Notes Made By Mani-Icon

JAVA SCRIPT INDEPTH (By Sudhakar Sharma Sir)

Day 1 : JavaScript Introduction – 28/02/23

- JavaScript Language
- Plugin's
- Libraries
- Widgets
- API

JavaScript
https://www.ecma-international.org

- JavaScript is light weight Interpretted and Just-in-Time compiled programming language.
- Light weight refers to memory occupied and how heavy the application is.
- Interpretted refers to line by line translation.
- Compiled refers to traslating entire program, all lines of program are translated simultaneously at the same time.
    a) JIT compiled
    b) AOT compiled

- JIT [Just-in-Time] is the process where JavaScript is loaded into browser and compiled in browser.

- AOT [Ahead-of-Time] is the process where JavaScript is compiled and processed at application level.

- We can use various engines and compilers
    a) Ivy
    b) Babel
    c) Node
    d) V8

- JavaScript is a language, which is used
    a) Client Side : with HTML
    b) Server Side : with Node JS
    c) Database : MongoDB
    d) Animation Tools : Flash, 3DS Max, etc..

- JavaScript supports various programming techniques and approaches
    a) Structural Programming
    b) Functional Programming
    c) Imperative Programming
    d) Object Oriented Programming etc..

- JavaScript is not an OOP language, It supports only few features of OOP.

Evolution of JavaScript

- 1990's Tim Berners Lee introduced HTML and Web
- 1990's early browsers were Mosaic, Netscape
- These browsers used HTML as Markup language and ECMAScript as client side script.

- In early 1995 Netscape appointed "Brendan Eich" to develop a script for browser. [MDN]
    - ECMA International
    - MDN
- First it was named as "Moca" after that renamed as "Live Script".
- Script belongs - Netscape
- Netscape given the rights of maintaing script to a company called "Sun Micro Systems".
- Sun Micro System named the script as "Java-Script".
- 2000 Netscape stopped it services, JavaScript was given to ECMA
- 2014 JavaScript
- 2015 ECMA Version ES5 = ECMAScript 2015,...2022..Next
- Current Latest version of JavaScript is "ECMAScript 2022" [ES2022]
  [ES5, ES6,ES7, ES8...ES9, 2020, 2021, 2022]

Where to implement?
- HTML Client Side
- Node JS server side
- MongoDB database
- Flash, 3DS Max animations

What is the role of JavaScript with HTML?
- DOM Manipulations
    - Adding Elements into page
    - Remove Elements from Page
    - Update Data into Elements
- Client Side Validations
- Client Side Interactions
- Handling Plugin and Extentions

What is the role of JavaScript Server Side?
- Server Side Interactions
    a) Request
    b) Response
- Server side objects
    a) Memory
    b) OS
    c) Files
- Configuring API's
- Handling communication between client and database etc..

What is the role of JavaScript in Database?
- To handle CRUD Operations
- DBA

What is the role of JavaScript in Animations?
- Transitions
- Animations
- Keyframes
- 2D and 3D Games

What is the role of JavaScript in Mobile Apps?
- Toch events
- View Design
- Apache Cordova, Ionic, Native Script - JS from mobile

Setup Environment for JS

- WebPack
- Parcel

Day 2 : Environment Setup For JavaScript – 01/03/23

- What is JavaScript?
- What are the Versions of JavaScript?
- Where JavaScript is used?
- ECMA 2022

Setup Environment for JavaScript Project

1. Install "Node JS" on your PC

Node JS : Server Side Programming Library
Node Compiler : It is used to compile and run JavaScript programs.
NPM : Node Package Manager
[Yarn, Composer, NuGet, Bower etc..]

- https://nodejs.org/en/
- Download and Install 18x version

FAQ: What is Package Manager?
Ans : It is a software tool used by developers to install, update and remove libraries from project.

2. Check Node JS Version

```
C:\>node -v
C:\>npm -v
```

Note: Make sure that your PC is have Node version > 14 , NPM > 6

3. Download and Install "Visual Studio Code Editor"
- Editor provides an IDE [Integrated Development Environment]
- Build, Debug, Test, Deploy
"editorconfig.org"

https://code.visualstudio.com/

4. Open Visual Studio Code Editor and Install extentions

- Install "Live Server"
- Install "VsCode-Icons"

Setup A Project for JavaScript
---
1. Create a new folder on your PC for JavaScript Project

D:\JavaScript-Project

2. Open your Project folder in Visual Studio Code

3. Add following folders into project

a) public : It is used to keep all static resources
[html, images, text, pdf, docx, ppt, mp4,..]

b) src : It is used to keep all dynamic resource
[js, ts, css, sass, less etc..]

4. Open Terminal in VS Code [Terminal is command line to run commands]
- Change Power Shell to Command Prompt
- Run the command

> npm init [-y]

- It generates a file "package.json".
- package.json is a file that contains project meta data.

5. Install ESLint configuration, It is used to verify the code and report the issues in code. [issues related to coding standards]

> npm init @eslint/config

Day 3 : JavaScript Features and Issues, Integrating JS – 02/03/23

Environment Setup
- Node JS
  Node Compiler, NPM
- Visual Studio Code Editor
  Live Server
  Vscode-icons
- Create a new Project
- Setup Project
  > npm init [package.json]
  public
  src
  > npm init @eslint/config

JavaScript Client Side

- JavaScript is used client side with HTML.
- JavaScript is used for
  a) DOM Manipulations
    - Adding Elements
    - Removing Elements
    - Updating data into elements
  b) Handling browser objects
    - Location
    - Navigator
    - History
    - Document
    - Window
  c) Client Side Validations
    - Verifying Input
    - Ensure that contradictionary and unauthorized data is not stored into database.
    - Reduce burden on server

FAQ: What are the issues with JavaScript?
Ans:
- JavaScript is not a strongly typed language.

```
var  age = 23;
age = "John";       // valid
age = true;       // valid
```

- JavaScript is not implicitly strictly typed

```
x = 10;          // valid
```

- JavaScript is not an OOP language
  - Extensibility Issues
  - Code Level Security Issues
  - Reusability Issues

- JavaScript is not having features for dynamic polymorphism.

- JavaScript can be disabled by browser.

FAQ: What is solution?
Ans : TypeScript

FAQ: How JavaScript converts Static DOM into Dynamic DOM?
Ans : By Integrating JavaScript functions into page.

FAQ: How JavaScript can be integrated into Page?
Ans: In 3 different ways

a) Inline
b) Embedded
c) External File

Inline JavaScript:

- In this technique JavaScript functions are directly written in HTML elements start tag.

```
<button  onclick="window.print()"> Print </button>
```

- It is faster in responding.
- It is not good for reusability.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
</head>
<body>
   <h2>JavaScript - ES6</h2>
   <button onclick="window.print()">Print</button>
</body>
</html>
```

Embedded

- JavaScript functions are kept in a <script> container and can be accessed from any element.
- You can reuse the funcitons.
- The script container can be in <head> or <body>.
- You have to define functions in <script> container

```
<script>
    function  PrintPage(){
      window.print();
    }
</script>

<button onclick="PrintPage()"> Print </button>
```

- It is slower when compared to inline.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function PrintPage(){
         window.print();
      }
   </script>
</head>
<body>
   <h2>JavaScript - ES6</h2>
   <button onclick="PrintPage()">Print</button>
   <button onclick="PrintPage()">Print Page</button>
</body>
</html>
```

FAQ: What is difference between script in head and body?
Ans: Script in head section is intended to load into browser memory and later accessed by page when ever required.
Script in body section is intended to load into page directly, it is not in memory of browser.

Day 4 : Embedded And External Files – 03/03/23

1. JavaScript Inline
2. JavaScript Embedded

Ms-Excel	.xls, .xlsx, .csv	MIME	application/msexcel
Images	.jpg, .jpeg, .jfif	MIME	image/jpeg

FAQ: Where to embed in head or body?
FAQ: What is the MIME type of script? [Multi purpose Internet Mail Extentions]
Ans : JavaScript is used in various methods
a) Interpretted

b) Compiled

If your JavaScript is used with HTML in browser then MIME type is defined as

"text/javascript" - Interpretted
"text/babel" - JavaScript is used with babel compiler
"text/module" - JavaScript module system

Syntax:

```
<script  type="text/javascript">  </script>
<script  type="text/babel"> </script>
```

FAQ: What is strict mode for JavaScript?

Ans: JavaScript is not implicitly strictly typed. It will not follow programming rules and standards. You have to manually turn on "Strict Mode".

If strict mode is ON then developer have to write as per standards.

You can turn on strict mode by using "use strict"; in your code snippet.

EX:

```
<script>
   "use strict";
   x = 10;                          // invalid   - x is not defined   // valid if strict off
   document.write("x=" + x);
</script>
```

FAQ: How to target JavaScript for Legacy browsers? [Old Version Browsers]

Ans : Developer can target new JavaScript code to the legacy browser by enclosing the code in HTML comments.

Syntax

```
<script type="text/javascirpt">
   <!--
       "use strict";
      ......
      ......
   -->
</script>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript">
       <!--
         "use strict";
         function PrintPage(){
         window.print();
         }
       -->
   </script>
</head>
<body>
   <!--this is body section-->
   <h2>JavaScript Embedded</h2>
   <button onclick="PrintPage()">Print</button>
```

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

- Issue with embedded technique is
  a) It is slow
  b) You can't re-use across pages

JavaScript in External File

- JavaScript functions are written in a separate script file with extention ".js"

index.js

```
<!--
"use strict";
function PrintPage(){
   window.print();
}
-->
```

- You can link the script file to any HTML page by using <script> element.

```
<script  type="text/javascript" src="index.js"> </script>
```

- Features
  Clean spearation of code and design.
  Hard to test if it is embedded , [easy]
  Hard to extend if it is embedded , [easy]

- Issue
  Using a external file for HTML page will increase the number of requests.
  If number of requests are increased for page, then page load time will increase.

Ex: src/scripts
index.js

```
<!--
"use strict";
function PrintPage(){
window.print();
}
-->
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript" src="src/scripts/index.js">
   </script>
</head>
<body>
   <!--this is body section-->
   <h2>JavaScript External File</h2>
   <button onclick="PrintPage()">Print</button>
```

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

Minification

- It is coding technique used by developers to reduce the size of file. [Compress]
- Minified files are used in Production
- Unminified files are use is Development

Ex:

1. Visit any minification site "https://www.toptal.com/developers/javascript-minifier"

2. Paste your actual JS code

3. Minify

4. Copy minified code

5. Create a new file

   index.min.js

6. paste the minified code

7. link the minified file to page.

Bundling
[WebPack, Parcel]

Day 5 : About JavaScript – 04/03/23

1. FAQ: How JavaScript takes control over HTML elements?

   1. JavaScript can access HTML elements in page by using DOM hierarchy.
      window => document => images[], forms[]

   - It is faster in accessing elements.
   - It is the native method for JS.
   - If any element changes its position in page, then you have to update the index everytime.

Ex:

index.js

```
function bodyload(){
  window.document.images[0].src = "public/images/shoe.jpg";
  window.document.images[0].width = 200;
  window.document.forms[0].elements[0].value = "Register";
  window.document.forms[1].elements[1].value  = "Login";
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body onload="bodyload()">
    <img width="100" height="100" border="1">
    <div>
        <form>
            <h2>Reigster</h2>
            Your Email <input type="button"> <input type="email">
        </form>
    </div>
    <div>
        <form>
            <h2>Login</h2>
            Your Mobile : <input type="text"> <input type="button">
        </form>
    </div>
</body>
</html>
```

2. JavaScript can refer elements by using "name".

- Every element can be defined with a name.
- JavaScript can access element by using name.
- You can't access any child element directly.
- Everytime to access a child you have to refer its parent.
- HTML can have same name for multiple elements.

Ex:
index.js

```
function bodyload(){
    pic.src = "public/images/shoe.jpg";
    pic.width = 200;
    frmRegister.btnRegister.value = "Register";
    frmLogin.btnLogin.value = "Login";
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body onload="bodyload()">
    <img name="pic" width="100" height="100" border="1">
    <div>
        <form name="frmRegister">
            <h2>Reigster</h2>
            Your Email  <input type="email"> <input type="button" name="btnRegister">
        </form>
    </div>
```

```
    <div>
        <form name="frmLogin">
            <h2>Login</h2>
            Your Mobile : <input type="text"> <input name="btnLogin" type="button">
        </form>
    </div>
</body>
</html>
```

3. JavaScript can refer HTML elements by using ID

- JS provides a method "document.getElementById()"
- You can access any element from any level of hierarchy.
- ID reference have a confilct with CSS ID.
- In CSS same ID can be defined for multiple elements.

Ex:

index.js

```
function bodyload(){
    document.getElementById("pic").src = "public/images/shoe.jpg";
    document.getElementById("btnRegister").value = "Register";
    document.getElementById("btnLogin").value = "Login";
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body onload="bodyload()">
    <img name="pic" id="pic" width="100" height="100" border="1">
    <div>
        <form name="frmRegister">
            <h2>Reigster</h2>
            Your Email  <input type="email"> <input id="btnRegister" type="button" name="btnRegister">
        </form>
    </div>
    <div>
        <form name="frmLogin">
            <h2>Login</h2>
            Your Mobile : <input type="text"> <input id="btnLogin" name="btnLogin" type="button">
        </form>
    </div>
</body>
</html>
```

4. JavaScript can access HTML elements by using CSS selectors.

Primary Selectors
type, id, class
Rational Selectors
child, adjacent, siblings
Attriubte Selectors

Dynamic Pseudo classes
Structural Pseudo classes
Element state pseudo classes
Validation state pseudo classes

- JavaScript uses the method "document.querySelector()"

Ex:
index.js

```
function bodyload(){
   document.querySelector("img").src = "public/images/shoe.jpg";
   document.querySelector("#btnRegister").value = "Register";
   document.querySelector(".btn-login").value = "Login";
   document.querySelector("nav div img").src = "public/images/shoe.jpg";
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body onload="bodyload()">
   <img name="pic" id="pic" width="100" height="100" border="1">
   <nav>
      <div>
         <img width="300" height="200">
      </div>
   </nav>
   <div>
      <form name="frmRegister">
         <h2>Reigster</h2>
         Your Email  <input type="email"> <input id="btnRegister" type="button" name="btnRegister">
      </form>
   </div>
   <div>
      <form name="frmLogin">
         <h2>Login</h2>
         Your Mobile : <input type="text"> <input id="btnLogin" class="btn btn-login" name="btnLogin"
type="button">
      </form>
   </div>
</body>
</html>
```

5. JavaScript provides various techniques to access multiple elements

document.getElementsByTagName()
document.getElementsByClassName()
document.getElementsByName()

Day 6 : JavaScript Output Methods – 06/03/23

Summary
- Introduction to JavaScript
- Integrating JavaScript into HTML page [Client Side]
  a) Inline
  b) Embedded
  c) External Files
  d) Minification
- Strict Mode
- Legacy Browsers
- MIME Type
- JavaScript HTML reference methods
  a) DOM hierarchy
  b) By using Name
  c) By using ID
  d) By query selector

JavaScript output techniques
- It is the process of presenting data dynamically in browser.
- JavaScript provides various techniques

alert()
confirm()
document.write()
console methods
innerHTML
outerHTML
innerText

alert()
- It is used to display output in a message box.
- Message box pops-up in browser window.
- It comprises of only "OK", you can't cancel.
- Using "esc" key you can cancel.

Syntax:
```
alert("message");          // single line
alert("message\n line2");    // multiple lines
```

Ex:
index.js

```
function DeleteClick()
{
   alert("Delete\nRecord will be deleted");
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
```

```
   <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body>
  <button onclick="DeleteClick()">Delete</button>
</body>
</html>
```

confirm()

- It is similar to alert but allows to cancel.
- It is boolean method that returns
  - true = OK
  - false = Cancel

Syntax:

```
   confirm("Message\nLine2");       // true or false
```

Ex:

index.js

```
function DeleteClick()
{
  flag = confirm("Delete\nRecord will be deleted");
  if(flag==true){
    alert("Record Deleted..");
  } else {
    alert("You canceled..");
  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body>
  <button onclick="DeleteClick()">Delete</button>
</body>
</html>
```

document.write()

- It is an output method, which can display output on a new screen.
  [It is on same page but a new screen]

Syntax:

```
      document.write("message");       \\ no  "\n" for line break
      document.write(<markup>);       \\ <br>
```

Note: Markup is not allowed for alert() and confirm().

Ex:

index.js

```
function DeleteClick()
{
  flag = confirm("Delete\nRecord will be deleted");
  if(flag==true){
    document.write("<b><i><font color=red>Record Deleted..</font></b>");
  } else {
    alert("You canceled..");
  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body>
  <button onclick="DeleteClick()">Delete</button>
</body>
</html>
```

innerText

- It can display output in any container of HTML page, which can show text.
- It will not allow formats for text.
- It is only for plain text content.
- Markup not allowed for output.

Syntax:

```
document.querySelector("reference").innerText = "message";
```

Ex:

index.js

```
function DeleteClick()
{
  flag = confirm("Delete\nRecord will be deleted");
  if(flag==true){
    document.querySelector("h2").innerText = "Delete Confirmed";
    document.querySelector("p").innerText = "Record Deleted Successfully..";
  } else {
    alert("You canceled..");
  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
```

```
    <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body>
  <button onclick="DeleteClick()">Delete</button>
  <h2></h2>
  <p></p>
</body>
</html>
```

innerHTML

- It is similar to innerText but allows formats with markup or functions.

Syntax:

```
    document.querySelector("reference").innerHTML = "markup/message";
```

index.js

```
function DeleteClick()
{
  flag = confirm("Delete\nRecord will be deleted");
  if(flag==true){
    document.querySelector("h2").innerHTML = "<font color=red>Delete Confirmed</font>";
    document.querySelector("p").innerHTML = "<i><font color=red>Record Deleted
Successfully..</font></i>";
  } else {
    alert("You canceled..");
  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body>
  <button onclick="DeleteClick()">Delete</button>
  <h2></h2>
  <p></p>
</body>
</html>
```

Ex:

index.js

```
function DeleteClick()
{
  flag = confirm("Delete\nRecord will be deleted");
  if(flag==true){
    document.querySelector("input[type=text]").value = "Delete Confirmed";
  } else {
    alert("You canceled..");
  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body>
  <button onclick="DeleteClick()">Delete</button>
  <input type="text">

</body>
</html>
```

outerHTML

- It can replace the target markup with the specified.

Syntax:

```
   document.querySelector("targetElement").outerHTML = "<newElement>";
```

Ex:

index.js

```
function DeleteClick()
{
  flag = confirm("Delete\nRecord will be deleted");
  if(flag==true){
    document.querySelector("p").outerHTML = "<h2>Delete Confirmed</h2>";
  } else {
    alert("You canceled..");
  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body>
  <button onclick="DeleteClick()">Delete</button>
  <p></p>

</body>
</html>
```

console

- It is a developer tool.
- Developer can test all his logic in console.

- Developer can log information in console with various contextual messages
  - warning
  - success
  - error
  - debug
  - info

Note: Make sure that console methods are not in production.

Value is not provided : NullException
Please provide a value, Name is required

Syntax:
```
console.log(), debug(), info(), error(), warn() ...
```

Ex:
index.js

```
function DeleteClick()
{
   console.log("Delete Button Clicked");
  flag = confirm("Delete\nRecord will be deleted");
  if(flag==true){
    console.warn("OK Button Clicked - Record will delete");
    document.querySelector("p").outerHTML = "<h2>Delete Confirmed</h2>";
  } else {
    alert("You canceled..");
    console.error("Cancel Clicked");
  }
}
```

Day 7 : JavaScript Input Methods – 07/03/23

Output Techniques
- alert() : undefined
- confirm() : true/false
- document.write() : new screen
- innerText : text without formats
- innerHTML : text with markup
- outerHTML : replacing the element
- console methods : developer tools

JavaScript Input Methods
- Input is the process of taking value from client and handle the value in browser memory or to manage DOM.

- JavaScript can take input from
  a) Query String
  b) Prompt()
  c) Form Input Elements

Query String

- It is a reference value created and appended into browser as URL.
- It is present in browser address bar.
- User can input a value into page from query string.
- Query string is appended into URL using "?"

page.html ? uname=john
page.html ? uname=john&age=23

- Querystring is accessed and used in page by using Browser Object [BOM] "location".

location.search => return the query string

Ex:
index.js

```
function bodyload(){
  string = location.search;
  username = string.substring(string.indexOf("=")+1);
  document.querySelector("span").innerHTML = username;
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body onload="bodyload()">
  Your Name: <span></span>
</body>
</html>
```

prompt()

- It is an input box provided by browser to accept input from user.

Syntax :

```
prompt("Message", "default_value");
prompt("Message");
```

- Prompt returns

Empty String [" "] : On OK without value
String [value entered] : On OK with value = string
null : On Cancel with or without value

Ex:
index.js

```
function bodyload(){
   username = prompt("Enter User Name");
   age = prompt("Enter Age");
   if(username=="") {
      alert("Name can't be empty");
   } else if(username==null) {
      alert("You Canceled");
   } else {
       document.querySelector("span").innerHTML = username + "<br>" + "Your Age :" + age;
   }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body onload="bodyload()">
  Your Name: <span></span>
</body>
</html>
```

Form Input Elements

- You can accept input from user by using various form elements like
  - textbox
  - password
  - checkbox
  - dropdown
  - listbox
  - number, range, radio, etc..

Ex:
index.js

```
function CreateClick(){
   folderTextBox = document.getElementById("FolderName");
   error = document.getElementById("Error");

   if(folderTextBox.value=="") {
      error.innerHTML = "Folder Name Required";
   } else {
```

```
    document.querySelector("p").innerHTML += folderTextBox.value + "<br>";
    folderTextBox.value = "";
    error.innerHTML = "";
    }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="text/javascript" src="src/scripts/index.js"></script>
</head>
<body>
    <input type="text" id="FolderName" placeholder="Folder Name"> <button
onclick="CreateClick()">Create Folder</button>
    <div id="Error" style="color:red"></div>
   <p></p>
</body>
```

Day 8 : JavaScript Input Methods Using Form – 08/03/23

Various Input methods

- Query String
  - ?ref=value
  - location.search
- Prompt , string | null
- Form Input Elements
  - text, number, check, radio, dropdown, listbox etc.. | string

Ex: HTML, CSS, Bootstrap

1. Install Bootstrap for your project

```
> npm install bootstrap  --save
> npm install bootstrap-icons --save
> npm install jquery --save
```

node_modules

```
bootstrap/dist/css/bootstrap.css
bootstrap-icons/font/bootstrap-icons.css

jquery/dist/jquery.js
bootstrap/dist/js/bootstrap-bundle.js
bootstrap/dist/js/popper.js
```

2. Link all bootstrap file to your HTML page

Ex: Input with Modal in bootstrap

input-demo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Input Demo</title>
   <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
   <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
   <script src="../src/scripts/input-demo.js"></script>
</head>
<body class="container-fluid">
   <div id="RegisterContainer">
   <h2>JavaScript Form Input</h2>
   <button class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#register">Register
Product</button>
   </div>

   <div class="modal fade" id="register">
      <div class="modal-dialog">
       <div class="modal-content">
         <div class="modal-header">
           <h3> <span class="bi bi-cart4"></span> Register Product</h3>
           <button class="btn-close" data-bs-dismiss="modal"></button>
         </div>
         <div class="modal-body">
           <dl>
              <dt>Product Name</dt>
              <dd><input class="form-control" type="text" id="Name"></dd>
              <dt>Price</dt>
              <dd><input class="form-control" type="number" id="Price"></dd>
              <dt>City</dt>
              <dd>
                 <select class="form-select" id="City">
                    <option>Delhi</option>
                    <option>Hyd</option>
                    <option>Mumbai</option>
                 </select>
              </dd>
              <dt>Stock</dt>
              <dd class="form-switch">
                 <input class="form-check-input" id="Stock" type="checkbox"> Available
              </dd>
              <dt>Manufactured</dt>
```

```
                <dd>
                    <input class="form-control" type="date" id="Manufactured">
                </dd>
            </dl>
        </div>
        <div class="modal-footer">
            <button id="btnModelRegister" data-bs-dismiss="modal" onclick="RegisterClick()"
class="btn btn-success">Register</button>
            <button class="btn btn-danger" data-bs-dismiss="modal">Cancel</button>
        </div>
      </div>
    </div>
  </div>

  <div id="DetailsContainer" style="display:none">
    <h3>Product Details</h3>
    <dl class="row">
      <dt class="col-4">Name</dt>
      <dd class="col-8" id="lblName"></dd>
      <dt class="col-4">Price</dt>
      <dd class="col-8" id="lblPrice"></dd>
      <dt class="col-4">City</dt>
      <dd class="col-8" id="lblCity"></dd>
      <dt class="col-4">Stock</dt>
      <dd class="col-8" id="lblStock"></dd>
      <dt class="col-4">Manufactured</dt>
      <dd class="col-8" id="lblMfd"></dd>
    </dl>
    <button onclick="EditClick()" data-bs-toggle="modal" data-bs-target="#register" class="btn btn-
warning">
      <span class="bi bi-pen"></span> Edit
    </button>
    <a href="./input-demo.html" class="btn btn-primary">
      New Product
    </a>
  </div>

  <script src="../node_modules/jquery/dist/jquery.js"></script>
  <script src="../node_modules/bootstrap/dist/js/bootstrap.bundle.js"></script>
</body>
</html>
```

input-demo.js

```
function RegisterClick(){
  document.getElementById("DetailsContainer").style.display = "block";
  document.getElementById("RegisterContainer").style.display = "none";

  document.getElementById("lblName").innerHTML = document.getElementById("Name").value;
  document.getElementById("lblPrice").innerHTML = document.getElementById("Price").value;
```

```
    document.getElementById("lblCity").innerHTML = document.getElementById("City").value;
    document.getElementById("lblMfd").innerHTML =
document.getElementById("Manufactured").value;

    stockStatus = "";
    stockCheckBox = document.getElementById("Stock");
    if(stockCheckBox.checked) {
        stockStatus = "Available";
    } else {
        stockStatus = "Out of Stock";
    }
    document.getElementById("lblStock").innerHTML = stockStatus;

}

function EditClick(){
    document.getElementById("btnModelRegister").innerHTML = "Save";
    document.getElementById("btnModelRegister").className = "btn btn-info";
}
```

Task:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Inox Movies</title>
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
</head>
<body class="container-fluid">
    <h2>Inox Movies</h2>
    <button class="btn btn-danger" data-bs-target="#movies" data-bs-toggle="modal">Book
Ticket</button>
    <div class="modal" id="movies">
      <div class="modal-dialog modal-fullscreen">
          <div class="modal-content">
            <div class="modal-header">
              <h4>Book Ticket</h4>
              <button class="btn-close" data-bs-dismiss="modal"></button>
            </div>
            <div class="modal-body">
              <div class="d-flex justify-content-around">
                <div>
                  <select class="form-select">
                    <option>Select Movie</option>
                    <option>Top Gun - Maverick</option>
                    <option>Pathaan - Hindi</option>
                  </select>
```

```
                </div>
                <div>
                  <select class="form-select">
                    <option>Select Cinema</option>
                    <option>Inox GVK </option>
                    <option>Inox Ammerpet</option>
                  </select>
                </div>
                <div>
                  <select class="form-select">
                    <option>Select Date</option>
                    <option>Today 08-March </option>
                    <option>Tomorrow 09-March</option>
                  </select>
                </div>
                <div>
                  <select class="form-select">
                    <option>Select Time</option>
                    <option> 10:30 AM </option>
                    <option> 5:40 PM</option>
                  </select>
                </div>
                <div>
                  <select class="form-select">
                    <option>Select Seats</option>
                    <option>1</option>
                    <option>2</option>
                  </select>
                </div>
                <div>
                  <button class="btn btn-danger">Book</button>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
      <script src="../node_modules/jquery/dist/jquery.js"></script>
      <script src="../node_modules/bootstrap/dist/js/bootstrap.bundle.js"></script>
    </body>
    </html>
```

Day 9 : Variables In JavaScript – 09/03/23

Summary
- JavaScript history and versions
- Features and Drawbacks
- Integrating JavaScript client side
- JavaScript reference techniques
- JavaScript Input and Output

JavaScript Language Basics

1. Variables
2. Data Types
3. Operators
4. Statements
5. Functions

Variables

- Variables are storage locations in memory, where you can store a value and use it as a part of any expression.

- Variables have 3 phase of configuration

a) Declaration
b) Assignment
c) Initialization

```
var x;              declaring
x = 10;             assignment

var y=10;           initialization
```

- JavaScript allows to use variables without declaring if it is not in strict mode

```
<script>
    x = 10;                     // valid
    document.write("x=" + x);
</script>

<script>
  "use strict";
   x = 10;                      //invalid  x is not defined
   document.write("x=" + x);
</script>
```

- If Javascript is in strict mode, then you have to declare or initialize a variable.

- JavaScript variables can be initialized or declared by using 3 keywords

a) var
b) let
c) const

Var

- It defines a function scope variable
- You can declare in any block of a function and access from any another block in the same function.
- It allows declaring, initialization and assignment.

