Uraan
SOFTSKILLS PVT. LTD.

Learning HTML CSS & JS

With Aditya G.

Agenda

- Get to know each other
  - Let me and all others know you
  - Know me
- About this training program
  - Why do I do this course? - Benefits
  - What after the course?
  - What is all included in this course?
- About scope of React JS in current market
- Course syllabus & Activities
  - A walkthrough for course syllabus
  - A walkthrough for activities involved with this course

About this training program

- Why do I do this course? - **Benefits**
  - Easy to use & Popular among developers
  - Rapidly growing community = Rich Library / plugins
  - Unique React features:
    - Performance & Reusability
    - Patterns of React: Declarative, Small and big projects suitable.
    - Backward compatibility
    - Concurrent mode
    - SEO Friendly Framework
- What after the course? - **Future**
  - Internship / Job / StartUp / Product development
  - Community contributor / Public speaker
- What is the Objectives of this course? - **Objectives**
  - Develop interactive user interfaces (UIs) and web applications using React, JSX, and ES6.
  - Build dynamic websites and front-end applications quickly and easily with reusable React components.
  - Communicate and exchange data with external services using GET, POST, UPDATE, and DELETE requests.
  - Employ and work with various React concepts and features including props, states, hooks, forms, and Redux

About this training program

- English communication - MERN
- Logic building classes
- Basic programming knowledge (C, C++)
- Frontend basic included (HTML, CSS, JavaScript, JQuery, Bootstrap)
- Project guidance and brainstorming sessions

How a software works?

- In order to start learning about Node.js let’s first understand how a software works
- Type of Application software
    - Desktop application software
    - Web application software
    - Mobile application software
    - Car application software
    - TV application software
    - Watch application software
    - VR application software
    - Embedded application software
    - ETC.
- What is the source of data for the application?
    - For applications like: Facebook, Whatsapp, Twitter, LinkedIn etc?

How a software works?

- In order to start learning about Node.js let’s first understand how a software works
- Type of Application software
    - Desktop application software
    - Web application software
    - Mobile application software
    - Car application software
    - TV application software
    - Watch application software
    - VR application software
    - Embedded application software
    - ETC.
- What is the source of data for the application?
    - For applications like: Facebook, Whatsapp, Twitter, LinkedIn etc?
        - Central Application Database

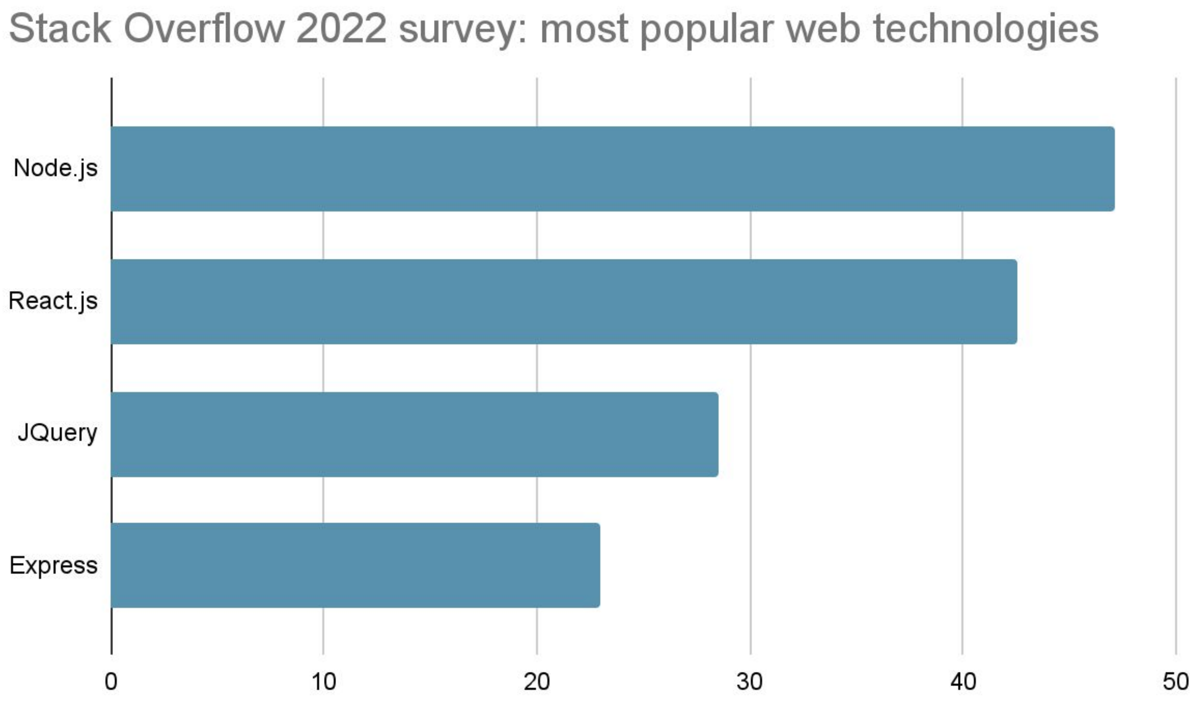
What is a Client & a Server?

Client	**Server**
Types: Web, Desktop, Mobile, Watch, Car, VR etc.	**Types:** DB server, Application server, Web server ● **DB server**: Keeps, retrieve, & deliver data ● **Web servers:** Ensures the client requests are responded to. Using HTTP request. ● **Application server:** sits between Web server and DB server to run the Application logic
Languages: HTML+CSS+JS	**Languages:** Java, PHP, JS, C#, Python, Ruby
Frameworks: React.js, Angular, etc.	**Frameworks:** Django (Python), Ruby on Rails (Ruby), Express.js (JS)

What is a Frontend & Backend?

Frontend	**Backend**
Runs on a client machine	Runs on the backend server
Runs in Browser engine	Is communicated by browser engine to get details to show in Frontend
Frontend developers	Devs write code to communicate with the browser’s engine & backend server
Includes: Client side application, frontend logic which doesn’t need backend	Includes: servers, DB, Web APIs, programming Languages, frameworks & runtimes
Provide user interface to user to interact with the Software/Application	Provides data and communication to a client

Why Node.js & Express?



HTML

Module 1 - HTML | Introduction to HTML

- **What is HTML?**
  a. In simple words - Skeleton of a website is created with HTML
  b. Hypertext Markup Language (HTML)
     i. HTML is the standard markup language for creating Web pages
     ii. HTML describes the structure of a Web page
     iii. HTML consists of a series of elements
     iv. HTML elements tell the browser how to display the content like text, links, images, and other forms of multimedia on a webpage
     v. HTML sets up the basic structure of a website, and then CSS and JavaScript add style and interactivity to make it look and function better.
     vi. HTML sets up the basic structure of a website, and then CSS and JavaScript add style and interactivity to make it look and function better.
  c. How Does HTML Work?
     i. HTML documents are plain-text files saved with an .html extension.
     ii. Browsers read these documents, interpret the markup (tags and attributes), and render the formatted content on your screen.

Module 1 - HTML | HTML Features

- **Simple HTML page**

```
<!DOCTYPE html>
<html>
     <head>
          <title>Page Title</title>
     </head>
     <body>

          <h1>My First Heading</h1>
          <p>My first paragraph.</p>

     </body>
</html>
```

Module 1 - HTML | HTML Features

- **Creating first HTML web page**
  a. Open an editor - VSCode / Notepad / TextEdit etc
  b. Write the code shown in previous slide
  c. Save it with a name with .html extension. Example: “introduction.html”
  d. View the page in your browser
     i. By selecting file -> Open -> locate and select the same file in your browser
     ii. Or by copying the path of the file location and pasting in the URL field of a browser window to open the page

Module 1 - HTML | HTML Features

- The `<!DOCTYPE html>` declaration defines that this document is an HTML5 document - **Optional**
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the HTML page
- The `<title>` element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The `<body>` element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The `<h1>` element defines a large heading
- The `<p>` element defines a paragraph

Module 1 - HTML | HTML Features

What is an HTML Element?

An HTML element is defined by a start tag, some content, and an end tag:

- `<tagname>` Content goes here... `</tagname>`
- The HTML element is everything from the start tag to the end tag:
- `<h1>`My First Heading`</h1>`
- `<p>`My first paragraph.`</p>`

Start Tag	Element Content	End Tag
<h1>	My First Heading	</h1>
<p>	My First Paragraph	</p>
<br>	none	none

Module 1 - HTML | HTML Features

What is an HTML Attributes/Properties?

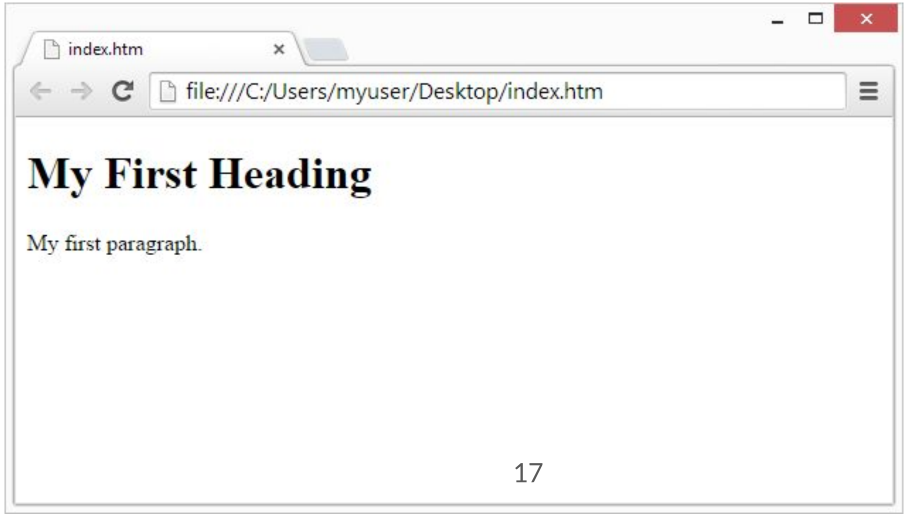
- All HTML elements can have attributes
- Attributes provide additional information about elements
- Attributes are always specified in the start tag
- Attributes usually come in name/value pairs like: name="value"
- Example:
  a. Anchor tag
     i. `<a href="https://www.w3schools.com">Visit W3Schools</a>`
     ii. Absolute URL - https://www.w3schools.com
     iii. Relative URL - `src="/images/img_girl.jpg”`
  b. Image tag
     i. `<img src="img_girl.jpg">`
     ii. `<img src="img_girl.jpg" width="500" height="600">`
     iii. `<img src="img_girl.jpg" alt="Girl with a jacket">`
  c. Others
     i. `<p style="color:red;">This is a red paragraph.</p>`

Module 1 - HTML | HTML Features

Web Browser

The purpose of a web browser (Chrome, Edge, Firefox, Safari) is to read HTML documents and display them correctly.

