Day 1 - Introduction to HTML
- What is HTML?
- Basic HTML structure
- HTML tags and attributes
- Creating headings, paragraphs, and lists

Day 2 - More HTML tags

Day 3 - Introduction to CSS

Day 4 - CSS layout
- Box model
- Display properties
- Positioning elements
- Float and clear

Day 5 - More CSS styling
- Fonts and typography
- Colors and backgrounds
- Borders and shadows
- Responsive design

Day 6 - Introduction to JavaScript
- What is JavaScript?
- Basic JavaScript structure

- Variables, data types, and operators
- Control structures (if, else, switch)

Day 7 - More JavaScript concepts
- Loops (for, while)
- Functions
- Arrays and objects
- DOM manipulation

Day 8 - More DOM manipulation
- Creating and deleting elements
- Modifying element attributes
- Responding to events (click, hover)

Day 9 - JavaScript in the browser
- JavaScript in the head and body
- Loading external scripts
- Using libraries (jQuery)

Day 10 - Putting it all together: HTML, CSS, and JavaScript
- Creating a simple website from scratch
- Adding styles and layout with CSS
- Adding interactivity with JavaScript

Day 11 - Advanced CSS techniques
- Flexbox
- Grid layout
- Animations and transitions

Day 12 - Advanced JavaScript concepts
- Scope and closures
- Higher-order functions
- Asynchronous programming (callbacks, promises)

Day 13 - More advanced JavaScript
- Working with APIs
- Fetching data with AJAX
- Displaying data with templates

Day 14 - Project work day
- Students work on a project using HTML, CSS, and JavaScript
- Instructor provides guidance and feedback

Day 15 - Project presentations and wrap-up
- Students present their projects to the class
- Wrap-up discussion of key concepts and skills learned
- Opportunities for further learning and exploration.

Day 1: Introduction to HTML.

- What is web design?

Web design is the process of creating and designing websites. It involves different skills and disciplines, including graphic design, user experience (UX) design, interface design, and search engine optimization (SEO). The goal of web design is to create a visually appealing, user-friendly, and functional website that meets the needs and expectations of its target audience.

- Importance of web design

The importance of web design lies in the fact that it directly affects how users perceive and interact with a website. A well-designed website can attract and retain users, while a poorly designed one can lead to frustration and abandonment. Additionally, a website's design can impact its search engine ranking and overall credibility. A good design takes into account the website's purpose, target audience, user experience, and visual appeal, among other factors. Overall, web design plays a critical role in creating a successful and effective online presence for individuals and businesses alike.

- Overview of HTML, CSS, and JavaScript

HTML, CSS, and JavaScript are the three core technologies used for building websites and web applications.

HTML (Hypertext Markup Language) is the markup language used for creating the structure and content of web pages. It provides a way to describe the elements on a web page, such as headings, paragraphs, images, and links.

CSS (Cascading Style Sheets) is used for styling and layout of web pages. It allows web developers to control the visual appearance of the content created with HTML. With CSS, you can change the font, color, size, and positioning of text and other HTML elements.

JavaScript is a programming language that is used for creating dynamic and interactive web pages. It allows web developers to add interactivity to web pages, such as creating forms, validating input, and responding to user events like clicks and scrolls.

Together, these three technologies form the foundation for modern web design, and knowledge of all three is essential for creating engaging and effective websites and web applications.

Day 1 - Introduction to HTML

- What is HTML?

HTML stands for HyperText Markup Language. It is a standard markup language used for creating web pages and applications that can be displayed on the internet. HTML is used to structure content and add meaning to it, such as headings, paragraphs, links, and images. It provides a way to describe the structure of a web page using tags and attributes that allow the browser to render the page in a visually appealing and functional way. HTML is one of the core technologies used to build the internet, along with CSS and JavaScript.

- Basic HTML structure

The basic structure of an HTML document consists of several elements that define the structure and content of a web page.

Here is an example of a basic HTML structure:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Heading 1</h1>
    <p>Paragraph 1</p>
```

```
  </body>
</html>
```

- HTML tags and attributes

HTML (Hypertext Markup Language) uses tags to structure content and give meaning to web pages. Tags are surrounded by angle brackets (< and >), and can have attributes that provide additional information about the tag.

Here's an example of a simple HTML document with tags and attributes:

````
```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page</title>
<meta
  </head>
  <body>
    <h1>This is the heading</h1>
    <p>This is a paragraph.</p>
    <img src="image.jpg" alt="A beautiful image">
    <a href="https://www.example.com">This is a link</a>
  </body>
</html>
````
````

In this example, we have:

- `**<!DOCTYPE html>**`: This declares the document type as HTML5.
- `**<html>**`: This is the root element of the document, and contains all the other elements.
- `**<head>**`: This section contains metadata about the document, such as the title of the page.
- `**<title>**`: This specifies the title of the document, which appears in the browser's title bar.
- `**<body>**`: This contains the visible content of the document.
- `**<h1>**`: This is a heading tag, which is used to indicate the main heading of the page.
- `**<p>**`: This is a paragraph tag, used to structure text into paragraphs.
- `**<img>**`: This is an image tag, used to display an image on the page. The `src` attribute specifies the URL of the image, and the `alt` attribute provides alternative text in case the image cannot be displayed.
- `**<a>**`: This is an anchor tag, used to create links to other pages or resources. The `href` attribute specifies the URL of the page or resource being linked to.

- Creating headings, paragraphs, and lists

In HTML, we can create various types of content such as headings, paragraphs, and lists using specific tags.

Headings:
HTML provides six levels of headings from H1 to H6. We can use these headings to define the hierarchy of content on a web page. For example, H1 is usually used for the main heading of a page, while H2 can be used for subheadings, and so on.

Example:

```
```

```
<h1>This is the main heading</h1>
<h2>This is a subheading</h2>
```

Paragraphs:
To create paragraphs in HTML, we use the `<p>` tag. We can use multiple `<p>` tags to create multiple paragraphs.

Example:

