs (`<p>` elements) and sets their font size to 16 pixels.

Exercise 2

Create a CSS rule that selects all heading elements (`<h1>`, `<h2>`, `<h3>`, etc.) and gives them a background color of light gray.

Solution

```
h1, h2, h3, h4, h5, h6 {  
    background-color: lightgray;  
}
```

```
1  h1,  
2  h2,  
3  h3,  
4  h4,  
5  h5,  
6  h6 {  
7      background-color: lightgray;  
8  }
```

This CSS rule selects all heading elements (`<h1>`, `<h2>`, `<h3>`, etc.) and gives them a background color of light gray.

Exercise 3

Create a CSS rule that selects all list item elements (``) inside an unordered list (``) and adds a left margin of 20 pixels.

Solution

```
ul li {  
    margin-left: 20px;  
}
```

```
1  ul li {  
2      margin-left: 20px;  
3  }
```

This CSS rule selects all list item elements (``) that are inside an unordered list (``) and adds a left margin of 20 pixels.

Universal Selector

Targets all elements in the document.

Syntax: *

Exercises

Exercise 1

Create a CSS rule that selects all elements on a webpage and sets their margin to 0.

Solution

```
* {  
    margin: 0;  
}
```

```
1 * {  
2     margin: 0;  
3 }
```

This CSS rule selects all elements on the webpage using the universal selector (*) and sets their margin to 0.

Exercise 2

Create a CSS rule that selects all elements with a class of "highlight" and sets their background color to yellow.

Solution

```
*.highlight {  
    background-color: yellow;  
}
```

```
1  *.highlight {  
2      background-color: yellow;  
3  }
```

This CSS rule selects all elements with a class of "highlight" (`<div class="highlight">`, `<p class="highlight">`, etc.) using the universal selector (*) and sets their background color to yellow.

Exercise 3

Create a CSS rule that selects all elements inside a <div> and sets their text color to red.

Solution

```
div * {  
    color: red;  
}  
  
1 div * {  
2     color: red;  
3 }
```

This CSS rule selects all elements that are descendants of a <div> element using the universal selector (*) and sets their text color to red.

Class Selector

Targets elements with a specific class attribute.

Syntax: .className

Exercises

Exercise 1

Create a CSS rule that selects all elements with the class "box" and sets their background color to blue.

Solution

```
.box {  
    background-color: blue;  
}  
  
1 .box {  
2     background-color: blue;  
3 }
```

This CSS rule selects all elements with the class "box" and sets their background color to blue.

Exercise 2

Create a CSS rule that selects all `<a>` elements with the class "link" and sets their text color to green.

Solution

```
a.link {  
    color: green;  
}
```

```
1 a.link {  
2     color: green;  
3 }  
4
```

This CSS rule selects all `<a>` elements with the class "link" and sets their text color to green.

Exercise 3

Create a CSS rule that selects all elements with both the classes "highlight" and "large" and sets their font weight to bold.

Solution

```
.highlight.large {  
    font-weight: bold;  
}
```

```
1  .highlight.large {  
2      font-weight: bold;  
3  }
```

This CSS rule selects all elements with both the classes "highlight" and "large" and sets their font weight to bold.

ID Selector

Targets an element with a specific ID attribute.

Syntax: #idName

Exercises

Exercise 1

Create a CSS rule that selects the element with the ID "header" and sets its background color to gray.

Solution

```
#header {  
    background-color: gray;  
}
```

```
1 #header {  
2     background-color: gray;  
3 }
```

This CSS rule selects the element with the ID "header" and sets its background color to gray.

Exercise 2

Create a CSS rule that selects the element with the ID "logo" and sets its width to 200 pixels.

Solution

```
#logo {  
    width: 200px;  
}
```

```
1  #logo {  
2      width: 200px;  
3  }
```

This CSS rule selects the element with the ID "logo" and sets its width to 200 pixels.

Exercise 3

Create a CSS rule that selects the element with the ID "navigation" and sets its text color to blue.

Solution

```
#navigation {  
    color: blue;  
}
```

```
1 #navigation {  
2     color: blue;  
3 }
```

This CSS rule selects the element with the ID "navigation" and sets its text color to blue.

Attribute Selector

Targets elements based on attribute presence or specific attribute-value combinations.

Syntax: [attributeName], [attributeName="value"],
[attributeName^="value"], [attributeName\$="value"],
[attributeName*="value"]

Exercises

Exercise 1

Create a CSS rule that selects all `<a>` elements with a `target` attribute and sets their text color to red.

Solution

```
a[target] {  
    color: red;  
}
```

```
1 a[target] {  
2     color: red;  
3 }
```

This CSS rule selects all `<a>` elements with a `target` attribute and sets their text color to red.

Exercise 2

Create a CSS rule that selects all `<input>` elements with a type attribute value of "text" and sets their border color to green.

Solution

```
input[type="text"] {  
    border-color: green;  
}
```

```
1  input[type="text"] {  
2      border-color: green;  
3  }
```

This CSS rule selects all <input> elements with a type attribute value of "text" and sets their border color to green.

Exercise 3

Create a CSS rule that selects all elements with a data-theme attribute value of "dark" and sets their background color to black.

Solution

```
[data-theme="dark"] {  
    background-color: black;  
}  
  
1  [data-theme="dark"] {  
2      background-color: black;  
3  }
```

This CSS rule selects all elements with a data-theme attribute value of "dark" and sets their background color to black.

Pseudo-Class Selectors

Targets elements based on a specific state or condition.

Syntax: :pseudoClass

Examples: :hover, :active, :focus, :first-child, :last-child, :nth-child(), :nth-of-type(), :nth-last-child(), :nth-last-of-type(), :not(), :checked, :disabled, :enabled, :target, :empty, :root, :lang(), etc.

Exercises

Exercise 1

Create a CSS rule that selects all `<a>` elements when they are being hovered over and sets their text color to red.

Solution

```
a:hover {  
    color: red;  
}
```

```
1 a:hover {  
2     color: red;  
3 }
```

This CSS rule selects all <a> elements when they are being hovered over and sets their text color to red.

Exercise 2

Create a CSS rule that selects all `<input>` elements that are in focus and sets their border color to blue.

Solution

```
input:focus {  
    border-color: blue;  
}
```

```
1  input:focus {  
2      border-color: blue;  
3  }
```

This CSS rule selects all <input> elements that are in focus and sets their border color to blue.

Exercise 3

Create a CSS rule that selects every other `` element within an ordered list (``) and sets their background color to light gray.

Solution

```
ol li:nth-child(even) {  
    background-color: lightgray;  
}
```

```
1  ol li:nth-child(even) {  
2      background-color: lightgray;  
3  }
```

This CSS rule selects every other `` element within an ordered list (``) and sets their background color to light gray.

Pseudo-Element Selectors

Targets a specific part of an element's content.

Syntax: ::pseudoElement

Examples: ::before, ::after, ::first-letter, ::first-line, ::selection, ::placeholder, ::marker, ::backdrop, etc.

Exercises

Exercise 1

Create a CSS rule that adds a content before each `<h1>` element, displaying the text "Chapter: ".

Solution

```
h1::before {  
    content: "Chapter: ";  
}
```

```
1 h1::before {  
2     content: "Chapter: ";  
3 }
```

This CSS rule adds the content "Chapter: " before each <h1> element.

Exercise 2

Create a CSS rule that adds a double border below each `<blockquote>` element.

Solution

```
blockquote::after {  
    content: "";  
    display: block;  
    border-bottom: 2px double;  
}
```

```
1  blockquote::after {  
2      content: "";  
3      display: block;  
4      border-bottom: 2px double;  
5  }
```

This CSS rule adds a double border below each `<blockquote>` element using the `::after` pseudo-element.

Exercise 3

Create a CSS rule that adds a background color to the first letter of each paragraph (`<p>`) element.

Solution

```
p::first-letter {  
    background-color: yellow;  
}  
  
1 p::first-letter {  
2     background-color: yellow;  
3 }
```

This CSS rule adds a background color of yellow to the first letter of each `<p>` element using the `::first-letter` pseudo-element.

Descendant Selector

Targets elements that are descendants of a specific ancestor.

Syntax: ancestor descendant

Exercises

Exercise 1

Create a CSS rule that selects all `<h2>` elements that are descendants of a `<div>` element and sets their text color to blue.

Solution

```
div h2 {  
    color: blue;  
}
```

```
1  div h2 {  
2      color: blue;  
3  }
```

This CSS rule selects all `<h2>` elements that are descendants of a `<div>` element and sets their text color to blue.

Exercise 2

Create a CSS rule that selects all `` elements that are descendants of an `` element with the class "menu" and sets their font weight to bold.

Solution

```
ul.menu li {  
    font-weight: bold;  
}
```

```
1  ul.menu li {  
2      font-weight: bold;  
3  }
```

This CSS rule selects all elements that are descendants of an element with the class "menu" and sets their font weight to bold.

Exercise 3

Create a CSS rule that selects all `` elements that are descendants of an element with the class "container" and sets their background color to yellow.

Solution

```
.container span {  
    background-color: yellow;  
}
```

```
1 .container span {  
2     background-color: yellow;  
3 }
```

This CSS rule selects all elements that are descendants of an element with the class "container" and sets their background color to yellow.

Child Selector

Targets elements that are direct children of a specific parent.

Syntax: parent > child

Exercises

Exercise 1

Create a CSS rule that selects all direct child `` elements of an `` element and sets their text color to blue.

Solution

```
ul > li {  
    color: blue;  
}
```

```
1  ul > li {  
2      color: blue;  
3  }
```

This CSS rule selects all direct child `` elements of an `` element and sets their text color to blue.

Exercise 2

Create a CSS rule that selects all direct child `<div>` elements of a `<section>` element with the class "container" and sets their background color to gray.

Solution

```
section.container > div {  
    background-color: gray;  
}
```

```
1 section.container > div {  
2     background-color: gray;  
3 }
```

This CSS rule selects all direct child `<div>` elements of a `<section>` element with the class "container" and sets their background color to gray.

Exercise 3

Create a CSS rule that selects all direct child `` elements of a `<p>` element with the class "highlight" and sets their font weight to bold.

Solution

```
p.highlight > span {  
    font-weight: bold;  
}
```

```
1  p.highlight > span {  
2      font-weight: bold;  
3  }
```

This CSS rule selects all direct child `` elements of a `<p>` element with the class "highlight" and sets their font weight to bold.

Adjacent Sibling Selector

Targets elements that are immediately following a specific element.

Syntax: element + sibling

Exercises

Exercise 1

Create a CSS rule that selects the `<h2>` element that immediately follows a `<h1>` element and sets its text color to red.

Solution

```
h1 + h2 {  
    color: red;  
}
```

```
1  h1 + h2 {  
2      color: red;  
3  }
```

This CSS rule selects the <h2> element that immediately follows a <h1> element and sets its text color to red.

Exercise 2

Create a CSS rule that selects the element that immediately follows another element and sets its background color to lightgray.

Solution

```
1 li + li {  
2   background-color: lightgray;  
3 }
```

```
1 li + li {  
2   background-color: lightgray;  
3 }
```

This CSS rule selects the element that immediately follows another element and sets its background color to lightgray.

Exercise 3

Create a CSS rule that selects the `` element that immediately follows an `<a>` element with the class "link" and sets its font weight to bold.

Solution

```
a.link + span {  
    font-weight: bold;  
}
```

```
1 a.link + span {  
2     font-weight: bold;  
3 }
```

This CSS rule selects the element that immediately follows an <a> element with the class "link" and sets its font weight to bold.

General Sibling Selector

Targets elements that are siblings of a specific element (not necessarily immediately following).

Syntax: element ~ sibling

Exercises

Exercise 1

Create a CSS rule that selects all `<p>` elements that are siblings of an `<h2>` element and sets their text color to blue.

Solution

```
h2 ~ p {  
  color: blue;  
}
```

```
1  h2 ~ p {  
2    color: blue;  
3  }
```

This CSS rule selects all `<p>` elements that are siblings of an `<h2>` element and sets their text color to blue.

Exercise 2

Create a CSS rule that selects all `<input>` elements that are siblings of a `<label>` element and sets their border color to red.

Solution

```
label ~ input {  
    border-color: red;  
}
```

```
1  label ~ input {  
2      border-color: red;  
3  }
```

This CSS rule selects all <input> elements that are siblings of a <label> element and sets their border color to red.

Exercise 3

Create a CSS rule that selects all `` elements that are siblings of an element with the class "highlight" and sets their background color to yellow.

Solution

```
.highlight ~ span {  
    background-color: yellow;  
}  
  
1 .highlight ~ span {  
2     background-color: yellow;  
3 }
```

This CSS rule selects all elements that are siblings of an element with the class "highlight" and sets their background color to yellow.

Enabled Selector

Targets form elements that are enabled.

Syntax: `:enabled`

Exercises

Exercise 1

Create a CSS rule that selects all enabled <input> elements and sets their background color to lightgreen.

Solution

```
input:enabled {  
    background-color: lightgreen;  
}
```

```
1  input:enabled {  
2      background-color: lightgreen;  
3  }
```

This CSS rule selects all enabled <input> elements and sets their background color to lightgreen.

Exercise 2

Create a CSS rule that selects all enabled <button> elements and sets their text color to white and background color to blue.

Solution

```
button:enabled {  
    color: white;  
    background-color: blue;  
}
```

```
1 button:enabled {  
2     color: white;  
3     background-color: blue;  
4 }
```

This CSS rule selects all enabled <button> elements and sets their text color to white and background color to blue.

Exercise 3

Create a CSS rule that selects all enabled `<a>` elements and sets their border color to red.

Solution

```
a:enabled {  
    border-color: red;  
}  
  
1 a:enabled {  
2     border-color: red;  
3 }
```

This CSS rule selects all enabled <a> elements and sets their border color to red.

Disabled Selector

Targets form elements that are disabled.

Syntax: :disabled

Exercises

Exercise 1

Create a CSS rule that selects all disabled `<input>` elements and sets their background color to lightgray.

Solution

```
input:disabled {  
    background-color: lightgray;  
}
```

```
1  input:disabled {  
2      background-color: lightgray;  
3  }
```

This CSS rule selects all disabled <input> elements and sets their background color to lightgray.

Exercise 2

Create a CSS rule that selects all disabled <button> elements and sets their text color to gray and background color to lightblue.

Solution

```
button:disabled {  
    color: gray;  
    background-color: lightblue;  
}
```

```
1 button:disabled {  
2     color: gray;  
3     background-color: lightblue;  
4 }
```

This CSS rule selects all disabled <button> elements and sets their text color to gray and background color to lightblue.

Exercise 3

Create a CSS rule that selects all disabled `<a>` elements and sets their text color to red.

Solution

```
a:disabled {  
    color: red;  
}
```

```
1 a:disabled {  
2     color: red;  
3 }
```

This CSS rule selects all disabled `<a>` elements and sets their text color to red.

Checked Selector

Targets form elements that are checked.

Syntax: `:checked`

Exercises

Exercise 1

Create a CSS rule that selects all checked `<input>` elements of type "checkbox" and sets their background color to lightgreen.

Solution

```
input[type="checkbox"] :checked {  
    background-color: lightgreen;  
}
```

```
1  input[type="checkbox"] :checked {  
2      background-color: lightgreen;  
3  }
```

This CSS rule selects all checked <input> elements of type "checkbox" and sets their background color to lightgreen.

Exercise 2

Create a CSS rule that selects all checked <input> elements of type "radio" and sets their border color to blue.

Solution

```
input[type="radio"]:checked {  
    border-color: blue;  
}
```

```
1  input[type="radio"]:checked {  
2      border-color: blue;  
3  }
```

This CSS rule selects all checked <input> elements of type "radio" and sets their border color to blue.

Exercise 3

Create a CSS rule that selects all checked `<option>` elements within a `<select>` element and sets their text color to red.

Solution

```
select option:checked {  
    color: red;  
}
```

```
1 select option:checked {  
2     color: red;  
3 }
```

This CSS rule selects all checked <option> elements within a <select> element and sets their text color to red.

Empty Selector

Targets elements that have no children or content.

Syntax: `:empty`

Exercises

Exercise 1

Create a CSS rule that selects all empty `<p>` elements and sets their border color to red.

Solution

```
p:empty {  
    border-color: red;  
}
```

```
1 p:empty {  
2     border-color: red;  
3 }
```

This CSS rule selects all empty <p> elements and sets their border color to red.

Exercise 2

Create a CSS rule that selects all empty <div> elements and sets their background color to yellow.

Solution

```
div:empty {  
    background-color: yellow;  
}
```

```
1  div:empty {  
2      background-color: yellow;  
3  }
```

This CSS rule selects all empty <div> elements and sets their background color to yellow.

Exercise 3

Create a CSS rule that selects all empty `` elements and sets their text color to gray.

Solution

```
span:empty {  
    color: gray;  
}
```

```
1  span:empty {  
2      color: gray;  
3  }
```

This CSS rule selects all empty `` elements and sets their text color to gray.

Root Selector

Targets the root element of the document (typically <html>).

Syntax: :root

Exercises

Exercise 1

Create a CSS rule that selects the root element (`<html>`) and sets its background color to lightgray.

Solution

```
:root {  
    background-color: lightgray;  
}  
  
1 :root {  
2     background-color: lightgray;  
3 }
```

This CSS rule selects the root element (<html>) and sets its background color to lightgray.

Exercise 2

Create a CSS rule that selects the root element (`<html>`) and sets the font size to 16 pixels.

Solution

```
:root {  
  font-size: 16px;  
}
```

```
1  :root {  
2    font-size: 16px;  
3  }
```

This CSS rule selects the root element (<html>) and sets the font size to 16 pixels.

Exercise 3

Create a CSS rule that selects the root element (<html>) and sets the text color to navy blue.

Solution

```
:root {  
  color: navy;  
}  
  
1 :root {  
2   color: navy;  
3 }
```

This CSS rule selects the root element (<html>) and sets the text color to navy blue.

Lang Selector

Targets elements based on the language attribute.

Syntax: `:lang(language)`

Exercises

Exercise 1

Create a CSS rule that selects all `<h1>` elements in English and sets their text color to blue.

Solution

```
h1:lang(en) {  
    color: blue;  
}
```

```
1 h1:lang(en) {  
2     color: blue;  
3 }
```

This CSS rule selects all `<h1>` elements in English and sets their text color to blue.

Exercise 2

Create a CSS rule that selects all `<p>` elements in French and sets their font style to italic.

Solution

```
p:lang(fr) {  
    font-style: italic;  
}  
  
1 p:lang(fr) {  
2     font-style: italic;  
3 }
```

This CSS rule selects all <p> elements in French and sets their font style to italic.

Exercise 3

Create a CSS rule that selects all `` elements in Spanish and sets their text decoration to underline.

Solution

```
span:lang(es) {  
    text-decoration: underline;  
}  
  
1  span:lang(es) {  
2      text-decoration: underline;  
3  }
```

This CSS rule selects all elements in Spanish and sets their text decoration to underline.

UI-State Pseudo-Classes

Targets form elements based on their user interface state.

Examples: :focus-within, :read-only, :read-write, :optional, :required, :valid, :invalid, :in-range, :out-of-range, etc.

Exercises

Exercise 1

Create a CSS rule that selects the <button> element in its default state and sets its background color to lightgray.

Solution

```
button:default {  
    background-color: lightgray;  
}
```

```
1 button:default {  
2     background-color: lightgray;  
3 }
```

This CSS rule selects the <button> element in its default state and sets its background color to lightgray.

Exercise 2

Create a CSS rule that selects the <input> element when it is being hovered over and sets its border color to blue.

Solution

```
input:hover {  
    border-color: blue;  
}  
  
1  input:hover {  
2      border-color: blue;  
3  }
```

This CSS rule selects the <input> element when it is being hovered over and sets its border color to blue.

Exercise 3

Create a CSS rule that selects the `<a>` element when it has focus (when it is selected using the keyboard or mouse) and sets its text color to red.

Solution

```
a:focus {  
    color: red;  
}
```

```
1 a:focus {  
2     color: red;  
3 }
```

This CSS rule selects the <a> element when it has focus (when it is selected using the keyboard or mouse) and sets its text color to red.

1.2 Specificity rules and calculations

Here's an example that demonstrates CSS3 specificity rules and calculations:

```
<!DOCTYPE html>
<html>
<head>
    <title>Specificity Example</title>
    <style>
        /* Rule 1 */
        h1 {
            color: blue;
        }
        /* Rule 2 */
        #header h1 {
            color: red;
        }
        /* Rule 3 */
        body #header h1 {
            color: green;
        }
    </style>
</head>
<body>
    <div id="header">
        <h1>Hello, world!</h1>
    </div>
</body>
```

```
</html>
```

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Specificity Example</title>
5      <style>
6        /* Rule 1 */
7        h1 {
8          color: blue;
9        }
10
11       /* Rule 2 */
12       #header h1 {
13         color: red;
14       }
15
16       /* Rule 3 */
17       body #header h1 {
18         color: green;
19       }
20     </style>
21   </head>
22   <body>
23     <div id="header">
24       <h1>Hello, world!</h1>
25     </div>
26   </body>
27 </html>
```

In this example, we have three CSS rules targeting the `<h1>` element within different selectors. Let's calculate their specificities:

- Rule 1: Specificity = 0,0,1,0 (one element selector)
- Rule 2: Specificity = 0,1,1,0 (one ID selector and one element selector)
- Rule 3: Specificity = 0,1,2,0 (one ID selector and two element selectors)

Based on the specificity values, Rule 3 has the highest specificity. Therefore, the text color of the `<h1>` element will be green, overriding the styles defined in Rule 2 and Rule 1.

By understanding specificity and calculating the specificity values of CSS selectors, you can determine which styles take precedence over others and ensure the desired styles are applied to your HTML elements.

Exercises

Exercise 1

Given the following CSS rules, calculate the specificity of each selector and determine which rule will take precedence:

```
/* Rule 1 */
h1 {
    color: red;
}

/* Rule 2 */
#header h1 {
    color: blue;
}

/* Rule 3 */
body #header h1 {
    color: green;
}
```

```
1  /* Rule 1 */
2  h1 {
3      color: red;
4  }
5
6  /* Rule 2 */
7  #header h1 {
8      color: blue;
9  }
10
11 /* Rule 3 */
12 body #header h1 {
13     color: green;
14 }
```

Solution

- Rule 1: Specificity = 0,0,1,0 (one element selector)
- Rule 2: Specificity = 0,1,1,0 (one ID selector and one element selector)
- Rule 3: Specificity = 0,1,2,0 (one ID selector and two element selectors)

Since Rule 3 has the highest specificity, it will take precedence over Rule 2 and Rule 1.

Exercise 2

Given the following CSS rules, calculate the specificity of each selector and determine which rule will take precedence:

```
/* Rule 1 */
#header .title {
    color: red;
}

/* Rule 2 */
.title {
    color: blue;
}

/* Rule 3 */
#header {
    color: green;
}
```

```
1  /* Rule 1 */
2  #header .title {
3      color: red;
4  }
5
6  /* Rule 2 */
7  .title {
8      color: blue;
9  }
10
11 /* Rule 3 */
12 #header {
13     color: green;
14 }
```

Solution

- Rule 1: Specificity = 0,1,1,1 (one ID selector, one class selector, and one element selector)
- Rule 2: Specificity = 0,0,1,1 (one class selector and one element selector)
- Rule 3: Specificity = 0,1,0,0 (one ID selector)

Since Rule 1 has the highest specificity, it will take precedence over Rule 2 and Rule 3.

Exercise 3

Given the following CSS rules, calculate the specificity of each selector and determine which rule will take precedence:

```
/* Rule 1 */
body .container p {
    color: red;
}

/* Rule 2 */
.container p {
    color: blue;
}

/* Rule 3 */
p {
    color: green;
}
```

```
1  /* Rule 1 */
2  body .container p {
3      color: red;
4  }
5
6  /* Rule 2 */
7  .container p {
8      color: blue;
9  }
10
11 /* Rule 3 */
12 p {
13     color: green;
14 }
```

Solution

- Rule 1: Specificity = 0,0,2,1 (two class selectors and one element selector)
- Rule 2: Specificity = 0,0,1,1 (one class selector and one element selector)
- Rule 3: Specificity = 0,0,0,1 (one element selector)

Since Rule 1 has the highest specificity, it will take precedence over Rule 2 and Rule 3.

1.3 Advanced selector techniques in CSS3

Here's an example that demonstrates advanced selector techniques in CSS3:

```
<!DOCTYPE html>
<html>
<head>
    <title>Advanced Selector Techniques
Example</title>
<style>
    /* Descendant Selector */
    .container h1 {
        color: blue;
    }
    /* Child Selector */
    .container > p {
        font-weight: bold;
    }
    /* Adjacent Sibling Selector */
    h1 + p {
        text-decoration: underline;
    }
    /* Attribute Selector */
    a[href="https://www.example.com"] {
        color: red;
    }
    /* :not() Selector */
    p:not(.special) {
```

```
        font-style: italic;  
    }  
  </style>  
</head>  
<body>  
  <div class="container">  
    <h1>Title</h1>  
    <p>Regular paragraph</p>  
    <p class="special">Special paragraph</p>  
  </div>  
  <a href="https://www.example.com">Visit  
Example</a>  
</body>  
</html>
```

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Advanced Selector Techniques Example</title>
5      <style>
6        /* Descendant Selector */
7        .container h1 {
8          color: blue;
9        }
10
11       /* Child Selector */
12       .container > p {
13         font-weight: bold;
14       }
15
16       /* Adjacent Sibling Selector */
17       h1 + p {
18         text-decoration: underline;
19       }
20
21       /* Attribute Selector */
22       a[href="https://www.example.com"]
23       {
24         color: red;
25       }
26
27       /* :not() Selector */
28       p:not(.special) {
29         font-style: italic;
30       }
31     </style>
32   </head>
33   <body>
34     <div class="container">
35       <h1>Title</h1>
36       <p>Regular paragraph</p>
37       <p class="special">Special paragraph</p>
38     </div>
39     <a href="https://www.example.com">Visit Example</a>
40   </body>
41 </html>
```

In this example, we'll explore different advanced selector techniques:

- Descendant Selector (.container h1): This selector targets all `<h1>` elements that are descendants of elements with the class "container." The selected `<h1>` element will have a blue text color.
- Child Selector (.container > p): This selector targets all `<p>` elements that are direct children of elements with the class "container." The selected `<p>` element will have bold font weight.

- Adjacent Sibling Selector (`h1 + p`): This selector targets the `<p>` element that immediately follows an `<h1>` element. The selected `<p>` element will have an underline text decoration.
- Attribute Selector (`a[href="https://www.example.com"]`): This selector targets the `<a>` element with the specific `href` attribute value of "https://www.example.com". The selected `<a>` element will have a red text color.
- `:not()` Selector (`p:not(.special)`): This selector targets all `<p>` elements that do not have the class "special". The selected `<p>` elements will have italic font style.

These advanced selector techniques allow you to target specific elements based on their relationships, attributes, or exclusion of certain classes. They provide more flexibility and precision in styling your HTML elements.

Descendant Selector

The descendant selector allows you to select an element that is a descendant of another element. It uses the space () character between two selectors. For example, `div p` selects all `<p>` elements that are descendants of `<div>` elements.

Exercises

Exercise 1

Create a CSS rule that selects all `` elements that are descendants of a `` element and sets their font weight to bold.

Solution

```
ul li {  
    font-weight: bold;  
}
```

```
1  ul li {  
2      font-weight: bold;  
3  }
```

This CSS rule selects all elements that are descendants of a element and sets their font weight to bold.

Exercise 2

Create a CSS rule that selects all `` elements that are descendants of a `<div>` element with the class "container" and sets their text color to red.

Solution

```
div.container span {  
    color: red;  
}
```

```
1 div.container span {  
2     color: red;  
3 }
```

This CSS rule selects all elements that are descendants of a <div> element with the class "container" and sets their text color to red.

