Web Development Notes

Basic Essentials

What is Internet?

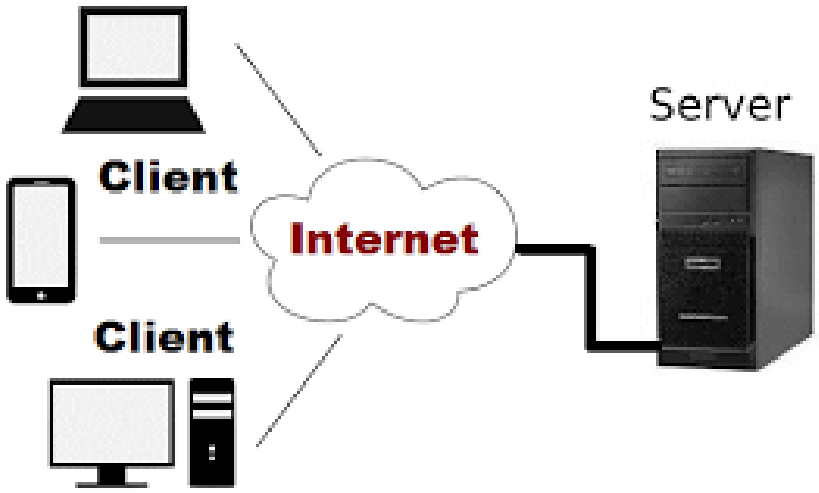
The internet is a global network of interconnected computers and devices that communicate using standardized protocols, enabling the exchange of data, information, and services worldwide.

How internet works?

The Internet works by connecting computers and devices worldwide through a network infrastructure. In web development, the client-server model refers to how information is exchanged between users' devices (clients) and remote computers (servers) that host websites or web applications.

Client: The user's device (such as a computer or smartphone) that requests and receives web content or services from servers.

Server: A remote computer that stores website data, processes requests from clients, and sends back the requested information or resources.



The client-server interaction in web development typically follows these steps:

- **Request**: The client sends an HTTP request to the server, specifying the resource (e.g., web page, image) it needs.

- **Processing**: The server processes the request, executes necessary scripts or accesses databases to generate the requested content.
- **Response**: The server sends an HTTP response back to the client, containing the requested data (e.g., HTML, JSON, images).
- **Delivery**: The client's web browser receives the response and renders the content for the user to interact with, completing the interaction.

What is Operating System?

An operating system (OS) is software that manages computer hardware and provides services for computer programs. It acts as an intermediary between users and hardware, controlling system resources, running applications, and facilitating communication between hardware components. **Examples:** Windows, Linux, macOS, Android etc…

What is Browser?

A browser is a software application that allows users to access and navigate the World Wide Web. It interprets HTML documents, displays web pages, and facilitates interaction with web content, such as clicking links, submitting forms, and running scripts. **Examples:** Chrome, Firefox, Opera, Edge, UC Browser, Safari etc…

What is IDE?

A software application that provides tools for writing, debugging, and testing code in a single interface, making software development more efficient and streamlined.

In short, an IDE is a comprehensive tool for coding, debugging, and building software.

Introduction to Web Development

What is web development?

Web development is the process of creating and maintaining websites or web applications. It involves various tasks such as web design, content creation, server-side scripting, client-side scripting, network security configuration, and more. Web development can range from creating simple static web pages to complex dynamic websites and web-based applications.

Categories:

There are typically two main categories of web development:

Front-end Development:
This involves creating the user interface and user experience of a website or web application. Front-end developers work with technologies such as HTML, CSS, and JavaScript to design and implement the visual and interactive elements that users see and interact with in their web browsers.

Back-end Development:
This focuses on the server-side of web development. Back-end developers work with server-side languages such as Python, Ruby, PHP, or Node.js, along with databases like MySQL, PostgreSQL, or MongoDB, to handle data storage, retrieval, and processing. They also manage the server configuration and ensure the proper functioning of the web application.

What is Database?

A collection of organized data that is stored in a way that allows for efficient retrieval and manipulation. A database is a structured repository of data that enables easy access, management, and updating of information.

HTML

What is HTML?

HTML stands for **Hypertext Markup Language.** It is the standard markup language used to create and structure content on web pages. It consists of elements and tags that define the structure and semantics of web documents. HTML is the foundation of web development and is used to format text, add images, create links, embed multimedia, and build the overall layout of a webpage.

Basic structure of HTML

The basic structure of an HTML document consists of the following elements:

1. **<! DOCTYPE html>**: Declaration that specifies the HTML version being used (HTML5 in this case).
2. **<html>**: The root element that wraps all content on the page.
    - **<head>**: Contains meta-information about the document, such as title, metadata, stylesheets, and scripts.
        - **<title>**: Sets the title of the web page displayed in the browser tab.
    - **<body>**: Contains the main content of the web page, including text, images, links, forms, and other elements.

    Here's an example of the basic structure:

```
<! DOCTYPE html>
<html>
        <head>
                <title>My Web Page</title>
        </head>

        <body>
                <h1 >Hello, World! </h1>
                <p> this is a basic HTML document. </p>
        </body>
</html>
```

Tag in HTML

A tag in HTML is a specific syntax used to define elements within an HTML document. Tags are enclosed in angle brackets (**<>**) and typically come in pairs: an opening tag and a closing tag. Opening tags indicate the beginning of an element, while closing tags signify the end of that element.

In HTML, there are several types of tags, each serving a specific purpose in structuring and formatting web content. Here are the main types of tags:

Opening and Closing Tags (Pair Tags):
These are paired tags used to define the start (**<tag>**) and end (**</tag>**) of an element. Examples include **<p>** for paragraphs, **<h1>** to **<h6>** for headings, **<ul>** and **<ol>** for unordered and ordered lists, ** ** for divisions, and **** for inline styling.

Self-Closing Tags:
Some tags do not require a closing tag because they don't enclose content. Instead, they self-close with a slash before the closing angle bracket. Examples include **
** for line breaks, **<img>** for images, **<input>** for form inputs, and **<meta>** for metadata.

Difference between tag and element

Tags are the specific syntax used in HTML to define elements, whereas **elements** refer to the entire structure created using these tags, including the opening and closing tags along with the content they enclose.

Headings in HTML

Headings in HTML are used to define the hierarchy and structure of textual content on a web page. HTML provides six levels of headings, from **<h1>** (most important) to **<h6>** (least important), with **<h1>** typically used for the main heading or title of a page.

Text Formatting

Text formatting refers to the process of styling and arranging text to enhance readability and visual appeal. In HTML, text formatting is achieved using various tags and attributes to modify the appearance of text elements.

1. **<Strong>**: Makes text bold for emphasis.
2. ****: Italicizes text for emphasis.
3. **<u>**: Underlines text.

4. **<del>** or **<strike>**: Strikes through text.
5. **<Sup>**: Renders text in superscript.
6. **<Sub>**: Renders text in subscript.
7. **<font>**: Sets text color, size, and font family (deprecated in HTML5; use CSS for styling).
8. **<b>**: Makes text bold (semantically less meaningful than ****).
9. **<i>**: Italicizes text (semantically less meaningful than ****).
10. **<Small>**: Reduces text size for smaller text content.
11. **<kbd>:** Represents keyboard input from the user.
12. **<mark>**: Highlights text for reference or to denote significance.
13. **<pre>**: Preformatted text with preserved spacing and line breaks.

What is Attribute?

Attributes in HTML provide additional information or properties to HTML elements, helping define their behavior, appearance, or functionality. Attributes are specified within the opening tag of an element and are written as name-value pairs, separated by an equal sign (=).

Examples: href, id, class, src, align, height, width etc…

Links in HTML

In HTML, links are created using the <a> (anchor) tag, allowing users to navigate to different web pages, resources, or sections within the same page. Here's an overview of how links are used in HTML:

External Link: In external link navigate from one page to other page.

Example: Visit Example.

Internal Link: In external link navigate from one section to other section by using id of that section.

Example: Jump to Section

Section content is here…….

Img Tag

The **<img>** tag in HTML is used to insert images into a web page. It is a self-closing tag, meaning it does not have a closing tag and is written with an optional **alt** attribute to provide alternative text for the image. Here's an example of how the **<img>** tag is used:

Example: <img src="image.jpg" alt="Description of the image" height=”50px” width=”50px” title=”This is title”>

src: Specifies the URL or file path of the image (**image.jpg** in this case).
alt: Provides alternative text that describes the image. This is important for accessibility purposes and is displayed if the image fails to load.

Additionally, you can add other attributes to the **<img>** tag for further customization, such as **width** and **height** to set the dimensions of the image, **title** for a tooltip when hovering over the image, and **style** for inline CSS styling.

List in HTML

A list in HTML is a structured way to organize and display information, such as items, terms, or categories.

There are main types of lists: ordered lists.

1. Ordered List
2. Unordered List
3. Description List
   There is another list called nested list this is also used but not considered list type.

Ordered List (ol): In ordered list data is arranged in numbered list. Used for lists where the order of items matters.

Syntax:

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

Output will be

1. Item1
2. Item2
3. Item3

The format can be change in ordered by **type** attribute list number to alphabets, roman counting and can be reversed by **reversed** attribute its order. Also define the starting point by **start** attribute in ol.

Unordered List (ul): In ordered list data is arranged in bullets list. Used for lists where the order of items is not matters.

Syntax:

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

Output will be

- Item1
- Item2
- Item3

The format can be change in ordered by **type** attribute list bullets to circle, disc, square but cannot be reversed and cannot be define the starting point.

Description List (dl): A description list (also known as a definition list) is a type of list that consists of terms and their corresponding descriptions or definitions. It is defined using the **<dl>, <dt>,** and **<dd>** elements.

<dl>: Defines the description list container.

<dt>: Defines a term or phrase.

<dd>: Defines the description or definition of the term.

Syntax:

```
<dl>
    <dt> Apple </dt>
        <dd> Apple is a fruit.  </dd>
    <dt> Potato </dt>
        <dd> Potato is a vegetable.
</dd>
</dl>
```

Output will be

Apple

Apple is a fruit.