```
<p>This is the first paragraph</p>
<p>This is the second paragraph</p>
```

Lists:
There are two types of lists in HTML - ordered and unordered. Ordered lists are used to create numbered lists, while unordered lists are used to create bullet points.

Example:

```
<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>
```

Day 2 - More HTML tags

- Links and anchors

Links and anchors are an essential part of HTML. Links are used to navigate from one web page to another or to another section of the same page.

To create a link in HTML, you use the anchor tag `<a>`. Here's an example:

```
<a href="https://www.example.com">Click here to visit Example.com</a>
```

In this example, the `href` attribute specifies the URL that the link goes to. When a user clicks on the link, they will be taken to the Example.com website.

You can also use relative links to link to other pages on your own website. For example:

```
```

```
<a href="about.html">Click here to visit the About page</a>
```

In this example, `about.html` is the file name of the About page on your website.

You can also add additional attributes to the anchor tag, such as `target="_blank"` to open the link in a new window, or `title="Description of link"` to add a tooltip when a user hovers over the link.

Overall, using links and anchors is an important part of creating a well-functioning website that is easy for users to navigate.

- Images and attributes

Images are an important element in web design as they add visual interest and help to break up text. In HTML, you can add images using the <img> tag.

The <img> tag has several attributes that allow you to specify the source of the image, its dimensions, and alternative text. Here are some common attributes:

- src: specifies the URL of the image
- alt: specifies alternative text that is displayed if the image cannot be loaded or for accessibility purposes
- width and height: specify the dimensions of the image in pixels
- title: specifies additional information about the image that can be displayed as a tooltip

Here's an example of how to add an image to an HTML document:

```
```

```
<img url="https://aviationweek.com/sites/default/files/styles/crop_freeform/public/2021-12/PHOTO2021-BEST-Aldo_Wicki.jpg?itok=sLkNUOuT" alt="Example Image" width="400" height="300">
```

In this example, the image file "example.jpg" is displayed with the alternative text "Example Image" and dimensions of 400 pixels by 300 pixels**.**

-Video Tag in Html

The `<video>` tag in HTML is used to embed a video or multimedia content in a web page. It provides a container for displaying video content and supports various attributes and options for customization.

Here is the basic structure of the `<video>` tag:

```html
<video co>
  <!-- Video sources -->
  <source src="path/to/video.mp4" type="video/mp4">
  <source src="path/to/video.webm" type="video/webm">

  <!-- Fallback content -->
  Your browser does not support the video tag.
</video>
```

Let's break down the elements within the `<video>` tag:

- `<source>`: This is used to specify the video sources, defined by the `src` attribute. It allows providing multiple `<source>` elements with different file formats (`mp4`, `webm`, etc.) to ensure cross-browser compatibility. The `type` attribute indicates the MIME type of the video.

- Fallback content: The text "Your browser does not support the video tag." is included within the `<video>` tags. It serves as fallback content that will be displayed if the browser does not support the `<video>` tag or any of the specified video sources.

- Other attributes: The `<video>` tag supports additional attributes such as `controls`, `autoplay`, `loop`, `muted`, etc., which allow controlling the video playback and appearance. These attributes can be used based on your specific requirements.

The `<video>` tag provides a flexible and powerful way to embed and control video content within a web page.

- Forms and form elements

Forms are a crucial part of web development, allowing users to input data and interact with a website. HTML provides a variety of form elements to facilitate this interaction, including text input fields, checkboxes, radio buttons, and dropdown menus.

HTML form elements consist of the `<form>` tag and its various child elements such as `<input>`, `<textarea>`, and `<select>`. The `<form>` tag contains various attributes such as `action`, `method`, and `target` which determine where the form data is submitted, how it's submitted, and where the response is displayed.

CSS can be used to style form elements, including changing the font, color, and layout of form fields, and providing visual feedback to users as they interact with the form. JavaScript can also be used to

enhance form functionality, such as validating input, displaying error messages, and dynamically updating form content based on user actions.

As a Freelancer, Web Designer, and Web Master with skills in HTML, CSS, and JavaScript, you have the ability to create responsive and visually appealing forms that offer a seamless user experience. Your knowledge of other technologies such as Python, Android Application Development, NodeJS, React, and Angular only enhances your ability to create dynamic and engaging web applications.

Day 3 - Introduction to CSS

- What is CSS?

CSS stands for Cascading Style Sheets, and it is a style sheet language used to describe the presentation and layout of HTML documents. CSS allows web designers to separate the presentation of a document from its structure, making it easier to create visually appealing and consistent websites. With CSS, designers can control the colors, fonts, spacing, and other visual aspects of a webpage.

- Basic CSS structure

CSS (Cascading Style Sheets) is a language used to describe the presentation of a web document written in HTML or XML. It provides the style of the webpage by defining colors, fonts, layout, and other visual effects.

The basic structure of a CSS code block includes a selector, followed by a declaration block enclosed in curly braces. The declaration block contains one or more property-value pairs, separated by a colon and terminated by a semicolon.

Here is an example of basic CSS structure:

```CSS
```

```
/* Selector */
h1 {
  /* Declaration Block */
  color: blue;
  font-size: 24px;
  text-align: center;
}
```

In the example above, `h1` is the selector and defines which HTML element to apply the styles to. The declaration block contains the style properties and values that will be applied to the selected HTML element. In this case, the styles will set the color of the text to blue, the font size to 24 pixels, and the text alignment to center.

It is important to note that CSS can be written in an external style sheet file or included within the HTML document using the `<style>` tag.

- CSS selectors and properties

CSS selectors are patterns that are used to select and apply styles to specific HTML elements. There are several types of selectors such as element selectors, class selectors, ID selectors, attribute selectors, and pseudo-selectors.

CSS properties, on the other hand, are used to specify the visual style of an HTML element, such as its color, font-size, width, height, and margin. These properties can be applied to HTML elements using CSS selectors.

For example, the following CSS code selects all the paragraphs on a page and sets their font-size to 16 pixels:

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

In this example, `p` is the selector, and `font-size` is the property. The value of the `font-size` property is set to `16px`.

Selectors and properties can be combined to create more specific and targeted styling. For example, the following CSS code selects all elements with the class name "my-class" and sets their background color to blue**:**

