Applications Development and Emerging Technologies [1]

A Study Guide for Students of Sorsogon State University - Bulan Campus[2]

Jarrian Vince G. Gojar[3]

February 23, 2025

[1]A course in the Bachelor of Science in Computer Science.
[2]This book is a study guide for students of Sorsogon State University - Bulan Campus taking up the course Applications Development and Emerging Technologies.
[3]https://github.com/godkingjay

Sorsogon State University - Bulan Campus

Contents

List of Figures

List of Tables

List of Codes

Preface

–

Jarrian Vince G. Gojar
https://github.com/godkingjay

1

Introduction to Applications Development and Emerging Technologies

1.1 Introduction

1.1.1 Application

An **application** is a software program that allows users to perform specific tasks. Applications for desktop or laptop computers are sometimes called **desktop applications**, while those for mobile devices are called **mobile apps**. When you open an application, it runs inside the operating system until you close it. Most of the time, you will have more than one application open at the same time, which is known as **multitasking**.

1.1.2 Development

Development is the process of creating a software application. It includes designing the user interface, writing code, and testing the application for bugs. The goal of software development is to create a program that is easy to use and works correctly.

1.1.3 Application Development

Application development is the process of planning, designing, creating, testing, and deploying an application to perform various business operations. It can be done by massive organizations with large teams working on projects or by a single freelance developer.

1.2 Different Types of Application Development

There are several different types of application development, including:

- Web Development
- Mobile Application Development
- Desktop Application Development
- Game Development
- Cloud Development

1.2.1 Web Development

Web development is the process of creating websites and web applications. It involves designing the user interface, writing code, and testing the website for bugs. Web development can be divided into two categories: front-end development and back-end development.

1.2.1.1 Front-End Development

Front-end development is the process of creating the user interface of a website. It involves designing the layout, colors, and fonts of the website. Front-end developers use HTML, CSS, and JavaScript to create the user interface of a website.

1. **HTML (HyperText Markup Language)** is the standard markup language used to create web pages. It defines the structure of a web page using a series of elements.

2. **CSS (Cascading Style Sheets)** is a style sheet language used to define the appearance of a web page. It allows developers to control the layout, colors, and fonts of a website.

3. **JavaScript** is a programming language used to create interactive elements on a web page. It allows developers to add functionality such as animations, pop-ups, and form validation to a website.

4. **Bootstrap** is a front-end framework that allows developers to create responsive and mobile-first websites. It provides a set of pre-designed components, such as buttons, forms, and navigation bars, that can be easily customized.

5. **React** is a JavaScript library used to create user interfaces for single-page applications. It allows developers to build reusable components that update automatically when the data changes.

6. **Angular** is a front-end framework that allows developers to create dynamic web applications. It provides a set of tools and libraries for building interactive user interfaces.

7. **Vue** is a progressive JavaScript framework used to create user interfaces and single-page applications. It allows developers to build interactive web applications with ease.

The above tools and technologies are commonly used in front-end development to create responsive and interactive websites. HTML, CSS, and JavaScript are the building blocks of front-end development, while frameworks such as Bootstrap, React, Angular, and Vue provide additional features for building modern web applications.

1.2.1.2 Back-End Development

Back-end development is the process of creating the server-side logic of a website. It involves writing code that interacts with the database and processes data. Back-end developers use programming languages such as PHP, Python, and Ruby to create the server-side logic of a website.

1. **Node.js** is a JavaScript runtime environment that allows developers to run JavaScript on the server-side. It provides a set of libraries and tools for building scalable and high-performance web applications.

2. **Express** is a web application framework for Node.js. It provides a set of features for building web applications, such as routing, middleware, and templating.

3. **Django** is a high-level web framework for Python. It allows developers to build web applications quickly and efficiently. Django provides a set of tools and libraries for building secure and scalable web applications.

4. **Flask** is a lightweight web framework for Python. It allows developers to build web applications with minimal code. Flask provides a set of tools and libraries for building simple and scalable web applications.

5. **Ruby on Rails** is a web application framework for Ruby. It provides a set of tools and libraries for building web applications quickly and efficiently. Ruby on Rails follows the convention over configuration principle, which allows developers to write less code and focus on building the application.

6. **Laravel** is a web application framework for PHP. It provides a set of tools and libraries for building web applications quickly and efficiently. Laravel follows the model-view-controller (MVC) architecture, which allows developers to separate the business logic from the presentation layer.

7. **Spring** is a web application framework for Java. It provides a set of tools and libraries for building enterprise-level web applications. Spring follows the inversion of control (IoC) principle, which allows developers to write loosely coupled code and focus on building the application.

The above tools and technologies are commonly used in back-end development to create the server-side logic of a website. Back-end developers use these tools to interact with the database, process data, and handle user requests on the server-side.

1.2.2 Mobile Application Development

Mobile application development is the process of creating mobile applications for smartphones and tablets. It involves designing the user interface, writing code, and testing the mobile application for bugs. Mobile development can be divided into two categories: iOS development and Android development.

1.2.2.1 iOS Development

iOS development is the process of creating mobile applications for Apple devices, such as iPhones and iPads. It involves designing the user interface using Xcode and writing code in Swift or Objective-C. iOS developers use Xcode, Swift, and Objective-C to create mobile applications for Apple devices.

1. **Xcode** is an integrated development environment (IDE) used to create iOS applications. It provides a set of tools and libraries for building mobile applications for Apple devices.

2. **Swift** is a programming language used to create iOS applications. It provides a set of features for building mobile applications, such as type safety, optionals, and generics.

3. **Objective-C** is a programming language used to create iOS applications. It provides a set of features for building mobile applications, such as dynamic typing, message passing, and memory management.

4. **React Native** is a JavaScript framework used to create mobile applications for Android and iOS devices. It allows developers to build cross-platform mobile applications with a single codebase.

5. **Flutter** is a mobile UI framework used to create mobile applications for Android and iOS devices. It allows developers to build cross-platform mobile applications with a single codebase.

The above tools and technologies are commonly used in iOS development to create mobile applications for Apple devices. iOS developers use these tools to design the user interface and write code for mobile applications.

1.2.2.2 Android Development

Android development is the process of creating mobile applications for Android devices. It involves designing the user interface using Android Studio and writing code in Java or Kotlin. Android developers use Android Studio, Java, and Kotlin to create mobile applications for Android devices.

1. **Android Studio** is an integrated development environment (IDE) used to create Android applications. It provides a set of tools and libraries for building mobile applications for Android devices.

2. **Java** is a programming language used to create Android applications. It provides a set of features for building mobile applications, such as object-oriented programming, inheritance, and polymorphism.

3. **Kotlin** is a programming language used to create Android applications. It provides a set of features for building mobile applications, such as null safety, extension functions, and coroutines.

4. **React Native** is a JavaScript framework used to create mobile applications for Android and iOS devices. It allows developers to build cross-platform mobile applications with a single codebase.

5. **Flutter** is a mobile UI framework used to create mobile applications for Android and iOS devices. It allows developers to build cross-platform mobile applications with a single codebase.

The above tools and technologies are commonly used in Android development to create mobile applications for Android devices. Some of the tools here are also used in iOS development to create mobile applications for Apple devices. React Native and Flutter in particular are used to build cross-platform mobile applications for both Android and iOS devices.

1.2.3 Desktop Application Development

Desktop application development is the process of creating desktop applications for Windows, macOS, and Linux. It involves designing the user interface, writing code, and testing the desktop application for bugs.

1. **Electron** is a framework used to create desktop applications with web technologies. It allows developers to build cross-platform desktop applications with HTML, CSS, and JavaScript.

2. **JavaFX** is a framework used to create desktop applications with Java. It provides a set of tools and libraries for building cross-platform desktop applications with Java.

3. **Qt** is a framework used to create desktop applications with C++. It provides a set of tools and libraries for building cross-platform desktop applications with C++.

4. **WinForms** is a framework used to create desktop applications with C#. It provides a set of tools and libraries for building desktop applications for Windows.

5. **WPF** is a framework used to create desktop applications with C#. It provides a set of tools and libraries for building desktop applications for Windows.

The above tools and technologies are commonly used in desktop development to create desktop applications for Windows, macOS, and Linux. For Windows, developers use WinForms and WPF to create desktop applications with C#. For cross-platform desktop applications, developers use Electron, JavaFX, and Qt to build desktop applications with web technologies, Java, and C++.

1.2.4 Game Development

Game development is the process of creating video games for consoles, computers, and mobile devices. It involves designing the gameplay, writing code, and testing the game for bugs.

1. **Unity** is a game engine used to create 2D and 3D games for consoles, computers, and mobile devices. It provides a set of tools and libraries for building cross-platform games with C#.

2. **Unreal Engine** is a game engine used to create 2D and 3D games for consoles, computers, and mobile devices. It provides a set of tools and libraries for building cross-platform games with C++.

3. **Godot** is a game engine used to create 2D and 3D games for consoles, computers, and mobile devices. It provides a set of tools and libraries for building cross-platform games with GDScript.

4. **GameMaker Studio** is a game engine used to create 2D games for consoles, computers, and mobile devices. It provides a set of tools and libraries for building cross-platform games with GML.

5. **Construct** is a game engine used to create 2D games for consoles, computers, and mobile devices. It provides a set of tools and libraries for building cross-platform games with events.

The above tools and technologies are commonly used in game development to create video games for consoles, computers, and mobile devices. Unity, Unreal Engine, Godot, GameMaker Studio, and Construct are popular game engines used by game developers to create 2D and 3D games. These game engines provide a set of tools and libraries for building cross-platform games with C#, C++, GDScript, and GML.

1.2.5 Cloud Development

Cloud development is the process of creating cloud-based applications that run on remote servers. It involves designing the user interface, writing code, and testing the cloud application for bugs.

1. **Amazon Web Services (AWS)** is a cloud platform used to create cloud-based applications. It was one of the first cloud platforms and is widely used by developers to build scalable and secure cloud applications. It is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals,

2. **Microsoft Azure**, similarly to AWS, is a cloud platform used to create cloud-based applications. Microsoft Azure is a cloud computing service created by Microsoft for

building, testing, deploying, and managing applications and services through Microsoft-managed data centers.

3. **Google Cloud Platform (GCP)**, similarly to AWS and Microsoft Azure, is a cloud platform used to create cloud-based applications. Google Cloud Platform is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search, Gmail, file storage, and YouTube.

4. **Heroku** is a cloud platform used to create cloud-based applications. It provides a set of tools and services for building scalable and secure cloud applications. Heroku is a cloud platform as a service supporting several programming languages.

5. **Firebase**, also developed by Google, is a cloud platform used to create cloud-based applications. Firebase is a platform developed by Google for creating mobile and web applications. It was originally an independent company founded in 2011. In 2014, Google acquired the platform and it is now their flagship offering for app development.

The above tools and technologies are commonly used in cloud development to create cloud-based applications that run on remote servers. AWS, Microsoft Azure, GCP, Heroku, and Firebase are popular cloud platforms used by developers to build scalable and secure cloud applications. These cloud platforms provide a set of tools and services for building cloud-based applications with ease.

2

Web Development

2.1 Introduction

There are around 3.58 billion internet users on the planet. This implies that over half of the world's 7.6 billion people have access to the internet, which they use for everything from entertainment to education, communication to commerce, keeping up with current events, and keeping up with business experts. Indeed, for most people, the internet is the first (and often only) channel through which we communicate with the world in all of its complexities.

There are three interactive elements on the internet:

1. **Websites** - A collection of web pages that are linked together and share a common domain name.
2. **Servers** - A computer or computer program that manages access to a centralized resource or service in a network.
3. **Browsers** - A software application used to access and view websites on the internet.

The frontend (client side) and the backend (server side) are two parts of any website. The frontend comprises everything the user sees and experiences instantly while visiting a website. The backend is behind the scenes that store, send and receive information.

HTML, CSS, and Javascript files make up everything a user sees on a website. As a web developer, these are the most basic tools needed. They are the languages that required to build websites.

2.2 HTML

HTML (HyperText Markup Language) is the standard markup language used to create web pages. It defines the structure of a web page using a series of elements. It contains the essential elements of a website, such as words, titles, and paragraphs, as well as links, images, and other media. HTML elements are represented by tags, which are enclosed in angle brackets. HTML forms the backbone of any webpage, dictating its structure and content.

```html
<!DOCTYPE html>
<html lang="en">

```

```html
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>First Web Page</title>
</head>

<body>
    <h1>Hello, World!</h1>
    <p>Welcome to my website.</p>
</body>

</html>
```

Code 2.1: HTML Example

Code 2.1 shows an example of an HTML document. An HTML boilerplate usually looks like this:

- **<!DOCTYPE html>** - Defines the document type and version of HTML. In this case, it is HTML5.
- **<html>** - Defines the root element of an HTML page.
- **<head>** - Contains meta-information about the document, such as the title, character set, and viewport.
- **<body>** - Contains the content of the document, such as headings, paragraphs, and images.