Potato

Potato is a vegetable.

Nested List: Nested list is the list of any item of ordered/unordered. In short it is the sub list of any list and it can be many further creates lists.

Syntax:

```
<ul>
    <li>Web Development
        <ol>
            <li>front-end
                <ul>
                 <li>HTML</li>
                 <li>CSS</li>
                 <li>JavaScript</li>
                </ul>
             </li>
        <li>back-end</li>
      </ol>
    <li> App Development </li>
</ul>
```

Output will be

- Web Development
  1. Front-end
     - HTML
     - CSS
     - JavaScript
  2. Back-end
- App Development

Media Tags

Audio, Video and Img tags are known as media tags because by these tags we can show audio, video and any picture show on the web page. The audio and video tags are supported in HTML5. The src attribute is required for both tags for the file path.

Audio Tag (<audio>): Used to embed audio content, such as music or podcasts. Supports various audio formats, including MP3, WAV, and OGG.

Example: <audio> <source src=’audio.mp3’ controls loop muted autoplay > </audio>

Video Tag (<video>): Used to embed video content, such as movies or clips. Supports various video formats, including MP4, WebM, and OGG.

Example: <video> <source src='video.pm4' controls loop muted autoplay > </video>

Common Attributes:

controls: Displays audio/video controls, such as play, pause, and volume.

src: Specifies the URL of the audio/video file.

type: Specifies the MIME type of the audio/video file.

autoplay: Automatically starts playing the audio/video when the page loads.

loop: Loops the audio/video playback.

muted: Mutes the audio/video by default.

Iframe Tag:

The **<iframe>** tag in HTML is used to embed another HTML document or a resource (such as a webpage, image, or video) within an existing HTML document. This creates a nested browsing context, allowing the embedded content to be displayed within a frame on the parent page.

It is used to embedding third-party content (e.g., YouTube videos, social media feeds, google maps) on the webpage.

Example: **<**iframe src="(link)" width="800" height="600" frameborder="1" scrolling="yes"></iframe>

Table in HTML

The **<table>** tag in HTML is used to define a table, which is a structured representation of data in rows and columns. Here is a detailed overview of the <table> tag and its attributes:

Syntax:

```
<table border='1' width='80 %' align='center'>
<caption>Student Data </caption>
   <thead>
       <tr>
          <th> ID </th>
          <th> Name </th>
          <th> City </th>
       </tr>
    </thead>
    <tbody>
        <tr>
            <td> 1 </th>
             <th> Ali </th>
             <th> Lodhran </th>
        </tr>
        <tr>
           <td> 1 </th>
           <th> Ahmad </th>
           <th> Multan </th>
        </tr>
    </tbody>
</table>
```

Output will be

Student Data

ID	Name	City
1	Ali	Lodhran
2	Ahmad	Multan

Table Structure:

<table>: Defines the table container**.**

<tr>: Defines a table row.

<th>: Defines a table header cell (column title).

<td>: Defines a table data cell (contains the actual data).

<caption>: Defines a caption for the table.

<thead>: Defines the table header (contains the <tr> and <th> elements).

<tbody>: Defines the table body (contains the <tr> and <td> elements).

Attributes:

border: Specifies the border width of the table.

cellpadding: Specifies the space between the cell contents and the cell border.

cellspacing: Specifies the space between cells.

width: Specifies the width of the table.

height: Specifies the height of the table.

align: Specifies the alignment of the table (left, right, or center).

bgcolor: Specifies the background color of the table.

colspan: Specifies the number of columns the cell spans.

rowspan: Specifies the number of rows the cell spans.

align: Specifies the alignment of the cell contents (left, right, or center).

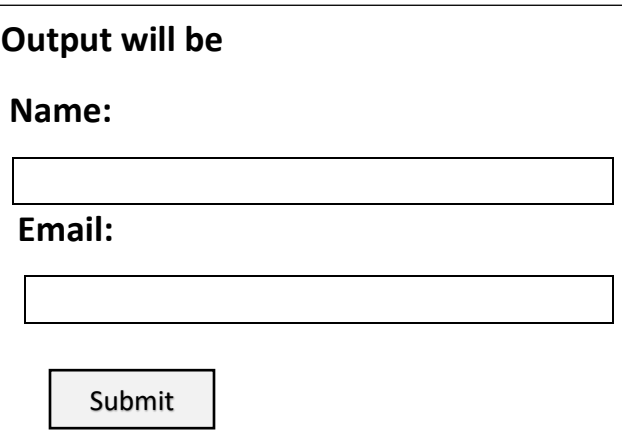
Form in HTML

The **<form>** tag in HTML is used to create a form for user input. It's a container element that holds various form elements like input fields, checkboxes, radio buttons, and buttons. Form main purpose to get data from the user and give response according to that data.

Syntax:

```
<form>
<label for="name"> Name : </label>
<input type="text" id="name"><br>
<label for="email">Email : </label>
<input type="email" id="email"><br>
<input type="submit" value="Submit">
</form>
```

Output will be



Form Elements:

1. **input:** Text input, checkbox, radio button, file upload, etc.
2. **textarea:** Multiline text input.
3. **select:** Dropdown list.
4. **label:** Associates a text label with a form element.
5. **button:** Submit button or reset button.
6. **fieldset:** Groups related form elements.
7. **legend:** Specifies a caption for a fieldset.

The main element is input, Here are all the input types in HTML, along with a short description:

1. **text:** Single-line text input.
2. **password:** Single-line text input, characters are masked.
3. **email:** Single-line text input, email address format.
4. **tel:** Single-line text input, telephone number format.
5. **number:** Number input, allows numeric values only.
6. **date:** Date input, allows date selection.
7. **time:** Time input, allows time selection.
8. **datetime-local:** Date and time input, allows date and time selection.
9. **month:** Month input, allows month selection.
10. **week:** Week input, allows week selection.
11. **url:** Single-line text input, URL format.
12. **search:** Single-line text input, search query format.
13. **color:** Color input, allows color selection.
14. **checkbox:** Checkboxes, allows multiple selections.
15. **radio:** Radio buttons, allows single selection.
16. **file:** File input, allows file uploads.
17. **hidden:** Hidden input, not visible to users.
18. **range:** Slider input, allows numeric value selection within a range.

19. **image:** Image input, allows image selection.

20. **reset:** Reset button, resets form values.

21. **submit:** Submit button, submits form data.

22. **button:** Button, allows custom button functionality.

Form Validation in HTML

HTML provides several attributes and elements for form validation, making it easier to ensure user input data meets specific requirements. Here are some key aspects of form validation in HTML:

1. **required:** Specifies that a field must be filled in.

2. **type:** Specifies the input data type (e.g., email, password, date).

3. **pattern:** Specifies a regular expression pattern for input validation.

4. **minlength:** Specifies the minimum input length.

5. **maxlength:** Specifies the maximum input length.

6. **min:** Specifies the minimum value for numerical inputs.

7. **max:** Specifies the maximum value for numerical inputs.

8. **validationMessage:** Specifies a custom error message for each field.

Example:

<input type="text" name="username" required pattern="[A-Za-z]{5,10}" minlength="5" maxlength="10" validationMessage="Please enter a valid username">

CSS

Introduction to CSS

CSS (Cascading Style Sheets) is a styling language used to control the layout and appearance of web pages written in HTML. It is used to separate the presentation of a document from its structure, allowing for more flexibility and control over the layout, visual styling, and user experience.

Some common uses of CSS include:

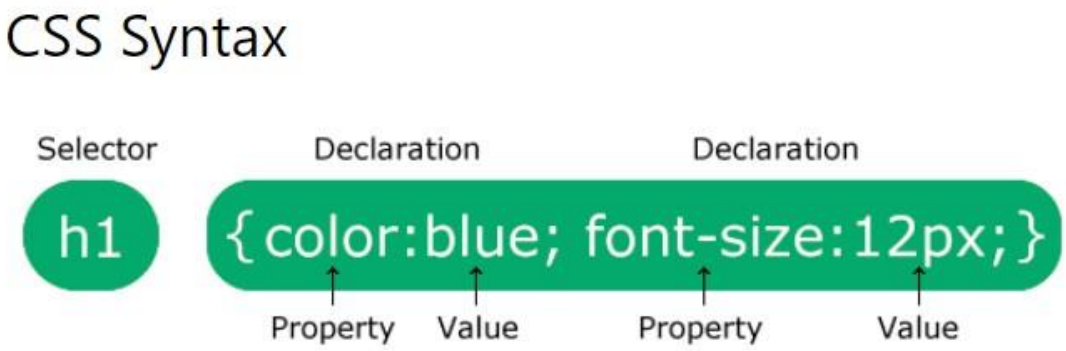
- Styling text and fonts
- Creating layouts and grid systems
- Designing visual effects and animations
- Customizing the appearance of HTML elements
- Creating responsive designs for mobile and tablet devices
- Implementing accessibility features, such as high contrast modes

There are several types of CSS, including:

1. **External CSS:** Written in a separate file and linked to the HTML document.

2. **Internal CSS:** Written directly in the HTML document, using the <style> tag.

3. **Inline CSS:** Written directly in the HTML element, using the style attribute.