Ex:

```
<script>
    "use strict";
     function f1(){
       var x;              // declaring
       x = 10;              // assignment
       if(x==10)
       {
          var y = 20;       // initialization
       }
       document.write("x=" + x + "<br>" + "y=" + y);
     }
     f1();
</script>
```

- Var allows shadowing. It is the process of re-declaring or re-initializing same name identifier within the function scope.

Syntax:

```
   <script>
      "use strict";
       var x = 10;
       var x = 20;                    // shadowing
       document.write("x=" + x);
    </script>
```

Ex:

```
<script>
    "use strict";
     function f1(){
       var x;              // declaring
       x = 10;              // assignment
       if(x==10)
       {
          x = 30;           // assigning
          x = 40;           // assigning
          var x;
          x = 15;           // shadowing
          var y = 20;       // initialization
          y = 50;          // assigning
          var y = 60;        // shadowing
       }
       document.write("x=" + x + "<br>" + "y=" + y);
     }
     f1();
</script>
```

- Var allows hoisting. It is the process of declaring or initializing a variable after using.

Ex:

```
<script>
    "use strict";
     function f1(){
```

```
        x = 10;
        document.write("x=" + x);
        var x;        // hoisting
    }
    f1();
</script>
```
- Interpeter uses Lexical approach [bottom to top]

Let
- It is used to define a block scope variable.
- It is accessible within the specified block and its inner blocks.

```
    {
      block outer - a
      {
       block inner - a is accessible to inner
       b - is not accessible to outer
      }
    }
```

- It allows declaring, initialization and assignment.
- It will not allow shadowing and hoisting.

const
- It is also block scope variable.
- It allows only initialization.
- It will not allow declaring and assigning.
- It will not allow shadowing and hoisting.

Syntax:
```
    const x;          // invalid
    x = 10;          // invalid

    const x = 10;       // valid
```

INOX – TASK :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inox Movies</title>
  <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
  <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
</head>
<body class="container-fluid">
  <h2>Inox Movies</h2>
  <button class="btn btn-danger" data-bs-target="#movies" data-bs-toggle="modal">Book
Ticket</button>
```

```
<div class="modal" id="movies">
  <div class="modal-dialog modal-fullscreen">
    <div class="modal-content">
      <div class="modal-header">
        <h4>Book Ticket</h4>
        <button class="btn-close" data-bs-dismiss="modal"></button>
      </div>
      <div class="modal-body">
        <div class="d-flex justify-content-around">
          <div>
            <select class="form-select">
              <option>Select Movie</option>
              <option>Top Gun - Maverick</option>
              <option>Pathaan - Hindi</option>
            </select>
          </div>
          <div>
            <select class="form-select">
              <option>Select Cinema</option>
              <option>Inox GVK </option>
              <option>Inox Ammerpet</option>
            </select>
          </div>
          <div>
            <select class="form-select">
              <option>Select Date</option>
              <option>Today 08-March </option>
              <option>Tomorrow 09-March</option>
            </select>
          </div>
          <div>
            <select class="form-select">
              <option>Select Time</option>
              <option> 10:30 AM </option>
              <option> 5:40 PM</option>
            </select>
          </div>
          <div>
            <select class="form-select">
              <option>Select Seats</option>
              <option>1</option>
              <option>2</option>
            </select>
          </div>
          <div>
            <button class="btn btn-danger">Book</button>
          </div>
        </div>
      </div>
    </div>
  </div>
```

```
    </div>
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script src="../node_modules/bootstrap/dist/js/bootstrap.bundle.js"></script>
</body>
</html>
```

Day 10 : Variables Rules In JavaScript – 10/03/23

- Global scope of variable is defined by declaring outside function and inside a module scope.

Ex:

```
<script>
   "use strict";
    var x = 10;
    let y = 20;
    const z = 30;
    function f1()
    {
       document.write("function 1 <br> x=" + x + "<br>y=" + y + "<br>z=" + z + "<br>");
    }
    function f2()
    {
      document.write("function 2 <br> x=" + x + "<br>y=" + y + "<br>z=" + z + "<br>");
    }
    f1();
    f2();
</script>
```

FAQ: Can we define a global variable inside a function?
Ans : Yes.

a) You have to turn off strict mode and define a variable in function without keyword.

Ex:

```
<script>
    var x = 10;
    let y = 20;
    const z = 30;
    function f1()
    {
        a = 100;
       document.write("function 1 <br> x=" + x + "<br>y=" + y + "<br>z=" + z + "<br> a=" + a + "<br>");
    }
    function f2()
    {
      document.write("function 2 <br> x=" + x + "<br>y=" + y + "<br>z=" + z + "<br> a=" + a + "<br>");
    }
    f1();
    f2();
</script>
```

b) You can declare a global variable in side function using
"window" object.

Ex:

```
<script>
   "use strict";
   var x = 10;
   let y = 20;
   const z = 30;
   function f1()
   {
      window.a = 100;
     document.write("function 1 <br> x=" + x + "<br>y=" + y + "<br>z=" + z + "<br> a=" + a + "<br>");
   }
   function f2()
   {
    document.write("function 2 <br> x=" + x + "<br>y=" + y + "<br>z=" + z + "<br> a=" + a + "<br>");
   }
   f1();
   f2();
</script>
```

- Variable naming rules

  * Name must start with an alphabet or can start with _ .
  * _ is used for configures lot of constructs.
  * _ refers that variable requires further implementation.

```
var product;          // its functionality is final
var _product;          // its need to be implemented
```

  * Don't use special chars in variable
  * Name can be alpha numeric

```
var product2020;
```

  * Name is case sensitive.
  * Name can be max 255 chars long.
  * Avoid using single char and long variable name.
  * Avoid using keywords

```
var const;       // invalid
var if;          // invalid
```

  * Always use cameCase for name.

```
var userPassword;
```

PMD, Sonar - Code Analyzer

- You can configure multiple variables with single keyword reference.

```
<script>
   var x, y, z;
   x = 10;
   y = 20;
   z = 30;
   document.write("x=" + x +"<br>y=" + y + "<br>z=" + z);
</script>

<script>
   var x, y=50, z;
   x = 10;
   z = 30;
   document.write("x=" + x +"<br>y=" + y + "<br>z=" + z);
</script>

<script>
   const x=10, y=50, z=30;
   document.write("x=" + x +"<br>y=" + y + "<br>z=" + z);
</script>


<script>
   const x=y=z=30;
   document.write("x=" + x +"<br>y=" + y + "<br>z=" + z);
</script>

<script>
  let y;                  // undefined
  let x=y;                // undefined
  document.write("x=" + x + "<br>y=" + y);
</script>
```

- ES6 introduced de-structuring of variables, which allows to define multiple variables using "[ ]" meta character.

```
var [a, b, c]  = [10, 20, 30];
```

[ ] is an iterator, it needs only a collection of values to read.

```
var [a,b,c] = 10;       // invalid  10 is not iterable
```

FAQ: How to create a collection of constants? [Enum]
Ex:

```
<script>
  const [x,y,z] = [10, 50, 20];
  document.write("x=" + x + "<br>y=" + y + "<br>z=" + z);
</script>
```

- Variable allows object de-structuring, object comprises properties, values are stored under the reference of a property.

```
var {property1,  property2} = {property1:value, property2:value}
```

Ex:

```
<script>
  const {rate, count} = {rate:4.3, count:2000};
  document.write("Product Rating : " + rate + "<br> Count : " + count);
</script>
```

```
[   ]        Iterator    collection of values
{   }        object       set of properties and value
```

JavaScript Data Types

- JavaScript is not a strongly typed language.
- All JavaScript variables are implicitly typed.
- The data type of variables is decided according to value initialized or assigned.
- There is no restriction for type.
- Data Type defines "Data Structure". [DS]
- Data Structure defines data rules. [type, size]

```
var x = 10;          // x is number
x = "A";             // x is string
x = true;            // x is boolean
```

- JavaScript allows various types of data, which is classified into 2 groups

1. Primitive Types
2. Non-Primitive Types

Primitive Types

- They are Immutable types.
- Their structure can't be changed
- They have fixed range for values.
- They use a stack. [LIFO]

1. Number
2. String
3. Boolean
4. Null
5. Undefined
6. Symbol [ES6]

Day 11 : Number Type – 11/03/23

- Not Strongly Typed = Weakly typed
- Implicitly Typed = Dyamic Types

```
var x;       undefined
x = 10;    number
x = true;    boolean
```

1. Primitive Types
- They are immutable
- Fixed range for values
- Stored in memory stack [LIFO]
- JavaScript Primitive Types

a) Number
b) String
c) Boolean
d) Null
e) Undefined
f) Symbol
g) Bigint

Number Type

- It represents a numeric value.
- A numeric value can be

Signed Integer	-5
Unsigned Integer	5
Floating Point	34.30
Double	420.40
Decimal	4560.44 [29 places]
Exponent	2e3 [2 x 10(3) = 2000]
Hexa	0f0033 [0 to f]
Octa	0o748
Binary	0b1010
Bigint	2n [binary object] [bmp]

```
var x = 10;
var x = 2e3;
var x = 0o76;
var x = 0b1010;
```

- JavaScript is not strongly typed, so we have to explicitly verify the number type by using "isNaN()".

- It is a boolean function that returns true if value is not a number.

Ex:

```
<script>
  var age = prompt("Enter Age");
  if(isNaN(age)){
    document.write("Age must be number");
  } else {
    document.write("Your Age : " + age);
  }
</script>
```

- JavaScript provides parsing methods to convert a numeric string into number.
- A numeric string starts with number and can contain chars.

"10" can be converted
"10A" can be converted
"A10" invalid
"10A20" can be converted

- Parsing method
  parseInt()
  parseFloat()

```
<script>
    var age = parseInt(prompt("Enter Age"));
    document.write("Your Age : " + age + "<br>");
    document.write("Next Your you will be : " + (age+1));
</script>

<script>
    var rate = parseFloat(prompt("Interest Rate", "In %"));
    var interest = rate / 12 / 100;
    document.write("Interest = " + interest);
</script>
```

- JavaScript provides various operators to handle numeric expressions.
  + Add
  - Sub
  * Mul
  / Division
  % Modulus
  ** Power 2**3 = 8
  ++
  --

- JavaScript provides "Math" object to handle numeric expressions and mathematical operations.
- A developer can convert a mathematical or scientific equation into JavaScript expression by using Math object

  Math.PI
  Math.sqrt
  Math.min
  Math.max
  Math.avg
  Math.cos
  Math.sin
  Math.pow etc..

```
var emi = P * r * Math.pow(1 + r, n) / Math.pow(1+r,n) - 1
```

- You can use "typeof" operator to check the data type of any reference value.

```
var  x = 10;
document.write(typeof  x);
```

Ex:

```
<script>
```

```
    var age = parseInt(prompt("Enter Age"));
    if((typeof age)=="number") {
        document.write("You will be " + (parseInt(age)+1));
    } else {
        document.write("Please enter a number");
    }
</script>
```

Day 12 : EMI Calculator – 13/03/23

Number Types
- What are the values reffered as numbers?
- How to parse ?
- How to check the type?
- Math

onload
onclick

Ex:
emi.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>EMI Calculator</title>
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <style>
        .box {
            background-color: white;
            box-shadow: 6px 6px 3px gray;
            padding: 20px;
        }
    </style>
    <script src="../src/scripts/emi.js" type="text/javascript"></script>
</head>
<body class="container-fluid">
    <h1>Personal Loan EMI Calculator</h1>
    <div class="box">
        <div class="row">
            <div class="col">
                Amount you need &amp;#8377; <input type="text" onchange="AmountTextBoxChanged()"
id="txtAmount" size="10">
            </div>
            <div class="col">
                for <input type="text" size="2" id="txtYears" onchange="YearTextBoxChanged()"> years
            </div>
            <div class="col">
                interest rate <input type="text" id="txtRate" size="4" onchange="RateTextBoxChanged()"> %
            </div>
        </div>
        <div class="row mt-4">
```

```
        <div class="col">
          <div class="d-flex">
            <span>&#8377; 50,000 </span>
            <input type="range" onchange="AmountChange()" id="rangeAmount" style="width:150px"
class="form-range" value="50000" min="50000" max="1000000">
             <span>&#8377; 10,00,000</span>
          </div>
        </div>
        <div class="col">
          <div class="d-flex">
            <span> 1 </span>
            <input type="range" onchange="YearsChange()" id="rangeYears" style="width:150px"
class="form-range" value="1" min="1" max="5">
             <span>5</span>
          </div>
        </div>
        <div class="col">
          <div class="d-flex">
            <span> 10.25% </span>
            <input type="range" id="rangeRate" onchange="RateChange()" style="width:150px"
class="form-range" value="10.25" step="0.01" min="10.25" max="18.45">
             <span>18.45%</span>
          </div>
        </div>
      </div>
      <div class="row mt-4">
        <div class="col text-center">
          <button onclick="CalculateClick()" class="btn btn-primary">Calculate</button>
        </div>
      </div>
    </div>
    <p style="font-size:30px" class="mt-3 text-center" id="result"></p>
</body>
</html>
```

emi.js

```
function AmountChange(){
    document.getElementById("txtAmount").value = document.getElementById("rangeAmount").value;
}
function YearsChange(){
    document.getElementById("txtYears").value = document.getElementById("rangeYears").value;
}
function RateChange(){
    document.getElementById("txtRate").value = document.getElementById("rangeRate").value;
}
function CalculateClick(){
    var p = parseInt(document.getElementById("txtAmount").value);
    var n = parseInt(document.getElementById("txtYears").value) * 12;
    var r = parseFloat(document.getElementById("txtRate").value)/12/100;

    var emi = p * r * Math.pow(1 + r, n) / Math.pow(1 + r, n) - 1;

    document.getElementById("result").innerHTML = "Your monthy installment amount is <b> <span
class='text-primary'> &#8377;" + Math.round(emi) + "</span></b> for " + p + " in " + (n/12) + " years.";
}
function AmountTextBoxChanged(){
```

```
    document.getElementById("rangeAmount").value = document.getElementById("txtAmount").value;
}
function YearTextBoxChanged(){
    document.getElementById("rangeYears").value = document.getElementById("txtYears").value;
}
function RateTextBoxChanged(){
    document.getElementById("rangeRate").value = document.getElementById("txtRate").value;
}
```

Task : BMI Calculator

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>BMI</title>
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
    <script>
      function bodyload(){
       var yourWeight = parseInt(prompt("Enter Weight"));
          var yourStatus = document.getElementById("yourStatus");
          if(yourWeight<53) {
              yourStatus.style.marginLeft = "200px";
          } else if(yourWeight>54 && yourWeight<70) {
              yourStatus.style.marginLeft = "600px";
          }
      }
    </script>
</head>
<body class="container-fluid" onload="bodyload()">
    <h2>BMI Status</h2>
    <div class="progress">
       <div class="progress-bar bg-dark me-1" style="width:400px">
            Underweight below 53 kg
       </div>
       <div class="progress-bar bg-success me-1" style="width:400px">
          Normal Weight 54 to 70kg
      </div>
      <div class="progress-bar bg-warning me-1" style="width:400px">
        Overweight  70 to  86 kg
       </div>
       <div class="progress-bar bg-danger" style="width:400px">
          Obese above 86 kg
      </div>
    </div>
    <div>
     <div id="yourStatus">
       <span class="bi bi-caret-down-fill"></span>
       <div>You</div>
     </div>
    </div>
</body>
</html>
```

Day 13 : JavaScript String – 14/03/23

JavaScript String
- String is a literal with group of characters enclosed in
  a) Double Quotes " "
  b) Single Quotes ' '
  c) Backticks ` `

- Double and Single quotes are used for inner and outer strings.

```
var link = "<a href='home.html'> Home </a>";
var link  = '<a href="home.html"> Home </a>';
```

- String with single and double quote requires lot of contact technique with dynamic value.
[string + dynamic + string]

Syntax:
```
"string" + var  + "string" + (expression) + "string";
```

- ES5+ versions can use "backtick" for string.
- Backtick allows a string which can embed expression
- ES5+ versions introduced data binding expression "${ }"

Syntax:
```
`string ${var} string ${expression} string`
```

Ex:
```
<script>
   var username = prompt("Enter Name");
   var age = parseInt(prompt("Enter Age"));
   var msg1 = "Hello !" + " " + username + " " + "you will be" + " " + (age+1) + " " + "next year.<br>";
   var msg2 = `Hello ! ${username} you will be ${age+1} next year.<br>`;
   var msg3 = 'Hello ! ${username} you will be ${age+1} next year.';
   document.write(msg1);
   document.write(msg2);
   document.write(msg3);
</script>
```

Ex:
```
<script>
   var title = prompt("Enter Title");
   var loginName = prompt("Enter Login Name", "UserName, Email, Date");
   var loginType = prompt("Enter Login Type", "Text|Email|Date");
   var login = `
     <form>
      <h2>${title}</h2>
      <dl>
          <dt>${loginName}</dt>
          <dd><input type=${loginType}></dd>
```

```
      </dl>
      <button>Login</button>
     </form>
  `;
  document.write(login);
</script>
```

Note: Single and double quotes will not allow binding expressions.

Syntax: without binding expression

```
"Your monthy installment amount is <b> <span class='text-primary'> &#8377;" + Math.round(emi)
+ "</span></b> for " + p + " in " + (n/12) + " years.";
```

Syntax: with binding expression

```
`Your monthy installment amount is <b> <span class='text-primary'> &#8377; ${Math.round(emi)}
</span> </b> for ${p} in ${(n/12)} years`.
```

- Several chars in a string are non-printable chars.
- You can print the non-printable chars by using "\"

```
\char
\\      => \
```

- These are reffered as escape sequence chars.

```
\n      new line in console, alerts, confirm
\v      vertical space
\t      tab space
```

Syntax:

```
var path = "D:\images\movie.jpg";
document.write(path);

 D:imagesmovie.jpg
```

Syntax:

```
var path = "D:\\images\\movie.jpg";

D:\images\movie.jpg
```

- JavaScript provides various string methods for formatting and manipulations.

- String Formatting methods

```
bold()
italics()
fontcolor()
fontsize()
sup()
sub()
toUpperCase()
toLowerCase() etc..
```

- These string formatting functions must be used on "non-RC" type.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>String</title>
   <script type="text/javascript">
      function RegisterClick(){
         var username = document.getElementById("UserName").value;
         var userError = document.getElementById("UserError");
         if(username=="")
         {
            userError.innerHTML = "User Name Required".fontcolor('red').fontsize(2).bold().italics();
         } else {
            document.write("Registered..");
         }
      }
      function ChangeCase(){
         var ifsc = document.getElementById("ifsc").value;
         document.getElementById("ifsc").value = ifsc.toUpperCase();
      }
   </script>
</head>
<body>
   <dl>
      <dt>User Name</dt>
      <dd><input type="text" id="UserName"></dd>
      <dd id="UserError"></dd>
      <dt>IFSC Code</dt>
      <dd><input type="text" onkeyup="ChangeCase()" size="6" id="ifsc"></dd>
   </dl>
   <button onclick="RegisterClick()">Register</button>
</body>
</html>
```

- JavaScript allows to format a string using "style" and "class".

```
string.style.attributeName = "value";
```

Note: style attributes are written in camel case.

background-color	backgroundColor
text-align	textAlign
margin-left	marginLeft

styles are not directly applied to string, they are defined to element

that handles string.

Ex:

```
if(username=="")
    {
        userError.innerHTML = "User Name Required";
        userError.style.color = "red";
        userError.style.fontWeight = "bold";

    } else {
        document.write("Registered..");
    }
}
```

- JavaScript allows to configure formats for elements using "css class"
- CSS class is applied by using "className" property.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>String</title>
    <style>
        .error-style {
            color:red;
            font-weight: bold;
            font-style: italic;
        }
    </style>
    <script type="text/javascript">
        function RegisterClick(){
            var username = document.getElementById("UserName").value;
            var userError = document.getElementById("UserError");
            if(username=="")
            {
                userError.innerHTML = "User Name Required";
                userError.className = "error-style";

            } else {
                document.write("Registered..");
            }
        }
        function ChangeCase(){
            var ifsc = document.getElementById("ifsc").value;
            document.getElementById("ifsc").value = ifsc.toUpperCase();
        }
    </script>
```

```
</head>
<body>
   <dl>
      <dt>User Name</dt>
      <dd><input type="text" id="UserName"></dd>
      <dd id="UserError"></dd>
      <dt>IFSC Code</dt>
      <dd><input type="text" onkeyup="ChangeCase()" size="6" id="ifsc"></dd>
   </dl>
   <button onclick="RegisterClick()">Register</button>
</body>
</html>
```

Day 14 : String Formatting And Manipulations – 15/03/23

Ex: Apply Format to Element dynamically.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>String Format</title>
   <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
   <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
   <style>
      .dark-theme {
         padding: 10px;
         background-color: black;
         color:white;
      }
   </style>
   <script>
      function ThemeChange(){
         var ThemeCheckbox = document.getElementById("ThemeCheckbox");
         var frmLogin = document.getElementById("frmLogin");
         var loginButton = document.querySelector("button");

         if(ThemeCheckbox.checked) {
            frmLogin.className = "dark-theme";
            loginButton.className = "btn btn-light w-100";
         } else {
            frmLogin.className = "p-3";
            loginButton.className = "btn btn-dark w-100";
         }
      }
   </script>
</head>
<body class="container-fluid">
```

```
    <div class="d-flex justify-content-center align-items-center" style="height:400px">
      <form id="frmLogin">
        <div class="form-switch">
          <input type="checkbox" onchange="ThemeChange()" id="ThemeCheckbox" class="form-
check-input"> Dark Theme
        </div>
        <h2><span class="bi bi-personal-fill"></span> User Login</h2>
        <dl>
          <dt>User Name</dt>
          <dd><input type="text" class="form-control"></dd>
          <dt>Password</dt>
          <dd><input type="password" class="form-control"></dd>
        </dl>
        <button class="btn btn-dark w-100">Login</button>
      </form>
    </div>
</body>
</html>
```

String Manipulation Methods and Properties

1. length : It returns the total number of chars in a string.
String empty is verified by using empty quotes " ".

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>String</title>
    <script>
        function SubmitClick(){
            var UserName = document.getElementById("UserName").value;
            var UserError = document.getElementById("UserError");
            if(UserName==""){
                UserError.innerHTML = "User Name Required".fontcolor('red');
            } else {
                document.write("Registered..");
            }
        }
        function VerifyName(){
            var UserName = document.getElementById("UserName").value;
            var UserError = document.getElementById("UserError");
            if(UserName.length<4) {
                UserError.innerHTML = "Name too short min 4 chars".fontcolor('red');
            } else {
                UserError.innerHTML = "";
            }
            if(UserName.length>10) {
```

```
            UserError.innerHTML  = "Name too long max 10 chars".fontcolor('red');
        }
        if(UserName=="") {
            UserError.innerHTML = "User Name Required".fontcolor('red');
        }
    }
  </script>
</head>
<body>
  <dl>
    <dt>User Name</dt>
    <dd><input type="text" onkeyup="VerifyName()" id="UserName"></dd>
    <dd id="UserError"></dd>
  </dl>
  <button onclick="SubmitClick()">Submit</button>
</body>
</html>
```

2. charAt() : It returns the character present at specified index.

```
var str = "Welcome";
str.charAt(1);          // e
str.charAt(15);          // void - no return type
```

3. charCodeAt() : It returns the ASCII code character present at sepcified index.

```
var str = "Ajay";
str.charCodeAt(0);       // 65
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    function VerifyUser(){
      var UserName = document.getElementById("UserName").value;
      var UserError = document.getElementById("UserError");
      if(UserName.charCodeAt(0)>=65 && UserName.charCodeAt(0)<=90) {
        UserError.innerHTML = "";
      } else {
        UserError.innerHTML = "Name must start with uppercase letter".fontcolor('red');
      }
    }
  </script>
</head>
```

```
<body>
   User Name: <input type="text" onblur="VerifyUser()" id="UserName"> <span
id="UserError"></span>
</body>
</html>
```

4. indexOf() : It returns the index number of character present at specified index.

```
var str = "Welcome";
str.indexOf("e");      // 1    only first occurance
str.indexOf("b");      //  -1
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function VerifyEmail(){
         var Email = document.getElementById("Email").value;
         var EmailError = document.getElementById("EmailError");
         if(Email.indexOf("@")==-1) {
            EmailError.innerHTML = "Invalid Email - Please include @ in Email".fontcolor('red');
         } else {
            EmailError.innerHTML = "";
         }
      }
   </script>
</head>
<body>
   Email : <input type="email" onblur="VerifyEmail()" id="Email"> <span id="EmailError"></span>
</body>
</html>
```

5. lastIndexOf() : It gets the last occurance index number of a char.

```
var str = "Welcome";
str.indexOf("e");      // 1
str.lastIndexOf("e");     // 6
str.lastIndexOf("b");     // -1
```

Day 15 : String Methods – 16/03/23

String Manipulations
- length
- charAt()

- charCodeAt()
- indexOf()
- lastIndexOf()
- startsWith() : It returns boolean true if string starts with specified chars.
- endsWith() : It returns true if string ends with specified chars

Syntax:

```
var str = "Welcome";
str.startsWith("w");       // false
str.startsWith("W");       // true
str.endsWith("e");       // true
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function VerifyEmail(){
         var email = document.getElementById("Email").value;
         var emailError = document.getElementById("EmailError");

         if(email.endsWith("outlook.com")) {
            emailError.innerHTML = "";
         } else {
            emailError.innerHTML = "Please provide a valid Skype Account".fontcolor('red');
         }
      }
      function VerifyCard(){
         var card = document.getElementById("Card").value;
         var pic = document.getElementById("pic");
         if(card.startsWith("44")){
            pic.src="../public/images/visa.png";
         } else if (card.startsWith("55")){
            pic.src="../public/images/master.png";
         } else {
            pic.src = "";
            pic.alt = "N/A";
         }
      }
   </script>
</head>
<body>
   <dl>
      <dt>Your Skype Account</dt>
      <dd><input type="email" onblur="VerifyEmail()" placeholder="@outlook" id="Email"></dd>
      <dd id="EmailError"></dd>
```

```
        <dt>Your Card Number</dt>
        <dd><input type="text" onkeyup="VerifyCard()" id="Card"><img width="50" align="left"
height="20" id="pic"></dd>
    </dl>
</body>
</html>
```

- slice() : It reads and returns the chars between specified index.

Syntax:

```
        slice(startIndex, endIndex)     => chars between specified index
        slice(startIndex)               => chars from start to end index
        slice(7,4)                      => It can't read, end index must be
                                        the index after start.
```

- substr() : It reads specified chars from given index.

Syntax:

```
        substr(startIndex, howMany)
        substr(7, 4);                   => from 7 it reads 4 chars
        substr(7,0);                    => will not read any chars
        substr(7);                      => read upto end
```

- substring() : It reads specified chars bi-directional.

Syntax:

```
        substring(startIndex, endIndex)    => end Index can be any direction
        substring(7)                       => from 7 to end
        substring(7,0)                     => from 7 to 0 [start]
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function GetDetails(){
            var email = document.getElementById("Email").value;
            var atPos = email.indexOf("@");
            var id = email.substring(atPos,0);
            var domain = email.substring(atPos + 1);
            document.getElementById("Id").innerHTML = id;
            document.getElementById("domain").innerHTML = domain;
        }
    </script>
</head>
<body>
    Your Email : <input type="email" onblur="GetDetails()" id="Email">
```

```
    <dl>
        <dt>Your ID</dt>
        <dd id="Id"></dd>
        <dt>Domain</dt>
        <dd id="domain"></dd>
    </dl>
</body>
</html>
```

- split() : It splits a string at specified char and returns an array.

Syntax:

```
    string.split(' char ');
```

Ex:

```
<script>
    var products = "Samsung TV-46000.44, Nike Casuals-5000.44";
    var [tv, shoe] = products.split(',');
    var [name, price] = shoe.split('-');
    document.write(`<h2>Shoe Details</h2>
        Name : ${shoe.substring(shoe.indexOf("-"),0)} <br>
        Price: ${shoe.substring(shoe.indexOf("-")+1)} <br>
        <hr>
        Shoe Name: ${name} <br>
        Shoe Price: ${price}
    `);
</script>
```

Ex:

```
<script>
    var products = "Samsung TV-46000.44, Nike Casuals-5000.44";
    var [tv, shoe] = products.split(',');
    var details = shoe.split('-');
    document.write(`<h2>Shoe Details</h2>
        Name : ${shoe.substring(shoe.indexOf("-"),0)} <br>
        Price: ${shoe.substring(shoe.indexOf("-")+1)} <br>
        <hr>
        Shoe Name: ${details[0]} <br>
        Shoe Price: ${details[1]}
    `);
</script>
```

- trim() : It is used to remove leading spaces in a string.
[space before and after string]

Syntax:

```
    string.trim()
```

Ex:

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

```
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function SubmitClick(){
         var UserId = document.getElementById("UserId").value;
         if(UserId.trim()=="john_nit") {
            document.write("Success..");
         } else {
            alert("Invalid UserId");
         }
      }
   </script>
</head>
<body>
   Your UserId:
   <input type="text" id="UserId">
   <button onclick="SubmitClick()">Submit</button>
</body>
</html>
```

- match() : It verifies your string by matching it against a regular expression and return true if matched.

Syntax:

```
string.match(/regularExpression/);
```

- Regular expression is built with meta characters and quantifiers.

Meta Characters

? zero or one occurance of a character.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function SubmitClick(){
         var test = document.getElementById("test").value;
         if(test.match(/colou?r/)) {
            document.write(`Your entered : ${test}`);
         } else {
           alert("Only Color | Colour Allowed");
         }
```

```
    }
  </script>
</head>
<body>
      Your string : <input type="text" id="test"> <button onclick="SubmitClick()">Submit</button>
</body>
</html>
```

Text Attributes
- name
- id
- class
- value
- size
- placeholder
- autofocus
- minlength
- maxlength
- required
- list
- pattern : It uses a regular expression to verify the format of input.
- disabled
- readonly

Day 16 : Regular Expressions – 17/03/23

Meta Character

? Zero or one occurance of any character

Ex: colou?r => color, colour

+ One or more occurances of character

Ex: colou+r => colour, colouur, colouuur

* Zero or more occurance of character

Ex: colou*r => color, colour, colouur..