2.2.1 HTML Tags

HTML tags are used to define the structure and content of a web page. They are enclosed in angle brackets and come in pairs: an opening tag and a closing tag. The opening tag is used to define the beginning of an element, while the closing tag is used to define the end of an element.

When an HTML tag is opened, it must be closed to avoid errors. Some tags are self-closing, meaning they do not require a closing tag. HTML tags can also have attributes, which provide additional information about the element.

```html
<p>Welcome to my website.</p>
```

Code 2.2: HTML Open and Close Tag

Code 2.2 shows an example of an HTML tag with an opening tag (**<p>**) and a closing tag (**</p>**). The content of the paragraph is "Welcome to my website.".

```html
<img src="https://avatar.iran.liara.run/public" alt="Image" />
```

Code 2.3: HTML Self-Closing Tag

Code 2.3 shows an example of an HTML tag that is self-closing (**<img />**). This tag is used to insert an image into the document. The **src** attribute specifies the URL of the image, while the **alt** attribute provides a text description of the image.

2.2.1.1 <html>...</html>

This tag specifies that the webpage is written in HTML. It appears at the very first and last line of the webpage. It is mainly used to show that the page uses HTML5 – the latest version of the language. Also known as the root element, this tag can be thought of as a parent tag for every other tag used in the page.

```html
<!DOCTYPE html>
<html lang="en">
   <!-- Content goes here -->
</html>
```

Code 2.4: HTML <html> Tag

Code 2.4 shows an example of the <html> tag. Here, the **lang** attribute specifies the language of the document, which is English in this case.

2.2.1.2 <head>...</head>

This tag is used to define the head section of the webpage. The head section contains meta-information about the document, such as the title, character set, and viewport. It is not displayed on the webpage but is used to provide information about the document to the browser and search engines.

```html
<head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>First Web Page</title>
</head>
```

Code 2.5: HTML <head> Tag

Code 2.5 shows an example of the <head> tag. Here, the **<meta>** tag is used to define the character set and viewport of the document, while the **<title>** tag is used to define the title of the document. This title appears in the browser tab.

2.2.1.3 <title>...</title>

This tag is used to define the title of the document. It appears in the browser tab and is used to identify the webpage.

```html
<title>First Web Page</title>
```

Code 2.6: HTML <title> Tag

Code 2.6 shows an example of the <title> tag. Here, the title of the document is "First Web Page". This title appears in the browser tab when the document is opened.

2.2.1.4 <body>...</body>

This tag is used to define the body section of the webpage. The body section contains the content of the document, such as headings, paragraphs, and images. It is displayed on the webpage and is visible to the user.

```html
<body>
    <h1>Hello, World!</h1>
    <p>Welcome to my website.</p>
</body>
```

Code 2.7: HTML <body> Tag

Code 2.7 shows an example of the <body> tag. Here, the **<h1>** tag is used to define a heading, while the **<p>** tag is used to define a paragraph. This content is displayed on the webpage and is visible to the user.

2.2.1.5 <h1>...</h1> to <h6>...</h6>

These tags are used to define headings of different sizes. The **<h1>** tag defines the largest heading, while the **<h6>** tag defines the smallest heading. Headings are used to define the structure of the document and provide a hierarchy of information.

```html
<h1>This is a Heading 1</h1>
<h2>This is a Heading 2</h2>
<h3>This is a Heading 3</h3>
<h4>This is a Heading 4</h4>
<h5>This is a Heading 5</h5>
<h6>This is a Heading 6</h6>
```

Code 2.8: HTML <h1> to <h6> Tags

Code 2.8 shows an example of the **<h1>** to **<h6>** tags. These tags are used to define headings of different sizes, with **<h1>** being the largest and **<h6>** being the smallest.

2.2.1.6 <p>...</p>

This tag is used to define a paragraph of text. It is used to group text content together and provide structure to the document.

```html
<p>
  Excepteur officia tempor do laborum commodo cupidatat ea Lorem qui irure enim velit. Adipisicing dolor minim Lorem nulla dolor quis et aliqua. Officia anim adipisicing excepteur sint elit qui laboris reprehenderit non elit. Voluptate voluptate duis aliqua proident elit exercitation cillum anim reprehenderit nostrud minim culpa veniam.
</p>
```

Code 2.9: HTML <p> Tag

Code 2.9 shows an example of the **<p>** tag. This tag is used to define a paragraph of text, which is displayed on the webpage.

2.2.1.7 <div>...</div>

This tag is used to define a division or section of the document. It is a block-level element that can contain other block-level or inline elements.

```html
<div>
    <h1>Hello, World!</h1>
    <p>Welcome to my website.</p>
</div>

<div>
    <img src="https://avatar.iran.liara.run/public/boy" alt="Image" />
    <img src="https://avatar.iran.liara.run/public/girl" alt="Image" />
</div>
```

Code 2.10: HTML <div> Tag

Code 2.10 shows an example of the **<div>** tag. This tag is used to define a division or section of the document, which can contain other elements such as headings, paragraphs, and images.

2.2.1.8 <section>...</section>

This tag is used to define a section of the document. It is usually used to group related content together such as articles, blog posts, or product listings.

```html
<section>
    <h2>Section 1</h2>
    <p>Content for section 1 goes here.</p>
</section>

<section>
    <h2>Section 2</h2>
    <p>Content for section 2 goes here.</p>
</section>

<section>
    <h2>Section 3</h2>
    <p>Content for section 3 goes here.</p>
</section>
```

Code 2.11: HTML <section> Tag

Code 2.11 shows an example of the **<section>** tag. This tag is used to define a section of the document, which can contain related content such as headings and paragraphs.

2.2.1.9 <span>...</span>

This tag is used to define a span of text. It is an inline element that can contain other inline elements. It is usually used to apply styles to a specific section of text.

```html
<p>Welcome to my <span style="color: blue;">website</span>.</p>
```

Code 2.12: HTML <span> Tag

Code 2.12 shows an example of the **<span>** tag. This tag is used to define a span of text, which can be styled separately from the rest of the paragraph. In this example, the text "website" is styled with a blue color.

2.2.1.10 <br />

This tag is used to insert a line break in the document. It is a self-closing tag that does not require a closing tag. When used, it moves the content to the next line. In texts, it is used to separate paragraphs or lines.

```html
<p>Welcome to my <br /> website.</p>
```

Code 2.13: HTML <br /> Tag

Code 2.13 shows an example of the **<br />** tag. This tag is used to insert a line break in the document, which moves the content to the next line.

2.2.1.11 <hr />

This tag is used to insert a horizontal rule in the document. It is a self-closing tag that creates a horizontal line across the page. It can be used to separate sections of content or to create a visual break in the document.

```html
<p>Welcome to my website.</p>
<hr />
<p>Thank you for visiting.</p>
```

Code 2.14: HTML <hr /> Tag

Code 2.14 shows an example of the **<hr />** tag. This tag is used to insert a horizontal rule in the document, which creates a horizontal line across the page.

2.2.1.12 <img />

This tag is used to insert an image in the document. It is a self-closing tag that requires the **src** attribute to specify the image file.

```html
<img src="https://avatar.iran.liara.run/public" alt="Image" />
```

Code 2.15: HTML <img /> Tag

Code 2.15 shows an example of the **<img />** tag. This tag is used to insert an image in the document, which is displayed on the webpage. The **src** attribute specifies the image file, while the **alt** attribute provides alternative text for the image.

2.2.1.13 <a>...</a>

This tag is used to create a hyperlink in the document. It requires the **href** attribute to specify the URL of the link.

```html
<a href="https://www.github.com/godkingjay" target="_blank>Visit GitHub</a>
```

Code 2.16: HTML <a> Tag

Code 2.16 shows an example of the **<a>** tag. This tag is used to create a hyperlink in the document, which links to the specified URL. The **href** attribute specifies the URL of the link. The **target** attribute specifies where to open the link. Its value can be **_blank** to open the link in a new tab or **_self** to open the link in the same tab. By default, the link opens in the same tab.

2.2.1.14 <ul>...</ul>

This tag is used to create an unordered list in the document. It contains a list of items that are displayed with bullet points. In the list, each item is defined using the **<li>** tag.

```html
<ul style="list-style-type: square;">
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>
```

Code 2.17: HTML <ul> Tag

Code 2.17 shows an example of the **<ul>** tag. This tag is used to create an unordered list in the document, which contains a list of items displayed with bullet points. The **<li>** tag is used to define each item in the list. The **style** attribute is used to specify the style of the list, such as the type of bullet point. The **list-style-type** property specifies the type of bullet point to use, such as *square*, *circle*, or *disc*.

2.2.1.15 <ol>...</ol>

This tag is used to create an ordered list in the document. It contains a list of items that are displayed with numbers or letters.

```html
<ol type="A" start="3">
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ol>
```

Code 2.18: HTML <ol> Tag

Code 2.18 shows an example of the **<ol>** tag. This tag is used to create an ordered list in the document, which contains a list of items displayed with numbers or letters. The **<li>** tag is used to define each item in the list. The **type** attribute is used to specify the type of numbering to use, such as *1*, *A*, *a*, *I*, or *i*. The default type is *1*. The **start** attribute is used to specify the starting number of the list. The default start number is *1*.

2.2.1.16 <li>...</li>

This tag is used to define an item in a list. It is used inside the **<ul>** or **<ol>** tag to define each item in the list.

```
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>

<ol>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ol>
```

Code 2.19: HTML <li> Tag

Code 2.19 shows an example of the **<li>** tag. This tag is used to define an item in a list, which is displayed as part of an unordered or ordered list.

2.2.1.17 <strong>...</strong>

This tag is used to define text that should be displayed in a strong or bold font. It is used to emphasize important text content.

```
<p>Welcome to my <strong>website</strong>.</p>
```

Code 2.20: HTML <strong> Tag

Code 2.20 shows an example of the **<strong>** tag. This tag is used to define text that should be displayed in a strong or bold font, which emphasizes the importance of the text content.

2.2.1.18 <b>...</b>

Similar to the **<strong>** tag, this tag is used to define text that should be displayed in a bold font. It is used to emphasize important text content.

```
<p>Welcome to my <b>website</b>.</p>
```

Code 2.21: HTML <b> Tag

Code 2.21 shows an example of the **<b>** tag. This tag is used to define text that should be displayed in a bold font, which emphasizes the importance of the text content.

2.2.1.19 <em>...</em>

This is another inline element that is used to define text that should be displayed in an emphasized or italic font. It is used to provide emphasis to text content.

```html
<p>Welcome to my <em>website</em>.</p>
```

Code 2.22: HTML <em> Tag

Code 2.22 shows an example of the **<em>** tag. This tag is used to define text that should be displayed in an emphasized or italic font, which provides emphasis to the text content.

2.2.1.20 <i>...</i>

Similar to the **<em>** tag, this tag is used to define text that should be displayed in an italic font. It is used to provide emphasis to text content.

```html
<p>Welcome to my <i>website</i>.</p>
```

Code 2.23: HTML <i> Tag

Code 2.23 shows an example of the **<i>** tag. This tag is used to define text that should be displayed in an italic font, which provides emphasis to the text content.

2.2.1.21 <table>...</table>

This tag is used to create a table in the document. It contains a set of rows and columns that display data in a structured format.

```html
<table border="1">
    <tr>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>John</td>
        <td>25</td>
    </tr>
    <tr>
        <td>Jane</td>
        <td>30</td>
    </tr>
</table>
```

Code 2.24: HTML <table> Tag

Code 2.24 shows an example of the **<table>** tag. This tag is used to create a table in the document, which contains a set of rows and columns that display data in a structured format. The **<tr>** tag is used to define a row in the table, while the **<th>** tag is used to define a header cell and the **<td>** tag is used to define a data cell. The **border** attribute is used to specify the border width of the table.

2.2.1.22 <tr>...</tr>

This tag is used to define a row in a table. It is used inside the **<table>** tag to define each row in the table.

```html
<table border="1>
    <tr>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>John</td>
        <td>25</td>
    </tr>
    <tr>
        <td>Jane</td>
        <td>30</td>
    </tr>
</table>
```

Code 2.25: HTML <tr> Tag

Code 2.25 shows an example of the **<tr>** tag. This tag is used to define a row in a table, which contains a set of cells that display data in a structured format.