Exercise 3

Create a CSS rule that selects all `<p>` elements that are descendants of a `<div>` element with the ID "content" and sets their background color to yellow.

Solution

```
div#content p {  
    background-color: yellow;  
}  
  
1 div#content p {  
2     background-color: yellow;  
3 }
```

This CSS rule selects all `<p>` elements that are descendants of a `<div>` element with the ID "content" and sets their background color to yellow.

Child Selector

The child selector targets elements that are direct children of another element. It uses the greater than (>) symbol between two selectors. For example, `ul > li` selects all `` elements that are immediate children of `` elements.

Exercises

Exercise 1

Create a CSS rule that selects all `<p>` elements that are direct children of a `<div>` element and sets their font size to 16 pixels.

Solution

```
div > p {  
    font-size: 16px;  
}
```

```
1  div > p {  
2      font-size: 16px;  
3  }
```

This CSS rule selects all `<p>` elements that are direct children of a `<div>` element and sets their font size to 16 pixels.

Exercise 2

Create a CSS rule that selects all `` elements that are direct children of an `` element with the class "list" and sets their text color to blue.

Solution

```
ol.list > li {  
    color: blue;  
}
```

```
1  ol.list > li {  
2      color: blue;  
3  }
```

This CSS rule selects all elements that are direct children of an element with the class "list" and sets their text color to blue.

Exercise 3

Create a CSS rule that selects all `` elements that are direct children of a `<div>` element with the ID "container" and sets their background color to yellow.

Solution

```
div#container > span {  
    background-color: yellow;  
}  
  
1  div#container > span {  
2      background-color: yellow;  
3  }
```

This CSS rule selects all `` elements that are direct children of a `<div>` element with the ID "container" and sets their background color to yellow.

Adjacent Sibling Selector

The adjacent sibling selector selects an element that comes immediately after another element. It uses the plus (+) symbol between two selectors. For example, `h2 + p` selects the `<p>` element that directly follows an `<h2>` element.

Exercises

Exercise 1

Create a CSS rule that selects the `<h2>` element that immediately follows an `<h1>` element and sets its color to red.

Solution

```
h1 + h2 {  
    color: red;  
}
```

```
1  h1 + h2 {  
2      color: red;  
3  }
```

This CSS rule selects the <h2> element that immediately follows an <h1> element and sets its color to red.

Exercise 2

Create a CSS rule that selects the `<p>` element that immediately follows an `` element and sets its `margin-top` to 10 pixels.

Solution

```
img + p {  
  margin-top: 10px;  
}
```

```
1  img + p {  
2    margin-top: 10px;  
3  }
```

This CSS rule selects the <p> element that immediately follows an element and sets its margin-top to 10 pixels.

Exercise 3

Create a CSS rule that selects the <div> element that immediately follows a <h3> element with the class "subtitle" and sets its background color to green.

Solution

```
h3.subtitle + div {  
    background-color: green;  
}  
  
1  h3.subtitle + div {  
2      background-color: green;  
3  }
```

This CSS rule selects the <div> element that immediately follows a <h3> element with the class "subtitle" and sets its background color to green.

General Sibling Selector

The general sibling selector targets elements that are siblings of another element. It uses the tilde (~) symbol between two selectors. For example, h2 ~ p selects all `<p>` elements that are siblings of an `<h2>` element.

Exercises

Exercise 1

Create a CSS rule that selects all `<p>` elements that are siblings of a `<div>` element and sets their font size to 14 pixels.

Solution

```
div ~ p {  
    font-size: 14px;  
}
```

```
1  div ~ p {  
2      font-size: 14px;  
3  }
```

This CSS rule selects all <p> elements that are siblings of a <div> element and sets their font size to 14 pixels.

Exercise 2

Create a CSS rule that selects all `` elements that are siblings of an `<h2>` element and sets their text color to blue.

Solution

```
h2 ~ li {  
    color: blue;  
}
```

```
1  h2 ~ li {  
2      color: blue;  
3  }
```

This CSS rule selects all elements that are siblings of an <h2> element and sets their text color to blue.

Exercise 3

Create a CSS rule that selects all `` elements that are siblings of a `<p>` element with the class "highlight" and sets their background color to yellow.

Solution

```
p.highlight ~ span {  
    background-color: yellow;  
}  
  
1 p.highlight ~ span {  
2     background-color: yellow;  
3 }
```

This CSS rule selects all `` elements that are siblings of a `<p>` element with the class "highlight" and sets their background color to yellow.

Attribute Selectors

Attribute selectors allow you to select elements based on their attributes and attribute values. There are different types of attribute selectors, including:

- [attribute]: Selects elements with a specific attribute, regardless of its value.
- [attribute=value]: Selects elements with a specific attribute value.
- [attribute^=value]: Selects elements with an attribute value that starts with a specific value.
- [attribute\$=value]: Selects elements with an attribute value that ends with a specific value.
- [attribute*=value]: Selects elements with an attribute value that contains a specific value.

Exercises

Exercise 1

Create a CSS rule that selects all `<a>` elements with a "target" attribute set to "`_blank`" and sets their text color to red.

Solution

```
a[target="_blank"] {  
    color: red;  
}
```

```
1 a[target="_blank"] {  
2     color: red;  
3 }
```

This CSS rule selects all <a> elements with a "target" attribute set to "_blank" and sets their text color to red.

Exercise 2

Create a CSS rule that selects all `<input>` elements with a "type" attribute set to "email" and sets their border color to blue.

Solution

```
input[type="email"] {  
    border-color: blue;  
}
```

```
1  input[type="email"] {  
2      border-color: blue;  
3  }
```

This CSS rule selects all <input> elements with a "type" attribute set to "email" and sets their border color to blue.

Exercise 3

Create a CSS rule that selects all elements with a "data-highlight" attribute and sets their background color to yellow.

Solution

```
[data-highlight] {  
    background-color: yellow;  
}  
  
1 [data-highlight] {  
2     background-color: yellow;  
3 }
```

This CSS rule selects all elements with a "data-highlight" attribute and sets their background color to yellow.

`:nth-child()` and `:nth-of-type()`

These pseudo-classes allow you to select elements based on their position in relation to their parent. The `:nth-child()` pseudo-class selects elements based on their position among all the children of their parent, while `:nth-of-type()` selects elements based on their position among elements of the same type within their parent.

Exercises

Exercise 1

Create a CSS rule that selects every 2nd element and sets its background color to light gray.

Solution

```
1 li:nth-child(2n) {  
2   background-color: lightgray;  
3 }
```

```
1 li:nth-child(2n) {  
2   background-color: lightgray;  
3 }
```

This CSS rule selects every 2nd element and sets its background color to light gray.

Exercise 2

Create a CSS rule that selects every 3rd `<p>` element within a `<div>` and sets its font weight to bold.

Solution

```
div p:nth-of-type(3n) {  
    font-weight: bold;  
}
```

```
1 div p:nth-of-type(3n) {  
2     font-weight: bold;  
3 }
```

This CSS rule selects every 3rd `<p>` element within a `<div>` and sets its font weight to bold.

Exercise 3

Create a CSS rule that selects every odd `` element within a `<div>` with the class "container" and sets its text color to blue.

Solution

```
div.container span:nth-of-type(odd) {  
    color: blue;  
}
```

```
1 div.container span:nth-of-type(odd) {  
2     color: blue;  
3 }
```

This CSS rule selects every odd element within a <div> with the class "container" and sets its text color to blue.

:not() Selector

The :not() selector allows you to select elements that do not match a specific selector. It is useful for excluding certain elements from a selection. For example, `p:not(.special)` selects all `<p>` elements that do not have the class "special".

Exercises

Exercise 1

Create a CSS rule that selects all `<p>` elements except those with the class "special" and sets their text color to blue.

Solution

```
p:not(.special) {  
    color: blue;  
}
```

```
1 p:not(.special) {  
2     color: blue;  
3 }
```

This CSS rule selects all `<p>` elements except those with the class "special" and sets their text color to blue.

Exercise 2

Create a CSS rule that selects all `<input>` elements except those with the type "checkbox" and sets their border color to red.

Solution

```
input:not([type="checkbox"]) {  
    border-color: red;  
}
```

```
1  input:not([type="checkbox"]) {  
2      border-color: red;  
3  }
```

This CSS rule selects all <input> elements except those with the type "checkbox" and sets their border color to red.

Exercise 3

Create a CSS rule that selects all `<a>` elements except those with the class "external" and sets their font weight to bold.

Solution

```
a:not(.external) {  
    font-weight: bold;  
}
```

```
1 a:not(.external) {  
2     font-weight: bold;  
3 }
```

This CSS rule selects all <a> elements except those with the class "external" and sets their font weight to bold.

:focus and :hover Selectors

The :focus selector targets elements that currently have keyboard focus, such as form inputs. The :hover selector targets elements when the mouse pointer is hovering over them. These selectors are commonly used for styling interactive elements.

Exercises

Exercise 1

Create a CSS rule that applies a border to an <input> element when it is in focus.

Solution

```
input:focus {  
    border: 2px solid blue;  
}
```

```
1  input:focus {  
2      border: 2px solid blue;  
3  }
```

This CSS rule applies a border to an <input> element when it is in focus.

Exercise 2

Create a CSS rule that changes the background color of a <button> element when it is hovered over.

Solution

```
button:hover {  
    background-color: lightgray;  
}
```

```
1 button:hover {  
2     background-color: lightgray;  
3 }
```

This CSS rule changes the background color of a <button> element when it is hovered over.

Exercise 3

Create a CSS rule that increases the font size of a <a> element when it is both hovered over and in focus.

Solution

```
a:hover:focus {  
    font-size: 18px;  
}
```

```
1  a:hover:focus {  
2      font-size: 18px;  
3  }
```

This CSS rule increases the font size of a <a> element when it is both hovered over and in focus.

Chapter 2: CSS3 Box Model

2.1 Box Sizing

Here's an example that demonstrates the concept of CSS3 box sizing:

```
<style>
  .container {
    width: 300px;
    height: 200px;
    padding: 20px;
    border: 2px solid black;
    box-sizing: border-box;
  }
  .box {
    width: 100%;
    height: 100%;
    background-color: lightgray;
  }
</style>
<div class="container">
  <div class="box"></div>
</div>
```

```
1 <style>
2   .container {
3     width: 300px;
4     height: 200px;
5     padding: 20px;
6     border: 2px solid black;
7     box-sizing: border-box;
8   }
9
10  .box {
11    width: 100%;
12    height: 100%;
13    background-color: lightgray;
14  }
15 </style>
16
17 <div class="container">
18   <div class="box"></div>
19 </div>
```

In this example, we have a container `<div>` with a class of "container". It has a width of 300 pixels, height of 200 pixels, and a padding of 20 pixels. It also has a border of 2 pixels solid black. The box-sizing property is set to "border-box", which means that the specified width and height of the container include the padding and border dimensions.

Inside the container, we have another `<div>` with a class of "box". It is set to have a width and height of 100%, meaning it will fill the entire width and height of its parent container. The background color of the box is set to light gray.

By setting the box-sizing property to "border-box" on the container, the total width and height of the container, including its padding and border, will be 300 pixels and 200 pixels respectively. This ensures that the inner box element, which has a width and height of 100%, will fit perfectly inside the container without overflowing due to the padding and border.

This example demonstrates how the box-sizing property can be used to control the sizing behavior of elements, taking into account the padding and border dimensions.

Exercises

Exercise 1

Create a CSS rule that sets the box sizing of all elements to border-box.

Solution

```
* {  
    box-sizing: border-box;  
}  
  
1 * {  
2     box-sizing: border-box;  
3 }
```

This CSS rule sets the box sizing of all elements to border-box, which includes the padding and border dimensions in the specified width and height.

Exercise 2

Create a CSS rule that sets the box sizing of all `<div>` elements with the class "box" to content-box.

Solution

```
div.box {  
    box-sizing: content-box;  
}  
  
1 div.box {  
2     box-sizing: content-box;  
3 }
```

This CSS rule sets the box sizing of all <div> elements with the class "box" to content-box, which only includes the content dimensions in the specified width and height.

Exercise 3

Create a CSS rule that sets the box sizing of all `` elements to border-box and gives them a border of 1px solid red.

Solution

```
img {  
    box-sizing: border-box;  
    border: 1px solid red;  
}
```

```
1  img {  
2      box-sizing: border-box;  
3      border: 1px solid red;  
4  }
```

This CSS rule sets the box sizing of all elements to border-box, which includes the padding and border dimensions in the specified width and height. It also adds a 1px solid red border to the images.

2.2 Margin and Padding Properties

Here's an example that demonstrates the CSS3 margin and padding properties:

```
<style>
  .container {
    width: 300px;
    height: 200px;
    background-color: lightgray;
    padding: 20px;
  }
  .box {
    margin: 10px;
    padding: 20px;
    background-color: white;
  }
</style>
<div class="container">
  <div class="box">Box</div>
</div>
```

```
1 <style>
2   .container {
3     width: 300px;
4     height: 200px;
5     background-color: lightgray;
6     padding: 20px;
7   }
8
9   .box {
10    margin: 10px;
11    padding: 20px;
12    background-color: white;
13  }
14 </style>
15
16 <div class="container">
17   <div class="box">Box</div>
18 </div>
```

In this example, we have a container `<div>` with a class of "container". It has a width of 300 pixels, height of 200 pixels, and a background color of light gray. The container also has a padding of 20 pixels, which creates space between the content and the container's edges.

Inside the container, we have another `<div>` with a class of "box". It has a margin of 10 pixels, which creates space around the box. The box also has a padding of 20 pixels, which creates space between the content and the box's edges. The background color of the box is set to white.

The combination of margin and padding properties allows you to control the spacing around and within elements. The margin creates space outside the element, whereas the padding creates space inside the element. In this example, the container has padding to create space between its content and edges, and the box has margin to create space around it.

You can adjust the values of margin and padding to achieve different spacing effects and customize the layout of your web page.

Exercises

Exercise 1

Create a CSS rule that sets the margin of all paragraphs (`<p>` elements) to 10 pixels and the padding to 5 pixels.

Solution

```
p {  
    margin: 10px;  
    padding: 5px;  
}
```

```
1 p {  
2     margin: 10px;  
3     padding: 5px;  
4 }
```

This CSS rule sets the margin of all paragraphs (<p> elements) to 10 pixels and the padding to 5 pixels.

Exercise 2

Create a CSS rule that sets the margin of all images (`` elements) to 20 pixels and the padding to 0.

Solution

```
img {  
    margin: 20px;  
    padding: 0;  
}
```

```
1 img {  
2     margin: 20px;  
3     padding: 0;  
4 }
```

This CSS rule sets the margin of all images (elements) to 20 pixels and the padding to 0.

Exercise 3

Create a CSS rule that sets the margin of all <div> elements with the class "box" to 10 pixels on the top and bottom, and 20 pixels on the left and right. Set the padding to 15 pixels.

Solution

```
div.box {  
    margin: 10px 20px;  
    padding: 15px;  
}
```

```
1  div.box {  
2      margin: 10px 20px;  
3      padding: 15px;  
4  }
```

This CSS rule sets the margin of all <div> elements with the class "box" to 10 pixels on the top and bottom, and 20 pixels on the left and right. It also sets the padding to 15 pixels.

2.3 Width and Height Properties

Here's an example that demonstrates the CSS3 width and height properties:

```
<style>
  .container {
    width: 300px;
    height: 200px;
    background-color: lightgray;
  }
  .box {
    width: 150px;
    height: 100px;
    background-color: white;
  }
</style>
<div class="container">
  <div class="box">Box</div>
</div>
```

```
1  <style>
2    .container {
3      width: 300px;
4      height: 200px;
5      background-color: lightgray;
6    }
7
8    .box {
9      width: 150px;
10     height: 100px;
11     background-color: white;
12   }
13  </style>
14
15 <div class="container">
16   <div class="box">Box</div>
17 </div>
```

In this example, we have a container `<div>` with a class of "container". It has a width of 300 pixels and a height of 200 pixels. The background color of the container is set to light gray. Inside the container, we have another `<div>` with a class of "box". It has a width of 150 pixels and a height of 100 pixels. The background color of the box is set to white.

The width and height properties allow you to specify the dimensions of elements on your web page. In this example, the container has a width of 300 pixels and a height of 200 pixels, which determines its size on the page. The box inside the container has a width of 150 pixels and a height of 100 pixels, which determines its size relative to the container.

You can adjust the values of width and height to control the size of elements and customize the layout of your web page.

Exercises

Exercise 1

Create a CSS rule that sets the width of all `` elements to 200 pixels and the height to auto, maintaining the aspect ratio.

Solution

```
img {  
    width: 200px;  
    height: auto;  
}
```

```
1 img {  
2     width: 200px;  
3     height: auto;  
4 }
```

This CSS rule sets the width of all elements to 200 pixels and the height to auto, which maintains the aspect ratio and ensures that the image scales proportionally.

Exercise 2

Create a CSS rule that sets the width of all `<div>` elements with the class "box" to 50% of their parent container's width and the height to 100 pixels.

Solution

```
div.box {  
    width: 50%;  
    height: 100px;  
}
```

```
1 div.box {  
2     width: 50%;  
3     height: 100px;  
4 }
```

This CSS rule sets the width of all <div> elements with the class "box" to 50% of their parent container's width and the height to 100 pixels.

Exercise 3

Create a CSS rule that sets the width and height of all `<a>` elements with the class "button" to 120 pixels, creating square buttons.

Solution

```
a.button {  
    width: 120px;  
    height: 120px;  
}
```

```
1 a.button {  
2     width: 120px;  
3     height: 120px;  
4 }
```

This CSS rule sets the width and height of all `<a>` elements with the class "button" to 120 pixels, creating square buttons.

Chapter 3: CSS Positioning

3.1 The position property

Here's an example that demonstrates positioning elements with CSS3:

```
<style>

  .container {
    position: relative;
    height: 200px;
    background-color: lightgray;
  }

  .box {
    position: absolute;
    width: 100px;
    height: 100px;
    background-color: white;
  }

  .box:nth-child(1) {
    top: 20px;
    left: 20px;
  }

  .box:nth-child(2) {
    top: 50px;
    right: 20px;
  }

  .box:nth-child(3) {
    bottom: 20px;
    left: 50%;
    transform: translateX(-50%);
  }

</style>
```

```

        }
    </style>
<div class="container">
    <div class="box">Box 1</div>
    <div class="box">Box 2</div>
    <div class="box">Box 3</div>
</div>

```

```

1  <style>
2      .container {
3          position: relative;
4          height: 200px;
5          background-color: lightgray;
6      }
7
8      .box {
9          position: absolute;
10         width: 100px;
11         height: 100px;
12         background-color: white;
13     }
14
15     .box:nth-child(1) {
16         top: 20px;
17         left: 20px;
18     }
19
20     .box:nth-child(2) {
21         top: 50px;
22         right: 20px;
23     }
24
25     .box:nth-child(3) {
26         bottom: 20px;
27         left: 50%;
28         transform: translateX(-50%);
29     }
30 </style>
31
32 <div class="container">
33     <div class="box">Box 1</div>
34     <div class="box">Box 2</div>
35     <div class="box">Box 3</div>
36 </div>
37

```

In this example, we have a container `<div>` with a class of "container". It has a height of 200 pixels and a background color of

light gray. The container is set to position: relative, which establishes it as the containing block for the positioned elements inside.

Inside the container, we have three <div> elements with a class of "box". They are set to position: absolute, which allows us to position them precisely within the container. Each box has a width and height of 100 pixels and a background color of white.

The first box (Box 1) is positioned using top: 20px and left: 20px, which places it 20 pixels from the top and left edges of the container. The second box (Box 2) is positioned using top: 50px and right: 20px, which places it 50 pixels from the top and 20 pixels from the right edges of the container.

The third box (Box 3) is positioned using bottom: 20px, left: 50%, and transform: translateX(-50%). This aligns the box 20 pixels from the bottom and horizontally centers it within the container using a combination of percentage-based positioning and the transform property.

By manipulating the position and properties of the elements, you can achieve precise positioning and layout effects on your web page.

Exercises

Exercise 1

Create a CSS rule that positions an image at the top-right corner of its parent container, with a margin of 10 pixels.

Exercise 2:

Create a CSS rule that positions a `<div>` element in the center of the browser window, with a fixed position. The element should have a width of 300 pixels and a height of 200 pixels.

Exercise 3:

Create a CSS rule that positions a `<p>` element at the bottom of its parent container, sticking to the bottom even when scrolling. The element should have a fixed position and a width of 100%.

Solution

```
img {  
    position: absolute;  
    top: 10px;  
    right: 10px;  
    margin: 10px;  
}
```

```
1 img {  
2     position: absolute;  
3     top: 10px;  
4     right: 10px;  
5     margin: 10px;  
6 }
```

In this solution, we set the position: absolute property on the img element to position it precisely. The top: 10px and right: 10px properties determine the distance of the image from the top and right edges of its parent container, respectively. The margin: 10px property adds a margin of 10 pixels to provide some spacing around the image.

Exercise 2

Create a CSS rule that positions a <div> element with the class "box" at the bottom-center of its parent container, and set its width to 200 pixels and height to 100 pixels.

Solution

```
div {  
    position: fixed;  
    top: 50%;  
    left: 50%;  
    transform: translate(-50%, -50%);  
    width: 300px;  
    height: 200px;  
}
```

```
1 div {  
2     position: fixed;  
3     top: 50%;  
4     left: 50%;  
5     transform: translate(-50%, -50%);  
6     width: 300px;  
7     height: 200px;  
8 }
```

In this solution, we set the `position: fixed` property on the `div` element to position it relative to the browser window. The `top: 50%` and `left: 50%` properties position the element at the center of the window. The `transform: translate(-50%, -50%)` property combination horizontally and vertically centers the element by moving it back 50% of its own width and height. We also set the `width` and `height` of the element to 300 pixels and 200 pixels, respectively.

Exercise 3

Create a CSS rule that positions an element at the center of its parent container, both horizontally and vertically.

Solution

```
p {  
    position: fixed;  
    bottom: 0;  
    width: 100%;  
}
```

```
1 p {  
2     position: fixed;  
3     bottom: 0;  
4     width: 100%;  
5 }
```

In this solution, we set the position: fixed property on the p element to position it relative to the browser window. The bottom: 0 property positions the element at the bottom of the window. The width: 100% property makes the element span the full width of its parent container.

3.2 The display property

Here's an example that demonstrates the CSS3 display property and different layout types to create a three-column layout:

```
<style>
  .container {
    display: flex;
  }
  .column {
    width: 33%;
    margin: 0 20px;
    background-color: lightgray;
  }
</style>
<div class="container">
  <div class="column">Column 1</div>
  <div class="column">Column 2</div>
  <div class="column">Column 3</div>
</div>
```

```
1  <style>
2    .container {
3      display: flex;
4    }
5
6    .column {
7      width: 33%;
8      margin: 0 20px;
9      background-color: lightgray;
10   }
11 </style>
12
13 <div class="container">
14   <div class="column">Column 1</div>
15   <div class="column">Column 2</div>
16   <div class="column">Column 3</div>
17 </div>
```

In this example, we have a container `<div>` with a class of "container". By setting `display: flex` on the container, we enable flexbox layout, which allows us to create a flexible and responsive layout.

Inside the container, we have three `<div>` elements with a class of "column". Each column is set to `width: 33%`, which makes each column occupy one-third of the container's width. The `margin: 0 20px` adds a 20-pixel margin on the left and right sides of each column, creating spacing between them.

By using `display: flex` on the container, the columns are aligned horizontally by default. Flexbox automatically adjusts the size and positioning of the columns based on the available space and the defined widths.

You can adjust the width, margin, and other properties to customize the layout according to your needs. Additionally, you can explore other layout types such as `display: grid` or `display: inline-block` to achieve different types of layouts with the CSS3 `display` property.

Exercises

Exercise 1

Create a CSS rule that uses the `display: inline-block` property to create a horizontal navigation menu. Each menu item should be displayed as an inline block element with a background color and padding.

Solution

```
.nav-menu {  
    display: inline-block;  
}  
.nav-menu li {  
    display: inline-block;  
    background-color: lightgray;  
    padding: 10px;  
}
```

```
1 .nav-menu {  
2     display: inline-block;  
3 }  
4  
5 .nav-menu li {  
6     display: inline-block;  
7     background-color: lightgray;  
8     padding: 10px;  
9 }
```

In this solution, we set the `display: inline-block` property on the `.nav-menu` element to create a horizontal navigation menu. Each menu item (`li` element) is also set to `display: inline-block` to display them as inline block elements. We add a background color and padding to the menu items for styling.

Exercise 2

Create a CSS rule that uses the `display: grid` property to create a responsive grid layout. The grid should have three columns and three rows, with equal widths and heights for each grid cell.

Solution

```
.grid-container {  
    display: grid;  
    grid-template-columns: repeat(3, 1fr);  
    grid-template-rows: repeat(3, 1fr);  
}  
  
.grid-item {  
    background-color: lightgray;  
}
```

```
1 .grid-container {  
2     display: grid;  
3     grid-template-columns: repeat(3, 1fr);  
4     grid-template-rows: repeat(3, 1fr);  
5 }  
6  
7 .grid-item {  
8     background-color: lightgray;  
9 }
```

In this solution, we set the `display: grid` property on the `.grid-container` element to create a responsive grid layout. The `grid-template-columns` and `grid-template-rows` properties set the number and size of the grid columns and rows. In this example, we have three columns and three rows, each with equal widths and heights (1fr represents a fraction of the available space). The `.grid-item` class sets the background color for each grid cell.

Exercise 3

Create a CSS rule that uses the `display: flex` property to create a centered, vertically aligned layout. The layout should have a header at the top, a main content section in the middle, and a footer at the bottom.

Solution

```
.layout-container {  
    display: flex;  
    flex-direction: column;  
    height: 100vh;  
}  
.header {  
    background-color: lightgray;  
    padding: 20px;  
}  
.main-content {  
    flex-grow: 1;  
    background-color: white;  
    padding: 20px;  
}  
.footer {  
    background-color: lightgray;  
    padding: 20px;  
}
```

```
1 .layout-container {  
2   display: flex;  
3   flex-direction: column;  
4   height: 100vh;  
5 }  
6  
7 .header {  
8   background-color: lightgray;  
9   padding: 20px;  
10 }  
11  
12 .main-content {  
13   flex-grow: 1;  
14   background-color: white;  
15   padding: 20px;  
16 }  
17  
18 .footer {  
19   background-color: lightgray;  
20   padding: 20px;  
21 }
```

In this solution, we set the `display: flex` property on the `.layout-container` element to create a centered, vertically aligned layout. The `flex-direction: column` property positions the child elements in a vertical direction. The `.header`, `.main-content`, and `.footer` classes represent the header, main content section, and footer of the layout, respectively. We set background colors and padding for each element. The `flex-grow: 1` property on the `.main-content` class allows it to grow and occupy the remaining available space in the container.

3.3 The z-index property

Here's an example that demonstrates the CSS3 z-index property:

```
<style>
  .container {
    position: relative;
  }
  .box {
    position: absolute;
    width: 200px;
    height: 200px;
    padding: 20px;
    text-align: center;
    font-size: 24px;
  }
  .box1 {
    background-color: lightblue;
    z-index: 1;
  }
  .box2 {
    background-color: lightgreen;
    z-index: 2;
  }
</style>
<div class="container">
  <div class="box box1">Box 1</div>
  <div class="box box2">Box 2</div>
</div>
```

```
1 <style>
2   .container {
3     position: relative;
4   }
5
6   .box {
7     position: absolute;
8     width: 200px;
9     height: 200px;
10    padding: 20px;
11    text-align: center;
12    font-size: 24px;
13  }
14
15  .box1 {
16    background-color: lightblue;
17    z-index: 1;
18  }
19
20  .box2 {
21    background-color: lightgreen;
22    z-index: 2;
23  }
24 </style>
25
26 <div class="container">
27   <div class="box box1">Box 1</div>
28   <div class="box box2">Box 2</div>
29 </div>
30
```

In this example, we have a container `<div>` with a class of "container". We set `position: relative` on the container to establish it as the containing block for the absolutely positioned elements inside it. Inside the container, we have two `<div>` elements with classes "box1" and "box2". These boxes are absolutely positioned using `position: absolute` and given a width and height of 200 pixels. We also set padding, text alignment, and font size for styling purposes.