Selectors

CSS selectors are used to target specific HTML elements on a web page and apply styles to them. Selectors are crucial for styling elements effectively and are one of the fundamental concepts in CSS. Here are some common CSS selectors:

1. Element Selectors:

- h1 {styles} (targets all h1 elements)
- p { styles } (targets all p elements)

2. Class Selectors:

- .className {styles} (targets all elements with the class "className")

3. ID Selectors:

- #idName {styles} (targets the element with the id "idName")

4. Attribute Selectors:

- [attribute] {styles} (targets elements with the specified attribute)
- [attribute=value] {styles} (targets elements with the specified attribute and value)

5. Pseudo-Classes:

- : hover {styles} (targets elements when they are hovered over)
- : active {styles} (targets elements when they are active)
- : focus {styles} (targets elements when they are focused)

6. Pseudo-Elements:

- ::before { styles } (targets the pseudo-element before the element)
- ::after { styles } (targets the pseudo-element after the element)

7. Combinators:

- element1 element2 {styles} (targets elements that are direct children of element1)
- element1 > element2 {styles} (targets elements that are direct children of element1)
- element1 + element2 {styles} (targets elements that are adjacent siblings of element1)

8. Universal Selector:

- * {styles} (targets all elements)

9. Grouping Selector:

- element1, element2 {styles} (targets multiple elements)

Typography

Typography in CSS refers to the control and styling of text elements on a web page. Here are some key aspects of typography in CSS:

1**. Font Family:** Sets the typeface or font family for text elements. Example: font-family: Arial, sans-serif;

2**. Font Size:** Sets the size of text elements. Example: font-size: 18px;

3**. Font Style:** Sets the style of text elements, such as italic or oblique. Example: font-style: italic;

4**. Font Weight**: Sets the weight or boldness of text elements. Example: font-weight: bold;

5. **Line Height:** Sets the height of each line of text. Example: line-height: 1.5;

6. **Text Alignment**: Sets the alignment of text elements, such as left, right, or center. Example: text-align: center;

7**. Text Decoration:** Sets the decoration of text elements, such as underline or strikethrough. Example: text-decoration: underline;

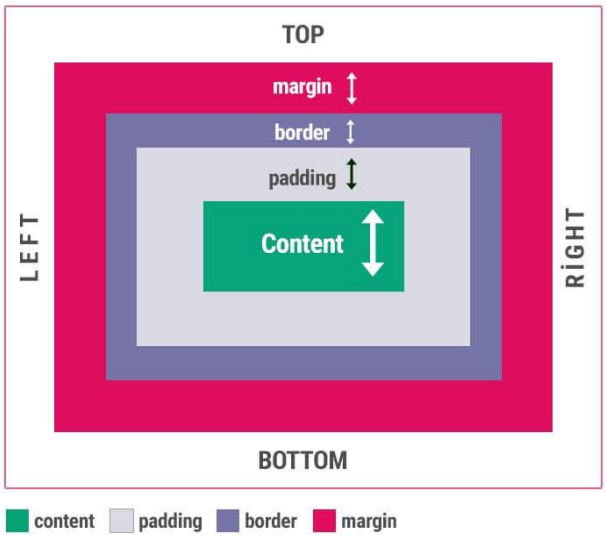
8. **Text Transform:** Sets the capitalization of text elements, such as uppercase or lowercase. Example: text-transform: uppercase;

9**. Letter Spacing:** Sets the spacing between letters in text elements. Example: letter-spacing: 2px;

10. **Word Spacing**: Sets the spacing between words in text elements. Example: word-spacing: 2px;

11. **Text Shadow**: Adds a shadow effect to text elements. Example: text-shadow: 2px 2px 4px #000000;

Box Modal



The box model is a fundamental concept in CSS that describes the layout of elements on a web page. It defines how elements are rendered in terms of their content area, padding, border, and margin. Understanding the box model is crucial for controlling the spacing, sizing, and layout of elements effectively.

The box model consists of the following components:

1. **Content**:
    a. The actual content of the element, such as text, images, or other media.
    b. The content area is determined by the width and height properties of the element.

2. **Padding**:
    a. The padding is the space between the content area and the element's border.
    b. Padding can be set using the **padding** property and its shorthand variants (**padding-top**, **padding-right**, **padding-bottom**, **padding-left**).

3. **Border**:
    a. The border is the line that surrounds the padding and content area of an element.
    b. Borders can have different styles (solid, dashed, dotted, etc.), widths, and colors.
    c. Borders are defined using the **border** property and its sub-properties (**border-width**, **border-style**, **border-color**).

4. **Margin**:
    a. The margin is the space outside the border of an element, creating separation between elements.
    b. Margins can be set using the **margin** property and its shorthand variants (**margin-top**, **margin-right**, **margin-bottom**, **margin-left**).

Box-Sizing:

c. The box-sizing property determines whether the padding and border are included in the element's width and height.

Width and Height:

d. The width and height properties set the size of the content area.

Colors and Backgrounds

Colors and backgrounds in CSS are used to enhance the visual appearance of web pages. Here are some key concepts:

Colors

- Color properties: color, background-color
- Color values: hex codes (e.g. #FF0000), RGB (e.g. rgb(255, 0, 0)), HSL (e.g. hsl(0, 100%, 50%))
- Color keywords (e.g. red, blue, green)

Backgrounds

- Background properties: background-color, background-image, background-repeat, background-position, background-size
- Background values: color values, image URLs (e.g. url('image.jpg')), repeat values (e.g. repeat-x, repeat-y, no-repeat), position values (e.g. center, top, bottom), size values (e.g. cover, contain)

Examples:

- h1 { color: #FF0000; } sets the text color of h1 elements to red
- body { background-color: #F2F2F2; } sets the background color of the body element to a light gray
- header { background-image: url('header.jpg'); background-repeat: no-repeat; background-position: center; } sets the background image of the header element to header.jpg, with no repeat and centered positioning

Gradients

- Linear gradients: linear-gradient(to bottom, #FF0000, #FFFFFF)
- Radial gradients: radial-gradient(circle, #FF0000, #FFFFFF)

Transparency

- Opacity: opacity: 0.5; sets the opacity of an element to 50%
- Transparent colors: rgba(255, 0, 0, 0.5) sets the color to red with 50% opacity

Display

The display property in CSS determines the display type of an element, which affects its layout and visibility. Here are the most common values:

1. **block:** The element takes up the full width of its parent container and starts on a new line.

2. **inline**: The element takes up only the space needed for its content and does not start on a new line.

3. **inline-block**: The element takes up only the space needed for its content, but starts on a new line.

4. **none**: The element is hidden and takes up no space.

5. **flex**: The element is displayed as a flexible container.

Display Flex

Display Flex in CSS is a powerful layout mode that allows you to create flexible and responsive layouts. It's commonly used for creating rows and columns, aligning items, and distributing space.

Here are some key aspects of Display Flex:

1. **Flex Container**: The element that contains the flex items. Set display: flex or display: inline-flex on this element.

2. **Flex Items**: The child elements of the flex container.

3. **Main Axis**: The primary axis of the flex container, which can be either horizontal (row) or vertical (column).

4. **Cross Axis**: The secondary axis, perpendicular to the main axis.

Properties:

1**. flex-direction**: Sets the main axis direction (row, row-reverse, column, column-reverse).

2**. justify-content**: Aligns items along the main axis (flex-start, flex-end, center, space-between, space-around).

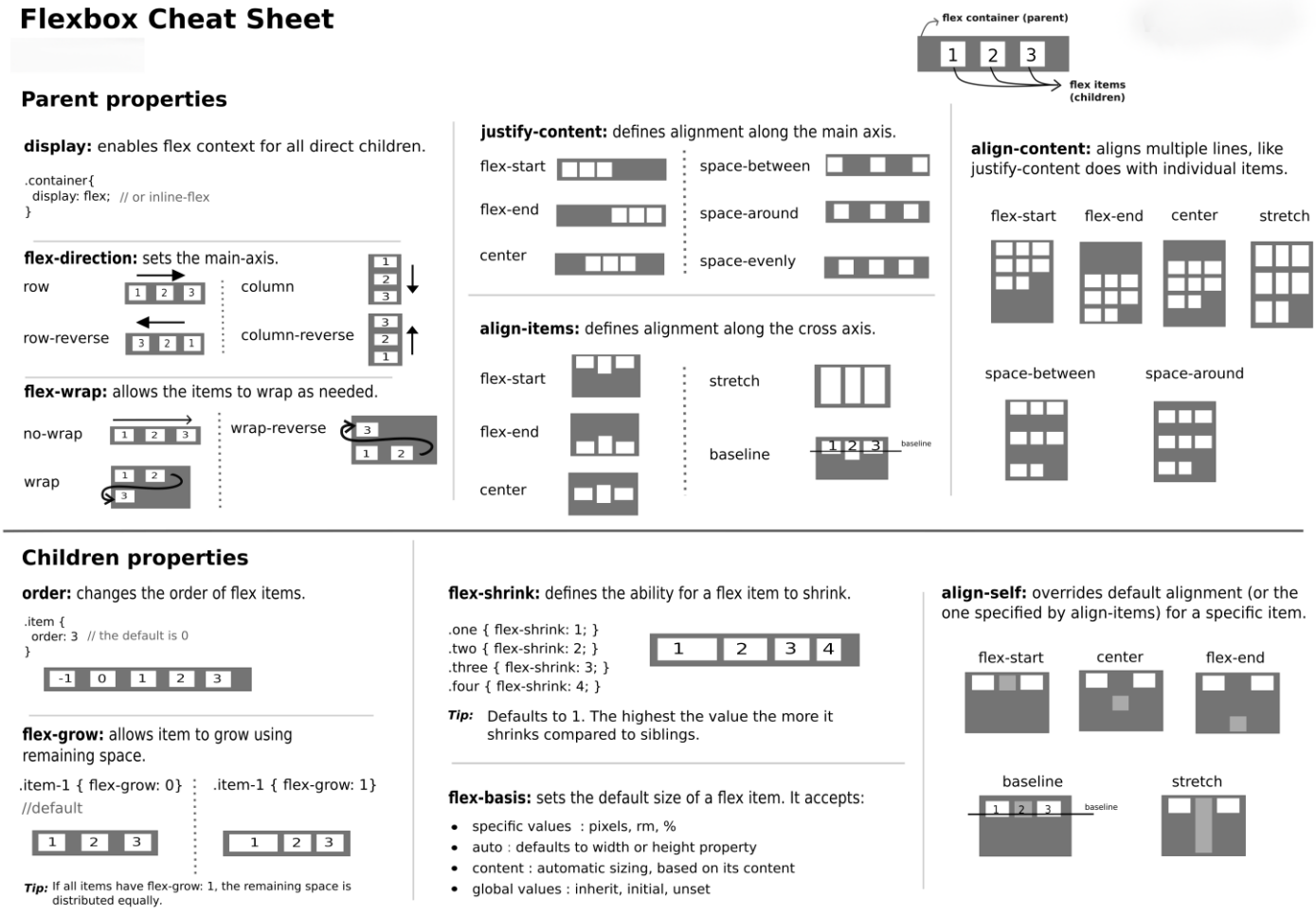
3**. align-items**: Aligns items along the cross axis (flex-start, flex-end, center, baseline, stretch).