. Any single character

Ex: .at => cat, bat, mat, 1at, $at
.o. => boy, cow, toy, dos, 1o$

\ It changes the meta character into normal character and vice versa.

Ex: gmail\.com

| It is used as OR, allows multiple choices

Ex: red|green|blue

\^ Expression starts with

$ Expression end width

Ex: \^.....$

\d It allows only single digit number

Ex: \d => 1, 4, 6, 2, 3
\d\d => 22, 11, 24, 13
\d?\d => 1, 33, 35

\D It allows only non-numeric chars, alphabet and special

Ex: \D => A, a, !@#
Ex: \D\d => A3, _4

\w It allows word chars [A-Z, a-z, 0-9, _]

\W It allows non-word chars, only special chars

\s It refers to blank space.

\i It ignores case [not case sensitive]

Ex: colour\i

[ ] Random and Range

( ) Union of chars

[A-Z] Only upper case letters allowed.

[a-z] Only lower case

[a-zA-Z] Both upper and lower

[a-Z] Both upper and lower

[a-m] Only chars with in specified range.

[a,d,s,m] Only specified chars allowed

[0-9] Only numeric

[a-zA-Z0-9] alpha numeric

[a-mA-M0-4] Only specified range of chars allowed

[!@#$%^&] Only specified special chars allowed

Quantifiers

{n} Exactly n-number of chars allowed

Ex: \d{10} only 10 digits number allowed
\w{4} exactly 4 chars

{n, m} Lower Bound - n and upper bound - m

Ex: \d{4,7}

{n, } Lower Bound is "n" and upper bound is any.

Ex: \d{4, } Minimum 4 and maximum any

Ex: Write an expression to validate the following format of phone number

+91 and 10 digits number

pattern="\+91\d{10}" | \+91[0-9]{10}

Ex: Write a pattern for following

+(1)(425) 555-0100 - US
+(44)(20) 1234 5678 - UK

pattern="\+\(1\)\(\d{3}\)\s\d{3}-\d{4}"
"\+\(\d{2}\)\(\d{2}\)\s\d{4}\s\d{4}"

- Pre Defined Expression format

(?=.*[A-Z]) at least one upper case letter
(?=.*[0-9]) at least one numeric

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
```

```
      function SubmitClick(){
        var test = document.getElementById("test").value;
        if(test.match(/\+\(1\)\(\d{3}\)\s\d{3}-\d{4}/)) {
           document.write(`Your entered : ${test}`);
        } else {
          alert("Invalid - +(1)(425) 555-0100");
        }
      }
    </script>
</head>
<body>
      Your string : <input type="text" id="test"> <button onclick="SubmitClick()">Submit</button>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
      function SubmitClick(){
        var test = document.getElementById("test").value;
        if(test.match(/(?=.*[A-Z])\w{4,10}/)) {
           document.write(`Your entered : ${test}`);
        } else {
          alert("Invalid - Name 4 to 10 chars with atleast one uppercase letter");
        }
      }
    </script>
</head>
<body>
      Your string : <input type="text" id="test"> <button onclick="SubmitClick()">Submit</button>
</body>
</html>
```

Task:

1. Write expression for validating email address
2. Write expression for validating date format mm-dd-yyyy
3. Write expression for validating name 4 to 15 chars with at least one uppercase, number and special char.

Day 17 : String And Boolean – 21/03/23

Ex:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
   <script>
      var regExp = / /;
      function CountryChanged(){
         var countryName = document.getElementById("lstCountries").value;
         var flag = document.getElementById("flag");
         var txtMobile = document.getElementById("txtMobile");

         if(countryName=="India") {
            flag.src = "images/india.png";
            txtMobile.placeholder = "+91 and 10 digits";
            regExp = /\+91\d{10}/;

         } else if (countryName=="US") {
            flag.src = "images/us.png";
            txtMobile.placeholder = "+(1)(425) 555-0100";
            regExp = /\+\(1\)\(\d{3}\)\s\d{3}-\d{4}/;

         } else if (countryName=="UK") {
            flag.src = "images/uk.png";
            txtMobile.placeholder = "+(44)(20) 1234 5678";
            regExp = /\+\(44\)\(\d{2}\)\s\d{4}\s\d{4}/;
         }
      }
      function RegisterClick(){
         var mobile = document.getElementById("txtMobile").value;
         var mobileError = document.getElementById("mobileError");
         if(mobile.match(regExp)){
            document.write("Registered...");
         } else {
            mobileError.innerHTML = `Invalid Mobile -
${document.getElementById("txtMobile").placeholder}`;
         }
      }
   </script>
</head>
<body class="container-fluid">
   <h2>Verify Mobile</h2>
   <dl class="w-50">
      <dt>Your Country</dt>
      <dd class="input-group">
         <select id="lstCountries" class="form-select" onchange="CountryChanged()">
            <option>Select Country</option>
            <option>India</option>
            <option>US</option>
```

```
            <option>UK</option>
          </select>
          <img width="80" height="40" id="flag" border="1" class="input-group-text">
        </dd>
        <dt>Your Mobile</dt>
        <dd>
          <input type="text" class="form-control"  id="txtMobile">
        </dd>
        <dd class="text-danger" id="mobileError"></dd>
    </dl>
    <button onclick="RegisterClick()" class="btn btn-primary">Register</button>
</body>
</html>
```

Boolean Types

- Boolean types are used in Decision Making.
- JavaScript boolean types can handle 2 values
  a) true
  b) false
- JavaScript boolean expression can handle true and false using 1 and 0.

```
true = 1
false = 0

var x = true;
var y = false;

if(x==1)       // OK      x == true
{
}

true + true = ?  2
true + "A"  = ?  trueA
true + 10  = ?  11
```

Note: Booleans are defined only with "true or false" as value.
However you can compare booleans using 0 or 1

Ex:

```
<script>
    var x = true;
    if(x==1) {                                  x==true  Good
        document.write("X is true");
    } else {
        document.write("x is false");
    }
</script>
```

Attributes of HTML which are boolean type
- checked

- selected
- required
- readonly
- disabled
- border [0, 1]

Note: You use Ternary operator simple decision making [? :]

(condtion) ? true : false

e1 e2
e1+e2
e1~e2
div~p

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function StockChanged(){
         var stockCheckBox = document.getElementById("Stock");
         var lblStock = document.getElementById("lblStock");
         lblStock.innerHTML = (stockCheckBox.checked)?"Available":"Out of Stock";

      }
   </script>
   <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
   <script>
      function SortClick(){
         var span = document.querySelector("button span");
         span.className = (span.className=="bi bi-sort-alpha-down")? "bi bi-sort-alpha-up" : "bi bi-
sort-alpha-down";
      }
   </script>
</head>
<body>
   <h2>Check Box Toggle</h2>
   <input type="checkbox" onchange="StockChanged()" id="Stock"> <span id="lblStock">Out of
Stock</span>
   <h2>Button Toggle</h2>
   <button id="sort" onclick="SortClick()">
      <span class="bi bi-sort-alpha-down"></span>
   </button>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function StockChanged(){
         var stockCheckBox = document.getElementById("Stock");
         var lblStock = document.getElementById("lblStock");
         lblStock.innerHTML = (stockCheckBox.checked)?"Available":"Out of Stock";

      }
   </script>
   <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
   <script>
      function SortClick(){
         var span = document.querySelector("button span");
         span.className = (span.className=="bi bi-sort-alpha-down")? "bi bi-sort-alpha-up" : "bi bi-
sort-alpha-down";
      }
      function VerifyName(){
         var txtName = document.getElementById("txtName");
         var btnRegister = document.getElementById("btnRegister");
         if(txtName.value=="") {
            btnRegister.disabled = true;
         } else {
            btnRegister.disabled = false;
         }
      }
   </script>
</head>
<body>
   <h2>Your Name</h2>
   <input type="text" onblur="VerifyName()" id="txtName"> <button disabled
id="btnRegister">Register</button>
   <h2>Check Box Toggle</h2>
   <input type="checkbox" onchange="StockChanged()" id="Stock"> <span id="lblStock">Out of
Stock</span>
   <h2>Button Toggle</h2>
   <button id="sort" onclick="SortClick()">
      <span class="bi bi-sort-alpha-down"></span>
   </button>
</body>
</html>
```

Day 18 : Undefined Null and Symbol Type – 23/03/23

JavaScript Data Types
- Number
- String
- Boolean

FAQ: How to convert a numeric string into number?
Ans: parseInt(), parseFloat()

FAQ: How to convert a number into string?
Ans: toString()

```
var x =  10;
x.toString()
```

FAQ: How to convert a string into boolean?
Ans:

```
var x = "true";
var y = (x=="true")?true:false;
```

Ex:

```
<script>
   var x = "true";
   var y = (x=="true")?true:false;
   document.write(`
      X is ${typeof x} <br>
      Y is ${typeof y}
   `);
</script>
```

undefined type

- It specifies that there is no value defined in a reference.

```
var x;
document.write("x=" + x);
```
x = undefined

- Undefined is a marker, that marks the variable to specifiy that there is no value defined.

- You can use "undefined" keyword to verify the value in any reference.

FAQ: Why JavaScript sets undefined into a variable?
Ans : As it is "Implicitly Typed", it requires a data type to determine according to value assigned. If value is not assigned then it is "undefined" data type.

Ex:

```
<script>
   var name = "Samsung TV";
   var price = 10300.33;
```

```
    if(price==undefined) {                    => not good
        document.write("Name = " + name);
    } else {
        document.write(`Name=${name}<br>Price=${price}`);
    }
</script>
```

FAQ: How to verify value defined?
Ans: if(referenceName) { }

```
<script>
    var name = "Samsung TV";
    var price;
    if(price){
        document.write(`Name=${name}<br>Price=${price}`);
    } else {
        document.write(`Name=${name}`);
    }
</script>
```

FAQ: What is difference between undefined and not-defined?
Ans : Undefined verifies the type.
      Not-defined verifies the reference.

Ex: Price not defined

```
<script>
    var name = "Samsung TV";
    if(price){
        document.write(`Name=${name}<br>Price=${price}`);
    } else {
        document.write(`Name=${name}`);
    }
</script>
```

null type

- Undefined is configured by javascript for any reference if value is not found during compile time.
- Null is configured by javascript for any reference if value is not found during run time.
- Null is related to exception.

Ex:

```
<script>
    var price = prompt("Enter Price");
    if(price==null) {
        document.write(`You canceled - Please provide Price`);
    }
    else if(price=="") {
        document.write(`Price can't be empty`);
    }
    else {
        document.write(`Price=${price}`);
```

```
  }
</script>
```

symbol type

- JavaScript introduced symbol type from ES6.
- Earlier JS uses symbol type implicitly.
- It is used to configure a unique identification key for objects.

```
{
  UserName: "John",
  Age : 22
}
```

- Symbol configures a hidden field in object, which is present in object but not used by iterators.
- Iterator is used to access all properties of object.

Ex: HTML Hidden Input

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <form>
    <h2>Edit User Details</h2>
    <dl>
      <input type="hidden" name="UserId" value="john_nit">
      <dt>User Name</dt>
      <dd><input type="text" name="UserName"></dd>
      <dt>Age</dt>
      <dd><input type="number" name="Age"></dd>
    </dl>
    <button>Submit</button>
  </form>
</body>
</html>
```

Ex:

```
<script>
    var UserId = Symbol();

    var userDetails = {
      [UserId] : "john_nit",
      UserName : "John",
      Age: 22
    }
    for(var property in userDetails){
```

```
        document.write(`${property} : ${userDetails[property]} <br>`);
    }
    document.write("User Id : " + userDetails[UserId] + "<br>");
</script>
```

FAQ: How to configure a field which can display value on screen but will not submit?
Ans :
- If name is not defined for any field in a form then it can't submit.
- If field is not in form then it can't submit
- If field is disabled then it can't submit.

```
<input type="text" name="UserId"  value="john_nit" disabled>
```

Day 19 : JavaScript Non-Primitive Types – 24/03/23

JavaScript Non Primitive Types
- They are mutable types.
- They don't have fixed range for values.
- Value range varies according to memory available.
- They are store in memory heap.
- Heap allows to access in any order.
- JavaScript non-primitive types are
    a) Array
    b) Object
    c) Map

Array

- Arrays are used in computer programming to reduce overhead and complexity.
- Array reduces overhead by storing values in sequential order.
- Array can reduce complexity by storing multiple values under one name.
- JavaScript array can store any type of value, which is not possible for several programming languages.
- Array can change its size dynamically in JavaScript.
- Array refers to a formation where items are in order but can be accessed random.

Configuring Array

Array configuration comprises of 2 phases

1. Declaring Array
2. Initializing memory or assigning memory for array

Syntax: Declaring

```
var  products;
let   products;
const products;    // it requires initializaiton
```

Syntax: Initialize memory or assign memory by using "[ ]" or "Array()"

```
var products = [ ];
var products = new Array();

        (or)

var products;
 products = [ ];
 products = new Array();
```

FAQ: What is difference between array [ ] and "Array()" ?
Ans: Array() refers to discreet memory. Disconnected
[ ] refers to connected architecture. Contineous memory

Array() uses single call mechanism.
[ ] uses single ton mechanism.

Storing data into Array:
1. You can initialize data into array.
2. You can assign data into array.

Syntax: Initialization

```
var products = ["A", 10, true];
var products = new Array("A", 10, true);
```

Syntax: Assignment by using array property

```
var values = [ ];
values[1]    = 10;          [1] - string
values["2"] = 20;           [2] - string
```

Ex:

```
<script>
   var values = [10, "john", true];
   values["3"] = "david";
   for(var property in values)
   {
      document.write(`[${property}]-${typeof property} : ${values[property]}-${typeof
values[property]}<br>`);
   }
   document.write(values[2]);
</script>
```

- Array can store any type of data
  a) Primitive
  b) Non Primitive
  c) Function

Ex:

```
<script>
   var values = [10, "TV", true, ["Delhi", "Hyd"], function(){document.write("Hello !")}];
```

```
    document.write(values[3][1] + "<br>");
    values[4]();
</script>
```

- Array supports de-structuring.

```
<script>
    var values = [10, "TV", true, ["Delhi", "Hyd"], function(){document.write("Hello !")}];
    var [id, name, stock, cities, hello] = values;
    document.write(cities[1] + "<br>");
    hello();
</script>
```

Array Manipulations

- JavaScript array object provides a set of properites and methods to control array.

1. length : returns total count of elements in array.

2. Methods for reading values

    a) toString() separated with ","
    b) join() separated with custom delimeter
    c) slice() from specified index
    d) find() it returns the first occurance that match condition
    e) filter() it returns all occurance that match condition
    f) map() it returns all using an iterator.

Ex:

```
<script>
    var categories = ["Electronics", "Footwear", "Fashion"];
    document.write(categories.toString() + "<br>");
    document.write(categories.join("==>") + "<br>");
    document.write(categories.slice(1,2) + "<br>");
    categories.map(function(category){
        document.write(`<button>${category}</button><br>`);
    })
</script>
```

Ex:

```
<script>
   var sales = [35000, 57000, 24400, 67000, 21000];
   var result = sales.find(function(value){
       return value>50000;
   })
   document.write(result);
</script>
```

Ex:

```
<script>
   var sales = [35000, 57000, 24400, 67000, 21000];
   var result = sales.filter(function(value){
```

```
        return value>50000;
    })
    document.write(result);
</script>
```

Day 20 : Array Methods – 27/03/23

What is Array?
What is purpose of Array?
How to configure Array?
How to assign and initialize memory for array?
Array Methods

1. Read array properties and elements

toString()
join()
map()
find()
filter()
slice()
for..in
for..of

for..in

- It is an iterator used to read all properties from array.

Syntax:

```
for(var property in collection)
{
}
```

for..of

- It is an iterator used to read all elements [values] from array.

Syntax:

```
for(var item of collection)
{
}
```

FAQ: How to read both properties and values?
Ans: By using "for..in"

```
for(var property in collection)
{
   document.write(property + "-" + collection[property]);       collection[0]
}
```

FAQ: What is difference between a loop and iterator?
Ans: Loop comprises of initialization, condition, counter to read values from collection.

```
for(var i=0; i<collection.length; i++)
{
}
```

Iterator is a design pattern that allows to read values from collection in sequential order. It doesn't require initialization, condition and counter.

```
for(var item of collection)
{
}
```

```
<script>
  var sales = [35000, 57000, 24400, 67000, 21000];
  for(var property in sales){
    document.write(`[${property}]-${sales[property]}<br>`);
  }
</script>
```

2. Add elements into array

push()	It adds new element(s) as last elements. [bottom]
unshift()	It adds new element(s) as first elements. [top]
splice()	It adds new element(s) at any specific location.

Syntax:

```
collection.push("item1", "item2", ...);
collection.unshift("item1", "item2",..);
collection.splice(indexNumber, deleteCount, "item1", "item2"..);
```

Ex:

```
<script>
  var categories = ["Electronics", "Footwear", "Fashion"];
  categories.unshift("All");
  categories.splice(2,0,"Kids", "Men's Clothing");
  for(var property in categories)
  {
    document.write(`[${property}] ${categories[property]}<br>`);
  }
</script>
```

3. Removing elements from array

pop()	It removes and returns the last element
shift()	It removes and returns the first element
splice()	It removes and returns the specified element(s)

Ex:

```
<script>
  var categories = ["Electronics", "Footwear", "Fashion"];
  document.write(`${categories.splice(1,2)} removed.<br>`);
  for(var property in categories)
  {
```

```
    document.write(`[${property}] ${categories[property]}<br>`);
  }
</script>
```

4. How to Empty Array?
   - set length to "0".
   - assign "[]" to array.

Ex:
```
<script>
  var categories = ["Electronics", "Footwear", "Fashion"];
  categories.length = 0;
  document.write(categories);
</script>
```

Ex:
```
<script>
  var categories = ["Electronics", "Footwear", "Fashion"];
  categories = [];
  document.write(categories);
</script>
```

5. How to create copy of array?

Ex:
```
<script>
  var categories = ["Electronics", "Footwear", "Fashion"];
  var menu = categories;
  document.write(`
      Categories : ${categories} <br>
      Menu : ${menu}
  `);
</script>
```

Ex: Shallow Copy
```
<script>
  var categories = ["Electronics", "Footwear", "Fashion", "Kids"];
  var menu = Array.from(categories);
  document.write(`
      Categories : ${categories} <br>
      Menu : ${menu}
  `);
</script>
```

Ex: Deep Copy

```
<script>
  var categories = ["Electronics", "Footwear", "Fashion", "Kids", "Men"];
  var menu = JSON.parse(JSON.stringify(categories));
  document.write(`
      Categories : ${categories} <br>
```

```
        Menu : ${menu}
    `);
</script>
```

6. Sorting and Reverse Array

sort()	Arranges array elements in ascending order
reverse()	Arrange is reverse order [bottom to top]

Ex:

```
<script>
    var cities = ["Delhi","Hyd","Mumbai", "Chennai", "Bangalore"];
    cities.sort();
    cities.reverse();
    for(var item of cities) {
     document.write(item  + "<br>");
    }
</script>
```

Dynamically Adding Elements into Page

1. Create a new HTML element dynamically.

```
    var ref =    document.createElement("elementName");
    var pic =    document.createElement("img");
    var tableCell =    document.createElement("td");
```

2. Define properties for element

```
       ref.id
       ref.name
       ref.className
       ref.width
       ref.height
       ref.src
```

3. Add new element into page

```
       appendChild(newElement);

       document.querySelector("body").appendChild(ref);
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
```

```
  <script>
    function AddClick(){
       var pic = document.createElement("img");
       pic.width = "200";
       pic.height = "200";
       pic.border = "2";
       pic.src = "../public/images/shoe.jpg";
       document.getElementById("container").appendChild(pic);
    }
  </script>
</head>
<body>
  <button onclick="AddClick()">Add Image</button>
  <div id="container">

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

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    function AddClick(){
       var pic = document.createElement("img");
       pic.width = "200";
       pic.height = "200";
       pic.border = "2";
       pic.src = "../public/images/shoe.jpg";
       document.getElementById("container").appendChild(pic);
    }
    function AddOption(){
       var option = document.createElement("option");
       option.text = "Delhi";
       option.value = "Delhi";
       document.querySelector("select").appendChild(option);
    }
  </script>
</head>
<body>
  <button onclick="AddClick()">Add Image</button>
  <button onclick="AddOption()">Add Option</button>
  <div id="container">

  </div>
  <select>
```

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

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    var categories = ["All", "Electronics", "Footwear", "Fashion"];

    function AddOption(){
       document.querySelector("select").innerHTML = "";
       for(var item of categories){
           var option = document.createElement("option");
           option.text = item;
           option.value = item;
           document.querySelector("select").appendChild(option);
       }
    }
  </script>
</head>
<body>
  <button onclick="AddOption()">Add Option</button>

  <select>

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

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    var categories = ["All", "Electronics", "Footwear", "Fashion"];

    function AddOption(){
       document.querySelector("select").innerHTML = "";
       for(var item of categories){
```

```
            var option = document.createElement("option");
            option.text = item;
            option.value = item;
            document.querySelector("select").appendChild(option);

            var li = document.createElement("li");
            li.innerHTML = item;
            document.querySelector("ol").appendChild(li);

            var tr = document.createElement("tr");
            var td = document.createElement("td");
            td.innerHTML = item;
            tr.appendChild(td);
            document.querySelector("tbody").appendChild(tr);

        }
      }
    </script>
  </head>
  <body>
    <button onclick="AddOption()">Add Option</button>

    <select>

    </select>
    <h2>Categories</h2>
    <ol>

    </ol>
    <h2>Table</h2>
    <table border="1" width="400">
      <thead>
        <tr>
          <th>Categories</th>
        </tr>
      </thead>
      <tbody>

      </tbody>
    </table>
  </body>
  </html>
```

Day 21 : Array Examples – 28/03/23

Finding an Element in Array:

find()	It returns the matching value using a call back function
indexOf()	It returns index number of specified element, if not found it returns -1.
lastIndexOf()	It returns last occurance index number.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Array</title>
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
    <script>
        var cities = ["Delhi", "Hyd", "Mumbai"];
        function LoadCities(){
            document.getElementById("lstCities").innerHTML = "";
            for(var city of cities){
                var option = document.createElement("option");
                option.text = city;
                option.value = city;
                document.getElementById("lstCities").appendChild(option);
            }
            document.getElementById("lblCount").innerHTML = `Total No of Cities : ${cities.length}`;
        }
        function bodyload(){
            LoadCities();
        }
        function AddClick(){
            var cityName = document.getElementById("txtCity").value;
            if(cities.indexOf(cityName)==-1) {
                cities.push(cityName);
                alert(`${cityName} added to list`);
                document.getElementById("txtCity").value = "";
                cities.sort();
                LoadCities();
            } else {
                alert(`${cityName} Exists`);
            }
        }
        function RemoveClick(){
            var selectedCityName = document.getElementById("lstCities").value;
            var selectedIndex = cities.indexOf(selectedCityName);
            var flag = confirm(`Delete ${selectedCityName}\nAre you sure want to delete?`);
            if(flag==true){
                cities.splice(selectedIndex,1);
                LoadCities();
            }
        }
        function ClearClick(){
            cities.length = 0;
            LoadCities();
```

```
        }
        function SortAsc(){
            cities.sort();
            LoadCities();
        }
        function SortDsc(){
            cities.sort();
            cities.reverse();
            LoadCities();
        }
    </script>
</head>
<body class="container-fluid" onload="bodyload()">
    <h2>Array Manipulations</h2>
    <div class="mt-3 mb-3 w-25">
        <div class="input-group">
            <span class="input-group-text">New City : </span>
            <input type="text" id="txtCity" class="form-control">
            <button onclick="AddClick()" class="btn btn-primary">Add</button>
        </div>
    </div>
    <div class="w-25">
        <label class="form-label">Your Cities</label>
        <div>
            <div class="mt-2 mb-2">
                <button class="btn btn-primary" onclick="SortAsc()">
                    <span class="bi bi-sort-alpha-down"></span>
                </button>
                <button class="btn btn-primary" onclick="SortDsc()">
                    <span class="bi bi-sort-alpha-up"></span>
                </button>
            </div>
            <select class="form-select" size="3" id="lstCities">

            </select>
            <label class="form-label" id="lblCount"></label>
        </div>
        <div class="mt-3">
            <button class="btn btn-danger" onclick="RemoveClick()"> <span class="bi bi-trash-
fill"></span>Remove City</button>
            <button class="btn btn-danger" onclick="ClearClick()">Clear All</button>
        </div>

    </div>
</body>
</html>


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

```
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
       var photos = ["images/m1.jpg", "images/m2.jpg", "images/m3.jpg", "images/m4.jpg",
"images/m5.jpg"];
        function bodyload(){
           for(var path of photos) {
              var img = document.createElement("img");
              img.src = path;
              img.width = "200";
              img.height = "200";
              document.getElementById("gallery").appendChild(img);
           }
        }
   </script>
   <style>
      body {
         height: 500px;
         display: flex;
         justify-content: center;
         align-items: center;
      }
      img:hover {
         transform: scale(1.5);
         transition: 2s;
         box-shadow: 10px 10px 3px gray;
      }
      img {
         transition: 2s;
      }
      marquee {
         padding: 100px;
      }
   </style>
</head>
<body onload="bodyload()">
   <div>
      <marquee scrollamount="10" id="gallery" onmouseover="this.stop()"
onmouseout="this.start()">

      </marquee>
   </div>
</body>
</html>
```

Object Type

- Object is set of properties and functions.
- Object is to store all related data and logic together.

- "Alan Kay" introduced the concept of Object into computer programming in early 1960's.
- Joahn Olay and Nygaard developer object oriented programming in early 1967.
- It was formulated with a language "SIMULA".

Syntax:

```
let  tv  =  { }
```

- Data and logic is stored in object in the form of "Key - Value" collection.

Syntax:

```
let  tv = {
   Key : value,
   Key : value
}
```

- All keys are "string" type.
- Value can any type, Primitive or Non-primitive.
- If object comprises of only data and no logic then it is reffered as "JSON".
  [JavaScript Object Notation]
- JSON is format used to represent data. [JSON, XML]

JSON

```
{
   "Name": "Samsung TV",
   "Price" : 30000.44,
   "Stock" : true
}
```

XML

```
<Product>
   <Name> Samsung TV </Name>
   <Price> 30000.44 </Price>
   <Stock> true </Stock>
</Product>
```

- You can access any key with reference of object.

Ex:

```
<script>
    var tv = {
      "Name": "Samsung TV",
      "Price": 50000.33,
      "Stock": true,
      "Cities": ["Delhi", "Hyd"],
      "Rating": {"Rate":4.3, "Count": 3500}
    }
    document.write(`
        Name  : ${tv.Name} <br>
        Price : ${tv.Price} <br>
        Stock : ${tv.Stock} <br>
        Cities: ${tv.Cities.toString()} <br>
```

```
        Rating : ${tv.Rating.Rate} [${tv.Rating.Count}]
    `);
</script>
```

Day 22 : JSON Type – 29/03/23

What is an Object in JavaScript?
- Key and Value collection
- It is used to keep all related data and logic together.
- If object maps only to data then it is known as JSON.
  [JavaScript Object Notation]
- Every key is "string" type.
- Every value can be any type, primitive or non-primitive
- If you want to maintain data in a separate location then you can create "JSON" file.
  file.json

Syntax:

```
var product = {
   "Name": "Samsung TV",
   "Price" : 45000.44,
   "Stock": true,
   "Cities": ["Delhi", "Hyd"],
   "Rating": {"Rate":4.4, "Count":4500}
 }
```

- Object can contain both data and logic

Syntax:

```
var product =  {
   "Name":    "TV",
   "Price": 45000.44,
   "Qty": 3,
   "Total": function() {
    },
   "Print": function() {
   }
}
```

- You can access the properties and functions with in object by using "this" keyword.

```
"Total": function() {
    return this.Qty * this.Price;
 }
```

- You can access the properties and functions outside object by using object reference.