The `.box1` class is assigned a `background-color` of light blue and a `z-index` of 1. The `.box2` class is assigned a `background-color` of light green and a `z-index` of 2.

The `z-index` property determines the stacking order of positioned elements. Elements with a higher `z-index` value will be displayed on top of elements with a lower `z-index` value. In this example, the box

with the class "box2" has a higher z-index value, so it will be displayed on top of the box with the class "box1". You can adjust the z-index values and the styles of the boxes to see how the stacking order changes.

Exercises

Exercise 1

Create a CSS rule that positions two images on top of each other and use the z-index property to control their stacking order. The first image should have a higher z-index value and appear on top of the second image.

Solution

```
.image-container {  
    position: relative;  
}  
.image-container img {  
    position: absolute;  
    top: 0;  
    left: 0;  
}  
.image1 {  
    z-index: 2;  
}  
.image2 {  
    z-index: 1;  
}
```

```
1 .image-container {  
2     position: relative;  
3 }  
4  
5 .image-container img {  
6     position: absolute;  
7     top: 0;  
8     left: 0;  
9 }  
10  
11 .image1 {  
12     z-index: 2;  
13 }  
14  
15 .image2 {  
16     z-index: 1;  
17 }
```

In this solution, we create a container with the class "image-container" and set its position to relative. Inside the container, we have two images with classes "image1" and "image2". The images are absolutely positioned within the container, and the top: 0 and left:

0 properties position them at the top-left corner. The "image1" class is assigned a higher z-index value of 2, making it appear on top of the "image2" class, which has a z-index value of 1.

Exercise 2

Create a CSS rule that positions a navigation bar at the top of the page and a content section below it. Use the z-index property to ensure that the navigation bar appears on top of the content section.

Solution

```
.nav-bar {  
    position: fixed;  
    top: 0;  
    left: 0;  
    width: 100%;  
    z-index: 2;  
}  
.content-section {  
    margin-top: 50px; /* Adjust as needed to create  
    space for the navigation bar */  
    z-index: 1;  
}
```

```
1 .nav-bar {  
2     position: fixed;  
3     top: 0;  
4     left: 0;  
5     width: 100%;  
6     z-index: 2;  
7 }  
8  
9 .content-section {  
10    margin-top: 50px; /* Adjust as needed to create space for the navigation bar */  
11    z-index: 1;  
12 }
```

In this solution, we create a navigation bar with the class "nav-bar" and set its position to fixed to keep it at the top of the page. The top: 0 and left: 0 properties position it at the top-left corner, and the width: 100% property makes it span the full width of its parent container. The z-index property is set to 2 to ensure that the navigation bar appears on top of other elements on the page.

The content section below the navigation bar is given a class of "content-section". We create some space for the navigation bar using

margin-top and set the z-index property to 1 to ensure that it appears below the navigation bar.

Exercise 3

Create a CSS rule that positions multiple overlapping elements, such as `<div>` or `` elements, and use the `z-index` property to control their stacking order. Experiment with different `z-index` values to achieve the desired visual hierarchy.

Solution

```
.element {  
    position: absolute;  
}  
.element1 {  
    background-color: lightblue;  
    z-index: 2;  
}  
.element2 {  
    background-color: lightgreen;  
    z-index: 1;  
}  
.element3 {  
    background-color: lightyellow;  
    z-index: 3;  
}
```

```
1 .element {  
2     position: absolute;  
3 }  
4  
5 .element1 {  
6     background-color: lightblue;  
7     z-index: 2;  
8 }  
9  
10 .element2 {  
11     background-color: lightgreen;  
12     z-index: 1;  
13 }  
14  
15 .element3 {  
16     background-color: lightyellow;  
17     z-index: 3;  
18 }
```

In this solution, we create multiple elements with classes "element1", "element2", and "element3". Each element is given a unique

background color for visual distinction. The elements are absolutely positioned using position: absolute to allow them to overlap. By assigning different z-index values to each element, we control their stacking order. The "element3" class has the highest z-index value of 3, making it appear on top of both "element1" and "element2". The "element1" class has a z-index value of 2, making it appear on top of "element2".

Chapter 4: Modern CSS layouts

4.1 Flexbox Layout

Here's an example of a CSS3 Flexbox layout:

```
<style>
  .container {
    display: flex;
    justify-content: space-between;
  }
  .item {
    background-color: lightblue;
    padding: 20px;
  }
</style>
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

```
1  <style>
2    .container {
3      display: flex;
4      justify-content: space-between;
5    }
6
7    .item {
8      background-color: lightblue;
9      padding: 20px;
10   }
11  </style>
12
13 <div class="container">
14   <div class="item">Item 1</div>
15   <div class="item">Item 2</div>
16   <div class="item">Item 3</div>
17 </div>
18
```

In this example, we use CSS3 Flexbox to create a simple layout with three items in a row.

The `<div>` with a class of "container" is the parent container that holds the items. We apply `display: flex` to the container to enable Flexbox layout.

The `justify-content: space-between` property is used to distribute the items along the main axis (horizontal axis) with equal spacing between them. This pushes the first item to the left edge, the last item to the right edge, and automatically adjusts the spacing between the items.

Each item is represented by a `<div>` with a class of "item". We apply some styles to the items, such as a background color and padding, to visually distinguish them.

By using Flexbox, the items will automatically adjust their width and spacing based on the available space in the container. This allows for responsive and dynamic layouts that adapt to different screen sizes and content lengths.

You can customize the styles and add more items to create more complex Flexbox layouts. Additionally, there are various properties and values in Flexbox, such as `flex-direction`, `align-items`, and `flex-wrap`, that can be used to achieve different layouts and alignments based on your specific needs.

Exercises

Exercise 1

Create a Flexbox layout with a header, three columns, and a footer. The columns should have equal width and be vertically aligned in the center.

Solution

```
<style>
  .container {
    display: flex;
    flex-direction: column;
    height: 100vh;
  }
  .header,
  .footer {
    background-color: lightblue;
    padding: 20px;
  }
  .column {
    flex: 1;
    display: flex;
    justify-content: center;
    align-items: center;
    background-color: lightgray;
    padding: 20px;
  }
</style>
<div class="container">
  <header class="header">Header</header>
  <div class="column">Column 1</div>
  <div class="column">Column 2</div>
  <div class="column">Column 3</div>
  <footer class="footer">Footer</footer>
</div>
```

```
1 <style>
2   .container {
3     display: flex;
4     flex-direction: column;
5     height: 100vh;
6   }
7
8   .header,
9   .footer {
10    background-color: lightblue;
11    padding: 20px;
12  }
13
14  .column {
15    flex: 1;
16    display: flex;
17    justify-content: center;
18    align-items: center;
19    background-color: lightgray;
20    padding: 20px;
21  }
22 </style>
23
24 <div class="container">
25   <header class="header">Header</header>
26   <div class="column">Column 1</div>
27   <div class="column">Column 2</div>
28   <div class="column">Column 3</div>
29   <footer class="footer">Footer</footer>
30 </div>
31
```

Exercise 2

Create a Flexbox layout for a product gallery. Display six product items in two rows and three columns. Each item should have equal width and height.

Solution

```
<style>
  .container {
    display: flex;
    flex-wrap: wrap;
  }
  .product {
    flex-basis: 33.33%;
    background-color: lightblue;
    padding: 20px;
    box-sizing: border-box;
  }
</style>
<div class="container">
  <div class="product">Product 1</div>
  <div class="product">Product 2</div>
  <div class="product">Product 3</div>
  <div class="product">Product 4</div>
  <div class="product">Product 5</div>
  <div class="product">Product 6</div>
</div>
```

```
1 <style>
2   .container {
3     display: flex;
4     flex-wrap: wrap;
5   }
6
7   .product {
8     flex-basis: 33.33%;
9     background-color: lightblue;
10    padding: 20px;
11    box-sizing: border-box;
12  }
13 </style>
14
15 <div class="container">
16   <div class="product">Product 1</div>
17   <div class="product">Product 2</div>
18   <div class="product">Product 3</div>
19   <div class="product">Product 4</div>
20   <div class="product">Product 5</div>
21   <div class="product">Product 6</div>
22 </div>
23
```

Exercise 3

Create a Flexbox layout for a pricing table. Display three pricing plans side by side. Each plan should have a title, price, and features. Ensure that the columns have equal width and the features are vertically aligned.

Solution

```
<style>
  .container {
    display: flex;
  }
  .plan {
    flex: 1;
    background-color: lightblue;
    padding: 20px;
    box-sizing: border-box;
  }
  .title {
    font-size: 20px;
    font-weight: bold;
  }
  .price {
    font-size: 18px;
  }
  .features {
    margin-top: 10px;
    list-style-type: none;
    padding: 0;
  }
</style>
<div class="container">
  <div class="plan">
    <h3 class="title">Plan 1</h3>
    <p class="price">$10/month</p>
```

```
<ul class="features">
    <li>Feature 1</li>
    <li>Feature 2</li>
    <li>Feature 3</li>
</ul>
</div>
<div class="plan">
    <h3 class="title">Plan 2</h3>
    <p class="price">$20/month</p>
    <ul class="features">
        <li>Feature 1</li>
        <li>Feature 2</li>
        <li>Feature 3</li>
    </ul>
</div>
<div class="plan">
    <h3 class="title">Plan 3</h3>
    <p class="price">$30/month</p>
    <ul class="features">
        <li>Feature 1</li>
        <li>Feature 2</li>
        <li>Feature 3</li>
    </ul>
</div>
</div>
```

```
1 <style>
2   .container {
3     display: flex;
4   }
5
6   .plan {
7     flex: 1;
8     background-color: lightblue;
9     padding: 20px;
10    box-sizing: border-box;
11  }
12
13  .title {
14    font-size: 20px;
15    font-weight: bold;
16  }
17
18  .price {
19    font-size: 18px;
20  }
21
22  .features {
23    margin-top: 10px;
24    list-style-type: none;
25    padding: 0;
26  }
27 </style>
28
```

```
29 <div class="container">
30   <div class="plan">
31     <h3 class="title">Plan 1</h3>
32     <p class="price">$10/month</p>
33     <ul class="features">
34       <li>Feature 1</li>
35       <li>Feature 2</li>
36       <li>Feature 3</li>
37     </ul>
38   </div>
39   <div class="plan">
40     <h3 class="title">Plan 2</h3>
41     <p class="price">$20/month</p>
42     <ul class="features">
43       <li>Feature 1</li>
44       <li>Feature 2</li>
45       <li>Feature 3</li>
46     </ul>
47   </div>
48   <div class="plan">
49     <h3 class="title">Plan 3</h3>
50     <p class="price">$30/month</p>
51     <ul class="features">
52       <li>Feature 1</li>
53       <li>Feature 2</li>
54       <li>Feature 3</li>
55     </ul>
56   </div>
57 </div>
```


4.2 Grid Layout

Here's an example of a CSS3 Grid layout:

```
<style>
  .container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-gap: 20px;
  }
  .item {
    background-color: lightblue;
    padding: 20px;
  }
</style>
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
  <div class="item">Item 4</div>
  <div class="item">Item 5</div>
  <div class="item">Item 6</div>
</div>
```

```
1 <style>
2   .container {
3     display: grid;
4     grid-template-columns: repeat(3, 1fr);
5     grid-gap: 20px;
6   }
7
8   .item {
9     background-color: lightblue;
10    padding: 20px;
11  }
12 </style>
13
14 <div class="container">
15   <div class="item">Item 1</div>
16   <div class="item">Item 2</div>
17   <div class="item">Item 3</div>
18   <div class="item">Item 4</div>
19   <div class="item">Item 5</div>
20   <div class="item">Item 6</div>
21 </div>
```

In this example, we use CSS3 Grid to create a simple layout with a grid of six items.

The `<div>` with a class of "container" is the parent container that holds the items. We apply `display: grid` to the container to enable Grid layout.

The `grid-template-columns` property is used to define the columns of the grid. In this case, we use the `repeat()` function to create three columns, each with a width of `1fr`, which means they will share the available space equally. You can adjust the number of columns and their sizes to fit your desired layout.

The `grid-gap` property sets the gap between grid items. In this example, we set it to `20px` to create some spacing between the items. Feel free to adjust this value to suit your design preferences.

Each item is represented by a `<div>` with a class of "item". We apply some styles to the items, such as a background color and padding, to visually distinguish them.

By using CSS3 Grid, the items will be automatically positioned in a grid layout based on the defined columns and any specified gaps.

This allows for creating complex and responsive layouts with ease. You can customize the styles, adjust the number of columns, and add more items to create more complex Grid layouts. Additionally, there are various Grid properties and values, such as `grid-template-rows`, `grid-area`, and `grid-template-areas`, that can be used to achieve different layouts and alignments based on your specific needs.

Exercises

Exercise 1

Create a Grid layout with a header, three columns, and a footer. The header should span across all three columns, and the footer should span across the last two columns.

Solution

```
<style>
  .container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-gap: 20px;
  }
  .header {
    grid-column: 1 / span 3;
    background-color: lightblue;
    padding: 20px;
  }
  .column {
    background-color: lightgray;
    padding: 20px;
  }
  .footer {
    grid-column: 2 / span 2;
    background-color: lightblue;
    padding: 20px;
  }
</style>
<div class="container">
  <header class="header">Header</header>
  <div class="column">Column 1</div>
  <div class="column">Column 2</div>
  <div class="column">Column 3</div>
  <footer class="footer">Footer</footer>
```

```
</div>
```

```
1  <style>
2    .container {
3      display: grid;
4      grid-template-columns: repeat(3, 1fr);
5      grid-gap: 20px;
6    }
7
8    .header {
9      grid-column: 1 / span 3;
10   background-color: lightblue;
11   padding: 20px;
12 }
13
14   .column {
15     background-color: lightgray;
16     padding: 20px;
17 }
18
19   .footer {
20     grid-column: 2 / span 2;
21     background-color: lightblue;
22     padding: 20px;
23 }
24 </style>
25
26 <div class="container">
27   <header class="header">Header</header>
28   <div class="column">Column 1</div>
29   <div class="column">Column 2</div>
30   <div class="column">Column 3</div>
31   <footer class="footer">Footer</footer>
32 </div>
33
```

Exercise 2

Create a Grid layout for a photo gallery. Display six photos in a 2x3 grid. Each photo should have equal width and height.

Solution

```
<style>
  .container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-template-rows: repeat(2, 1fr);
    grid-gap: 20px;
  }
  .photo {
    background-color: lightblue;
    padding: 20px;
  }
</style>
<div class="container">
  <div class="photo">Photo 1</div>
  <div class="photo">Photo 2</div>
  <div class="photo">Photo 3</div>
  <div class="photo">Photo 4</div>
  <div class="photo">Photo 5</div>
  <div class="photo">Photo 6</div>
</div>
```

```
1 <style>
2   .container {
3     display: grid;
4     grid-template-columns: repeat(3, 1fr);
5     grid-template-rows: repeat(2, 1fr);
6     grid-gap: 20px;
7   }
8
9   .photo {
10    background-color: lightblue;
11    padding: 20px;
12  }
13 </style>
14
15 <div class="container">
16   <div class="photo">Photo 1</div>
17   <div class="photo">Photo 2</div>
18   <div class="photo">Photo 3</div>
19   <div class="photo">Photo 4</div>
20   <div class="photo">Photo 5</div>
21   <div class="photo">Photo 6</div>
22 </div>
23
```

Exercise 3

Create a Grid layout for a dashboard. Display four widgets in a 2x2 grid. Each widget should have a specific size and be positioned in different grid cells.

Solution

```
<style>
  .container {
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-template-rows: 200px 200px;
    grid-gap: 20px;
  }
  .widget {
    background-color: lightblue;
    padding: 20px;
  }
  .widget1 {
    grid-column: 1;
    grid-row: 1;
  }
  .widget2 {
    grid-column: 2;
    grid-row: 1;
  }
  .widget3 {
    grid-column: 1;
    grid-row: 2;
  }
  .widget4 {
    grid-column: 2;
    grid-row: 2;
  }
}
```

```
</style>

<div class="container">
    <div class="widget widget1">Widget 1</div>
    <div class="widget widget2">Widget 2</div>
    <div class="widget widget3">Widget 3</div>
    <div class="widget widget4">Widget 4</div>
</div>
```

```
1  <style>
2      .container {
3          display: grid;
4          grid-template-columns: 1fr 1fr;
5          grid-template-rows: 200px 200px;
6          grid-gap: 20px;
7      }
8
9      .widget {
10         background-color: lightblue;
11         padding: 20px;
12     }
13
14     .widget1 {
15         grid-column: 1;
16         grid-row: 1;
17     }
18
19     .widget2 {
20         grid-column: 2;
21         grid-row: 1;
22     }
23
24     .widget3 {
25         grid-column: 1;
26         grid-row: 2;
27     }
28
29     .widget4 {
30         grid-column: 2;
31         grid-row: 2;
32     }
33 </style>
34
35 <div class="container">
36     <div class="widget widget1">Widget 1</div>
37     <div class="widget widget2">Widget 2</div>
38     <div class="widget widget3">Widget 3</div>
39     <div class="widget widget4">Widget 4</div>
40 </div>
```

4.3 Multi-column Layout

Here's an example of a CSS3 Multi-column Layout:

```
<style>
  .container {
    column-count: 2;
    column-gap: 20px;
  }
  p {
    margin-bottom: 20px;
  }
</style>
<div class="container">
  <p>
    Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Sed sed lectus
    mattis, gravida leo a, dignissim metus.
  </p>
  <p>
    Nullam consequat mi sit amet lacus congue, non
    blandit odio facilisis. Nulla
    facilisi. Nam id metus ut justo cursus viverra.
  </p>
  <p>
    Phasellus nec diam non nisl pellentesque
    rhoncus. Ut nec purus ut mi
    ultricies varius.
  </p>
```

<p>Aenean commodo velit id velit eleifend, eu viverra est congue.</p>

<p>

Curabitur in enim et lectus aliquam aliquam.
Integer ut sem mollis, interdum
risus eget, posuere leo.

</p>

<p>

Morbi non metus id nulla faucibus gravida a non
tortor. Donec tristique,
metus ut lobortis suscipit, felis risus commodo
neque, id imperdiet lacus
dolor quis nunc.

</p>

</div>

```
1 <style>
2   .container {
3     column-count: 2;
4     column-gap: 20px;
5   }
6
7   p {
8     margin-bottom: 20px;
9   }
10 </style>
11
12 <div class="container">
13   <p>
14     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sed lectus
15     mattis, gravida leo a, dignissim metus.
16   </p>
17   <p>
18     Nullam consequat mi sit amet lacus congue, non blandit odio facilisis. Nulla
19     facilisi. Nam id metus ut justo cursus viverra.
20   </p>
21   <p>
22     Phasellus nec diam non nisl pellentesque rhoncus. Ut nec purus ut mi
23     ultricies varius.
24   </p>
25   <p>Aenean commodo velit id velit eleifend, eu viverra est congue.</p>
26   <p>
27     Curabitur in enim et lectus aliquam aliquam. Integer ut sem mollis, interdum
28     risus eget, posuere leo.
29   </p>
30   <p>
31     Morbi non metus id nulla faucibus gravida a non tortor. Donec tristique,
32     metus ut lobortis suscipit, felis risus commodo neque, id imperdiet lacus
33     dolor quis nunc.
34   </p>
35 </div>
36
```

In this example, we use CSS3 Multi-column Layout to create a two-column layout for a container with multiple paragraphs.

The `<div>` with a class of "container" is the parent container that holds the paragraphs. We apply the `column-count` property to the container and set it to 2 to create two columns.

The `column-gap` property is used to specify the gap between columns. In this case, we set it to 20px to create some spacing between the columns. You can adjust this value according to your design preferences.

Each paragraph is represented by a `<p>` element. We apply some styles to the paragraphs, such as a `margin-bottom`, to create spacing

between them and improve readability.

By using CSS3 Multi-column Layout, the paragraphs will be automatically arranged in columns within the container. The content will flow from one column to the next, creating a multi-column layout. You can customize the styles, adjust the number of columns, and modify the gap between columns to achieve different layouts based on your specific needs. Additionally, there are additional properties, such as column-width, column-rule, and column-span, that can be used to further control the appearance and behavior of multi-column layouts.

Exercises

Exercise 1

Create a multi-column layout for a news article. The article should have three columns, and each column should have a width of 300px. Apply a column gap of 20px.

Solution

```
<style>
  .container {
    column-count: 3;
    column-width: 300px;
    column-gap: 20px;
  }
  .news-article {
    /* Styles for the news article */
  }
</style>
<div class="container">
  <article class="news-article">
    <!-- Content of the news article goes here -->
  </article>
</div>
```

```
1  <style>
2    .container {
3      column-count: 3;
4      column-width: 300px;
5      column-gap: 20px;
6    }
7
8    .news-article {
9      /* Styles for the news article */
10   }
11  </style>
12
13 <div class="container">
14   <article class="news-article">
15     <!-- Content of the news article goes here -->
16   </article>
17 </div>
18
```

Exercise 2

Create a multi-column layout for a recipe list. Display six recipe items in two columns. Each item should have equal width, and there should be no gap between the columns.

Solution

```
<style>
  .container {
    column-count: 2;
    column-gap: 0;
  }
  .recipe-item {
    /* Styles for the recipe items */
  }
</style>
<div class="container">
  <ul class="recipe-list">
    <li class="recipe-item">Recipe 1</li>
    <li class="recipe-item">Recipe 2</li>
    <li class="recipe-item">Recipe 3</li>
    <li class="recipe-item">Recipe 4</li>
    <li class="recipe-item">Recipe 5</li>
    <li class="recipe-item">Recipe 6</li>
  </ul>
</div>
```

```
1 <style>
2   .container {
3     column-count: 2;
4     column-gap: 0;
5   }
6
7   .recipe-item {
8     /* Styles for the recipe items */
9   }
10 </style>
11
12 <div class="container">
13   <ul class="recipe-list">
14     <li class="recipe-item">Recipe 1</li>
15     <li class="recipe-item">Recipe 2</li>
16     <li class="recipe-item">Recipe 3</li>
17     <li class="recipe-item">Recipe 4</li>
18     <li class="recipe-item">Recipe 5</li>
19     <li class="recipe-item">Recipe 6</li>
20   </ul>
21 </div>
22
```

Exercise 3

Create a multi-column layout for a product catalog. Display twelve product cards in three columns. Each card should have a width of 250px, and there should be a column gap of 10px.

Solution

```
<style>
  .container {
    column-count: 3;
    column-width: 250px;
    column-gap: 10px;
  }
  .product-card {
    /* Styles for the product cards */
  }
</style>
<div class="container">
  <div class="product-catalog">
    <!-- Product cards go here -->
  </div>
</div>
```

```
1  <style>
2    .container {
3      column-count: 3;
4      column-width: 250px;
5      column-gap: 10px;
6    }
7
8    .product-card {
9      /* Styles for the product cards */
10   }
11  </style>
12
13 <div class="container">
14   <div class="product-catalog">
15     <!-- Product cards go here -->
16   </div>
17 </div>
```

4.4 Shapes

Here's an example of CSS3 Shapes:

```
<style>

  .shape-container {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 200px;
  }

  .circle {
    width: 200px;
    height: 200px;
    background-color: red;
    border-radius: 50%;
  }

  .triangle {
    width: 0;
    height: 0;
    border-left: 100px solid transparent;
    border-right: 100px solid transparent;
    border-bottom: 173px solid blue;
  }

  .rectangle {
    width: 200px;
    height: 100px;
    background-color: green;
  }

</style>
```

```

</style>

<div class="shape-container">
  <div class="circle"></div>
  <div class="triangle"></div>
  <div class="rectangle"></div>
</div>

```

```

1  <style>
2    .shape-container {
3      display: flex;
4      justify-content: center;
5      align-items: center;
6      height: 200px;
7    }
8
9    .circle {
10      width: 200px;
11      height: 200px;
12      background-color: red;
13      border-radius: 50%;
14    }
15
16   .triangle {
17     width: 0;
18     height: 0;
19     border-left: 100px solid transparent;
20     border-right: 100px solid transparent;
21     border-bottom: 173px solid blue;
22   }
23
24   .rectangle {
25     width: 200px;
26     height: 100px;
27     background-color: green;
28   }
29 </style>
30
31 <div class="shape-container">
32   <div class="circle"></div>
33   <div class="triangle"></div>
34   <div class="rectangle"></div>
35 </div>

```

In this example, we create three different shapes using CSS3 properties.

The parent container `<div>` with a class of "shape-container" is used to hold the shapes. We apply some CSS properties to center the shapes both vertically and horizontally within the container.

The first shape, represented by the <div> with a class of "circle", is created using the border-radius property. By setting the border-radius to 50%, we achieve a circular shape. Adjust the width and height to control the size of the circle.

The second shape, represented by the <div> with a class of "triangle", is created using CSS border properties. We set the width and height of the element to 0 to create a triangle shape. By manipulating the border properties (border-left, border-right, and border-bottom) with different sizes and colors, we can define the shape of the triangle.

The third shape, represented by the <div> with a class of "rectangle", is a simple rectangle shape. We set the width and height of the element to the desired dimensions, and the shape is achieved.

You can customize the sizes, colors, and other CSS properties to create different shapes based on your specific needs. Additionally, CSS3 provides advanced properties and techniques for creating more complex shapes, such as polygons and ellipses, using clip-path and SVG.

shape-outside

- The shape-outside property allows you to define a shape that content should wrap around.
- You can use various values to define the shape, such as circle(), ellipse(), inset(), polygon(), and url().
- This property is commonly used with floated elements to create text wrapping effects around non-rectangular shapes.

Exercises

Exercise 1

Create a text wrap around an image with a polygon shape. Use the shape-outside property to define a custom polygon shape that the text should flow around.

Solution

```
<style>
    .text-container {
        width: 400px;
        margin: 0 auto;
    }
    .float-image {
        float: left;
        width: 200px;
        height: 200px;
        shape-outside: polygon(50% 0%, 100% 50%, 50%
100%, 0% 50%);
        margin: 0 20px 20px 0;
    }
    .text {
        text-align: justify;
    }
</style>
<div class="text-container">
    
    <p class="text">
        Lorem ipsum dolor sit amet, consectetur
        adipiscing elit. Nullam vitae neque
        non diam lacinia convallis. Aliquam auctor erat
        sit amet magna sagittis, in
        pharetra nulla congue. Vivamus bibendum mi ac
        ex varius, id molestie tortor
    </p>
</div>
```

mattis. Sed in metus vel est porttitor gravida.
Etiam ac mattis mauris, ac
sollicitudin est.

</p>

</div>

```
1 <style>
2   .text-container {
3     width: 400px;
4     margin: 0 auto;
5   }
6
7   .float-image {
8     float: left;
9     width: 200px;
10    height: 200px;
11    shape-outside: polygon(50% 0%, 100% 50%, 50% 100%, 0% 50%);
12    margin: 0 20px 20px 0;
13  }
14
15  .text {
16    text-align: justify;
17  }
18 </style>
19
20 <div class="text-container">
21   
22   <p class="text">
23     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam vitae neque
24     non diam lacinia convallis. Aliquam auctor erat sit amet magna sagittis, in
25     pharetra nulla congue. Vivamus bibendum mi ac ex varius, id molestie tortor
26     mattis. Sed in metus vel est porttitor gravida. Etiam ac mattis mauris, ac
27     sollicitudin est.
28   </p>
29 </div>
30
```

Exercise 2

Create a text wrap around an image with an ellipse shape. Use the shape-outside property to define an ellipse shape that the text should flow around.

Solution