4. **flex-wrap**: Controls whether items wrap to a new line or not (wrap, nowrap).

5**. flex-grow**: Sets the growth factor of an item (number).

6**. flex-shrink**: Sets the shrink factor of an item (number).

7. **flex-basis**: Sets the initial main size of an item (length, percentage).



Positions

Positions in CSS determine how an element is positioned within its parent element or the viewport. There are five main positions:

1. Static (default): The element is positioned according to the normal flow of the document.

2. Relative: The element is positioned relative to its original position within the normal flow.

3. Absolute: The element is removed from the normal flow and positioned relative to its nearest positioned ancestor or the viewport.

4. Fixed: The element is removed from the normal flow and positioned relative to the viewport, remaining fixed even when scrolling.

5. Sticky (experimental): The element is positioned relative to its nearest scrolling ancestor, "sticking" to a specified position when scrolling.

Properties used with positions:

- **top:** Sets the top position.
- **right:** Sets the right position.
- **bottom:** Sets the bottom position.
- **left:** Sets the left position.
- **z-index:** Sets the stacking order of overlapping elements.

Transition

Transition in CSS is a property that allows you to animate changes to an element's properties over a specified duration. It creates a smooth and gradual transition between two states, making the user experience more engaging and interactive.

The transition property is used to define the transition effect, and it takes three values:

1. **property:** The name of the property to be transitioned (e.g., opacity, width, color).

2. **duration:** The length of time the transition takes to complete (e.g., 0.5s, 1s, 2s).

3. **delay:** The delay time to start the transition (e.g., 0.5s, 1s, 2s).

4**. timing-function:** The speed curve of the transition (e.g., ease, linear, ease-in-out).

5**. transition**: shorthand property of transition in which all properties are defines.

Example:

```
.btn {
background-color: #333;
color: #fff;
transition: background-color 0.5s ease-in-out; /* Add transition effect */
}
.btn:hover {
background-color: #555; /* Change background color on hover */
}
```

Animation

Animations in CSS allow you to create dynamic and interactive effects by changing an element's properties over time. You can create animations using the @keyframes rule, which defines the animation's key frames (starting and ending points).

Basic syntax:

@keyframes animation-name { from { /* starting styles */ } to { /* ending styles */ } }

Or:

@keyframes animation-name { 0% { /* starting styles */ } 100% { /* ending styles */ } }

Properties:

1. **animation-name:** specifies the animation's name
2. **animation-duration:** sets the animation's length (e.g., 2s, 5s)
3. **animation-delay**: sets a delay before the animation starts (e.g., 1s, 3s)
4. **animation-iteration-count:** sets the number of times the animation repeats (e.g., 1, 2, infinite)
5. **animation-direction:** sets the animation's direction (e.g., normal, reverse, alternate)

Example:

```
element {
```

```
animation: fade-in 2s;  }`
`@keyframes fade-in {
    from {
    opacity: 0; }
    to {
    opacity: 1; }
    }
```

This creates a 2-second animation that fades in the element.

Other animation properties:

- **animation-fill-mode:** sets how the animation applies styles before/after the animation
- **animation-play-state**: sets the animation's play state (running, paused)
- **animation-timing-function**: sets the animation's timing function (e.g., ease, linear, cubic-bezier)

Transform

The transform property in CSS is used to apply transformations to an element, such as rotating, scaling, translating, or skewing. It allows you to change the position and appearance of an element without affecting its original size or shape.

Here are some common transformations:

1. **Rotate:** transform: rotate(45deg); rotates an element 45 degrees clockwise.

2. **Scale**: transform: scale(1.5); scales an element to 1.5 times its original size.

3. **Translate**: transform: translate(10px, 20px); moves an element 10 pixels to the right and 20 pixels down.

4. **Skew**: transform: skew(10deg); skews an element 10 degrees.

5. **Matrix**: transform: matrix(1, 2, 3, 4, 5, 6); applies a complex transformation using a matrix.

Variables in CSS

Variables in CSS, also known as custom properties, allow you to define reusable values in your stylesheet. They make it easy to maintain consistency in your design and simplify changes to your styles.

Here's how to define and use variables in CSS:

1. Define a variable:

```
:root {
--main-color: #333;
--secondary-color: #666;
 }
```

The :root selector targets the root element of the document, and the -- prefix indicates a custom property.

1. Use a variable:

```
header {
background-color: var(--main-color);
}
```

The var() function replaces the variable with its defined value.

Benefits of using variables:

- Consistency: Ensure consistent colors, spacing, and typography throughout your design.

- Reusability: Define a value once and use it multiple times.

- Easy maintenance: Update a variable in one place, and it affects all instances.

- Simplifies calculations: Use variables for calculations, like font sizes or spacing.

Media Queries

Media queries in CSS allow you to apply different styles based on various device screen sizes, orientations, and even device types. This is useful for creating responsive designs that adapt to different screen sizes and devices.

Basic syntax:

```
@media (condition) {
 /* styles here */
}
```

Common media query conditions:

- min-width and max-width for screen size ranges
- min-height and max-height for screen height ranges

Example:

```
@media (min-width: 768px) {
 /* styles for screens with a minimum width of 768px */
}

@media (max-width: 480px) {
 /* styles for screens with a maximum width of 480px */
}
```

Media queries can be used to:

- Hide or show elements based on screen size
- Change layout and grid systems
- Adjust font sizes and line heights
- Swap images or backgrounds
- Apply different styles for different devices or orientations

By using media queries, you can create a responsive design that adapts to different screen sizes and devices, providing an optimal user experience.

Bootstrap

Introduction to Bootstrap

Bootstrap is a popular open-source front-end framework used for developing responsive and mobile-first websites and web applications. It provides a collection of HTML, CSS, and JavaScript components, as well as pre-styled templates and utilities, to streamline the process of building modern and consistent user interfaces.

History

Bootstrap originated in 2011 at Twitter, created by Mark Otto and Jacob Thornton to simplify web development tasks within the company. It quickly gained popularity due to its responsive design, grid system, and pre-designed components. As an open-source project, Bootstrap evolved through versions like Bootstrap 2, 3, 4, and 5, each adding improvements and new features. Today, Bootstrap is widely used by developers worldwide for building responsive and mobile-friendly websites and applications with ease.

What is framework?

A framework is a pre-built software environment or structure that provides developers with a foundation of tools, libraries, and guidelines for building applications or software solutions. It is designed to streamline the development process by offering reusable components, standard practices, and a structured approach to coding.

Difference between framework and library?

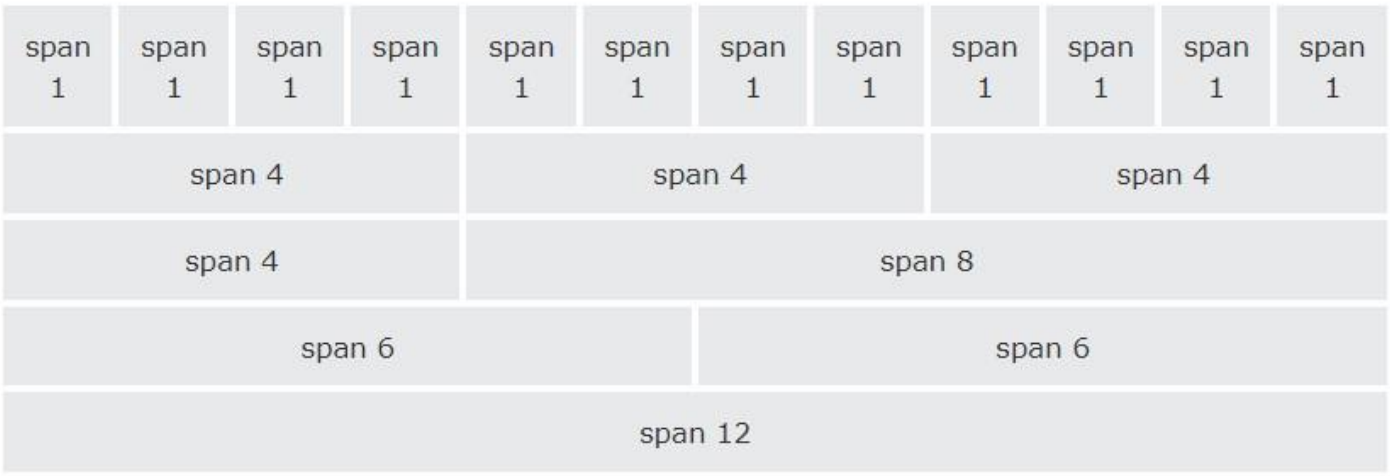
Framework: A framework is a structured software environment that provides a foundation with tools, libraries, and guidelines for building applications. It offers a predefined structure, architecture, and set of rules for developers to follow when creating software solutions.

Library: A library is a collection of reusable code modules or functions that provide specific functionalities to developers. Unlike a framework, a library does not dictate the overall structure or flow of an application but rather provides tools and utilities that can be used as needed.

Grid System

Bootstrap's grid system allows up to 12 columns across the page.

If you do not want to use all 12 column individually, you can group the columns together to create wider columns.



Bootstrap's grid system is responsive, and the columns will re-arrange depending on the screen size: On a big screen it might look better with the content organized in three columns, but on a small screen it would be better if the content items were stacked on top of each other.