```
product.Qty
product.Total()
product.Print()
```

- Data is stored in property and logic is defined in function.

Ex:

```
<script>
    var tv = {
      "Name": "Samsung TV",
      "Price": 0,
      "Qty": 2,
      "Total": function(){
        return this.Qty * this.Price;
      },
      "Print": function(){
        document.write(`Name=${this.Name}<br>Price=${this.Price}<br>Qty=${this.Qty}<br>Total=${t
his.Total()}`);
      }
    }
   tv.Price = parseFloat(prompt("Enter Price"));
   tv.Print();
</script>
```

Note: Often JavaScript object is called as "Pseudo Class".

Array of Objects

Data Source: Table

ProductId	Name	Price
1	TV	35000.44
2	Mobile	12000.33

Middleware : API [Application Programming Interface]
To make the data reachable to every device and OS services.

Data Format: JSON

```
var products =
  [
    {"ProductId":1, "Name": "TV", "Price": 35000.44},       => 0
    {"ProductId":2, "Name": "Mobile", "Price": 12000.33}    => 1
  ]
```

Mobile Price : products[1].Price

- To access data from JSON file or API URL browser needs "XMLHttpRequest" object.
- JavaScript provides a promise called "fetch()" which uses XMLHttpRequest.
- Promise is a function that returns accurate status of any task performed.
  - Promise Fullfilled
  - Promise Rejected
  - Promise Failed

Syntax:

```
fetch("url | path")
```

```
        .then(function(){
            on success
        })
        .catch(function(){
            on failure
        })
        .finally(function(){
            always..
        })
```

Ex:

1. add a new folder into project

   "data"

2. add a new file into data folder

   "flipkart.json"

```
{
   "title": "vivo T1 44W (Midnight Galaxy, 128 GB)  (4 GB RAM)",
   "price": 14499,
   "actualPrice":17999,
   "offers": [
       "Bank Offer10% off on Bank of Baroda Credit Card EMI Transactions, up to ₹1,000 on orders of ₹5,000 and aboveT&C",
       "Bank Offer10% off on IDFC FIRST Bank Credit Card EMI Transactions, up to ₹1,000 on orders of ₹5,000 and aboveT&C",
       "Bank Offer10% off on IndusInd Bank Credit Card EMI Transactions, up to ₹1,000 on orders of ₹7,500 and aboveT&C",
       "Buy this Product and Get Extra ₹500 Off on Bikes & ScootersT&C"
   ],
   "rating": {"rate":4.5, "ratings":96490, "reviews": 5600},
   "photo": "../public/images/m1.jpg"
}
```

3. Add a new HTML page

   flipkart.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Flipkart</title>
   <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
   <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
   <script>
      function bodyload(){
```

```
        fetch("../data/flipkart.json")
        .then(function(response){
            return response.json();
        })
        .then(function(product){
            document.getElementById("productImage").src= product.photo;
            document.getElementById("productTitle").innerHTML = product.title;
            document.getElementById("productRating").innerHTML = product.rating.rate;
            document.getElementById("productReviews").innerHTML = `${product.rating.ratings}
Ratings & ${product.rating.reviews} Reviews`.fontcolor('gray').bold();
            document.getElementById("productPrice").innerHTML = `&#8377; ${product.price} <font
size='5' color='gray'><strike> &#8377; ${product.actualPrice}</strike></font>`;
            for(var offer of product.offers){
                var li = document.createElement("li");
                li.innerHTML = `<span class="bi bi-tag-fill text-success"></span> ${offer}`;
                li.className = "mb-2 mt-2";
                document.querySelector("ul").appendChild(li);
            }
        })
    }
  </script>
</head>
<body class="container-fluid" onload="bodyload()">
  <section class="row mt-4">
    <div class="col-3">
      <img width="300" height="400" id="productImage">
    </div>
    <div class="col-9">
      <h3 id="productTitle"></h3>
      <div class="d-flex">
        <div class="bg-success text-white p-1 rounded" style="width:60px">
          <span id="productRating"></span> <span class="bi bi-star-fill"></span>
        </div>
        <div class="ms-3">
          <span id="productReviews"></span>
        </div>
      </div>
      <div class="mt-4">
        <h1 id="productPrice"></h1>
      </div>
      <div>
        <h3>Offers</h3>
        <ul>

        </ul>
      </div>
    </div>
  </section>
</body>
</html>
```

Day 23 : Fakestore and NASA API – 31/03/23

Nasa API

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <script>
        function bodyload(){
            fetch("https://api.nasa.gov/mars-
photos/api/v1/rovers/curiosity/photos?sol=1000&api_key=6ffzhNjjV1XA2HkP9u2ghEmZVMK8Rb
2M2ZG4n6FI&quot;)
            .then(function(res){
                return res.json();
            })
            .then(function(marsObject){
                console.log(marsObject);
                for(var item of marsObject.photos)
                {
                    var div = document.createElement("div");
                    div.className = "card p-2 m-2";
                    div.style.width = "200px";
                    div.innerHTML = `
                        <img src=${item.img_src} height="150" class="card-img-top">
                        <div class="card-header">
                            <h2>${item.id}</h2>
                        </div>
                        <div class="card-body">
                            <dl>
                              <dt>Camera</dt>
                              <dd>${item.camera.full_name}</dd>
                              <dt>Rover</dt>
                              <dd>${item.rover.name}</dd>
                            </dl>
                        </div>
                    `;
                    document.querySelector("main").appendChild(div);
                }
            })
        }
    </script>
</head>
<body class="container-fluid" onload="bodyload()">
    <h2>Mars Rover Photos</h2>
    <main class="d-flex flex-wrap">
```

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

Ex:
products.json

```
[
    {
        "Name": "Samsung TV",
        "Price": 45000.44,
        "Rating": {"Rate":4.3, "Count":5000}
    },
    {
        "Name": "Nike Casuals",
        "Price": 5700.33,
        "Rating": {"Rate":3.6, "Count":3200}
    },
    {
        "Name": "Watch",
        "Price": 3500.33,
        "Rating":{"Rate":4.2, "Count":5000}
    }
]
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function bodyload(){
            fetch("../data/products.json")
            .then(function(response){
                return response.json();
            })
            .then(function(products){
                for(var product of products){
                    var tr = document.createElement("tr");
                    var tdName = document.createElement("td");
                    var tdPrice = document.createElement("td");
                    var tdRating = document.createElement("td");

                    tdName.innerHTML = product.Name;
                    tdPrice.innerHTML = product.Price;
                    tdRating.innerHTML = `${product.Rating.Rate} [${product.Rating.Count}]`;
```

```
                tr.appendChild(tdName);
                tr.appendChild(tdPrice);
                tr.appendChild(tdRating);

                document.querySelector("tbody").appendChild(tr);
            }
        })
    }
  </script>
</head>
<body onload="bodyload()">
  <table border="1" width="400">
    <thead>
      <tr>
        <th>Name</th>
        <th>Price</th>
        <th>Rating</th>
      </tr>
    </thead>
    <tbody>

    </tbody>
  </table>
</body>
</html>
```

Fakestore API

GET Requests

http://fakestoreapi.com

/products	[{ }, { }] 20
/products/1	{ } specific product by id
/products/categories	[""] list of all categories
/products/category/jewelery	[{},{}] specific category products

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Shopper|Online Shopping</title>
  <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
  <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
  <script type="text/javascript">
```

```
function LoadCategories(){
    fetch("http://fakestoreapi.com/products/categories&amp;quot;)
    .then(function(response){
        return response.json();
    })
    .then(function(categories){
        categories.unshift("all");
        for(var category of categories){
            var option = document.createElement("option");
            option.text = category.toUpperCase();
            option.value = category;
            document.getElementById("lstCategories").appendChild(option);
        }
    })
}
function LoadProducts(url){
    document.querySelector("main").innerHTML="";
    fetch(url)
    .then(function(response){
        return response.json();
    })
    .then(function(products){
        for(var product of products){
            var card = document.createElement("div");
            card.className = "card p-2 m-2";
            card.style.width = "200px";
            card.innerHTML = `
                <img src=${product.image} class="card-img-top" height="100">
                <div class="card-header overflow-auto" style="height:80px">
                    <p>${product.title}</p>
                </div>
                <div class="card-body">
                    <dl>
                        <dt>Price</dt>
                        <dd>${product.price}</dd>
                        <dt>Rating</dt>
                        <dd>
                            <span class="bi bi-star-fill text-success"> </span>
                            ${product.rating.rate} [${product.rating.count}]
                        </dd>
                    </dl>
                </div>
                <div class="card-footer">
                    <button onclick="AddToCartClick(${product.id})" class="btn btn-danger w-100">
                     <span class="bi bi-cart4"></span> Add to Cart
                    </button>
                </div>
            `;
            document.querySelector("main").appendChild(card);
        }
    })
```

```
}
function bodyload(){
    LoadCategories();
    LoadProducts("http://fakestoreapi.com/products&quot;);
    GetCartCount();
}
function CategoryChanged(){
    var categoryName = document.getElementById("lstCategories").value;
    if(categoryName=="all"){
        LoadProducts("http://fakestoreapi.com/products&quot;);
    } else {
        LoadProducts(`http://fakestoreapi.com/products/category/${categoryName}`);
    }
}
function NavClick(categoryName){
    document.getElementById("lstCategories").value = categoryName;
    if(categoryName=="all"){
        LoadProducts("http://fakestoreapi.com/products&quot;);
    } else {
        LoadProducts(`http://fakestoreapi.com/products/category/${categoryName}`);
    }
}
var cartItems = [];
function GetCartCount(){
    document.getElementById("cartCount").innerHTML = cartItems.length;
}
function AddToCartClick(id) {
    fetch(`http://fakestoreapi.com/products/${id}`)
    .then(function(response){
        return response.json();
    })
    .then(function(product){
        alert(`${product.title}\nAdded to Cart`);
        cartItems.push(product);
        GetCartCount();
    })
}
function LoadCartItems(){
    document.querySelector("tbody").innerHTML= "";
    for(var item of cartItems){
        var tr = document.createElement("tr");
        var tdTitle = document.createElement("td");
        var tdImage = document.createElement("td");
        var tdPrice = document.createElement("td");
        var tdAction = document.createElement("td");

        tdTitle.innerHTML = item.title;
        tdImage.innerHTML = `<img src=${item.image} width="50" height="50">`;
        tdPrice.innerHTML = item.price;
        tdAction.innerHTML = `
         <button class="bi bi-trash-fill btn btn-danger"></button>
```

```
                `;

                tr.appendChild(tdTitle);
                tr.appendChild(tdImage);
                tr.appendChild(tdPrice);
                tr.appendChild(tdAction);

                document.querySelector("tbody").appendChild(tr);
            }
        }
    </script>
    <style>
        a {
            color:white;
            text-decoration: none;
        }
        a:hover {
            color:yellow;
        }
    </style>
</head>
<body class="container-fluid" onload="bodyload()">
    <header class="d-flex justify-content-between p-2 bg-dark text-white mt-2">
        <div>
            <span style="font-size:25px; font-weight:bold">Shopper.</span>
        </div>
        <div style="font-size: 20px;">
            <span class="me-3"><a href="javascript:NavClick('all')">Home</a></span>
            <span class="me-3"><a href="javascript:NavClick('electronics')">Electronics</a></span>
            <span class="me-3"><a href="javascript:NavClick('jewelery')">Jewelery</a></span>
            <span class="me-3"><a href="javascript:NavClick('men\'s clothing')">Men's
Fashion</a></span>
            <span class="me-3"><a href="javascript:NavClick('women\'s clothing')">Women's
Fashion</a></span>
        </div>
        <div>
            <span class="bi bi-search me-3"></span>
            <span class="bi bi-person me-3"></span>
            <span class="bi bi-heart me-3"></span>
            <button onclick="LoadCartItems()" data-bs-target="#cart" data-bs-toggle="modal"
class="btn text-white position-relative">
                <span class="bi bi-cart"></span>
                <span id="cartCount" class="badge bg-danger position-absolute rounded rounded-
circle"></span>
            </button>
            <div class="modal fade" id="cart">
                <div class="modal-dialog">
                    <div class="modal-content">
                        <div class="modal-header">
                            <h4 class="text-primary">Your Cart Summary</h4>
                            <button class="btn-close" data-bs-dismiss="modal"></button>
```

```
                </div>
                <div class="modal-body">
                  <table class="table table-hover">
                    <thead>
                      <tr>
                        <th>Title</th>
                        <th>Preview</th>
                        <th>Price</th>
                      </tr>
                    </thead>
                    <tbody>

                    </tbody>
                  </table>
                </div>
              </div>
            </div>
          </div>
        </div>
      </header>
      <section class="mt-3 row">
        <nav class="col-2">
          <div>
            <label class="form-label fw-bold">Select Category</label>
            <div>
              <select onchange="CategoryChanged()" class="form-select" id="lstCategories">

              </select>
            </div>
          </div>
        </nav>
        <main class="col-10 d-flex flex-wrap overflow-auto" style="height:500px">

        </main>
      </section>
      <script src="../node_modules/jquery/dist/jquery.js"></script>
      <script src="../node_modules/bootstrap/dist/js/bootstrap.bundle.js"></script>
    </body>
    </html>
```

Day 24 : Object Types – 01/04/23

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
```

```
    <script>
        var data = [
            {Category:"Electronics", Products:["Televisions","Mobiles"]},
            {Category:"Footwear", Products:["Casuals", "Boots"]}
        ];

        function bodyload(){
            for(var item of data) {
                var li = document.createElement("li");
                li.innerHTML = item.Category;

                var optgroup = document.createElement("optgroup");
                optgroup.label = item.Category;

                for(var product of item.Products){
                    var ul = document.createElement("ul");
                    var ulLi = document.createElement("li");
                    ulLi.innerHTML = product;
                    ul.appendChild(ulLi);
                    li.appendChild(ul);

                    var option = document.createElement("option");
                    option.text = product;
                    option.value = product;
                    optgroup.appendChild(option);
                }
                document.querySelector("ol").appendChild(li);
                document.querySelector("select").appendChild(optgroup);
            }
        }
    </script>
</head>
<body onload="bodyload()">
    <ol>

    </ol>
    <select>

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

Ex: Details and Summary

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Document</title>
    <script>
        var data = [
            {Category:"Electronics", Products:["Televisions","Mobiles"]},
            {Category:"Footwear", Products:["Casuals", "Boots"]}
        ];

        function bodyload(){
            for(var item of data) {
                var details = document.createElement("details");
                var summary = document.createElement("summary");
                summary.innerHTML = item.Category;
                details.appendChild(summary);
                for(var product of item.Products){
                    var div = document.createElement("div");
                    div.innerHTML = product;
                    div.style.marginBottom = "20px";
                    div.style.marginLeft = "30px";
                    details.appendChild(div);
                }
                document.getElementById("menu").appendChild(details);
            }
        }
    </script>
</head>
<body onload="bodyload()">
    <div id="menu">

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

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        var data = [
            {Topic: "JavaScript", Description: "JavaScript is a language."},
            {Topic: "HTML", Description: "It is a markup language."}
        ]
        function bodyload(){
            for(var item of data){
                var dt = document.createElement("dt");
                var dd = document.createElement("dd");
                dt.innerHTML = item.Topic;
                dd.innerHTML = item.Description;
```

```
            document.querySelector("dl").appendChild(dt);
            document.querySelector("dl").appendChild(dd);
        }
    }
  </script>
</head>
<body onload="bodyload()">
  <dl>

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

Array
Object
Array of Objects
Embedded Object

FAQ: How to get the list of all properties or keys in an object?
Ans : for..in operator

FAQ: How to get the list of both properties and their type?
Ans: typeof operator

Ex:

```
<script>
  fetch("http://fakestoreapi.com/products/1&quot;)
  .then(function(response){
    return response.json();
  })
  .then(function(product){
    for(var property in product){
      document.write(`${property} [${typeof product[property]}] => ${product[property]}  <br>`);
    }
  })
</script>
```

FAQ: How to check for any property in object?
Ans: by using "in" operator

```
    "propertyName"  in  objectName   => true / false
```

EX:

```
<script>
  fetch("http://fakestoreapi.com/products/1&quot;)
  .then(function(response){
    return response.json();
  })
  .then(function(product){
```

```
        for(var property in product){
            document.write(`${property} [${typeof product[property]}] => ${product[property]}  <br>`);
        }
        document.write(`Is Stock property available in Product : ${"Stock" in product}`);
    })
</script>
```

FAQ: How to remove any property from object?
Ans: by using "delete" operator

Ex:

```
<script>
    fetch("http://fakestoreapi.com/products/1&quot;)
    .then(function(response){
        return response.json();
    })
    .then(function(product){
        delete product.category;
        delete product.rating;
        for(var property in product){
            document.write(`${property} [${typeof product[property]}] => ${product[property]}  <br>`);
        }
    })
</script>
```

FAQ: What are the issues with Object type data?
Ans: All manipulations are done explicitly
It is slow.
You need lot of explicit techniques to handle object manipulaions.
[delete property, find property, get all properties etc...]

Day 25 : JavaScript Data Types – 03/04/23

What are the issues with Object type?

1. All keys are string type.
2. There is not size of keys. You can't get the count of keys by using any implicit technique.
3. You need various explicit iterators for reading properties and values.
4. You need explicit operators and methods to check the availablility of any key and delete a key.
5. It is slow in access.

Map Type

- It is a key - value collection same like object.
- Key's can be any type.
- It provides implicit iterators and methods for reading and manipulation.
- It is faster in access.
- It is schema less. [Structure less]

Note: If you are dealing with structured data then use object, if it is schema less then use map

type.

Syntax:

```
var  refName = new Map();
```

- Map provides following methods for manipulation

set()	It is used to assign a new key with value.
get()	It is used to access a value with reference of key.
keys()	It returns all keys
values()	It returns all values
entries()	It returns all keys and values
delete()	It remove a key and value.
has()	It checks the availablility of any key.
size	It returns the total count of keys.
clear()	It removes all keys

Ex:

```
<script>
   var data = new Map();
   data.set(1, "Samsung TV");
   data.set("HTML", "Iti s a markup language");
   document.write(data.get("HTML"));
</script>
```

Ex:

```
<script>
   var data = new Map();
   data.set(1, "Samsung TV");
   data.set("HTML", "It is a markup language");
   data.delete(1);
   if(data.has(1)) {
      document.write(data.get(1))
   } else {
      document.write(`There no key by name 1 <br>`);
   }
   document.write(`Total Number of Keys : ${data.size}<br>`);
   for(var item of data.entries()){
      document.write(item + "<br>");
   }
</script>
```

Date Type

- Date type is defined by using JavaScript "Date()" constructor.
- It allocates memory for storing date type value.
- Date type is stored in "Year-Month-Day" format.
- It can handle both date and time values.

Syntax:

```
var  mfd = new Date();              => loads the current system date and time

var mdf = new Date("YYYY-MM-DD  Hrs:Min:Sec.MilliSeconds");
var mdf = new Date("YYYY-MM-DD");
```

- JavaScript date object provides various date and time methods.

```
getHours()              It returns hour number 0 to 23
getMinutes()            It returns minutes number 0 to 59
getSeconds()            It returns seconds number 0 to 59
getMilliSeconds()       It returns milli seconds number 0 to 99
getDay()                It returns weekday number  0=Sunday
getDate()               It returns date number 1 to 28,29,30,31
getMonth()              It returns month number 0=January
getFullYear()           It returns full year [4 chars]
getYear() [obsolete]       It returns year number as per Y2K
toLocaleDateString()
toLocaleTimeString()
toString()

setHours()
setMinutes()
setSeconds()
setMilliSeconds()
setDate()
setMonth()
setYear()
```

EX:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
   <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
   <script>
      function GetTime(){
         var now = new Date();
         document.getElementById("time").innerHTML = now.toLocaleTimeString();
      }
      function bodyload(){
         var weekdays = ["Sunday", "Monday", "Tuesday", "Wed","Thr","Friday","Saturday"];
         var months = ["Jan", "Feb", "Mar", "April", "May", "June"];

         var msg = document.getElementById("msg");
         var now = new Date();
         if(now.getHours()>=0 &&  now.getHours()<=12){
```

```
            msg.innerHTML = "Good Morning";
          } else if (now.getHours()>12 && now.getHours()<=17) {
            msg.innerHTML = "Good Afternoon";
          } else {
            msg.innerHTML = "Good Evening";
          }
          now.setMonth(2);
          document.getElementById("cal").innerHTML = `${weekdays[now.getDay()]}
${now.getDate()} ${months[now.getMonth()]}, ${now.getFullYear()}`;
          setInterval(GetTime, 1000);
        }
    </script>
</head>
<body class="container-fluid" onload="bodyload()">
    <div class="btn-toolbar bg-danger justify-content-between">
        <div class="btn-group">
            <button class="btn btn-danger">Home</button>
            <button class="btn btn-danger">About</button>
            <button class="btn btn-danger">Contact</button>
        </div>
        <div class="btn-group">
            <button class="btn btn-danger">
                <span id="msg"> </span>
            </button>
        </div>
        <div class="btn-group">
            <button class="btn btn-danger">
                <span class="bi bi-calendar" id="cal"></span>
            </button>
            <button class="btn btn-danger">
                <span class="bi bi-clock" id="time"></span>
            </button>
        </div>

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

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function DateChange(){
            var weekdays = ["Sunday", "Monday", "Tuesday", "Wed","Thr","Friday","Saturday"];
```

```
        var months = ["Jan", "Feb", "Mar", "April", "May", "June"];

        var now= new Date(document.getElementById("date").value);
        document.querySelector("p").innerHTML =`${weekdays[now.getDay()]}`;
      }
    </script>
</head>
<body>
    Departure : <input type="datetime-local" onchange="DateChange()" id="date">
    <p></p>
</body>
</html>
```

Day 26 : Operators – 04/04/23

Variables
Data Types
Primitive Types
- number
- string
- boolean
- null
- undefined
- symbol

Non Primitive Types
- array
- object
- map

Date Type

JavaScript Operators

- Operator is a object in computer programming that evaluates are value based on the operands.
- Operators are classified into various types based on the number of operands they can handle.

a + b => a, b are operands

- Operand usually stores the data.
- Operators based on number of operands are classified into
    a) Unary Operator
    b) Binary Operator
    c) Ternary Operator

- Unary operator handles only one operand.

    x++, --y

- Binary operator handle two operands.

    x + y

- Ternary operator handles 3 operands

(condition)?true:false

one?two:three

- JavaScript operators are again classified into various groups based on the type of value they return

1. Arithematic Operators
2. Comparision Operators
3. Assignment Operators
4. Logical Operators
5. Bitwise Operators
6. Special Operators

Arithematic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus Division
**	Exponent [Math.pow()] New in ES5+ 2**3 = 8 => Math.pow(2,3)
++	Increment
--	Decrement

Increment
- Post
- Pre

```
var  x = 10;
var  y = x++;          Post Increment       x = x + 1
                  x = 11, y=10
```

Post increment specifies that increment by 1 is done after assigning.

```
var x  = 10;
var y = ++x;          Pre Increment       x = x + 1
                  x = 11, y=11
```

Pre increment specifies that first increment is done and later it is assigned.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
   <script>
```

```
        var count=0;
        function ShowStatus(){
            count++;
            if(count==100){
                location.href="shopper-template.html";
            } else {
                document.getElementById("status").innerHTML = count +  " % ";
            }
        }
        function LoadClick(){
            document.getElementById("loading").style.display="block";
            document.querySelector("button").style.display = "none";
            setInterval(ShowStatus,200);
        }
    </script>
</head>
<body class="container-fluid">
    <div class="d-flex justify-content-center align-items-center" style="height: 400px;">
        <div>
            <button onclick="LoadClick()" class="btn btn-primary">Load Template</button>
            <div class="text-center" id="loading" style="display: none;">
                <span class="spinner-border"></span>
                <p id="status"></p>
                <div>Loading</div>
            </div>
        </div>
    </div>
</body>
</html>

Ex:
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <script>
        var count=0;
        function ShowStatus(){
            count++;
            document.getElementById("progress").value = count;
            if(count==100){
                location.href="shopper-template.html";
            } else {
                document.getElementById("status").innerHTML = count +  " % ";
            }
        }
        function LoadClick(){
```

```
            document.getElementById("loading").style.display="block";
            document.querySelector("button").style.display = "none";
            setInterval(ShowStatus,200);
        }
    </script>
</head>
<body class="container-fluid">
    <div class="d-flex justify-content-center align-items-center" style="height: 400px;">
        <div>
            <button onclick="LoadClick()" class="btn btn-primary">Load Template</button>
            <div class="text-center" id="loading" style="display: none;">
                <progress id="progress" min="1" max="100" value="1"></progress>
                <p id="status"></p>
                <div>Loading</div>
            </div>
        </div>
    </div>
</body>
</html>


Ex:
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
    <script>
        var id = 1;
        function LoadProduct(){
            id++;
            fetch(`http://fakestoreapi.com/products/${id}`)
            .then(function(res){
                return res.json();
            })
            .then(function(product){
                document.getElementById("title").innerHTML = product.title;
                document.getElementById("poster").src = product.image;
                document.getElementById("price").innerHTML = `&#8377; ${product.price}`;
            })
        }
        function bodyload(){
            LoadProduct();
        }
        function NextClick(){
            id++;
            LoadProduct();
```

```
        }
        function PrevClick(){
            id--;
            LoadProduct();
        }
        var timerMemory;
        function PlayClick(){
            timerMemory = setInterval(LoadProduct,5000);
            document.getElementById("status").innerHTML = "(Slide Show - Started)";
        }
        function PauseClick(){
            clearInterval(timerMemory);
            document.getElementById("status").innerHTML = "(Slide Show - Paused)";
        }
    </script>
    <style>
        #price {
            width: 120px;
            height: 80px;
            border: 1px solid black;
            border-radius: 100px;
            background-color: green;
            color:white;
            position: absolute;
            top: 100px;
            right: 150px;
            padding: 20px;
            text-align: center;
            font-size: 20px;
        }
    </style>
</head>
<body class="container-fluid" onload="bodyload()">
    <div id="price">

    </div>
    <div class="d-flex justify-content-center align-items-center" style="height: 500px;">
        <div class="card w-50 p-2">
            <div class="card-header text-center">
                <p id="title"></p>
            </div>
            <div class="card-body">
                <div class="row">
                    <div class="col-2 d-flex flex-column align-items-center">
                        <button class="btn btn-primary" onclick="PrevClick()">
                            <span class="bi bi-chevron-left"></span>
                        </button>
                    </div>
                    <div class="col-8 text-center">
                        <img id="poster" width="100%" height="300">
                        <h4 id="status"></h4>
```

```
            </div>
            <div class="col-2">
              <button class="btn btn-primary" onclick="NextClick()">
                <span class="bi bi-chevron-right"></span>
              </button>
            </div>
          </div>
        </div>
        <div class="card-footer text-center">
          <button class="btn btn-success" onclick="PlayClick()">
            <span class="bi bi-play"></span>
          </button>
          <button class="btn btn-danger" onclick="PauseClick()">
            <span class="bi bi-pause"></span>
          </button>
        </div>
      </div>
    </div>
</body>
</html>
```

Comparision Operators

```
>          greater than
>=          greater than or equal
<          less than
<=          less than or equal
==          equal
===        identical equal
!=          not equal
!==          not identical
```

Note: All comparion operators return boolean

```
        var x = "10";
        var y = 10;
        x = y;            => assign
        x == y;           => true    [converts and compares]
        x === y;          => false  [true only when values are same type]
```

Ex:

```
<script>
   var x = 10;
   var y = "10";
   document.write(`x(${typeof x})==y(${typeof y}) ? ` + (x==y) + "<br>");
   document.write(`x(${typeof x})===y(${typeof y}) ? ` + (x===y) + "<br>");

</script>
```

Day 27 : JS Assignment & Conditional Operators – 05/04/23

Arithematic Operators

string + string ? string
string + number ? string
string + boolean ? string

number+number ? number
number+boolean ? number => 1 + true => 2
number+string ? string

boolean+boolean ? number => true + true = 2

string - string ? NaN
string * string ? NaN
string / string ? NaN

Logical Operators

&& AND
|| OR
! NOT

(condition1) && (condition2) => true if all conditions true
false if any one condition is false.

(condition1) || (condition2) => true if any one condition is true.
=> false if all conditions are false.