```
<style>
    .text-container {
        width: 400px;
        margin: 0 auto;
    }
    .float-image {
        float: left;
        width: 200px;
        height: 200px;
        shape-outside: ellipse(50% 50% at 50% 50%);
        margin: 0 20px 20px 0;
    }
    .text {
        text-align: justify;
    }
</style>
<div class="text-container">
    
    <p class="text">
        Lorem ipsum dolor sit amet, consectetur
        adipiscing elit. Nullam vitae neque
        non diam lacinia convallis. Aliquam auctor erat
        sit amet magna sagittis, in
        pharetra nulla congue. Vivamus bibendum mi ac
        ex varius, id molestie tortor
    </p>
</div>
```

mattis. Sed in metus vel est porttitor gravida.
Etiam ac mattis mauris, ac
sollicitudin est.

</p>

</div>

```
1 <style>
2   .text-container {
3     width: 400px;
4     margin: 0 auto;
5   }
6
7   .float-image {
8     float: left;
9     width: 200px;
10    height: 200px;
11    shape-outside: ellipse(50% 50% at 50% 50%);
12    margin: 0 20px 20px 0;
13  }
14
15  .text {
16    text-align: justify;
17  }
18 </style>
19
20 <div class="text-container">
21   
22   <p class="text">
23     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam vitae neque
24     non diam lacinia convallis. Aliquam auctor erat sit amet magna sagittis, in
25     pharetra nulla congue. Vivamus bibendum mi ac ex varius, id molestie tortor
26     mattis. Sed in metus vel est porttitor gravida. Etiam ac mattis mauris, ac
27     sollicitudin est.
28   </p>
29 </div>
30
```

Exercise 3

Create a text wrap around a non-rectangular image. Use the shape-outside property to define a shape that matches the outline of the image, allowing the text to flow around its irregular edges.

Solution

```
<style>
  .text-container {
    width: 400px;
    margin: 0 auto;
  }
  .float-image {
    float: left;
    width: 200px;
    height: 200px;
    shape-outside: url(image-mask.svg);
    margin: 0 20px 20px 0;
  }
  .text {
    text-align: justify;
  }
</style>
<div class="text-container">
  
  <p class="text">
    Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Nullam vitae neque
    non diam lacinia convallis. Aliquam auctor erat
    sit amet magna sagittis, in
    pharetra nulla congue. Vivamus bibendum mi ac
    ex varius, id molestie tortor
  </p>
</div>
```

mattis. Sed in metus vel est porttitor gravida.
Etiam ac mattis mauris, ac
sollicitudin est.

</p>

</div>

```
1 <style>
2   .text-container {
3     width: 400px;
4     margin: 0 auto;
5   }
6
7   .float-image {
8     float: left;
9     width: 200px;
10    height: 200px;
11    shape-outside: url(image-mask.svg);
12    margin: 0 20px 20px 0;
13  }
14
15  .text {
16    text-align: justify;
17  }
18 </style>
19
20 <div class="text-container">
21   
22   <p class="text">
23     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam vitae neque
24     non diam lacinia convallis. Aliquam auctor erat sit amet magna sagittis, in
25     pharetra nulla congue. Vivamus bibendum mi ac ex varius, id molestie tortor
26     mattis. Sed in metus vel est porttitor gravida. Etiam ac mattis mauris, ac
27     sollicitudin est.
28   </p>
29 </div>
30
```

shape-inside

- The shape-inside property allows you to define a shape that content should fit within.
- It can be used to create interesting text layouts within custom shapes.
- Like shape-outside, it supports values such as circle(), ellipse(), inset(), polygon(), and url().

Exercises

Exercise 1

Create a shape inside a text container using the `shape-inside` property. Use a custom polygon shape that the text should wrap around.

Solution

```
<style>
  .text-container {
    width: 400px;
    margin: 0 auto;
    shape-inside: polygon(0% 0%, 100% 0%, 80% 100%,
20% 100%);
  }
  .text {
    text-align: justify;
  }
</style>
<div class="text-container">
  <p class="text">
    Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Nullam vitae neque
    non diam lacinia convallis. Aliquam auctor erat
    sit amet magna sagittis, in
    pharetra nulla congue. Vivamus bibendum mi ac
    ex varius, id molestie tortor
    mattis. Sed in metus vel est porttitor gravida.
    Etiam ac mattis mauris, ac
    sollicitudin est.
  </p>
</div>
```

```
1 <style>
2   .text-container {
3     width: 400px;
4     margin: 0 auto;
5     shape-inside: polygon(0% 0%, 100% 0%, 80% 100%, 20% 100%);
6   }
7
8   .text {
9     text-align: justify;
10  }
11 </style>
12
13 <div class="text-container">
14   <p class="text">
15     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam vitae neque
16     non diam lacinia convallis. Aliquam auctor erat sit amet magna sagittis, in
17     pharetra nulla congue. Vivamus bibendum mi ac ex varius, id molestie tortor
18     mattis. Sed in metus vel est porttitor gravida. Etiam ac mattis mauris, ac
19     sollicitudin est.
20   </p>
21 </div>
```

Exercise 2

Create a shape inside a text container using the shape-inside property. Use a circle shape that the text should wrap around.

Solution

```
<style>
    .text-container {
        width: 400px;
        margin: 0 auto;
        shape-inside: circle(50%);
    }
    .text {
        text-align: justify;
    }
</style>
<div class="text-container">
    <p class="text">
        Lorem ipsum dolor sit amet, consectetur
        adipiscing elit. Nullam vitae neque
        non diam lacinia convallis. Aliquam auctor erat
        sit amet magna sagittis, in
        pharetra nulla congue. Vivamus bibendum mi ac
        ex varius, id molestie tortor
        mattis. Sed in metus vel est porttitor gravida.
        Etiam ac mattis mauris, ac
        sollicitudin est.
    </p>
</div>
```

```
1 <style>
2   .text-container {
3     width: 400px;
4     margin: 0 auto;
5     shape-inside: circle(50%);
6   }
7
8   .text {
9     text-align: justify;
10  }
11 </style>
12
13 <div class="text-container">
14   <p class="text">
15     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam vitae neque
16     non diam lacinia convallis. Aliquam auctor erat sit amet magna sagittis, in
17     pharetra nulla congue. Vivamus bibendum mi ac ex varius, id molestie tortor
18     mattis. Sed in metus vel est porttitor gravida. Etiam ac mattis mauris, ac
19     sollicitudin est.
20   </p>
21 </div>
```

Exercise 3

Create a shape inside a text container using the shape-inside property. Use an image mask to define the shape of the text container.

Solution

```
<style>
    .text-container {
        width: 400px;
        margin: 0 auto;
        shape-inside: url(image-mask.svg);
    }
    .text {
        text-align: justify;
    }
</style>
<div class="text-container">
    <p class="text">
        Lorem ipsum dolor sit amet, consectetur
        adipiscing elit. Nullam vitae neque
        non diam lacinia convallis. Aliquam auctor erat
        sit amet magna sagittis, in
        pharetra nulla congue. Vivamus bibendum mi ac
        ex varius, id molestie tortor
        mattis. Sed in metus vel est porttitor gravida.
        Etiam ac mattis mauris, ac
        sollicitudin est.
    </p>
</div>
```

```
1 <style>
2   .text-container {
3     width: 400px;
4     margin: 0 auto;
5     shape-inside: url(image-mask.svg);
6   }
7
8   .text {
9     text-align: justify;
10  }
11 </style>
12
13 <div class="text-container">
14   <p class="text">
15     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam vitae neque
16     non diam lacinia convallis. Aliquam auctor erat sit amet magna sagittis, in
17     pharetra nulla congue. Vivamus bibendum mi ac ex varius, id molestie tortor
18     mattis. Sed in metus vel est porttitor gravida. Etiam ac mattis mauris, ac
19     sollicitudin est.
20   </p>
21 </div>
22
```

shape-margin

- The shape-margin property allows you to define a margin around the shape created with shape-outside or shape-inside.
- It determines the space between the shape and the content that wraps or fits within it.
- By adjusting the shape-margin, you can control the spacing between the shape and the surrounding content.

Exercises

Exercise 1

Create a shape with a margin using the shape-margin property.

Apply a triangle shape to an element and set a margin around it.

Solution

```
<style>
  .shape-container {
    width: 200px;
    height: 200px;
    background-color: red;
    shape-outside: polygon(0% 0%, 100% 0%, 50%
100%);
    shape-margin: 20px;
  }
</style>
<div class="shape-container"></div>
```

```
1 <style>
2   .shape-container {
3     width: 200px;
4     height: 200px;
5     background-color: red;
6     shape-outside: polygon(0% 0%, 100% 0%, 50% 100%);
7     shape-margin: 20px;
8   }
9 </style>
10
11 <div class="shape-container"></div>
12
```

Exercise 2

Create a shape with a margin using the shape-margin property.

Apply a circle shape to an element and set a margin around it.

Solution

```
<style>
  .shape-container {
    width: 200px;
    height: 200px;
    background-color: blue;
    shape-outside: circle(50%);
    shape-margin: 20px;
  }
</style>
<div class="shape-container"></div>
```

```
1 <style>
2   .shape-container {
3     width: 200px;
4     height: 200px;
5     background-color: blue;
6     shape-outside: circle(50%);
7     shape-margin: 20px;
8   }
9 </style>
10
11 <div class="shape-container"></div>
12
```

Exercise 3

Create a shape with a margin using the shape-margin property.
Apply a custom polygon shape to an element and set a margin
around it.

Solution

```
<style>
  .shape-container {
    width: 200px;
    height: 200px;
    background-color: green;
    shape-outside: polygon(0% 0%, 100% 0%, 80%
100%, 20% 100%);
    shape-margin: 20px;
  }
</style>
<div class="shape-container"></div>
```

```
1 <style>
2   .shape-container {
3     width: 200px;
4     height: 200px;
5     background-color: green;
6     shape-outside: polygon(0% 0%, 100% 0%, 80% 100%, 20% 100%);
7     shape-margin: 20px;
8   }
9 </style>
10 <div class="shape-container"></div>
11
```

clip-path

- Although not specifically part of CSS3 Shapes, the `clip-path` property is often used in conjunction with `shape-outside` to create more complex shapes.
- It allows you to define a clipping path that clips an element to a specified shape.
- The shape can be created using various functions and values, such as `circle()`, `ellipse()`, `polygon()`, and more.

Exercises

Exercise 1

Create a clipped shape using the clip-path property. Apply a polygon shape to an element and clip it to show only a specific portion.

Solution

```
<style>
  .shape-container {
    width: 200px;
    height: 200px;
    background-color: red;
    clip-path: polygon(0% 0%, 100% 0%, 100% 50%, 0% 100%);
  }
</style>
<div class="shape-container"></div>
```

```
1 <style>
2   .shape-container {
3     width: 200px;
4     height: 200px;
5     background-color: red;
6     clip-path: polygon(0% 0%, 100% 0%, 100% 50%, 0% 100%);
7   }
8 </style>
9
10 <div class="shape-container"></div>
11
```

Exercise 2

Create a clipped shape using the clip-path property. Apply a circle shape to an element and clip it to show only a circular portion.

Solution

```
<style>
  .shape-container {
    width: 200px;
    height: 200px;
    background-color: blue;
    clip-path: circle(50% at 50% 50%);
  }
</style>
<div class="shape-container"></div>
```

```
1 <style>
2   .shape-container {
3     width: 200px;
4     height: 200px;
5     background-color: blue;
6     clip-path: circle(50% at 50% 50%);
7   }
8 </style>
9
10 <div class="shape-container"></div>
```

Exercise 3

Create a clipped shape using the clip-path property. Apply a custom path shape to an element and clip it to show only a specific portion.

Solution

```
<style>
  .shape-container {
    width: 200px;
    height: 200px;
    background-color: green;
    clip-path: path("M0 0 L100 0 L50 100 Z");
  }
</style>
<div class="shape-container"></div>
```

```
1 <style>
2   .shape-container {
3     width: 200px;
4     height: 200px;
5     background-color: green;
6     clip-path: path("M0 0 L100 0 L50 100 Z");
7   }
8 </style>
9
10 <div class="shape-container"></div>
```

4.5 Scroll Snap

CSS3 Scroll Snap is a feature that provides control over the scrolling behavior of a container element, allowing you to create smooth and precise snapping effects when scrolling through content. It enables you to define specific snap points within a scroll container, making it easier for users to navigate through sections or items.

Exercises

Exercise 1

Create a horizontal image carousel using CSS3 Scroll Snap. The carousel should display multiple images horizontally, and each image should snap to a visible position when scrolling.

Solution

```
<style>
  .carousel {
    display: flex;
    overflow-x: scroll;
    scroll-snap-type: x mandatory;
  }
  .carousel img {
    width: 100%;
    scroll-snap-align: start;
  }
</style>
<div class="carousel">
  
  
  
</div>
```

```
1 <style>
2   .carousel {
3     display: flex;
4     overflow-x: scroll;
5     scroll-snap-type: x mandatory;
6   }
7
8   .carousel img {
9     width: 100%;
10    scroll-snap-align: start;
11  }
12 </style>
13
14 <div class="carousel">
15   
16   
17   
18 </div>
```


Exercise 2

Create a vertical scrollable section with scroll snap. Each section should contain some content, and when scrolling vertically, each section should snap to a visible position. Add navigation buttons or indicators to allow users to navigate between the sections.

Solution

```
<style>
  .scroll-section {
    height: 100vh;
    overflow-y: scroll;
    scroll-snap-type: y mandatory;
  }
  .section {
    height: 100vh;
    scroll-snap-align: start;
  }
  .navigation {
    position: fixed;
    bottom: 10px;
    left: 50%;
    transform: translateX(-50%);
  }
  .navigation button {
    margin: 5px;
  }
</style>
<script>
  function scrollToSection(index) {
    const sections =
      document.querySelectorAll(".section");
    sections[index].scrollIntoView({ behavior:
      "smooth" });
  }
</script>
```

```
</script>

<div class="scroll-section">
  <div class="section">Section 1</div>
  <div class="section">Section 2</div>
  <div class="section">Section 3</div>
</div>

<div class="navigation">
  <button onclick="scrollToSection(0)">1</button>
  <button onclick="scrollToSection(1)">2</button>
  <button onclick="scrollToSection(2)">3</button>
</div>
```

```
1 <style>
2   .scroll-section {
3     height: 100vh;
4     overflow-y: scroll;
5     scroll-snap-type: y mandatory;
6   }
7
8   .section {
9     height: 100vh;
10    scroll-snap-align: start;
11  }
12
13  .navigation {
14    position: fixed;
15    bottom: 10px;
16    left: 50%;
17    transform: translateX(-50%);
18  }
19
20  .navigation button {
21    margin: 5px;
22  }
23 </style>
24
25 <script>
26   function scrollToSection(index) {
27     const sections = document.querySelectorAll(".section");
28     sections[index].scrollIntoView({ behavior: "smooth" });
29   }
30 </script>
31
32 <div class="scroll-section">
33   <div class="section">Section 1</div>
34   <div class="section">Section 2</div>
35   <div class="section">Section 3</div>
36 </div>
37 <div class="navigation">
38   <button onclick="scrollToSection(0)">1</button>
39   <button onclick="scrollToSection(1)">2</button>
40   <button onclick="scrollToSection(2)">3</button>
41 </div>
42
```

Exercise 3

Create a page with multiple sections, and each section should have scroll snap enabled. Add pagination indicators at the bottom of the page to indicate the current section and allow users to navigate between them.

Solution

```
<style>
  .page {
    height: 100vh;
    overflow-y: scroll;
    scroll-snap-type: y mandatory;
  }
  section {
    height: 100vh;
    scroll-snap-align: start;
  }
  .pagination {
    position: fixed;
    bottom: 10px;
    left: 50%;
    transform: translateX(-50%);
  }
  .pagination button {
    margin: 5px;
  }
</style>
<script>
  function scrollToPage(index) {
    const pages =
      document.querySelectorAll("section");
    pages[index].scrollIntoView({ behavior:
      "smooth" });
  }
</script>
```

```
</script>

<div class="page">
  <section>Section 1</section>
  <section>Section 2</section>
  <section>Section 3</section>
</div>

<div class="pagination">
  <button onclick="scrollToPage(0)">1</button>
  <button onclick="scrollToPage(1)">2</button>
  <button onclick="scrollToPage(2)">3</button>
</div>
```

```
1 <style>
2   .page {
3     height: 100vh;
4     overflow-y: scroll;
5     scroll-snap-type: y mandatory;
6   }
7
8   section {
9     height: 100vh;
10    scroll-snap-align: start;
11  }
12
13  .pagination {
14    position: fixed;
15    bottom: 10px;
16    left: 50%;
17    transform: translateX(-50%);
18  }
19
20  .pagination button {
21    margin: 5px;
22  }
23 </style>
24
25 <script>
26   function scrollToPage(index) {
27     const pages = document.querySelectorAll("section");
28     pages[index].scrollIntoView({ behavior: "smooth" });
29   }
30 </script>
31
32 <div class="page">
33   <section>Section 1</section>
34   <section>Section 2</section>
35   <section>Section 3</section>
36 </div>
37 <div class="pagination">
38   <button onclick="scrollToPage(0)">1</button>
39   <button onclick="scrollToPage(1)">2</button>
40   <button onclick="scrollToPage(2)">3</button>
41 </div>
42
```

Chapter 5: CSS3 Backgrounds and Borders

5.1 Styling Backgrounds with CSS3

CSS3 provides several powerful features for styling backgrounds on web pages. These features allow you to apply various effects, gradients, images, and patterns to the background of elements.

Exercises

Exercise 1

Create a gradient background using CSS3 linear or radial gradients. Experiment with different color combinations and gradient directions to create visually appealing backgrounds.

Solution

```
<style>
  .gradient-background {
    width: 100%;
    height: 100vh;
    background: linear-gradient(to bottom, #ffcc00,
#ff3300);
  }
</style>
<div class="gradient-background"></div>
```

```
1 <style>
2   .gradient-background {
3     width: 100%;
4     height: 100vh;
5     background: linear-gradient(to bottom, #ffcc00, #ff3300);
6   }
7 </style>
8
9 <div class="gradient-background"></div>
```

In this example, we create a div element with the class .gradient-background. We then use the linear-gradient property in the background CSS declaration to create a vertical gradient background from #ffcc00 at the top to #ff3300 at the bottom. You can experiment with different color combinations and gradient directions by modifying the values.

Exercise 2

Create a background image overlay effect using CSS3. Apply a transparent color overlay on top of a background image to create a visually pleasing effect.

Solution

```
<style>
  .background-image {
    width: 100%;
    height: 100vh;
    background-image: url("image.jpg");
    background-size: cover;
    position: relative;
  }
  .background-image::before {
    content: "";
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.5);
  }
</style>
<div class="background-image"></div>
```

```
1 <style>
2   .background-image {
3     width: 100%;
4     height: 100vh;
5     background-image: url("image.jpg");
6     background-size: cover;
7     position: relative;
8   }
9
10 .background-image::before {
11   content: "";
12   position: absolute;
13   top: 0;
14   left: 0;
15   width: 100%;
16   height: 100%;
17   background-color: rgba(0, 0, 0, 0.5);
18 }
19 </style>
20
21 <div class="background-image"></div>
```

In this example, we create a div element with the class .background-image. We set the background-image property to the URL of the desired image. The background-size property is set to cover to ensure the image covers the entire element.

To create the overlay effect, we use the ::before pseudo-element and set its background-color property to rgba(0, 0, 0, 0.5). This creates a semi-transparent black overlay on top of the background image. You can adjust the opacity by modifying the alpha value (0.5 in this example).

Exercise 3

Create an animated background using CSS3 animations or keyframes. Apply animations to the background to create movement, transitions, or color changes.

Solution

```
<style>
  .animated-background {
    width: 100%;
    height: 100vh;
    background-color: #000;
    animation: background-animation 5s infinite;
  }
  @keyframes background-animation {
    0% {
      background-color: #000;
    }
    50% {
      background-color: #ff0000;
    }
    100% {
      background-color: #000;
    }
  }
</style>
<div class="animated-background"></div>
```

```
1 <style>
2   .animated-background {
3     width: 100%;
4     height: 100vh;
5     background-color: #000;
6     animation: background-animation 5s infinite;
7   }
8
9   @keyframes background-animation {
10    0% {
11      background-color: #000;
12    }
13    50% {
14      background-color: #ff0000;
15    }
16    100% {
17      background-color: #000;
18    }
19  }
20 </style>
21
22 <div class="animated-background"></div>
```

In this example, we create a div element with the class `.animated-background`. We set the initial background color to `#000`. We define an animation called `background-animation` that lasts for 5 seconds and repeats infinitely (infinite).

The `@keyframes` rule is used to define the animation. We specify the background color at different keyframe percentages. In this example, the background color starts at `#000`, changes to `#ff0000` at the 50% mark, and then returns to `#000` at 100%. You can modify the keyframe percentages and colors to create different animation effects.

5.2 Gradient Backgrounds and Image Effects

CSS3 provides powerful features for creating gradient backgrounds and applying various image effects to elements

Exercises

Exercise 1

Create a div with a gradient background that transitions smoothly from one color to another. Implement a hover effect that changes the gradient colors dynamically.

Solution

```
<style>
  .gradient-background {
    width: 100%;
    height: 100vh;
    background: linear-gradient(to bottom, #ff9900,
#ff3300);
    transition: background 0.5s ease;
  }
  .gradient-background:hover {
    background: linear-gradient(to bottom, #3366ff,
#3300ff);
  }
</style>
<div class="gradient-background"></div>
```

```
1 <style>
2   .gradient-background {
3     width: 100%;
4     height: 100vh;
5     background: linear-gradient(to bottom, #ff9900, #ff3300);
6     transition: background 0.5s ease;
7   }
8
9   .gradient-background:hover {
10     background: linear-gradient(to bottom, #3366ff, #3300ff);
11   }
12 </style>
13
14 <div class="gradient-background"></div>
```

In this example, we create a div element with the class `.gradient-background`. The initial background is a linear gradient that transitions from `#ff9900` to `#ff3300`. We apply a transition property to smoothly animate the background color change.

On hover, the background transitions to a different linear gradient, changing the colors to #3366ff and #3300ff. Adjust the colors and gradient direction as desired.

Exercise 2

Create an image with a transparent overlay effect. When hovering over the image, the overlay should fade in to reveal additional content or details.

Solution

```
<style>
  .image-container {
    position: relative;
    width: 300px;
  }
  .image-container img {
    width: 100%;
    height: auto;
  }
  .overlay {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.5);
    color: #fff;
    opacity: 0;
    transition: opacity 0.3s ease;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
  }
  .image-container:hover .overlay {
    opacity: 1;
  }
</style>
```

```

</style>

<div class="image-container">
  
  <div class="overlay">
    <h2>Image Title</h2>
    <p>Additional details about the image.</p>
  </div>
</div>

```

```

1  <style>
2    .image-container {
3      position: relative;
4      width: 300px;
5    }
6
7    .image-container img {
8      width: 100%;
9      height: auto;
10   }
11
12   .overlay {
13     position: absolute;
14     top: 0;
15     left: 0;
16     width: 100%;
17     height: 100%;
18     background: rgba(0, 0, 0, 0.5);
19     color: #fff;
20     opacity: 0;
21     transition: opacity 0.3s ease;
22     display: flex;
23     flex-direction: column;
24     justify-content: center;
25     align-items: center;
26   }
27
28   .image-container:hover .overlay {
29     opacity: 1;
30   }
31 </style>
32
33 <div class="image-container">
34   
35   <div class="overlay">
36     <h2>Image Title</h2>
37     <p>Additional details about the image.</p>
38   </div>
39 </div>

```

In this example, we create an image container with an image and an overlay div. The overlay div is absolutely positioned inside the

container and covers the entire area.

The overlay div has a semi-transparent black background using `rgba(0, 0, 0, 0.5)`, making it partially transparent. Adjust the opacity value to control the transparency level. Inside the overlay, you can add additional content such as a title and description.

When hovering over the image container, the overlay's opacity changes to 1, revealing the content. Adjust the transition duration and easing as desired.

Exercise 3

Create a container with multiple layers of gradient backgrounds using CSS3. Each layer should have a different gradient and blend together harmoniously to create a visually appealing effect.

Solution

```
<style>
  .gradient-container {
    width: 100%;
    height: 100vh;
    background: linear-gradient(to bottom, #ff9900,
#ff3300),
      linear-gradient(to right, #00ccff, #0066ff);
    background-blend-mode: multiply, overlay;
  }
  .gradient-container h1 {
    color: #fff;
    font-size: 3rem;
    text-align: center;
    margin-top: 40vh;
  }
</style>
<div class="gradient-container">
  <h1>Multiple Gradient Backgrounds</h1>
</div>
```

```
1 <style>
2   .gradient-container {
3     width: 100%;
4     height: 100vh;
5     background: linear-gradient(to bottom, #ff9900, #ff3300),
6       linear-gradient(to right, #00ccff, #0066ff);
7     background-blend-mode: multiply, overlay;
8   }
9
10  .gradient-container h1 {
11    color: #fff;
12    font-size: 3rem;
13    text-align: center;
14    margin-top: 40vh;
15  }
16 </style>
17
18 <div class="gradient-container">
19   <h1>Multiple Gradient Backgrounds</h1>
20 </div>
```

In this example, we create a div with the class `.gradient-container`. The background property includes two linear gradients separated by a comma. The first gradient transitions from `#ff9900` to `#ff3300` vertically, and the second gradient transitions from `#00ccff` to `#0066ff` horizontally.

The `background-blend-mode` property is used to specify the blending mode for the multiple backgrounds. In this case, we use "multiply" for the first gradient and "overlay" for the second gradient. Adjust the colors and gradient directions to achieve your desired effect.

Inside the container, we add an `h1` element with white text color for visibility. Customize the text and styling as needed.

5.3 Border Properties in CSS3

CSS3 provides a wide range of border properties that allow you to control the appearance and style of borders around HTML elements.

Exercises

Exercise 1

Create a div with a dashed border and rounded corners. Experiment with different border colors, widths, and corner radii.

Solution

```
<style>
  .border-example {
    width: 200px;
    height: 100px;
    border: 2px dashed #ff9900;
    border-radius: 10px;
  }
</style>
<div class="border-example"></div>
```

```
1 <style>
2   .border-example {
3     width: 200px;
4     height: 100px;
5     border: 2px dashed #ff9900;
6     border-radius: 10px;
7   }
8 </style>
9
10 <div class="border-example"></div>
```

In this example, we create a div element with the class `.border-example`. The div has a width of 200px and a height of 100px. The border property is set to 2px dashed #ff9900, which creates a dashed border with a width of 2px and a color of #ff9900. Adjust the border color, width, and style (e.g., dashed, dotted, etc.) as desired. The border-radius property is set to 10px, which adds rounded corners to the border. Adjust the border-radius value to control the roundness of the corners.

Exercise 2

Create a button with a solid border. Apply a hover effect that animates the border, such as changing the color or increasing the border width.

Solution

```
<style>
  .border-button {
    padding: 10px 20px;
    border: 2px solid #3366ff;
    transition: border-color 0.3s ease;
  }
  .border-button:hover {
    border-color: #ff3300;
  }
</style>
<button class="border-button">Hover me</button>
```