A browser does not display the HTML tags, but uses them to determine how to display the document:


index.htm

file:///C:/Users/myuser/Desktop/index.htm

My First Heading

My first paragraph.

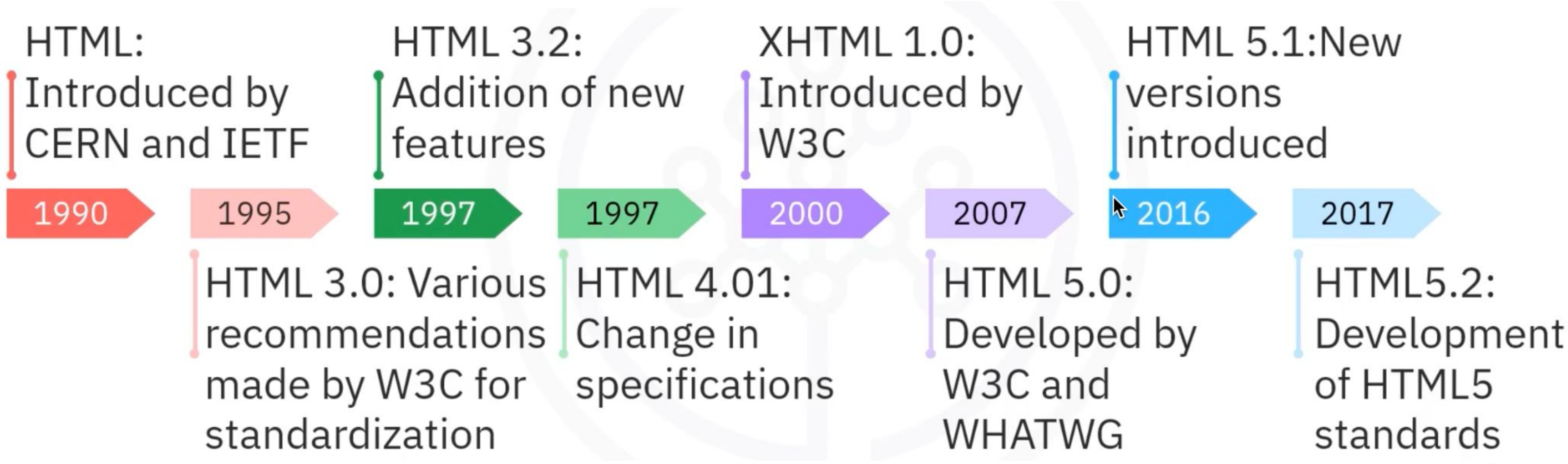
Module 1 - HTML | HTML Features

HTML Page Structure

```
<html>
  <head>
      <title> Page title </title>
  </head>
  <body>
     <h1> This is a heading </h1>
     <P> This is a paragraph </p>
  </body>
</html>
```

Module 1 - HTML | Introduction to HTML

- **Evolution of HTML**



1990 — HTML: Introduced by CERN and IETF

1995 — HTML 3.0: Various recommendations made by W3C for standardization

1997 — HTML 3.2: Addition of new features

1997 — HTML 4.01: Change in specifications

2000 — XHTML 1.0: Introduced by W3C

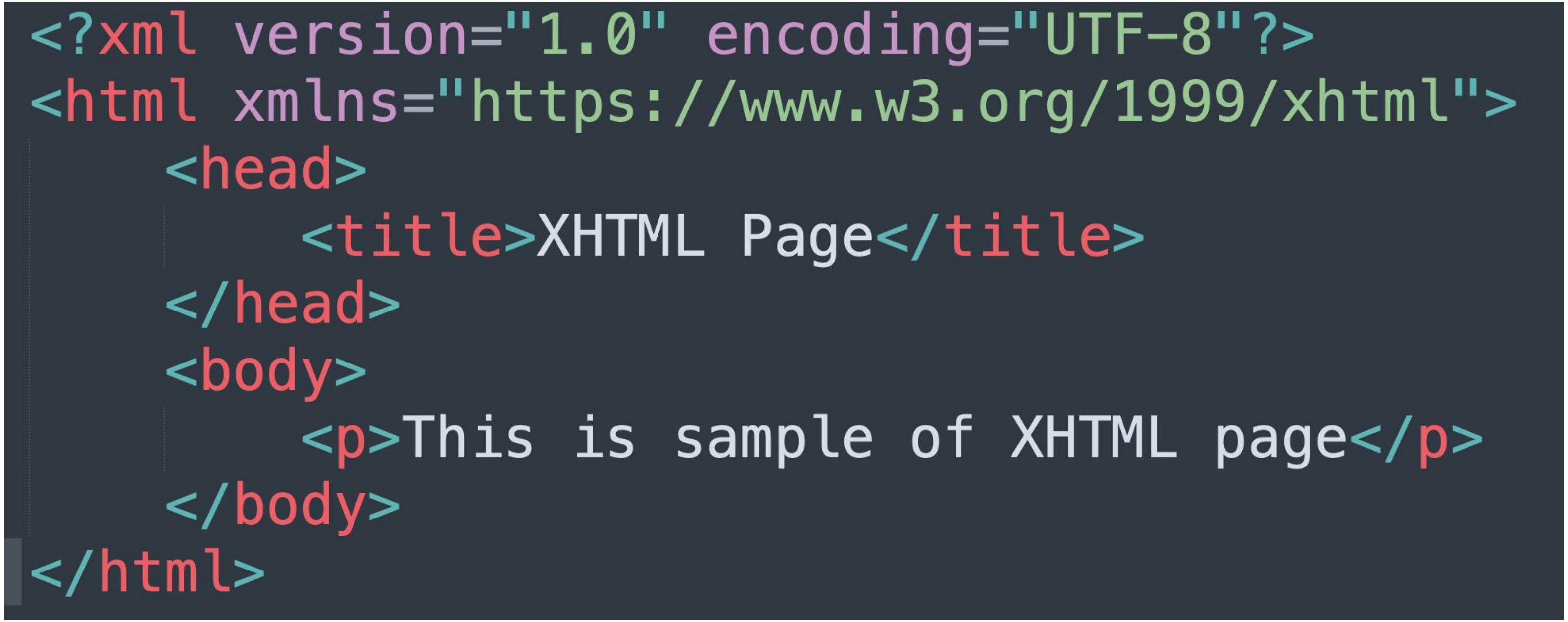
2007 — HTML 5.0: Developed by W3C and WHATWG

2016 — HTML 5.1:New versions introduced

2017 — HTML5.2: Development of HTML5 standards

Module 1 - HTML | HTML Features

- XML example



```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="https://www.w3.org/1999/xhtml">
    <head>
        <title>XHTML Page</title>
    </head>
    <body>
        <p>This is sample of XHTML page</p>
    </body>
</html>
```

Module 1 - HTML | HTML Features

- **XHTML**
  a. Tags need to be in lowercase.
  b. Codes must be well-formed
  c. XML parser will stop processing if it encounters a situation where the syntax is not well-formed

- **HTML**
  a. Case user doesn’t matter
  b. Unmatched quotation marks, non-terminated and uncontained elements are allowed
  c. Syntax is less rigorous than XHTML syntax

Module 1 - HTML | Intro to HTML (JSFiddle)

- https://jsfiddle.net/

Module 1 - HTML | Working with HTML

Let’s create these elements in our new website now:

- Headings
- Paragraph
- Horizontal line
- Line break
- Hyperlink / Anchor
- List
- Table
- Image

Module 1 - HTML | Working with HTML

Heading Tag

```
<html>
     <head>
          <title>25th Sept</title>
     </head>
     <body>
          <h1>Heading with h1</h1>
          <h2>Heading with h2</h2>
          <h3>Heading with h3</h3>
          <h4>Heading with h4</h4>
          <h5>Heading with h5</h5>
          <h6>Heading with h6</h6>
          <h7>Heading with h7</h7>
     </body>
</html>
```

Module 1 - HTML | Working with HTML

Heading Tag

```
<html>
      <head>      <title>25th Sept</title>      </head>
      <body>
            <p>This is my first html page testing with different tags <br> This is my first
html page testing with different tags </p>
            <p>This paragraph
            contains a lot of lines
            in the source code,
            but the browser
            ignores it.</p>
            <p>This paragraph
            contains      a lot of spaces
            in the source     code,
            but the    browser
            ignores it.</p>
            <p>The number of lines in a paragraph depends on the size of the browser window.
            If you resize the browser window, the number of lines in this paragraph will
            change.</p>

      </body>
</html>
```

Module 1 - HTML | Working with HTML

Horizontal line & Line break

```
<html>
     <head>    <title>25th Sept</title> </head>
     <body>
          <p>This is my first html page testing with different tags **<br>** This is
my first html page testing with different tags</p>
          **<hr>**
          <p>The number of lines in a paragraph depends on the size of the browser
          window. If you resize the browser window, the number of lines in this
          paragraph will change.</p>

     </body>
</html>
```

Module 1 - HTML | Working with HTML

Problem with P tag:

A Poem will be shown in a single line:

```
<p>
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.

  Oh, bring back my Bonnie to me.
</p>
```

Module 1 - HTML | Working with HTML

Problem with P tag:

A Poem will be shown in a single line:

```
<p>
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.

  Oh, bring back my Bonnie to me.
</p>
```