! Not => !true = false
!false = true

Assignment Operators

+= Add and assign
-= sub and assign
/= divide and assign
*= multiply and assign
%= modulus and assign

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
```

```
  <script>
    var margin = 1;
    function MoveClick(){
      margin *= 2;
      document.getElementById("pic").style.marginLeft = margin + "px";
    }
    function ClearClick(){
      margin=1;
      document.getElementById("pic").style.marginLeft = margin + "px";
    }
  </script>
</head>
<body>
  <button onclick="MoveClick()">Move</button>
  <button onclick="ClearClick()">Clear</button>
  <div>
    <img src="images/shoe.jpg" width="100" height="100" id="pic">
  </div>
</body>
</html>
```

- JavaScript provides DOM methods to accessing multiple elements

    document.getElementsByTagName() [ ]
    document.getElementsByName() [ ]
    document.getElementsByClassName() [ ]

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    var categoryName = "";
    function SubmitClick(){
      categoryName = "";
      var categoryCheckBoxes = document.getElementsByName("Category");
      for(var checkbox of categoryCheckBoxes){
        if(checkbox.checked) {
          categoryName += checkbox.value + "<br>";
        }
      }
      document.querySelector("p").innerHTML = categoryName;
    }
  </script>
```

```
</head>
<body>
   <h3>Categories</h3>
   <ul style="list-style: none;">
      <li><input type="checkbox" name="Category" value="Electronics"> Electronics</li>
      <li><input type="checkbox" name="Category" value="Footwear"> Footwear</li>
      <li><input type="checkbox" name="Category" value="Kids Fashion"> Kids Fashion</li>
      <li><input type="checkbox" name="Category" value="Jewelery"> Jewelery </li>
   </ul>
   <button onclick="SubmitClick()">Submit</button>
   <p></p>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      var categoryName = "";
     function CategoryChanged(){
       categoryName = "";
       var categoryCheckBoxes = document.getElementsByName("Category");
       for(var checkbox of categoryCheckBoxes){
          if(checkbox.checked) {
             categoryName += checkbox.value + "<br>";
          }
       }
       document.querySelector("p").innerHTML = categoryName;
     }
   </script>
</head>
<body>
   <h3>Categories</h3>
   <ul style="list-style: none;">
      <li><input type="checkbox" onchange="CategoryChanged()" name="Category"
value="Electronics"> Electronics</li>
      <li><input type="checkbox" onchange="CategoryChanged()" name="Category"
value="Footwear"> Footwear</li>
      <li><input type="checkbox" onchange="CategoryChanged()" name="Category" value="Kids
Fashion"> Kids Fashion</li>
      <li><input type="checkbox" onchange="CategoryChanged()" name="Category"
value="Jewelery"> Jewelery </li>
   </ul>
   <p></p>
</body>
```

```
</html>
```

Ex:

```
<script>
    var products = [
        {Name:"TV", Category:"Electronics"},
        {Name:"Nike Casuals", Category:"Footwear"},
        {Name:"Shirt", Category:"Fashion"},
        {Name:"Watch", Category:"Electronics"}
    ];
    var result = products.filter(function(product){
        return product.Category=="Footwear" || product.Category=="Fashion";
    })
    for(var item of result) {
        document.write(item.Name + "<br>");
    }
</script>
```

Ex:

```
<script>
    fetch("http://fakestoreapi.com/products&amp;quot;)
    .then(function(res){
        return res.json();
    })
    .then(function(products){
        var result = products.filter(function(product){
            return product.category=="electronics" || product.category=="jewelery";
        });
        for(var item of result) {
            document.write(item.title + "-" + item.category + "<br>");
        }
    })
</script>
```

Day 28 : Statements – 06/04/23

```
<!DOCTYPE html>
 <html lang="en">
 <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        var userDetails = {
          "Card": "4444555566667890",
          "Cvv": "4534",
          "Expiry": "2024"
        }
        function VerifyCard(){
```

```
        var card  = document.getElementById("Card").value;
        if(card==userDetails.Card) {
          document.getElementById("Cvv").disabled=false;
          document.getElementById("Cvv").focus();
          document.getElementById("cvvContainer").style.display="block";
          document.getElementById("Card").readOnly = true;
        }
      }
      function VerifyCvv(){
        var cvv  = document.getElementById("Cvv").value;
        if(cvv==userDetails.Cvv) {
          document.getElementById("Expiry").disabled=false;
          document.getElementById("expiryContainer").style.display="block";
        }
      }
      function VerifyExpiry(){
        var expiry  = document.getElementById("Expiry").value;
        if(expiry==userDetails.Expiry) {
          document.getElementById("btnPay").disabled=false;
          document.getElementById("btnPay").style.display="block";
        }
      }
    </script>
</head>
<body>
    <fieldset>
      <legend>Payment Details</legend>
      <dl>
        <dt>Card Number</dt>
        <dd><input type="text" id="Card" onblur="VerifyCard()"></dd>
        <div id="cvvContainer" style="display: none;">
          <dt>Cvv</dt>
          <dd><input type="text" id="Cvv" onblur="VerifyCvv()" size="4" disabled></dd>
        </div>
        <div id="expiryContainer" style="display: none;">
          <dt>Expiry</dt>
          <dd>
            <select id="Expiry" onchange="VerifyExpiry()" disabled>
              <option>2023</option>
              <option>2024</option>
              <option>2025</option>
            </select>
          </dd>
        </div>
      </dl>
      <button id="btnPay" disabled style="display: none;">Pay</button>
    </fieldset>
</body>
</html>
```

Day 29 : If Select and Switch – 07/04/23

Selection Statements
- IF Select
  a) Forward Jump
  b) Simple Decision

c) Multi Level Decisions

- In this approach developer will test only one condition and defined multi level conditions to verify multiple values.
- It is not recommended to write all conditions in one line, if there are many status messages to display.

Syntax:

```
if (condition1)
{
   if(condition2) {
      statement when all conditions evaluate to true;
   }
   else {
    statement if condition 2 is false;
   }
}
else
{
  statement if condition-1 false;
}
```

Summary:
- If many conditions are available for test.
- There is a order for testing conditions
- Have to display individual errors messages of every condition

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      var userDetails = {
         "UserName": "john_nit",
         "Password": "john@123"
      }
      function LoginClick(){
         var username = document.getElementById("txtName").value;
         var password = document.getElementById("txtPwd").value;
```

```
        var error = document.querySelector("p");

        if(username===userDetails.UserName)
        {
            if(password===userDetails.Password){
                document.write("Login Success");
            } else {
                error.innerHTML = "Invalid Password".fontcolor('red');
            }
        } else {
            error.innerHTML = "Invalid User Name".fontcolor('red');
        }
    }
  </script>
</head>
<body>
  <dl>
    <dt>User Name</dt>
    <dd><input type="text" id="txtName"></dd>
    <dt>Password</dt>
    <dd><input type="password" id="txtPwd"></dd>
  </dl>
  <button onclick="LoginClick()">Login</button>
  <p align="center"></p>
</body>
</html>
```

d) Multiple Choices

- It is a decision making approache where we provide multiple choices for handling a specific task.
- User can choose any one out of multiple choices.
- It must continue to with the selected choice.

Syntax:

```
    if (choice-1)
    {
      statement on choice1;
    }
     else if(choice-2)
    {
       statement on choice2;
    }
     else if(choice-3)
    {
      statement on choice3;
    }
    else
    {
       statement when it is not among the given choices
     }
```

Ex: Amazon Login

data/user.json

```
{
   "Email": "john11@gmail.com",
   "Mobile": "+919876543210",
   "Password": "john@123"
}
```

login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Amazon Login</title>
   <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
   <script>
       var flag = "";
       function ContinueClick(){
         var userid = document.getElementById("txtUserId").value;
         var userError = document.getElementById("userError");
         var userContainer = document.getElementById("userContainer");
         var passwordContainer = document.getElementById("passwordContainer");

         function ToggleContainers(){
               userContainer.style.display = "none";
               passwordContainer.style.display = "block";
         }

         fetch("../data/user.json")
         .then(function(res){
            return res.json();
         })
         .then(function(user){
             if(userid===user.Email) {
               flag = `Login Success and verification code sent to your registered email ${userid}`;
               ToggleContainers();

             } else if(userid===user.Mobile) {
               flag = `Login Success and OTP sent to registered mobile ${userid}`;
               ToggleContainers();

             } else {
               userError.innerHTML = `${userid} doesn't exist`;
             }
```

```
            })
        }
        function LoginClick(){
            var password = document.getElementById("txtPwd").value;
            fetch("../data/user.json")
            .then(function(res){
                return res.json();
            })
            .then(function(user){
                if(user.Password===password) {
                    document.write(`<h2>${flag}</h2>`);
                } else {
                    document.getElementById("pwdError").innerHTML = "Invalid Password";
                }
            })
        }
    </script>
</head>
<body class="container-fluid">
    <div class="d-flex justify-content-center align-items-center" style="height:500px">
        <div class="w-25">
            <h2>SignIn</h2>
            <div id="userContainer">
                <div class="mb-3">
                <label class="form-label fw-bold">Email or mobile phone number</label>
                <div>
                    <input type="text" id="txtUserId" class="form-control">
                    <div class="text-danger" id="userError"></div>
                </div>
                </div>
                <div>
                    <button onclick="ContinueClick()" class="btn btn-warning w-100">Continue</button>
                </div>
            </div>
            <div id="passwordContainer" style="display:none">
                <div class="mb-3">
                    <label class="form-label fw-bold">Password</label>
                    <div>
                        <input type="password" id="txtPwd" class="form-control">
                        <div id="pwdError" class="text-danger"></div>
                    </div>
                </div>
                <div>
                    <button class="btn btn-warning w-100" onclick="LoginClick()">Login</button>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

FAQ: What is issue with multiple choices?
Ans: Compiler can't choose directly the condition, which is matching. It have to verify all conditions until the specified is true.

Switch Selector

- A switch in electronics is used to interrupt the flow of electrons.
- There are various types of switches
  - push button
  - toggle switch
  - joystick
  - selector switch etc..
- A selector switch can choose exactly the statement that matches given condition, it will not execute the other blocks of statements.

Syntax:

```
switch(value)
{
  case  value1:
   statement;
   jump_statement;
  case value2:
   statement;
   jump_statement;
  default:
   statement if any value is not matching;
   jump_statement;
}
```

Ex:

```
<script>
   var n = parseInt(prompt("Enter Number"));
   switch(n)
   {
      case 1:
      document.write("One");
      break;
      case 2:
      document.write("Two");
      break;
      case 3:
      document.write("Three");
      break;
      case 4:
      document.write("Four");
      break;
      default:
      document.write("Please Enter 1 to 4 only");
      break;
   }
</script>
```

Day 30 : Conditional Programs Test – 08/04/23

1. Write a program to find if the given number is with in the specified range?

```
function  FindRange(number, min, max) {

    }

FindRange(3, 1, 10)          => In range
FindRange(12, 20,30)       => Out of range
```

Ex:

```
<script>
   function FindRange(n, min, max) {
      if(n>=min && n<=max) {
         document.write(`Your number ${n} is in range of ${min} - ${max}`);
      } else {
         document.write(`Your number ${n} is out of range ${min} - ${max}`);
      }
   }
   FindRange(14, 10, 20);
</script>
```

2. Write a program to find the largest among 3 numbers.

```
function  FindLargest(a, b, c) {

  }

FindLargest(10, 5, 19);
```

Ex:

```
<script>
  function FindLargest(a,b,c){
     if(a>b && a>c) {
         document.write("A is Large");
     } else if (b>c) {
         document.write("B is Greater");
     } else {
         document.write("C is Greater");
     }
  }
  FindLargest(10,34, 64);
</script>
```

3. Write a program to find if the given values are for a triangle or square?

```
function  FindShape(values)
{
}
FindShape([100, 200, 50]);   => It is a triangle
FindShape([10, 20, 40, 50]);    => It is a square
```

Ex:

```
<script>
  function FindShape(values){
      if(values.length==3) {
         document.write("Values are for Triangle");
      } else if (values.length==4) {
         document.write("values are for Square");
      } else {
         document.write("Please provide values only for square or triangle");
      }
  }
  FindShape([10,40,20,20, 20]);
</script>
```

4. Write a program to find if the values are for a square or rectangle?

```
function FindSquare(side1, side2, side3, side4)
{
}
```

Ex:

```
<script>
   function FindSquare(side1, side2, side3, side4) {
      if(side1==side2 && side2==side3 && side3==side4) {
         document.write("Values are for Square");
      } else {
         document.write("Values are for Rectangle");
      }
   }
   FindSquare(10,20,20,10);
</script>
```

5. Write a program to find the given number is an even or odd ?

```
function  FindEvenOdd(number)
{
    if(number % 2 == 0)
    {
       even;
    } else {
       odd;
    }
}
```

6. Write to program to Check if a triangle is equilateral, scalene, or isosceles

```
function findTrangleType(side1, side2, side3) {

  }
```

side1==side2 and side1==side3 = Equilateral
side1==side2 or side2==side3 or side1==side3 = Isosceles

```
findTriangleType(12,12,12) //"Equilateral triangle."
findTriangleType(20,20,31) //"Isosceles triangle."
findTriangleType(5,4,3) //"Scalene triangle."
```

7. Write a program to find the x,y values are in origin or in which quadrant.

8. Write a program to convert number into words?
453 = Four Hundred Fifty Three

Day 31 : Switch Case – 10/04/23

Switch Statement
- It is a selector that select and executes only the block that matches given condition.

Syntax:

```
switch(value)
{
  case  value:
   statement;
   jump_statement;
 default:
   statement;
   jump_statement;
}
```

Ex:

```
<script>
   var n = parseInt(prompt("Enter Number"));
   switch(n)
   {
     case 1:
     document.write("One");
     break;
     case 2:
```

```
      document.write("Two");
      break;
      case 3:
      document.write("Three");
      break;
      default:
      document.write("Please Enter value between 1 to 3 only");
      break;
    }
</script>
```

FAQ's:

1. Can we define switch without default?

A. Yes.

2. Can we define default before case or between case?

A. Yes.

3. Can we define case or default without break?

A. Yes.

4. Can we define "return" as jump for case or default?

A. Yes.

5. What is difference between break and return?

A. "break" will terminate the block but still stays in function. Execution will not stop.

"return" will terminate the block and stop execution, It exits the script.

Note: return keyword can be used only in a function.

6. Can we define "string" as case value?

A. Yes

```
<script>
   function f1(){
     var n = prompt("Your choice", "y/n");
    switch(n)
    {
      case "y":
      document.write("You Selected Yes");
      break;
      case "n":
      document.write("You Selected No.");
      break;
      default:
      document.write("Please enter y or n");
      break;
    }
   }
```

```
   f1();
</script>
```

7. Can we define multiple cases for same block of statement?
A. Yes

```
<script>
   function f1(){
     var n = prompt("Your choice", "y/n");
    switch(n)
    {
       case "y":
       case "Y":
       document.write("You Selected Yes");
       break;
       case "n":
       case "N":
       document.write("You Selected No.");
       break;
       default:
       document.write("Please enter y or n");
       break;
    }
   }
   f1();
</script>
```

8. If case string is a word then how to handle capitalization?
A. By converting the case value to upper or lowercase.

Ex:

```
<script>
   function f1(){
     var n = prompt("Your choice", "yes/no");
    switch(n.toLowerCase())
    {
       case "yes":
       document.write("You Selected Yes");
       break;
       case "no":
       document.write("You Selected No.");
       break;
       default:
       document.write("Please enter y or n");
       break;
    }
   }
   f1();
</script>
```

9. How to handle case for range of values?

A. By using boolean expression in "case".
    If case is using boolean expression, then switch value must be only "true".

EX:

```
<script>
   function f1(){
     var n = parseInt(prompt("Enter number"));
     switch(true)
     {
       case (n>=1 && n<=10):
       document.write(`your number ${n} is between 1 to 10`);
       break;
       case (n>10 && n<=20):
       document.write(`Your number ${n} is between 11 to 20`);
       break;
     }
   }
   f1();
</script>
```

10. Can we define case as regular expression?
A. You can do by using conditions
       if(),

11. Can we define If condtion in switch?
A. Yes.

12. Can we define a switch in switch?
A. Yes. But it is a bad coding technique to write a switch in switch, hence we recommend to write in a function and call in case.

Ex: Cascading Dropdown

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
      var categories = ["Select Category", "Electronics", "Footwear"];
      var electronics = ["Select Electronics", "Televisions", "Mobiles"];
      var footwear = ["Select Footwear", "Casuals", "Boots"];
      var products = [];
      function LoadCategories(){
```

```
        for(var category of categories)
        {
          var option = document.createElement("option");
          option.text = category;
          option.value = category;
          document.getElementById("lstCategories").appendChild(option);
        }
      }
      function LoadProducts(){
       document.getElementById("lstProducts").innerHTML = "";
       for(var product of products)
        {
          var option = document.createElement("option");
          option.text = product;
          option.value = product;
          document.getElementById("lstProducts").appendChild(option);
        }
      }
      function bodyload(){
       LoadCategories();
      }
      function CategoryChanged(){
       var categoryName = document.getElementById("lstCategories").value;
       switch(categoryName){
          case "Electronics":
            products = electronics;
            LoadProducts();
            break;
          case "Footwear":
            products = footwear;
            LoadProducts();
            break;
            default:
            products = ["Please Select Category"];
            LoadProducts();
            break;
       }
      }
  </script>
</head>
<body onload="bodyload()">
  <dl>
   <dt>Select Category</dt>
   <dd><select id="lstCategories" onchange="CategoryChanged()"></select></dd>
   <dt>Select Product</dt>
   <dd><select id="lstProducts"></select></dd>
  </dl>
</body>
</html>
```

if, else, switch, case, default

Day 32 : Loops – 11/04/23

Looping Control Statements

- Looping is the process of executing a set of statements repeatedly until the given condition is satisfied.
- Every loop contains
    a) Initialization
    b) Condition
    c) Counter
- Loops are created by using
    a) for
    b) while
    c) do while

The For Loop:
- Developer uses "for" loop when he is sure about the number of iterations and iteration count will not change dynamically.

Syntax:

```
for(initialization; condition; counter)
{
}
```

Ex:

```
<script>
   for(var i=10; i>=1; i=i-1){
      document.write(i + "<br>");
   }
</script>
```

```
<script>
   for(var i=1; i<=10; i++){
      document.write(i + "<br>");
   }
</script>
```

Ex:

```
<script>
     var names = [];
     for(var i=0;i<=9;i++){
        var name = prompt(`Enter Name[${i+1}]`);
        names[i] = name;
     }
     document.write(names);
</script>
```

- You can initialize and check for condition explicitly outside the for(), but the memory allocation must be defined.

Syntax:

```
        for( ;  ; i++)          // valid
        {
        }
```

Ex:

```
<script>
    var names = [];
    var i=0;
    for( ;i<=9;i++){
       var name = prompt(`Enter Name[${i+1}]`);
       names[i] = name;
    }
    document.write(names);
</script>
```

Ex:

```
<script>
   var menu = [
      {Category: "Electronics", Products: ["Televisions", "Mobiles"]},
      {Category: "Footwear", Products: ["Boots", "Casuals"]}
   ];
  function bodyload(){
     for(var i=0; i<menu.length; i++){
        var li = document.createElement("li");
        li.innerHTML = menu[i].Category;
        for(var j=0; j<menu[i].Products.length;j++){
           var ul = document.createElement("ul");
           var ulLi = document.createElement("li");
           ulLi.innerHTML = menu[i].Products[j];
           ul.appendChild(ulLi);
           li.appendChild(ul);
           document.querySelector("ol").appendChild(li);
        }

     }
  }
</script>
<body onload="bodyload()">
  <ol>

  </ol>
</body>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        var topics = ["Web Technologies", "Front-End", "HTML", "Semantics", "Layout", "Header"];
        function bodyload(){
            for(var i=0; i<topics.length; i++){
                var h = document.createElement(`h${i+1}`);
                h.innerHTML = topics[i];
                document.getElementById("container").appendChild(h);
            }
        }
    </script>
</head>
<body onload="bodyload()">
    <div id="container">

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

2. The While loop
- It is used by developer when he is not sure about the number of iterations and iteration counter may change dynamically.
- While can start only when the condition evaluates to true.

Syntax:

```
    var i=0;
    while(i<10)
    {
        i++;
        console.log(i);
     }
```

3. Do While
- It is similar to while loop but it ensures that the statements will execute at least once even when the condition is false.

Syntax:

```
    do
    {
    }while(condition);
```

Ex:

```
<script>
    var i = 10;
    do
    {
        i++;
        document.write(i + "<br>");
    }while(i<10);
</script>
```

Task:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        var userDetails = {
            UserName: "john",
            Pin: 4535
        }
        function SubmitClick(){
            var count = 0;
            var pin = parseInt(document.getElementById("pin").value);
            if(userDetails.Pin==pin){
                document.write("PIN Verified Successfully..");
            } else {
                count++;
                do {
                  document.write(`Invalid Pin - ${3-count} Attempts left`);
                }while(count<=3)
            }
        }
    </script>
</head>
<body>
    <fieldset>
        <legend>Verify PIN</legend>
        <div>
            Your PIN : <input type="text" id="pin"> <button onclick="SubmitClick()">Submit</button>
        </div>
        <p aling="center" id="error"></p>
    </fieldset>
</body>
</html>
```

Day 33 : Pin Verification – 12/04/23

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
```

```
        var userDetails = {
            UserName: "john",
            Pin: 4535
        }
        var count  = 0;
        function SubmitClick(){
            var pin = parseInt(document.getElementById("pin").value);
            if(pin==userDetails.Pin) {
                document.write("Success..");
            } else {

                do {
                    count++;
                    document.querySelector("p").innerHTML = `${(3-count)} left`;
                    if(count==3){
                        document.querySelector("p").innerHTML="Card Blocked";
                        document.querySelector("button").disabled= true;
                        break;
                    }
                } while(count>=3);
            }
        }
    </script>
</head>
<body>
    <fieldset>
        <legend>Verify PIN</legend>
        <div>
            Your PIN : <input type="text" id="pin"> <button onclick="SubmitClick()">Submit</button>
        </div>
        <p aling="center" id="error"></p>
    </fieldset>
</body>
</html>
```

Day 34 : JavaScript Statements – 13/04/23

FAQ's:

1. How to read and print values from 2D array?

```
var  values = [ [10, 20], [30, 40], [50,60] ];
```

Ex:

```
<script>
    var values = [[10, 20, 90], [30,40], [50, 60, 80, 100], [120]];
    for(var i=0; i<values.length; i++)
    {
        for(j=0; j<values[i].length; j++){
            document.write(values[i][j] + "  ");
        }
```

```
        document.write("<br>");
    }
</script>
```

Iteration Statements

- Iteration is the process of reading elements from a collection in seqential order.
- It doesn't require initialization, condition and counter.
- Iterations can be created by using

```
for..in          => Reading all properties from collection
for..of          => Reading all values from collection
```

Syntax:

```
for(var item of collection)
{
}
```

Jump Statements

a) break
b) return
c) continue

- Break will terminate the block and stays in function.
- Return will terminate the execution of script. It stops compiling.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function VerifyUserId(){
            var userid = document.getElementById("txtUserId").value;
            var userError = document.getElementById("UserError");
            fetch("../data/users.json")
            .then(function(res){
                return  res.json();
            })
            .then(function(users){
                for(var user of users){
                    if(user.UserId==userid){
                        userError.innerHTML = "User Id Taken - Try Another".fontcolor('red');
                        break;
                    } else {
                        userError.innerHTML = "User Id Available".fontcolor('green');
                    }
                }
            })
```

```
        }
    </script>
</head>
<body>
    <fieldset>
        <legend>Register User</legend>
        <dl>
            <dt>User Id</dt>
            <dd><input type="text" onkeyup="VerifyUserId()" id="txtUserId"></dd>
            <dd id="UserError"></dd>
        </dl>
    </fieldset>
</body>
</html>
```

- Return can be used only with in a function.

Ex:

```
<script>
   function PrintNumbers(){
    var stopNumber = parseInt(prompt("Loop Print 1 to 20 Set your Stop Number : "));

    for(var i=1; i<=20; i++){
        if(i==stopNumber){
            return;
        }
        document.write(i + "<br>");
    }
   }
   PrintNumbers();
</script>
```

- Break can be used only with a condition block.

Note: return is not a conditional jump.
break is a conditional jump.

- Continue is a jump statement, which skips the current context and continue to next.

Ex:

```
<script>
    for(var i = 1; i<=20; i++) {

      if(i==4 || i==14) {
         continue;
      }
      document.write(i + "<br>");

    }
```

```
</script>
```

Ex:

```
<script>
   var products = [
     {Name:"TV", Category:"Electronics"},
     {Name:"Nike Casuals", Category:"Footwear"},
     {Name:"Mobile", Category:"Electronics"},
     {Name:"Shirt", Category:"Fashion"},
     {Name:"Boots", Category:"Footwear"}
   ];
   for(var item of products){
      if(item.Category=="Fashion" || item.Category=="Electronics") {
        continue;
      }
      document.write(item.Name + "<br>");
   }
</script>
```

Exception Handling Statements

- In computer programming 2 types of errors are present.

  a) Compile Time Errors
  b) Run Time Errors

- Compile time errors are syntactical errors, due to which program fails to compile and start.

- Run time errors are the issues that occur at the time of using application.
  your application is unable to understand to instructions given.

- If any application is unable to process the instructions given by user at runtime, then it leads to "abnormal termination". [application closes]

- To avoid abnormal termination we have to implement Exception Handling.

- If application unable to understand instructions then exception will trigger, which leads to abnormal termination.

- Exception handling statements are

```
try          => monitioring block
catch        => handler block
throw        => throws exception
finally      => executes statement always
```

Ex:

```
<script>
  try {
   var a = parseInt(prompt("Enter A value"));
```

```
    var b = parseInt(prompt("Enter B value"));
    if(b==0){
      throw "Can't divide by Zero";
    }
    if(b>a) {
      throw "Number too large..";
    }
    var c = a / b;
    document.write(`Division = ${c}<br>`);
   }
   catch(ex) {
     document.write(ex);
   }
   finally {
     alert("Program End");
   }
</script>
```

Day 35 : Functions – 14/04/23

Functions in JavaScript

What is the purpose of function?

- Function is used to "refactor" the code.
- "Refactor" is the process of encapsulating the code into a block and extract to a file or function name.
- So that you can reuse the code.

Ex:
1. Add a new JS file
   "demo.js"

2. Write a set of code

```
for(var i =1; i<=10; i++)
{
  document.write(i + "<br>");
}
```

3. Select all lines => right click on selection => Refactor

4. Name the function
   "PrintNumbers()"

Function Configuration
- JavaScript function can be configured by using 2 techniques

a) function declaration
b) function expression

Declaration Syntax:

```
function  Name(params)
{
}
```

Expression Syntax:

```
var Name = function(params) {

}
```

- Declaration allocates memory for a function, which is not accessible to another function.
- Expression allocates memory which can be share to multiple functions.

Ex:

```
<script>
  var msg = prompt("Enter Message","hello | welcome");
  var fun;
  if(msg=="hello") {
    fun = function(){
     document.write("Hello ! JavaScript");
    }
  } else {
    fun = function(){
     alert("Welcome to Javascript");
    }
  }
  fun();
</script>
```

Function Structure
- Every javascript function comprises of 3 basic elements

a) Declaration
b) Signature
c) Definition

Syntax:

```
function  Print(msg)
{
}

function  Print(msg)       => declaration
Print(msg)                 => signature
{ }                        => definition
```

- A function signature is used to access the function from any location.

Ex:
```
<script>
   function PrintNumbers()
   {
     for(var i=1; i<=10; i++){
       document.write(i + "<br>");
     }
   }
   PrintNumbers();
   PrintNumbers();
</script>
```

Function Parameters:
- A function parameter is required to modify the function.
- A function can change its functionality according to state and situation.
- Parameter is used for changing the functionality.
- Every function have parameters configure as
    a) Formal Parameter
    b) Actual Parameter

Syntax:
```
    function  Print(msg)       => msg is formal parameter
    {                          It is just a memory allocated to store value
    }

    Print("welcome");       => actual parameter
```

    Formal Parameter is a reference and
    Actual Parameter is a value.

- The parameters defined in function declaration are known as
  "Formal Parameters".

- The parameters defined in function calling are known as
   "Actual Parameters".

```
        Formal_Parameter = Actual_Parameter;
```

- Actual parameter can be any type
   a) Primitive
   b) Non Primitive Type
   c) Function type

Ex:
```
<script>
   function PrintNumbers(howMany)
   {
     for(var i=1; i<=howMany; i++){
       document.write(i + "<br>");
```

```
    }
  }
  PrintNumbers(9);
</script>
```

Ex:

```
<script>
  function VerifyValue(value)
  {
     document.write(`Your value ${value} is : ${typeof value}<br>`);
  }
  VerifyValue("Hello");
  VerifyValue(30);
  VerifyValue(true);
  VerifyValue(["TV", "Mobile"]);
  VerifyValue({Name:"TV", Price:4500});
  VerifyValue(function(){document.write("Welcome");});
</script>
```

- Every Parameter is mandatory
- The parameters defined in function declaration are considered as "arguments"
- If a function is having parameter and you are not passive value for parameter, and you are using the parameter in function.

Ex:

```
<script>
    function Demo(collection){
      for(var item of collection){
        document.write(item + "<br>");
      }
    }
    Demo("Welcome");
</script>
```

- A function can have multiple parameters
- The maximum number of parameters allowed as per ECMA are 1024.

Syntax:

```
  function Name(param1, param2, ...)
  {
  }
```

- Every parameter is mandatory and have order dependency.

Ex:

```
<script>
  function Product(id, name, price){
    if(id==undefined){
      document.write(`
```

```
            Name  : ${name} <br>
            Price : ${price}
        `);
      } else {
        document.write(`
            Id    : ${id} <br>
            Name  : ${name} <br>
            Price : ${price}
        `);
      }
    }
    Product(1,"TV", 34000.33);

</script>
```

Day 36 : JS Functions with return statement – 15/04/23

- Function Purpose
- Function Configuration
  a) Declaration
  b) Expression
- Declaring, Signature, Definition
- Function Parameters
- Parameter Types
- Formal Parameter
- Actual Parameter
- Undefined Parameters

FAQ: What is the purpose of a function as parameter inside any function?

```
function Name(param) {
}

Name(function(){})
```

Ans: It is used to design callbacks.