```
1 <style>
2   .border-button {
3     padding: 10px 20px;
4     border: 2px solid #3366ff;
5     transition: border-color 0.3s ease;
6   }
7
8   .border-button:hover {
9     border-color: #ff3300;
10  }
11 </style>
12
13 <button class="border-button">Hover me</button>
```

In this example, we create a button element with the class `.border-button`. The button has padding, a solid border with a width of 2px, and a color of `#3366ff`. Adjust the padding, border color, width, and other styles as desired.

The transition property is used to define the animation effect. In this case, we specify that the border-color property should transition with a duration of 0.3s and an easing function of ease. Adjust the transition duration and easing function to achieve the desired animation effect.

On hover, the border-color property changes to #ff3300, creating an animation effect. Customize the hover color to your liking.

Exercise 3

Create a div with a linear gradient border. Experiment with different gradient directions and colors to achieve an interesting border effect.

Solution

```
<style>
  .border-gradient {
    width: 200px;
    height: 200px;
    border: 5px solid transparent;
    background-clip: padding-box;
    background-image: linear-gradient(to right,
#ff9900, #ff3300);
  }
</style>
<div class="border-gradient"></div>
```

```
1 <style>
2   .border-gradient {
3     width: 200px;
4     height: 200px;
5     border: 5px solid transparent;
6     background-clip: padding-box;
7     background-image: linear-gradient(to right, #ff9900, #ff3300);
8   }
9 </style>
10 <div class="border-gradient"></div>
```

In this example, we create a div element with the class `.border-gradient`. The div has a width of 200px and a height of 200px. The border property is set to 5px solid transparent, which creates a solid border with a width of 5px. The transparent value makes the border invisible.

The `background-clip` property is set to `padding-box`, which ensures that the background gradient is confined within the padding area and does not extend into the border.

The `background-image` property is set to a linear gradient that transitions from `#ff9900` to `#ff3300` from left to right. Adjust the

gradient direction and colors to achieve the desired effect.

5.4 Rounded Corners and Box Shadows

CSS3 provides features to create rounded corners and apply box shadows to elements, allowing you to add visual effects and enhance the appearance of your web pages.

Exercises

Exercise 1

Create a card-like container with rounded corners and a box shadow. Experiment with different corner radii, shadow colors, and shadow properties.

Solution

```
<style>
  .card {
    width: 300px;
    height: 200px;
    background-color: #fff;
    border-radius: 10px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);
  }
</style>
<div class="card"></div>
```

```
1 <style>
2   .card {
3     width: 300px;
4     height: 200px;
5     background-color: #fff;
6     border-radius: 10px;
7     box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);
8   }
9 </style>
10 <div class="card"></div>
```

In this example, we create a div element with the class .card. The card has a width of 300px and a height of 200px.

The background-color property sets the background color of the card to #fff (white). Adjust the color as desired.

The border-radius property is set to 10px, which adds rounded corners to the card. Adjust the border-radius value to control the roundness of the corners.

The box-shadow property adds a box shadow effect to the card. The values 0 2px 10px rgba(0, 0, 0, 0.3) specify a horizontal offset of 0, a vertical offset of 2px, a blur radius of 10px, and a color of rgba(0, 0, 0,

0.3). Adjust the shadow properties (offsets, blur radius, and color) to achieve the desired effect.

Exercise 2

Create a button with rounded corners. Apply a hover effect that increases the size of the button and adds a box shadow for a visual effect.

Solution

```
<style>
  .rounded-button {
    padding: 10px 20px;
    border: none;
    background-color: #3366ff;
    color: #fff;
    border-radius: 20px;
    transition: transform 0.3s, box-shadow 0.3s;
  }
  .rounded-button:hover {
    transform: scale(1.1);
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
  }
</style>
<button class="rounded-button">Hover me</button>
```

```
1 <style>
2   .rounded-button {
3     padding: 10px 20px;
4     border: none;
5     background-color: #3366ff;
6     color: #fff;
7     border-radius: 20px;
8     transition: transform 0.3s, box-shadow 0.3s;
9   }
10
11  .rounded-button:hover {
12    transform: scale(1.1);
13    box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
14  }
15 </style>
16
17 <button class="rounded-button">Hover me</button>
```

In this example, we create a button element with the class `.rounded-button`. The button has padding, no border (border: none), a background color of `#3366ff` (blue), and white text color (color: `#fff`).

The border-radius property is set to 20px, which adds rounded corners to the button. Adjust the border-radius value to control the roundness of the corners.

The transition property is used to define the hover animation effects. We specify that the transform and box-shadow properties should transition with a duration of 0.3s. Adjust the transition duration as desired.

On hover, the button scales up to 1.1 times its original size (transform: scale(1.1)) and adds a box shadow effect (box-shadow: 0 0 10px rgba(0, 0, 0, 0.3)). Customize the hover effects to your liking.

Exercise 3

Create an image thumbnail with rounded corners and a box shadow. Customize the corner radii, shadow properties, and image dimensions to achieve an appealing visual effect.

Solution

```
<style>
  .image-thumbnail {
    width: 200px;
    height: 200px;
    overflow: hidden;
    border-radius: 10px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);
  }
  .image-thumbnail img {
    width: 100%;
    height: 100%;
    object-fit: cover;
  }
</style>
<div class="image-thumbnail">
  
</div>
```

```
1  <style>
2    .image-thumbnail {
3      width: 200px;
4      height: 200px;
5      overflow: hidden;
6      border-radius: 10px;
7      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);
8    }
9
10   .image-thumbnail img {
11     width: 100%;
12     height: 100%;
13     object-fit: cover;
14   }
15 </style>
16
17 <div class="image-thumbnail">
18   
19 </div>
```

In this example, we create a div element with the class .image-thumbnail. The div has a width and height of 200px.

The overflow property is set to hidden to ensure that the image does not overflow beyond the thumbnail container. Adjust this property if needed.

The border-radius property is set to 10px, which adds rounded corners to the thumbnail. Adjust the border-radius value to control the roundness of the corners.

The box-shadow property adds a box shadow effect to the thumbnail. The values 0 2px 10px rgba(0, 0, 0, 0.3) specify a horizontal offset of 0, a vertical offset of 2px, a blur radius of 10px, and a color of rgba(0, 0, 0, 0.3). Adjust the shadow properties (offsets, blur radius, and color) to achieve the desired effect.

Inside the thumbnail container, we add an img element that fills the entire container using width: 100% and height: 100%. The object-fit: cover property ensures that the image scales proportionally to cover the entire thumbnail without distortion.

Chapter 6: CSS3 Typography and Web Fonts

6.1 Web Fonts and @font-face Rule

CSS3 provides the @font-face rule, which allows you to use custom web fonts on your web pages. With web fonts, you can apply unique typography and use fonts that may not be installed on users' devices.

Exercises

Exercise 1

Use a custom web font for a heading text. Find a web font of your choice and apply it to the heading element.

Solution

```
<style>
  @font-face {
    font-family: "CustomFont";
    src: url("customfont.woff2") format("woff2"),
         url("customfont.woff") format("woff");
  }
  .custom-heading {
    font-family: "CustomFont", sans-serif;
  }
</style>
<h1 class="custom-heading">Hello, World!</h1>
```

```
1 <style>
2   @font-face {
3     font-family: "CustomFont";
4     src: url("customfont.woff2") format("woff2"),
5          url("customfont.woff") format("woff");
6   }
7
8   .custom-heading {
9     font-family: "CustomFont", sans-serif;
10  }
11 </style>
12
13 <h1 class="custom-heading">Hello, World!</h1>
```

In this example, we use the `@font-face` rule to define a custom font called 'CustomFont'. We provide the paths to the font files (`customfont.woff2` and `customfont.woff`) using the `src` property. Adjust the file names and paths to match the location of your font files. Next, we apply the custom font to the `.custom-heading` class using the `font-family` property. We specify 'CustomFont' as the `font-family` name and fall back to a generic sans-serif font using the `sans-serif` keyword.

Exercise 2

Use the @font-face rule to include multiple font formats (e.g., .woff, .woff2, .ttf) to ensure cross-browser compatibility and support for different font file types.

Solution

```
<style>
  @font-face {
    font-family: "CustomFont";
    src: url("customfont.woff2") format("woff2"),
         url("customfont.woff") format("woff"),
         url("customfont.ttf") format("truetype");
  }
  .custom-paragraph {
    font-family: "CustomFont", sans-serif;
  }
</style>
<p class="custom-paragraph">This is a paragraph
with a custom font.</p>
```

```
1 <style>
2   @font-face {
3     font-family: "CustomFont";
4     src: url("customfont.woff2") format("woff2"),
5          url("customfont.woff") format("woff"),
6          url("customfont.ttf") format("truetype");
7   }
8
9   .custom-paragraph {
10     font-family: "CustomFont", sans-serif;
11   }
12 </style>
13
14 <p class="custom-paragraph">This is a paragraph with a custom font.</p>
```

In this example, we extend the previous exercise by including multiple font formats in the `@font-face` rule. We provide the paths to the font files in three different formats: `.woff2`, `.woff`, and `.ttf`. Adjust the file names and paths to match your font files.

By including multiple font formats, we ensure cross-browser compatibility and support for different font file types. Browsers will

select the appropriate font format based on their capabilities.

Exercise 3

Use the @font-face rule to include a subset of a web font, containing only the characters you need for your web page, to optimize performance and reduce file size.

Solution

```
<style>
  @font-face {
    font-family: "SubsetFont";
    src: url("subsetfont-subset.woff2")
      format("woff2");
  }
  .subset-paragraph {
    font-family: "SubsetFont", sans-serif;
  }
</style>
<p class="subset-paragraph">This paragraph contains
a subset of a font.</p>
```

```
1 <style>
2   @font-face {
3     font-family: "SubsetFont";
4     src: url("subsetfont-subset.woff2") format("woff2");
5   }
6
7   .subset-paragraph {
8     font-family: "SubsetFont", sans-serif;
9   }
10 </style>
11
12 <p class="subset-paragraph">This paragraph contains a subset of a font.</p>
```

In this example, we use the `@font-face` rule to include a subset of a font. The font file used is `subsetfont-subset.woff2`, which contains only the necessary characters for the web page.

By including a font subset, we optimize performance and reduce the file size, as only the required characters are loaded. Adjust the font file name and path to match your subset font file.

We then apply the '`SubsetFont`' font family to the `.subset-paragraph` class using the `font-family` property. The browser will load the subset

font and apply it to the text within the .subset-paragraph element.

6.2 Customizing Typography with CSS3

CSS3 provides a range of properties for customizing typography on web pages. These properties allow you to control the font family, size, weight, style, spacing, and more.

Exercises

Exercise 1

Create a paragraph of text and customize the font size and line height to achieve a visually appealing and readable typography style.

Solution

```
<style>
  .custom-paragraph {
    font-size: 18px;
    line-height: 1.5;
  }
</style>
<p class="custom-paragraph">
  Lorem ipsum dolor sit amet, consectetur
  adipiscing elit.
</p>
```

```
1 <style>
2   .custom-paragraph {
3     font-size: 18px;
4     line-height: 1.5;
5   }
6 </style>
7
8 <p class="custom-paragraph">
9   Lorem ipsum dolor sit amet, consectetur adipiscing elit.
10 </p>
```

In this example, we create a paragraph with the class `.custom-paragraph`. We customize the font size to 18px using the `font-size` property. Adjust the value to your desired font size. We also adjust the line height to 1.5 using the `line-height` property. This will increase the space between lines of text, improving readability. Adjust the value as desired.

Exercise 2

Customize the text color and text decoration (such as underline, overline, or line-through) for a heading or paragraph to create emphasis and visual interest.

Solution

```
<style>
  .custom-heading {
    color: #ff0000;
    text-decoration: underline;
  }
</style>
<h1 class="custom-heading">Welcome to my
website</h1>
```

```
1 <style>
2   .custom-heading {
3     color: #ff0000;
4     text-decoration: underline;
5   }
6 </style>
7
8 <h1 class="custom-heading">Welcome to my website</h1>
```

In this example, we create a heading with the class `.custom-heading`. We customize the text color to red (`#ff0000`) using the `color` property. Adjust the color value to your desired text color. We also apply the underline text decoration using the `text-decoration` property. This adds an underline to the heading text. You can experiment with other text decoration values such as `overline` or `line-through` to achieve different effects.

Exercise 3

Experiment with letter spacing and word spacing to adjust the spacing between characters and words in a paragraph, creating different typographic effects.

Solution

```
<style>
  .custom-paragraph {
    letter-spacing: 2px;
    word-spacing: 5px;
  }
</style>
<p class="custom-paragraph">This is a sample
paragraph for spacing.</p>
```

```
1 <style>
2   .custom-paragraph {
3     letter-spacing: 2px;
4     word-spacing: 5px;
5   }
6 </style>
7
8 <p class="custom-paragraph">This is a sample paragraph for spacing.</p>
```

In this example, we create a paragraph with the class `.custom-paragraph`. We adjust the letter spacing to `2px` using the `letter-spacing` property. This increases the spacing between individual characters. Adjust the value as desired.

We also adjust the word spacing to `5px` using the `word-spacing` property. This increases the spacing between words. Adjust the value as desired.

6.3 Text Shadows and Text Effects

CSS3 provides properties that allow you to add text shadows and various text effects to enhance the visual appearance of text on web pages.

Exercises

Exercise 1

Apply a text shadow to a heading or paragraph to create a visually appealing effect.

Solution

```
<style>
  .text-shadow-heading {
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
  }
</style>
<h1 class="text-shadow-heading">Hello, World!</h1>
```

```
1 <style>
2   .text-shadow-heading {
3     text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
4   }
5 </style>
6
7 <h1 class="text-shadow-heading">Hello, World!</h1>
```

In this example, we create a heading with the class `.text-shadow-heading`. We apply a text shadow effect using the `text-shadow` property. The values `2px 2px 4px rgba(0, 0, 0, 0.5)` define the horizontal offset, vertical offset, blur radius, and color of the shadow. Adjust these values to achieve your desired text shadow effect.

Exercise 2

Create a gradient text effect using CSS3 properties to apply a smooth transition of colors to the text.

Solution

```
<style>
  .text-gradient-heading {
    background: linear-gradient(to right, #ff0000,
#00ff00);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
  }
</style>
<h1 class="text-gradient-heading">Welcome to
CSS3</h1>
```

```
1 <style>
2   .text-gradient-heading {
3     background: linear-gradient(to right, #ff0000, #00ff00);
4     -webkit-background-clip: text;
5     -webkit-text-fill-color: transparent;
6   }
7 </style>
8
9 <h1 class="text-gradient-heading">Welcome to CSS3</h1>
```

In this example, we create a heading with the class `.text-gradient-heading`. We apply a gradient text effect using the `background` property with the `linear-gradient()` function. Adjust the colors (`#ff0000` and `#00ff00`) and gradient direction (to right, to bottom, etc.) to your liking.

To make the text show the gradient, we use the `-webkit-background-clip` property with the value `text`, and set the text color to transparent using `-webkit-text-fill-color`. Note that this solution works best on webkit-based browsers such as Chrome and Safari.

Exercise 3

Animate the text in a heading or paragraph using CSS3 keyframes and animation properties to create an eye-catching effect.

Solution

```
<style>
  .text-animation-heading {
    animation: textAnimation 2s infinite;
  }
  @keyframes textAnimation {
    0% {
      transform: scale(1);
    }
    50% {
      transform: scale(1.2);
      color: blue;
    }
    100% {
      transform: scale(1);
    }
  }
</style>
<h1 class="text-animation-heading">Hello, World!
</h1>
```

```
1 <style>
2   .text-animation-heading {
3     animation: textAnimation 2s infinite;
4   }
5
6   @keyframes textAnimation {
7     0% {
8       transform: scale(1);
9     }
10    50% {
11      transform: scale(1.2);
12      color: blue;
13    }
14    100% {
15      transform: scale(1);
16    }
17  }
18 </style>
19
20 <h1 class="text-animation-heading">Hello, World!</h1>
```

In this example, we create a heading with the class `.text-animation-heading`. We apply a text animation effect using the `animation` property. The animation is defined by the `textAnimation` keyframes animation.

In the `textAnimation` keyframes, we define three keyframe percentages (0%, 50%, 100%) and specify the desired transformations and styles at each point. At 50%, the text scales up to 1.2 times its original size and changes color to blue. Adjust the animation duration (2s in this example) and the transformations and styles to match your desired effect.

6.4 Creating Responsive Typography

Creating responsive typography is crucial for ensuring that text on your web pages adapts and scales appropriately across different screen sizes and devices. CSS3 provides several techniques and properties to achieve responsive typography.

Exercises

Exercise 1

Create a fluid typography effect where the font size adjusts based on the viewport size, ensuring optimal readability on different devices and screen sizes.

Solution

```
<style>
  .fluid-typography {
    font-size: calc(1rem + 1vw);
  }
</style>
<h1 class="fluid-typography">Welcome to my
website</h1>
```

```
1 <style>
2   .fluid-typography {
3     font-size: calc(1rem + 1vw);
4   }
5 </style>
6
7 <h1 class="fluid-typography">Welcome to my website</h1>
```

In this example, we create a heading with the class `.fluid-typography`. The `calc()` function is used to calculate the font size based on a combination of a base font size (`1rem`) and the viewport width (`1vw`). This creates a fluid typography effect where the font size adjusts based on the width of the viewport.

Exercise 2

Use media queries to define different font sizes for specific breakpoints, allowing the typography to adapt and respond to different screen sizes.

Solution

```
<style>
    .responsive-typography {
        font-size: 18px;
    }
    @media (min-width: 768px) {
        .responsive-typography {
            font-size: 22px;
        }
    }
    @media (min-width: 1024px) {
        .responsive-typography {
            font-size: 26px;
        }
    }
</style>
<h1 class="responsive-typography">Welcome to my website</h1>
```

```
1  <style>
2      .responsive-typography {
3          font-size: 18px;
4      }
5
6      @media (min-width: 768px) {
7          .responsive-typography {
8              font-size: 22px;
9          }
10     }
11
12     @media (min-width: 1024px) {
13         .responsive-typography {
14             font-size: 26px;
15         }
16     }
17 </style>
18
19 <h1 class="responsive-typography">Welcome to my website</h1>
```

In this example, we create a heading with the class `.responsive-typography`. We set the default font size to be 18 pixels. Using media queries, we define different font sizes for specific breakpoints. When the viewport width is 768 pixels or wider, the font size increases to 22 pixels. When the viewport width is 1024 pixels or wider, the font size further increases to 26 pixels. Adjust the breakpoints and font sizes according to your desired responsiveness.

Exercise 3

Utilize viewport units, such as `vw` and `vh`, to set font sizes that are relative to the viewport size, ensuring consistent typography scaling across different devices.

Solution

```
<style>
  .viewport-typography {
    font-size: 5vw;
  }
</style>
<h1 class="viewport-typography">Welcome to my
website</h1>
```

```
1 <style>
2   .viewport-typography {
3     font-size: 5vw;
4   }
5 </style>
6
7 <h1 class="viewport-typography">Welcome to my website</h1>
```

In this example, we create a heading with the class `.viewport-typography`. We use the `vw` (viewport width) unit to set the font size to be 5% of the viewport width. This ensures that the font size scales relative to the viewport size, providing consistent typography across different devices.

Chapter 7: CSS3 Transitions and Animations

7.1 Transition Properties and Timing Functions

CSS3 provides a range of transition properties and timing functions that allow you to control and customize the behavior of CSS transitions.

Exercises

Exercise 1

Create a hover animation where an element smoothly transitions its background color when the mouse hovers over it.

Solution

```
<style>
  nav ul {
    list-style: none;
    padding: 0;
    margin: 0;
  }
  nav li {
    position: relative;
  }
  nav a {
    display: block;
    padding: 10px;
    background-color: lightblue;
    color: white;
    text-decoration: none;
  }
  .submenu {
    max-height: 0;
    overflow: hidden;
    transition: max-height 0.3s ease;
  }
  nav li:hover .submenu {
    max-height: 200px;
  }
</style>
<nav>
  <ul>
```

```
<li>
  <a href="#">Menu Item 1</a>
  <ul class="submenu">
    <li><a href="#">Submenu Item 1</a></li>
    <li><a href="#">Submenu Item 2</a></li>
    <li><a href="#">Submenu Item 3</a></li>
  </ul>
</li>
<li><a href="#">Menu Item 2</a></li>
<li><a href="#">Menu Item 3</a></li>
</ul>
</nav>
```

```

1  <style>
2    nav ul {
3      list-style: none;
4      padding: 0;
5      margin: 0;
6    }
7
8    nav li {
9      position: relative;
10   }
11
12   nav a {
13     display: block;
14     padding: 10px;
15     background-color: lightblue;
16     color: white;
17     text-decoration: none;
18   }
19
20   .submenu {
21     max-height: 0;
22     overflow: hidden;
23     transition: max-height 0.3s ease;
24   }
25
26   nav li:hover .submenu {
27     max-height: 200px;
28   }
29 </style>
30
31 <nav>
32   <ul>
33     <li>
34       <a href="#">Menu Item 1</a>
35       <ul class="submenu">
36         <li><a href="#">Submenu Item 1</a></li>
37         <li><a href="#">Submenu Item 2</a></li>
38         <li><a href="#">Submenu Item 3</a></li>
39       </ul>
40     </li>
41     <li><a href="#">Menu Item 2</a></li>
42     <li><a href="#">Menu Item 3</a></li>
43   </ul>
44 </nav>

```

In this example, we create a simple navigation menu with a dropdown submenu. The submenu is initially hidden with a max-height of 0 and an overflow set to hidden. When the parent menu item is hovered over, the max-height of the submenu is transitioned to 200px, causing it to smoothly slide down and reveal the submenu.

Exercise 2

Apply a fade-in effect to an image so that it gradually appears when the page loads.

Solution

```
<style>

  .card {
    width: 200px;
    height: 200px;
    perspective: 1000px;
  }

  .card-front,
  .card-back {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    backface-visibility: hidden;
    transition: transform 0.8s ease;
  }

  .card-front {
    background-color: lightblue;
  }

  .card-back {
    background-color: lightgray;
    transform: rotateY(180deg);
  }

  .card:hover .card-front {
    transform: rotateY(180deg);
  }

  .card:hover .card-back {
```

```
    transform: rotateY(0deg);  
}  
</style>  
<div class="card">  
  <div class="card-front">  
    <h2>Front Side</h2>  
  </div>  
  <div class="card-back">  
    <h2>Back Side</h2>  
  </div>  
</div>
```

```

1 <style>
2   .card {
3     width: 200px;
4     height: 200px;
5     perspective: 1000px;
6   }
7
8   .card-front,
9   .card-back {
10    position: absolute;
11    top: 0;
12    left: 0;
13    width: 100%;
14    height: 100%;
15    backface-visibility: hidden;
16    transition: transform 0.8s ease;
17  }
18
19  .card-front {
20    background-color: lightblue;
21  }
22
23  .card-back {
24    background-color: lightgray;
25    transform: rotateY(180deg);
26  }
27
28  .card:hover .card-front {
29    transform: rotateY(180deg);
30  }
31
32  .card:hover .card-back {
33    transform: rotateY(0deg);
34  }
35 </style>
36
37 <div class="card">
38   <div class="card-front">
39     <h2>Front Side</h2>
40   </div>
41   <div class="card-back">
42     <h2>Back Side</h2>
43   </div>
44 </div>

```

In this example, we have a card element with a front and back side. The card is set to have perspective, which enables the 3D effect. The front and back sides are placed absolutely within the card element. By default, the backface is hidden, and the back side is rotated 180deg around the Y-axis, making it invisible. When the card is hovered over, the transform property is used to rotate the card, revealing the back side with a flip animation.

Exercise 3

Add an animation to a button element, making it change color and size smoothly when clicked.

Solution

```
<style>
  .color-cycle {
    animation: color-cycle 5s infinite;
  }
  @keyframes color-cycle {
    0% {
      color: red;
    }
    25% {
      color: blue;
    }
    50% {
      color: green;
    }
    75% {
      color: orange;
    }
    100% {
      color: red;
    }
  }
</style>
<h1 class="color-cycle">Hello, World!</h1>
```

```
1 <style>
2   .color-cycle {
3     animation: color-cycle 5s infinite;
4   }
5
6   @keyframes color-cycle {
7     0% {
8       color: red;
9     }
10    25% {
11      color: blue;
12    }
13    50% {
14      color: green;
15    }
16    75% {
17      color: orange;
18    }
19    100% {
20      color: red;
21    }
22  }
23 </style>
24
25 <h1 class="color-cycle">Hello, World!</h1>
```

In this example, we have a heading element with the class `.color-cycle`. The `animation` property is used to apply the `color-cycle` animation, which lasts for 5s and repeats infinitely. The `@keyframes` rule defines the different keyframe percentages and the corresponding colors. In this case, the text color cycles through red, blue, green, and orange in a smooth and continuous loop.

7.2 Creating Animations with CSS3 Keyframes

CSS3 keyframes allow you to create complex animations by defining a sequence of keyframe rules that specify the styles at various points in time.

Exercises

Exercise 1

Create an animation where a ball bounces up and down within a container.

Solution

```
<style>
  .container {
    width: 200px;
    height: 200px;
    background-color: lightgray;
    position: relative;
    overflow: hidden;
  }
  .ball {
    width: 50px;
    height: 50px;
    background-color: red;
    position: absolute;
    bottom: 0;
    animation: bounce 1s infinite;
  }
  @keyframes bounce {
    0% {
      transform: translateY(0);
    }
    50% {
      transform: translateY(-100px);
    }
    100% {
      transform: translateY(0);
    }
  }
</style>
```

```
</style>

<div class="container">
  <div class="ball"></div>
</div>
```

```
1  <style>
2    .container {
3      width: 200px;
4      height: 200px;
5      background-color: lightgray;
6      position: relative;
7      overflow: hidden;
8    }
9
10   .ball {
11     width: 50px;
12     height: 50px;
13     background-color: red;
14     position: absolute;
15     bottom: 0;
16     animation: bounce 1s infinite;
17   }
18
19   @keyframes bounce {
20     0% {
21       transform: translateY(0);
22     }
23     50% {
24       transform: translateY(-100px);
25     }
26     100% {
27       transform: translateY(0);
28     }
29   }
30 </style>
31
32 <div class="container">
33   <div class="ball"></div>
34 </div>
```

In this example, we have a container with a ball inside. The container is a square with a gray background. The ball is a red circle positioned at the bottom of the container. The bounce animation is applied to the ball, causing it to move up and down within the container. The `@keyframes` rule defines the different keyframe percentages and the corresponding translations. In this case, the ball moves up by 100px at the midpoint of the animation and returns to its original position by the end, creating a bouncing effect.

Exercise 2

Create an animation where a series of images fade in and out to create a slideshow effect.

Solution

```
<style>
.slideshow {
    position: relative;
    height: 300px;
    width: 400px;
    overflow: hidden;
}
.slideshow img {
    position: absolute;
    top: 0;
    left: 0;
    opacity: 0;
    animation: fade 5s infinite;
}
.slideshow img:nth-child(1) {
    animation-delay: 0s;
}
.slideshow img:nth-child(2) {
    animation-delay: 2s;
}
.slideshow img:nth-child(3) {
    animation-delay: 4s;
}
@keyframes fade {
    0% {
        opacity: 0;
    }
}
```

```
50% {
    opacity: 1;
}
100% {
    opacity: 0;
}
}

</style>
<div class="slideshow">
    
    
    