Solution: pre tag - element defines preformatted text. The text inside a `<pre>` element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks

```
<pre>
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.

  Oh, bring back my Bonnie to me.
</pre>
```

Module 1 - HTML | Working with HTML

Anchor tag

```
<html>
     <head>
          <title>25th Sept</title>
     </head>
     <body>
          <p>This is my first html page testing with different tags <br> This is
my first html page testing with different tags</p>
          <a href="https://uraansoftskills.com" target="_blank">Uraan
Softskills</a>
     </body>
</html>
```

Module 1 - HTML | Working with HTML

Lists (Ordered list)

```
<html>
     <head>
          <title>25th Sept</title>
     </head>
     <body>
          <ol>
               <li>Abc</li>
               <li>Def</li>
               <li>Ghi</li>
               <li>Jkl</li>
               <li>Mno</li>
          </ol>
     </body>
</html>
```

Module 1 - HTML | Working with HTML

Lists (Unordered list)

```
<html>
     <head>
          <title>25th Sept</title>
     </head>
     <body>
          <ul>
               <li>Abc</li>
               <li>Def</li>
               <li>Ghi</li>
               <li>Jkl</li>
               <li>Mno</li>
          </ul>
     </body>
</html>
```

Module 1 - HTML | Working with HTML

Table

```
<html>
      <head>      <title>25th Sept</title>          </head>
      <body>
            <table>
                  <tr>
                        <th>Heading 1</th>
                        <th>Heading 2</th>
                        <th>Heading 3</th>
                  </tr>
                  <tr>
                        <td>Data 1</td>
                        <td>Data 2</td>
                        <td>Data 3</td>
                  </tr>
                  <tr>
                        <td>Data 1</td>
                        <td>Data 2</td>
                        <td>Data 3</td>
                  </tr>
            </table>
      </body>
</html>
```

Module 1 - HTML | Working with HTML

Image tag

```
<html>
      <head>
            <title>25th Sept</title>
      </head>
      <body>
            <img
src="https://upload.wikimedia.org/wikipedia/commons/e/ef/Virat_Kohli_during_the_India
_vs_Aus_4th_Test_match_at_Narendra_Modi_Stadium_on_09_March_2023.jpg"
            width="100"
            height="100">
            <img src="img1.png" width="200" height="200">
      </body>
</html>
```

Module 1 - HTML | Working with HTML

div tag

a. Used as a section or section in the HTML document
b. Example:

form tag

c. A form is used to create an HTML form for user input
d. A form can have these in it:
   i. <input>
   ii. <textarea>
   iii. <button>
   iv. <select>
   v. <option>
   vi. <optgroup>
   vii. <fieldset>
   viii. <label>

Module 1 - HTML | Working with HTML

- `<input type="button">`
- `<input type="checkbox">`
- `<input type="color">`
- `<input type="date">`
- `<input type="datetime-local">`
- `<input type="email">`
- `<input type="file">`
- `<input type="hidden">`
- `<input type="image">`
- `<input type="month">`
- `<input type="number">`
- `<input type="password">`
- `<input type="radio">`
- `<input type="range">`
- `<input type="reset">`
- `<input type="search">`
- `<input type="submit">`
- `<input type="tel">`
- `<input type="text">` (default value)
- `<input type="time">`
- `<input type="url">`
- `<input type="week">`

Module 1 - HTML | Working with HTML

```
<label>This is text label This is text label This is text label This is text label
This is text label This is text label This is text label This is text label This is
text label This is text label </label>


<textarea rows="4" cols="50">This is text label This is text label This is text label
This is text label This is text label This is text label This is text label This is
text label This is text label This is text label </textarea>
```

Module 1 - HTML | Working with HTML

```
<label for="cars">Choose a car:</label>

     <select name="cars" id="cars">

          <option value="volvo">Volvo</option>

          <option value="saab">Saab</option>

          <option value="mercedes">Mercedes</option>

          <option value="audi">Audi</option>

     </select>
```

Module 1 - HTML | Working with HTML

```
<label for="cars">Choose a car:</label>

     <select  name="cars" id="cars">

       <optgroup label="Swedish Cars">

         <option value="volvo">Volvo</option>

         <option value="saab">Saab</option>

       </optgroup>

       <optgroup label="German Cars">

         <option value="mercedes">Mercedes</option>

         <option value="audi">Audi</option>

       </optgroup>

     </select>
```

Module 1 - HTML | Working with HTML

```
<fieldset>
    <legend>Personalia:</legend>
    <label for="fname">First name:</label>
    <input type="text" id="fname" name="fname"><br><br>
    <label for="lname">Last name:</label>
    <input type="text" id="lname" name="lname"><br><br>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email"><br><br>
    <label for="birthday">Birthday:</label>
    <input type="date" id="birthday" name="birthday"><br><br>
    <input type="submit" value="Submit">
</fieldset>
```

Module 1 - HTML | Creating some HTML pages

Task

- **Sign up page**
  a. What all do we need?
- **Login page**
  a. What all do we need?
- **Profile view page**
  a. What all do we need?
- **Profile edit page**
  a. What all do we need?

Module 1 - HTML | Creating some HTML pages

Task

- **Sign up page**
  a. User perspective
     i. Name
     ii. Username
     iii. Email id
     iv. Contact number
     v. DOB
     vi. Gender
     vii. Address
     viii. Password (abc123 -> avc124)& Confirm password (abc123)
     ix. Already have an account? Login here
  b. Developer’s perspective
     i. Form, Submit button, Labels, Input elements, date picker, Radio button
     ii. Fieldset
- **Login page**
  a. What all do we need?
- **Profile view page**
  a. What all do we need?
- **Profile edit page**
  a. What all do we need?

Module 1 - HTML | Creating some HTML pages

Task

- **Sign up page**
- **Login page**
    a. What all do we need?
        i. User's Perspective
            1. Email Id/Username
            2. Password
            3. Remember me next time
            4. Don't have an account? Sign up
        ii. Developer's Perspective
            1. Form, fieldset (optional)
            2. Input tag (email, Password, text, checkbox, & Button)
- **Profile view page**
    a. What all do we need?
- **Profile edit page**
    a. What all do we need?

Module 1 - HTML | Creating some HTML pages

Task

- **Login page**
- **Sign up page**
- **Profile view page**
  a. User perspective
     i. Name
     ii. Profile Picture
     iii. Username
     iv. Email id
     v. Contact number
     vi. DOB
     vii. Gender
     viii. Address
     ix. Hobbies
     x. Last Education
     xi. Edit Profile
  b. Developer’s
     i. img, a, labels, div, Button
- **Profile edit page**

Module 1 - HTML | Creating some HTML pages

- **Login page**
- **Sign up page**
- **Profile view page**
- **Profile edit page**
    a. User perspective
        i. Name
        ii. Profile Picture
        iii. Username
        iv. Email id
        v. Contact number
        vi. DOB
        vii. Gender
        viii. Address
        ix. Hobbies
        x. Last Education
        xi. Update profile button
    b. Developer’s perspective
        i. Input [text, date, submit, tel, radio/select-option,], img, fieldset (optional)

Module 1 - HTML | Creating some HTML pages

- Login page
- Sign up page
- Profile view page
- Profile edit page
- Now, What should be the flow among these pages?

Module 1 - HTML | Creating some HTML pages

- **Login page**
- **Sign up page**
- **Profile view page**
- **Profile edit page**
- **Now, What should be the flow among these pages?**
    a. Login -> Sign up -> Login -> View profile -> Edit profile
        i. Login -> View profile, Sign up
        ii. Sign up -> View profile, login
        iii. View Profile -> Edit profile, Login(Logout)

Module 1 - HTML | Working with HTML

HTML Styles

The HTML style attribute is used to add styles to an element, such as color, font, size, and more.

Syntax: `<tagname style="property:value;">`

Example:

```
----------------------------------------------------------------
<body style="background-color:powderblue;">
     <h1>This is a heading</h1>
     <p>This is a paragraph.</p>
</body>
----------------------------------------------------------------
<body>
     <h1 style="background-color:powderblue;">This is a heading</h1>
     <p style="background-color:tomato;">This is a paragraph.</p>
</body>
----------------------------------------------------------------
<h1 style="color:blue;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>
```

Module 1 - HTML | Working with HTML

HTML Styles

The HTML style attribute is used to add styles to an element, such as color, font, size, and more.

Syntax: `<tagname style="property:value;">`

Example:

```
-------------------------------------------------------------------------
<h1 style="font-family:verdana;">This is a heading</h1>
<p style="font-family:courier;">This is a paragraph.</p>
-------------------------------------------------------------------------
<h1 style="font-size:300%;">This is a heading</h1>
<p style="font-size:160%;">This is a paragraph.</p>
-------------------------------------------------------------------------
<h1 style="text-align:center;">Centered Heading</h1>
<p style="text-align:center;">Centered paragraph.</p>
```

Module 1 - HTML | Working with HTML

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.
Formatting elements were designed to display special types of text:
Elements:

`<b>` - Bold text
`<strong>` - Important text
`<i>` - Italic text
`<em>` - Emphasized text
`<mark>` - Marked text
`<small>` - Smaller text
`<del>` - Deleted text
`<ins>` - Inserted text
`<sub>` - Subscript text
`<sup>` - Superscript text

Module 1 - HTML | Working with HTML

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.
Formatting elements were designed to display special types of text:

Elements:
`<b>` - Bold text. The HTML `<b>` element defines bold text, without any extra importance

Example:
```
<b>This text is bold</b>
```

Module 1 - HTML | Working with HTML

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.
Formatting elements were designed to display special types of text:

Elements:

`<strong>` - Important text. The HTML `<strong>` element defines text with strong importance. The content inside is typically displayed in bold.

Example:

```
<strong>This text is important!</strong>
```

Module 1 - HTML | Working with HTML

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.
Formatting elements were designed to display special types of text:

Elements:
`<i>` - Italic text. The HTML `<i>` element defines a part of text in an alternate voice or mood. The content inside is typically displayed in italic. The `<i>` tag is often used to indicate a technical term, a phrase from another language, a thought, a ship name, etc.