FAQ: What is a callback function?

Ans: It is used to handle various functionalities according to state and situation. A function can change according to state and situation.

You define a set of functions, condition chooses which function to execute according to situation.

Note: When you define a function as parameter then it is not configured with "name".

```
Name(function(){})
```

Ex:

```
<script>
   function VerifyPassword(password,success,failure){
     if(password=="admin"){
       success();
     } else {
       failure();
     }
   }
   VerifyPassword("admin",function(){
      document.write("Login Success");
   }, function(){
      document.write("Invalid Password");
   })
</script>
```

Note: Allowing parameter into a function leads to "Injection Attacks".

XSS - Cross Site Scripting Attacks
Injection Attacks
CORS - Cross Origin Resource Sharing
Cross Page Posting
Double Posting

Rest Parameters

- JavaScript ES6 introduced Rest parameters.
- One parameter can allow multiple arguments.
- It is defined by using "...paramName".

Syntax:

```
function  Name(...paramName)
{
}
```

- Rest parameter is an array type[].
- You can access the values by using Index [property] reference of by using de-structuring.

Ex:

```
<script>
   function Product(...details)
   {
      var [id, name, price, stock] = details;
      document.write(`
         Id    : ${id} <br>
         Name  : ${name} <br>
         Price : ${price} <br>
         Stock : ${stock}
      `);
```

```
    }
    Product(1, "TV", 34000.33, true);
</script>
```

- Every function can have only one rest parameter.

```
function Name(...param1, ...param2)        => invalid
```

- You can have rest parameter with other parameters.
- Rest parameter must be the last parameter in formal list.
  [as it reads upto end]

Ex:

```
<script>
    function Product(title,...details)
    {
        var [id, name, price, stock] = details;
        document.write(`
            <h2>${title}</h2>
            Id    : ${id} <br>
            Name  : ${name} <br>
            Price : ${price} <br>
            Stock : ${stock}
        `);
    }
    Product("Product Details",1, "TV", 34000.33, true);
</script>
```

Spread Syntax

- It is the process of configuring one actual parameter with multiple values, which can spread into multiple formal parameters.

Syntax:

```
function  Name(param1, param2, param3)
{
}

Name(...[val1, val2, val3]);
```

Ex:

```
<script>
    function PrintDetails(id, name, price, stock){
        document.write(`
            Id    : ${id} <br>
            Name  : ${name} <br>
            Price : ${price} <br>
            Stock : ${stock}
        `);
    }
```

```
    var values = [1,"Samsung TV",45000.33,true];
    PrintDetails(...values);
</script>
```

Ex:

```
<script>
    function PrintCollection(a, b, c , d)
    {
      document.write(`
        A =  ${a} <br>
        B = ${b} <br>
        C = ${c} <br>
        D = ${d}
      `);
    }
    var data = [10, 20, 30];

    PrintCollection(...data, 40);
</script>
```

FAQ: What is difference between spread and rest?
Ans : Rest is about formal parameters
Spread is about actual parameters

Function with Return

- Every JavaScript function is Void type by default.
- Void refers to : discard the return.
- Every JavaScript function
    1. allocates memory
    2. performs operation in memory
    3. destroys memory
    4. It will not return any value as there no memory.

```
function  Name()
{

}       // end of function => memory is erased
```

- A function can use "return" jump statement.

```
function  Name()
{
   return  value;
}

Name = value;
```

- You can keep the memory of a function available even after the function ends.
- If there is no "return" it is void type, which removes the memory allocated for function.

- It is a mechanism of creating function which can use its own memory instead of accessing any global memory.
- These type of functions are known as "Pure Functions"

FAQ: What is Impure Function?
Ans:
- Every functional programming language used impure functions by default.
- They can access the memory outside function and modify the memory.
- It slow in access.
- It is heavy on application as multiple memory allocations are required.

Ex: Impure
```
<script>
   var c = 0;
   function Addition(a, b){
      c = a + b;
   }

   function Result(){
      document.write(`Addition = ${c}`);
   }
   Addition(10, 20);
   Result();
</script>
```

FAQ: What is pure function?
Ans:
- It allocates memory for functionality and result.
- It uses its own memory and makes it global in access.
- Without declaring a global variable you can access and use the value from any another function.

Ex: Pure
```
<script>
   function Addition(a, b){
      return a + b;
   }

   function Result(){
      document.write(`Addition = ${Addition(10,30)}`);
   }
   Result();
</script>
```

FAQ: Can a function have more that one return defined?
Ans: Yes. It is used to handle conditional return. [Conditional Rendering]

Ex:
```
<script>
```

```
  function Template(template){
     if(template=="login"){
        return `
          <h2>User Login </h2>
          <dl>
            <dt>User Id</dt>
            <dd><input type="text"></dd>
            <dt>Password</dt>
            <dd><input type="password"></dd>
          </dl>
          <button> Login </button>
        `;
     } else {
       return `
          <h2>Register User </h2>
          <dl>
            <dt>User Id</dt>
            <dd><input type="text"></dd>
            <dt>Password</dt>
            <dd><input type="password"></dd>
            <dt>Email</dt>
            <dd><input type="email"></dd>
            <dt>Age</dt>
            <dd><input type="number"></dd>
          </dl>
          <button> Login </button>
        `;
     }
  }
  function TemplateChange(){
      var template = document.querySelector("select").value;
      if(template=="login") {
         document.querySelector("p").innerHTML = Template(template);
      } else {
         document.querySelector("p").innerHTML = Template(template);
      }
  }
</script>
<body>
   <select onchange="TemplateChange()">
      <option>Select Template</option>
      <option value="login">Login</option>
      <option value="register">Register</option>
   </select>
   <p></p>
</body>
```

Day 37 : JS Functions continued… – 17/04/23

Function Parameter

spread : one actual parameter can spread values to multiple formal parameter.

rest : one formal parameter can accept multiple values as actual parameter.

Function Return

pure with return

impure without return

Anonymous Functions

- It is a function without name.

Syntax:

```
function() {
   ...some functionality...
}
```

- The anonymous functions are accessed by using a pattern called as "IIFE" [Immediately Invoked Function Expression]

Syntax:

```
(function(){

})();
```

Ex:

```
<script>
  (function(){
    document.write("Anonymous Function");
  })();
</script>
```

- Anonymous functions are mostly used in callbacks.

Function Recursion

- It is the process of accessing a function with in the context of same function.

Syntax:

```
function f1()
{
  f1();
}
```

- Recursion is the process a executing any specified task repeatedly without looping and iterations.

- It is used for creating batch operations.

FAQ: Write a function without using loops and iterations to print numbers from 1 to 10.

```
function PrintNumber(n) {

}
```

Ex:

```
<script>
   function PrintNumber(n){
      document.write(n + "<br>");
      n++;
      if(n>10){
         return;
      }
      PrintNumber(n);
   }
   PrintNumber(1);
</script>
```

Write a program to print factorial of given number?

```
function Factorial(n)
{

}
Factorial(5)
 5 * 4 * 3 * 2 * 1 = 120
```

Ex:

```
<script>
   function Fact(n){
      if(n<1) {
         return 1;
      }
      return n * Fact(n-1);
   }
   document.write(Fact(6));
</script>
```

Arrow Functions

- It is a short hand technique of writing a function.
- But arrow requires function expression.
- It is good for callbacks.

```
 ( )          function parameters
 =>       return / single statement to execute
```

{ } multiple statements to execute

Ex:
```
function hello() {
    document.write("Hello ! JS");
}

var hello  = () => document.write("Hello ! JS");
```

Ex:
```
function hello(uname)
{
  document.write(`Hello ! ${uname}`);
}

var hello  = uname => document.write(`Hello ! ${uname}`);
```

Ex:
```
function  Addition(a, b)
{
   return a + b;
}

var addition = (a,b) => a + b;
```

Ex:
```
<script>
   var hello = uname => `Hello ! ${uname}`;
   document.write(hello("John"));
</script>
```

Ex:
```
<script>
   fetch(`http://fakestoreapi.com/products`)
   .then(response=> response.json())
   .then(products=> {
      products.map(product=> document.write(`<p>${product.title}</p>`))
   })
</script>
```

Day 38 : Function Closure and Generator – 18/04/23

Anonymous Functions [IIFE]
Arrow Functions

Function Closure

- Closure is a technique where inner function can access the value of outer function.

- It is not just about inner function, any outer variable accessed inside a function is known as "Closure".

Syntax:

```
<script>
   var x  =10;
   function  Print(){
     console.log(x);
   }
</script>
```

Ex:

```
<script>
   function Outer()
   {
      var x = 10;
      function Inner(){
         document.write(`x=${x}`);
      }
      Inner();
   }
   Outer();
</script>
```

FAQ: Is the value of inner function accessable to outer?
Ans: No. You can implement by using function return.

Ex:

```
<script>
   function Outer()
   {
      var x = 10;
      function Inner(){
         var y = 20;
         return `x=${x} y=20`;
      }
     var result = Inner();
     document.write(result);
   }
   Outer();
</script>
```

Ex:

```
<script>
  function Outer(outerValue){
     return function Inner(innerValue){
       return `Outer: ${outerValue} Inner : ${innerValue}`;
     }
  }
  var result = Outer('Outer Value');
  document.write(result('Inner Value'));
```

```
</script>
```

Syntax:

```
   Outer("outerValue")("innerValue");
```

Ex:

```
<script>
  function Outer(outerValue){
     return function Inner(innerValue){
       return `Outer: ${outerValue} <br> Inner : ${innerValue}`;
     }
  }
  document.write(Outer("Outer function Value")("Inner function Value"))
</script>
```

Function Generator

- Function generator can return mutliple values.
- It is used to configure an Iterator.
- Iterator is a software design pattern used to access elements from a collection in sequential order.
- Generator comprises of
  - a) value : any type
  - b) done : boolean
  - c) next()
  - d) return()

```
   for(var item of collection)
   {
   }
```

```
   for()       => Generator function
   map()       => Generator Function
   find()      => Generator Function
   filter()    => Generator Function
```

- Generator function is configured with "function*"

Syntax:

```
       function*  Name()
       {
       }
```

- Generator function will return a value by using "yeild" operator

Ex:

```
<script>
   function* Demo(){
      yield 1;
```

```
        yield 2;
        yield 3;
    }
    var obj = Demo();
    document.write(`Value:${obj.next().value}<br>`);
    document.write(`Value:${obj.next().value}<br>`);
    document.write(`Value:${obj.next().value}<br>`);

</script>
```

Ex:

```
<script>
    function* Demo(){
        yield function(a,b){return a+b};
        yield function(a,b){return a-b};
        yield function(a,b){return a*b};
    }
    var obj = Demo();
    console.log(obj.next().value(10,20));
    console.log(obj.next().value(20,10));
    console.log(obj.next().value(2,5));
</script>
```

Ex:

```
<script>
    function* Demo(){
        yield function(a,b){return a+b};
        yield function(a,b){return a-b};
        yield function(a,b){return a*b};
    }
    var obj = Demo();
    console.log(obj.next().value(10,20));
    console.log(obj.next().value(20,10));
    obj.return();
    console.log(obj.next().value(2,5));        // error-not defined
</script>
```

Ex:

```
<script>
    function* Demo(){
        let x = 1;
        while(true){
            let increment = yield x;
            if(increment!=null) {
                x+=increment;
            } else {
                x++;
            }
        }
    }
```

```
    var obj = Demo();
    console.log(obj.next());
    console.log(obj.next());
    console.log(obj.next(10));
    console.log(obj.next());
    console.log(obj.next());
</script>
```

Day 39 : JavaScript OOPs – 19/04/23

Function Expression
Function Declaration
Function Parameters
  a) Rest
  b) Spread
Function Return
Arrow Functions
Function Recursion
Function Generator
Function Colsure

JavaScript OOP

- JavaScript is not an OOP language.
- It supports only few features of OOP.
- Various programming systems are used in software development

  a) POPS
  b) OBPS
  c) OOPS

POPS
- Process Oriented Programming System
- It supports low level features.
- It can directly interact with hardware services.
- It uses less memory.
- It is faster.

EX: C, Pascal
- Code resuability issues
- Code separation issues
- Code extensibility issues
- Dynamic memory allocation issues.

OBPS:
- Object Based Programming System
- Provides reusability
- Supports separation and extensibility
- Supports dynamic memory allocations

Ex: JavaScript, VB

- No code level security
- No dynamic polymorphism
- No templates [Abstract members]
- No contracts [Interfaces]

OOPS
- Object Oriented Programming System
- Code reusability
- Code separation
- Code extensibility
- Easy testablility
- Code level security
- Supports dynamic polymorphism
- Provides templates and contracts

Ex: C++, Java, C#
- Will not support low level features
- Can't directly interact with hardware services
- Uses more memory
- It is slow.

Evolution of OOP
- 1967 SIMULA - first OOP - Johan Olay , Nygaard - Code Resusability
- 1970's Small Talk - Trygve - Code Separation [MVC]

Java	MVC - Spring
PHP	MVC - Code Igniter, Cake PHP
Python	MVC - Django
.NET	MVC - ASP.NET MVC

- 1975's C++
- 1990 Java
- 2003 C#

Modules in OOP
- JavaScript module is a set of functions and classes.
- Modules are used to build a library for application.
- So that you can import and reuse the library.
- Module system will not work of every device.
- You have to install a module system or use any browser, which is using a module system.
- There are different types of module systems
  a) Common JS
  b) UMD [Universal Module Distribution]
  c) AMD [Asyncrhonous Module Distribution]
  d) ECMA Module System

Note: Every JavaScript file is a Module.
Every module comprises of
a) variables
b) functions
c) classes

Ex:
1. Add a new folder into project by name "library"
2. Add subfolder "modules"
3. Add a new file into modules by name

home.module.js

```
function PrintName() {

 }
```

4. Every member of module is in module scope and not accessible outside.
5. If you wan any module member accessbile outside the scope then you have to mark it as "export".

```
export function PrintName() {  }
```

Ex:
home.module.js

```
var userName = "John";

export function PrintName(){
   return `Hello ! ${userName}`;
}

export function Addition(a, b){
   return a +  b;
}
```

products.module.js

```
export function GetProducts(){
   return[
      {Name: "TV", Price: 64000.33},
      {Name: "Mobile", Price: 21000.23}
   ];
}
```

6. Create a new HTML page and import function

```
import { functionName } from "../path..";
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="module">
        import {PrintName, Addition} from "../library/modules/home.module.js";
        import {GetProducts} from "../library/modules/products.module.js";

        document.querySelector("p").innerHTML = PrintName() + "<br>" + "Addition=" + Addition(20,
32);
        GetProducts().map(function(product){
            var li = document.createElement("li");
            li.innerHTML = product.Name;
            document.querySelector("ol").appendChild(li);
        })
    </script>
</head>
<body>
    <p></p>
    <ol>

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

- Every module can have one default export.
- "Default" function is loaded into memory automatically after importing.
- Non-Default function is loaded into memory only when requested.

```
    export default function  Name() {  }

    import  Name  from  "path";
    import { Name}  from "path"; //invalid default function will not use { }
```

- Every module can have only one default member.
- Default member must be the first member to import.

Ex:
home.module.js

```
var userName = "John";

export function PrintName(){
    return `Hello ! ${userName}`;
}

export default function Addition(a, b){
    return a +  b;
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="module">
        import Addition,{PrintName} from "../library/modules/home.module.js";
        import {GetProducts} from "../library/modules/products.module.js";

        document.querySelector("p").innerHTML = PrintName() + "<br>" + "Addition=" + Addition(20,
32);
        GetProducts().map(function(product){
            var li = document.createElement("li");
            li.innerHTML = product.Name;
            document.querySelector("ol").appendChild(li);
        })
    </script>
</head>
<body>
    <p></p>
    <ol>

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

Day 40 : JavaScript Class – 20/04/23

1. Modules
    a) Configuring functions in modules
    b) Export
    c) Export Default
    d) Import

FAQ: How to import all functions and classes of any module?
Ans: by using "*" meta character. But you can't import default members.

Syntax:

```
import  *  as refObj  from "../moduleName.js"

refObj.functionName()
refObj.className
```

Ex:
home.module.js

```
export var userName = "John";
```

```
export function PrintName(){
   return `Hello ! ${userName}`;
}

export default function Addition(a, b){
   return a +  b;
}
```

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script type="module">

      import Addition, * as home from "../library/modules/home.module.js";

      document.querySelector("p").innerHTML = home.PrintName() + "<br>" + Addition(20,50);

   </script>
</head>
<body>
   <p></p>

</body>
</html>
```

Class in OOP

- Class is a program template with data and logic, which you can customize and implement according to the requirements.
- Class is known as a Model or Entity.
- Class refers to a logical entity where all the data and logic is defined.
- You can access the logic and data by using "Instance" of class.
- Class is a template that implements business or data.
- If a class is mapping to business requirements then it is called as "Entity".
- If a class is mapping to data requirements then it is called as "Model".
- It have the behaviour of blue-print, which is a prototype.
- Blue-Print is a source from where various instances are created.

Summary

- Class is Template
- Maps to Business => Entity
- Maps to Data => Model
- Prototype => Blue Print

Configuring class:

- You can configure a class in JavaScript by using 2 techniques

  a) Class Declaration
  b) Class Expression

Syntax: Declaration

```
class  className
{
   // members
}
```

Syntax: Expression

```
const  Name = class  {
   // members
 }
```

Ex:

```
<script>
   var prototype = "Employee";
   var memory = class {
      // employee details
   }
   if(prototype=="Employee"){
      memory = class {
         //employee details
      }
   } else {
      memory = class {
         //student details
      }
   }
</script>
```

Class Members:
- A JavaScript class can have only
  a) Property
  b) Accessor
  c) Method
  d) Constructor
  as class members.

FAQ: Can we define a variable as class member?
Ans: No.

FAQ: Can we define a variable in class?
Ans. Yes. But not as class member, it can be a member of any method in class.

FAQ: Why a variable is not allowed as class member?

Ans: Variables are immutable types. And class member can't be immutable as class is a template.

FAQ: Can we define a function as class member?
Ans: No. It is immutable

FAQ: Can we define a function in class?
Ans: Yes.

Property

- In a class data is stored in Property.
- Property is a memory reference name.
- JavaScript property have just a reference name initialized with value.

Syntax:

```
class  Product
{
   Price= 4500.44;
}
```

- A property of class can store any type of data,
  a) Primitive Type
  b) Non Primitive Type

Syntax:

```
<script>
  class Product
  {
    ProductId = 1;
    Name = "Samsung TV";
    Stock = true;
    Cities = ["Delhi", "Hyd"];
    Rating = {Rate:4.3, Count:3520}
  }
</script>
```

- You can access the properties of a class by using instance of class.

```
var  refName = new ClassName;
```

Ex:

```
<script>
  class Product
  {
    ProductId = 1;
    Name = "Samsung TV";
    Stock = true;
    Cities = ["Delhi", "Hyd"];
    Rating = {Rate:4.3, Count:3520}
  }
```

```
  var tv = new Product;
  tv.Name = prompt("Enter Name");
  document.write(`
      Id      :  ${tv.ProductId} <br>
      Name    : ${tv.Name} <br>
      Stock   : ${tv.Stock} <br>
      Cities  : ${tv.Cities.toString()} <br>
      Rating  : Rate=${tv.Rating.Rate} Count=${tv.Rating.Count}
  `);
</script>
```

- You can have fine control over property and change according to state and situation by using "accessors"

Accessors

- Accessor gives a fine grained control over the property.
- It is used to handle read and write operations on property.
- Accessors are 2 types
    a) Getter => get()
    b) Setter => set()

- Get is used to read value from a property
- Set is used to write data into a property.

- You can control the behaviour of a property by using accessors.

Day 41 : Properties and Accessors – 21/04/23

Class
Class Members
Properties

Accessors

- Accessors provide a fine grained control over property.
- You can control read and write operations of property using accessor.
- Accessors are 2 types
    a) Getter
    b) Setter
- Getter is used to read value from a property and return the value.

```
PropertyName;

get aliasName() {
   return  this.PropertyName;
}
```

- Setter is used to set value into a property.

```
PropertyName;
```

```
        set  aliasName(newValue) {
          this.PropertyName = newValue;
        }
```

Ex:

1. Add a new file into modules

product.module.js

```
export class Product
{
   UserName;
   Role;
   Error;

   _productName;

   get ProductName(){
      return this._productName;
   }

   set ProductName(newName) {
      if(this.Role=="admin") {
         this._productName = newName;
      } else {
         this.Error = `Hello ! ${this.UserName} your role ${this.Role} is not authorized to set product
name`;
      }

   }
}
```

2. Page home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script type="module">
     import { Product } from "../library/modules/product.module.js";
     var obj = new Product();
     obj.UserName = prompt("Enter User Name");
     obj.Role = prompt("Enter your Role");
     obj.ProductName = prompt("Enter Product Name");
     if(obj.ProductName){
       document.querySelector("p").innerHTML = `Product Name : ${obj.ProductName}`;
     } else {
```

```
      document.querySelector("p").innerHTML = obj.Error;
    }
  </script>
</head>
<body>
   <p></p>
</body>
</html>
```

Note: You can't configure event handler in HTML page and access in module. You have to define events for elements in HTML page by using
"addEventListener()"

Syntax:

```
   document.querySelect("button").addEventListener("eventName", functionName);
   document.querySelect("button").addEventListener("click", DetailsClick);
```

Ex:

product.module.js

```
export class Product
{
   Name;
   Price;
   Stock;
}
```

Home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script type="module">

      import {Product} from "../library/modules/product.module.js";

      function DetailsClick(){
        var obj = new Product();
        obj.Name = prompt("Enter Name");
        obj.Price = prompt("Enter Price");
        obj.Stock = prompt("Enter Stock");
        document.querySelector("p").innerHTML  = `
          Name =  ${obj.Name} <br>
          Price = ${obj.Price} <br>
          Stock = ${obj.Stock}
        `
      }
```

```
      document.getElementById("btnDetails").addEventListener("click", DetailsClick);
  </script>
</head>
<body>
   <button id="btnDetails">Get Details</button>
   <p></p>
</body>
</html>
```

Ex: home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script type="module">
      import {Product} from "../library/modules/product.module.js";

      function DetailsClick(){
        var obj = new Product();
        obj.Name = document.getElementById("Name").value;
        obj.Price = document.getElementById("Price").value;
        obj.Stock = (document.getElementById("Stock").checked==true)?"Available":"Out of Stock";

        document.querySelector("p").innerHTML  = `
          Name =  ${obj.Name} <br>
          Price = ${obj.Price} <br>
          Stock = ${obj.Stock}
        `
      }
      document.getElementById("btnDetails").addEventListener("click", DetailsClick);
  </script>
</head>
<body>
   <dl>
     <dt>Product Name</dt>
     <dd><input type="text" id="Name"></dd>
     <dt>Price</dt>
     <dd><input type="text" id="Price"></dd>
     <dt>Stock</dt>
     <dd><input type="checkbox" id="Stock">Yes</dd>
   </dl>
   <button id="btnDetails">Get Details</button>
   <p></p>
</body>
</html>
```

- You can also create an accessor to access any element from a hierarchy of elements.

```
        get  aliasName()
        {
          return this.Parent.Child.innerChild ;
        }
```

Ex:
product.module.js

```
export class Product
{
   Name = "";
   Price = 0;
   Stock = false;
   Rating = {
      "CustomerRating": {"Rate":4.2, "Count":3400},
      "VendorRating": {"Rate":3.4, "Count": 23}
   }
   get CustomerRating(){
      return this.Rating.CustomerRating.Rate;
   }
}
```

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Document</title>
 <script type="module">
     import {Product} from "../library/modules/product.module.js";

     function DetailsClick(){
       var obj = new Product();
       obj.Name = document.getElementById("Name").value;
       obj.Price = document.getElementById("Price").value;
       obj.Stock = (document.getElementById("Stock").checked==true)?"Available":"Out of Stock";

       document.querySelector("p").innerHTML  = `
         Name =  ${obj.Name} <br>
         Price = ${obj.Price} <br>
         Stock = ${obj.Stock} <br>
         Customer Rating = ${obj.CustomerRating}
       `
     }
     document.getElementById("btnDetails").addEventListener("click", DetailsClick);
 </script>
```

```
</head>
<body>
   <dl>
     <dt>Product Name</dt>
     <dd><input type="text" id="Name"></dd>
     <dt>Price</dt>
     <dd><input type="text" id="Price"></dd>
     <dt>Stock</dt>
     <dd><input type="checkbox" id="Stock">Yes</dd>
   </dl>
   <button id="btnDetails">Get Details</button>
   <p></p>
</body>
</html>
```

functions, methods, procedures

function always return value
methods are void type
procedure may or maynot return value
select * from tblName

Day 42 : Constructor and Extensibility – 24/04/23

Methods
FAQ: What is difference between function, method and procedure?
Ans:
- Function is intended to return a value, It is used for building expression
- Method is void type, used to refactor the code.
- Procedure can return value or can be void, It can change according to state and situation.
- JavaScript class can't have function as class member.
- Logic in class is defined using method.

Syntax:

```
class ClassName
{
   method() {

   }
}
```

- The methods are accessed within class by using "this" and outside class by using instance of class.

- All features of methods are same like functions in JS.
    1. parameter less
    2. parameterized
    3. rest params
    4. spread operator etc..

- Methods are mutable and functions are immutable.

Ex:

```
export class Product
{
   Details = "";
   Name = "";
   Price = 0;
   Qty = 0;
   Total(){
      return this.Qty * this.Price;
   }
   Print(){
      return this.Details =
`Name=${this.Name}<br>Price=${this.Price}<br>Qty=${this.Qty}<br>Total=${this.Total()}`;
   }
}
```

EX

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script type="module">
     import { Product } from "../library/modules/product.module.js";
     var obj = new Product();
     obj.Name = prompt("Enter name");
     obj.Price = parseFloat(prompt("Enter Price"));
     obj.Qty = parseInt(prompt("Enter Quantity"));
     document.querySelector("p").innerHTML = obj.Print();
  </script>
</head>
<body>
  <p></p>
</body>
</html>
```

Ex:

```
<script>
   class Product
   {
      Print(...args) {
        for(var value of args){
          document.write(value + "<br>");
        }
      }
   }
   let obj = new Product();
```

```
    obj.Print(1, "TV", 45000.44, true);
</script>
```

Constructor

- Constructor is a special type of sub-route used for Instantiaton.
- Constructor is a design pattern used to create an object for class.
- Every class have a default constructor.
- It is used to for creating an object for class.
- JavaScript constructor is anonymous.

Syntax:

```
class  Product
{
  constructor() {
  }
}
```

- If you want any action to be performed at the time of creating an object for class then you can define by using constructor.

- Constructor is a special method, which executes automantically for every object.

Ex:

```
<script>
  class Database{
    constructor(){
      document.write(`Constructor Executed`);
    }
  }
  let obj = new Database();
</script>
```

- Constructor can be parameterized, where parameters are passed at the time of creating object for class.

Ex:

```
<script>
  class Database{
    constructor(name){
      document.write(`Hello ! ${name}`);
    }
  }
  let obj = new Database("john");
</script>
```

- JavaScript constructor can't overload.
- JavaScript constructor can't be static, public or private in access.

Ex:

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

```
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
     class Database
     {
        constructor(dbName){
          document.write(`Connected with ${dbName} database..<br>`);
        }
        Insert(){
          document.write(`Record Inserted`);
        }
        Delete(){
          document.write(`Record Deleted`);
        }
     }
     function InsertClick(){
       var obj = new Database(document.getElementById("lstDatabase").value);
       obj.Insert();
     }
     function DeleteClick(){
       var obj = new Database(document.getElementById("lstDatabase").value);
       obj.Delete();
     }
  </script>
</head>
<body>
   <select id="lstDatabase">
     <option>Choose Database</option>
     <option>Oracle</option>
     <option>MySql</option>
     <option>MongoDB</option>
   </select>
   <button onclick="InsertClick()">Insert</button>
   <button onclick="DeleteClick()">Delete</button>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
     class Database
```

```
    {
       ConnectionStatus = true;
       constructor(dbName){
         if(this.ConnectionStatus==true){
         document.write(`Connected with ${dbName} database..<br>`);
         } else {
           document.write(`Invalid - Operation Please Connect to database`);
         }
       }
       Insert(){
         document.write(`Record Inserted`);
       }
       Delete(){
         document.write(`Record Deleted`);
       }
    }
    function InsertClick(){
      var obj = new Database(document.getElementById("lstDatabase").value);
      if(this.ConnectionStatus){
       obj.Insert();
      }
    }
    function DeleteClick(){
      var obj = new Database(document.getElementById("lstDatabase").value);
      if(this.ConnectionStatus){
       obj.Delete();
      }
    }
  </script>
</head>
<body>
   <select id="lstDatabase">
     <option>Choose Database</option>
     <option>Oracle</option>
     <option>MySql</option>
     <option>MongoDB</option>
   </select>
   <button onclick="InsertClick()">Insert</button>
   <button onclick="DeleteClick()">Delete</button>
</body>
</html>
```

Summary Class Members
- Property
- Method
- Accessor
- Constructor

Code Extensibility and Reusability

- Code extensibility and reusability can be handled by using 2 techniques
  a) Aggregation

b) Inheritance

- You can extend the class with new features by using aggregation, which is object-to-object communication.
- Aggregation is the process of creating an object for existing class in newly created class. Without configuring relation between classes you can access the members of one class in another class.