</div>
```

```

1 <style>
2   .slideshow {
3     position: relative;
4     height: 300px;
5     width: 400px;
6     overflow: hidden;
7   }
8
9   .slideshow img {
10    position: absolute;
11    top: 0;
12    left: 0;
13    opacity: 0;
14    animation: fade 5s infinite;
15  }
16
17   .slideshow img:nth-child(1) {
18     animation-delay: 0s;
19   }
20
21   .slideshow img:nth-child(2) {
22     animation-delay: 2s;
23   }
24
25   .slideshow img:nth-child(3) {
26     animation-delay: 4s;
27   }
28
29   @keyframes fade {
30     0% {
31       opacity: 0;
32     }
33     50% {
34       opacity: 1;
35     }
36     100% {
37       opacity: 0;
38     }
39   }
40 </style>
41
42 <div class="slideshow">
43   
44   
45   
46 </div>

```

In this example, we have a slideshow container with multiple images. The container has a fixed height and width and is set to hide any overflow. Each image is positioned absolutely within the container, stacked on top of each other. The fade animation is applied to each image, causing them to fade in and out. The @keyframes rule defines the different keyframe percentages and the corresponding opacity

values. In this case, the images fade in at the midpoint of the animation and fade out by the end, creating a slideshow effect.

Exercise 3

Create an animation where a spinner rotates continuously.

Solution

```
<style>
  .spinner {
    width: 100px;
    height: 100px;
    border: 5px solid lightblue;
    border-top-color: transparent;
    border-radius: 50%;
    animation: spin 1s infinite linear;
  }
  @keyframes spin {
    0% {
      transform: rotate(0deg);
    }
    100% {
      transform: rotate(360deg);
    }
  }
</style>
<div class="spinner"></div>
```

```
1 <style>
2   .spinner {
3     width: 100px;
4     height: 100px;
5     border: 5px solid lightblue;
6     border-top-color: transparent;
7     border-radius: 50%;
8     animation: spin 1s infinite linear;
9   }
10
11 @keyframes spin {
12   0% {
13     transform: rotate(0deg);
14   }
15   100% {
16     transform: rotate(360deg);
17   }
18 }
19 </style>
20
21 <div class="spinner"></div>
```

In this example, we have a spinner element represented by a circle with a border. The spin animation is applied to the spinner, causing it to rotate continuously. The `@keyframes` rule defines the different keyframe percentages and the corresponding rotations. In this case, the spinner rotates from 0deg to 360deg, completing a full rotation over the duration of 1 second.

7.3 Advanced Animation Techniques

CSS3 offers advanced animation techniques that can bring your web pages to life.

Exercises

Exercise 1

Create a parallax scrolling effect where multiple layers of content move at different speeds as the user scrolls.

Solution

```
<style>
.parallax-container {
    height: 500px;
    overflow: hidden;
    position: relative;
}
.layer {
    position: absolute;
    width: 100%;
    height: 100%;
}
.layer1 {
    background-image: url("image1.jpg");
    background-size: cover;
    background-position: center;
    transform: translateZ(0) scale(1.2);
}
.layer2 {
    background-image: url("image2.jpg");
    background-size: cover;
    background-position: center;
    transform: translateZ(-100px) scale(1.4);
}
.layer3 {
    background-image: url("image3.jpg");
    background-size: cover;
    background-position: center;
```

```
        transform: translateZ(-200px) scale(1.6);  
    }  
  
```

```
</style>  
  
<div class="parallax-container">  
  <div class="layer layer1"></div>  
  <div class="layer layer2"></div>  
  <div class="layer layer3"></div>  
</div>
```

```
1  <style>  
2    .parallax-container {  
3      height: 500px;  
4      overflow: hidden;  
5      position: relative;  
6    }  
7  
8    .layer {  
9      position: absolute;  
10     width: 100%;  
11     height: 100%;  
12   }  
13  
14   .layer1 {  
15     background-image: url("image1.jpg");  
16     background-size: cover;  
17     background-position: center;  
18     transform: translateZ(0) scale(1.2);  
19   }  
20  
21   .layer2 {  
22     background-image: url("image2.jpg");  
23     background-size: cover;  
24     background-position: center;  
25     transform: translateZ(-100px) scale(1.4);  
26   }  
27  
28   .layer3 {  
29     background-image: url("image3.jpg");  
30     background-size: cover;  
31     background-position: center;  
32     transform: translateZ(-200px) scale(1.6);  
33   }  
34 </style>  
35  
36 <div class="parallax-container">  
37   <div class="layer layer1"></div>  
38   <div class="layer layer2"></div>  
39   <div class="layer layer3"></div>  
40 </div>  
41
```

In this solution, we have a parallax scrolling effect with multiple layers of content. The .parallax-container acts as the container for the layers. Each .layer has a different background image and is positioned absolutely within the container. The transform property is used to apply a 3D translation effect and scale the layers. This creates a parallax scrolling effect where the layers move at different speeds as the user scrolls.

Exercise 2

Create an animation where a shape morphs into another shape smoothly.

Solution

```
<style>
  .shape {
    width: 100px;
    height: 100px;
    background-color: red;
    border-radius: 50%;
    animation: morph 2s infinite alternate;
  }
  @keyframes morph {
    0% {
      border-radius: 50%;
    }
    50% {
      border-radius: 0%;
    }
    100% {
      border-radius: 50%;
    }
  }
</style>
<div class="shape"></div>
```

```
1 <style>
2   .shape {
3     width: 100px;
4     height: 100px;
5     background-color: red;
6     border-radius: 50%;
7     animation: morph 2s infinite alternate;
8   }
9
10 @keyframes morph {
11   0% {
12     border-radius: 50%;
13   }
14   50% {
15     border-radius: 0%;
16   }
17   100% {
18     border-radius: 50%;
19   }
20 }
21 </style>
22
23 <div class="shape"></div>
```

In this solution, we have a shape represented by a square div with a red background. The morph animation is applied to the shape, causing it to morph into a circle and back to a square repeatedly. The @keyframes rule defines the different keyframe percentages and the corresponding border-radius values. This creates a smooth transition between the square and circular shapes, resulting in a morphing shape animation.

Exercise 3

Create an animation where particles move across the screen in a random or predefined pattern.

Solution

```
<style>
  .particle-container {
    width: 500px;
    height: 500px;
    position: relative;
    overflow: hidden;
  }
  .particle {
    width: 20px;
    height: 20px;
    background-color: blue;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    animation: particle-move 5s infinite;
  }
  @keyframes particle-move {
    0% {
      transform: translate(-50%, -50%);
    }
    50% {
      transform: translate(200px, 200px);
    }
    100% {
      transform: translate(-50%, -50%);
    }
  }
</style>
```

```

        }
    </style>

<div class="particle-container">
    <div class="particle"></div>
    <div class="particle"></div>
    <div class="particle"></div>
    <!-- Add more particles as needed -->
</div>

```

```

1  <style>
2      .particle-container {
3          width: 500px;
4          height: 500px;
5          position: relative;
6          overflow: hidden;
7      }
8
9      .particle {
10         width: 20px;
11         height: 20px;
12         background-color: blue;
13         position: absolute;
14         top: 50%;
15         left: 50%;
16         transform: translate(-50%, -50%);
17         animation: particle-move 5s infinite;
18     }
19
20     @keyframes particle-move {
21         0% {
22             transform: translate(-50%, -50%);
23         }
24         50% {
25             transform: translate(200px, 200px);
26         }
27         100% {
28             transform: translate(-50%, -50%);
29         }
30     }
31 </style>
32
33 <div class="particle-container">
34     <div class="particle"></div>
35     <div class="particle"></div>
36     <div class="particle"></div>
37     <!-- Add more particles as needed -->
38 </div>

```

In this solution, we have a particle animation where multiple divs represent the particles. The .particle-container acts as the container

for the particles and has a fixed width and height. Each .particle is positioned absolutely within the container and starts at the center using the transform property. The particle-move animation is applied to each particle, causing them to move in a predefined pattern. The @keyframes rule defines the different keyframe percentages and the corresponding translations. This creates an animation where particles move across the screen in a random or predefined pattern.

Chapter 8: CSS3 Transformations and 3D Effects

8.1 2D Transforms

CSS3 2D transforms allow you to manipulate elements in two-dimensional space, providing a range of visual effects.

Exercises

Exercise 1

Create a box that scales up and rotates continuously.

Solution

```
<style>
  .box {
    width: 100px;
    height: 100px;
    background-color: red;
    animation: scale-rotate 3s infinite;
  }
  @keyframes scale-rotate {
    0% {
      transform: scale(1) rotate(0deg);
    }
    50% {
      transform: scale(1.5) rotate(180deg);
    }
    100% {
      transform: scale(1) rotate(360deg);
    }
  }
</style>
<div class="box"></div>
```

```
1 <style>
2   .box {
3     width: 100px;
4     height: 100px;
5     background-color: red;
6     animation: scale-rotate 3s infinite;
7   }
8
9   @keyframes scale-rotate {
10    0% {
11      transform: scale(1) rotate(0deg);
12    }
13    50% {
14      transform: scale(1.5) rotate(180deg);
15    }
16    100% {
17      transform: scale(1) rotate(360deg);
18    }
19  }
20 </style>
21
22 <div class="box"></div>
```

In this example, we have a box represented by a div with a red background. The scale-rotate animation is applied to the box, causing it to scale up by 1.5x and rotate 180 degrees at the midpoint of the animation. The box completes a full rotation (360 degrees) by the end of the animation. The @keyframes rule defines the different keyframe percentages and the corresponding scale and rotation values.

Exercise 2

Create a flip card animation where a card flips horizontally to reveal its backside.

Solution

```
<style>
  .card {
    width: 200px;
    height: 200px;
    perspective: 1000px;
  }
  .front,
  .back {
    width: 100%;
    height: 100%;
    position: absolute;
    backface-visibility: hidden;
  }
  .front {
    background-color: blue;
    transform: rotateY(0deg);
    z-index: 2;
  }
  .back {
    background-color: green;
    transform: rotateY(180deg);
  }
  .card:hover .front {
    transform: rotateY(180deg);
  }
  .card:hover .back {
    transform: rotateY(0deg);
  }
</style>
```

```

        }
    </style>
<div class="card">
    <div class="front">Front</div>
    <div class="back">Back</div>
</div>

```

```

1  <style>
2      .card {
3          width: 200px;
4          height: 200px;
5          perspective: 1000px;
6      }
7
8      .front,
9      .back {
10         width: 100%;
11         height: 100%;
12         position: absolute;
13         backface-visibility: hidden;
14     }
15
16     .front {
17         background-color: blue;
18         transform: rotateY(0deg);
19         z-index: 2;
20     }
21
22     .back {
23         background-color: green;
24         transform: rotateY(180deg);
25     }
26
27     .card:hover .front {
28         transform: rotateY(180deg);
29     }
30
31     .card:hover .back {
32         transform: rotateY(0deg);
33     }
34 </style>
35
36 <div class="card">
37     <div class="front">Front</div>
38     <div class="back">Back</div>
39 </div>

```

In this example, we have a flip card animation. The `.card` div represents the card container, and it has a `perspective` value applied to create a 3D perspective effect. The front and back faces of the card are represented by the `.front` and `.back` divs, respectively. The

`rotateY` transform property is used to rotate the card around the Y-axis, creating the flip effect. When the user hovers over the card, the `.front` div rotates to reveal the `.back` div, and vice versa.

Exercise 3

Create an animation where an element translates horizontally and skews simultaneously.

Solution

```
<style>
  .element {
    width: 100px;
    height: 100px;
    background-color: purple;
    animation: translate-skew 3s infinite;
  }
  @keyframes translate-skew {
    0% {
      transform: translateX(0) skewX(0deg);
    }
    50% {
      transform: translateX(200px) skewX(45deg);
    }
    100% {
      transform: translateX(0) skewX(0deg);
    }
  }
</style>
<div class="element"></div>
```

```
1 <style>
2   .element {
3     width: 100px;
4     height: 100px;
5     background-color: purple;
6     animation: translate-skew 3s infinite;
7   }
8
9   @keyframes translate-skew {
10    0% {
11      transform: translateX(0) skewX(0deg);
12    }
13    50% {
14      transform: translateX(200px) skewX(45deg);
15    }
16    100% {
17      transform: translateX(0) skewX(0deg);
18    }
19  }
20 </style>
21
22 <div class="element"></div>
```

In this example, we have an element represented by a div with a purple background. The translate-skew animation is applied to the element, causing it to translate horizontally by 200 pixels and skew by 45 degrees at the midpoint of the animation. The element then returns to its original position and skew by the end of the animation. The @keyframes rule defines the different keyframe percentages and the corresponding translate and skew values.

8.2 3D Transforms and Perspective

CSS3 3D transforms allow you to manipulate elements in three-dimensional space, providing the ability to create immersive and visually engaging effects on web pages.

Exercises

Exercise 1

Create a rotating cube with different colors for each face.

Solution

```
<style>
  .cube {
    width: 200px;
    height: 200px;
    position: relative;
    transform-style: preserve-3d;
    animation: rotate-cube 6s infinite linear;
  }
  .face {
    width: 200px;
    height: 200px;
    position: absolute;
    border: 1px solid black;
    text-align: center;
    line-height: 200px;
    font-size: 20px;
  }
  .front {
    background-color: red;
    transform: translateZ(100px);
  }
  .back {
    background-color: green;
    transform: translateZ(-100px) rotateY(180deg);
  }
  .right {
    background-color: blue;
```

```
    transform: rotateY(90deg) translateZ(100px);  
}  
.left {  
    background-color: yellow;  
    transform: rotateY(-90deg) translateZ(100px);  
}  
.top {  
    background-color: orange;  
    transform: rotateX(90deg) translateZ(100px);  
}  
.bottom {  
    background-color: purple;  
    transform: rotateX(-90deg) translateZ(100px);  
}  
@keyframes rotate-cube {  
    0% {  
        transform: rotateY(0deg);  
    }  
    100% {  
        transform: rotateY(360deg);  
    }  
}  
/></style>  
<div class="cube">  
    <div class="face front">Front</div>  
    <div class="face back">Back</div>  
    <div class="face right">Right</div>  
    <div class="face left">Left</div>  
    <div class="face top">Top</div>
```

```
<div class="face bottom">Bottom</div>
</div>
```

```
1 <style>
2   .cube {
3     width: 200px;
4     height: 200px;
5     position: relative;
6     transform-style: preserve-3d;
7     animation: rotate-cube 6s infinite linear;
8   }
9
10  .face {
11    width: 200px;
12    height: 200px;
13    position: absolute;
14    border: 1px solid black;
15    text-align: center;
16    line-height: 200px;
17    font-size: 20px;
18  }
19
20  .front {
21    background-color: red;
22    transform: translateZ(100px);
23  }
24
25  .back {
26    background-color: green;
27    transform: translateZ(-100px) rotateY(180deg);
28  }
29
30  .right {
31    background-color: blue;
32    transform: rotateY(90deg) translateZ(100px);
33  }
34
35  .left {
36    background-color: yellow;
37    transform: rotateY(-90deg) translateZ(100px);
38  }
39
40  .top {
41    background-color: orange;
42    transform: rotateX(90deg) translateZ(100px);
43  }
44
45  .bottom {
46    background-color: purple;
47    transform: rotateX(-90deg) translateZ(100px);
48  }
49
```

```
49
50  @keyframes rotate-cube {
51    0% {
52      transform: rotateY(0deg);
53    }
54    100% {
55      transform: rotateY(360deg);
56    }
57  }
58 </style>
59
60 <div class="cube">
61   <div class="face front">Front</div>
62   <div class="face back">Back</div>
63   <div class="face right">Right</div>
64   <div class="face left">Left</div>
65   <div class="face top">Top</div>
66   <div class="face bottom">Bottom</div>
67 </div>
```

In this example, we have a cube represented by a container div with six different faces. Each face is represented by a div with a different background color. The .cube div has the transform-style: preserve-3d property to maintain the 3D space. Each face is positioned absolutely within the cube and translated along the Z-axis to create the cube shape. The rotate-cube animation is applied to the cube, causing it to rotate 360 degrees around the Y-axis.

Exercise 2

Create a carousel animation where a set of images rotates in a 3D space.

Solution

```
<style>

.carousel {
    width: 400px;
    height: 300px;
    position: relative;
    perspective: 800px;
}

.image {
    width: 100%;
    height: 100%;
    position: absolute;
    background-size: cover;
    background-position: center;
}

.image:nth-child(1) {
    background-image: url("image1.jpg");
    transform: rotateY(0deg) translateZ(200px);
}

.image:nth-child(2) {
    background-image: url("image2.jpg");
    transform: rotateY(120deg) translateZ(200px);
}

.image:nth-child(3) {
    background-image: url("image3.jpg");
    transform: rotateY(240deg) translateZ(200px);
}

@keyframes rotate-carousel {
    0% { transform: rotateY(0deg) translateZ(200px); }
    100% { transform: rotateY(360deg) translateZ(200px); }
}
```

```
0%  {
    transform: rotateY(0deg);
}
100% {
    transform: rotateY(360deg);
}
}

</style>
<div class="carousel">
    <div class="image"></div>
    <div class="image"></div>
    <div class="image"></div>
    <!-- Add more images as needed -->
</div>
```

```

1  <style>
2      .carousel {
3          width: 400px;
4          height: 300px;
5          position: relative;
6          perspective: 800px;
7      }
8
9      .image {
10         width: 100%;
11         height: 100%;
12         position: absolute;
13         background-size: cover;
14         background-position: center;
15     }
16
17     .image:nth-child(1) {
18         background-image: url("image1.jpg");
19         transform: rotateY(0deg) translateZ(200px);
20     }
21
22     .image:nth-child(2) {
23         background-image: url("image2.jpg");
24         transform: rotateY(120deg) translateZ(200px);
25     }
26
27     .image:nth-child(3) {
28         background-image: url("image3.jpg");
29         transform: rotateY(240deg) translateZ(200px);
30     }
31
32     @keyframes rotate-carousel {
33         0% {
34             transform: rotateY(0deg);
35         }
36         100% {
37             transform: rotateY(360deg);
38         }
39     }
40 </style>
41
42 <div class="carousel">
43     <div class="image"></div>
44     <div class="image"></div>
45     <div class="image"></div>
46     <!-- Add more images as needed -->
47 </div>

```

In this example, we have a carousel animation where multiple images are arranged in a circular 3D space. The .carousel div represents the carousel container and has the perspective property applied to create a 3D perspective effect. Each image is represented by a div with a background image. The images are positioned absolutely within the carousel and translated along the Z-axis to create the circular

arrangement. The rotate-carousel animation is applied to the carousel, causing it to rotate 360 degrees around the Y-axis.

Exercise 3

Create an interactive card flip animation where a card flips vertically on user interaction.

Solution

```
<style>

  .card {
    width: 200px;
    height: 300px;
    position: relative;
    perspective: 1000px;
  }

  .front,
  .back {
    width: 100%;
    height: 100%;
    position: absolute;
    backface-visibility: hidden;
    transition: transform 0.6s;
  }

  .front {
    background-color: blue;
    transform: rotateY(0deg);
  }

  .back {
    background-color: green;
    transform: rotateY(180deg);
  }

  .card.flipped .front {
    transform: rotateY(180deg);
  }

  .card.flipped .back {
```

```

        transform: rotateY(0deg);
    }

```

</style>

```

<div class="card" id="card">
    <div class="front">Front</div>
    <div class="back">Back</div>
</div>

```

```

1  <style>
2      .card {
3          width: 200px;
4          height: 300px;
5          position: relative;
6          perspective: 1000px;
7      }
8
9      .front,
10     .back {
11         width: 100%;
12         height: 100%;
13         position: absolute;
14         backface-visibility: hidden;
15         transition: transform 0.6s;
16     }
17
18     .front {
19         background-color: blue;
20         transform: rotateY(0deg);
21     }
22
23     .back {
24         background-color: green;
25         transform: rotateY(180deg);
26     }
27
28     .card.flipped .front {
29         transform: rotateY(180deg);
30     }
31
32     .card.flipped .back {
33         transform: rotateY(0deg);
34     }
35 </style>
36
37 <div class="card" id="card">
38     <div class="front">Front</div>
39     <div class="back">Back</div>
40 </div>

```

In this example, we have a card represented by a div with a front and back face. The .card div has the perspective property applied to

create a 3D perspective effect. Each face is positioned absolutely within the card and has the backface-visibility property set to hidden to hide the back face when it's not being viewed. The transform property is used to rotate the card around the Y-axis. When the .flipped class is added to the card, the front and back faces swap their rotation values, creating the flip effect. The transition property is applied to both faces to animate the rotation over 0.6 seconds.

8.3 Creating 3D Effects and Animations

CSS3 provides powerful features for creating 3D effects and animations on web pages.

Exercises

Exercise 1

Create a 3D box that rotates and changes colors on hover.

Solution

```
<style>
  .box {
    width: 200px;
    height: 200px;
    background-color: red;
    transition: transform 0.5s, background-color
0.5s;
  }
  .box:hover {
    transform: rotateY(180deg);
    background-color: blue;
  }
</style>
<div class="box"></div>
```

```
1 <style>
2   .box {
3     width: 200px;
4     height: 200px;
5     background-color: red;
6     transition: transform 0.5s, background-color 0.5s;
7   }
8
9   .box:hover {
10    transform: rotateY(180deg);
11    background-color: blue;
12  }
13 </style>
14
15 <div class="box"></div>
```

In this example, we have a box represented by a div with a red background. The .box div has a transition property applied to animate the transform and background-color changes. When the user hovers over the box, the rotateY transform is applied, causing the box to

rotate 180 degrees along the Y-axis. Additionally, the background-color changes to blue.

Exercise 2

Create a parallax scrolling effect where background layers move at different speeds as the user scrolls.

Solution

```
<style>
.parallax-container {
    height: 400px;
    overflow: hidden;
    perspective: 1px;
}
.background-layer {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
}
.layer1 {
    background-image: url("layer1.jpg");
    background-position: center;
    background-size: cover;
    transform: translateZ(-1px) scale(2);
}
.layer2 {
    background-image: url("layer2.jpg");
    background-position: center;
    background-size: cover;
    transform: translateZ(-2px) scale(3);
}
.layer3 {
    background-image: url("layer3.jpg");
```

```
background-position: center;
background-size: cover;
transform: translateZ(-3px) scale(4);
}

.content {
    position: relative;
    z-index: 1;
    /* Additional styles for content */
}

</style>

<div class="parallax-container">
    <div class="background-layer layer1"></div>
    <div class="background-layer layer2"></div>
    <div class="background-layer layer3"></div>
    <!-- Add more background layers as needed -->
    <div class="content">
        <!-- Your content goes here -->
    </div>
</div>
```

```

1  <style>
2    .parallax-container {
3      height: 400px;
4      overflow: hidden;
5      perspective: 1px;
6    }
7
8    .background-layer {
9      position: absolute;
10   top: 0;
11   left: 0;
12   width: 100%;
13   height: 100%;
14 }
15
16 .layer1 {
17   background-image: url("layer1.jpg");
18   background-position: center;
19   background-size: cover;
20   transform: translateZ(-1px) scale(2);
21 }
22
23 .layer2 {
24   background-image: url("layer2.jpg");
25   background-position: center;
26   background-size: cover;
27   transform: translateZ(-2px) scale(3);
28 }
29
30 .layer3 {
31   background-image: url("layer3.jpg");
32   background-position: center;
33   background-size: cover;
34   transform: translateZ(-3px) scale(4);
35 }
36
37 .content {
38   position: relative;
39   z-index: 1;
40   /* Additional styles for content */
41 }
42 </style>
43
44 <div class="parallax-container">
45   <div class="background-layer layer1"></div>
46   <div class="background-layer layer2"></div>
47   <div class="background-layer layer3"></div>
48   <!-- Add more background layers as needed -->
49   <div class="content">
50     <!-- Your content goes here -->
51   </div>
52 </div>

```

In this example, we have a parallax scrolling effect. The `.parallax-container` div represents the container for the effect. Each background layer is represented by a div with the `.background-layer` class. The layers are positioned absolutely and translated along the

Z-axis to create the parallax effect. The .content div represents the content that will appear on top of the background layers. The perspective property is applied to the container to create the 3D perspective effect.

Exercise 3

Create a 3D card flip animation where a card flips horizontally to reveal its backside.

Solution

```
<style>

  .card {
    width: 200px;
    height: 300px;
    position: relative;
    perspective: 1000px;
  }

  .front,
  .back {
    width: 100%;
    height: 100%;
    position: absolute;
    backface-visibility: hidden;
    transition: transform 0.6s;
  }

  .front {
    background-color: blue;
    transform: rotateY(0deg);
  }

  .back {
    background-color: green;
    transform: rotateY(180deg);
  }

  .card:hover .front {
    transform: rotateY(180deg);
  }

  .card:hover .back {
```

```

        transform: rotateY(0deg);
    }

```

</style>

<div class="card">

 <div class="front">Front</div>

 <div class="back">Back</div>

</div>

```

1  <style>
2      .card {
3          width: 200px;
4          height: 300px;
5          position: relative;
6          perspective: 1000px;
7      }
8
9      .front,
10     .back {
11         width: 100%;
12         height: 100%;
13         position: absolute;
14         backface-visibility: hidden;
15         transition: transform 0.6s;
16     }
17
18     .front {
19         background-color: blue;
20         transform: rotateY(0deg);
21     }
22
23     .back {
24         background-color: green;
25         transform: rotateY(180deg);
26     }
27
28     .card:hover .front {
29         transform: rotateY(180deg);
30     }
31
32     .card:hover .back {
33         transform: rotateY(0deg);
34     }
35 </style>
36
37 <div class="card">
38     <div class="front">Front</div>
39     <div class="back">Back</div>
40 </div>

```

In this example, we have a card represented by a div with a front and back face. The .card div has the perspective property applied to

create a 3D perspective effect. Each face is positioned absolutely within the card and has the backface-visibility property set to hidden to hide the back face when it's not being viewed. The transform property is used to rotate the card around the Y-axis. When the user hovers over the card, the front and back faces swap their rotation values, creating the flip effect. The transition property is applied to both faces to animate the rotation over 0.6 seconds.

Chapter 9: CSS3 Media Queries and Responsive Design

9.1 Media Queries and Breakpoints

In responsive web design, media queries are used to apply different CSS styles based on the characteristics of the user's device, such as screen size, resolution, or orientation. Breakpoints, on the other hand, are specific points in the design where the layout needs to change to accommodate different screen sizes.

Exercises

Exercise 1

Create a responsive grid layout that adjusts the number of columns based on the screen size.

Solution

```
<style>
  .grid {
    display: grid;
    grid-template-columns: repeat(auto-fit,
minmax(200px, 1fr));
    grid-gap: 20px;
  }
  .grid-item {
    background-color: #eaeaea;
    padding: 20px;
  }
</style>
<div class="grid">
  <div class="grid-item">Item 1</div>
  <div class="grid-item">Item 2</div>
  <div class="grid-item">Item 3</div>
  <!-- Add more grid items as needed -->
</div>
```

```
1 <style>
2   .grid {
3     display: grid;
4     grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
5     grid-gap: 20px;
6   }
7
8   .grid-item {
9     background-color: #eaeaea;
10    padding: 20px;
11  }
12 </style>
13
14 <div class="grid">
15   <div class="grid-item">Item 1</div>
16   <div class="grid-item">Item 2</div>
17   <div class="grid-item">Item 3</div>
18   <!-- Add more grid items as needed -->
19 </div>
```

In this example, we have a responsive grid layout using CSS Grid. The .grid class sets up a grid container with columns that automatically adjust based on the available space. The repeat(auto-fit, minmax(200px, 1fr)) syntax ensures that the columns are responsive and collapse when the screen size is smaller, but each column will have a minimum width of 200px and expand to fill the available space using the 1fr unit. The .grid-item class represents the individual items within the grid and can be customized as needed.

Exercise 2

Hide or show specific elements based on the screen size using media queries.

Solution

```
<style>
  .element {
    display: block;
  }
  @media screen and (max-width: 600px) {
    .element:nth-child(2) {
      display: none;
    }
  }
</style>
<div class="container">
  <div class="element">Element 1</div>
  <div class="element">Element 2</div>
  <div class="element">Element 3</div>
  <!-- Add more elements as needed -->
</div>
```