`<em>` - Emphasized text. The HTML `<em>` element defines emphasized text. The content inside is typically displayed in italic.

Tip: A screen reader will pronounce the words in <em> with an emphasis, using verbal stress.

Example:

```
<i>This text is italic</i>
<em>This text is emphasized</em>
```

Module 1 - HTML | Working with HTML

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.
Formatting elements were designed to display special types of text:

Elements:

`<small>` - Smaller text. The HTML `<small>` element defines smaller text

Example:

```
<small>This is some smaller text.</small>
```

Module 1 - HTML | Working with HTML

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.
Formatting elements were designed to display special types of text:

Elements:
`<mark>` - Marked text. The HTML `<mark>` element defines text that should be marked or highlighted

Example:
```
<p>Do not forget to buy <mark>milk</mark> today.</p>
```

Module 1 - HTML | Working with HTML

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.
Formatting elements were designed to display special types of text:

Elements:

`<del>` - Deleted text. The HTML `<del>` element defines text that has been deleted from a document. Browsers will usually strike a line through deleted text:

Example:

```
<p>My favorite color is <del>blue</del> red.</p>
```

Module 1 - HTML | Working with HTML

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.
Formatting elements were designed to display special types of text:

Elements:

`<ins>` - Inserted text. The HTML `<ins>` element defines a text that has been inserted into a document. Browsers will usually underline inserted text:

Example:

```
<p>My favorite color is <del>blue</del> <ins>red</ins>.</p>
```

Module 1 - HTML | Working with HTML

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.
Formatting elements were designed to display special types of text:

Elements:

`<sub>` - Subscript text. The HTML `<sub>` element defines subscript text. Subscript text appears half a character below the normal line, and is sometimes rendered in a smaller font. Subscript text can be used for chemical formulas, like H_2O:

`<sup>` - Superscript text. The HTML `<sup>` element defines superscript text. Superscript text appears half a character above the normal line, and is sometimes rendered in a smaller font. Superscript text can be used for footnotes, like WWW[1]:

Example:

```
<p>This is <sub>subscripted</sub> text.</p>
<p>This is <sup>superscripted</sup> text.</p>
```

Module 1 - HTML | Working with HTML

HTML Quotation & Citation Elements

Elements:

`<blockquote>` - Quotations

`<q>` - short quotation.

`<abbr>` - abbreviation or acronym

`<address>` - contact information

`<cite>` - title of a creative work

`<bdo>` - BDO stands for Bi-Directional Override. override the current text direction

Module 1 - HTML | Working with HTML

HTML Quotation & Citation Elements

Elements:
`<blockquote>` - Quotations
`<q>` - short quotation.
`<abbr>` - abbreviation or acronym

Example:

```
<p>Here is a quote from WWF's website:</p>
<blockquote cite="http://www.worldwildlife.org/who/index.html">
For 60 years, WWF has worked to help people and nature thrive. As the world's leading
conservation organization, WWF works in nearly 100 countries. At every level, we
collaborate with people around the world to develop and deliver innovative solutions
that protect communities, wildlife, and the places in which they live.
</blockquote>

<p>WWF's goal is to: <q>Build a future where people live in harmony with
nature.</q></p>

<p>The <abbr title="World Health Organization">WHO</abbr> was founded in 1948.</p>
```

Module 1 - HTML | Working with HTML

HTML Quotation & Citation Elements

Elements:
`<address>` - contact information
`<cite>` - title of a creative work
`<bdo>` - BDO stands for Bi-Directional Override. override the current text direction

Example:

```
<address>
Written by John Doe.<br>
Visit us at:<br>
Example.com<br>
Box 564, Disneyland<br>
USA
</address>

<p><cite>The Scream</cite> by Edvard Munch. Painted in 1893.</p>

<bdo dir="rtl">This text will be written from right to left</bdo>
```

Module 1 - HTML | Working with HTML

HTML Comments

HTML comments are not displayed in the browser, but they can help document your HTML source code.
Syntax:

```
<!-- Write your comments here -->
```

Example:

```
<!-- This is a comment -->
<p>This is a paragraph.</p>
<!-- Remember to add more information here -->
----------------------------------------------------------
<p>This is a paragraph.</p>
<!-- <p>This is another paragraph </p> -->
<p>This is a paragraph too.</p>
----------------------------------------------------------
<p>This <!-- great text --> is a paragraph.</p>
```

Module 1 - HTML | Working with HTML

HTML Favicon

A favicon is a small image displayed next to the page title in the browser tab.
https://www.favicon.cc Use this website to create your own favicon
To add a favicon to your website, either save your favicon image to the root directory of your webserver, or create a folder in the root directory called images, and save your favicon image in this folder. A common name for a favicon image is "favicon.ico"
Next, add a `<link>` element to your "index.html" file, after the `<title>` element, like this:
Favicon file format/extension: ico, png, gif, jpeg, svg
Example:

```
<!DOCTYPE html>
<html>
     <head>
          <title>My Page Title</title>
          <link rel="icon" type="image/x-icon" href="/images/favicon.ico">
     </head>
     <body>
          <h1>This is a Heading</h1>
          <p>This is a paragraph.</p>
     </body>
</html>
```

Module 1 - HTML | Working with HTML

HTML Div element

The `<div>` element is used as a container for other HTML elements.

The `<div>` element is by default a block element, meaning that it takes all available width, and comes with line breaks before and after

Example:

```
Lorem Ipsum <div>I am a div</div> dolor sit amet.
```

Result:

Lorem Ipsum
I am a div
dolor sit amet.

Module 1 - HTML | Working with HTML

HTML Div element

The `<div>` element is often used to group sections of a web page together.

Example:

```
<div>
  <h2>London</h2>
  <p>London is the capital city of England.</p>
  <p>London has over 9 million inhabitants.</p>
</div>

<div>
  <h2>Oslo</h2>
  <p>Oslo is the capital city of Norway.</p>
  <p>Oslo has over 700,000 inhabitants.</p>
</div>

<div>
  <h2>Rome</h2>
  <p>Rome is the capital city of Italy.</p>
  <p>Rome has over 4 million inhabitants.</p>
</div>
```

CSS

Module 2 - CSS with HTML | What is CSS & how it to use it?

What is CSS?

- CSS is the language we use to style a Web page.
- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- CSS separates data from design
    - HTML is used for sending the data to the browser & CSS is used to apply the design
- CSS can be coded as a style attribute in an HTML tag, a head section of a document or an external document
    - Best way to apply CSS design is to code CSS in external documents

Module 2 - CSS with HTML | What is CSS & how it to use it?

Why Use CSS?

- CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.
- CSS Solved a Big Problem
    - HTML was NEVER intended to contain tags for formatting a web page!
    - HTML was created to describe the content of a web page, like:

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

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

    - When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large websites, where fonts and color information were added to every single page, became a long and expensive process.
    - To solve this problem, the World Wide Web Consortium (W3C) created CSS.
    - CSS removed the style formatting from the HTML page!

Module 2 - CSS with HTML | CSS format

Basic Syntax

```
html-tag-name
{
        css-property-key-1: css-value-1;
        css-property-key-2: css-value-2;
}
```

Example:

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

`body` is a selector in CSS (It points to the HTML element you want to style: <body>)

`background-color` is a property, and `lightblue` is the property value

Module 2 - CSS with HTML | Elements CSS can control

- **CSS can control a document's appearance & specifies style rules for below elements:**
    a. Fonts
    b. Text
    c. Colors
    d. Backgrounds
    e. Sizes
    f. Borders
    g. Spacing
    h. Positioning
    i. Visual effects
    j. Tables
    k. Lists

Module 2 - CSS with HTML | Applying CSS to HTML

Methods to insert CSS:

- Inline CSS:
  - Used for single HTML element
  - HTML documents can get messy and bulky
  - Insert the style attribute inside any HTML element
- Internal CSS
  - Used for a single page
  - Increases load time
  - Add a <style> tag to your HTML document
- External CSS:
  - Used to style an entire website
  - Can be linked to from other pages
  - Add a <link> atg to the <head> tag

Module 2 - CSS with HTML | Applying CSS to HTML

Methods to insert CSS Example:

- Inline CSS:

```
<!DOCTYPE html>
<html>
     <body>
          <h1 style="color:blue;text-align:center;">This is a heading</h1>
          <p style="color:red;">This is a paragraph.</p>
     </body>
</html>
```

- Priority
    - Inline > Internal > External

Module 2 - CSS with HTML | Applying CSS to HTML

Methods to insert CSS Example:

- Internal CSS:

```
<!DOCTYPE html>
<html>
     <head>
          <style>
               body {
                 background-color: linen;
               }
               h1 {
                 color: maroon;
                 margin-left: 40px;
               }
          </style>
     </head>
     <body>
          <h1>This is a heading</h1>
          <p>This is a paragraph.</p>
     </body>
</html>
```

Module 2 - CSS with HTML | Applying CSS to HTML

Methods to insert CSS Example:

- External CSS:

```
<!DOCTYPE html>
<html>
      <head>
            <link rel="stylesheet" href="mystyle.css">
      </head>
      <body>
            <h1>This is a heading</h1>
            <p>This is a paragraph.</p>
      </body>
</html>
```

"mystyle.css"

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

h1 {
  color: navy;
  margin-left: 20px;
}
```

Module 2 - CSS with HTML | Applying CSS to HTML

Methods to insert CSS Example:

- Priority
    - Inline > Internal > External

Module 2 - CSS with HTML | CSS format

CSS Selectors

- Tags/Elements
    - <a>, <div>, <li>, <label>, & etc. Example:

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

- ID reference
    - Displayed with a preceding hash symbol (#). E.g: #id-of-html-tag. Example:

```
#para1 {
  text-align: center;
  color: red;
}
```

- Class reference
    - Displayed with a preceding dot/period symbol (.). E.g: .class-of-html-tag. Example:

```
.center {
  text-align: center;
  color: red;
}
```

    - You can also specify that only specific HTML elements should be affected by a class.