2.2.1.23 <th>...</th> and <td>...</td>

These tags are used to define header cells and data cells in a table, respectively. The **<th>** tag is used to define a header cell, while the **<td>** tag is used to define a data cell.

```html
<table border="1>
    <tr>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>John</td>
        <td>25</td>
    </tr>
    <tr>
        <td>Jane</td>
        <td>30</td>
    </tr>
</table>
```

Code 2.26: HTML <th> and <td> Tags

Code 2.26 shows an example of the **<th>** and **<td>** tags. The **<th>** tag is used to define a header cell in a table, while the **<td>** tag is used to define a data cell in a table.

2.2.1.24 <form>...</form>

This tag is used to create a form in the document. It contains a set of form elements, such as input fields, buttons, and checkboxes, that allow users to submit data to a server.

```html
<form>
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" />
    <button type="submit">Submit</button>
</form>
```

Code 2.27: HTML <form> Tag

Code 2.27 shows an example of the **<form>** tag. This tag is used to create a form in the document, which contains a set of form elements that allow users to submit data to a server.

2.2.1.25 <input />

This tag is used to create an input field in a form. It is a self-closing tag that requires the **type** attribute to specify the type of input field.

```html
<form style="display: flex; flex-direction: column; gap: 8px;">
  <div>
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" />
  </div>

  <div>
    <label for="age">Age:</label>
    <input type="number" id="age" name="age" min="09" max="100" />
  </div>

  <div>
    <label for="civil-status">Civil Status:</label>
    <input type="radio" id="single" name="civil-status" value="single" />
       Single
    <input type="radio" id="married" name="civil-status" value="married" />
       Married
    <input type="radio" id="divorced" name="civil-status" value="divorced" />
       Divorced
  </div>

  <div>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" />
  </div>

  <div>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" />
  </div>

  <div>
    <label for="color">Favorite Color:</label>
    <input type="color" id="color" name="color" />
  </div>

  <div>
    <label for="date">Date of Birth:</label>
    <input type="date" id="date" name="date" />
  </div>

  <div>
    <label for="time">Time of Birth:</label>
    <input type="time" id="time" name="time" />
  </div>

  <div>
    <label for="file">Upload File:</label>
    <input type="file" id="file" name="file" />
  </div>

  <div>
    <label for="message">Message:</label>
    <textarea id="message" name="message"></textarea>
  </div>

  <div>
    <label for="agree">I agree to the terms and conditions:</label>
    <input type="checkbox" id="agree" name="agree" value="yes" />
  </div>

  <button type="submit">Submit</button>
</form>
```

Code 2.28: HTML <input /> Tag

Code 2.28 shows an example of the **<input />** tag. This tag is used to create an input field in a form, which allows users to enter data. The **type** attribute specifies the type of input field, such as text, number, email, or password.

2.2.1.26 <textarea>...</textarea>

This tag is used to create a textarea field in a form. It allows users to enter multiple lines of text.

```html
<label for="message">Message:</label>
<textarea id="message" name="message"></textarea>
```

Code 2.29: HTML <textarea> Tag

Code 2.29 shows an example of the **<textarea>** tag. This tag is used to create a textarea field in a form, which allows users to enter multiple lines of text.

2.2.1.27 <button>...</button>

This tag is used to create a button in a form. It is used to submit the form data to a server or perform an action when clicked.

```html
<button type="submit">Submit</button>
<button type="reset">Reset</button>
<button type="button">Click Me</button>
```

Code 2.30: HTML <button> Tag

Code 2.30 shows an example of the **<button>** tag. This tag is used to create a button in a form, which allows users to submit the form data to a server or perform an action when clicked. The **type** attribute specifies the type of button, such as submit, reset, or button.

2.2.1.28 <label>...</label>

This tag is used to create a label for an input field in a form. It is used to provide a description or name for the input field. Clicking on the label focuses the associated input field.

```html
<label for="name">Name:</label>
<input type="text" id="name" name="name" />
```

Code 2.31: HTML <label> Tag

Code 2.31 shows an example of the **<label>** tag. This tag is used to create a label for an input field in a form, which provides a description or name for the input field. The **for** attribute specifies the ID of the input field that the label is associated with.

2.2.1.29 <select>...</select>

This tag is used to create a dropdown list in a form. It contains a set of **<option>** tags that define the options in the dropdown list.

```html
<select id="color" name="color">
    <option value="red">Red</option>
    <option value="green">Green</option>
    <option value="blue">Blue</option>
</select>
```

Code 2.32: HTML <select> Tag

Code 2.32 shows an example of the **<select>** tag. This tag is used to create a dropdown list in a form, which contains a set of **<option>** tags that define the options in the dropdown list. The **id** attribute specifies the ID of the dropdown list, while the **name** attribute specifies the name of the dropdown list.

2.2.1.30 <iframe>...</iframe>

This tag is used to embed another document within the current document. It is used to display content from another website or source. It can also be used to embed videos, maps, or other media.

```
<iframe
  width="600"
  height="450"
  style="border:0;"
  allowfullscreen="true"
  loading="lazy"
  src="https://www.google.com/maps/embed?pb=!1m14!1m12!1m3!1d1075
  .0761255478576!2d123.88209023319297!3d12
  .66699124573315!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13
  .1!5e1!3m2!1sen!2sph!4v1737791388626!5m2!1sen!2sph"
>
</iframe>
```

Code 2.33: HTML <iframe> Tag

Code 2.33 shows an example of the **<iframe>** tag. This tag is used to embed another document within the current document, which displays content from another website or source. The **src** attribute specifies the URL of the document to be embedded, while the **width** and **height** attributes specify the dimensions of the embedded document. In this example, an embedded Google Maps is shown.

2.2.2 More about HTML

You can find more information about HTML tags from the following sources:

- MDN Web Docs HTML Elements
- W3Schools HTML Tags
- Hostinger HTML Cheatsheet

2.3 CSS

CSS (Cascading Style Sheets) is a style sheet that describes how HTML components appear on a page. CSS is used to manage your website's appearance, style, and formatting, including RGB values, border colors, background pictures, and more.

CSS files specify a set of rules for defining a set of properties and their values.

2.3.1 Core Concepts

2.3.1.1 Cascading and Specificity

2.3.1.1.1 Cascading

Cascading refers to the process of combining multiple style sheets and resolving conflicts between them. When multiple style rules apply to the same element, the browser uses a set of rules to determine which styles to apply.

1. **Specificity** - The more specific rule takes precedence.
2. **Order** - The last rule defined takes precedence.
3. **Importance** - The !important rule takes precedence.

In CSS there are three types of style sheets:

```
<p style="color: red;">This is a paragraph.</p>
```

Code 2.34: Inline CSS

1. **Inline Style** - The inline style is defined within the **style** attribute of an HTML element. It is used to define styles for a specific element.

```
<style>
  h1 {
    color: red;
  }
</style>

<h1>This is a heading.</h1>
```

Code 2.35: Internal CSS

2. **Internal Style Sheet** - The internal style sheet is defined within the **<style>** tag in the **<head>** section of the HTML document. It is used to define styles for a specific document.

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

<h1>This is a heading.</h1>
```

Code 2.36: External CSS

3. **External Style Sheet** - The external style sheet is a separate file where you can define all the styles that you want to use on your website. You can link the external style sheet to your HTML document using the **<link>** tag.

2.3.1.1.2 Specificity

Specificity is a set of rules that determines which style rules apply to an element when multiple rules conflict. Specificity is calculated based on the following factors:

1. **Inline Styles** - Inline styles have the highest specificity and override all other styles.
2. **Element Selectors** - Element selectors have the lowest specificity and are overridden by other selectors.
3. **Class Selectors** - Class selectors have a higher specificity than element selectors.
4. **ID Selectors** - ID selectors have a higher specificity than class selectors and element selectors.
5. **!important** - The !important rule overrides all other rules and has the highest specificity.
6. **Order of Appearance** - If two rules have the same specificity, the rule that appears last in the style sheet takes precedence.

2.3.1.2 Selectors

In CSS, selectors are used to target HTML elements and apply styles to them. There are different types of selectors that can be used to target elements based on their type, class, ID, or other attributes.

2.3.1.2.1 Element Selector

The element selector is used to target all elements of a specific type. It is defined by the element name without any additional characters.

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

Code 2.37: Element Selector

```css
p {
  color: red;
}
```

Code 2.38: Element Selector CSS

Code 2.37 and 2.38 show an example of the element selector. In this example, the **<p>** element is targeted using the element selector, and the color of the text is set to red.

2.3.1.2.2 Class Selector

The class selector is used to target elements with a specific class. It is defined by a period (.) followed by the class name. This selector is useful when you want to apply the same style to multiple elements. This is generally the most common selector used in CSS.

```html
<p>This is a <span class="highlight">paragraph</span>.</p>
```

Code 2.39: Class Selector

```css
.highlight-text {
  background: yellow;
}
```

Code 2.40: Class Selector CSS

Code 2.39 and 2.40 show an example of the class selector. In this example, the **<span>** element with the class **highlight** is targeted using the class selector, and the background color is set to yellow.

2.3.1.2.3 ID Selector

The ID selector is used to target a specific element with a unique ID. It is defined by a hash (#) followed by the ID name. This selector is used when you want to apply a style to a single element on the page.

```html
<p id="intro">This is an introduction.</p>
```

Code 2.41: ID Selector

```css
#intro {
  font-size: 24px;
}
```

Code 2.42: ID Selector CSS

Code 2.41 and 2.42 show an example of the ID selector. In this example, the **<p>** element with the ID **intro** is targeted using the ID selector, and the font size is set to 24 pixels.

2.3.1.2.4 Universal Selector

The universal selector is used to target all elements on the page. It is defined by an asterisk (*) character. This selector is used when you want to apply a style to all elements on the page.

```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

Code 2.43: Universal Selector CSS

Code 2.43 shows an example of the universal selector. In this example, the universal selector is used to apply a style to all elements on the page, setting the margin, padding, and box-sizing properties.

2.3.1.2.5 Pseudo-classes

Pseudo-classes are used to define a special state of an element. They are defined by a colon (:) followed by the pseudo-class name. Pseudo- classes are used to style elements based on user interaction or element state.

```html
<a href="https://www.github.com/godkingjay " target="_blank">Visit GitHub</a>
```

Code 2.44: Pseudo-class HTML

```css
a:link {
  color: blue;
}

a:hover {
  color: red;
}

a:active {
  color: green;
}

a:visited {
  color: purple;
}
```

Code 2.45: Pseudo-class CSS

Code 2.44 and 2.45 show an example of pseudo-classes. In this example, the **<a>** element is targeted using the pseudo-classes **:link**, **:hover**, **:active**, and **:visited**, which define the styles for the link in different states.

2.3.1.2.6 Pseudo-elements

Pseudo-elements are used to style a specific part of an element. They are defined by a double colon (::) followed by the pseudo-element name. Pseudo-elements are used to style elements based on their position or content.

```html
<p>Velit sit ad aliquip laborum labore. Excepteur tempor ad duis Lorem. Aute
   labore dolor dolor aliqua eiusmod pariatur ut duis deserunt esse velit.</p>
```

Code 2.46: Pseudo-element HTML

```css
p::first-line {
  font-weight: bold;
}

p::first-letter {
  font-size: 24px;
}

p::before {
  content: "Quote: ";
}

p::after {
  content: " - Author";
}
```

Code 2.47: Pseudo-element CSS

Code 2.46 and 2.47 show an example of pseudo-elements. In this example, the **<p>** element is targeted using the pseudo-elements **::first-line**, **::first-letter**, **::before**, and **::after**, which style the first line, first letter, and add content before and after the paragraph.

2.3.1.3 The Box Model

The box model is a fundamental concept in CSS that defines how elements are displayed on the page. It consists of four parts: content, padding, border, and margin.