```
1  <style>
2    .element {
3      display: block;
4    }
5
6    @media screen and (max-width: 600px) {
7      .element:nth-child(2) {
8        display: none;
9      }
10     }
11   </style>
12
13  <div class="container">
14    <div class="element">Element 1</div>
15    <div class="element">Element 2</div>
16    <div class="element">Element 3</div>
17    <!-- Add more elements as needed -->
18  </div>
```

In this example, we have a container with multiple elements represented by the .element class. By default, all elements are displayed as blocks. However, using a media query, we target screens with a maximum width of 600px and hide the second element using the display: none; property. You can adjust the breakpoint and customize the targeted elements based on your specific requirements.

Exercise 3

Make typography adjustments, such as font size and line height, based on different screen sizes using media queries.

Solution

```
<style>
  .heading {
    font-size: 36px;
    line-height: 1.2;
  }
  .text {
    font-size: 16px;
    line-height: 1.5;
  }
  @media screen and (max-width: 600px) {
    .heading {
      font-size: 24px;
    }
    .text {
      font-size: 14px;
    }
  }
</style>
<h1 class="heading">Main Heading</h1>
<p class="text">Some text content here.</p>
```

```
1 <style>
2   .heading {
3     font-size: 36px;
4     line-height: 1.2;
5   }
6
7   .text {
8     font-size: 16px;
9     line-height: 1.5;
10  }
11
12 @media screen and (max-width: 600px) {
13   .heading {
14     font-size: 24px;
15   }
16
17   .text {
18     font-size: 14px;
19   }
20 }
21 </style>
22
23 <h1 class="heading">Main Heading</h1>
24 <p class="text">Some text content here.</p>
```

In this example, we have a main heading represented by the `.heading` class and a paragraph of text represented by the `.text` class. Initially, the font sizes and line heights are set to specific values. However, using a media query, we target screens with a maximum width of 600px and adjust the font sizes for the heading and text elements. You can modify the breakpoints and adjust other typography properties based on your design needs.

9.2 Responsive Layouts with CSS3

Creating responsive layouts with CSS3 involves using various techniques and properties to ensure that your website adapts and looks good on different screen sizes.

Exercises

Exercise 1

Create a responsive navigation menu that adjusts its layout and appearance for different screen sizes.

Solution

```
<style>
  .navigation {
    background-color: #f0f0f0;
  }
  .menu {
    display: flex;
    justify-content: space-between;
    padding: 0;
    margin: 0;
    list-style: none;
  }
  .menu li {
    margin: 0 10px;
  }
  @media screen and (max-width: 600px) {
    .menu {
      flex-direction: column;
      align-items: center;
    }
    .menu li {
      margin: 10px 0;
    }
  }
</style>
<nav class="navigation">
  <ul class="menu">
    <li><a href="#">Home</a></li>
```

```

<li><a href="#">About</a></li>
<li><a href="#">Services</a></li>
<li><a href="#">Contact</a></li>
</ul>
</nav>
```

```

1 <style>
2   .navigation {
3     background-color: #f0f0f0;
4   }
5
6   .menu {
7     display: flex;
8     justify-content: space-between;
9     padding: 0;
10    margin: 0;
11    list-style: none;
12  }
13
14  .menu li {
15    margin: 0 10px;
16  }
17
18 @media screen and (max-width: 600px) {
19   .menu {
20     flex-direction: column;
21     align-items: center;
22   }
23
24   .menu li {
25     margin: 10px 0;
26   }
27 }
28 </style>
29
30 <nav class="navigation">
31   <ul class="menu">
32     <li><a href="#">Home</a></li>
33     <li><a href="#">About</a></li>
34     <li><a href="#">Services</a></li>
35     <li><a href="#">Contact</a></li>
36   </ul>
37 </nav>
```

In this example, we have a navigation menu represented by the `<nav>` element and a list of menu items represented by the `` element. The menu items are displayed horizontally using flexbox and have some margin between them. Using a media query, we target screens with a maximum width of 600px and change the flex-direction to column, which makes the menu items stack vertically. We

also adjust the margin to provide spacing between the stacked menu items.

Exercise 2

Create a layout with fluid images that automatically scale to fit the screen size while maintaining their aspect ratio.

Solution

```
<style>
  .image-container {
    max-width: 100%;
    height: auto;
  }
  .image-container img {
    width: 100%;
    height: auto;
  }
</style>
<div class="image-container">
  
</div>
```

```
1  <style>
2    .image-container {
3      max-width: 100%;
4      height: auto;
5    }
6
7    .image-container img {
8      width: 100%;
9      height: auto;
10   }
11  </style>
12
13 <div class="image-container">
14   
15 </div>
```

In this example, we have an image contained within a `<div>` with the `.image-container` class. The image is set to have a maximum width of 100% of its container and a height that adjusts automatically to maintain the original aspect ratio. This allows the image to scale fluidly and fit within the available space without distortion.

Exercise 3

Create a responsive grid of cards that rearranges and adjusts its layout based on the screen size.

Solution

```
<style>
  .card-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit,
minmax(200px, 1fr));
    grid-gap: 20px;
  }
  .card {
    background-color: #eaeaea;
    padding: 20px;
  }
</style>
<div class="card-grid">
  <div class="card">Card 1</div>
  <div class="card">Card 2</div>
  <div class="card">Card 3</div>
  <!-- Add more cards as needed -->
</div>
```

```
1 <style>
2   .card-grid {
3     display: grid;
4     grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
5     grid-gap: 20px;
6   }
7
8   .card {
9     background-color: #eaeaea;
10    padding: 20px;
11  }
12 </style>
13
14 <div class="card-grid">
15   <div class="card">Card 1</div>
16   <div class="card">Card 2</div>
17   <div class="card">Card 3</div>
18   <!-- Add more cards as needed -->
19 </div>
```

In this example, we have a card grid layout represented by the .card-grid class. The cards are arranged in a grid using the CSS Grid layout. The grid-template-columns property is set to repeat(auto-fit, minmax(200px, 1fr)), which allows the columns to automatically adjust based on the available space. Each card has a minimum width of 200px and expands to fill the available space using the 1fr unit. The grid-gap property adds some spacing between the cards.

9.3 Optimizing Images for Different Devices

When it comes to optimizing images for different devices using CSS3, there are several techniques you can employ to ensure that images load quickly and look good on various screen sizes.

Exercises

Exercise 1

Optimize an image to display a higher resolution version for devices with high pixel density (e.g., Retina displays).

Solution

```
<style>
  .retina-image {
    background-image: url("image.jpg");
    background-size: cover;
  }
  @media (-webkit-min-device-pixel-ratio: 2), (min-resolution: 192dpi) {
    .retina-image {
      background-image: url("image@2x.jpg");
    }
  }
</style>

```

```
1 <style>
2   .retina-image {
3     background-image: url("image.jpg");
4     background-size: cover;
5   }
6
7   @media (-webkit-min-device-pixel-ratio: 2), (min-resolution: 192dpi) {
8     .retina-image {
9       background-image: url("image@2x.jpg");
10    }
11  }
12 </style>
13
14 
```

In this example, we have an image represented by the `img` element with the `.retina-image` class. By default, the image is loaded from the `image.jpg` URL. However, using a media query, we target devices with a high pixel density, such as Retina displays, and replace the image source with a higher resolution version from the `image@2x.jpg`

URL. This ensures that the image looks crisp and sharp on high-density screens.

Exercise 2

Use the picture element to provide different image sources based on the device's screen size or orientation.

Solution

```
<picture>
  <source srcset="image-desktop.jpg" media="(min-width: 768px)" />
  <source srcset="image-mobile.jpg" media="(max-width: 767px)" />
  
</picture>
```

```
1 <picture>
2   <source srcset="image-desktop.jpg" media="(min-width: 768px)" />
3   <source srcset="image-mobile.jpg" media="(max-width: 767px)" />
4   
5 </picture>
```

In this example, we use the picture element to provide different image sources based on the device's screen size. Inside the picture element, we have two source elements with the srcset attribute specifying different image sources. The first source element targets devices with a minimum width of 768px and loads the image-desktop.jpg image. The second source element targets devices with a maximum width of 767px and loads the image-mobile.jpg image. The img element serves as a fallback and loads the image-desktop.jpg image by default. This allows us to provide different images optimized for different screen sizes or device orientations.

Exercise 3

Implement lazy loading techniques to load images only when they are needed, improving performance and reducing data usage.

Solution

```
<style>
    .lazy-load {
        opacity: 0;
        transition: opacity 0.3s ease-in-out;
    }
    .lazy-load.loaded {
        opacity: 1;
    }
</style>
<script>
    document.addEventListener("DOMContentLoaded",
    function () {
        var lazyLoadImages =
        [].slice.call(document.querySelectorAll(".lazy-
load"));
        if ("IntersectionObserver" in window) {
            var observer = new
            IntersectionObserver(function (entries, observer) {
                entries.forEach(function (entry) {
                    if (entry.isIntersecting) {
                        var lazyImage = entry.target;
                        lazyImage.src = lazyImage.dataset.src;
                        lazyImage.classList.add("loaded");
                        observer.unobserve(lazyImage);
                    }
                }) ;
            }) ;
        } ;
    }) ;
```

```
    lazyLoadImages.forEach(function
(lazyLoadImage) {
    observer.observe(lazyLoadImage);
})
} else {
// Fallback for browsers that don't support
Intersection Observer
    lazyLoadImages.forEach(function
(lazyLoadImage) {
    lazyLoadImage.src =
lazyLoadImage.dataset.src;
    lazyLoadImage.classList.add("loaded");
})
}
})
}

}) ;
</script>

```

```

1  <style>
2      .lazy-load {
3          opacity: 0;
4          transition: opacity 0.3s ease-in-out;
5      }
6
7      .lazy-load.loaded {
8          opacity: 1;
9      }
10 </style>
11
12 <script>
13     document.addEventListener("DOMContentLoaded", function () {
14         var lazyLoadImages = [].slice.call(document.querySelectorAll(".lazy-load"));
15
16         if ("IntersectionObserver" in window) {
17             var observer = new IntersectionObserver(function (entries, observer) {
18                 entries.forEach(function (entry) {
19                     if (entry.isIntersecting) {
20                         var lazyImage = entry.target;
21                         lazyImage.src = lazyImage.dataset.src;
22                         lazyImage.classList.add("loaded");
23                         observer.unobserve(lazyImage);
24                     }
25                 });
26             });
27
28             lazyLoadImages.forEach(function (lazyLoadImage) {
29                 observer.observe(lazyLoadImage);
30             });
31         } else {
32             // Fallback for browsers that don't support Intersection Observer
33             lazyLoadImages.forEach(function (lazyLoadImage) {
34                 lazyLoadImage.src = lazyLoadImage.dataset.src;
35                 lazyLoadImage.classList.add("loaded");
36             });
37         }
38     });
39 </script>
40
41 
42

```

In this example, we implement lazy loading techniques to load images only when they are needed. The image initially loads a placeholder image (placeholder.jpg) using the src attribute. The actual image source (image.jpg) is stored in the data-src attribute. The lazy-load class is applied to the image, which initially sets its opacity to 0 and adds a transition effect. Using JavaScript, we use the Intersection Observer API to detect when the image enters the viewport. Once the image is visible, we replace the placeholder

source with the actual image source, update the opacity to 1, and add the loaded class to reveal the image. If the browser does not support Intersection Observer, we provide a fallback that immediately loads the image.

Chapter 10: Advanced CSS3 Techniques

10.1 CSS Variables and Custom Properties

CSS3 variables, also known as custom properties, allow you to define reusable values in CSS that can be used throughout your stylesheets. They provide a way to store and manage values in a central location, making it easier to update and maintain your styles.

Exercises

Exercise 1

Create a theme switcher that changes the color scheme of a website using CSS variables.

Solution

```
<style>
:root {
    --primary-color: #ff0000;
}

.box {
    width: 200px;
    height: 200px;
    background-color: var(--primary-color);
}

.dark-theme {
    --primary-color: #000000;
}

</style>
<script>
    const themeSwitcher =
        document.getElementById("theme-switcher");
    const box = document.querySelector(".box");
    themeSwitcher.addEventListener("click", function
() {
    box.classList.toggle("dark-theme");
})
</script>
<button id="theme-switcher">Switch Theme</button>
<div class="box"></div>
```

```
1 <style>
2   :root {
3     --primary-color: #ff0000;
4   }
5
6   .box {
7     width: 200px;
8     height: 200px;
9     background-color: var(--primary-color);
10  }
11
12  .dark-theme {
13    --primary-color: #000000;
14  }
15 </style>
16
17 <script>
18   const themeSwitcher = document.getElementById("theme-switcher");
19   const box = document.querySelector(".box");
20
21   themeSwitcher.addEventListener("click", function () {
22     box.classList.toggle("dark-theme");
23   });
24 </script>
25
26 <button id="theme-switcher">Switch Theme</button>
27 <div class="box"></div>
```

In this example, we define a CSS variable `--primary-color` in the `:root` pseudo-class, which sets the primary color for the website to `#ff0000` (red) by default. The `.box` element uses this variable to set its background color. By adding the `.dark-theme` class to the `.box` element, we override the `--primary-color` variable to `#000000` (black). The JavaScript code adds an event listener to the theme switcher button, toggling the `.dark-theme` class on the `.box` element when clicked. This effectively switches the theme between the default and dark color schemes.

Exercise 2

Implement dynamic font sizes using CSS variables to allow users to adjust the font size of a webpage.

Solution

```
<style>
  :root {
    --font-size: 16px;
  }
  .content {
    font-size: var(--font-size);
  }
</style>
<script>
  const increaseFontSizeButton =
document.getElementById("increase-font-size");
  const decreaseFontSizeButton =
document.getElementById("decrease-font-size");
  const content =
document.querySelector(".content");
  increaseFontSizeButton.addEventListener("click",
function () {
  let currentFontSize =
parseFloat(getComputedStyle(content).fontSize);
  content.style.fontSize = `${currentFontSize +
2}px`;
});
  decreaseFontSizeButton.addEventListener("click",
function () {
  let currentFontSize =
parseFloat(getComputedStyle(content).fontSize);
```

```

        content.style.fontSize = `${currentFontSize -
2}px;
    });
</script>
<button id="increase-font-size">Increase Font
Size</button>
<button id="decrease-font-size">Decrease Font
Size</button>
<p class="content">Lorem ipsum dolor sit amet,
consectetur adipiscing elit.</p>

```

```

1  <style>
2    :root {
3      --font-size: 16px;
4    }
5
6    .content {
7      font-size: var(--font-size);
8    }
9  </style>
10
11 <script>
12   const increaseFontSizeButton = document.getElementById("increase-font-size");
13   const decreaseFontSizeButton = document.getElementById("decrease-font-size");
14   const content = document.querySelector(".content");
15
16   increaseFontSizeButton.addEventListener("click", function () {
17     let currentFontSize = parseFloat(getComputedStyle(content).fontSize);
18     content.style.fontSize = `${currentFontSize + 2}px`;
19   });
20
21   decreaseFontSizeButton.addEventListener("click", function () {
22     let currentFontSize = parseFloat(getComputedStyle(content).fontSize);
23     content.style.fontSize = `${currentFontSize - 2}px`;
24   });
25 </script>
26
27 <button id="increase-font-size">Increase Font Size</button>
28 <button id="decrease-font-size">Decrease Font Size</button>
29 <p class="content">Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>

```

In this example, we define a CSS variable `--font-size` in the `:root` pseudo-class, which sets the base font size for the website to 16px by default. The `.content` element uses this variable to set its font size. The JavaScript code adds event listeners to the increase and

decrease font size buttons. When clicked, these buttons retrieve the current font size of the .content element using getComputedStyle, adjust it by adding or subtracting 2px, and set the new font size using the style property. This allows users to dynamically increase or decrease the font size of the webpage.

Exercise 3

Use CSS variables to define responsive breakpoints for different screen sizes and adjust the layout accordingly.

Solution

```
<style>
:root {
    --breakpoint-small: 600px;
    --breakpoint-medium: 900px;
}

.box {
    width: 200px;
    height: 200px;
    background-color: red;
}

@media screen and (min-width: var(--breakpoint-small)) and (max-width: var(--breakpoint-medium)) {
    .box {
        background-color: green;
    }
}

@media screen and (min-width: var(--breakpoint-medium)) {
    .box {
        background-color: blue;
    }
}
</style>
<div class="box"></div>
```

```
1 <style>
2   :root {
3     --breakpoint-small: 600px;
4     --breakpoint-medium: 900px;
5   }
6
7   .box {
8     width: 200px;
9     height: 200px;
10    background-color: red;
11  }
12
13 @media screen and (min-width: var(--breakpoint-small)) and (max-width: var(--breakpoint-medium)) {
14   .box {
15     background-color: green;
16   }
17 }
18
19 @media screen and (min-width: var(--breakpoint-medium)) {
20   .box {
21     background-color: blue;
22   }
23 }
24 </style>
25
26 <div class="box"></div>
```

In this example, we define CSS variables `--breakpoint-small` and `--breakpoint-medium` in the `:root` pseudo-class, which set the responsive breakpoints for different screen sizes. The `.box` element has a default background color of red. Using media queries, we target specific screen sizes based on the defined breakpoints. When the screen width is between the small and medium breakpoints, the `.box` element's background color changes to green. When the screen width is equal to or larger than the medium breakpoint, the background color changes to blue. This allows the layout to adapt and adjust its appearance based on the screen size.

10.2 CSS Calc() Function

The calc() function in CSS3 allows you to perform mathematical calculations to determine values for CSS properties. It enables you to combine different units of measurement, perform basic arithmetic operations, and create more dynamic and flexible styles.

Exercises

Exercise 1

Create a responsive box with a fixed width and height, and adjust the padding based on the width of the box using the calc() function.

Solution

```
<style>
  .box {
    width: 200px;
    height: 200px;
    padding: calc(10px + 2%); /* Adjust padding
based on box width */
    background-color: red;
  }
</style>
<div class="box"></div>
```

```
1 <style>
2   .box {
3     width: 200px;
4     height: 200px;
5     padding: calc(10px + 2%); /* Adjust padding based on box width */
6     background-color: red;
7   }
8 </style>
9
10 <div class="box"></div>
```

In this example, we set the width and height of the `.box` element to 200px each. The padding property is set using the `calc()` function, which allows us to add a fixed value of 10px to a percentage value of 2% of the box width. As the box width changes, the padding will dynamically adjust, ensuring that the box remains responsive.

Exercise 2

Create a centered layout with a fixed width and height, and adjust the margins to center the content using the calc() function.

Solution

```
<style>
  .container {
    width: 100%;
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
  }
  .content {
    width: calc(100% - 40px); /* Adjust margins to
center content */
    height: 200px;
    background-color: blue;
    color: white;
    text-align: center;
    margin: calc(10vh + 20px) auto; /* Adjust
margins based on container height */
  }
</style>
<div class="container">
  <div class="content">Centered Content</div>
</div>
```

```
1 <style>
2   .container {
3     width: 100%;
4     height: 100vh;
5     display: flex;
6     justify-content: center;
7     align-items: center;
8   }
9
10  .content {
11    width: calc(100% - 40px); /* Adjust margins to center content */
12    height: 200px;
13    background-color: blue;
14    color: white;
15    text-align: center;
16    margin: calc(10vh + 20px) auto; /* Adjust margins based on container height */
17  }
18 </style>
19
20 <div class="container">
21   <div class="content">Centered Content</div>
22 </div>
```

In this example, the `.container` element spans the entire width and height of the viewport using `width: 100%` and `height: 100vh`. The `display: flex` property is used to center the content both horizontally (`justify-content: center`) and vertically (`align-items: center`). The `.content` element has a fixed height of `200px` and adjusts its margins using the `calc()` function. The top and bottom margins are set to `10vh + 20px`, which combines a percentage value of `10vh` of the container height with a fixed value of `20px`. The left and right margins are set to `auto`, which centers the element horizontally within the container.

Exercise 3

Implement fluid typography by adjusting the font size based on the viewport width using the calc() function.

Solution

```
<style>
  .text {
    font-size: calc(16px + 1vw); /* Adjust font
size based on viewport width */
  }
</style>
<p class="text">Fluid Typography</p>
```

```
1 <style>
2   .text {
3     font-size: calc(16px + 1vw); /* Adjust font size based on viewport width */
4   }
5 </style>
6
7 <p class="text">Fluid Typography</p>
```

In this example, we set the font size of the `.text` element using the `calc()` function. The base font size is `16px`, and we add `1vw` (viewport width) to it. As the viewport width changes, the font size will scale proportionally. This creates a fluid typography effect where the text adapts and adjusts its size based on the available width.

10.3 CSS Filters and Blend Modes

CSS3 filters and blend modes are powerful features that allow you to apply visual effects and manipulate the appearance of elements on a webpage.

Exercises

Exercise 1

Create an image overlay effect by applying a blend mode to a semi-transparent overlay element.

Solution

```
<style>
  .image-container {
    position: relative;
    display: inline-block;
  }
  .overlay {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.5);
    mix-blend-mode: multiply; /* Apply blend mode
  */
}
</style>
<div class="image-container">
  
  <div class="overlay"></div>
</div>
```

```
1 <style>
2   .image-container {
3     position: relative;
4     display: inline-block;
5   }
6
7   .overlay {
8     position: absolute;
9     top: 0;
10    left: 0;
11    width: 100%;
12    height: 100%;
13    background-color: rgba(0, 0, 0, 0.5);
14    mix-blend-mode: multiply; /* Apply blend mode */
15  }
16 </style>
17
18 <div class="image-container">
19   
20   <div class="overlay"></div>
21 </div>
```

In this example, we have an image inside an `.image-container` element. We add a semi-transparent overlay element with the `.overlay` class on top of the image. The overlay has a background color of `rgba(0, 0, 0, 0.5)`, which creates a black overlay with 50% opacity. The `mix-blend-mode` property is set to `multiply`, which multiplies the pixel values of the overlay with the image underneath, resulting in a darkened effect.

Exercise 2

Apply a grayscale filter to an image and add a hover effect that restores the original color.

Solution

```
<style>
  .image-container {
    display: inline-block;
  }
  .image-container img {
    filter: grayscale(100%); /* Apply grayscale
filter */
  }
  .image-container:hover img {
    filter: none; /* Remove grayscale filter on
hover */
  }
</style>
<div class="image-container">
  
</div>
```

```
1  <style>
2    .image-container {
3      display: inline-block;
4    }
5
6    .image-container img {
7      filter: grayscale(100%); /* Apply grayscale filter */
8    }
9
10   .image-container:hover img {
11     filter: none; /* Remove grayscale filter on hover */
12   }
13 </style>
14
15 <div class="image-container">
16   
17 </div>
```

In this example, we apply a grayscale filter to the image using the filter property with the value grayscale(100%). This turns the image

into grayscale. Then, on hover of the .image-container element, we remove the grayscale filter by setting filter: none, restoring the original color of the image.

Exercise 3

Create a duotone effect by combining two different colors using blend modes.

Solution

```
<style>
  .image-container {
    position: relative;
    display: inline-block;
  }
  .overlay {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: #ff0000; /* First color */
    mix-blend-mode: multiply; /* Apply blend mode
  */
}
.overlay::before {
  content: "";
  display: block;
  width: 100%;
  height: 100%;
  background-color: #00ff00; /* Second color */
  mix-blend-mode: screen; /* Apply blend mode */
}
</style>
<div class="image-container">
  
  <div class="overlay"></div>
```

```
</div>
```

```
1  <style>
2    .image-container {
3      position: relative;
4      display: inline-block;
5    }
6
7    .overlay {
8      position: absolute;
9      top: 0;
10     left: 0;
11     width: 100%;
12     height: 100%;
13     background-color: #ff0000; /* First color */
14     mix-blend-mode: multiply; /* Apply blend mode */
15   }
16
17   .overlay::before {
18     content: "";
19     display: block;
20     width: 100%;
21     height: 100%;
22     background-color: #00ff00; /* Second color */
23     mix-blend-mode: screen; /* Apply blend mode */
24   }
25 </style>
26
27 <div class="image-container">
28   
29   <div class="overlay"></div>
30 </div>
```

In this example, we have an image inside an `.image-container` element. We add an overlay element with the `.overlay` class on top of the image. The overlay is a block element that covers the entire image. The `background-color` property is set to the first color, `#ff0000`, which creates a red overlay. We apply the `multiply` blend mode to the overlay using `mix-blend-mode`. Additionally, we use the `::before` pseudo-element to add another color, `#00ff00`, to the overlay. This creates a duotone effect by combining the two colors using the `multiply` and `screen` blend modes.

10.4 Working with SVG in CSS3

Working with SVG (Scalable Vector Graphics) in CSS3 allows you to apply styles and transformations to SVG elements, making them more visually appealing and interactive.

Exercises

Exercise 1

Create a background pattern using an SVG image and apply it to an HTML element using CSS.

Solution

```
<style>
  .pattern {
    width: 400px;
    height: 400px;
    background-image: url(pattern.svg);
    background-size: 100px 100px;
    background-repeat: repeat;
  }
</style>
<div class="pattern"></div>
```

```
1 <style>
2   .pattern {
3     width: 400px;
4     height: 400px;
5     background-image: url(pattern.svg);
6     background-size: 100px 100px;
7     background-repeat: repeat;
8   }
9 </style>
10
11 <div class="pattern"></div>
```

In this example, you can use an SVG image file called pattern.svg. The SVG should contain the pattern you want to use as a background. In CSS, you set the background image using background-image and specify the path to the SVG file. Adjust the background-size property to control the size of the pattern, and set background-repeat to repeat to make the pattern repeat across the element.

Exercise 2

Animate an SVG icon using CSS keyframes and transforms.

Solution

```
<style>
  @keyframes spin {
    from {
      transform: rotate(0deg);
    }
    to {
      transform: rotate(360deg);
    }
  }
  .icon {
    width: 24px;
    height: 24px;
    animation: spin 2s linear infinite;
  }
</style>
<svg class="icon" viewBox="0 0 24 24">
  <path
    d="M12 2C6.48 2 2 6.48 2 12s4.48 10 10 10 10-4.48 10-10S17.52 2 12 2zm0 18c-4.41 0-8-3.59-8-8s3.59-8 8-8 8 3.59 8 8-3.59 8-8 8zm-1-13V7h2v2h-2zm0 4V9h2v4h-2z"
  />
</svg>
```

```
1 <style>
2   @keyframes spin {
3     from {
4       transform: rotate(0deg);
5     }
6     to {
7       transform: rotate(360deg);
8     }
9   }
10
11   .icon {
12     width: 24px;
13     height: 24px;
14     animation: spin 2s linear infinite;
15   }
16 </style>
17
18 <svg class="icon" viewBox="0 0 24 24">
19   <path
20     d="M12 2C6.48 2 2 6.48 2 12s4.48 10 10 10 10-4.48 10-10S17.52 2 12 2zm0 18c-4.41 0-8-3.59-8-8s3.59-8
-8 8 3.59 8-3.59 8-8 8zm-1-13V7h2v2h-2zm0 4V9h2v4h-2z"
21   />
22 </svg>
```

In this example, you have an SVG icon represented by the `<svg>` element. The `viewBox` attribute defines the coordinate system and aspect ratio of the SVG image. The `<path>` element inside the `<svg>` element contains the actual path of the icon. In CSS, you define a keyframe animation called `spin` that rotates the icon from 0 to 360 degrees. Then you apply the animation to the `.icon` class, setting it to run for 2 seconds with a linear timing function and repeat indefinitely.

Exercise 3

Animate text along a path in an SVG using CSS animations.

Solution

```
<style>
  @keyframes textAnimation {
    0% {
      startoffset: 0%;
    }
    100% {
      startoffset: 100%;
    }
  }
  .svg-container {
    width: 200px;
    height: 200px;
  }
  .text {
    font-size: 16px;
    fill: black;
    animation: textAnimation 5s linear infinite;
  }
</style>
<svg class="svg-container" viewBox="0 0 200 200">
  <path
    id="path"
    d="M20,100 C20,50 180,50 180,100 C180,150
    20,150 20,100"
  ></path>
  <text class="text" dy="-5">
```