```
p.center {
  text-align: center;
  color: red;
}
```

Module 2 - CSS with HTML | CSS format

CSS Selectors

- CSS Universal Selector
    - The universal selector (*) selects all HTML elements on the page. Example:

```
* {
  text-align: center;
  color: blue;
}
```

Module 2 - CSS with HTML | CSS format

CSS Selectors

- CSS Grouping Selector
  - The grouping selector selects all the HTML elements with the same style definitions.
  - Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

```
h1 {
  text-align: center;
  color: red;
}

h2 {
  text-align: center;
  color: red;
}

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

  - Can be written as:

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

Module 2 - CSS with HTML | CSS example

```
body
{
	background-color: #000003;
	color: #000000;
	margin: 0;
	padding: 0;
	text-align: left;
	font-size: 100%;
	font-family: sans-serif;
}
```

Module 2 - CSS with HTML | Guidelines for base styles

- Color use Red-Green-Blue (RGB) hexadecimal light values
- Size use pixels, em, or a percentage
- Text can be aligned left, right, or center
- Floats can also be left or right
- Vertical alignments must be top, middle, or bottom
- Fonts can be any specific font or font family, such as serif, sans-serif, or monospace or even a downloadable font

Module 2 - CSS with HTML | CSS Layouts

- **Fluid layout**
  - The height & width of elements are flexible
  - The expansion or contraction is based on screen size
  - The elements are specified using percentage & ems
- **Fixed layout**
  - Specify the height & width of elements
  - Values remain the same regardless of screen size
  - Specify elements using pixels

Module 2 - CSS with HTML | Applying CSS to HTML

CSS Comments

A CSS comment is placed inside the `<style>` element, and starts with `/*` and ends with `*/`

Example:

```
/* This is a single-line comment */
p {
  color: red;
}
----------------------
p {
  color: red;  /* Set text color to red */
}
----------------------
p {
  color: /*red*/blue;
}
```

Module 2 - CSS with HTML | Common CSS atts & props

Basic Properties of CSS

- color - Text color
- background-color - Background color of the element
- font-size - Changing the font size in the element
- height - Setting the height of the element
- width - Setting the width of the element
- border - Setting up border for the element
- background image - Adding a background image to the element
- border-radius - Setting up border radius to the element
- margin (default, top, left, right, bottom)
- padding (default, top, left, right, bottom)

Module 2 - CSS with HTML | Common CSS atts & props

Basic Properties of CSS

- color
  - `color: #000000;                        /* Hex Value */`
  - `color: rgba(0, 0, 0, 0.3);                  /* RGBA */`
  - `color: black;                          /* Named Colors */`
- background-color
  - `background-color: #FFB300`
  - `opacity: 0.3;`
- font-size
  - `font-size: 48px;`
- height
  - `height: 250px;`
- width
  - `width: 500px;`
- border
  - `border: 5px solid #FF6DBA;`
- background image
  - `background-image: url("../img/rainbow_bg.jpg");`
- border-radius
  - `border-radius: 25px;`
- margin (default, top, left, right, bottom)
- padding (default, top, left, right, bottom)

Module 2 - CSS with HTML | Common CSS atts & props

Basic Properties of CSS

- margin (default, top, left, right, bottom)
    - The CSS margin properties are used to create space around elements, outside of any defined borders.
    - With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).
    - Example:

```
p {
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
}
p {
  margin: 25px 50px 75px 100px;
}
```

- padding (default, top, left, right, bottom)

Module 2 - CSS with HTML | Common CSS atts & props

Basic Properties of CSS

- margin (default, top, left, right, bottom)
- padding (default, top, left, right, bottom)
    - Padding is used to create space around an element's content, inside of any defined borders.
    - The CSS `padding` properties are used to generate space around an element's content, inside of any defined borders.
    - With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).
    - Example:

```
div {
  padding-top: 50px;
  padding-right: 30px;
  padding-bottom: 50px;
  padding-left: 80px;
}
div {
  padding: 25px 50px 75px 100px;
}
```

JS

Module 3 - JavaScript | Introduction to JavaScript

- **Nature of JavaScript**
  a. Derived from the ECMAScript standard
  b. Original designed to run on Netscape Navigator
  c. Not related to Java
     i. Interpreted & compiled
  d. JavaScript interpretes embedded in web browsers
     i. Add dynamic behaviour to otherwise static web content
  e. Core feature of Asynchronous JavaScript and XML (Ajax)
     i. Works with HTML, CSS, XML, & JSON

Module 3 - JavaScript | Introduction to JavaScript

- **JavaScript Primitives**
    a. Primitives are values & have no properties or methods:
        i. **number**
        ii. **string**
        iii. **boolean**
        iv. **null**
        v. **undefined**

Module 3 - JavaScript | Introduction to JavaScript

- **JavaScript Primitives**
  a. Primitives are values & have no properties or methods:
    i. **number**
      1. Integers (decimal, octal, hexadecimal)
        a. Example: 16 (decimal), 020 (octal), 0x10 (hexadecimal)
      2. Floating point (decimal followed by decimal point & the decimal digits and/or an exponent)
        a. Example: 3.1412, 5e2
    ii. **string**
    iii. **boolean**
    iv. **null**
    v. **undefined**

Module 3 - JavaScript | Introduction to JavaScript

- **JavaScript Primitives**
  a. Primitives are values & have no properties or methods:
     i. **number**
        1. Integers (decimal, octal, hexadecimal)
           a. Example: 16 (decimal), 020 (octal), 0x10 (hexadecimal)
        2. Floating point (decimal followed by decimal point & the decimal digits and/or an exponent)
           a. Example: 3.1412, 5e2
     ii. **string**
        1. Enclosed by double quotes
        2. For example: “Hello world”, “” (empty string)
     iii. **boolean**
     iv. **null**
     v. **undefined**

Module 3 - JavaScript | Introduction to JavaScript

- **JavaScript Primitives**
    a. Primitives are values & have no properties or methods:
        i. **number**
            1. Integers (decimal, octal, hexadecimal)
                a. Example: 16 (decimal), 020 (octal), 0x10 (hexadecimal)
            2. Floating point (decimal followed by decimal point & the decimal digits and/or an exponent)
                a. Example: 3.1412, 5e2
        ii. **string**
            1. Enclosed by double quotes
            2. For example: “Hello world”, “” (empty string)
        iii. **boolean**
            1. True or False
        iv. **null**
        v. **undefined**

Module 3 - JavaScript | Introduction to JavaScript

- **JavaScript Primitives**
  a. Primitives are values & have no properties or methods:
     i. **number**
        1. Integers (decimal, octal, hexadecimal)
           a. Example: 16 (decimal), 020 (octal), 0x10 (hexadecimal)
        2. Floating point (decimal followed by decimal point & the decimal digits and/or an exponent)
           a. Example: 3.1412, 5e2
     ii. **string**
        1. Enclosed by double quotes
        2. For example: “Hello world”, “” (empty string)
     iii. **boolean**
        1. True or False
     iv. **null**
        1. Represented by null
     v. **undefined**
        1. A data type has not been assigned, or the variable does not exist.

Module 3 - JavaScript | Introduction to JavaScript

- **Wrapper Objects**
    a. Some of the primitive data types have corresponding wrapper objects
    b. Allows an object of the related type to be created
    c. Stores a primitive value and offers methods with which to process it
    d. Wrapper objects have the same name as the primitive type, but they begin with a capital letter

Primitive Type	**Wrapper Object**
a. boolean	a. Boolean
b. number	b. Number
c. string	c. String

Module 3 - JavaScript | Introduction to JavaScript

- **Wrapper Objects with example**
    a. The keyword new is used to create instance of the wrapper objects.
    b. JavaScript readily converts between wrapper objects & primitives
        i. typeof “abc”; // “string” (primitive)
        ii. typeof String(“abc”); // “string”
        iii. typeof new String(“abc”); // “object”
        iv. typeof (new String(“abc”)).valueof(); // “string”

Module 3 - JavaScript | Introduction to JavaScript

- **Comparison operator**
    a. ==
    b. ===
    c. !=
    d. >
    e. <
    f. >=
    g. <=

Module 3 - JavaScript | Introduction to JavaScript

- **Arithmetic operators**
  a. + (Arithmetic, String concatenation)
  b. -
  c. *
  d. /
  e. **
  f. %

Module 3 - JavaScript | Introduction to JavaScript

- **Assignment Operators**
  a. =
  b. +=
  c. -=
  d. *=
  e. /=
  f. **=
  g. %=

Module 3 - JavaScript | Introduction to JavaScript

- **Logical operator**
  a. &&
  b. ||
  c. !

Module 3 - JavaScript | Introduction to JavaScript

- **Short-Circuit Evaluation**

Short-circuit evaluation is a concept in which the compiler will skip checking sub-expressions in a compound statement (a statement with logical operators) once the value is determined.