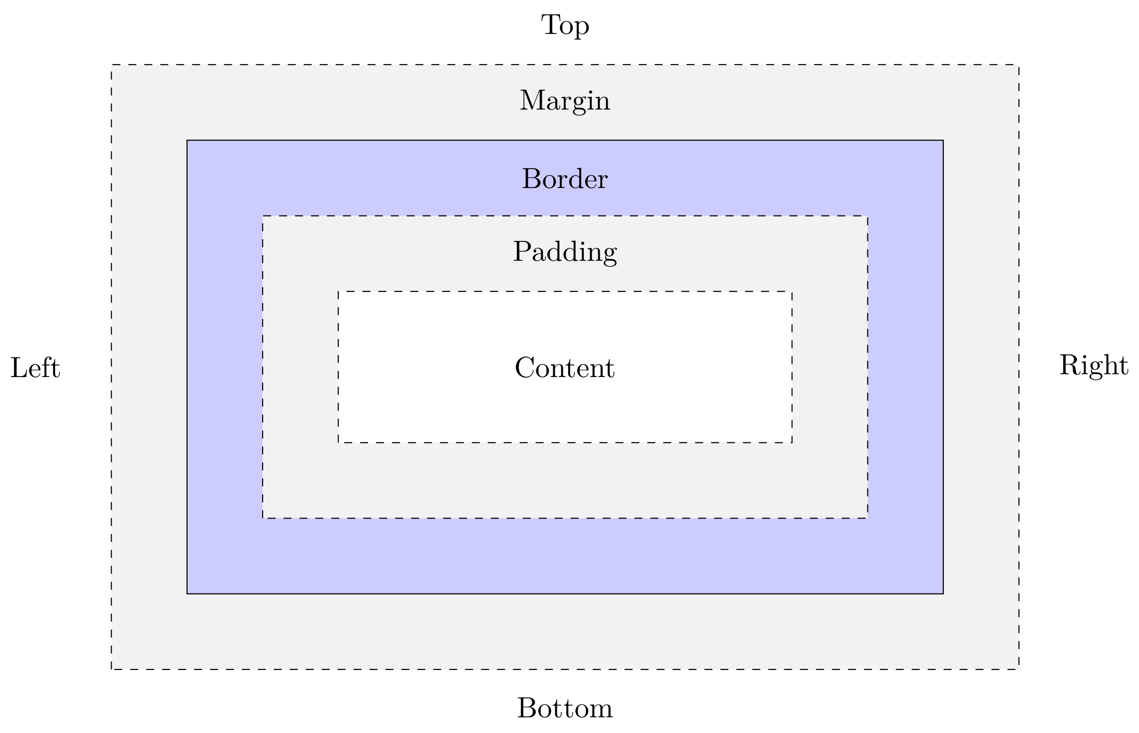

Figure 1: The Box Model

1. **Content** - The content area is where the text and images of the element are displayed. The content area is defined by the width and height of the element.

2. **Padding** - The padding area is the space between the content area and the border of the

element. Padding is used to create space around the content.

3. **Border** - The border area is the line that surrounds the padding and content of the element. The border is used to define the visual appearance of the element.

4. **Margin** - The margin area is the space outside the border of the element. The margin is used to create space between elements.

2.3.1.4 Units

In CSS, there are different units that can be used to define the size of elements. Some of the common units include:

1. **Pixels (px)** - Pixels are a fixed unit of measurement that is used to define the size of elements in terms of screen pixels. Pixels are an absolute unit and do not change based on the screen size.

2. **Percent (%)** - Percentages are a relative unit of measurement that is used to define the size of elements as a percentage of the parent element. Percentages are relative to the parent element and change based on the size of the parent element.

3. **Em (em)** - Em is a relative unit of measurement that is used to define the size of elements relative to the font size of the parent element. Em is relative to the font size of the parent element and changes based on the font size of the parent element.

4. **Rem (rem)** - Rem is a relative unit of measurement that is used to define the size of elements relative to the font size of the root element (html). Rem is relative to the font size of the root element and does not change based on the font size of the parent element.

5. **Viewport Width (vw)** - Viewport width is a relative unit of measurement that is used to define the size of elements relative to the width of the viewport. Viewport width is relative to the width of the viewport and changes based on the size of the viewport.

6. **Viewport Height (vh)** - Viewport height is a relative unit of measurement that is used to define the size of elements relative to the height of the viewport. Viewport height is relative to the height of the viewport and changes based on the size of the viewport.

2.3.2 Properties

2.3.2.1 Colors

In CSS, colors can be specified using different formats, such as color names, hexadecimal values, RGB values, and HSL values.

2.3.2.1.1 Font Color

The **color** property is used to set the color of the text.

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

Code 2.48: Font Color HTML

```css
p {
  color: red;
}
```

Code 2.49: Font Color CSS

Code 2.48 and 2.49 show an example of the **color** property. In this example, the color of the text in the **<p>** element is set to red.

2.3.2.1.2 Background Color

The **background-color** property is used to set the background color of an element.

```html
<div>
  <h1>Welcome to my website</h1>
</div>
```

Code 2.50: Background Color HTML

```css
div {
  background-color: lightblue;
}
```

Code 2.51: Background Color CSS

Code 2.50 and 2.51 show an example of the **background-color** property. In this example, the background color of the **<div>** element is set to light blue.

2.3.2.1.3 Border Color

The **border-color** property is used to set the color of the border of an element.

```html
<div>
  <h1>Welcome to my website</h1>
</div>
```

Code 2.52: Border Color HTML

```css
div {
  border: 1px solid black;
}
```

Code 2.53: Border Color CSS

Code 2.52 and 2.53 show an example of the **border-color** property. In this example, the border color of the **<div>** element is set to black.

2.3.2.1.4 Color Names

CSS provides a set of predefined color names that can be used to specify colors. Some of the common color names include:

- **Red** - red
- **Green** - green
- **Blue** - blue
- **Black** - black
- **White** - white
- **Yellow** - yellow
- **Purple** - purple
- **Orange** - orange
- **Gray** - gray
- **Brown** - brown
- **Pink** - pink
- More...

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

Code 2.54: Color Names HTML

```
p {
  color: blue;
  background-color: yellow;
  border-color: red;
  border-width: 1px;
  border-style: solid;
}
```

Code 2.55: Color Names CSS

Code 2.54 and 2.55 show an example of using color names to set the color of the text, background, and border of the **<p>** element to blue, yellow, and red, respectively.

2.3.2.1.5 Hexadecimal Colors

Hexadecimal colors are represented using a six-digit code that defines the amount of red, green, and blue as well as the transparency of the color. The code starts with a hash (#) followed by the six-digit hexadecimal code - **#RRGGBB**. Eight digits can be used to include the alpha channel - **#RRGGBBAA**.

```
<div>
  <h1>Welcome to my website</h1>
</div>
```

Code 2.56: Hexadecimal Colors HTML

```
div {
  color: #ff0000;
  background-color: #00ff00;
  border-color: #0000ff;
  border-width: 1px;
  border-style: solid;
}
```

Code 2.57: Hexadecimal Colors CSS

Code 2.56 and 2.57 show an example of using hexadecimal colors to set the color of the text, background, and border of the **<div>** element to red, green, and blue, respectively.

2.3.2.1.6 RGB or RGBA Colors

RGB colors are represented using the RGB color model, which defines the amount of red, green, and blue in the color. The RGB color is defined using the **rgb()** function with three values for red, green, and blue. The **RGBA colors** include an additional value for the alpha channel (transparency) and are defined using the **rgba()** function.

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

Code 2.58: RGB Colors HTML

```css
p {
  color: rgb(255, 0, 0);
  background-color: rgba(0, 255, 0, 0.5); /* Green with 50% opacity */
}
```

Code 2.59: RGB Colors CSS

Code 2.58 and 2.59 show an example of using RGB and RGBA colors to set the color of the text in the **<p>** element to red and the background color to green with 50% opacity.

2.3.2.1.7 HSL or HSLA Colors

HSL colors are represented using the HSL color model, which defines the hue, saturation, and lightness of the color. The HSL color is defined using the **hsl()** function with three values for hue, saturation, and lightness. The **HSLA colors** include an additional value for the alpha channel (transparency) and are defined using the **hsla()** function.

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

Code 2.60: HSL Colors HTML

```css
p {
  color: hsl(0, 100%, 50%);
  background-color: hsla(120, 100%, 50%, 0.5); /* Green with 50% opacity */
}
```

Code 2.61: HSL Colors CSS

Code 2.60 and 2.61 show an example of using HSL and HSLA colors to set the color of the text in the **<p>** element to red and the background color to green with 50% opacity.

2.3.2.1.8 Gradients

Gradients are used to create a smooth transition between two or more colors. Gradients can be linear or radial and can be defined using the **linear-gradient()** or **radial-gradient()** function.

```
<div>
  <h1>Welcome to my website</h1>
</div>
```

Code 2.62: Gradients HTML

```
div {
  padding: 20px;
  background-image: linear-gradient(to right, red, blue);
}
```

Code 2.63: Gradients CSS

Code 2.62 and 2.63 show an example of using gradients to set the background color of the **<div>** element to a linear gradient from red to blue.

2.3.2.1.9 CSS Background Image

The **background-image** property is used to specify the background image for an element. The value can be a URL to an image file or a gradient.

```
div {
  padding: 20px;
  background-image: url('https://picsum.photos/200');
}
```

Code 2.64: Background Image

Code 2.64 shows an example of using the **background-image** property to set the background image for the **<div>** element to an image from the URL.

2.3.2.1.10 CSS Background Size

The **background-size** property is used to specify the size of the background image. It can be set to a specific size, such as **cover** or **contain**, or to a percentage of the element's width and height.

```
div {
  padding: 20px;
  background-image: url('https://picsum.photos/200');
  background-size: cover;
}
```

Code 2.65: Background Size

Code 2.65 shows an example of using the **background-size** property to set the size of the background image for the **<div>** element to cover the entire element.

2.3.2.2 Box

2.3.2.2.1 Margin

The **margin** property is used to set the margin of an element. The margin is the space outside the border of the element and is used to create space between elements.

```html
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

Code 2.66: Margin HTML

```css
.container {
  display: flex;
  flex-direction: column;
}

.item {
  margin: 10px;
  width: 100px;
  height: 100px;
  background-color: lightblue;
}
```

Code 2.67: Margin CSS

Code 2.66 and 2.67 show an example of the **margin** property. In this example, the margin of the **<div>** element is set to 10 pixels, creating space around the element.

2.3.2.2.2 Padding

The **padding** property is used to set the padding of an element. The padding is the space between the content area and the border of the element and is used to create space around the content.

```html
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

Code 2.68: Padding HTML

```css
.container {
  display: flex;
  flex-direction: column;
}

.item {
  padding: 10px;
  width: 100px;
  height: 100px;
  background-color: lightblue;
}
```

Code 2.69: Padding CSS

Code 2.68 and 2.69 show an example of the **padding** property. In this example, the padding of the **<div>** element is set to 10 pixels, creating space around the content.

2.3.2.2.3 Border

The **border** property is used to set the border of an element. The border is the line that surrounds the padding and content of the element and is used to define the visual appearance of the element.

```html
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

Code 2.70: Border HTML

```css
.container {
  display: flex;
  flex-direction: column;
}

.item {
  padding: 4px;m
  margin: 4px;
  border: 1px solid black;
  width: 100px;
  height: 100px;
  background-color: lightblue;
}
```

Code 2.71: Border CSS

Code 2.70 and 2.71 show an example of the **border** property. In this example, the border of the **<div>** element is set to 1 pixel solid black, creating a border around the element.

2.3.2.2.4 Space Between

The **space-between** property is used to set the space between elements in a flex container. It distributes the extra space between elements in the container.

```
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

Code 2.72: Space Between HTML

```
.container {
  display: flex;
  justify-content: space-between;
}

.item {
  width: 100px;
  height: 100px;
  background-color: lightblue;
}
```

Code 2.73: Space Between CSS

Code 2.72 and 2.73 show an example of the **space-between** property. In this example, the **justify-content** property is set to **space-between** to distribute the extra space between the elements in the **<div>** container.

2.3.2.2.5 Width and Height

The **width** and **height** properties are used to set the width and height of an element. The width and height can be set to a specific size, such as pixels or percentages, or to **auto** to automatically adjust to the content.

```
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.74: Width and Height HTML

```
.container {
  display: flex;
  flex-direction: column;
}

.item {
  background-color: lightblue;
}

.item1 {
  width: 100px;
  height: 100px;
}

.item2 {
  width: 50%;
  height: 50%;
}

.item3 {
  width: auto;
  height: auto;
}
```

Code 2.75: Width and Height CSS

Code 2.74 and 2.75 show an example of the **width** and **height** properties. In this example, the width and height of the **<div>** elements are set to 100 pixels, 50%, and auto, creating different sizes for the elements.

2.3.2.2.6 Radius

The **border-radius** property is used to set the radius of the corners of an element. The radius can be set to a specific size, such as pixels or percentages, to create rounded corners.

```
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.76: Radius HTML

```css
.container {
  display: flex;
  flex-direction: column;
}

.item {
  width: 100px;
  height: 100px;
  background-color: lightblue;
}

.item1 {
  border-radius: 50%;
}

.item2 {
  border-radius: 10px;
}

.item3 {
  border-radius: 20%;
}
```

Code 2.77: Radius CSS

Code 2.76 and 2.77 show an example of the **border-radius** property. In this example, the radius of the corners of the **<div>** element is set to 50%, 10 pixels, and 20%, creating different levels of rounded corners.