```

<textPath xlink:href="#path">Animated
Text</textPath>
</text>
</svg>

```

```

1  <style>
2    @keyframes textAnimation {
3      0% {
4        startoffset: 0%;
5      }
6      100% {
7        startoffset: 100%;
8      }
9    }
10
11   .svg-container {
12     width: 200px;
13     height: 200px;
14   }
15
16   .text {
17     font-size: 16px;
18     fill: black;
19     animation: textAnimation 5s linear infinite;
20   }
21 </style>
22
23 <svg class="svg-container" viewBox="0 0 200 200">
24   <path
25     id="path"
26     d="M20,100 C20,50 180,50 180,100 C180,150 20,150 20,100"
27   ></path>
28   <text class="text" dy="-5">
29     <textPath xlink:href="#path">Animated Text</textPath>
30   </text>
31 </svg>

```

In this example, you have an SVG container with a specified viewBox to define the coordinate system. Inside the SVG, there is a <path> element that represents the path along which the text will be animated. The <text> element contains the text you want to animate, and the <textPath> element references the path using the xlink:href attribute. In CSS, you define a keyframe animation called textAnimation that animates the start offset of the text along the path. The animation runs for 5 seconds with a linear timing function and repeats indefinitely.

Chapter 11: CSS Best Practices and Optimization

11.1 Writing Efficient and Maintainable CSS

Writing efficient and maintainable CSS3 code is essential for creating scalable and maintainable stylesheets.

Exercises

Exercise 1

Refactor the following CSS code to reduce specificity and improve maintainability:

```
.container div.content {  
    background-color: #ffffff;  
    color: #333333;  
}  
.container div.content p.intro {  
    font-size: 18px;  
}  
.container div.content p.main {  
    font-size: 16px;  
}
```

```
1 .container div.content {  
2     background-color: #ffffff;  
3     color: #333333;  
4 }  
5  
6 .container div.content p.intro {  
7     font-size: 18px;  
8 }  
9  
10 .container div.content p.main {  
11     font-size: 16px;  
12 }
```

Solution

```
.container {  
    background-color: #ffffff;  
    color: #333333;  
}  
.container p.intro {  
    font-size: 18px;  
}  
.container p.main {  
    font-size: 16px;  
}
```

```
1 .container {  
2     background-color: #ffffff;  
3     color: #333333;  
4 }  
5  
6 .container p.intro {  
7     font-size: 18px;  
8 }  
9  
10 .container p.main {  
11     font-size: 16px;  
12 }
```

In this solution, we remove the unnecessary specificity by targeting the `.container` class directly instead of `div.content`. By doing so, we simplify the selectors and reduce the specificity while achieving the same styling.

Exercise 2

Convert the following CSS code to use the Block Element Modifier (BEM) methodology for better modularity:

```
.header {  
    background-color: #333333;  
}  
  
.header h1 {  
    color: #ffffff;  
    font-size: 24px;  
}  
  
.header .nav {  
    margin-top: 10px;  
}  
  
.header .nav li {  
    display: inline-block;  
    margin-right: 10px;  
}  
  
.header .nav li a {  
    color: #ffffff;  
    text-decoration: none;  
}
```

```
1 .header {  
2   background-color: #333333;  
3 }  
4  
5 .header h1 {  
6   color: #ffffff;  
7   font-size: 24px;  
8 }  
9  
10 .header .nav {  
11   margin-top: 10px;  
12 }  
13  
14 .header .nav li {  
15   display: inline-block;  
16   margin-right: 10px;  
17 }  
18  
19 .header .nav li a {  
20   color: #ffffff;  
21   text-decoration: none;  
22 }
```

Solution

```
.header {  
    background-color: #333333;  
}  
.header__title {  
    color: #ffffff;  
    font-size: 24px;  
}  
.header__nav {  
    margin-top: 10px;  
}  
.header__nav-item {  
    display: inline-block;  
    margin-right: 10px;  
}  
.header__nav-link {  
    color: #ffffff;  
    text-decoration: none;  
}
```

```
1 .header {  
2   background-color: #333333;  
3 }  
4  
5 .header__title {  
6   color: #ffffff;  
7   font-size: 24px;  
8 }  
9  
10 .header__nav {  
11   margin-top: 10px;  
12 }  
13  
14 .header__nav-item {  
15   display: inline-block;  
16   margin-right: 10px;  
17 }  
18  
19 .header__nav-link {  
20   color: #ffffff;  
21   text-decoration: none;  
22 }
```

In this solution, we follow the Block Element Modifier (BEM) methodology to create more modular and maintainable CSS. Each block and element is separated by double underscores (`__`). This approach makes it easier to understand the structure and relationships between different parts of the HTML, and it promotes reusability of styles.

Exercise 3

Refactor the following CSS code to use preprocessor variables for improved maintainability:

```
.container {  
    background-color: #ffffff;  
    color: #333333;  
}  
.container h1 {  
    font-size: 24px;  
}  
.container p {  
    font-size: 16px;  
}
```

```
1 .container {  
2     background-color: #ffffff;  
3     color: #333333;  
4 }  
5  
6 .container h1 {  
7     font-size: 24px;  
8 }  
9  
10 .container p {  
11     font-size: 16px;  
12 }
```

Solution

```
@color-white: #ffffff;
@color-dark: #333333;
@font-size-large: 24px;
@font-size-medium: 16px;

.container {
    background-color: @color-white;
    color: @color-dark;
}

.container h1 {
    font-size: @font-size-large;
}

.container p {
    font-size: @font-size-medium;
}

1 @color-white: #ffffff;
2 @color-dark: #333333;
3 @font-size-large: 24px;
4 @font-size-medium: 16px;
5
6 .container {
7     background-color: @color-white;
8     color: @color-dark;
9 }
10
11 .container h1 {
12     font-size: @font-size-large;
13 }
14
15 .container p {
16     font-size: @font-size-medium;
17 }
```

In this solution, we use preprocessor variables (represented here with @) to store commonly used values such as colors and font sizes. By using variables, we can easily update these values throughout the

CSS code, making it more maintainable. Additionally, this approach improves readability and reduces redundancy.

11.2 CSS Performance Optimization Techniques

Optimizing the performance of your CSS3 code is crucial for ensuring fast and efficient rendering of web pages.

Exercises

Exercise 1

Optimize the following CSS code by minifying and concatenating it:

```
/* File 1: normalize.css */
body {
    margin: 0;
    padding: 0;
}
/* File 2: main.css */
.container {
    width: 100%;
    max-width: 1200px;
    margin: 0 auto;
}
.heading {
    font-size: 24px;
    color: #333333;
}
```

```
1  /* File 1: normalize.css */
2  body {
3      margin: 0;
4      padding: 0;
5  }
6
7  /* File 2: main.css */
8  .container {
9      width: 100%;
10     max-width: 1200px;
11     margin: 0 auto;
12 }
13
14 .heading {
15     font-size: 24px;
16     color: #333333;
17 }
```

Solution

```
/* File 1: normalize.css */
body{margin:0;padding:0}
/* File 2: main.css */
.container{width:100%;max-width:1200px;margin:0
auto}.heading{font-size:24px;color:#333333}
```

In this solution, we have removed unnecessary whitespace and line breaks to minify the CSS code. Additionally, we have concatenated the code from two separate files (normalize.css and main.css) into a single file to reduce the number of HTTP requests.

Exercise 2

Refactor the following CSS code to reduce the number of selectors and improve rendering performance:

```
.container {  
    background-color: #ffffff;  
    color: #333333;  
}  
.container div {  
    font-size: 16px;  
}  
.container p {  
    font-size: 16px;  
}  
.container a {  
    color: #0066cc;  
    text-decoration: none;  
}
```

```
1 .container {  
2     background-color: #ffffff;  
3     color: #333333;  
4 }  
5  
6 .container div {  
7     font-size: 16px;  
8 }  
9  
10 .container p {  
11     font-size: 16px;  
12 }  
13  
14 .container a {  
15     color: #0066cc;  
16     text-decoration: none;  
17 }
```

Solution

```
.container {  
    background-color: #ffffff;  
    color: #333333;  
}  
.container div,  
.container p {  
    font-size: 16px;  
}  
.container a {  
    color: #0066cc;  
    text-decoration: none;  
}
```

```
1 .container {  
2     background-color: #ffffff;  
3     color: #333333;  
4 }  
5  
6 .container div,  
7 .container p {  
8     font-size: 16px;  
9 }  
10  
11 .container a {  
12     color: #0066cc;  
13     text-decoration: none;  
14 }
```

In this solution, we have combined the selectors for div and p to target them within the .container class. By doing so, we reduce the number of selectors, which improves rendering performance.

Exercise 3

Optimize the following CSS animation code for smoother performance:

```
@keyframes slideIn {  
    from {  
        opacity: 0;  
        transform: translateX(-100%);  
    }  
    to {  
        opacity: 1;  
        transform: translateX(0);  
    }  
}  
.element {  
    animation: slideIn 1s ease-in-out infinite;  
}
```

```
1 @keyframes slideIn {  
2     from {  
3         opacity: 0;  
4         transform: translateX(-100%);  
5     }  
6     to {  
7         opacity: 1;  
8         transform: translateX(0);  
9     }  
10 }  
11  
12 .element {  
13     animation: slideIn 1s ease-in-out infinite;  
14 }
```

Solution

```
@keyframes slideIn {  
    from {  
        opacity: 0;  
        transform: translateX(-100%);  
    }  
    to {  
        opacity: 1;  
        transform: translateX(0);  
    }  
}  
.element {  
    animation: slideIn 1s ease-in-out infinite;  
    will-change: transform, opacity;  
}
```

```
1 @keyframes slideIn {  
2     from {  
3         opacity: 0;  
4         transform: translateX(-100%);  
5     }  
6     to {  
7         opacity: 1;  
8         transform: translateX(0);  
9     }  
10 }  
11  
12 .element {  
13     animation: slideIn 1s ease-in-out infinite;  
14     will-change: transform, opacity;  
15 }
```

In this solution, we have added the `will-change` property to the animated element. This property informs the browser in advance about the specific CSS properties that are going to be animated, allowing it to optimize the rendering process. By specifying `will-change: transform, opacity`, we indicate that the transform and opacity

properties of the .element will be modified, enabling smoother performance.

Exams

Exam 1

Multiple-choice questions:

1. Which of the following CSS3 properties is used to create rounded corners?
 - a) border-radius
 - b) box-shadow
 - c) text-shadow
 - d) transition
2. Which CSS3 property is used to add a gradient background to an element?
 - a) background-color
 - b) background-image
 - c) background-repeat
 - d) background-gradient
3. Which CSS3 property is used to apply multiple styles to an element when hovering over it?
 - a) :hover
 - b) :active
 - c) :focus
 - d) :target
4. Which CSS3 property is used to create flexible and responsive layouts?
 - a) position
 - b) display
 - c) float
 - d) flexbox

5. Which CSS3 property is used to apply 3D transformations to an element?
 - a) transform
 - b) transition
 - c) animation
 - d) perspective
6. Which CSS3 property is used to adjust the transparency of an element?
 - a) opacity
 - b) visibility
 - c) background-color
 - d) filter
7. Which CSS3 property is used to control the order of stacked elements?
 - a) z-index
 - b) position
 - c) float
 - d) display
8. Which CSS3 property is used to create a drop shadow effect?
 - a) text-shadow
 - b) box-shadow
 - c) border-shadow
 - d) inset-shadow
9. Which CSS3 property is used to apply animations to elements?
 - a) animation
 - b) transition
 - c) transform

d) keyframes

10. Which CSS3 property is used to control the spacing between lines of text?
- a) text-align**
 - b) text-transform**
 - c) line-height**
 - d) text-decoration**

Coding:

Create a CSS3 stylesheet that styles a pricing table with three different pricing options. The table should have the following requirements:

1. Each pricing option should be displayed as a separate column.
2. Each column should have a background color.
3. The pricing option title should be displayed in a larger font size and bold.
4. The pricing amount should be displayed in a larger font size and with a different text color.
5. The features of each pricing option should be displayed as an unordered list.
6. The pricing option with the best value should have a special style applied to it, such as a border or a different background color.

Write the CSS code to style the pricing table based on the requirements mentioned above. You can assume that the HTML structure for the pricing table has already been created.

```
<div class="pricing-table">  
  <div class="pricing-option">  
    <h2 class="pricing-title">Basic</h2>
```

```
<p class="pricing-amount">$9.99</p>
<ul class="pricing-features">
    <li>Feature 1</li>
    <li>Feature 2</li>
    <li>Feature 3</li>
</ul>
</div>
<div class="pricing-option">
    <h2 class="pricing-title">Standard</h2>
    <p class="pricing-amount">$19.99</p>
    <ul class="pricing-features">
        <li>Feature 1</li>
        <li>Feature 2</li>
        <li>Feature 3</li>
    </ul>
</div>
<div class="pricing-option">
    <h2 class="pricing-title">Premium</h2>
    <p class="pricing-amount">$29.99</p>
    <ul class="pricing-features">
        <li>Feature 1</li>
        <li>Feature 2</li>
        <li>Feature 3</li>
    </ul>
</div>
</div>
```

```
1 <div class="pricing-table">
2   <div class="pricing-option">
3     <h2 class="pricing-title">Basic</h2>
4     <p class="pricing-amount">$9.99</p>
5     <ul class="pricing-features">
6       <li>Feature 1</li>
7       <li>Feature 2</li>
8       <li>Feature 3</li>
9     </ul>
10   </div>
11   <div class="pricing-option">
12     <h2 class="pricing-title">Standard</h2>
13     <p class="pricing-amount">$19.99</p>
14     <ul class="pricing-features">
15       <li>Feature 1</li>
16       <li>Feature 2</li>
17       <li>Feature 3</li>
18     </ul>
19   </div>
20   <div class="pricing-option">
21     <h2 class="pricing-title">Premium</h2>
22     <p class="pricing-amount">$29.99</p>
23     <ul class="pricing-features">
24       <li>Feature 1</li>
25       <li>Feature 2</li>
26       <li>Feature 3</li>
27     </ul>
28   </div>
29 </div>
```

Exam 2

Multiple-choice questions:

1. Which CSS3 property is used to create a responsive layout for mobile devices?
 - a) media-query
 - b) flexbox
 - c) grid-layout
 - d) position
2. Which CSS3 property is used to apply a 3D rotation effect to an element?
 - a) transform
 - b) transition
 - c) animation
 - d) perspective
3. Which CSS3 property is used to add a shadow effect to text?
 - a) text-shadow
 - b) box-shadow
 - c) border-shadow
 - d) inset-shadow
4. Which CSS3 property is used to create a responsive typography that adjusts based on the device's viewport size?
 - a) font-size
 - b) text-size-adjust
 - c) rem
 - d) vw

5. Which CSS3 property is used to create a sticky navigation bar that remains fixed at the top of the page?
 - a) position
 - b) display
 - c) float
 - d) sticky
6. Which CSS3 property is used to control the order of stacked elements in a flex container?
 - a) order
 - b) flex-direction
 - c) align-items
 - d) justify-content
7. Which CSS3 property is used to apply a blur effect to an element?
 - a) blur
 - b) filter
 - c) opacity
 - d) visibility
8. Which CSS3 property is used to create a gradient background for an element?
 - a) background-color
 - b) background-image
 - c) background-repeat
 - d) background-gradient
9. Which CSS3 property is used to create a responsive image that adjusts its size based on the available space?
 - a) width
 - b) max-width
 - c) min-width

d) height

10. Which CSS3 property is used to add rounded corners to an element's border?
- a) border-radius**
 - b) border-style
 - c) border-width
 - d) border-color

Coding:

Create a CSS3 stylesheet that styles a login form with the following requirements:

1. The form should have a centered layout on the page.
2. The input fields should have a consistent width and height.
3. The submit button should have a background color and change color when hovered over.
4. Add a CSS transition effect to the submit button to smoothly change its color.
5. Apply a box shadow effect to the form container.

Write the CSS code to style the login form based on the requirements mentioned above. You can assume that the HTML structure for the login form has already been created.

```
<div class="login-form">
  <form>
    <div class="form-group">
      <label for="username">Username:</label>
      <input type="text" id="username"
name="username" required />
    </div>
    <div class="form-group">
      <label for="password">Password:</label>
```

```
<input type="password" id="password"
name="password" required />
</div>
<button type="submit">Log In</button>
</form>
</div>
```

```
1 <div class="login-form">
2   <form>
3     <div class="form-group">
4       <label for="username">Username:</label>
5       <input type="text" id="username" name="username" required />
6     </div>
7     <div class="form-group">
8       <label for="password">Password:</label>
9       <input type="password" id="password" name="password" required />
10    </div>
11    <button type="submit">Log In</button>
12  </form>
13 </div>
```

Exam 3

Multiple-choice questions:

1. Which CSS3 property is used to animate elements with keyframes?
 - a) transition
 - b) animation
 - c) transform
 - d) opacity
2. Which CSS3 property is used to change the size of an element's font?
 - a) font-style
 - b) font-size
 - c) font-weight
 - d) font-family
3. Which CSS3 property is used to control the spacing between lines of text?
 - a) text-align
 - b) line-height
 - c) text-transform
 - d) text-decoration
4. Which CSS3 property is used to create a drop shadow effect for an element?
 - a) box-shadow
 - b) text-shadow
 - c) background-image
 - d) border-radius

5. Which CSS3 property is used to control the order of elements in a flex container?
 - a) order
 - b) flex-direction
 - c) align-items
 - d) justify-content
6. Which CSS3 property is used to create a responsive layout for mobile devices?
 - a) media-query
 - b) flexbox
 - c) grid-layout
 - d) position
7. Which CSS3 property is used to change the color of text within an element?
 - a) color
 - b) background-color
 - c) border-color
 - d) text-color
8. Which CSS3 property is used to create a gradient background for an element?
 - a) background-color
 - b) background-image
 - c) background-repeat
 - d) background-gradient
9. Which CSS3 property is used to add rounded corners to an element's border?
 - a) border-radius
 - b) border-style
 - c) border-width

- d) border-color**
10. Which CSS3 property is used to control the visibility of an element?
- a) display**
 - b) visibility**
 - c) opacity**
 - d) position**

Coding:

You have been assigned to create a CSS3 animation for a progress bar. The progress bar should start with a width of 0 and gradually increase to a specified width over a given duration. Write the CSS code to create the progress bar animation based on the requirements mentioned above.

```
<div class="progress-bar"></div>
```

```
1 <div class="progress-bar"></div>
2
```

Solutions

Exam 1

Multiple-choice questions:

1. a) border-radius
2. b) background-image
3. a) :hover
4. d) flexbox
5. a) transform
6. a) opacity
7. a) z-index
8. b) box-shadow
9. a) animation
10. c) line-height

Coding:

```
/* CSS code for styling the pricing table */
.pricing-table {
    display: flex;
}
.pricing-option {
    flex: 1;
    background-color: #f2f2f2;
    padding: 20px;
    text-align: center;
}
.pricing-title {
    font-size: 24px;
    font-weight: bold;
}
```

```
.pricing-amount {
    font-size: 36px;
    color: #ff5500;
}

.pricing-features {
    list-style-type: none;
    padding: 0;
    margin-top: 20px;
}

.pricing-features li {
    margin-bottom: 10px;
}

/* Additional style for best value */

.pricing-option-best {
    border: 2px solid #ff5500;
}
```

```
1  /* CSS code for styling the pricing table */
2  .pricing-table {
3    display: flex;
4  }
5
6  .pricing-option {
7    flex: 1;
8    background-color: #f2f2f2;
9    padding: 20px;
10   text-align: center;
11 }
12
13 .pricing-title {
14   font-size: 24px;
15   font-weight: bold;
16 }
17
18 .pricing-amount {
19   font-size: 36px;
20   color: #ff5500;
21 }
22
23 .pricing-features {
24   list-style-type: none;
25   padding: 0;
26   margin-top: 20px;
27 }
28
29 .pricing-features li {
30   margin-bottom: 10px;
31 }
32
33 /* Additional style for best value */
34 .pricing-option-best {
35   border: 2px solid #ff5500;
36 }
```

In this solution, the `.pricing-table` class applies a flexbox layout to create columns for each pricing option. The `.pricing-option` class defines the styling for each option, including a background color, padding, and center alignment. The `.pricing-title` and `.pricing-amount` classes specify the font sizes and styles for the title and amount. The `.pricing-features` class removes the default list styles and adds some top margin for the features list. The `.pricing-features li` style sets the margin at the bottom of each feature. Finally, the `.pricing-option-best` class adds an additional style, such as a border, to the pricing option that represents the best value. You can apply this CSS code to your HTML page containing the pricing table to achieve the desired styling. Feel free to customize the

styles further to match your specific design requirements.

Exam 2

Multiple-choice questions:

1. a) media-query
2. a) transform
3. a) text-shadow
4. d) vw
5. d) sticky
6. a) order
7. b) filter
8. b) background-image
9. b) max-width
10. a) border-radius

Coding:

```
/* CSS code for styling the login form */  
.login-form {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    height: 100vh;  
}  
  
form {  
    width: 300px;  
    padding: 20px;  
    background-color: #f2f2f2;  
    border-radius: 5px;  
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);  
}
```

```
.form-group {
    margin-bottom: 20px;
}

label {
    display: block;
    font-weight: bold;
}

input[type="text"],
input[type="password"] {
    width: 100%;
    height: 30px;
    padding: 5px;
    border-radius: 3px;
    border: 1px solid #ccc;
}

button[type="submit"] {
    width: 100%;
    height: 30px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 3px;
    transition: background-color 0.3s ease;
}

button[type="submit"]:hover {
    background-color: #0056b3;
}
```

```
1  /* CSS code for styling the login form */
2  .login-form {
3    display: flex;
4    justify-content: center;
5    align-items: center;
6    height: 100vh;
7  }
8
9  form {
10   width: 300px;
11   padding: 20px;
12   background-color: #f2f2f2;
13   border-radius: 5px;
14   box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
15 }
16
17 .form-group {
18   margin-bottom: 20px;
19 }
20
21 label {
22   display: block;
23   font-weight: bold;
24 }
25
26 input[type="text"],
27 input[type="password"] {
28   width: 100%;
29   height: 30px;
30   padding: 5px;
31   border-radius: 3px;
32   border: 1px solid #ccc;
33 }
34
35 button[type="submit"] {
36   width: 100%;
37   height: 30px;
38   background-color: #007bff;
39   color: #fff;
40   border: none;
41   border-radius: 3px;
42   transition: background-color 0.3s ease;
43 }
44
45 button[type="submit"]:hover {
46   background-color: #0056b3;
47 }
```

Exam 3

Multiple-choice questions:

1. b) animation
2. b) font-size
3. b) line-height
4. a) box-shadow
5. a) order
6. a) media-query
7. a) color
8. b) background-image
9. a) border-radius
10. b) visibility

Coding:

```
<style>

/* Your CSS code here */

.progress-bar {
    width: 0;
    height: 5px;
    background-color: #007bff;
    animation: progress 5s linear forwards;
}

@keyframes progress {
    0% {
        width: 0;
    }
    100% {
        width: 100%;
    }
}
```

```
        }
    }
</style>
<div class="progress-bar"></div>
1 <style>
2 /* Your CSS code here */
3 .progress-bar {
4     width: 0;
5     height: 5px;
6     background-color: #007bff;
7     animation: progress 5s linear forwards;
8 }
9
10 @keyframes progress {
11     0% {
12         width: 0;
13     }
14     100% {
15         width: 100%;
16     }
17 }
18 </style>
19
20 <div class="progress-bar"></div>
```

In the example above, the CSS code targets the progress bar element using the `.progress-bar` class selector. It sets the initial width to 0, height to 20 pixels, and background color to `#007bff`.

The `animation` property is used to create the progress bar animation. The progress animation is defined with a duration of 5 seconds, a linear timing function, and a forwards fill mode. This animation gradually increases the width of the progress bar from 0 to 100% over the specified duration.

The `@keyframes` rule defines the keyframes for the progress animation. At 0%, the progress bar is initially at a width of 0. At 100%, the progress bar reaches its final width of 100%. This creates the effect of the progress bar advancing from 0 to the specified width. You can use this CSS code in your HTML document to create the progress bar animation as described in the question. Feel free to

modify the styles, animation duration, and colors as needed to match your specific design requirements.

Conclusion

Congratulations on completing "CSS3 Workshop: Exercises and Exams for Students"! Throughout this book, you have embarked on a journey to enhance your skills and understanding of CSS3, the powerful styling language that brings life and creativity to web pages. In the pages of this book, you have explored a wide range of exercises and exams designed to challenge and reinforce your knowledge of CSS3 concepts and techniques. From selectors and box model to flexbox and animations, you have delved into the fundamental building blocks of CSS3 and expanded your proficiency in crafting beautiful, responsive, and interactive web designs.

By engaging with the exercises and exams presented in this book, you have not only gained practical experience but also fostered critical thinking and problem-solving skills. Each challenge you have tackled has helped solidify your understanding of CSS3 principles and encouraged you to think creatively when approaching real-world design scenarios.

Remember that learning is a continuous journey, and CSS3 is a dynamic and evolving technology. As you progress in your web development career, stay curious and explore further, keeping up with the latest updates and trends in CSS3. Practice, experiment, and apply your skills to real projects, pushing the boundaries of what you can achieve with CSS3.

We hope that "CSS3 Workshop: Exercises and Exams for Students" has provided you with a solid foundation in CSS3 and inspired you to continue honing your skills in this exciting field. Whether you are a

student, a professional, or an aspiring web developer, the knowledge and experience you have gained from this book will undoubtedly serve you well as you embark on your future endeavors.

Thank you for joining us on this educational journey. We wish you the very best in your ongoing exploration of CSS3 and the world of web design!

Happy styling!

Abdelfattah Ragab

Author, "CSS3 Workshop: Exercises and Exams for Students"

About the Author

Abdelfattah Ragab is an accomplished software developer and passionate educator specializing in web development and design. With years of industry experience, Abdelfattah has honed his skills in creating visually stunning and user-friendly web experiences. Throughout his career, Abdelfattah has worked on a wide range of projects, from small business websites to large-scale enterprise applications. His expertise lies in front-end development, where he leverages the power of CSS3 to bring designs to life and craft seamless user interfaces. Abdelfattah's deep understanding of CSS3, combined with his dedication to staying updated with the latest trends and techniques, has made him a sought-after expert in the field. He possesses comprehensive knowledge of CSS3's advanced features, including transitions, transformations, animations, and responsive design principles. As a passionate advocate for clean code and best practices, Abdelfattah emphasizes the importance of writing maintainable and scalable CSS code. His attention to detail and commitment to delivering high-quality solutions have earned him a reputation for excellence among his peers and clients. In addition to his professional work, Abdelfattah is a dedicated educator who enjoys sharing his knowledge and helping others succeed. He regularly conducts workshops and training sessions on CSS3 and web development, empowering aspiring developers to enhance their skills and achieve their goals.

"CSS3 Workshop: Exercises and Exams for Students" is the culmination of Abdelfattah's expertise and experience in the field. In this book, he provides readers with practical exercises and exams designed to enhance their understanding of CSS3 concepts and techniques. By combining theoretical knowledge with hands-on practice, Abdelfattah equips students with the tools and confidence to create visually appealing and responsive websites.

Abdelfattah's passion for web development, coupled with his commitment to helping others master CSS3, makes him an invaluable resource for anyone seeking to elevate their web styling skills.

Connect with Abdelfattah Ragab:

Website: <https://abdelfattah-ragab.com>

Copyright © Abdelfattah Ragab

ISBN: 9798890084231