- exp1 && exp2 will not evaluate exp2 if exp1 is **false** because if even one expression is false with an &&, the entire expression is false
- exp1 || exp2 will not evaluate exp2 if exp1 is **true** because if even one expression is true with an ||, the entire expression is true

Module 3 - JavaScript | Introduction to JavaScript

- **Creating a variable using let, var, const**
- **Control statements**
  a. if - else
  b. switch
- **Loops**
  a. for loop
  b. while loop
- **Functions**

Module 3 - JavaScript | Introduction to JavaScript

- **Creating a variable using let, var, const**
  a. let is used for local scopes mostly
  b. var is used for global scopes mostly
  c. let variables can't be re-declared
  d. var variables can re-declared
- **Control statements**
  a. if - else
  b. switch
  c. for loop
     i. Standard *for(var const =0; count<10; count++) {}*
     ii. for - in
  d. while loop
- **Functions**

Module 3 - JavaScript | Introduction to JavaScript

- **Control statements**
  a. if - else
  b. switch
  c. for loop
  d. while loop
- **Functions**

Module 3 - JavaScript | Introduction to JavaScript

- **Functions**
  a. Functions
  b. Anonymous functions
  c. Arrow functions
  d. Anonymous arrow functions
  e. Functions used as a variable

Module 3 - JavaScript | Introduction to JavaScript

- **Class**
  a. Class makes object-oriented programming feasible
  b. Class is a template or blueprint for creating objects
  c. Classed in JavaScript are built on prototype
- **Class - Constructors**
  a. A constructor is a method called when you want to create an object of class
  b. The body of a class is in curly brackets
  c. The code written here is subject to stricter syntax for increased performance

Module 3 - JavaScript | Introduction to JavaScript

```
class Student
{
        Name
        Age
        RollNo
        Email

        constructor(Name, Age, RollNo, Email)
        {
                this.Name = Name
                this.Age = Age;
                this.RollNo = RollNo;
                this.Email = Email;
        }

        printStudent()
        {
                console.log("Name: "+this.Name)
                console.log("Age: "+this.Age)
                console.log("RollNo: "+this.RollNo)
                console.log("Email: "+this.Email)
        }
}
```

```
let student = new Student()
student.Age = 11
student.Name = "Aditya"
student.RollNo = 111
student.Email = "abc@student.com"

let aditya = new Student("Gupta", 22,
101, "email@gmail.com")
aditya.printStudent()
console.log("student :"+student)
```

Module 3 - JavaScript | Introduction to JavaScript

- **Prototypes**
  a. Defines properties and methods for all instances of the object
  b. Exists for all JavaScript object that can be constructed with the new keyword
  c. Objects inherit all properties & methods from their prototype
  d. Prototype properties & methods can be changed in one call
  e. Adding new properties/methods to object requires modifying the object's code
  f. Adding new properties/methods to prototype can be done with one call
  g. Any object that gets instantiated inherits the current state of the prototype
  h. Changes made to a prototype causes all objects using it to automatically inherit the new properties and functions within that prototype
  i. Change to the an object's prototype affects all the instances of the class objects

Module 3 - JavaScript | Introduction to JavaScript

- **Array Objects**
  a. Accessible by indexed keys
  b. Use a zero-based indexing scheme
  c. Grow or shrink dynamically by adding or removing elements
  d. Length property is the largest integer index in the array
- **Declaring Array Objects**
  a. Declaration of an array can either an array literal or an array constructor
  b. Array constructor use the new array keyword & specify the array elements as parameters
     i. numArray = new array(0, 1, 2, 3, 4, 5)
     ii. numArray.length //returns 6
  c. Array literals create an array and initialize the elements in the array
     i. colors = [‘red’, “yellow”, “green”]

Module 3 - JavaScript | Introduction to JavaScript

- **Array functions**
  a. push
     i. *arr_name*.push(*value*)
  b. pop
     i. *arr_name*.pop()
  c. length
     i. *arr_name*.length
  d. indexOf
     i. *arr_name*.indexOf(*item*)
  e. lastIndexOf
     i. *arr_name*.lastIndexOf(*item*)
  f. entries
     i. *arr_name*.entries()
  g. find
     i. Array.find(<arrElemet>=>{ //return boolean based on a condition }
  h. filter
     i. Array.filter(<arrElemet>=>{ //return boolean based on a condition }
  i. map
     i. Array.map(<arrElemet>=>{ //return processed value }
  j. concat
     i. *arr_name*..concat(arr1.name);

Module 3 - JavaScript | Introduction to JavaScript

- **Array functions**
  a. find
     i.
```
Array.find(<arrElemet>=>{ //return boolean based on a condition }
let myarr = ["Mercury","Venus","Earth","Mars"];
let found = myarr.find(val=>
{
            return val.includes("s");
})
console.log(found);
```
  b. filter
     i.
```
Array.filter(<arrElemet>=>{ //return boolean based on a condition }
let myarr = ["Mercury","Venus","Earth","Mars"];
let found = myarr.filter(val=>
{
            return val.includes("s");
})
console.log(found);
```
  c. map
     i.
```
Array.map(<arrElemet>=>{ //return processed value }
let myarr = ["name","place","thing","animal"];
let found = myarr.map(val=>
{
            return val+"s";
})
console.log(found);
//Output [ 'names', 'places', 'things', 'animals' ]
```
  d. concat
     i.
```
arr_name..concat(arr1.name);
let hello = ["hello", "world" ];
let lorem = ["along","lorem"]
let h = hello.concat(lorem);
Output is  [ "hello" ,  "world" ,  "along" ,  "lorem" ]
```
  e. Splice
     i. receives - index, number elements to remove, element(s)
     ii. example: array.splice(index, number of elements to remove, [elements, elements…])

Module 3 - JavaScript | Introduction to JavaScript

- **String functions**
  a. length
     i. *string_obj*.length
  b. split
     i. *string_obj*.split(*separator*)
  c. charAt
     i. *string_obj*.charAt(*index*)
  d. replace
     i. *string_obj*.replace(*"SearchValue"*, *"NewValue"*)
  e. substring
     i. *string_obj*.substring(start, end)
  f. startswith
     i. *string_obj*.startsWith(searchvalue)
  g. endswith
     i. *string_obj*.endsWith(searchvalue))
  h. toUpperCase
     i. *string_obj*.toUpperCase()
  i. toLowerCase
     i. *string_obj*.toLowerCase()
  j. concat
     i. *string_obj*.concat(*string1*, *string2*,..,*stringN*)

Module 3 - JavaScript | Introduction to JavaScript

- **Map functions**
  a. set
     i. mapName.set(key,value);
  b. get
     i. mapName.get(key);
  c. keys
     i. mapName.keys();
  d. values
     i. mapName.values();
  e. has
     i. mapName.has(key_name);
  f. delete
     i. mapName.delete(key_name);
  g. size
  h. Clear
  i. forEach()

Module 3 - JavaScript | Introduction to JavaScript

- **Map functions**
  a. set
     i. var newMap = new Map();
     ii. newMap.set("h", 1);
     iii. Output is { “h” => 1}
  b. get
     i. mapName.get(key);
     ii. var newMap = new Map();
     iii. newMap.get("h");
     iv. Output is Map(0) {size: 0}
  c. keys
     i. mapName.keys();
  d. values
     i. mapName.values();
  e. has
     i. mapName.has(key_name);
  f. delete
     i. mapName.delete(key_name);

Module 3 - JavaScript | Introduction to JavaScript

- **Date Objects**
  a. The Date object is a snapshot of an exact millisecond in time
     i. Represented in the format YYYY-MM-DD 00:00:00
  b. There are number of ways to provide parameters to the Data constructor
     i. Example without a parameter
        1. var today = new Date(); // returns the local date & time
     ii. JavaScript automatically applies the toString() method if you attempt to display the date on a page or alert box

Module 3 - JavaScript | Introduction to JavaScript

- **Error Objects**
  a. Error objects instances are created when an exception is thrown
    i. Properties of the error object instance contain information about the error
    ii. message: A description of the message
    iii. name: Identifies the type of error, such as TypeError, RangeError, or URIError, EvalError, ReferrenceError, & SyntaxError
    iv. Custom error:
      1. throw new Error(“only values 1 - 10 are permitted”)

Module 3 - JavaScript | JavaScript APIs

- **JavaScript APIs**
    a. The common APIs used in JavaScript applications are:

Document	**Element**	**Window**
document.getElementById(id)	element.innerHTML	window.open
document.getElementByTagName(name)	element.style	window.onload
document.createElement(name)	element.setAttribute	window.scrollTo
parentNode.appendChild(node)	element.getAttribute	

Module 3 - JavaScript | JavaScript APIs

- **Retrieving a node reference**
  a. Use document.getElementById to:
     i. Pass the od value as an argument
     ii. Return one specific HTML or XML element that is based on the id attribute in the node
  b. Use document.getElementsByTagName(tagName)
     i. Retrieves a NodeList of element with a specific tag name
     ii. Nodelist contains an array of elements in the document
     iii. tagName parameter is the literal name of the HTML tag

JavaScript | JavaScript engine

1. A JavaScript engine takes JavaScript code [Human readable] & converts it into machine code [Machine readable]
2. When you write JavaScript code and run it in a browser, the code doesn't directly interact with your computer's hardware. Instead, it interacts with the JavaScript engine, which acts as an intermediary between your code and the underlying machine.
3. Every browser has its own JS engine, but the most well known one is Google's v8 Engine. This v8 Engine powers Google Chrome and also Node.js (used to build server side applications).
4. JS used to be a purely interpreted language. But the modern JS engine now use a mix of compilation and interpretation which is known as "just-in-time" compilation

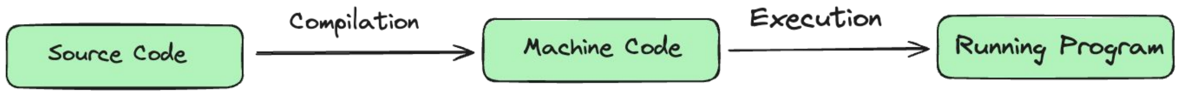
JavaScript | JavaScript engine

1. JS used to be a purely interpreted language. But the modern JS engine now use a mix of compilation and interpretation which is known as "just-in-time" compilation
2. In JIT compilation, the entire code is converted into machine code at once and then executed immediately.
3. Difference between the compilation and JIT.
    a. After compilation, the machine code is stored in a portable file. It can be executed at any time – there's no need to rush immediately after the compilation process.
    b. But in the case of JIT, the machine code needs to execute as soon as the compilation ends.

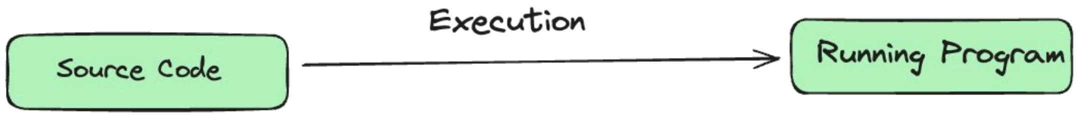
JavaScript | JavaScript engine

1. **Compilation vs Interpretation**

   In compilation, the entire code is converted into machine code at once and written in a binary file that can be executed by a computer.



   In interpretation, the interpreter runs through the source code and executes it line by line. The code still needs to get converted into machine code, but this time it is happening line by line while executing the program.