2.3.2.2.7 Box Shadow

The **box-shadow** property is used to add a shadow effect to an element. The shadow can be set to a specific size, color, and blur radius to create different shadow effects.

```html
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.78: Box Shadow HTML

```css
.container {
  display: flex;
  flex-direction: column;
}

.item {
  width: 100px;
  height: 100px;
  background-color: lightblue;
}

.item1 {
  box-shadow: 5px 5px 5px black;
}

.item2 {
  box-shadow: 10px 10px 10px red;
}

.item3 {
  box-shadow: 0 0 10px blue;
}
```

Code 2.79: Box Shadow CSS

Code 2.78 and 2.79 show an example of the **box-shadow** property. In this example, the shadow effect of the **<div>** element is set to 5 pixels black, 10 pixels red, and 10 pixels blue, creating different shadow effects.

2.3.2.2.8 Opacity

The **opacity** property is used to set the opacity of an element. The opacity can be set to a value between 0 and 1, where 0 is fully transparent and 1 is fully opaque.

```html
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.80: Opacity HTML

```css
.container {
  display: flex;
  flex-direction: column;
}

.item {
  width: 100px;
  height: 100px;
  background-color: lightblue;
}

.item1 {
  opacity: 0.5;
}

```

```
.item2 {
  opacity: 0.7;
}

.item3 {
  opacity: 0.9;
}
```

Code 2.81: Opacity CSS

Code 2.80 and 2.81 show an example of the **opacity** property. In this example, the opacity of the **<div>** element is set to 0.5, 0.7, and 0.9, creating different levels of transparency. The higher the value, the more opaque the element.

2.3.2.3 Text

2.3.2.3.1 Text Align

The **text-align** property is used to set the alignment of the text within an element. The text can be aligned to the **left**, **right**, **center**, or **justified**.

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

Code 2.82: Text Align HTML

```
p {
  text-align: center;
}
```

Code 2.83: Text Align CSS

Code 2.82 and 2.83 show an example of the **text-align** property. In this example, the text in the **<p>** element is aligned to the center.

2.3.2.3.2 Text Decoration

The **text-decoration** property is used to set the decoration of the text within an element. The text can be **none**, **underline**, **overline**, **line-through**, or a combination of these values.

```
<p>This is a <span class="underline">paragraph</span>.</p>
```

Code 2.84: Text Decoration HTML

```
.underline {
  text-decoration: underline;
}
```

Code 2.85: Text Decoration CSS

Code 2.84 and 2.85 show an example of the **text-decoration** property. In this example, the text in the **<span>** element with the class **underline** is underlined.

2.3.2.3.3 Text Transform

The **text-transform** property is used to set the transformation of the text within an element. The text can be transformed to **uppercase**, **lowercase**, **capitalize**, or **none**.

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

Code 2.86: Text Transform HTML

```css
p {
  text-transform: uppercase;
}
```

Code 2.87: Text Transform CSS

Code 2.86 and 2.87 show an example of the **text-transform** property. In this example, the text in the **<p>** element is transformed to uppercase.

2.3.2.3.4 Line Height

The **line-height** property is used to set the height of a line of text within an element. The line height can be set to a specific size, such as pixels or percentages, to create different line heights.

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

Code 2.88: Line Height HTML

```css
p {
  line-height: 1.5;
}
```

Code 2.89: Line Height CSS

Code 2.88 and 2.89 show an example of the **line-height** property. In this example, the line height of the text in the **<p>** element is set to 1.5 times the font size.

2.3.2.3.5 Font Family

The **font-family** property is used to set the font family of the text within an element. The font family can be set to a specific font name or a list of font names to use as fallbacks.

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

Code 2.90: Font Family HTML

```css
p {
  font-family: Arial, sans-serif;
}
```

Code 2.91: Font Family CSS

Code 2.90 and 2.91 show an example of the **font-family** property. In this example, the font family of the text in the **<p>** element is set to Arial, with a fallback to the sans-serif font family.

2.3.2.3.6 Font Size

The **font-size** property is used to set the font size of the text within an element. The font size can be set to a specific size, such as pixels or percentages, to create different font sizes.

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

Code 2.92: Font Size HTML

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

Code 2.93: Font Size CSS

Code 2.92 and 2.93 show an example of the **font-size** property. In this example, the font size of the text in the **<p>** element is set to 16 pixels.

2.3.2.3.7 Font Weight

The **font-weight** property is used to set the weight of the text within an element. The font weight can be set to a specific value, such as normal, bold, or a numeric value.

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

Code 2.94: Font Weight HTML

```css
p {
  font-weight: bold;
}
```

Code 2.95: Font Weight CSS

Code 2.94 and 2.95 show an example of the **font-weight** property. In this example, the font weight of the text in the **<p>** element is set to bold.

2.3.2.3.8 Font Style

The **font-style** property is used to set the style of the text within an element. The font style can be set to **normal**, **italic**, or **oblique**.

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

Code 2.96: Font Style HTML

```css
p {
  font-style: italic;
}
```

Code 2.97: Font Style CSS

Code 2.96 and 2.97 show an example of the **font-style** property. In this example, the font style of the text in the **<p>** element is set to italic.

2.3.2.4 Layout

2.3.2.4.1 Display

The **display** property is used to set the display behavior of an element. The display property can be set to **block**, **inline**, **inline-block**, **flex**, **grid**, or **none**.

```html
<div class="container1">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>

<div class="container2">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>

<div class="container3">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

Code 2.98: Display HTML

```css
.container1 {
  display: block;
}

```

```
.container2 {
  display: inline;
}

.container3 {
  display: flex;
}

.item {
  width: 100px;
  height: 100px;
  background-color: lightblue;
}
```

Code 2.99: Display CSS

Code 2.98 and 2.99 show an example of the **display** property. In this example, the display property of the **<div>** elements is set to **block**, **inline**, and **flex**, creating different display behaviors for the elements.

2.3.2.4.2 Flexbox

The **flex** property is used to set the flex behavior of an element. The flex property can be set to **flex-grow**, **flex-shrink**, **flex-basis**, or a combination of these values.

```
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.100: Flexbox HTML

```
.container {
  display: flex;
}

.item {
  flex: 1;
  height: 100px;
}

.item1 {
  flex: 2;
  background-color: lightblue;
}

.item2 {
  flex: 1;
  background-color: lightcoral;
}

.item3 {
  flex: 1;
  background-color: lightgreen;
}
```

Code 2.101: Flexbox CSS

Code 2.100 and 2.101 show an example of the **flex** property. In this example, the flex property of the **<div>** elements is set to 1, 2, and 1, creating different flex behaviors for the elements.

2.3.2.4.3 Grid

The **grid** property is used to set the grid behavior of an element. The grid property can be set to **grid-template-columns**, **grid-template-rows**, **grid-gap**, or a combination of these values.

```
<div class="container1">
  <div class="item1">Item 1</div>
  <div class="item2">Item 2</div>
  <div class="item3">Item 3</div>
</div>

<div class="container2">
  <div class="item1">Item 1</div>
  <div class="item2">Item 2</div>
  <div class="item3">Item 3</div>
</div>

<div class="container3">
  <div class="item1">Item 1</div>
  <div class="item2">Item 2</div>
  <div class="item3">Item 3</div>
  <div class="item4">Item 4</div>
  <div class="item5">Item 5</div>
  <div class="item6">Item 6</div>
  <div class="item7">Item 7</div>
  <div class="item8">Item 8</div>
  <div class="item9">Item 9</div>
</div>
```

Code 2.102: Grid HTML

```
  .container1 {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-gap: 10px;
    margin-block: 10px;
  }
```

```css

.container2 {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  grid-gap: 10px;
  margin-block: 10px;
}

.container3 {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  grid-template-rows: 1fr 2fr 1fr;
  grid-gap: 10px;
  margin-block: 10px;
}
```

Code 2.103: Grid CSS

Code 2.102 and 2.103 show an example of the **grid** property. In this example, the grid property of the **<div>** elements is set to **1fr**, **2fr**, and **1fr**, creating different grid behaviors for the elements.

2.3.2.4.4 More on Flexbox and Grid

You can learn more about Flexbox and Grid by visiting the following links:

- Flexbox
- Grid

2.3.2.5 Positioning

2.3.2.5.1 Static

The **position** property is used to set the positioning behavior of an element. The position property can be set to **static**, **relative**, **absolute**, **fixed**, or **sticky**. The **static** value is the default value and does not affect the position of the element.

```html
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

Code 2.104: Static HTML

```css
.container {
  display: flex;
  flex-direction: column;
  position: static;
}

```

```css
.item {
  width: 100px;
  height: 100px;
  background-color: lightblue;
}
```

Code 2.105: Static CSS

Code 2.104 and 2.105 show an example of the **position** property. In this example, the position property of the **<div>** element is set to **static**, which is the default value.

2.3.2.5.2 Relative

A **relative** position is used to set the position of an element relative to its normal position. The element is positioned based on its normal position in the document flow, and then offset by the specified values.

```html
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

Code 2.106: Relative HTML

```css
.container {
  display: flex;
  flex-direction: column;
  position: relative;
  background-color: lightgreen;
}

.item {
  position: relative;
  top: 10px;
  left: 10px;
  width: 100px;
  height: 100px;
  background-color: lightblue;
}
```

Code 2.107: Relative CSS

Code 2.106 and 2.107 show an example of the **position** property with a value of **relative**. In this example, the position of the **<div>** element is set to **relative**, and the **top** and **left** properties are used to offset the element by 10 pixels.

2.3.2.5.3 Absolute

An **absolute** position is used to set the position of an element relative to its nearest positioned ancestor. If no ancestor is positioned, the element is positioned relative to the document body.

```html
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.108: Absolute HTML

```css
.container {
  display: flex;
  flex-direction: column;
  position: relative;
  background-color: lightgreen;
  width: 200px;
  height: 200px;
}

.item {
  position: absolute;
  width: 100px;
  height: 100px;
}

.item1 {
  top: 0;
  left: 0;
  background-color: lightcoral;
}

.item2 {
  top: 0;
  right: 0;
  background-color: lightblue;
}

.item3 {
  bottom: 0;
  left: 0;
  background-color: yellow;
}
```

Code 2.109: Absolute CSS

Code 2.108 and 2.109 show an example of the **position** property with a value of **absolute**. In this example, the position of the **<div>** elements is set to **absolute**, and the **top**, **right**,

bottom, and **left** properties are used to position the elements in the container.

2.3.2.5.4 Fixed

A **fixed** position is used to set the position of an element relative to the viewport. The element remains fixed in the same position on the screen even when the page is scrolled.

```html
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2 - Fixed</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.110: Fixed HTML

```css
.container {
  display: flex;
  flex-direction: column;
  position: relative;
  background-color: lightgreen;
  height: 200vh;
}

.item {
  width: 100px;
  height: 100px;
}

.item1 {
  background-color: lightcoral;
}

.item2 {
  position: fixed;
  top: 50%;
  right: 0;
  transform: translateY(-50%);
  background-color: lightblue;
}

.item3 {
  background-color: yellow;
}
```

Code 2.111: Fixed CSS

Code 2.110 and 2.111 show an example of the **position** property with a value of **fixed**. In this example, the position of the **<div>** element is set to **fixed**, and the **top** and **right** properties are used to position the element on the screen.

2.3.2.5.5 Sticky

A **sticky** position is used to set the position of an element relative to its containing block. The element remains in the normal flow until it reaches a specified scroll position, at which point it becomes **fixed**.

```html
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2 - Sticky</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.112: Sticky HTML

```css
.container {
  display: flex;
  flex-direction: column;
  position: relative;
  background-color: lightgreen;
  height: 200vh;
}

.item {
  width: 100px;
  height: 100px;
}

.item1 {
  background-color: lightcoral;
}

.item2 {
  position: sticky;
  top: 0;
  background-color: lightblue;
}

.item3 {
  background-color: yellow;
}
```

Code 2.113: Sticky CSS

Code 2.112 and 2.113 show an example of the **position** property with a value of **sticky**. In this example, the position of the **<div>** element is set to **sticky**, and the **top** property is used to position the element on the screen.

2.3.2.5.6 More on Positioning

You can learn more about positioning by visiting the following links:

- Position
- Z-Index

2.3.2.6 Transforms

2.3.2.6.1 Scale

The **scale** property is used to scale an element. The scale value can be set to a specific size, such as 1.5 or 0.5, to scale the element up or down.