Note: Code Extensibility and Resuability is required to achive
"Backward Compatiblity"

Ex:

```
<script>
  class HDFC_Bank_Version1
  {
      Personal = "Personal Banking Services";
      NRI = "NRI Banking Services";
      Print(){
        document.write(`${this.Personal}<br>${this.NRI}<br>`);
      }
  }
  class HDFC_Bank_Version2
  {
     Loans = "Car and Personal Loans";
     Print(){
        let obj = new HDFC_Bank_Version1();
        obj.Print();
        document.write(`${this.Loans}`);
     }
  }
  function InstallClick(){
     var ver = document.querySelector("select").value;
     switch(ver){
        case "ver1":
        document.write("<h2>HDFC Version-1</h2>");
        let obj1 = new HDFC_Bank_Version1();
        obj1.Print();
        break;
        case "ver2":
        document.write("<h2>HDFC Version-2</h2>");
        let obj2 = new HDFC_Bank_Version2();
        obj2.Print();
        break;
        default:
        document.write("Please Select a Version");
        break;
     }
  }
</script>
<body>
  <h2>Install Bank App </h2>
  <select>
```

```
        <option>Choose Version</option>
        <option value="ver1">Version-1</option>
        <option value="ver2">Version-2</option>
    </select>
    <button onclick="InstallClick()">Install</button>
</body>
```

Day 43 : Polymorphism – 25/04/23

Code Extensibility and Reusability

1. Aggregation [Object-to-Object Communication]
   Has-A-Relation

Inheritance

- You can configure relation between classes.
- JavaScript uses "extends" keyword, to extend the existing class and add new features.

Syntax:

```
class   A
{

}
class   B extends A
{

}
```

- Existing class is known as "Super" class. [class A]
- Newly created class is known as "Derived" class. [class B]
- The members of super class are accessible to derived class by using "super" keyword.

Syntax:

```
super.member => property | method
super()      =>  constructor
```

Ex: Inheritance

```
<script>
    class HDFC_Bank_Version1
    {
        Personal = "Personal Banking Services";
        NRI = "NRI Banking Services";
        Print(){
          document.write(`${this.Personal}<br>${this.NRI}<br>`);
        }
    }
    class HDFC_Bank_Version2 extends HDFC_Bank_Version1
    {
       Loans = "Car and Personal Loans";
```

```
        Print(){
            super.Print();
            document.write(`${this.Loans}`);
        }
    }
    function InstallClick(){
        var ver = document.querySelector("select").value;
        switch(ver){
            case "ver1":
            document.write("<h2>HDFC Version-1</h2>");
            let obj1 = new HDFC_Bank_Version1();
            obj1.Print();
            break;
            case "ver2":
            document.write("<h2>HDFC Version-2</h2>");
            let obj2 = new HDFC_Bank_Version2();
            obj2.Print();
            break;
            default:
            document.write("Please Select a Version");
            break;
        }
    }
</script>
<body>
    <h2>Install Bank App </h2>
    <select>
        <option>Choose Version</option>
        <option value="ver1">Version-1</option>
        <option value="ver2">Version-2</option>
    </select>
    <button onclick="InstallClick()">Install</button>
</body>
```

Note: OOP inheritance Rule => If you configure relation between classes then you can access the members of base class using derived class object. The rule is, you have to invoke the base class constructor first, then followed by derived class constructor.

Syntax:

```
        class  Super
        {
            constructor() { }
        }
        class Derived extends Super
        {
          constructor() { }              ==> invalid
        }

        let obj = new Derived();
```

- In JavaScript it is mandatory to call a super constructor in derived class constructor.

Ex:

```
<script>
   class SuperClass
   {
      constructor(){
         document.write("Super Class Constructor<br>");
      }
   }
   class DerivedClass extends SuperClass
   {
      constructor(){
         super();
         document.write("Derived Class Constructor");
      }
   }
   let obj = new DerivedClass();
</script>
```

- Class Inheritance is classified into various types

   a) Single Inheritance
   b) Multi Level Inheritance

Single Inheritance:
- A super class is extended by using Derived class.
- One super class and one derived class.

Multi Level Inheritance:
- A derived class is again extended by another class.
- Pervious class members are accessed by using "super".

Ex:

```
<script>
   class HDFC_Bank_Version1
   {
       Personal = "Personal Banking Services";
       NRI = "NRI Banking Services";
       Print(){
          document.write(`${this.Personal}<br>${this.NRI}<br>`);
       }
   }
   class HDFC_Bank_Version2 extends HDFC_Bank_Version1
   {
      Loans = "Car and Personal Loans<br>";
      Print(){
         super.Print();
         document.write(`${this.Loans}`);
      }
   }
```

```
    class HDFC_Bank_Version3 extends HDFC_Bank_Version2
    {
        AGRI = "Govt. Schemes";
        Print(){
            super.Print();
            document.write(`${this.AGRI}`);
        }
    }
    function InstallClick(){
        var ver = document.querySelector("select").value;
        switch(ver){
            case "ver1":
            document.write("<h2>HDFC Version-1</h2>");
            let obj1 = new HDFC_Bank_Version1();
            obj1.Print();
            break;
            case "ver2":
            document.write("<h2>HDFC Version-2</h2>");
            let obj2 = new HDFC_Bank_Version2();
            obj2.Print();
            break;
            case "ver3":
            document.write("<h2>HDFC Version-3</h2>");
            let obj3 = new HDFC_Bank_Version3();
            obj3.Print();
            break;
            default:
            document.write("Please Select a Version");
            break;
        }
    }
</script>
<body>
    <h2>Install Bank App </h2>
    <select>
        <option>Choose Version</option>
        <option value="ver1">Version-1</option>
        <option value="ver2">Version-2</option>
        <option value="ver3">Version-3</option>
    </select>
    <button onclick="InstallClick()">Install</button>
</body>
```

Note: OOP language will not support multiple inheritance for classes.
Reason : "Constructor Deadlock".

Syntax:

```
class Derived extends  Super1, Super2, Super3..        =>invalid
```

```
        {

        }
```

Noe:
If same name methods are defined in classes that use inheritnace, then the derived class members will hide the super class members.

You can access the hidden members of super class by using "super" keyword.

Polymorphism

- Poly means "Many"
- Morphos means "Forms"
- The ability of a component to work for various situations is polymorphism.

Ex:

```
<script>
   class Employee
   {
      FirstName;
      LastName;
      Designation;
      Print(){
         document.write(`${this.FirstName} ${this.LastName} - ${this.Designation}<br>`);
      }
   }
   class Developer extends Employee
   {
       FirstName = "Raj";
       LastName = "Kumar";
       Designation = "Developer";
       Role = "Developer Role : Build, Debug, Test, Deploy";
       Print(){
         super.Print();
         document.write(`${this.Role}`);
       }
   }
   class Admin extends Employee
   {
       FirstName = "Kiran";
       LastName = "Rao";
       Designation = "Admin";
       Role = "Admin Role : Authorizations";
       Print(){
         super.Print();
         document.write(`${this.Role}`);
       }
   }
   class Manager extends Employee
   {
```

```
        FirstName = "Tom";
        LastName = "Hanks";
        Designation = "Manager";
        Role = "Manager Role : Approvals";
        Print(){
          super.Print();
          document.write(`${this.Role}`);
        }
    }
    let employees = new Array(new Developer(), new Admin(), new Manager());
    let designation = prompt("Enter Designation");
    for(var employee of employees){
       if(employee.Designation===designation){
          employee.Print();
       }
    }
</script>
```

Day 44 : Browser Objects – 26/04/23

JavaScript Built-in Objects
- String Object
- Date Object
- Math Object
- Array Object
- Map Object

JavaScript Browser Objects [BOM -Browser Object Model]
1. window
2. location
3. navigator
4. history
5. document

window

- It provides properties and methods that are used to control the browser window.

Members:

prompt()	: It popup an input box.
alert()	: It popup an message box
open()	: It can popup any file with specified features
close()	: It can close the window
print()	: It can invoke the printer properties.

Syntax:
```
window.open("path", "title", "features");
```

Ex:
```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
</head>
<body>
   <button onclick="window.open('images/m1.jpg','Mobile','width=300
height=400')">Open</button>
   <button onclick="window.close()">Close</button>
   <button onclick="window.print()">Print</button>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <style>
      @media screen {
         tr:nth-child(even) {
            background-color: lawngreen;
         }
         tr:nth-child(odd) {
            background-color: aquamarine;
         }
         th   {
            background-color: green;
            color:white;
         }
      }
      @media print {
         button {
            display: none;
         }
         table {
            background-color: lightgray;
         }
      }
   </style>
</head>
<body>
   <table border="1" width="400">
      <caption>Products Table</caption>
      <thead>
```

```
        <tr>
          <th>Name</th>
          <th>Price</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Samsung TV</td>
          <td>40000.33</td>
        </tr>
        <tr>
          <td>Mobile</td>
          <td>13000.33</td>
        </tr>
      </tbody>
      <tfoot>
        <tr>
          <td colspan="2" align="center">
            <button onclick="window.print()">Print</button>
            <br>
            <span class="copyright"> &copy; Copyright 2023</span>
          </td>
        </tr>
      </tfoot>
    </table>
</body>
</html>
```

location object

- It provides the properties and methods that are used to known the client location details.

Members:

host : It returns the server name or IP address.
protocol : It returns the protocol: http, https, file
port : It returns the port number
pathname : It returns the current file path
href : It gets and sets URL
search : It returns the query string
hash : It returns the current hash location.

Ex:

```
<script>
   switch(location.protocol)
   {
      case "http:":
         document.write(`
         Host Name      : ${location.host}  <br>
         Protocol       : ${location.protocol} - You are using Live Server <br>
         Port Number    : ${location.port} <br>
         Path           : ${location.pathname} <br>
         URL            : ${location.href}
```

```
        `);
        break;
        case "file:":
          document.write(`
            Protocol      : ${location.protocol} - You are using File System <br>
            Path          : ${location.pathname} <br>
            URL           : ${location.href}
          `);
    }

</script>

Ex:
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function LoginClick(){
            var username = document.getElementById("UserName").value;
            var password = document.getElementById("Password").value;
            if(username=="john" && password=="john@11"){
                location.href= "shopper-template.html";
            } else {
                location.href= "error.html";
            }
        }
    </script>
</head>
<body>
    <dl>
        <h2>User Login</h2>
        <dt>User Name</dt>
        <dd><input type="text" id="UserName"></dd>
        <dt>Password</dt>
        <dd><input type="password" id="Password"></dd>
    </dl>
    <button onclick="LoginClick()">Login</button>
</body>
</html>

EX: location.search

1. search.html

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

```
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Search</title>
</head>
<body>
    <form align="center" action="results.html">
        <h1>Google</h1>
        <input type="text" name="search" size="40">
        <p>
            <button>Search</button>
        </p>
    </form>
</body>
</html>
```

2. Results.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Results</title>
    <script>
        var topics = [
            {Title: "HTML", Content:["HTML Semantics", "HTML Images", "HTML Links"]},
            {Title: "JavaScript", Content: ["Variables", "Data Types", "Operators"]}
        ];

        function bodyload(){
            var queryString = location.search;
            var searchString = queryString.substring(queryString.indexOf("=")+1);
            topics.map(topic=>{
                if(topic.Title==searchString){
                    topic.Content.map(item=>{
                        var li = document.createElement("li");
                        li.innerHTML = item;
                        document.querySelector("ol").appendChild(li);
                    })
                }
            })
        }
    </script>
</head>
<body onload="bodyload()">
    <h3>Results</h3>
    <ol>
```

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

Day 45 : JS Browser Objects[Navigator, History] – 27/04/23

Browser Objects
- window
- location
    - host
    - port
    - protocol
    - href
    - pathname
    - search

- hash is used to access the current hash location, which is an ID reference.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .topic {
            width: 200px;
            box-shadow: 2px 2px 2px black;
            padding: 5px;
            margin: 10px;
        }
        .container {
            display: flex;
        }
        ul {
            list-style: none;
            display: flex;
            margin-bottom: 50px;
        }
        li {
            margin-right: 100px;
        }
        .topic:target {
            background-color: black;
            color:white;
        }
    </style>
```

```
    <script>
      function FetchClick(){
        var topic = "";
        var now = new Date();
        switch(location.hash){
          case "#html":
          topic += "HTML Tutorial - " + now.toLocaleTimeString() + "<br>";
          break;
          case "#js":
          topic += "JavaScript Basics - " + now.toLocaleTimeString() + "<br>";
          break;
          case "#css":
          topic += "CSS Examples - " + now.toLocaleTimeString() + "<br>";
          break;
          case "#jq":
          topic += "jQuery library - " + now.toLocaleTimeString() + "<br>";
          break;
        }
        document.getElementById("topics").innerHTML += topic;
      }
    </script>
  </head>
  <body>
    <ul>
      <li><a href="#html">HTML</a></li>
      <li><a href="#css">CSS</a></li>
      <li><a href="#js">JavaScript</a></li>
      <li><a href="#jq">jQuery</a></li>
    </ul>
    <div class="container">
      <div class="topic" id="html">
        <h2>HTML</h2>
        <p>It is a markup language.</p>
      </div>
      <div class="topic" id="css">
        <h2>CSS</h2>
        <p>It defines styles for HTML.</p>
      </div>
      <div class="topic" id="js">
        <h2>JavaScript</h2>
        <p>It is a language for DOM</p>
      </div>
      <div class="topic" id="jq">
        <h2>jQuery</h2>
        <p>It is a JS library</p>
      </div>
    </div>
    <p>
      <h3>Recently Viewed <button onclick="FetchClick()">Fetch</button></h3>
      <p id="topics"></p>
    </p>
```

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

location.reload()

- It can reload the current page.

Syntax:

```
<button onclick="location.reload()">
```

HTML:

```
<meta http-equiv="refresh" content="4">
```

Navigator Object

- It is used to access client browser details.

Members:

```
appName          : It returns the browser family name.
appVersion       : It returns the browser version
platform         : It returns the current platform
language         : It returns browser current language
cookieEnabled    : It returns the cookie status.
plugins[]        : It returns the collection of all plugins installed
mimeTypes[]      : It returns all MIME types supported.
geoLocation()    : It returns the current geo location client.
```

Ex:

```
<script>
   document.write(`
     Browser Family : ${navigator.appName} <br>
     Browser Version: ${navigator.appVersion} <br>
     Language       : ${navigator.language} <br>
     Platform       : ${navigator.platform} <br>
     Cookie         : ${(navigator.cookieEnabled==true)?"Cookies Enabled":"Cookies Disabled"} <br>
   `);
</script>
```

Ex:

```
<script>
   for(var item of navigator.plugins)
   {
      document.write(item.name + "<br>");
   }
</script>
```

Ex:

```
<script>
   if(navigator.plugins['PDF Viewer']==undefined){
      document.write(`PDF Viewer is not available`);
```

```
    } else {
        document.write(`PDF Viewer is Working`);
    }
</script>
```

Ex: MIME = Multipurpose Internet Mail Extentions

FAQ: What are the image types supported in Web?
Ans :

MIME	
image/jpeg	.jpg, .jpeg, .jfif
image/apng	
image/gif	
image/tiff	
image/png	
svg	
webp	
"text/javascript"	.js
"text/css"	.css

Ex:

```
<script>
    for(var item of navigator.mimeTypes){
        document.write(item.type  + "<br>");
    }
</script>
```

FAQ: How to verify JavaScript enabled or not?
Ans: By using HTML element <noscript>

Ex: Geo Location

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function bodyload(){
            navigator.geolocation.getCurrentPosition(function(position){
                document.querySelector("p").innerHTML = `
                    Latitude : ${position.coords.latitude} <br>
                    Longitude : ${position.coords.longitude}
                `;
            })
        }
    </script>
</head>
```

```
<body onload="bodyload()">
  <p></p>
</body>
</html>
```

history object

- It is used to access the current browsing history.

Members:

length : It returns total count of pages in current history
back() : move to previous page in history
forward() : move to next page in history
go() : goto any specific page in history

```
go('page.html')
go(1)       forward
go(-1)      backward
```

Ex:

```
<script>
   document.write(`
      Total Count of Page in Current History : ${history.length}
   `);
</script>
<button onclick="history.back()">Back</button>
```

Day 46 : Events – 28/04/23

JavaScript Browser Events

- Event is a message sent by sender to its subscriber in order to notify change.
- Event follows a software design pattern called "Observer".
- Observer is a communication pattern.
- Event uses a function pointer mechanism. [Delegate = function pointer]

Syntax:

```
function  InsertClick()
{
}
<button onclick="InsertClick()">
```

function InsertClick() { } => Subscriber
onclick="InsertClick()" => Sender

- Subscriber defines the actions to perform.
- Sender triggers the actions.

What is Event Handler? What is Event?

onclick => Event
onclick=InsertClick() => EventHandler

What is EventListner ?

document.querySelector("button").addEventListener = { }
It allows an element to configure the event dynamically.

Event Arguments:
- Every event handler have 2 default arguments

a) this
b) event

```
<button  onclick="InsertClick(this, event)">
```

this => sends information about current element. [button]
id, name, class, width, height, value etc..

event => sends information about current event. [onclick]
clientX, clientY, shiftKey, ctrlKey, altKey etc..

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function InsertClick(obj)
      {
         document.write(`
            Button Id   : ${obj.id} <br>
            Button Name : ${obj.name} <br>
            Class Name  : ${obj.className}
         `);
      }
   </script>
</head>
<body>
   <button id="btnInsert" name="Insert" class="btn btn-primary"
onclick="InsertClick(this)">Insert</button>
</body>
</html>
```

```
<button onclick="InsertClick(this.id)">        only ID is sent
<button onclick="InsertClick(this)">           all properties of button are sent
<button onclick="InsertClick(this.id, this.name,..)>
```

Ex: Event Argument

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function InsertClick(e)
      {
         document.write(`
            X Position  : ${e.clientX}  <br>
            Ctrl Key    : ${e.ctrlKey}
         `);
      }
   </script>
</head>
<body>
   <button id="btnInsert" name="Insert" class="btn btn-primary"
onclick="InsertClick(event)">Insert</button>
</body>
</html>
```

Syntax:

```
      <button onclick="InsertClick(event)">              all details
      <button onclick="InsertClick(event.altKey)">     specific
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function InsertClick(e, obj)
      {
         document.write(`
            X Position  : ${e.clientX}  <br>
            Ctrl Key    : ${e.ctrlKey}  <br>
            Id          : ${obj.id} <br>
            Name        : ${obj.name}
         `);
      }
```

```
    </script>
</head>
<body>
    <button id="btnInsert" name="Insert" class="btn btn-primary" onclick="InsertClick(event,
this)">Insert</button>
</body>
</html>
```

Event - Custom Arguments

- Event allows to send custom arguments.
- You can define along with default args or individually.
- Argument can be any type
    a) Primitive
    b) Non-Primitive

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function InsertClick(msg)
        {
            document.write(`<h2>${msg}</h2>`);
        }
    </script>
</head>
<body>
    <button onclick="InsertClick('Record Inserted')">Insert</button>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function InsertClick(obj, ...product)
        {
            var [id, name, stock, rating] = product;
            document.write(`
                Id: ${id} <br>
                Name : ${name} <br>
                Stock : ${stock} <br>
```

```
            Rating : ${rating.rate} [${rating.count}] <br>
            Button Id : ${obj.id}
        `);
    }
  </script>
</head>
<body>
  <button id="btnInsert" onclick="InsertClick(this, 1, 'Samsung TV', true, {rate:4.5,
count:3400})">Insert</button>
</body>
</html>
```

Browser Event Types

- Events is JavaScript are not related to elements, they are related to browser.
- Events are classified into various groups

1. Mouse Events
2. Keyboard Events
3. Button Events
4. Form Events
5. Timer Events
6. Clipboard Events
7. Touch Events
8. Element State Events etc..

Mouse Events

- onmouseover
- onmouseout
- onmousedown
- onmouseup
- onmousemove

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    function ShowDetails(){
      document.querySelector("p").innerHTML = `
        <b>Special Offer</b> <i> 50% OFF </i> on Nike Footwear.
      `;
      document.querySelector("img").style.width = "300px";
    }
    function HideDetails(){
      document.querySelector("p").innerHTML = "";
      document.querySelector("img").style.width = "100px";
    }
```

```
    </script>
</head>
<body>
    <img onmouseover="ShowDetails()" onmouseout="HideDetails()" src="images/shoe.jpg"
width="100" height="100">
    <p></p>
</body>
</html>


Ex:
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function ShowDetails(){
            document.querySelector("p").innerHTML = `
                <b>Special Offer</b> <i> 50% OFF </i> on Nike Footwear.
            `;
        }
        function HideDetails(){
            document.querySelector("p").innerHTML = "Hold down mouse button of Shoe to view
offer.";
        }
    </script>
</head>
<body>
    <img onmousedown="ShowDetails()" onmouseup="HideDetails()" src="images/shoe.jpg"
width="100" height="100">
    <p>Hold down mouse button of Shoe to view offer.</p>
</body>
</html>

Ex:
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        body {
            display: grid;
            grid-template-columns: 2fr 10fr;
        }
        nav div {
```

```
            border:1px solid blue;
            padding: 2px;
            width: 50px;
            height: 50px;
            margin-bottom: 20px;
        }
        nav div:hover {
            cursor: grab;
        }
    </style>
    <script>
        function ImageHover(src){
            document.getElementById("preview").src = src;
        }
    </script>
</head>
<body>
    <nav>
        <div>
            <img onmouseover="ImageHover(this.src)" src="images/m1.jpg" width="50" height="50">
        </div>
        <div>
            <img onmouseover="ImageHover(this.src)" src="images/m2.jpg" width="50" height="50">
        </div>
        <div>
            <img onmouseover="ImageHover(this.src)" src="images/m3.jpg" width="50" height="50">
        </div>
        <div>
            <img onmouseover="ImageHover(this.src)" src="images/m4.jpg" width="50" height="50">
        </div>
        <div>
            <img onmouseover="ImageHover(this.src)" src="images/m5.jpg" width="50" height="50">
        </div>
    </nav>
    <main>
        <img id="preview" src="images/m1.jpg" width="300" height="400">
    </main>
</body>
</html>
```

Day 47 : Mouse, Button and Keyboard Events – 29/04/23

Mouse Events
- onmouseover
- onmouseout
- onmousedown
- onmouseup
- onmousemove

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function GetPosition(x, y) {
         var flag = document.querySelector("img");
         flag.style.position = "fixed";
         flag.style.left = x + "px";
         flag.style.top = y + "px";
         document.querySelector("div").innerHTML = `X=${x}px <br> Y=${y}px`;
      }
   </script>
</head>
<body onmousemove="GetPosition(event.clientX, event.clientY)">
   <div style="height: 1000px;"></div>
   <img src="images/flag.gif" width="50" height="50">
</body>
</html>
```

Keyboard Events

-onkeyup] good to handle chars
-onkeydown] It will not recognize the code until the char is finished
-onkeypress : It is good for handling keycodes.

Event Properties
keyCode
charCode
which
shiftKey
ctrlKey
altKey

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function VerifyUserId(userid){
         var userError = document.getElementById("userError");

         fetch("../data/users.json")
         .then((res)=> {
            return res.json();
```

```
        })
        .then(users=>{
            for(var user of users){
                if(user.UserId===userid){
                    userError.innerHTML = "User Id Taken - Try Another".fontcolor('red');
                    break;
                } else {
                    userError.innerHTML = "User Id Available".fontcolor('green');
                }
            }
        })
    }
    function VerifyCaps(e){
        var pwdError = document.getElementById("pwdError");
        console.log(e.keyCode + "\n" + e.which);
        if(e.keyCode>=65 || e.which>=65 && e.keyCode<=90 || e.which<=90) {
            pwdError.style.display = "block";
        } else {
            pwdError.style.display = "none";
        }
    }
  </script>
</head>
<body>
  <dl>
    <h3>Register User</h3>
    <dt>User Id</dt>
    <dd><input type="text" onkeyup="VerifyUserId(this.value)" id="UserId"></dd>
    <dd id="userError"></dd>
    <dt>Password</dt>
    <dd><input type="password" id="password" onkeypress="VerifyCaps(event)"></dd>
    <dd id="pwdError" style="color:goldenrod; display: none;">
      Warning : Caps is ON
    </dd>
  </dl>
</body>
</html>
```

Button Events

- onclick
- ondblclick
- oncontextmenu
- onselectstart => click and drag

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
```

```
    <script>
        document.oncontextmenu = function(){
            alert(`Right Click not allowed`);
            return false;
        }
        document.onselectstart = function(){
            return false;
        }
    </script>
</head>
<body>
    <h2>Right Click is disabled on this page.</h2>
    <img src="images/m1.jpg" ondblclick="window.open('images/m1.jpg','Mobile','width=400
height=400')" width="50" height="50">
    <p>double click to view large</p>
</body>
</html>
```

Day 48 : Events and Examples – 02/05/23

Mouse Events
- onmouseover
- onmouseout
- onmousedown
- onmouseup
- onmousemove

Keyboard Events
- onkeyup
- onkeydown
- onkeypress

Button Events
- onclick
- ondblclick
- oncontextmenu
- onselectstart

Element State Events
- onfocus : Element gets focus
- onblur : Element looses focus
- onchecked : If checkbox or radio are checked
- onchange : If value changes

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function ShowTip(){
            document.getElementById("msg").innerHTML = "Name in Block Letters Only";
        }
        function ChangeCase(){
            document.getElementById("msg").innerHTML = "";
            var username = document.getElementById("UserName").value;
            document.getElementById("UserName").value = username.toUpperCase();
        }
    </script>
</head>
<body>
    <dl>
        <dt>Name</dt>
        <dd><input type="text" onfocus="ShowTip()" onblur="ChangeCase()" id="UserName"></dd>
        <dd id="msg"></dd>
    </dl>
</body>
</html>
```

Form Events
- onsubmit : It defines the actions to perform when form submitted.
- onreset : It defines the actions to perform when form resets.

Note: Form events are writted for <form> element.
Form events will fireup only on "submit and reset" buttons.

a) Generic : submit and reset
b) Non-Generic : button

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form onsubmit="alert('form will submit its data to server')" onreset="alert('Your form will reset-
Data will erase')">
        User Name :
        <input type="text" name="UserName">
        <button type="submit">Submit</button>
        <button type="reset">Cancel</button>
    </form>
</body>
</html>
```

FAQ: Can we submit form data to server on any other element event.
How to submit form on dropdown change.
[form can be submitted implicitly only by using submit button]

Ans: By using "form" element "submit()" methdo.

Syntax:
formName.submit();

FAQ: How to set focus to any element dynamically?
Ans: By using focus().

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function PostClick(){
        frmLogin.submit();
      }
      function VerifyName(){
         var username = document.getElementById("UserName").value;
         if(username.length==4) {
            document.getElementById("password").focus();
            document.getElementById("UserName").disabled=true;
         }
      }
   </script>
</head>
<body>
   <form name="frmLogin" onsubmit="alert('form will submit its data to server')"
onreset="alert('Your form will reset- Data will erase')">
      User Name :
      <input type="text" onkeyup="VerifyName()" id="UserName" name="UserName">
      Password:
      <input type="password" name="Password" id="password">
      <button type="submit">Submit</button>
      <button type="reset">Cancel</button>
      <button type="button" onclick="PostClick()">Post</button>
   </form>
</body>
</html>
```

Clipboard Events:

- oncut
- oncopy

- onpaste

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>

      function Cut(){
         document.querySelector("p").innerHTML="Removed - Copied to clipboard";
      }
      function Copy(){
         document.querySelector("p").innerHTML= "Copied to clipboard";
      }
      function Paste(){
         document.querySelector("p").innerHTML = "Inserted from clipboard";
      }
      document.oncut = function(){
         alert("Cut not allowed");
         return false;
      }
   </script>
</head>
<body>
  <textarea id="msg" oncut="Cut()" oncopy="Copy()" onpaste="Paste()" rows="4"
cols="40">Welcome to JavaScript Events</textarea>
  <p></p>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>

      document.oncut = function(){
         alert("Cut not allowed");
         return false;
      }
   </script>
```

```
</head>
<body>
  <textarea id="msg" oncut="Cut()" oncopy="Copy()" onpaste="Paste()" rows="4"
cols="40">Welcome to JavaScript Events</textarea>
  <p></p>
</body>
</html>
```

Touch Events

- ontouchstart
- ontouchend
- ontouchmove

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function MoveImage(e){
         var x = e.touches[0].clientX;
         var y = e.touches[0].clientY;
         var img = document.querySelector("img");

         img.style.position = "fixed";
         img.style.top = y + "px";
         img.style.left = x + "px";
      }
   </script>
</head>
<body ontouchmove="MoveImage(event)">
  <img src="../public/images/m1.jpg" width="200" height="300">
</body>
</html>
```

Day 49 : Timer Events – 03/05/23

Timer Events

- setInterval()
- clearInterval()
- setTimeout()
- clearTimeout()

setTimeout()

- It loads the given function into memory and delays its execution by specified time interval.