```css
.container{
  background-color: blue;
}
```

In this example, `.my-class` is the selector, and `background-color` is the property. The value of the `background-color` property is set to `blue`.

- Applying CSS to HTML documents

To apply CSS to HTML documents, you can use one of the following methods:

1. Inline CSS: You can add style attributes directly to HTML elements using the style attribute. For example:

```
```
<h1 style="color: red;">Hello World!</h1>
```
```

2. Internal CSS: You can add CSS code within the `<style>` tag in the head section of the HTML document. For example:

```
```
<head>
  <style>
    h1 {
      color: red;
    }
  </style>
</head>
<body>
  <h1>Hello World!</h1>
</body>
```
```

3. External CSS: You can create a separate CSS file with a .css extension and link it to the HTML document using the `<link>` tag in the head section of the HTML document. For example:

```
<head>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <h1>Hello World!</h1>
</body>
```

The `href` attribute should contain the path to the CSS file. The CSS file should contain CSS code that will be applied to the HTML document.

Day 4 - CSS layout

- Box model

The box model is a concept in CSS that defines how elements on a webpage are rendered as rectangular boxes. The box model consists of four components: content, padding, border, and margin.

The content is the actual content of the box, such as text, images, or other HTML elements. The padding is the space between the content and the border. The border is the line that surrounds the content and padding. The margin is the space between the border and the next element on the page.

Understanding the box model is important for creating layouts in CSS, as it allows you to control the spacing and positioning of elements on the page. By adjusting the padding, border, and margin, you can create spacing between elements, and control the distance between elements and the edges of the browser window.

Here is an example of the box model in action:

```
.box {
  width: 200px;
  height: 200px;
  border: 1px solid black;
  padding: 20px;
  margin: 10px;
}
```

In this example, we have created a box with a width and height of 200 pixels, a border of 1 pixel solid black, and 20 pixels of padding. We have also added a margin of 10 pixels around the box.

By adjusting the values of the padding, border, and margin, we can create different spacing and positioning effects on the page.

- Display properties

CSS display property is used to control the layout and rendering of an element on a webpage. The display property can take various values such as block, inline, inline-block, flex, grid, table, etc. Here are some examples of how to use the display property:

1. Block display: The block value of the display property will make an element display as a block-level element, which means that it will take up the full width available to it and any following elements will be displayed on the next line.

2. Inline display: The inline value of the display property will make an element display as an inline-level element, which means that it will only take up as much width as necessary and any following elements will be displayed on the same line.

```css
span {
  display: inline;
}
```

3. Inline-block display: The inline-block value of the display property will make an element display as an inline-level element that behaves like a block-level element. This means that it will take up only as much width as necessary but also allow for other elements to be displayed on the same line.

```css
button {
  display: inline-block;
}
```

4. Flex display: The flex value of the display property will create a flexible container for an element that allows for easy manipulation of the layout and positioning of its child elements.

```css
.container {
  display: flex;
}
```

5. Grid display: The grid value of the display property will create a grid container for an element that allows for easy creation of complex layouts with rows and columns.

```css
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr;
}
```

- Positioning elements

In CSS, we can position elements using the `position` property. There are several values for this property:

1. `static`: This is the default value, and it means the element will be positioned according to the normal document flow.

2. `relative`: This value positions the element relative to its normal position. We can then use the `top`, `right`, `bottom`, and `left` properties to offset the element from its original position.

3. `absolute`: This value positions the element relative to its closest positioned ancestor. If there is no positioned ancestor, it will be positioned relative to the document body. We can also use the `top`, `right`, `bottom`, and `left` properties to offset the element.

4. `fixed`: This value positions the element relative to the viewport, meaning it will stay in the same position even when the user scrolls the page. We can also use the `top`, `right`, `bottom`, and `left` properties to offset the element.

Here's an example of using the `position` property to position an element:

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

```css
.container {
  position: relative;
  height: 200px;
  width: 200px;
  background-color: #ccc;
}

.box {
  position: absolute;
  top: 50%;
```

```
  left: 50%;
  transform: translate(-50%, -50%);
  height: 50px;
  width: 50px;
  background-color: blue;
}
```

In this example, we have a container element with a gray background color. Inside the container, we have a box element that is positioned absolutely. We use the `top` and `left` properties to center the box element within the container, and we also use the `transform` property to center it vertically. The result is a blue square in the center of the gray container.

- Float and clear

In CSS, the `float` property is used to specify how an element should float in relation to its container. The `clear` property is used to specify which sides of an element should not be adjacent to any floated elements.

When an element is floated, it is taken out of the normal flow of the document and positioned to the left or right of its container. This allows other elements to flow around it.

Here's an example of how to use the `float` property to float an image to the left of some text:

```
<img src="example.jpg" style="float: left;">
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam fringilla augue nibh, eget pharetra lectus rutrum nec. Donec ac tortor vitae leo euismod mattis. Duis eu elit elit. Sed
```

```
commodo, velit in consequat malesuada, est nulla bibendum justo, id dictum libero massa id ipsum.</p>
```

In this example, the `float` property is set to `left` on the `img` element. This causes the image to float to the left of the paragraph element.

However, if there are other elements in the container that are also floated, they may end up adjacent to each other and cause layout issues. In this case, the `clear` property can be used to specify which sides of an element should not be adjacent to any floated elements.

For example, to ensure that an element is not adjacent to any floated elements on its left side, you can set the `clear` property to `left`. Here's an example:

```
<div style="clear: left;">
  <p>This element will not be adjacent to any floated elements on its left side.</p>
</div>
```

In this example, the `clear` property is set to `left` on the `div` element, which ensures that it is not adjacent to any floated elements on its left side.

Day 5 - More CSS styling

- Fonts and typography

Fonts and typography are an important aspect of web design, as they can greatly affect the readability and overall look of a website. CSS provides a number of ways to control the fonts and typography used on a webpage.

Here's an example of how to use CSS to style fonts and typography:

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Font and Typography Example</title>
    <style>
      /* Set the font family and font size for the entire page */
      body {
        font-family: Arial, sans-serif;
        font-size: 16px;
      }

      /* Set the font size and line height for headings */
      h1, h2, h3, h4, h5, h6 {
        font-size: 24px;
        line-height: 1.5;
      }

      /* Set the font weight for headings */
      h1, h2 {
        font-weight: bold;
```

```
    }

    /* Set the font style for paragraphs */
    p {
      font-style: italic;
    }

    /* Set the text color for links */
    a {
      color: #0000ff;
    }

    /* Set the text decoration for links */
    a:hover {
      text-decoration: underline;
    }
  </style>
 </head>
 <body>
  <h1>Welcome to our website</h1>
  <p>This is an example of how to style fonts and typography using CSS.</p>

  <h2>Headings</h2>
  <p>Headings should be used to provide structure to your content. They can also help with
SEO (search engine optimization).</p>
  <h3>Heading 3</h3>
  <h4>Heading 4</h4>
```

```
    <h5>Heading 5</h5>
    <h6>Heading 6</h6>

    <h2>Paragraphs</h2>
    <p>Paragraphs are used to provide additional information and context to your content. They should be used to break up long blocks of text and make your content more readable.</p>

    <h2>Links</h2>
    <p>Links are used to provide additional information or resources to your users. They should be used sparingly and only when necessary.</p>
    <p><a href="#">Click here</a> to learn more about our company.</p>
  </body>
</html>
```

In this example, we've used CSS to set the font family and font size for the entire page using the `body` selector. We've also set the font size and line height for headings using the `h1`, `h2`, `h3`, `h4`, `h5`, and `h6` selectors, as well as the font weight for the `h1` and `h2` headings.

We've set the font style for paragraphs using the `p` selector, and the text color and text decoration for links using the `a` selector and the `:hover` pseudo-class**.**

- Colors and backgrounds

Colors and backgrounds are an important aspect of CSS styling. They can be used to enhance the visual appeal of web pages and make them more user-friendly.

CSS provides several ways to define colors, including named colors, hexadecimal values, RGB values, and HSL values. For example, the following code sets the background color of a webpage to light blue:

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

In addition to setting background colors, CSS also provides ways to add background images and control their position, repeat, and size. For example, the following code sets a background image for the body of a webpage:

```css
body {
  background-image: url("image.png");
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
}
```

This code sets an image named "image.png" as the background image of the body element. The `background-repeat` property sets the image to not repeat, the `background-position` property centers the image on the page, and the `background-size` property covers the entire background area.

CSS also provides other properties for controlling the color and appearance of text, including font-family, font-size, line-height, text-align, and text-decoration. For example, the following code sets the font family, size, and color of text in a webpage:

```css
body {
  font-family: Arial, sans-serif;
  font-size: 16px;
  color: #333;
}
```

This code sets the font family to Arial or any sans-serif font, the font size to 16 pixels, and the color to a dark gray (`#333`). These are just a few examples of how CSS can be used to style the colors and backgrounds of web pages.

- Borders and shadows

Borders and shadows are important CSS properties that can be used to enhance the look and feel of web pages.

Borders are used to create a boundary around an HTML element. You can use the `border` property to specify the size, style, and color of the border. For example, the following code sets a red border of 2 pixels around a div element:

```
div {
```

```
  border: 2px solid red;
}
```

In this example, `2px` specifies the width of the border, `solid` specifies the style of the border, and `red` specifies the color of the border.

Shadows can be used to create depth and texture on an HTML element. You can use the `box-shadow` property to add shadows to an element. For example, the following code adds a gray shadow to a button element when it is hovered over:

```
button:hover {
  box-shadow: 2px 2px 5px gray;
}
```

In CSS, however, there are conditional selectors such as `:hover`, `:focus`, and `:active` which allow developers to style elements based on certain conditions such as user interaction.
In this example, `2px` specifies the horizontal offset of the shadow, `2px` specifies the vertical offset of the shadow, `5px` specifies the blur radius of the shadow, and `gray` specifies the color of the shadow.

- Responsive design

Responsive design is the practice of creating a website that adapts and changes its layout and appearance based on the size of the device it is being viewed on. This ensures that the website is

accessible and easy to use across a variety of devices, including desktop computers, tablets, and mobile phones.

There are several techniques used in CSS for creating responsive designs, including:

1. Media Queries: This technique involves defining styles based on the width of the screen or device. For example, you can define styles for screens that are 768 pixels wide or larger, and then apply those styles only when the screen is that size or larger. This allows you to create different styles for different screen sizes.