```html
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.114: Scale HTML

```css
.container {
  display: flex;
  flex-direction: column;
}

.item {
  width: 100px;
  height: 100px;
}

.item1 {
  transform: scale(1.5);
  background-color: lightblue;
}

.item2 {
  transform: scale(0.5);
  background-color: lightcoral;
}

.item3 {
  transform: scale(2);
  background-color: lightgreen;
}
```

Code 2.115: Scale CSS

Code 2.114 and 2.115 show an example of the **transform** property with a value of scale. In this example, the scale value of the **<div>** elements is set to 1.5, 0.5, and 2, creating different scale effects for the elements.

2.3.2.6.2 Rotate

The **rotate** property is used to rotate an element. The rotate value can be set to a specific angle, such as 45 degrees or 90 degrees, to rotate the element.

```html
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.116: Rotate HTML

```css
.container {
  display: flex;
  flex-direction: column;
}

.item {
  width: 100px;
  height: 100px;
}

.item1 {
  transform: rotate(45deg);
  background-color: lightblue;
}

.item2 {
  transform: rotate(90deg);
  background-color: lightcoral;
}

.item3 {
  transform: rotate(180deg);
  background-color: lightgreen;
}
```

Code 2.117: Rotate CSS

Code 2.116 and 2.117 show an example of the **transform** property with a value of rotate. In this example, the rotate value of the **<div>** elements is set to 45 degrees, 90 degrees, and 180 degrees, creating different rotate effects for the elements.

2.3.2.6.3 Skew

The **skew** property is used to skew an element. The skew value can be set to a specific angle, such as 30 degrees or -30 degrees, to skew the element.

```html
<div class="container">
```

```
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.118: Skew HTML

```
.container {
  display: flex;
  flex-direction: column;
}

.item {
  width: 100px;
  height: 100px;
}

.item1 {
  transform: skew(30deg);
  background-color: lightblue;
}

.item2 {
  transform: skew(-30deg);
  background-color: lightcoral;
}

.item3 {
  transform: skew(45deg);
  background-color: lightgreen;
}
```

Code 2.119: Skew CSS

Code 2.118 and 2.119 show an example of the **transform** property with a value of skew. In this example, the skew value of the **<div>** elements is set to 30 degrees, -30 degrees, and 45 degrees, creating different skew effects for the elements.

2.3.2.6.4 Translate

The **translate** property is used to move an element. The translate value can be set to a specific distance, such as 10 pixels or -10 pixels, to move the element.

```
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>

```

```
\begin{lstlisting}[language=HTML, caption={Translate CSS},
    label={lst:translate-css}]
.container {
  display: flex;
  flex-direction: column;
}

.item {
  width: 100px;
  height: 100px;
}

.item1 {
  transform: translate(10px, 10px);
  background-color: lightblue;
}

.item2 {
  transform: translate(-10px, -10px);
  background-color: lightcoral;
}

.item3 {
  transform: translate(20px, 20px);
  background-color: lightgreen;
}
```

Code 2.120: Translate HTML

Code 2.120 and **??** show an example of the **transform** property with a value of translate. In this example, the translate value of the **<div>** elements is set to 10 pixels, -10 pixels, and 20 pixels, creating different translate effects for the elements.

2.3.2.6.5 Origin

The **transform-origin** property is used to set the origin of the transformed element. The origin value can be set to a specific position, such as top left, center, or bottom right, to change the origin of the transformation.

```html
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>
```

Code 2.121: Origin HTML

```css
.container {
  display: flex;
  flex-direction: column;
}

.item {
  width: 100px;
  height: 100px;
}

.item1 {
  transform: rotate(45deg);
  transform-origin: top left;
  background-color: lightblue;
}

.item2 {
  transform: rotate(45deg);
  transform-origin: center;
  background-color: lightcoral;
}

.item3 {
  transform: rotate(45deg);
  transform-origin: bottom right;
  background-color: lightgreen;
}
```

Code 2.122: Origin CSS

Code 2.121 and 2.122 show an example of the **transform-origin** property. In this example, the origin of the transformation of the **<div>** elements is set to top left, center, and bottom right, creating different origin effects for the elements.

2.3.2.7 Interactivity

2.3.2.7.1 Hover

The **hover** property is used to change the style of an element when the mouse hovers over it. The hover property can be used to change the color, background color, or other styles of the element.

```
<button>Hover Over Me</button>
```

Code 2.123: Hover HTML

```
button {
  background-color: lightblue;
  color: white;
  padding: 10px 20px;
  border: none;
}

```

```css
button:hover {
  background-color: lightcoral;
}
```

Code 2.124: Hover CSS

Code 2.123 and 2.124 show an example of the **hover** property. In this example, the background color of the **<button>** element is changed to **lightcoral** when the mouse hovers over it.

2.3.2.7.2 Focus

The **focus** property is used to change the style of an element when it receives focus. The focus property can be used to change the color, background color, or other styles of the element.

```html
<input type="text" placeholder="Enter your name">
```

Code 2.125: Focus HTML

```css
input {
  padding: 10px;
  background: lightblue;
}

input:focus {
  background: lightcoral;
}
```

Code 2.126: Focus CSS

Code 2.125 and 2.126 show an example of the **focus** property. In this example, the background color of the **<input>** element is changed to **lightcoral** when it receives focus.

2.3.2.7.3 Active

The **active** property is used to change the style of an element when it is activated. The active property can be used to change the color, background color, or other styles of the element.

```html
<button>Click Me</button>
```

Code 2.127: Active HTML

```css
button {
  background-color: lightblue;
  color: white;
  padding: 10px 20px;
  border: none;
}

```

```
button:active {
  background-color: lightcoral;
}
```

Code 2.128: Active CSS

Code 2.127 and 2.128 show an example of the **active** property. In this example, the background color of the **<button>** element is changed to **lightcoral** when it is activated.

2.3.2.7.4 Transition

The **transition** property is used to create smooth transitions between different styles of an element. The transition property can be set to a specific duration, such as 0.5 seconds or 1 second, to create a smooth transition effect.

```
<button>Hover Over Me</button>
```

Code 2.129: Transition HTML

```
button {
  background-color: lightblue;
  color: white;
  padding: 10px 20px;
  border: none;
  transform: scale(1);
  transition: all 0.5s;
}

button:hover {
  transform: scale(1.1);
  background-color: lightcoral;
}
```

Code 2.130: Transition CSS

Code 2.129 and 2.130 show an example of the **transition** property. In this example, the background color of the **<button>** element is changed to **lightcoral** and the scale is increased when the mouse hovers over it, creating a smooth transition effect.

Read more about CSS transitions from the following sources:

- MDN Web Docs CSS Transitions
- W3Schools CSS Transitions
- CSS-Tricks Transition

2.3.2.7.5 Animation

The **animation** property is used to create animations for an element. The animation property can be set to a specific duration, timing function, delay, and iteration count to create different animation effects.

```html
<div class="container">
  <div class="item">Item 1</div>
</div>
```

Code 2.131: Animation HTML

```css
.container {
  display: flex;
  justify-content: center;
  align-items: center;
}

.item {
  width: 100px;
  height: 100px;
  background-color: lightblue;
  animation: spin 2s linear infinite;
}

@keyframes spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Code 2.132: Animation CSS

Code 2.131 and 2.132 show an example of the **animation** property. In this example, the **<div>** element is rotated 360 degrees over a duration of 2 seconds, creating a spinning animation.

Read more about CSS animations from the following sources:

- MDN Web Docs CSS Animations
- W3Schools CSS Animations
- CSS-Tricks Animation

2.3.3 Filters

The **filter** property is used to apply visual effects to an element. The filter property can be set to **blur**, **brightness**, **contrast**, **grayscale**, **hue-rotate**, **invert**, **saturate**, **sepia**, **drop-shadow**, or a combination of these values.

```html
<div class="container">
  <div class="item blur">Blur</div>
  <div class="item brightness">Brightness</div>
  <div class="item contrast">Contrast</div>
  <div class="item grayscale">Grayscale</div>
  <div class="item hue-rotate">Hue Rotate</div>
  <div class="item invert">Invert</div>
  <div class="item saturate">Saturate</div>
  <div class="item sepia">Sepia</div>
  <div class="item drop-shadow">Drop Shadow</div>
</div>
```

Code 2.133: Filter HTML

```
.container {
  display: flex;
  flex-direction: column;
  gap: 8px;
}

.item {
  width: 100px;
  height: 100px;
  display: flex;
  justify-content: center;
  align-items: center;
}

.blur {
  background-color: blue;
  filter: blur(5px);
}

.brightness {
  background-image: url('https://picsum.photos/200');
  filter: brightness(150%);
}

.contrast {
  background-color: red;
  filter: contrast(200%);
}

.grayscale {
  background-color: yellow;
  filter: grayscale(100%);
}

.hue-rotate {
  background-color: purple;
  filter: hue-rotate(90deg);
}

.invert {
  background-color: orange;
  filter: invert(100%);
}

.saturate {
  background-color: pink;
  filter: saturate(50%);
}

.sepia {
  background-color: brown;
  filter: sepia(100%);
}

.drop-shadow {
  background-color: lightblue;
  filter: drop-shadow(10px 10px 5px rgba(0, 0, 0, 0.5));
}
```

Code 2.134: Filter CSS

Code 2.133 and 2.134 show an example of the **filter** property. In this example, different visual effects are applied to the **<div>** elements using the filter property.

Read more about CSS filters from the following sources:

- MDN Web Docs CSS Filters
- W3Schools CSS Filters
- CSS-Tricks Filter

2.3.3.1 Responsive Design

2.3.3.1.1 Media Queries

The **@media** rule is used to apply different styles for different media types, such as screen sizes, devices, or orientations. Media queries can be used to create responsive designs that adapt to different screen sizes.

```html
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

Code 2.135: Media Queries HTML

```css
.container {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  gap: 12px;
}

.item {
  display: flex;
  justify-content: center;
  align-items: center;
  width: 150px;
  height: 150px;
  background-color: lightblue;
}

@media screen and (max-width: 600px) {
  .item {
    width: 100px;
    height: 100px;
  }
}

@media screen and (max-width: 400px) {
  .container {
    flex-direction: column;
  }
}
```

Code 2.136: Media Queries CSS

Code 2.135 and 2.136 show an example of media queries. In this example, the flex direction of the **<div>** elements is changed to column when the screen width is less than 400 pixels, and the width and height of the elements are changed when the screen width is less than 600 pixels.

Read more about media queries from the following sources:

- MDN Web Docs Media Queries
- W3Schools Media Queries
- CSS-Tricks Media Queries

2.3.3.1.2 Responsive Images

The **max-width** property is used to make images responsive by setting the maximum width to 100% of the container. This allows the image to scale down proportionally to fit the container while maintaining its aspect ratio.

```
<div class="container">
  <img src="https://picsum.photos/400" alt="Placeholder Image" />
</div>
```

Code 2.137: Responsive Images HTML

```
.container {
  max-width: 600px;
}

img {
  max-width: 100%;
  height: auto;
}
```

Code 2.138: Responsive Images CSS

Code 2.137 and 2.138 show an example of responsive images. In this example, the maximum width of the image is set to 100% of the container, allowing the image to scale down proportionally to fit the container.

2.3.4 Frameworks

2.3.4.1 Bootstrap

Bootstrap is a popular CSS framework that provides a set of pre-designed components and styles to create responsive web designs quickly. Bootstrap includes a grid system, typography, forms, buttons, navigation bars, and other components that can be customized to create modern web designs.

Read more about Bootstrap from its official documentation:

Bootstrap Documentation

2.3.4.2 Tailwind CSS

Tailwind CSS is a utility-first CSS framework that provides a set of utility classes to style web designs. Tailwind CSS allows you to create custom designs by applying utility classes directly in the HTML markup, making it easier to build responsive and customizable web designs.

Read more about Tailwind CSS from its official documentation:

Tailwind CSS Documentation

2.3.5 More about CSS

You can find more information about CSS properties from the following sources:

- MDN Web Docs CSS Reference
- W3Schools CSS Reference
- Toptal CSS Cheat Sheet

3

Version Control

3.1 Introduction to Version Control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. It allows you to track changes, collaborate with others, and maintain a history of your work.

3.1.1 Types of Version Control Systems

There are three main types of version control systems:

- **Local Version Control System**: A local version control system stores changes to files in a database on the local machine. It allows you to track changes to files and revert to previous versions.
- **Centralized Version Control System**: A centralized version control system stores changes to files in a central server. It allows multiple users to collaborate on a project and maintain a history of changes.
- **Distributed Version Control System**: A distributed version control system stores changes to files in a distributed repository. It allows multiple users to work on a project independently and merge changes later.