JavaScript | JavaScript engine

1. JIT & JavaScript
    a. Whenever a piece of JavaScript code enters the engine, the first step is to parse the code.
    b. During this parsing process, the code is parsed into a data structure called the AST (Abstract Syntax Tree). This works by first splitting up each line of code into pieces that are meaningful to the language (like the const or function keywords), and then saving all these pieces into the tree in a structured way.
    c. This step also checks if there are any syntax errors. The resulting tree will later be used to generate the machine code.

JavaScript | JavaScript engine

1. JIT & JavaScript Example
   a. `const greet = "Hello";`
   b. AST (Abstract Syntax Tree)

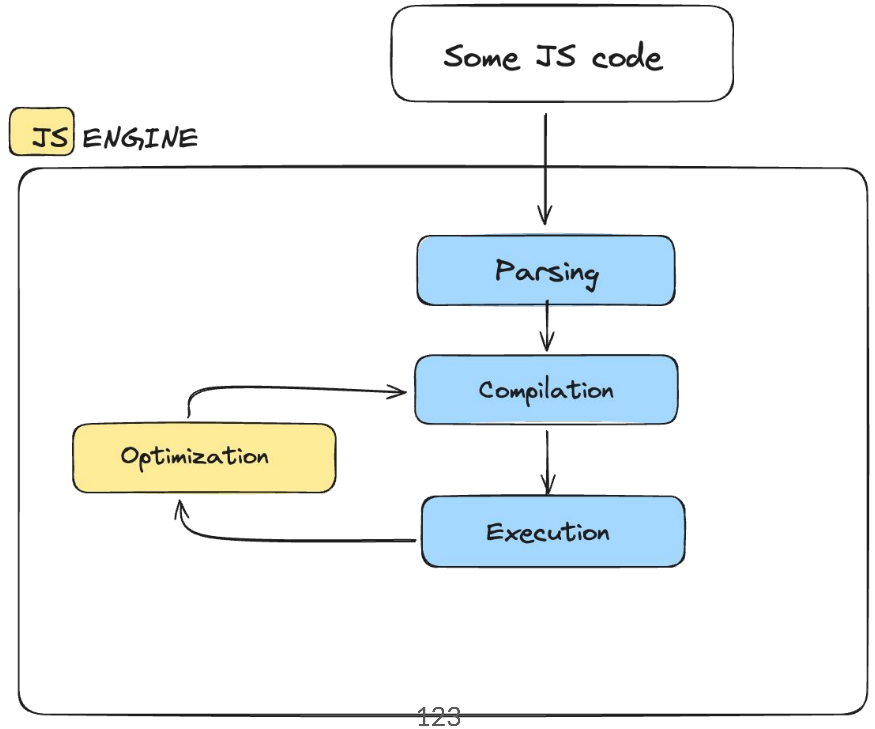
```
{
  "type": "Program",
  "start": 0,
  "end": 201,
  "body": [
    {
      "type": "VariableDeclaration",
      "start": 179,
      "end": 200,
      "declarations": [
        {
          "type": "VariableDeclarator",
          "start": 185,
          "end": 200,
          "id": {
            "type": "Identifier",
            "start": 185,
            "end": 190,
            "name": "greet"
          },
          "init": {
            "type": "Literal",
            "start": 193,
            "end": 200,
            "value": "Hello",
            "raw": "\"Hello\""
          }
        }
      ],
      "kind": "const"
    }
  ],
  "sourceType": "module"
```

JavaScript | JavaScript engine

1. JIT & JavaScript Example
   a. The next step is compilation. Here the engine takes the AST and compiles it to machine code.
   b. Then, this machine code gets executed immediately because its using JIT – remember that this execution happens in the call stack.
   c. But this is not the end. The modern JS engine generates inefficient machine code just to execute the program as fast as possible. Then the engine takes the already pre-compiled code to optimise and recompile the code during the already running program execution. All this optimisation happens in the background

JavaScript | JavaScript engine

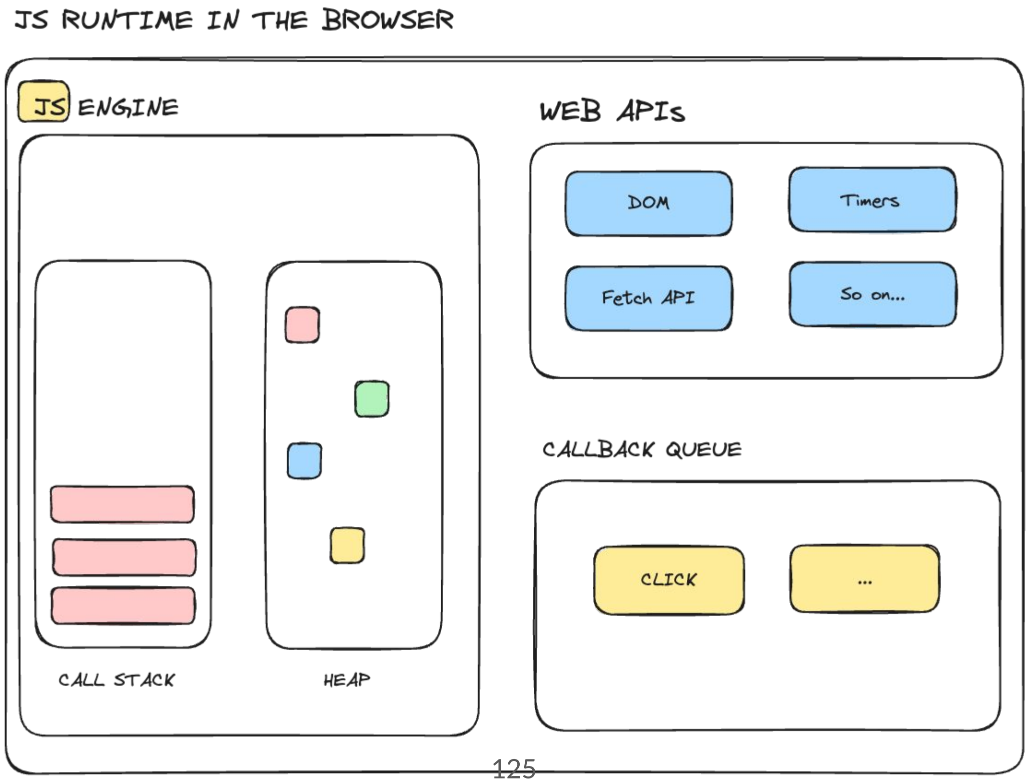
1. JIT & JavaScript Example



JavaScript | JavaScript runtime

1. A JavaScript (JS) runtime is a comprehensive environment that enables the execution of JavaScript code. It consists of various components working together to facilitate the execution of JavaScript applications. When we refer to a JS runtime, we're typically talking about the entire ecosystem that extends beyond the basic execution of code.
2. JS runtimes, especially in the context of web browsers, come with Web APIs that provide additional functionalities beyond the core JavaScript language. These APIs include interactions with the Document Object Model (DOM), XMLHttpRequest (for making HTTP requests), timers, and more.
3. So, basically Web APIs are functionalities that are provided to the engine but they are not a part of the JavaScript language itself. JavaScript gets access to these APIs through the `window` object.
4. Asynchronous operations in JavaScript, such as handling user input or making network requests, utilize callback functions. These functions are placed in a queue known as the callback queue, awaiting execution. The callback queue ensures that asynchronous tasks are handled in an organized manner.

JavaScript | JavaScript runtime



JavaScript | JavaScript runtime

For example, we attach event handler functions to DOM events like a button to react to certain events. These event handler functions are also called callback functions. So, as the click event happens, the callback function will be called

Here is how it works behind the scenes:

- First, after the event occurs, the callback function is put into the callback queue.
- Then, when the call stack is empty, the callback queue gets passed to the call stack so that it can be executed and this happens by something called the Event Loop.
- So, in short, the event loop takes callback functions from the callback queue and puts it into the call stack, so that it can be executed.

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
     i. **Add to-do**
     ii. Show list of to-do
     iii. Delete to-do
     iv. Complete a to-do
     v. Edit a to-do

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
    a. Features
        i. Add to-do
            1. Components needed
                a. Header
                b. Input box
                c. Button to add a todo
        ii. Show to-do list
        iii. Delete to-do
        iv. Complete a to-do
        v. Edit a to-do

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
     i. Add to-do
        1. Components needed
           a. Header
           b. Input box
           c. Button to add a todo
        2. Actions
           a. Handle button click to add a todo to list

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
    i. Add to-do
      1. **Components needed**
        a. **Header**
        b. **Input box**
        c. **Button to add a todo**
- **Adding HTML Code**

```
<html>
    <head>
        <h1>To-Do Application</h1>
    </head>
    <body>
           <input type="text" placeholder="Enter a to-do" id="todoText"/>
          <button >Add To-do</button>
    </body>
</html>
```

- **Link to Code: Click here**

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let's create a sample To-do app
  a. Features
     i. Add to-do
        1. **Actions**
           a. **Handle button click to add a todo to list**

- **HTML Changes:**

  <button **onclick="addTodo()"**>Add To-do</button>

- **Link a JS file:**

```
 <head>
        <script src="appJS.js" type="text/javascript"></script>
        <h1>To-Do Application</h1>
```

- **Link to Code: Click here**

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
     i. Add to-do
        1. Actions
           a. Handle button click to add a todo to list

- **JS Code to add in your JS file:**

```
function addTodo(event)
{
    console.log("-------------------addTodo-------------------")
    let tempTodoText = document.getElementById("todoText").value //getting text from Input
    console.log("Text added: "+tempTodoText)
    console.log(todo)
    todo.push(tempTodoText)
    console.log(todo)
    document.getElementById("todoText").value = ""
}
```

- **Link to HTML Code: Click here**
- **Link to JS Code: Click here**

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let's create a sample To-do app
  a. Features
     ~~i. Add to-do~~
        ~~1. Components needed~~
           ~~a. Header~~
           ~~b. Input box~~
           ~~c. Button to add a todo~~
        ~~2. Actions~~
           ~~a. Handle button click to add a todo to list~~

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
     i. ~~Add to-do~~
     ii. Show to-do list
     iii. Delete to-do
     iv. Complete a to-do
     v. Edit a to-do