Grid Classes

The Bootstrap grid system has four classes:

- xs (for phones - screens less than 768px wide)
- sm (for tablets - screens equal to or greater than 768px wide)
- md (for small laptops - screens equal to or greater than 992px wide)
- lg (for laptops and desktops - screens equal to or greater than 1200px wide)

Typography

Here are some typography classes in Bootstrap

- **.h1, .h2, .h3, .h4, .h5, .h6**:
  Heading classes for defining various levels of headings with decreasing font sizes.
- **.display-1, .display-2, .display-3, .display-4**
  Display classes for creating large, attention-grabbing headings.
- **.lead**
  Adds extra emphasis to a paragraph by increasing font size and adding margin.
- **.text-muted**:
  Applies a muted color to text for indicating less important information.
- **.text-primary, .text-secondary, .text-success, .text-danger, .text-warning, .text-info, .text-light, .text-dark**:
  Sets text color to predefined Bootstrap theme colors.
- **.text-white, .text-black-50, .text-white-50**:
  Sets text color to white, semi-transparent black, or semi-transparent white.
- **.font-weight-bold, .font-weight-normal, .font-weight-light, .font-weight-italic**:
  Sets font weight and style for text.
- **.text-center, .text-left, .text-right, .text-justify**:
  Aligns text horizontally within its container.

Colors in Bootstrap

- **Primary**: Sets the color to **blue**.
- **Secondary**: Sets the color to **gray**.
- **Success**: Sets the color to **green**.
- **Danger**: Sets the color to **green**.
- **Warning**: Sets the color to **yellow**.
- **Info**: Sets the color to **Cyan**.
- **Light**: Sets the color to a **lighter shade for contrast**.
- **Dark**: Sets the color to **black**.
- **White**: Sets the color to **white.**

Components

Buttons:

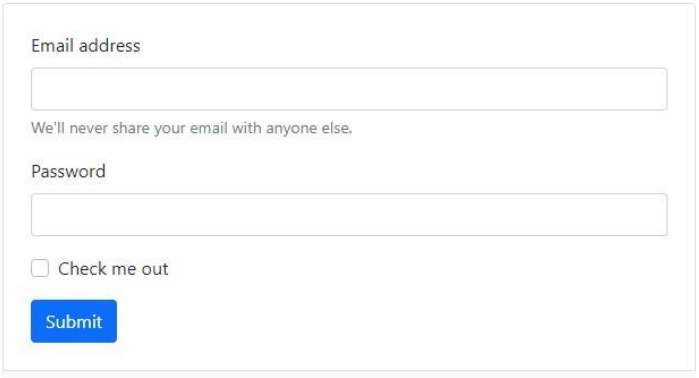
- **.btn**: Base class for buttons
- **.btn-primary**, **.btn-secondary**, **.btn-success**, **.btn-danger**, **.btn-warning**, **.btn-info**, **.btn-light**, **.btn-dark**: Different button styles

- **.btn-outline-primary**, **.btn-outline-secondary**, **.btn-outline-success**, **.btn-outline-danger**, **.btn-outline-warning**, **.btn-outline-info**, **.btn-outline-light**, **.btn-outline-dark**: Outline button styles

- **.btn-lg**, **.btn-sm**, **.btn-block**: Large, small, and block-level buttons

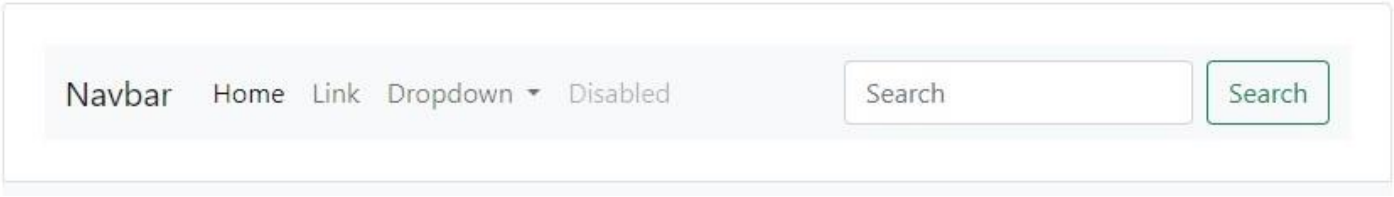
Forms:

- **.form-control**: Styles form controls like input fields, selects, textarea, etc.
- **.form-group**: Groups form controls together for proper spacing and alignment
- **.form-check**: Styles form checkboxes and radio buttons
- **.form-check-input**, **.form-check-label**: Specific classes for form checkboxes and radio buttons



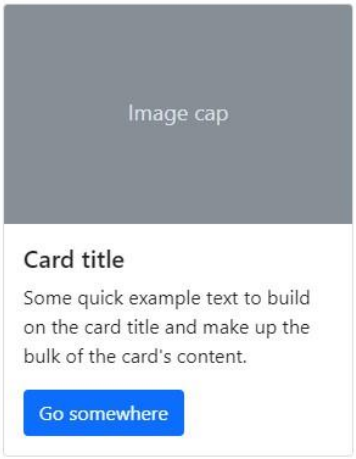
Navbar:

- **.navbar**: Base class for navbar
- **.navbar-expand-lg**, **.navbar-expand-md**, **.navbar-expand-sm**, **.navbar-expand-xl**: Responsive classes for expanding navbar
- **.navbar-light**, **.navbar-dark**: Light or dark navbar styles
- **.navbar-brand**: Styles for navbar brand/logo
- **.navbar-toggler**: Toggler button for responsive Navbar



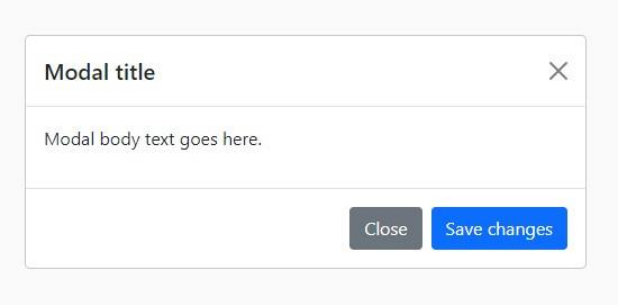
Cards:

- **.card**: Base class for card component
- **.card-header**, **.card-footer**: Header and footer sections of the card
- **.card-body**: Main content area of the card
- **.card-title**, **.card-subtitle**: Title and subtitle sections
- **.card-text**: Text content of the card
- **.card-img-top**, **.card-img-bottom**: Top and bottom image sections



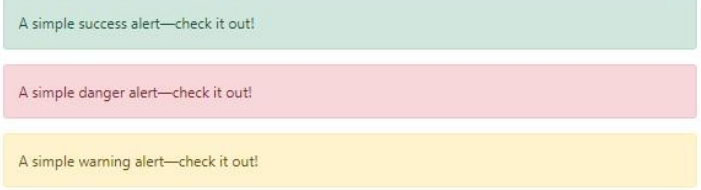
Modal:

- **.modal**: Base class for modal
- **.modal-dialog**: Dialog box container
- **.modal-content**: Content container within the modal
- **.modal-header**, **.modal-body**, **.modal-footer**: Sections within the modal
- **.modal-title**: Title of the modal



Alert:

- **.alert**: Base class for alerts
- **.alert-primary**, **.alert-success**, **.alert-danger**, **.alert-warning**, **.alert-info**: Different alert styles



Dropdown:

- **.dropdown**: Base class for dropdowns
- **.dropdown-toggle**: Toggle button for the dropdown
- **.dropdown-menu**: Dropdown menu container

Progress Bar:

- **.progress**: Base class for progress bar
- **.progress-bar**: Progress bar element
- **.progress-bar-striped**: Striped style for progress bar
- **.progress-bar-animated**: Animated progress bar

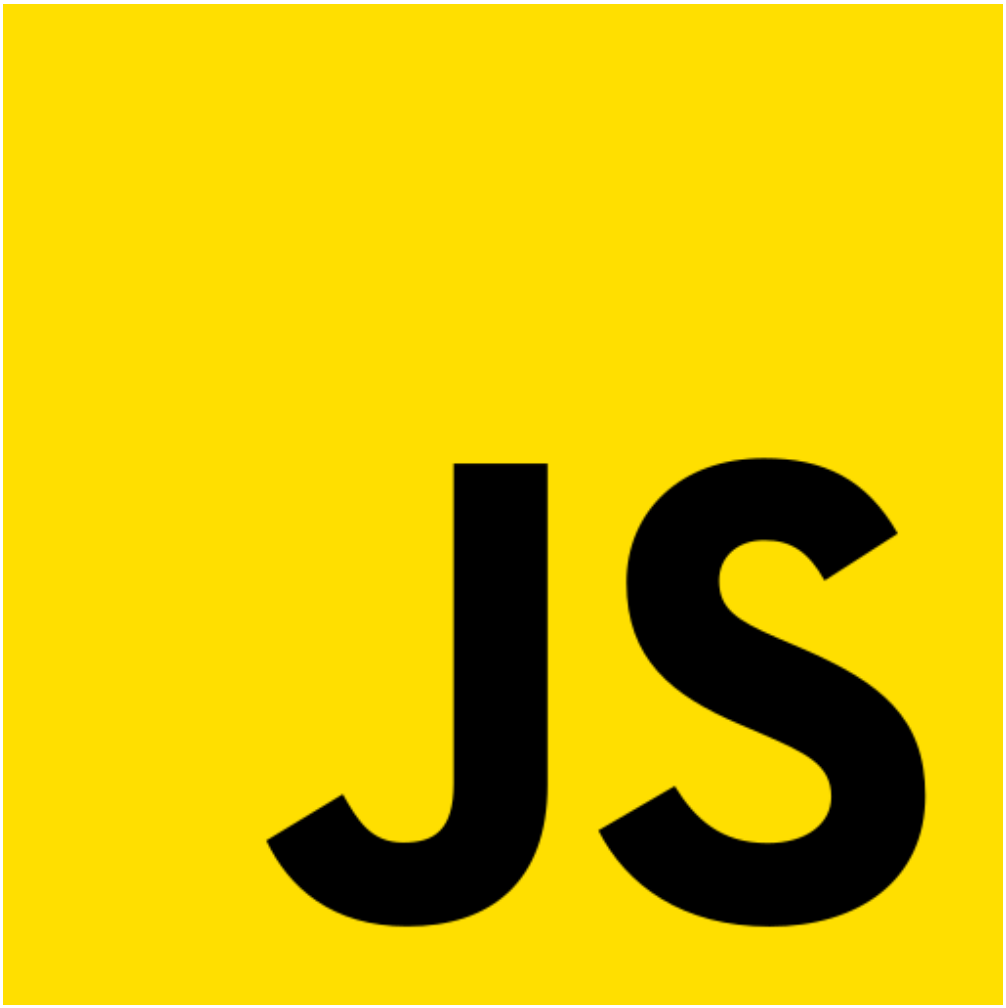
Badge:

- **.badge**: Base class for badges
- **.badge-primary**, **.badge-secondary**, **.badge-success**, **.badge-danger**, **.badge-warning**, **.badge-info**, **.badge-light**, **.badge-dark**: Different badge styles

Images:

- **.img-fluid**: Responsive image class
- **.rounded**: Adds rounded corners to an image
- **.circle**: Makes an image circular
- **.rounded-circle**: Makes an image circular with rounded edges
- **.thumbnail**: Adds a border and padding to an image

JavaScript



Introduction to JavaScript

What is JavaScript?