3.1.2 Benefits of Version Control

Version control provides several benefits for developers and teams:

- **History Tracking**: Version control systems maintain a history of changes to files, allowing you to track who made changes and when.
- **Collaboration**: Version control systems enable multiple users to work on the same project and merge changes seamlessly.
- **Branching and Merging**: Version control systems support branching and merging, allowing you to work on different features independently and merge changes later.
- **Backup and Recovery**: Version control systems provide a backup of your work and allow you to recover previous versions of files.

3.1.3 Common Use Cases

Version control is commonly used for the following purposes:

- **Software Development**: Version control is essential for software development to track changes, collaborate with others, and maintain a history of code changes.
- **Documentation**: Version control is used to manage documentation, track changes to documents, and collaborate on writing projects.
- **Academic Research**: Version control is used in academic research to track changes to research papers, collaborate with colleagues, and maintain a history of research projects.

3.2 Git Basics

3.2.1 Introduction to Git

Git is a distributed version control system that allows you to track changes to files, collaborate with others, and maintain a history of your work. Git is widely used in software development to manage code changes, track project history, and collaborate with other developers. It was created by Linus Torvalds in 2005 and is an open-source tool that is free to use. Nowadays, Git is the most popular version control system and is used by millions of developers around the world. Not only is Git used for software development, but it is also used for managing configuration files, documentation, and other types of files.

3.2.2 Setting Up Git

To get started with Git, you need to install Git on your local machine and configure your Git username and email address. You can download Git from the official Git website and follow the installation instructions for your operating system. Once Git is installed, you can configure your username and email address using the following commands:

```
git config --global user.name "Your Name"
git config --global user.email "example@example.com"
```

Code 3.1: Setting Up Git

Code 3.1 shows the commands to set up your Git username and email address. Replace "Your Name" with your actual name and "example@example.com" with your actual email address. These settings are used to identify you as the author of the changes you make in Git.

3.2.3 Core Concepts

Git has several core concepts that you need to understand to use Git effectively:

- **Repository**: A repository is a collection of files and directories that are tracked by Git. It contains the project history, branches, and configuration settings.
- **Commit**: A commit is a snapshot of the project at a specific point in time. It records changes to files and includes a commit message that describes the changes.
- **Branch**: A branch is a parallel version of the project that allows you to work on different features independently. It is used to isolate changes and merge them later.
- **Merge**: Merging is the process of combining changes from one branch into another branch. It is used to integrate changes and resolve conflicts between branches.
- **Remote**: A remote is a shared repository that is hosted on a server. It allows multiple users to collaborate on a project and share changes with others.
- **Staging Area**: The staging area is a temporary storage area where changes are prepared before committing them to the repository. It allows you to review changes and select which

changes to commit.

3.2.4 Git Workflow

The typical Git workflow consists of the following steps:

3.2.4.1 Initialize Repository

To start using Git, you need to initialize a new repository in your project directory. You can do this by running the following command:

```
git init
```

Code 3.2: Initialize Repository

Code 3.2 shows the command to initialize a new Git repository in your project directory. This command creates a hidden **.git** directory that contains the project history and configuration settings.

3.2.4.2 Add and Commit Changes

After initializing a repository, you can add files to the staging area and commit changes to the repository. You can add files to the staging area using the **git add** command and commit changes using the **git commit** command:

```
git add file.txt
git commit -m "Add file.txt"
```

Code 3.3: Add and Commit Changes

Code 3.3 shows the commands to add a file to the staging area and commit changes to the repository. Replace **file.txt** with the name of the file you want to add, and **Add file.txt** with a descriptive commit message.

3.2.4.3 View History and Changes

You can view the project history and changes using the **git log** and **git diff** commands. The **git log** command displays the commit history, and the **git diff** command shows the changes between files:

```
git log
git diff file.txt
```

Code 3.4: View History and Changes

Code 3.4 shows the commands to view the project history and changes. The **git log** command displays the commit history, and the **git diff** command shows the changes between files.

3.2.4.4 Undoing Changes

You can undo changes in Git using the **git checkout**, **git reset**, and **git revert** commands. The **git checkout** command is used to discard changes in the working directory, the **git reset** command is used to unstage changes in the staging area, and the **git revert** command is used to revert commits in the repository.

3.2.4.4.1 Checkout

The **git checkout** command is used to discard changes in the working directory. You can use it to revert changes to a specific file or restore the project to a previous commit:

```
git checkout file.txt
git checkout HEAD~1
```

Code 3.5: Checkout

Code 3.5 shows the commands to discard changes to a specific file and restore the project to a previous commit. Replace **file.txt** with the name of the file you want to discard changes to, and **HEAD 1** with the commit you want to restore the project to.

3.2.4.4.2 Reset

The **git reset** command is used to unstage changes in the staging area. You can use it to unstage changes to a specific file or reset the staging area to the last commit:

```
git reset file.txt
git reset
```

Code 3.6: Reset

Code 3.6 shows the commands to unstage changes to a specific file and reset the staging area to the last commit. Replace **file.txt** with the name of the file you want to unstage changes to.

3.2.4.4.3 Revert

The **git revert** command is used to revert commits in the repository. You can use it to create a new commit that undoes the changes introduced by a specific commit:

```
git revert HEAD
```

Code 3.7: Revert

Code 3.7 shows the command to revert a specific commit in the repository. Replace **HEAD** with the commit you want to revert.

3.3 Branching and Merging

3.3.1 What is Branching?

Branching allows developers to diverge from the main codebase to work independently. Branches are pointers to commits, enabling parallel development.

3.3.2 Creating and Switching Branches

You can create and switch branches in Git using the **git branch** and **git checkout** commands. The **git branch** command is used to create a new branch, and the **git checkout** command is used to switch to a branch:

```
git branch feature
git checkout feature
```

Code 3.8: Creating and Switching Branches

Code 3.8 shows the commands to create a new branch named **feature** and switch to the **feature** branch. Replace **feature** with the name of the branch you want to create and switch to.

3.3.3 Merging Branches

You can merge branches in Git using the **git merge** command. The **git merge** command is used to combine changes from one branch into another branch:

```
git checkout main
git merge feature
```

Code 3.9: Merging Branches

Code 3.9 shows the commands to switch to the **main** branch and merge changes from the **feature** branch into the **main** branch. Replace **main** with the name of the branch you want to merge changes into.

3.3.4 Resolving Merge Conflicts

Merge conflicts occur when Git is unable to automatically merge changes from different branches. You can resolve merge conflicts by editing the conflicting files, marking the conflicts as resolved, and committing the changes:

```
git status
# Edit conflicting files
git add file.txt
git commit -m "Resolve merge conflicts"
```

Code 3.10: Resolving Merge Conflicts

Code 3.10 shows the commands to resolve merge conflicts in Git. After editing the conflicting files, you need to add the files to the staging area and commit the changes to mark the conflicts as resolved.

3.4 Collaboration with GitHub

3.4.1 Introduction to GitHub

GitHub is a web-based platform that provides hosting for Git repositories. It allows developers to collaborate on projects, share code, and contribute to open-source projects. GitHub provides features such as remote repositories, pull requests, issues, and projects to facilitate collaboration and project management.

3.4.2 Key Features

GitHub provides several key features that make it a popular platform for collaboration and open-source development:

- **Remote Repositories**: GitHub hosts remote repositories that allow developers to store and share code with others.
- **Pull Requests**: Pull requests enable developers to propose changes, review code, and merge changes into the main codebase.
- **Issues and Projects**: GitHub provides tools for issue tracking and project management, allowing developers to track bugs, feature requests, and tasks.
- **GitHub Pages**: GitHub Pages is a feature that allows developers to host static websites directly from a GitHub repository.

3.4.2.1 Remote Repositories

Remote repositories in GitHub allow developers to store code in the cloud and collaborate with others. You can push changes to a remote repository using the **git push** command and pull changes from a remote repository using the **git pull** command:

```
git remote add origin
git push -u origin main
git pull origin main
```

Code 3.11: Remote Repositories

Code 3.11 shows the commands to add a remote repository, push changes to the remote repository, and pull changes from the remote repository. Replace **origin** with the name of the remote repository.

3.4.2.2 Pull Requests

Pull requests in GitHub allow developers to propose changes, review code, and merge changes into the main codebase. You can create a pull request from a branch in your repository to another branch in the same repository or a different repository.

3.4.2.3 Issues and Projects

Issues and **Projects** in GitHub provide tools for tracking bugs, feature requests, and tasks. You can create issues to report bugs or request features, assign issues to team members, and track the progress of issues using labels and milestones. Projects allow you to organize and prioritize tasks, track progress, and collaborate with others.

3.4.2.4 GitHub Pages

GitHub Pages is a feature that allows developers to host static websites directly from a GitHub repository. You can create a GitHub Pages site by enabling the feature in the repository settings and configuring the source for the site.

3.5 Best Practices

3.5.1 Git Ignore

The **.gitignore** file is used to specify files and directories that should be ignored by Git. You can create a **.gitignore** file in the root directory of your project and add patterns for files and directories that you want to exclude from version control:

```
# Ignore build files
build/

# Ignore configuration files
config.json
```

Code 3.12: Git Ignore

Code 3.12 shows an example of a **.gitignore** file that excludes the **build** directory and **config.json** file from version control. You can add patterns for files and directories that you want to ignore in the **.gitignore** file.

3.5.2 Commit Messages

Commit messages should be clear, concise, and descriptive. A good commit message should explain the changes made in the commit and provide context for the changes. You can follow the **imperative mood** and use the **subject line** and **body** to structure your commit messages:

```
git commit -m "Add feature X"

Add feature X to improve performance
```

Code 3.13: Commit Messages

Code 3.13 shows an example of a commit message with a subject line and body. The subject line is concise and describes the changes made in the commit, while the body provides additional context for the changes.

3.5.3 Branch Naming

Branch names should be descriptive and follow a consistent naming convention. You can use prefixes such as **feature/**, **bugfix/**, or **hotfix/** to categorize branches based on the type of changes:

```
git branch feature/add-x
git branch bugfix/fix-y
git branch hotfix/patch-z
```

Code 3.14: Branch Naming

Code 3.14 shows examples of branch names with prefixes to categorize branches based on the type of changes. You can use prefixes to organize branches and make it easier to identify the purpose of each branch.

3.5.4 Code Reviews

Code reviews are an essential part of the development process that helps improve code quality, identify bugs, and share knowledge among team members. You can use pull requests in GitHub to request code reviews, provide feedback on code changes, and collaborate with others to review and merge changes.

3.5.5 More about Version Control, Git, and GitHub

You can find more information about version control, Git, and GitHub from the following sources:

- Git Documentation
- GitHub Guides
- Atlassian Git Tutorials

4

NextJS

4.1 Introduction to NextJS

NextJS is a popular React framework that provides a set of tools and features to build modern web applications. NextJS is built on top of React and provides server-side rendering, static site generation, and other performance optimizations out of the box. It is widely used for building static websites, e-commerce platforms, blogs, and other types of web applications.

4.1.1 What is NextJS?

NextJS is a React framework that provides a set of tools and features to simplify the development of web applications. It includes built-in support for server-side rendering, static site generation, and other performance optimizations to improve the user experience. NextJS is designed to be easy to use and flexible, allowing developers to build modern web applications quickly and efficiently.

4.1.2 History and Evolution

NextJS was created by Vercel (formerly ZEIT) in 2016 and has since become one of the most popular React frameworks for building web applications. It has evolved over the years with new features and improvements to simplify the development process and improve performance. NextJS is actively maintained by the Vercel team and the open-source community.

4.2 Installation

4.2.1 Pre-requisites

Before installing NextJS, you need to have Node.js and npm (Node Package Manager) installed on your local machine. You can download and install Node.js from the official Node.js website. npm is included with Node.js, so you don't need to install it separately. You can download Node.js from the following link:

Node.js Download

4.2.2 Automatic Installation

You can create a new NextJS project using the **create-next-app** command provided by the NextJS team. This command sets up a new NextJS project with all the necessary dependencies and configuration files:

```
npx create-next-app@latest
```

Code 4.1: Automatic Installation

On installation, you'll see the following prompts:

```
What is your project named? my-app
Would you like to use TypeScript? No / Yes
Would you like to use ESLint? No / Yes
Would you like to use Tailwind CSS? No / Yes
Would you like your code inside a ‘src/‘ directory? No / Yes
Would you like to use App Router? (recommended) No / Yes
Would you like to use Turbopack for ‘next dev‘? No / Yes
Would you like to customize the import alias (‘@/*‘ by default)? No / Yes
What import alias would you like configured? @/*
```

Code 4.2: Installation Prompts

After the prompts, **create-next-app** will create a folder with your project name and install the required dependencies.