Example:

```
```
@media screen and (max-width: 768px) {
  body {
    font-size: 14px;
  }
}
@media screen and (min-width: 768px) and (max-width: 1024px) {
  body {
    font-size: 16px;
  }
}
@media screen and (min-width: 1024px) {
  body {
    font-size: 18px;
  }
}
```
```

```

2. Flexible Grids: This technique involves using a grid-based layout that adjusts based on the size of the screen or device. The columns of the grid are defined using percentages, which allows them to adjust based on the screen size.

**Example:**

```
.container {
  display: flex;
  flex-wrap: wrap;
}
.item {
  width: 100%;
  padding: 1rem;
}
@media (min-width: 768px) {
  .item {
    width: 50%;
  }
}
@media (min-width: 1024px) {
  .item {
    width: 33.3333%;
  }
}
```
```

```

3. Fluid Images: This technique involves using images that adjust their size based on the width of the container they are in. This ensures that images always fit within the available space, regardless of the size of the device.

**Example:**

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

4. Mobile-first design: This technique involves designing the website for mobile devices first and then gradually adding more styles and features for larger screens. This ensures that the website is optimized for mobile devices and performs well on larger screens as well.

**Example:**

```
body {
  font-size: 16px;
}
@media (min-width: 768px) {
  body {
```
```

```
    font-size: 18px;
  }
}
@media (min-width: 1024px) {
  body {
    font-size: 20px;
  }
}
```

Day 6 - Introduction to JavaScript

- What is JavaScript?

JavaScript is a high-level, interpreted programming language that is commonly used to add interactivity and dynamic effects to web pages. It can be used to manipulate HTML and CSS, create animations and visual effects, validate forms, perform calculations, and more. JavaScript code can be included directly in HTML files or in separate files with a .js extension, and it is executed by the user's web browser rather than a server.

- Basic JavaScript structure

Sure! Here's an example of the basic structure of a JavaScript program:

```javascript
// Define a function that takes two arguments and returns their sum
```

```
function addNumbers(num1, num2) {
  return num1 + num2;
}

// Call the function with two numbers and log the result to the console
let result = addNumbers(2, 3);
console.log(result);
```

In this example, we define a function called `addNumbers` that takes two arguments (`num1` and `num2`) and returns their sum using the `+` operator. We then call this function with the numbers `2` and `3` and store the result in a variable called `result`. Finally, we log the value of `result` to the console using the `console.log()` function.

- Variables, data types, and operators

I believe you meant to say "with examples" instead of "with css." Here is an explanation of variables, data types, and operators in JavaScript with examples:

Variables are used to store values in JavaScript. You can declare a variable using the `var`, `let`, or `const` keywords. Here is an example:

```javascript
var age = 20;
let name = "John";
const pi = 3.14;
```

In this example, `age`, `name`, and `pi` are variables. `age` is assigned the value `20`, `name` is assigned the string value `"John"`, and `pi` is assigned the numerical value `3.14`. `var` and `let` are used to declare variables whose values can change, while `const` is used to declare variables whose values are constant.

JavaScript has several data types, including strings, numbers, booleans, arrays, and objects. Here are some examples:

```javascript
let name = "John";
let age = 20;
let isStudent = true;
let hobbies = ["reading", "music", "sports"];
let person = {name: "John", age: 20};
```

In this example, `name` is a string, `age` is a number, `isStudent` is a boolean, `hobbies` is an array of strings, and `person` is an object with two properties, `name` and `age`.

JavaScript has several operators for performing arithmetic, comparison, logical, assignment operations and compound operations. Here are some examples:

```javascript
let x = 10;
let y = 5;
```

```
// Arithmetic operators
let sum = x + y; // 15
let difference = x - y; // 5
let product = x * y; // 50
let quotient = x / y; // 2

// Comparison operators
let isEqual = x == y; // false
let isNotEqual = x != y; // true
let isGreater = x > y; // true
let isLess = x < y; // false

// Logical operators
let and = x > 0 && y < 10; // true
let or = x > 0 || y < 10; // true
let not = !(x > y); // false

// Assignment operators
x += y; // equivalent to x = x + y;
x -= y; // equivalent to x = x - y;
x *= y; // equivalent to x = x * y;
x /= y; // equivalent to x = x / y;
```

After assignment operators, there are compound assignment operators which combine an arithmetic, bitwise, or logical operation with an assignment. These operators are shorter and more concise than writing out the full operation and assignment separately. Here are some examples:

```javascript
let x = 10;

// Compound assignment with addition
x += 5; // equivalent to x = x + 5;

// Compound assignment with subtraction
x -= 5; // equivalent to x = x - 5;

// Compound assignment with multiplication
x *= 5; // equivalent to x = x * 5;

// Compound assignment with division
x /= 5; // equivalent to x = x / 5;

// Compound assignment with remainder
x %= 5; // equivalent to x = x % 5;

// Compound assignment with bitwise AND
x &= 5; // equivalent to x = x & 5;

// Compound assignment with bitwise OR
x |= 5; // equivalent to x = x | 5;

// Compound assignment with bitwise XOR
x ^= 5; // equivalent to x = x ^ 5;
```

```
// Compound assignment with left shift
x <<= 5; // equivalent to x = x << 5;

// Compound assignment with right shift
x >>= 5; // equivalent to x = x >> 5;
```

Note that compound assignment operators can only be used with variables, and not with literals or constants.

- Control structures (if, else, switch)

Control structures in CSS? There are no control structures in CSS as it is a style sheet language used to describe the presentation of markup languages such as HTML.

Control structures are part of programming languages such as JavaScript, which allow developers to control the flow of their code based on certain conditions.

In JavaScript, `if`, `else`, and `switch` are control structures used for making decisions based on certain conditions.

`if` statements are used to execute a block of code if a certain condition is true:

```javascript
let x = 10;

if (x > 5) {
```

```
  console.log("x is greater than 5");
}
```

`else` statements are used to execute a block of code if the `if` condition is false:

```javascript
let x = 2;

if (x > 5) {
  console.log("x is greater than 5");
} else {
  console.log("x is less than or equal to 5");
}
```

`switch` statements are used to select one of many code blocks to be executed based on a given value:

```javascript
let day = "Monday";

switch (day) {
  case "Monday":
    console.log("Today is Monday");
    break;
  case "Tuesday":
    console.log("Today is Tuesday");
```

```
  break;
 default:
  console.log("Today is not Monday or Tuesday");
}
```

Day 7 - More JavaScript concepts

- Loops (for, while)

Loops are a fundamental concept in JavaScript that allow you to execute a block of code multiple times. There are two main types of loops in JavaScript: `for` and `while`.