JavaScript is a high-level, dynamic, and interpreted programming language primarily used for client-side scripting on the web. It's executed on the client-side (in the user's web browser) rather than on the server-side, allowing for interactive web pages and web applications.

JavaScript used for:

- Creating responsive and interactive web pages
- Developing desktop and mobile applications
- Building server-side applications with technologies like Node.js
- Creating games and animations
- Adding functionality to web pages, such as form validation and calculations

History

JavaScript was created by **Brendan Eich in 1995** while he was working at Netscape Communications Corporation. Originally named Mocha, then LiveScript, and finally JavaScript, it was designed in just 10 days. In **1996**, JavaScript was submitted to **ECMA International** for standardization, leading to ECMAScript as its standardized specification. ECMAScript 3 (1999) became the de facto standard for web browsers for many years. The 2000s saw JavaScript's rise with Web 2.0 and AJAX. ECMAScript continued evolving, with ECMAScript 5 (2009) introducing significant enhancements. **ECMAScript 6 (ES6)** in 2015 marked a major milestone, bringing modern features. Today, JavaScript is used in web, server-side (Node.js), mobile apps (React Native), and desktop apps (Electron), with a thriving ecosystem of frameworks and libraries.

What is ECMA?

ECMA International, formerly known as the European Computer Manufacturers Association, is a standards organization responsible for developing and maintaining standards for information and communication technology (ICT) systems. They create standards for various technologies, including programming languages like JavaScript (through the ECMAScript standard), file formats, communication protocols, and more.

Variables

A variable is a named container used to store data values in a computer program. It acts as a placeholder for various types of information such as numbers, strings, Boolean, arrays, objects, and functions.

Types of Variables in JavaScript:

In JavaScript, variables can be categorized into three main types based on how they are declared and used:

1. **Var:** Variables declared using the **var** keyword are function-scoped or globally scoped. They can be redeclared and reassigned within their scope.

   Example: var age = 30;
   Var age = 40; // Redeclaration allowed
   age = 50; // Reassignment allowed

2. **Let Variables:** Introduced in ECMAScript 6 (ES6), variables declared with the **let** keyword are block-scoped, meaning they are only accessible within the block in which they are defined (enclosed by **{}**). They cannot be redeclared within the same scope, but they can be reassigned.

   Example: let name = "John";
   let name = "Jane"; Error: Cannot redeclare 'name' within the same scope
   name = "Jane"; // Reassignment allowed

3. **Const Variables:** Constants are declared using the **const** keyword. They are also block-scoped and cannot be reassigned after initialization. However, if the constant holds an object or array, the properties or elements of the object/array can be modified.

   Example: const PI = 3.14;
   // PI = 3.14159; // Error: Cannot reassign a constant
   Const person = {name: "John"};
   person.name = "Jane"; // Allowed, as only the property value is changed, not the reference

Naming Rules:

- Variable names must begin with a letter, dollar sign ($), or underscore (_).
- Subsequent characters can be letters, digits, dollar signs, or underscores.
- Variable names are case-sensitive.
- Avoid using reserved keywords as variable names.

Data types

A data type is a classification that specifies which type of value a variable can hold in a computer program. It defines the characteristics of the data and how it can be manipulated. In JavaScript, data types are categorized into primitive data types and non-primitive/reference data types:

Primitive Data Types: These are basic data types that represent single values.

- **Number:** Represents numeric values like all numbers and floating-point numbers.
  E.g. 10, 3.14, -5
- **String:** Represents textual data enclosed in single or double quotes.
  "Hello", ‘JavaScript’, “C", “This is a string”, “78”.
- **Boolean:** Represents a logical value of true or false.
- **Undefined:** Represents a variable that has been declared but not assigned a value. E.g. var a; // type is undefined because value is not assigned
- **Null:** Represents the intentional absence of any value. E.g. var a = null;
- **Symbol (ES6 and later):** Represents a unique and immutable value that may be used as an identifier for object properties. E.g. const mySymbol = Symbol()
- **BigInt (ES11 and later):** Represents integers with arbitrary precision.
  E.g. const bigNumber = 1234567890123456789012345678901234567890n;

Non-Primitive Data Types: These are data types that can hold multiple values or other data types in a single variable.

- **Object:** Represents a collection of key-value pairs where keys are strings and values can be any data type, including other objects. E.g. const boy = {name: “Ali” , age: 20}
- **Array:** Represents an ordered collection of elements, which can be of any data type.
  E**.**g. let arr = [2, 4, true, null, “Aslam”, 90]
- **Function:** A function in programming is a reusable block of code that performs a specific task or calculates a value, often taking inputs (parameters) and producing outputs (return values).
  E.g. function addNumbers (a, b)
  {return a+b ;}

By use of **typeof** operator datatype can be checked of any variable.

Data types play a crucial role in programming as they determine how data is stored, processed, and manipulated within a program. Different programming languages may have variations in their data types and how they are implemented.

Operators

In JavaScript, operators are symbols or keywords that perform operations on operands (variables or values). Short operators, also known as shorthand operators or compound assignment operators, are a concise way of combining arithmetic, bitwise, logical, and other operations with assignment.

1. Arithmetic Operators (+, -, *, /, %)

- **+**: Addition - Adds two operands.
- **-**: Subtraction - Subtracts the right operand from the left operand.
- ***:** Multiplication - Multiplies two operands.
- **/**: Division - Divides the left operand by the right operand.
- **%**: Modulus - Returns the remainder of the division of the left operand by the right operand.

2. Assignment Operators (=, +=, -=, *=, /=, %=)

- **=**: Assigns a value to a variable.

- **+=**: Adds the right operand to the left operand and assigns the result to the left operand.
- **-=**: Subtracts the right operand from the left operand and assigns the result to the left operand.
- ***=**: Multiplies the left operand by the right operand and assigns the result to the left operand.
- **/=**: Divides the left operand by the right operand and assigns the result to the left operand.
- **%=**: Computes the modulus of the left operand with the right operand and assigns the result to the left operand.

3. Comparison Operators (==, ===, !=, !==, >, <, >=, <=)

- **==**: Equality - Checks if two operands are equal in value (with type conversion).
- **===**: Strict Equality - Checks if two operands are equal in value and type.
- **!=**: Inequality - Checks if two operands are not equal in value (with type conversion).
- **!==**: Strict Inequality - Checks if two operands are not equal in value or type.
- **>**: Greater Than - Checks if the left operand is greater than the right operand.
- **<**: Less Than - Checks if the left operand is less than the right operand.
- **>=**: Greater Than or Equal To - Checks if the left operand is greater than or equal to the right operand.
- **<=**: Less Than or Equal To - Checks if the left operand is less than or equal to the right operand.

4. Logical Operators (&&, ||, !)

- **&&**: Logical AND - Returns true if both operands are true.
- **||**: Logical OR - Returns true if at least one operand is true.
- **!** Logical NOT - Returns true if the operand is false and vice versa.

5. Unary Operators(Increment/Decrement ++ , --)

- **++i** or **i++**: Increment - Adds 1 to the operand.
- **--i** or **i--**: Decrement - Subtracts 1 from the operand.

6. Conditional (Ternary) Operator (condition ? expr1 : expr2)

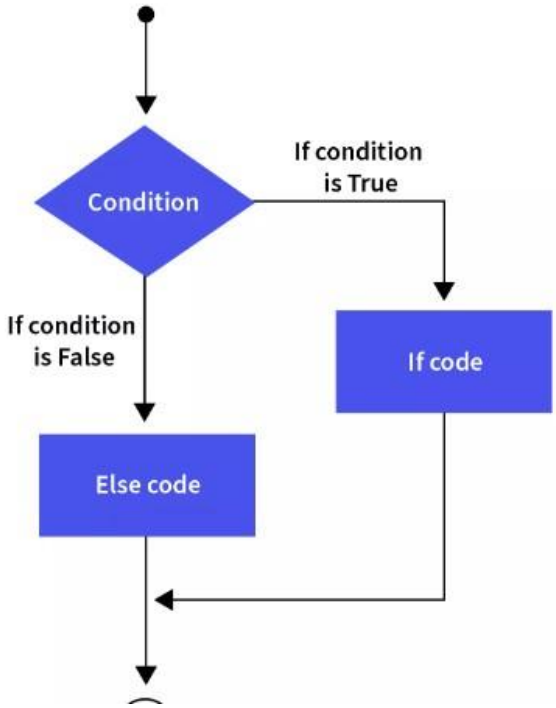
- **condition ? expr1 : expr2**: Evaluates the condition and returns **expr1** if true, otherwise returns **expr2**.