4.3 Core Features of NextJS

4.3.1 File-based Routing

NextJS uses a file-based routing system that allows you to create pages by adding files to the **pages** directory. Each file in the **pages** directory corresponds to a route in the application, making it easy to create dynamic and static pages:

```
pages/
index.js
about.js
contact.js
```

Code 4.3: File-based Routing

The example above shows the file-based routing system in NextJS. The **index.js** file corresponds to the home page, the **about.js** file corresponds to the about page, and the **contact.js** file corresponds to the contact page.

NextJS has two types of routing methods: **pages router** and **app router**.

4.3.1.1 Pages Router

Pages router is the traditional way of routing in NextJS. It uses the file-based routing system to create pages by adding files to the **pages** directory. Each file in the **pages** directory corresponds to a route in the application.

```
pages/
├── index.js          // Maps to '/'
├── about.js          // Maps to '/about'
├── blog/
│   └── [id].js       // Maps to '/blog/:id'
└── api/
│   └── hello.js      // API endpoint at '/api/hello'
```

The example above shows the structure of the pages router in NextJS. The **index.js** file corresponds to the home page, the **about.js** file corresponds to the about page, the **[id].js** file corresponds to dynamic blog pages, and the **hello.js** file corresponds to an API endpoint.

4.3.1.2 App Router

App router is a new routing method introduced in NextJS 13. It offers a new approach to routing, leveraging React's latest features like Server Components and Suspense. It uses the **app** directory instead of pages and provides enhanced capabilities for building modern applications.

```
app/
├── layout.js              // Root layout
├── page.js                // Maps to '/'
├── about/
│   └── page.js            // Maps to '/about'
├── blog/
│   ├── [id]/
│   │   └── page.js        // Maps to '/blog/:id'
│   └── page.js            // Maps to '/blog'
└── api/
│   └── hello/route.js     // API endpoint at '/api/hello'
```

The example above shows the structure of the app router in NextJS. The **layout.js** file corresponds to the root layout, the **page.js** file corresponds to the home page, the **about/page.js** file corresponds to the about page, the **blog/[id]/page.js** file corresponds to dynamic blog pages, and the **api/hello/route.js** file corresponds to an API endpoint.

4.3.2 Image and Font Optimization

NextJS provides built-in support for image and font optimization to improve performance and reduce loading times. It automatically optimizes images and fonts by compressing, resizing, and caching them to deliver the best possible user experience.

4.3.3 Multiple Rendering Methods

NextJS supports multiple rendering methods, including server-side rendering (SSR), client-side rendering (CSR), static site generation (SSG), and incremental static regeneration (ISR). You can choose the rendering method that best suits your application requirements and performance goals.

4.3.3.1 Server-side Rendering (SSR)

Server-side rendering (SSR) is a rendering method that generates HTML on the server and sends it to the client. It allows you to pre-render pages with dynamic data and improve SEO and performance by delivering fully rendered pages to the client.

4.3.3.2 Client-side Rendering (CSR)

Client-side rendering (CSR) is a rendering method that generates HTML on the client using JavaScript. It allows you to build interactive and dynamic web applications by rendering pages in the browser and fetching data dynamically.

4.3.3.3 Static Site Generation (SSG)

Static site generation (SSG) is a rendering method that generates HTML at build time and serves static files to the client. It allows you to pre-render pages with static data and improve performance by delivering static content to the client.

4.3.3.4 Incremental Static Regeneration (ISR)

Incremental static regeneration (ISR) is a rendering method that generates HTML at build time and revalidates and updates pages at runtime. It allows you to update static content dynamically and improve performance by serving updated content to the client.

4.3.4 API Routes

NextJS provides built-in support for API routes that allow you to create serverless functions to handle API requests. You can create API routes by adding files to the **pages/api** or **app/api** directory and defining the route logic.

4.3.5 Built-in CSS Support

NextJS provides built-in support for CSS modules, CSS-in-JS libraries, and global CSS styles. You can import CSS files directly into your components and stylesheets and use them to style your web applications.

4.3.6 TypeScript Support

NextJS has built-in support for TypeScript, a statically typed superset of JavaScript that helps you write safer and more maintainable code. You can use TypeScript in your NextJS projects by adding a **tsconfig.json** file and installing the necessary dependencies.

4.3.7 More about NextJS

You can find more information about NextJS from the following sources:

- NextJS Documentation
- NextJS Learn

4.4 Components

4.4.1 Pages

A page is UI that is rendered on a specific route. To create a page, add a page file inside the app directory and default export a React component. For example, to create an index page (/):

```
app/
├── page.js
```

```
export default function Page() {
  return <h1>Hello, Next.js!</h1>;
}
```

Code 4.4: Creating a Page

Code 4.4 shows an example of creating a page in NextJS. The **page.js** file exports a React component that renders an **h1** element with the text "Hello, Next.js!".

4.4.2 Layouts

A layout is UI that is shared between multiple pages. On navigation, layouts preserve state, remain interactive, and do not rerender.

You can define a layout by default exporting a React component from a layout file. The component should accept a children prop which can be a page or another layout.

For example, to create a layout that accepts your index page as child, add a layout file inside the app directory:

```
app/
├── layout.js
├── page.js
```

```
export default function Layout({ children }) {
  return (
    <div>
      <h1>My Layout</h1>
      {children}
    </div>
  );
}
```

Code 4.5: Creating a Layout

Code 4.5 shows an example of creating a layout in NextJS. The **layout.js** file exports a React component that renders an **h1** element with the text "My Layout" and accepts a **children** prop to render the child components.

4.4.3 Reusable Components

Reusable components are UI elements that can be shared across multiple pages and layouts. You can create reusable components by defining React components and importing them into your pages and layouts.

For example, to create a reusable button component, define a button file inside the components directory:

```
components/
├── button.js
app/
```

```
├── layout.js
├── page.js
```

```
export default function Button({ children }) {
  return <button>{children}</button>;
}
```

Code 4.6: Creating a Reusable Component

Code 4.6 shows an example of creating a reusable button component in NextJS. The **button.js** file exports a React component that renders a **button** element with the children passed as props.

4.4.4 More about Components in NextJS

You can find more information about components in NextJS from the following sources:

- NextJS Documentation: Pages
- NextJS Documentation: Layouts
- NextJS Documentation: Components

4.5 Styling

4.5.1 CSS Modules

CSS modules are a way to scope CSS locally by default. It allows you to write CSS styles for a specific component without worrying about global styles or naming conflicts. CSS modules generate unique class names for each component, making it easy to style components in isolation.

```
/* styles.module.css */
.container {
  background-color: lightblue;
}
```

Code 4.7: CSS Modules

```
import styles from './styles.module.css';

export default function Component() {
  return <div className={styles.container}>Hello, Next.js!</div>;
}
```

Code 4.8: CSS Modules Usage

Code 4.7 shows an example of a CSS module file that defines a **container** class with a background color of light blue. Code 4.8 shows how to use the CSS module in a React component by importing the styles and applying the **container** class to the **div** element.

4.5.2 External Stylesheets

NextJS allows you to import external stylesheets into your components by importing the CSS file directly into the component. You can use global CSS stylesheets to style your components and apply styles across the entire application.

```css
/* styles.css */
.btn-success {
  background-color: #28a745;
  color: #fff;
  border-radius: 4px;
}
```

Code 4.9: External Stylesheet

```jsx
import '../styles.css';

export default function Component() {
  return <button className="btn-success">Click me</button>;
}
```

Code 4.10: External Stylesheet Usage

Code 4.9 shows an example of an external CSS file that defines a **btn-success** class with styles for a success button. Code 4.10 shows how to use the external stylesheet in a React component by importing the CSS file and applying the **btn-success** class to the **button** element.

4.5.3 CSS-in-JS Libraries

NextJS supports CSS-in-JS libraries that allow you to write CSS styles directly in your JavaScript code. CSS-in-JS libraries provide a way to create dynamic styles, share styles between components, and improve performance by reducing the number of HTTP requests. There are several popular CSS-in-JS libraries available, such as styled-components, emotion, @emotion/styled, or the inline style prop.

```jsx
export default function Component() {
  return (
    <button style={{
      backgroundColor: '#007bff',
      color: '#fff',
      borderRadius: '4px',
    }}>
      Click Me
    </button>
  );
}
```

Code 4.11: Inline Style Prop

Code 4.11 shows an example of using the inline style prop to apply styles directly to a React component. The **style** prop accepts an object with CSS properties and values to style the component.

4.5.4 Tailwind CSS

Tailwind CSS is a utility-first CSS framework that provides a set of utility classes to style web designs. Tailwind CSS allows you to create custom designs by applying utility classes directly in the HTML markup, making it easier to build responsive and customizable web designs.

```
<button class="bg-blue-500 text-white font-bold py-2 px-4 rounded">
  Click Me
</button>
```

Code 4.12: Tailwind CSS Utility Classes

4.5.5 More about Styling in NextJS

You can find more information about styling in NextJS from the following sources:

- NextJS Documentation: Built-in CSS Support
- NextJS Documentation: CSS Modules
- NextJS Documentation: External Stylesheets
- NextJS Documentation: CSS-in-JS
- Tailwind CSS Documentation

Code 4.12 shows an example of using Tailwind CSS utility classes to style a button component. The utility classes apply background color, text color, font weight, padding, and border radius styles to the button.

4.6 Routing

4.6.1 Link Component

The **Link** component in NextJS is used to navigate between pages in the application. It is an optimized component that pre-fetches linked pages in the background to improve performance and provide a seamless navigation experience.

```
import Link from 'next/link';

export default function Page() {
  return (
    <div>
      <h1>Home</h1>
      <Link href="/about">
        About
      </Link>
    </div>
  );
}
```

Code 4.13: Link Component

Code 4.13 shows an example of using the **Link** component in NextJS to create a link to the about page. The **Link** component accepts the **href** prop with the path to the linked page.

4.6.2 Dynamic Routes

A Dynamic Segment can be created by wrapping a folder's name in square brackets: [folderName]. For example, [id] or [slug].

Dynamic Segments are passed as the params prop to **layout**, **page**, **route**, and **generateMetadata** functions.

```
app/
├── blog/
│   └── [slug]
│       └── page.js
```

```
export default async function Page({ params }) {
  const slug = (await params).slug;
  return (<div>My Post: {slug}</div>);
}
```

Code 4.14: Dynamic Routes

Code 4.14 shows an example of using dynamic routes in NextJS. The **[slug]/page.js** file accepts the **params** prop and extracts the **slug** value to render the post content.

4.6.3 Nested Routes

Nested routes can be created by nesting folders inside the pages directory. Each folder represents a nested route, and each file inside the folder represents a page in the nested route.

```
pages/
├── blog/
│   └── [slug]
│   │   └── page.js
│   └── page.js
```

The example above shows the structure of nested routes in NextJS. The **blog/[slug]/page.js** file represents a dynamic post page, and the **blog/page.js** file represents the blog index page.

4.6.4 More about Routing in NextJS

You can find more information about routing in NextJS from the following sources:

- NextJS Documentation: Routing

4.7 State Management

4.7.1 Local State

Local state in NextJS can be managed using React's built-in **useState** hook. You can define state variables and update them using the **useState** hook in functional components.

```
import { useState } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

Code 4.15: Local State

Code 4.15 shows an example of managing local state in NextJS using the **useState** hook. The **Counter** component defines a **count** state variable and an **increment** function to update the count value.

4.7.2 Global State

Global state in NextJS can be managed using external state management libraries such as Redux, MobX, React Context API, Recoil, Zustand, or SWR. These libraries provide a way to share state across components and manage complex state interactions in the application.

4.8 Hooks

There are several built-in hooks in NextJS that allow you to add functionality to your components. Some of the commonly used hooks in NextJS are **useState** and **useEffect**.

4.8.1 useState

The **useState** hook is used to add state to functional components in NextJS. It returns a state variable and a function to update the state, allowing you to manage local state in functional components.

```
import { useState } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

Code 4.16: useState Hook

Code 4.16 shows an example of using the **useState** hook in NextJS to manage local state in a functional component. The **Counter** component defines a **count** state variable and an **increment** function to update the count value.

4.8.2 useEffect

The **useEffect** hook is used to add side effects to functional components in NextJS. It allows you to perform side effects such as fetching data, subscribing to events, or updating the DOM in functional components.

```
import { useState, useEffect } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0);


  useEffect(() => {
    console.log(`Count: ${count}`);
  }, [count]);

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

Code 4.17: useEffect Hook

Code 4.17 shows an example of using the **useEffect** hook in NextJS to add a side effect to a functional component. The **Counter** component logs the count value to the console when the count state changes.

4.9 More about NextJS

You can find more information about NextJS from the following sources:

- NextJS Documentation
- NextJS Learn

A. Books

-

B. Other Sources

-