The `for` loop is used when you know how many times you want to execute the loop, and it has the following syntax:

```
for (initialization; condition; increment) {
  // code to be executed
}
```

The `while` loop is used when you don't know how many times you want to execute the loop, and it has the following syntax:

```
while (condition) {
  // code to be executed
}
```

In both types of loops, the code inside the loop is executed repeatedly until the condition becomes false. The `for` loop is commonly used for iterating over arrays or performing a set number of iterations, while the `while` loop is used for looping until a specific condition is met.
Sure, here are some examples of loops in JavaScript:

1. For loop:

The for loop is used to iterate over a block of code a specific number of times.

Example:

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

Output:

```
0
1
```

```
2
3
4
```

2. While loop:

The while loop is used to execute a block of code as long as a condition is true.

Example:

```
let i = 0;

while (i < 5) {
  console.log(i);
  i++;
}
```

Output:

```
0
1
2
3
4
```

```

3. Do...while loop:

The do...while loop is similar to the while loop, but the block of code is executed at least once, even if the condition is false.

Example:

```
let i = 0;

do {
  console.log(i);
  i++;
} while (i < 5);
```

Output:

```
0
1
2
3
4
```
```

- Functions

Functions in JavaScript are a way to group and organize a block of code that can be reused throughout the program. Here's an example of a function that takes two parameters and returns their sum:

```
function addNumbers(num1, num2) {
  let sum = num1 + num2;
  return sum;
}

let result = addNumbers(2, 3); // calls the function with arguments 2 and 3 and stores the result in the variable 'result'
console.log(result); // prints 5
```

In this example, we define a function named `addNumbers` that takes two parameters, `num1` and `num2`. The function adds the two numbers together and stores the result in a variable named `sum`. Finally, the function returns the value of `sum`.

We then call the function with arguments `2` and `3`, and store the result in a variable named `result`. Finally, we log the value of `result` to the console, which should print `5`.

- Arrays and objects

Arrays and objects are data structures in JavaScript used to store and manipulate collections of values.

An array is a collection of values of any data type that are stored in a single variable. Here is an example of an array:

```
const fruits = ["apple", "banana", "orange"];
```

In this example, `fruits` is an array that contains three elements, each of which is a string. The elements of an array can be accessed using their index, which starts from 0. For example, `fruits[0]` would give us the value `"apple"`, `fruits[1]` would give us `"banana"`, and so on.

An object, on the other hand, is a collection of key-value pairs, where each key is a unique string that identifies a value. Here is an example of an object:

```
const person = {
  name: "John",
  age: 30,
  address: {
    street: "123 Main St",
    city: "Anytown",
    state: "CA"
  }
};
```

In this example, `person` is an object that has three properties: `name`, `age`, and `address`. The `address` property itself is an object with three properties: `street`, `city`, and `state`. The values of an object can be accessed using dot notation or bracket notation. For example, `person.name` would give us the value `"John"`, `person["age"]` would give us the value `30`, and `person.address.city` would give us the value `"Anytown"`.

- DOM manipulation

DOM manipulation refers to the process of using JavaScript to interact with the Document Object Model (DOM) of an HTML document. This involves accessing, modifying, and updating HTML elements and their attributes dynamically using JavaScript code.

For example, you can use DOM manipulation to:

- Change the content of an HTML element dynamically based on user input
- Add or remove HTML elements dynamically based on certain conditions
- Change the styles or classes of HTML elements based on user actions or events
- Retrieve data from HTML elements and store them in variables for further processing.

To manipulate the DOM, you can use various methods and properties provided by the DOM API, such as `getElementById()`, `querySelector()`, `appendChild()`, `setAttribute()`, `classList`, and many more.

Day 8 - More DOM manipulation

- Creating and deleting elements

To create and delete elements in DOM manipulation using JavaScript, you can use the following methods and techniques:

Creating Elements:
1. createElement: Use the `document.createElement()` method to create a new element. Pass the element tag name as the parameter.

```javascript
const newElement = document.createElement('div');
```

2. createTextNode: Use the `document.createTextNode()` method to create a text node to be inserted inside an element.

```javascript
const textNode = document.createTextNode('This is some text');
```

3. appendChild: Use the `appendChild()` method to append a child element or text node to a parent element.

```javascript
const parentElement = document.getElementById('parent');
parentElement.appendChild(newElement);
parentElement.appendChild(textNode);
```

Deleting Elements:
1. removeChild: Use the `removeChild()` method to remove a child element from its parent.

```javascript
const parentElement = document.getElementById('parent');
const childElement = document.getElementById('child');
parentElement.removeChild(childElement);
```

2. remove: Use the `remove()` method to remove an element directly without needing to access its parent.

```javascript
const elementToRemove = document.getElementById('element');
elementToRemove.remove();
```

It's important to note that when manipulating the DOM, you need to ensure that the elements you want to create or delete exist in the DOM tree. You can use methods like `getElementById()`, `querySelector()`, or other DOM selection methods to access the desired elements.

Here's an example that demonstrates creating and deleting elements:

```html
<!DOCTYPE html>
<html>
<head>
  <title>DOM Manipulation</title>
  <style>
    /* CSS Styles here */
  </style>
```

```
</head>
<body>
 <div id="parent">
  <button id="addButton">Add Element</button>
  <ul id="list">
   <li>Item 1</li>
   <li>Item 2</li>
   <li>Item 3</li>
  </ul>
 </div>

 <script>
  const addButton = document.getElementById('addButton');
  const list = document.getElementById('list');

  addButton.addEventListener('click', function() {
   // Create a new list item
   const newListItem = document.createElement('li');
   const listItemText = document.createTextNode('New Item');
   newListItem.appendChild(listItemText);

   // Append the new list item to the list
   list.appendChild(newListItem);
  });

  // Example of deleting an element when clicked
  list.addEventListener('click', function(event) {
```

```
    const clickedItem = event.target;
    if (clickedItem.tagName === 'LI') {
      // Remove the clicked list item
      clickedItem.remove();
    }
  });
 </script>
</body>
</html>
```

In this example, clicking the "Add Element" button creates a new list item and appends it to the existing list. Clicking on any list item deletes it from the list using the `remove()` method.