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let's create a sample To-do app
    a. Features
        ~~i. Add to-do~~
        ii. Show to-do list
            1. Frontend
                a. A List to show todo items
            2. Backend
                a. Dynamically showing the list of todos in list

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
     ~~i. Add to-do~~
     ii. Show to-do list
        1. Frontend
           a. A List to show todo items
        2. Backend
           a. Dynamically showing the list of todos in list

- **Update HTML body with list ([Link to HTML File](#))**

```
<body>
       <input type="text" placeholder="Enter a to-do" id="todoText"/>
       <button onclick="addTodo()">Add To-do</button>
       <ul id="todoList">
       </ul>
   </body>
```

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
     ~~i. Add to-do~~
     ii. Show to-do list
        ~~1. Frontend~~
           ~~a. A List to show todo items~~
        2. Backend
           a. Dynamically showing the list of todos in list

- **Update JS for dynamic list (Link to JS File)**

```
function addTodo(event)
{
   console.log("-------------------addTodo-------------------")
   let tempTodoText = document.getElementById("todoText").value //getting text from Input
   console.log("Text added: "+tempTodoText)
   console.log(todo)
   todo.push(tempTodoText)
   console.log(todo)
   document.getElementById("todoText").value = ""
   //updating Frontend
   todoList.innerHTML = ""
   todo.forEach(element=>
   {
      todoList.innerHTML +=
            "<li>"+
               element.text+
            "</li>"
   }
}
```

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
     ~~i. Add to-do~~
     ~~ii. Show to-do list~~
     iii. Delete to-do
        1. Components needed
           a. Add a button
        2. Actions
           a. Handle button click to delete a todo from list
     iv. Complete a to-do
     v. Edit a to-do

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
     ~~i. Add to-do~~
     ~~ii. Show to-do list~~
     iii. Delete to-do
        1. Components needed
           a. Add a button

- **Update JS for delete button for each list item ([Link to JS File])**
  - Convert array items from text to an object, so we can assign an ID to each element of the array

Update addTodo function

```
let tempTodo = {
id: counter++,
text: tempTodoText,
completed: false,
}
todo.push(tempTodo)
```

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
    ~~i. Add to-do~~
    ~~ii. Show to-do list~~
    iii. Delete to-do
      1. Components needed
        a. Add a button

- **Update JS for delete button for each list item (Link to JS File)**

```
todoList.innerHTML +=
"<li>"+
        element.text+
        "  <button onclick=\"deleteTodo("+element.id+")\">Delete</button>"+
"</li>"
```

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let's create a sample To-do app
    a. Features
        ~~i. Add to-do~~
        ~~ii. Show to-do list~~
        iii. Delete to-do
            ~~1. Components needed~~
                ~~a. Add a button~~
            2. Actions
                a. Handle button click to delete a todo from list

- **Update JS for delete button for each list item (Link to JS File)**

```
function deleteTodo(id)
{
   console.log("------------------Delete todo------------------")
   console.log("id: "+id)
   todo = todo.filter(element=>{
      //return true / false
      return element.id != id
   })

   updateFrontend() // we moved frontend updating related code to a new function for re-use
}
```

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
  a. Features
     ~~i. Add to-do~~
     ~~ii. Show to-do list~~
     ~~iii. Delete to-do~~
     iv. Complete a to-do
        1. Components needed
           a. Check box
        2. Actions
           a. Manage state of Checkbox
           b. Handle checkbox action
     v. Edit a to-do

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let's create a sample To-do app
  a. Features
     ~~i. Add to-do~~
     ~~ii. Show to-do list~~
     ~~iii. Delete to-do~~
     iv. Complete a to-do
        1. Components needed
           a. Check box

- **Update JS for checkbox for each list item (Link to JS File)**

Update dynamic HTML

```
          todo.forEach(element=>
          {
                    if(element.completed)
                    {
                              todoList.innerHTML +=
                              "<li>"+
                                        "<input type=\"checkbox\" checked onclick=\"checkboxListener("+element.id+")\"></input><s>"+
                              element.text+
                                        "</s>  <button onclick=\"deleteTodo("+element.id+")\">Delete</button>"+
                              "</li>"
                    }
                    else
                    {
                              todoList.innerHTML +=
                              "<li>"+
                                        "<input type=\"checkbox\" onclick=\"checkboxListener("+element.id+")\"></input>"+
                              element.text+
                                        "<button onclick=\"deleteTodo("+element.id+")\">Delete</button>"+
                              "</li>"

                    }
          }
```

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let's create a sample To-do app
  a. Features
     ~~i. Add to-do~~
     ~~ii. Show to-do list~~
     ~~iii. Delete to-do~~
     iv. Complete a to-do
        ~~1. Components needed~~
           ~~a. Check box~~
        2. Actions
           ~~**a. Manage state of Checkbox**~~
           b. Handle checkbox action

- **Update JS for checkbox click listener for each list item (Link to JS File)**

```
function checkboxListener(id)
{
   console.log("id: "+id)
   //console.log(todo)
   todo = todo.map(element=>
      {
         if(element.id == id)
         {
               element.completed = !element.completed
         }
         return element
      })
   console.log(todo)
   updateFrontend()
}
```

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let’s create a sample To-do app
    a. Features
        i. ~~Add to-do~~
        ii. ~~Show to-do list~~
        iii. ~~Delete to-do~~
        iv. ~~Complete a to-do~~
        v. Edit a to-do
            1. Components needed
                a. Edit button
                b. Input box for To-do
                c. Update button for saving it
            2. Actions
                a. Handle Edit button
                b. Handle Input box for To-do
                c. Handle Update button for saving it

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let's create a sample To-do app
    - a. Features
        - i. **Edit a to-do**
            - 1. **Components needed**
                - a. **Edit button**
                - b. **Input box for To-do**
                - c. **Update button for saving it**
            - 2. Actions
                - a. Handle Edit button
                - b. Handle Input box for To-do
                - c. Handle Update button for saving it

- **Add a global variable flag for identifying if we are editing a todo or not with default value of -1**

```
var editingFlag = -1
```

- **Check for the condition with editingFlag in the else of element.completed condition check**

```
else
{
        if(editingFlag == element.id)//Editing
        {
                todoList.innerHTML +=
                "<li>"+
                        "<input type=\"checkbox\"  onclick=\"checkboxListener("+element.id+")\"></input>"+
                        "<input type=\"text\" placeholder=\"Edit your todo\" value=\""+element.text+"\"/>"+
                        "  <button onclick=\"deleteTodo("+element.id+")\">Delete</button>"+
                        "  <button onclick=\"updateTodo()\">Save Todo</button>"+
                "</li>"
        }
        else  //Not editing
        {
                todoList.innerHTML +=
                "<li>"+
                        "<input type=\"checkbox\"  onclick=\"checkboxListener("+element.id+")\"></input>"+
                        element.text+
                        "  <button onclick=\"deleteTodo("+element.id+")\">Delete</button>"+
                        "<button onclick=\"editTodo("+element.id+")\">Edit</button>"+
                "</li>"
        }
}
```

Module 3 - JavaScript - Create a Simple HTML, CSS & JS App

- Let's create a sample To-do app
    a. Features
        i. **Edit a to-do**
            ~~1. Components needed~~
                ~~a. Edit button~~
                ~~b. Input box for To-do~~
                ~~c. Update button for saving it~~
            2. **Actions**
                a. **Handle Edit button**
                b. **Handle Input box for To-do**
                c. **Handle Update button for saving it**

- **Add Edit button handler function**

```
function editTodo(id)
{
        console.log("editTodo")
        editingFlag = id
        updateFrontend()
}
```

- **Add update todo function for save button**

```
function updateTodo()
{
        console.log("updateTodo")
        todo.map(element=>
        {
                if(element.id == editingFlag)
                {
                        element.text = document.getElementById("editTodo").value
                        return element
                }
                else
                        return element
        })
        editingFlag = -1
        updateFrontend()
}
```

M7 - JS Remaining topics

- Functions in array and objects

```
var array = [
   1,
   2,
   "Hello",
   function() { console.log("Hello"); },
]


var myObject = {
         numberOne: 1,
         numberTwo: 2,
         string: "Hello string",
         abc: function() { console.log("Hello function") },
}
```

M7 - JS Remaining topics - Object assignment

- Object assignment

```
const obj = {
        a: 1,
        b: 2,
}

const obj2 = Object.assign({}, obj)
```

M7 - JS Remaining topics - Hoisting

- **Hoisting**
    - Definition
        - The process whereby the interpreter appears to move the declaration of functions, variables, classes, or imports to the top of their scope, prior to execution of the code
        - Hoisting is JavaScript's default behavior of moving declarations to the top.
        - Hoisting is the default behavior of moving all the declarations at the top of the scope before code execution
    - Function hoisting
    - const anonymous function
    - var/let anonymous function

M7 - JS Remaining topics - Clouser

- **Clouser**
  - Functions that refer to variables declared by parent function still have access to those variables
  - Possible because of JavaScript’s scoping
  - Definitions
    - A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment)
    - JavaScript variables can belong to the local or global scope. Global variables can be made local (private) with closures.
    - JavaScript closure is a feature that allows inner functions to access the outer scope of a function. Closure helps in binding a function to its outer boundary and is created automatically whenever a function is created.

```
function makeHelloFunction()
{
   const message = "Hello!"

   function sayHello()
   {
      console.log(message)
   }

   return sayHello
}

const sayHello = makeHelloFunction()

//console.log(message) //Error

sayHello()
```

M7 - JS Remaining topics - IIFE

- IIFE (Immediately Invoked Function Expression)
  - Self-executing functions
    - Self-executing or auto-invocation functions:
      - Start running immediately after they have been declared
      - Functions and variables are isolated from the rest of the script
      - Can be anonymous (Unnamed) function
    - Used to initialize or to declare DOM elements on the page

```
(function (){
//statements
})();
```

JS Remaining topics - Coercion

- Coercion or Type casting
  - x = 42
  - const explicit = String(x)   //explicit
  - const implicit = x+””   //implicit