Conditional Statements (if – else)

Conditional statements in JavaScript, such as **if**, **else if**, and **else**, are used to make decisions in code based on certain conditions. They allow you to execute different blocks of code depending on whether a condition evaluates to true or false.

If statement: It executes a block of code if the specified condition is true.

Example: if (condition) {
// Code is execute when condition is true }



Else statement: It executes a block of code if the condition in the **if** statement is false.

Example: if (condition) {
// Code is execute when condition is true }
else {
// Code is execute when condition is false
}

Else if statement: It allows you to check multiple conditions. If the first **if** condition is false, it checks the next **else if** condition and so on.

Example:
```
if (condition1) {
        // Code is execute if condition1 is true}
 else if (condition2) {
        // Code is execute if condition2 is true}
else if (condition3) {
        // Code is execute if condition3 is true}
 else {
        // Code is execute when all conditions are false
}
```

Nested if: Nested if statements refer to placing one if statement inside another if statement. This structure allows for more complex decision-making logic by checking multiple conditions

sequentially. Each nested if statement is evaluated only if its outer if statement's condition is true.

Example:

```
let num = 15;
If (num > 0) {
        If (num % 2 === 0) {
            Console.log (“Number is Positive and even”);
          else {
            Console.log (“Number is Positive and odd”);   }
else {
            Console.log (“Number is non-Positive”);  }
```

Loops

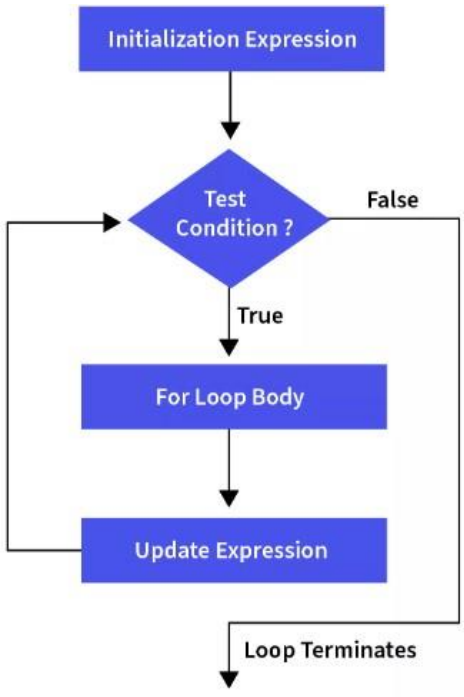
A loop is a programming construct that allows a section of code to be repeated a certain number of times, or until a specific condition is met. Loops are used to perform repetitive tasks, iterate over data, and control the flow of a program.

Loops are useful for:

- Repeating a task multiple times
- Iterating over data structures such as arrays or objects
- Implementing algorithms that require repetition
- Controlling the flow of a program

For Loop

A for loop in JavaScript is a control structure that allows you to execute a block of code repeatedly for a specified number of times. It's typically used when you know the exact number of iterations required. The for loop consists of three parts: initialization, condition, and increment/decrement. You initialize a variable before the loop starts, set a condition that determines whether the loop should continue or exit, and specify how the variable should change after each iteration.



Syntax: for (initialization; condition; increment) {

//code to be executed repeatedly until given condition become false }

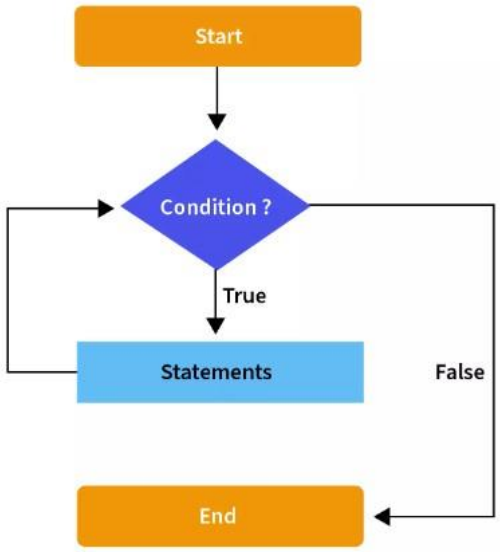
Example: for (let i = 0; i < 5; i++) {

Console.log (i);

}

While Loop

A while loop in JavaScript is a control structure that repeatedly executes a block of code as long as a specified condition remains true. Unlike a for loop where you specify the number of iterations, a while loop relies solely on the condition to control the loop's execution.



Syntax: while (condition) {

//code to be executed repeatedly until given condition become false

Increment / decrement (++/--) at the end }

Example: var i = 0;

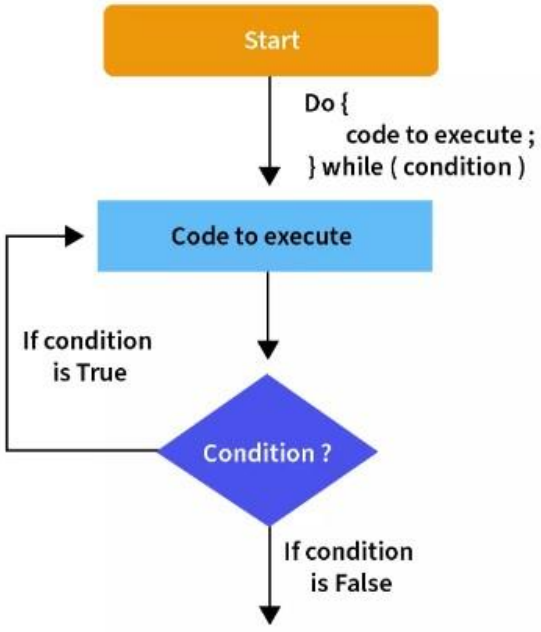
while (i < 5) {

console.log (i);

i++; }

Do While Loop

A do...while loop in JavaScript is a control structure that is similar to a while loop but with one key difference: it guarantees that the code block will be executed at least once, even if the condition initially evaluates to false. After the first iteration, the condition is checked, and if it's true, the loop continues to execute. If the condition is false, the loop exits.



Syntax: do {

Code to be executed

Increment/decrement (++/--) at the end

} while (condition);

Example:

```
var i = 0;
do {
    console.log (i);
     i++;
} while (i < 5);
```

Break Statement

The **break** statement is used to terminate the execution of a loop (for, while, do...while) or a switch statement prematurely. When encountered, the **break** statement immediately exits the loop or switch block and continues with the next statement after the loop or switch.

Example:

```
for (let i = 0; i < 5; i++) {
if ( i === 3 ){
break          // Exit the loop when i is 3        }
Console.log (i);      }
```

Continue Statement:

The **continue** statement is used to skip the current iteration of a loop and continue with the next iteration. When encountered, the **continue** statement skips the remaining code in the loop for the current iteration and proceeds with the next iteration.

Example:

```
for (let i = 0; i < 5; i++) {
if ( i === 2){
continue      // Skip the iteration when i is 2          }
Console.log (i);      }
```

For of Loop

The **for...of** loop in JavaScript is a convenient way to iterate over elements in iterable objects such as arrays, strings, maps, sets, and more. Unlike the traditional **for** loop that uses numeric indices or the **for...in** loop that iterates over object keys, the **for...of** loop directly accesses the values of the elements in the iterable object.

Syntax: for (var variable of iterable) {code to be executed}

Example:

```
var fruits = ['apple', 'banana', 'cherry'];
```

for (var fruit of fruits) { console.log(fruit); }

For each Loop

In JavaScript, the **forEach** loop is a higher-order function available for arrays. It iterates through each element of an array and executes a provided callback function for each element. The **forEach** loop is a concise and expressive way to perform operations on array elements without explicitly writing a traditional **for** loop.

Syntax: for (var variable in object) { code to be executed }

Example: var fruits = ['apple', 'banana', 'cherry'];
fruits.forEach(function(f){ console.log(f) })

String Methods

String methods are functions that operate on strings in JavaScript. They allow you to manipulate and work with text data efficiently. Some common string methods include:

- **length**: Returns the length of a string, i.e., the number of characters it contains.
- **concat(string)**: Combines two or more strings and returns a new string.
- **indexOf(substring)**: Returns the index of the first occurrence of a specified substring within a string.
- **slice(start, end)**: Extracts a section of a string and returns it as a new string.
- **toUpperCase()**: Converts all characters in a string to uppercase.
- **toLowerCase()**: Converts all characters in a string to lowercase.
- **trim()**: Removes whitespace from both ends of a string.
- **replace(old, new)**: Replaces occurrences of a specified substring with another string.
- **split(separator)**: Splits a string into an array of substrings based on a specified separator.
- **charAt(index)**: Returns the character at the specified index in a string.

Array Methods

Array methods are functions used to manipulate arrays in JavaScript. They enable efficient handling of collections of data elements. Here are some common array methods along with short descriptions:

- **push(element)**: Adds one or more elements to the end of an array and returns the new length of the array.
- **pop()**: Removes the last element from an array and returns that element.
- **shift()**: Removes the first element from an array and returns that element, shifting all other elements down by one index.
- **unshift(element)**: Adds one or more elements to the beginning of an array and returns the new length of the array.
- **concat(array)**: Combines two or more arrays and returns a new array.
- **join(separator)**: Joins all elements of an array into a string, optionally separated by the specified separator.
- **slice(start, end)**: Extracts a section of an array and returns a new array.
- **splice(start, count, item1, item2, ...)**: Adds or removes elements from an array at a specified index.
- **indexOf(element)**: Returns the index of the first occurrence of a specified element within an array, or -1 if not found.
- **forEach(callback)**: Calls a function once for each element in an array.
- **map(callback)**: Creates a new array with the results of calling a provided function on every element in the array.
- **filter(callback)**: Creates a new array with all elements that pass a test implemented by the provided function.
- **reduce(callback, initialValue)**: Applies a function to each element of an array, resulting in a single output value.
- **Sort (compareFunction)**: Sorts the elements of an array in place according to a provided comparison function.
- **reverse()**: Reverses the order of elements in an array.