- Modifying element attributes

To modify element attributes in JavaScript using DOM manipulation, you can use the following methods:

1. getAttribute: Use the `getAttribute()` method to retrieve the value of an attribute for a specific element. Pass the attribute name as the parameter.

```javascript
const element = document.getElementById('myElement');
const attributeValue = element.getAttribute('attributeName');
```

2. setAttribute: Use the `setAttribute()` method to set the value of an attribute for a specific element. Pass the attribute name and the new value as parameters.

```javascript
const element = document.getElementById('myElement');
element.setAttribute('attributeName', 'attributeValue');
```

3. removeAttribute: Use the `removeAttribute()` method to remove a specific attribute from an element. Pass the attribute name as the parameter.

```javascript
const element = document.getElementById('myElement');
element.removeAttribute('attributeName');
```

4. className: Use the `className` property to modify the class attribute of an element. Assign a new class name as a string.

```javascript
const element = document.getElementById('myElement');
element.className = 'newClassName';
```

5. classList: Use the `classList` property to add, remove, or toggle CSS classes on an element. It provides methods like `add()`, `remove()`, and `toggle()`.

```
```

```javascript
const element = document.getElementById('myElement');
element.classList.add('className'); // Add a class
element.classList.remove('className'); // Remove a class
element.classList.toggle('className'); // Toggle a class
```

Here's an example that demonstrates modifying element attributes:

```html
<!DOCTYPE html>
<html>
<head>
  <title>Modify Element Attributes</title>
  <style>
    /* CSS Styles here */
  </style>
</head>
<body>
  <div id="myElement" class="originalClass" customAttribute="value">Hello, World!</div>

  <button onclick="modifyAttributes()">Modify Attributes</button>

  <script>
    function modifyAttributes() {
      const element = document.getElementById('myElement');
```

```
    // Retrieve attribute value
    const attributeValue = element.getAttribute('customAttribute');
    console.log('Attribute value:', attributeValue);

    // Set attribute value
    element.setAttribute('customAttribute', 'newValue');

    // Remove attribute
    element.removeAttribute('customAttribute');

    // Modify class name
    element.className = 'newClass';

    // Add and remove classes using classList
    element.classList.add('additionalClass');
    element.classList.remove('originalClass');
   }
  </script>
</body>
</html>
```

In this example, clicking the "Modify Attributes" button triggers the `modifyAttributes()` function. The function demonstrates retrieving the value of a custom attribute, setting a new value, removing an attribute, modifying the class name, and adding/removing classes using the `classList` property.

- Responding to events (click, hover)

To respond to events such as click and hover using JavaScript and DOM manipulation, you can use event listeners. Event listeners allow you to specify a function that should be executed when a specific event occurs on an element. Here are examples of responding to click and hover events:

1. Click Event:

```javascript
const element = document.getElementById('myElement');

// Add a click event listener
element.addEventListener('click', function() {
  // Code to execute when the element is clicked
  console.log('Element clicked!');
});
```

In this example, when the element with the id "myElement" is clicked, the provided function will be executed. You can replace the `console.log()` statement with any desired code or functionality.

2. Hover Event:

```javascript
const element = document.getElementById('myElement');

// Add a mouseover event listener for hover in
element.addEventListener('mouseover', function() {
```

```
  // Code to execute when the mouse hovers over the element
  console.log('Mouse over the element!');
});

// Add a mouseout event listener for hover out
element.addEventListener('mouseout', function() {
  // Code to execute when the mouse leaves the element
  console.log('Mouse left the element!');
});
```

In this example, the `mouseover` event is triggered when the mouse pointer enters the element, and the `mouseout` event is triggered when the mouse pointer leaves the element. You can replace the `console.log()` statements with your desired code or functionality.

You can attach event listeners to any HTML element using its corresponding DOM method, such as `getElementById()`, `querySelector()`, etc. The `addEventListener()` method allows you to specify the event type and the function to be executed when that event occurs.

Remember to place your JavaScript code either in the `<head>` section, inside a `<script>` tag, or just before the closing `</body>` tag.

Step 1: Install Java Development Kit (JDK)
1. Visit the Oracle website (https://www.oracle.com/java/technologies/javase-jdk11-downloads.html) and download the latest version of the JDK for Windows.
2. Run the installer and follow the on-screen instructions to complete the installation.
3. Set up the JAVA_HOME environment variable by following these steps:
   - Right-click on the "Start" button and select "System".
   - Click on "Advanced system settings" on the left-hand side.
   - In the System Properties window, click on the "Environment Variables" button.
   - In the Environment Variables window, under "System variables," click on "New".
   - Enter "JAVA_HOME" as the variable name and the path to your JDK installation directory as the variable value (e.g., C:\Program Files\Java\jdk-11).
   - Click "OK" to save the environment variable.

Step 2: Download and Install Apache Tomcat
1. Go to the Apache Tomcat website (https://tomcat.apache.org) and download the latest stable version of Apache Tomcat (e.g., Apache Tomcat 9).
2. Extract the downloaded archive to a directory of your choice (e.g., C:\apache-tomcat).
3. Open a command prompt and navigate to the Tomcat installation directory (e.g., C:\apache-tomcat\bin).
4. Run the command "startup.bat" to start the Tomcat server.

Step 3: Install Eclipse IDE for Java EE Developers
1. Go to the Eclipse website (https://www.eclipse.org/downloads/) and download the Eclipse IDE for Java EE Developers package.
2. Extract the downloaded archive to a directory of your choice (e.g., C:\eclipse).

Step 4: Configure Eclipse with Tomcat Server
1. Launch Eclipse by running the "eclipse.exe" file in the Eclipse installation directory.
2. In the Eclipse welcome screen, click on "Workbench" to enter the development environment.
3. Go to "Window" -> "Preferences" to open the Preferences dialog.
4. Expand "Server" and click on "Runtime Environments".
5. Click "Add" to select Apache Tomcat as the server runtime environment.
6. Browse to the Tomcat installation directory and click "Finish" to add it to Eclipse.

Step 5: Create a Dynamic Web Project
1. Go to "File" -> "New" -> "Project" to open the New Project wizard.
2. Expand "Web" and select "Dynamic Web Project".
3. Enter a project name and click "Next".
4. Choose the target runtime (Apache Tomcat) or create a new runtime configuration.
5. Configure the project settings and click "Finish" to create the project.

Step 6: Create and Run a JSP File
1. Right-click on the project in the Project Explorer and go to "New" -> "JSP File".
2. Enter a JSP file name and click "Finish".
3. Eclipse will generate a basic JSP template for you to start coding.
4. To run the JSP file on the configured Tomcat server:
   - Right-click on the JSP file and go to "Run As" -> "Run on Server".
   - Choose the configured server (Apache Tomcat) and click "Finish".
   - Eclipse will launch the server and deploy your JSP file.
   - The JSP file will open in a web browser, and you can see the output.

JSP (JavaServer Pages) and JavaScript are two different technologies used in web development. Here's a comparison between JSP and JavaScript:

JSP (JavaServer Pages):
- JSP is a technology used for server-side web development.
- It is based on Java and allows embedding Java code within HTML pages.
- JSP files are processed by the server to generate dynamic HTML content that is sent to the client's web browser.
- JSP is typically used in combination with Java servlets and other Java technologies to build web applications.
- It provides a way to separate the presentation logic (HTML) from the business logic (Java code).
- JSP allows the use of tags and expressions to execute Java code and dynamically generate HTML content.
- It provides features like session management, database connectivity, and accessing Java objects.

JavaScript:
- JavaScript is a client-side scripting language.
- It is primarily used for adding interactivity and dynamic behavior to web pages.
- JavaScript code is executed directly in the client's web browser, without the need for server-side processing.
- It enables manipulating the Document Object Model (DOM) of the web page, handling events, making AJAX requests, and more.
- JavaScript can be used to validate forms, create interactive elements, perform animations, and enhance the user experience.
- It is supported by all major web browsers and provides a wide range of built-in functions and objects for various tasks.
- JavaScript can also be used on the server-side (e.g., with Node.js), but its primary focus is client-side scripting.

Server-Side Scripting:
- Server-side scripting languages run on the server.
- The code is executed on the server before the response is sent to the client's browser.
- Server-side scripts generate dynamic content, handle data processing, and interact with databases or other server resources.
- The output of server-side scripts is usually HTML, CSS, or other markup languages that are sent to the client.
- Server-side scripting languages include PHP, Java (with technologies like JSP and Servlets), Python (with frameworks like Django), Ruby (with frameworks like Ruby on Rails), and more.
- Server-side scripting is typically used for tasks like server-side form validation, database interactions, generating dynamic web pages, handling user authentication, and performing server-side processing.

Client-Side Scripting:
- Client-side scripting languages run in the client's browser.
- The code is embedded within HTML documents or included as external files and is executed by the browser.
- Client-side scripts enhance interactivity, manipulate the Document Object Model (DOM), handle user events, and modify the appearance and behavior of web pages.
- Common tasks performed by client-side scripts include form validation, dynamic content updates, handling user interactions, and making AJAX requests to the server.
- JavaScript is the primary client-side scripting language supported by all major web browsers. Other languages like TypeScript and CoffeeScript can also be compiled to JavaScript.
- Client-side scripting provides a more responsive and interactive user experience without requiring frequent server round-trips.

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
  <title>My JSP Page</title>
</head>
<body>
  <h1>Welcome to my JSP Page!</h1>

  <%-- Java code embedded in JSP --%>
  <% String name = "John"; %>

  <%-- JSP expression to display variable value --%>
  <p>Hello, <%= name %>!</p>

  <%-- JSP scriptlet to perform server-side logic --%>
  <%
    int num1 = 5;
    int num2 = 10;
    int sum = num1 + num2;
  %>

  <%-- JSP expression to display the sum --%>
  <p>The sum of <%= num1 %> and <%= num2 %> is <%= sum %>.</p>

  <%-- JSP directive to include another JSP --%>
  <%@ include file="another.jsp" %>
```

```
  <%-- JSP declaration to define a method --%>
  <%!
    public void myMethod() {
      // Method logic goes here
    }
  %>

  <%-- JSP useBean action to create and use a JavaBean --%>
  <jsp:useBean id="myBean" class="com.example.MyBean" scope="session" />

  <%-- JSP scriptlet to invoke a method on the JavaBean --%>
  <% myBean.doSomething(); %>
</body>
</html>
```

1. `<%@ page %>` directive: Specifies page-level attributes such as language, content type, and encoding.
2. `<%= expression %>`: JSP expression to display the value of a variable or expression.
3. `<% Java code %>`: JSP scriptlet to embed Java code within the JSP page.
4. `<%-- Comment --%>`: JSP comment to add comments within the code.
5. `<%@ include file="filename" %>`: JSP directive to include the content of another JSP file.
6. `<%! Declaration %>`: JSP declaration to define variables, methods, or classes.
7. `<jsp:useBean>`: JSP action to create and use JavaBeans.
8. `<% Action %>`: JSP scriptlet to perform server-side logic or invoke methods.