Objects

In JavaScript, objects are a fundamental data type that represents a collection of key-value pairs. They are used to store and manipulate data in a structured way.

Here are some key aspects of objects in JavaScript:

Creating objects:

Using the object literal syntax**: var obj = { key: value, key: value };**

Using the new keyword: **var obj = new Object();**

Properties:

Objects have properties, which are key-value pairs.

Properties can be added, removed, and modified.

Accessing properties:

Using dot notation: **obj.key**

Using bracket notation: **obj['key']**

Iterating over properties:

Using for...in loop: **for (var key in obj) { console.log(key, obj[key]); }**

Functions

Functions in JavaScript are blocks of code that can be called multiple times from different parts of your program. They help to:

- Organize code
- Reduce repetition
- Make code reusable
- Simplify complex tasks

Here are some key aspects of functions in JavaScript:

1. Declaring functions:

- Function declaration: function name (parameters) {code to be executed}

- Function expression: var name = function (parameters) {code to be executed}

2. Calling functions:

- name (parameters)

3. Parameters:

- Functions can take any number of parameters
- Parameters can be of any data type

4. Return values:

- Functions can return any data type
- Use the return statement to specify the return value

5. Scope:

- Functions have their own scope
- Variables declared inside a function are local to that function

6. Hoisting:

- Function declarations are hoisted to the top of their scope
- Function expressions are not hoisted

7. Arrow functions (ES6+):

- Concise syntax for functions
- Automatically bind the this keyword
- **Example**: var name = (parameters) => { code to be executed }

DOM (Document Object Model)

The Document Object Model (DOM) in JavaScript is a programming interface that represents the structure of a web document as a tree of objects. Each element, attribute, and text node in an HTML or XML document is represented as a node in the DOM tree. JavaScript interacts with the DOM to dynamically manipulate the content, structure, and style of a web page.

The DOM allows JavaScript to:

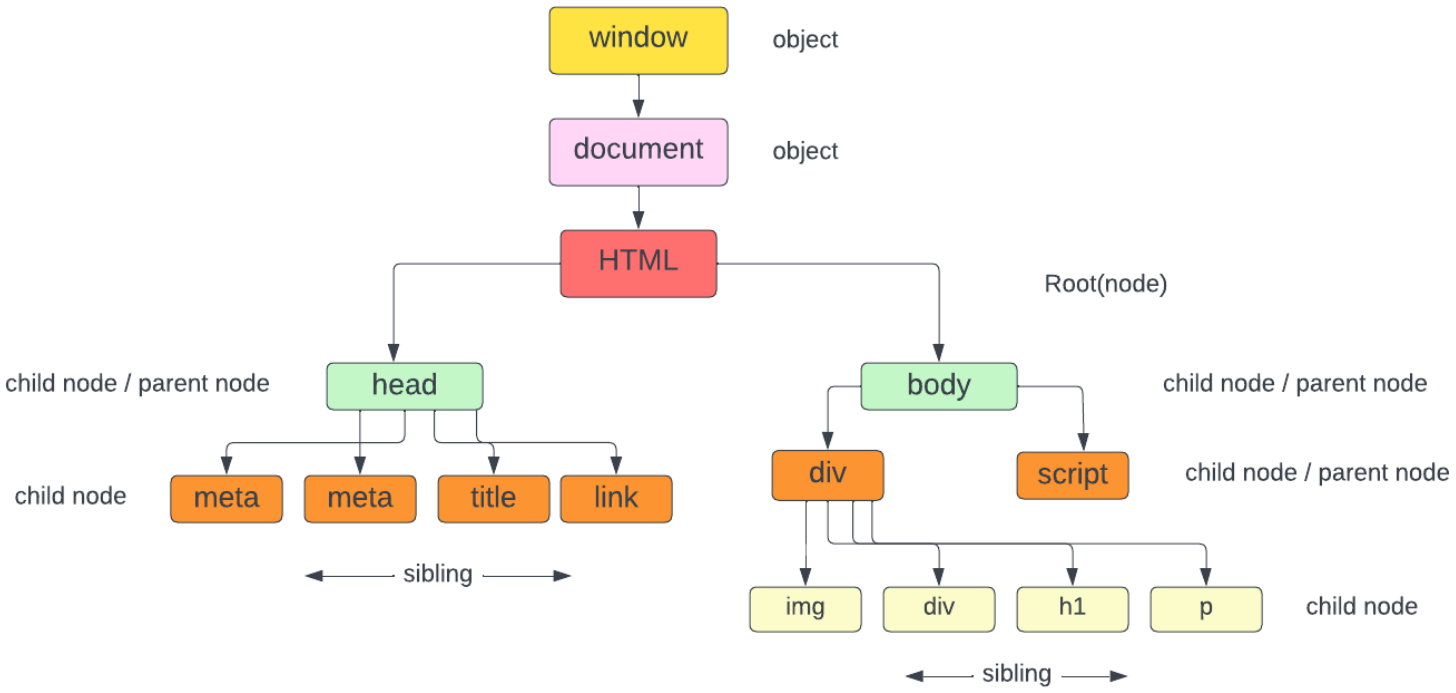
- Access and manipulate elements and their properties
- Add, remove, and modify elements and their content
- Handle events and user interactions
- Traverse and search the document tree

Here are some key aspects of the DOM in JavaScript:

1. **Nodes:** Represent elements, attributes, text, and other parts of the document
2. **Elements:** Represent HTML elements, such as tags and attributes
3. **Tags:** Represent the name of an HTML element, such as or <p>
4. **Attributes:** Represent additional information about an element, such as id or class
5. **Text:** Represent the text content of an element
6. **Events:** Represent user interactions, such as clicks or key presses

Common DOM methods and properties in JavaScript:

1. **document.getElementById():** Retrieves an element by its ID
2. **document.querySelector():** Retrieves an element by its selector
3. **element.innerHTML:** Sets or gets the HTML content of an element
4. **element.textContent:** Sets or gets the text content of an element
5. **element.setAttribute():** Sets an attribute on an element
6. **element.addEventListener():** Adds an event listener to an element
7. **element.parentNode:** Retrieves the parent node of an element
8. **element.childNodes:** Retrieves a collection of child nodes



Events

In JavaScript, events are actions or occurrences that happen in a web page, such as:

- User interactions (e.g., clicks, hover, key presses)
- Browser events (e.g., page loads, scrolls, resizes)
- Timer events (e.g., setTimeout, setInterval

Common events in JavaScript:

1. Mouse events:

- click
- dblclick
- mouseover
- mouseout
- mousemove

2. Keyboard events:

- keydown
- keyup
- keypress

3. Form events:

- submit
- change
- focus
- blur

4. Document and window events:

- load
- unload
- scroll
- resize

Date Object

The Date object in JavaScript is a built-in object that represents a single moment in time. It has various methods and properties that allow you to work with dates and times.

Here are some key properties and methods of the Date object:

Properties:

- **date:** Returns the day of the month (1-31)
- **day:** Returns the day of the week (0-6)
- **FullYear:** Returns the full year (4 digits)
- **hours:** Returns the hour (0-23)
- **milliseconds:** Returns the number of milliseconds since January 1, 1970, 00:00:00 UTC

- **minutes:** Returns the minutes (0-59)
- **month:** Returns the month (0-11)
- **seconds:** Returns the seconds (0-59)
- **time**: Returns the number of milliseconds since January 1, 1970, 00:00:00 UTC

Get Methods:

- **getDate():** Returns the day of the month (1-31)
- **getDay():** Returns the day of the week (0-6)
- **getFullYear():** Returns the full year (4 digits)
- **getHours():** Returns the hour (0-23)
- **getMilliseconds():** Returns the number of milliseconds since January 1, 1970, 00:00:00.
- **getMinutes():** Returns the minutes (0-59)
- **getMonth():** Returns the month (0-11)
- **getSeconds():** Returns the seconds (0-59)
- **getTime():** Returns the number of milliseconds since January 1, 1970, 00:00:00 UTC

Set Methods:

- **setDate():** Sets the day of the month (1-31)
- **setFullYear():** Sets the full year (4 digits)
- **setHours():** Sets the hour (0-23)
- **setMilliseconds():** Sets the number of milliseconds since January 1, 1970, 00:00:00 UTC
- **setMinutes():** Sets the minutes (0-59)
- **setMonth():** Sets the month (0-11)
- **setSeconds():** Sets the seconds (0-59)
- **setTime():** Sets the number of milliseconds since January 1, 1970, 00:00:00 UTC

Note that the Date object is a mutable object, meaning that it can be modified after it's created. Also, the Date object is a global object, meaning it can be accessed from anywhere in your code.

Math Object

The Math object in JavaScript is a built-in object that provides mathematical functions and constants. It has various methods and properties that allow you to perform mathematical operations and calculations.

Methods:

- **max(x, y):** Returns the maximum of x and y
- **min(x, y):** Returns the minimum of x and y
- **pow(x, y):** Returns x raised to the power of y
- **random():** Returns a random number between 0 and 1
- **round(x):** Returns the nearest integer to x, rounding according to the prevailing rounding mode
- **ceil(x):** Returns the smallest integer greater than or equal to x
- **floor(x):** Returns the largest integer less than or equal to x
- **sqrt(x):** Returns the square root of x

setTimeout:

- setTimeout takes two arguments: a function to be executed and a time in milliseconds.
- It schedules the function to run after the specified time has elapsed.
- The function is executed only once.

Example:

```
setTimeout(function() {
  console.log("Hello, World!");
}, 3000); // runs after 3 seconds
```

setInterval:

- setInterval takes two arguments: a function to be executed and a time in milliseconds.
- It schedules the function to run at regular intervals, specified by the time argument.
- The function is executed repeatedly at the specified interval.

Example:

```
setInterval(function() {
  console.log("Hello, World!");
}, 3000); // runs every 3 seconds
```

Common use cases:

- Delaying the execution of code
- Creating animations or visual effects
- Polling or checking for updates at regular intervals
- Implementing timers or countdowns