Syntax:

```
    setTimeout(function(){ }, timeInterval)

Ex:
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      function msg1(){
         document.querySelector("h2").innerHTML = "Hello !";
      }
      function msg2(){
         document.querySelector("h2").innerHTML = "How are you?";
      }
      function msg3(){
         document.querySelector("h2").innerHTML = "Welcome to JavaScript";
      }
      var m1, m2, m3;
      function bodyload(){
        m1 =  setTimeout(msg1, 3000);
        m2 =  setTimeout(msg2, 5000);
        m3 = setTimeout(msg3, 10000);
      }
      function ClearMessage2(){
         clearTimeout(m2);
      }
   </script>
</head>
<body onload="bodyload()">
   <button onclick="ClearMessage2()">Clear Message 2</button>
   <h2 align="center"></h2>
</body>
</html>

                        setInterval()
- It loads the task into memory and will release at regular time interval.
- It will repeat the task until removed from memory.

Ex:
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Interval</title>
   <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
   <style>
```

```
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 500px;
            text-align: center;
        }
    </style>
    <script>
        function FetchClick(){
            document.getElementById("buttonContainer").style.display = "none";
            document.getElementById("statusContainer").style.display = "block";
            setInterval(SetCount, 100);
        }
        var count = 0;
        function SetCount(){
            count++;
            document.getElementById("count").innerHTML = `${count} % completed`;
            if(count==100) {
                document.getElementById("statusContainer").style.display = "none";
                document.getElementById("imageContainer").style.display = "block";
            }
        }
    </script>
</head>
<body >
    <div>
        <div id="buttonContainer">
            <button onclick="FetchClick()" class="btn btn-primary">Fetch Image</button>
        </div>
        <div id="statusContainer" style="display: none;">
            <span class="spinner-border"></span>
            <div>Loading...
            <p id="count"></p>
        </div>
    </div>
    <div id="imageContainer" style="display: none;">
        <img src="../public/images/m1.jpg" width="300" height="300">
    </div>
</body>
</html>

Ex:
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Interval</title>
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
```

```
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 500px;
            text-align: center;
        }
    </style>
    <script>
        var timer;
        function FetchClick(){
            document.getElementById("buttonContainer").style.display = "none";
            document.getElementById("statusContainer").style.display = "block";
            timer = setInterval(SetCount, 100);
        }
        var count = 0;
        function SetCount(){
            count++;
            document.getElementById("progress").value = count;
            document.getElementById("count").innerHTML = `${count} % completed`;
            if(count==100) {
                document.getElementById("statusContainer").style.display = "none";
                document.getElementById("imageContainer").style.display = "block";
            }
        }
        function PauseClick(){
            clearInterval(timer);
            document.getElementById("count").innerHTML = `${count} % Paused`;
        }
        function ResumeClick(){
            timer = setInterval(SetCount, 100);
            document.getElementById("count").innerHTML = `${count} % Completed`;
        }
    </script>
</head>
<body >
    <div>
        <div id="buttonContainer">
            <button onclick="FetchClick()" class="btn btn-primary">Fetch Image</button>
        </div>
        <div id="statusContainer" style="display: none;">
            <progress id="progress" min="1" max="100" value="1"></progress>
            <p id="count"></p>
            <p>
                <button onclick="ResumeClick()">></button>
                <button onclick="PauseClick()">||</button>
            </p>
        </div>
    </div>
    <div id="imageContainer" style="display: none;">
```

```
        <img src="../public/images/m1.jpg" width="300" height="300">
    </div>
</body>
</html>
```

Day 50 : JS Events Interval – 04/05/23

Ex: Set Interval

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Slide Show</title>
    <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <script>
        var count = 0;
        function GetProduct(){
            count++;
            fetch(`http://fakestoreapi.com/products/${count}`)
            .then((response)=> {
                return response.json();
            })
            .then((product)=>{
                console.log(product);
                document.getElementById("title").innerHTML = product.title;
                document.getElementById("pic").src = product.image;
            })
        }
        function bodyload(){
            GetProduct();

        }
        var show;
        function PlayClick(){
            show = setInterval(GetProduct, 5000);
            document.getElementById("status").innerHTML = "Slide Show Started";
        }
        function PauseClick(){
            clearInterval(show);
            document.getElementById("status").innerHTML = "Slide Show - Paused";
        }
    </script>
</head>
<body onload="bodyload()" class="container-fluid d-flex justify-content-center">
    <div class="mt-2 card w-50">
        <div class="card-header text-center">
```

```
            <p id="title"></p>
            <p id="status"></p>
        </div>
        <div class="card-body">
            <img width="100%" height="300px" id="pic">
        </div>
        <div class="card-footer text-center">
            <button class="btn btn-primary" onclick="PlayClick()">
                <span class="bi bi-play"></span>
            </button>
            <button class="btn btn-danger" onclick="PauseClick()">
                <span class="bi bi-pause"></span>
            </button>
        </div>
    </div>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .container{
            display: flex;
            justify-content: center;
            align-items: center;
            height: 500px;
        }
        @keyframes zoomIn
        {
            from {
                width: 30px;
                height: 40px;
            }
            to {
                width: 300px;
                height: 400px;
            }
        }

    </style>
    <script>
        function ZoomClick(){
            var img = document.querySelector("img");
            img.style.animationName = "zoomIn";
            img.style.animationDuration = "5s";
```

```
            img.style.animationIterationCount = "infinite";
        }
    </script>
</head>
<body>
    <button onclick="ZoomClick()">Zoom</button>
    <div class="container">
        <div>
            <img src="images/m1.jpg" width="300" height="400">
        </div>
    </div>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Slide Show</title>
    <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css">
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <script>
        var count = 0;
        function GetProduct(){
            count++;
            fetch(`http://fakestoreapi.com/products/${count}`)
            .then((response)=> {
                return response.json();
            })
            .then((product)=>{
                console.log(product);
                document.getElementById("title").innerHTML = product.title;
                document.getElementById("pic").src = product.image;
            })
        }
        function bodyload(){
            GetProduct();

        }
        var show;
        function PlayClick(){
            show = setInterval(GetProduct, 5000);
            document.getElementById("status").innerHTML = "Slide Show Started";
        }
        function PauseClick(){
            clearInterval(show);
            document.getElementById("status").innerHTML = "Slide Show - Paused";
        }
```

```
        function SetAnimation(){
            var img = document.getElementById("pic");
            img.style.animationName = "ZoomIn";
            img.style.animationDuration = "5s";
        }
    </script>
    <style>
        @keyframes ZoomIn {
            from {
              opacity: 0;
            }
            to {
               opacity: 1;
            }
        }
    </style>
</head>
<body onload="bodyload()" class="container-fluid d-flex justify-content-center">
    <div class="mt-2 card w-50">
        <div class="card-header text-center">
            <p id="title"></p>
            <p id="status"></p>
        </div>
        <div class="card-body" style="height:300px">
            <img width="100%" onload="SetAnimation()" height="250px" id="pic">
        </div>
        <div class="card-footer text-center">
            <button class="btn btn-primary" onclick="PlayClick()">
                <span class="bi bi-play"></span>
            </button>
            <button class="btn btn-danger" onclick="PauseClick()">
                <span class="bi bi-pause"></span>
            </button>
        </div>
    </div>
</body>
</html>
```

Summary of Events

1. mouse
2. keyboard
3. button
4. clipboard
5. form
6. element state
7. touch
8. timer

Adding Event Listener

FAQ: How to create a button and add to HTML page dynamically?

Ans : By use "createElement()" method of document object.

```
document.createElement("button");
```

FAQ: How to define event of dynamic element?
Ans: By using "addEventListener"

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Event</title>
   <script>
      function bodyload(){
         var button = document.createElement("button");
         button.innerHTML = "Dynamic Button";
         document.querySelector(".container").appendChild(button);
         button.addEventListener("click", function(){
            alert(`Dynamic Button Clicked`);
         })
      }
   </script>
</head>
<body onload="bodyload()">
   <div class="container">

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

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Event</title>
   <script>
      function bodyload(){
         var label =  document.createElement("label");
         var input = document.createElement("input");
         var button = document.createElement("button");

         label.innerHTML = "Your Password : ";
         input.type  = "password";
```

```
            input.id  = "txtPassword";

            button.innerHTML = "Login";

            var container = document.querySelector(".container");
            container.appendChild(label);
            container.appendChild(input);
            container.appendChild(button);

            button.addEventListener("click", function(){
               if(input.value=="admin") {
                  document.write("Success");
               } else {
                  document.write("Invalid");
               }
            })
         }
      </script>
   </head>
   <body onload="bodyload()">
      <div class="container">

      </div>
   </body>
   </html>

   Ex:
   <!DOCTYPE html>
   <html lang="en">
   <head>
      <meta charset="UTF-8">
      <meta http-equiv="X-UA-Compatible" content="IE=edge">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Event</title>
      <script>
         function bodyload(){
            var label =  document.createElement("label");
            var input = document.createElement("input");
            var button = document.createElement("button");

            label.innerHTML = "Your Password : ";
            input.type  = "password";
            input.id  = "txtPassword";

            button.innerHTML = "Login";

            var container = document.querySelector(".container");
            container.appendChild(label);
            container.appendChild(input);
            container.appendChild(button);
```

```
            button.addEventListener("click", function(e){
                console.log(e.clientX);
                if(input.value=="admin") {
                    document.write("Success");
                } else {
                    document.write("Invalid");
                }
            })
        }
    </script>
</head>
<body onload="bodyload()">
    <div class="container">

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

Day 51 : JavaScript Promises – 05/05/23

JavaScript Promise

FAQ: What is call back?
Ans:
- It is a technique of configuring functions, which execute according state and situation.
- It comprises of a set of functions, which are executed based on any boolean expression.
- It contains function to execute on success and another that executes on failure.

Syntax:

```
    function Name(value, success, failure)
    {
        if(condition) {
            success();
        } else {
            failure();
        }
    }
```

- Callback is synchronous.
- It will not move to next task until if finish the given task.
- It is slow in resolving the function.

Ex:

```
<script>
    function FetchData(url, success, failure){
        if(url=="http://fakestoreapi.com/products&quot;) {
            success('Fetched Data Successfully..');
        } else {
            failure('Unable to fetch data');
        }
```

```
    }
    FetchData(prompt("Enter URL"),
      (msg)=>{
        document.write(`Success: ${msg}`);
      },
      (msg)=>{
        document.write(`Failure: ${msg}`);
      }
    )
</script>
```

Promise

- Promise is a proxy.
- A proxy defines about uncertenity in resolving the given issues.
- Promises comprises 3 states

    a) Pending   => Initial state neigther resolved nor rejected.
    b) Resolved   => Promise fulfilled
    c) Rejected   => Promise broken

- Promise is impiclity Asynchronous. [async, await]

Ex: Blocking Technique [Synchronous]

> npm install fs --save

> create a new file "read.js"

```
var fs = require("fs");

var data = fs.readFileSync("help.txt");

console.log(`--Reading File---`);
console.log(data.toString());
console.log(`--Read Complete--`);
```

> node read.js

Ex: Unblocking - Asyncrhonous

> read.js

```
var fs = require("fs");

console.log(`--- Reading File ----`);

fs.readFile("help.txt", function(err, data){
   if(!err){
      console.log(data.toString());
   } else {
      console.log(err);
```

```
    }
});

console.log(`---Read Complete----`);

>node read.js
```

- Promise can be used instead of callback to handle async operations.

Syntax:

```
    const   name =  new Promise(resolve, reject);
```

- Promise provides "then" and "catch".
- "then()" is executed when promise is fulfilled.
- "catch()" is executed when promise is rejected.
- "finally()" is executed in all situations.

Ex:

```
<script>
    var FetchData = new Promise((resolve, reject)=>{
        var url = prompt("Enter URL");
        if(url=="http://fakestoreapi.com/products&quot;){
          resolve('Fetched Data Successfully..');
        } else {
          reject('Unable to fetch data');
        }
    });
    FetchData.then((msg)=>{
      document.write(`Success: ${msg}`);
    }).catch((msg)=>{
      document.write(`Failure: ${msg}`);
    })
</script>
```

- Promise provides the members

```
    all()       It is an array of promises
    race()      It is a single promise
```

Ex:

```
<script>
    var GetProducts = new Promise((resolve)=>{
        resolve('Gets all products');
    });
    var GetCategories = new Promise((resolve)=>{
        resolve('Get all Categories list');
    })
    var GetCart = new Promise((resolve)=>{
        resolve('Get Your Cart Details');
    })

    Promise.all([
```

```
        GetProducts,
        GetCategories,
        GetCart
    ]);
    Promise.race([
        GetProducts,
        GetCategories,
        GetCart
    ])
</script>
```

Day 52 : JavaScript AJAX – 08/05/23

Promise - all
Promise - race

- all : It executes all promises in async technique.

Syntax:

```
Promise.all([
     promise1,
    promise2,
    promise3
]).then(collection=>{

})
```

all() [ ] type hence its return type is array.

Ex: All [Asyncrhonous]

```
<script>
    var GetProducts = new Promise((resolve)=>{
        resolve('Gets all products');
    });
    var GetCategories = new Promise((resolve)=>{
        resolve('Get all Categories list');
    })
    var GetCart = new Promise((resolve)=>{
        resolve('Get Your Cart Details');
    })
    Promise.all([
        GetProducts,
        GetCategories,
        GetCart
    ]).then(result=>{
        for(var item of result){
            console.log(item);
        }
    })
</script>
```

Ex: race [Synchronous]

```
<script>
   var GetProducts = new Promise((resolve)=>{
      resolve('Gets all products');
   });
   var GetCategories = new Promise((resolve)=>{
      resolve('Get all Categories list');
   })
   var GetCart = new Promise((resolve)=>{
      resolve('Get Your Cart Details');
   })
   Promise.race([
      GetProducts,
      GetCategories,
      GetCart
   ]).then(result=>{
      console.log(result);
   })
</script>
```

JavaScript Ajax

- Asynchronous JavaScript And XML.
- AJAX is used to configure partial post back.
- Partial post back allows to post only specific portion of page without posting entier page.
- AJAX makes the page more responsive and interactive.
- It can add new details to page without reloading the complete page.

FAQ: How to reload a page at regular time intervals?
Ans: By using <meta http-equiv="refresh" content="4">

FAQ: How page can load using JavaScript?
Ans: location.reload()

- User can stay on one page and can get access to everything on to page without reloading the complete page.

- This mechanism of application is often reffered as "SPA".
  [Single Page Applications]

- JavaScript provides "XMLHttpRequest" object, which is used to configure AJAX in web applications.

```
jQuery  $.ajax()       => XMLHttpRequest
React   axios          => XMLHttpRequest
Angular  HttpClient    => XMLHttpRequest
```

Syntax:

```
var http = new XMLHttpRequest();
```

- "http" is an XMLHttpRequest object, which is an Ajax object.

Member	Description
onreadystatechange	It defines a function to execute when ajax state changes.
readyState	It defines the ajax state [0] Initial state - request not initialized [1] Connection established with server [2] Request received [3] Request processed [4] Response ready
status	It returns status code 200 302 404 500
statusText	It returns status message OK METHOD NOT FOUND Internal Server Error 2xx Success 3xx Redirections 4xx Client Side Issues 5xx Server Side Issues
responseType	It defines the response MIME text/plain application/pdf application/xml application/json image/jpeg
responseText	It returns the response from the file that you requested.
open()	It defines the ajax request type and URL request type = GET, POST... url = path / resource location Syntax: open("GET", "URL", true) => true is async

send() It sends response to client.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Ajax</title>
   <script>
      var http = new XMLHttpRequest();
      function GetEMI(){
         http.open("GET", "emi.html", true);
         http.send();
         http.onreadystatechange = function() {
            if(http.readyState==4) {
               document.getElementById("container").innerHTML  = http.responseText;
            }
         }
      }
      function GetNasa(){
         http.open("GET", "slide-show.html", true);
         http.send();
         http.onreadystatechange = function() {
            if(http.readyState==4) {
               document.getElementById("container").innerHTML  = http.responseText;
            }
         }
      }
      function bodyload(){
         var now = new Date();
         document.getElementById("status").innerHTML = now.toLocaleTimeString();
      }
   </script>
   <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
</head>
<body class="container-fluid" onload="bodyload()">
   <h3>Page Last Fetched : <span id="status"></span></h3>
   <button onclick="GetEMI()">EMI Calculator</button>
   <button onclick="GetNasa()">Slide Show</button>
   <hr size="3" noshade color="red">
   <div id="container">

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

Day 53 : JavaScript Request Types – 09/05/23

Response Type

- HTML Response, Text Response => XMLHttpRequest.responseText

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Ajax</title>
   <script>
      var http = new XMLHttpRequest();
      function HelpClick(){
         http.open("get", "../data/help.txt");
         http.send();
         http.onreadystatechange = function(){
            if(http.readyState==4) {
               document.getElementById("container").innerHTML = http.responseText;
            }
         }
      }
   </script>
</head>
<body>
   <button onclick="HelpClick()">Help</button>
   <hr size="2" noshade>
   <pre id="container"></pre>
</body>
</html>
```

Distributed Computing

- Distributed computing allows 2 different applications running on 2 different mechines to share information.
- It also allow 2 different application running on same mechines on 2 different process to share information.
- Distributed computing architectures

```
CORBA       - Common Object Request Broken Architecture
DCOM        - Distribute Component Object Model
RMI         - Remote Method Invocation
EJB         - Enterprise Java Beans
WebService  - Java, .NET, PHP, Python, etc..
Remoting    - .NET
```

- Web Service Specifications

  a) SOAP
  b) REST
  c) JSON

SOAP
- Service Oriented Architecture Protocol
- Consumer sends XML request
- Provider sends XML response

REST
- Representational State Transfer
- Consumer sends query request
- Provider sends XML response, optionally JSON

JSON
- JavaScript Object Notation
- Consumer sends JSON request
- Provider sends JSON response.

JavaScript XMLHttpRequest object can manage the response in various formats.
        responseText   => text, html, json
        responseXML   => xml

- XML document comprises of various properties

parentNode
parentElement
childNodes
firstChild
lastChild
previousSibling
nextSibling
nodeValue
textContent

Ex:
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Ajax</title>
   <script>
      var http = new XMLHttpRequest();
      function HelpClick(){
         http.open("get", "../data/help.txt", true);
         http.send();
         http.onreadystatechange = function(){
```

```
            if(http.readyState==4) {
                document.getElementById("container").innerHTML = http.responseText;
            }
        }
    }
    function ProductsClick(){
        http.open("get", "../data/products.json", true);
        http.send();
        http.onreadystatechange = function(){
            if(http.readyState==4){
                var data = JSON.parse(http.responseText);
                for(var item of data){
                    var tr = document.createElement("tr");
                    var tdName = document.createElement("td");
                    var tdPrice = document.createElement("td");

                    tdName.innerHTML = item.Name;
                    tdPrice.innerHTML = item.Price;

                    tr.appendChild(tdName);
                    tr.appendChild(tdPrice);

                    document.querySelector("tbody").appendChild(tr);
                }
            }
        }
    }
    function XMLClick(){
        http.open("get", "../data/product.xml", true);
        http.send()
        http.onreadystatechange = function(){
            let xmldoc = http.responseXML;
            var root = xmldoc.querySelector("root");
            document.getElementById("container").innerHTML = root.firstChild.data;
        }
    }
  </script>
</head>
<body>
  <button onclick="HelpClick()">Help-Text</button>
  <button onclick="ProductsClick()">Products-JSON</button>
  <button onclick="XMLClick()">XML Data</button>
  <hr size="2" noshade>
  <pre id="container"></pre>
  <table border="1" width="300">
    <thead>
      <tr>
        <th>Name</th>
        <th>Price</th>
      </tr>
    </thead>
```

```
        <tbody>

        </tbody>
    </table>
</body>
</html>
```

- To handle Ajax calls JavaScript uses "fetch()" promise. It is implicitly asyncrhonous.

- You also used "async and await" keywords for creating async functions.
- Modern JavaScript prefers using promise.

Syntax: without a promise

```
    async  function Name(){
        // create a asyncrhonous function
        // execute using "await"
    }
```

Syntax: With promise

```
     function Name(){
        return Promise.resolve();
     }
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        async function Welcome(){
              let pro = new Promise((resolve, reject)=>{
                 resolve('Welcome to JavaScript Async');
              });
              document.querySelector("p").innerHTML = await pro;
        }
        function btnClick(){
           Welcome();
        }
    </script>
</head>
<body>
    <button onclick="btnClick()">Click</button>
    <p></p>
</body>
</html>
```

Day 54 : JavaScript Data Structures – 10/05/23

JavaScript Data Structure and Algorithms

- Data Structure in computer programming defines how data is stored and manipulated, which includes adding, removing, sorting, filtering etc.

- JavaScript provides array as collection, which allows random access.
- JavaScript provides other type of data structures like map(), set(), object..
- You have to manually configure and create the structure for data manipulation.
- Commonly used Data Structures in programming

1. Stack
2. Queue
3. Hash Table
4. Linked List
5. Tree - Binary Tree
6. Graph etc..

Stack

- It uses the mechanism LIFO. [Last-in-First Out]
- The last value added into collection will be the first value to read.
- Usually stack comprisess of methods

pop()	It removes and returns the last item
push()	It adds a new item as last item.
peek()	It reads and returns the last item.
size()	It returns the size

Ex:
stack.js

```
export class Stack
{
   data = {};
   length = 0;
   pop(){
     this.length--;
     var value = this.data[this.length];
     delete this.data[this.length];
     return value;
   }
   push(value){
    this.data[this.length]=value;
    this.length++;
   }
   peek(){
     return this.data[this.length-1];
   }
   size(){
```

```
        return this.length;
    }
}
```

stack-demo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="module">
        import { Stack } from "../library/ds/stack.js";
        let collection = new Stack();
        collection.push("A");
        collection.push("B");
        console.log(collection.size());
        console.log(collection.pop());
        console.log(collection.size());
        console.log(collection.peek());
        console.log(collection.size());

    </script>
</head>
<body>

</body>
</html>
```

Queue

- It uses the mechanism FIFO

Methods:

enqueue()	add new item
dequeue()	remove return first item
size()	

Ex:

queue.js

```
export class Queue
{
    collection = [];

    enqueue(value){
      this.collection.push(value);
    }
    dequeue(){
```

```
        return this.collection.shift();
    }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="module">
        import  { Queue } from "../library/ds/queue.js";
        let collection = new Queue();
        collection.enqueue("A");
        collection.enqueue("B");
        collection.enqueue("C");
        console.log(collection.dequeue());

    </script>
</head>
<body>

</body>
</html>
```

Linked List

- Collection with nodes where the current node will invoke the next node.

Methods:

```
head()      returns element at 0
add()
addAt()
remove()
removeAt()
size()
```

Day 55 : JS Bank Token System – Speech Enabled – 11/05/23

Bank Token System Example With Speech Enabled

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
<script type="module">
    import { Queue } from "../library/ds/queue.js";
    var tokens = new Queue();
    var tokenNumber = 1;
    var GenerateTokenButton = document.getElementById("btnGenerateToken");

    let speech = new SpeechSynthesisUtterance();

    function LoadTokens(){
        document.getElementById("lstTokens").innerHTML="";
        for(var item of tokens.collection){
            var opt = document.createElement("option");
            opt.text = item;
            opt.value = item;
            document.getElementById("lstTokens").appendChild(opt);
        }
    }

    GenerateTokenButton.addEventListener("click",()=>{
        var username = prompt("Enter Your Name");
        tokens.enqueue(`${tokenNumber} [${username}]`);
        tokenNumber++;
        alert(`Token Generated`);
        LoadTokens();
    })

    document.getElementById("btnCounter1").addEventListener("click",()=>{
        document.getElementById("lstCounter1").innerHTML = "";
        var opt = document.createElement("option");
        opt.text = tokens.dequeue();
        document.getElementById("lstCounter1").appendChild(opt);
        if(tokens.size()==0) {
            document.getElementById("lblCounter1").innerHTML = `No more Customers`;
        } else {

            document.getElementById("lblCounter1").innerHTML = `Serving : ${opt.text}`;
        }
        LoadTokens();
    })

    document.getElementById("btnCounter2").addEventListener("click",()=>{
        document.getElementById("lstCounter2").innerHTML = "";
        var opt = document.createElement("option");
        opt.text = tokens.dequeue();
        speech.text = `Token ${opt.text} go to Counter Number 2`;
        window.speechSynthesis.speak(speech);
        document.getElementById("lstCounter2").appendChild(opt);
        LoadTokens();
```

```
        })

        document.getElementById("btnCounter3").addEventListener("click",()=>{
            document.getElementById("lstCounter3").innerHTML = "";
            var opt = document.createElement("option");
            opt.text = tokens.dequeue();
            speech.text = `Token ${opt.text} go to Counter Number 3`;
            window.speechSynthesis.speak(speech);
            document.getElementById("lstCounter3").appendChild(opt);
            LoadTokens();
        })

        document.getElementById("Counter1CheckBox").addEventListener("change",(e)=>{
            if(e.target.checked){
                document.getElementById("btnCounter1").disabled=false;
            } else {
                document.getElementById("btnCounter1").disabled=true;
            }
        })

        document.getElementById("Counter2CheckBox").addEventListener("change",(e)=>{
            if(e.target.checked){
                document.getElementById("btnCounter2").disabled=false;
            } else {
                document.getElementById("btnCounter2").disabled=true;
            }
        })

        document.getElementById("Counter3CheckBox").addEventListener("change",(e)=>{
            if(e.target.checked){
                document.getElementById("btnCounter3").disabled=false;
            } else {
                document.getElementById("btnCounter3").disabled=true;
            }
        })
    </script>
</head>
<body class="container-fluid">
    <div class="row mt-3">
        <div class="col">
          <h4>Counter-1 <span class="form-switch"><input type="checkbox"  id="Counter1CheckBox"
class="form-check-input"></span> </h4>
          <select id="lstCounter1" size="3" class="form-select">

          </select>
          <button id="btnCounter1" disabled class="btn mt-2 btn-success">Call Customer</button>
          <label class="form-label mt-3 bg-dark text-white" id="lblCounter1"></label>
        </div>
        <div class="col">
            <h4>Counter-2 <span class="form-switch"><input type="checkbox" id="Counter2CheckBox"
class="form-check-input"></span> </h4>
```

```
            <select id="lstCounter2" size="3" class="form-select">

            </select>
            <button id="btnCounter2" disabled class="btn mt-2 btn-success">Call Customer</button>
            <label class="form-label bg-dark text-white" id="lblCounter2"></label>
        </div>
        <div class="col">
            <h4>Counter-3 <span class="form-switch"><input type="checkbox" id="Counter3CheckBox"
class="form-check-input"></span> </h4>
            <select id="lstCounter3" size="3" class="form-select">

            </select>
            <button id="btnCounter3" disabled class="btn mt-2 btn-success">Call Customer</button>
            <label class="form-label bg-dark text-white" id="lblCounter3"></label>
        </div>
    </div>
    <div class="row text-center" style="margin-top: 200px;">
      <div>
       <select id="lstTokens" size="3" class="form-select" style="height: 100px;">

       </select>
       <button class="btn mt-2 btn-primary" id="btnGenerateToken">Generate Token</button>
      </div>
    </div>
</body>
</html>
```

Day 56 : JS Algorithms and Storage – 12/05/23

JavaScript Algorithms

- Algorithms define how your application perform when input size grows.
- Time taken to perform any task.
- Big O Notation
    Linear - O(n)

Ex:

```
<script>
    function PrintTotal(number)
    {
        var result = 0;
        for(var i=1; i<=number; i++){
            result = result + i;
        }
        return result;
    }
    let startProfiling = 0;
    let endProfiling = 0;
        startProfiling = performance.now();
        document.write(PrintTotal(100000) + "<br>");
        endProfiling = performance.now();
```

```
        let totalProfileTime = endProfiling - startProfiling;
        document.write("Total Time : " + totalProfileTime);
</script>
```

Constant - O(1)

Ex:

```
<script>
   function PrintPow(number, p)
   {
      return Math.pow(number,p);
   }
   let startProfiling = 0;
   let endProfiling = 0;
      startProfiling = performance.now();
      document.write(PrintPow(3,10) + "<br>");
      endProfiling = performance.now();
      let totalProfileTime = endProfiling - startProfiling;
      document.write("Total Time : " + totalProfileTime);
</script>
```

Quad O(n2)

Cubic O(n3)

Ex:

```
<script>
   function FetchData(){
      let startProfiling = 0;
      let endProfiling = 0;
      startProfiling = performance.now();
      fetch("http://fakestoreapi.com/products&quot;)
      .then(res=>res.json())
      .then(data=>{

         for(var item of data){
            document.write(item.title + "<br>");
         }
      })
      endProfiling = performance.now();
      let totalProfileTime = endProfiling - startProfiling;
      console.log(totalProfileTime);
   }

   FetchData();


</script>
```

Ex:

```
<script>
   function FetchData(){
```

```
        let startProfiling = 0;
        let endProfiling = 0;
        startProfiling = performance.now();
        fetch("http://fakestoreapi.com/products&quot;)
        .then(res=>res.json())
        .then(data=>{
           var result = data.filter(item=> item.category=="electronics");
           result.map(product=>
             document.write(`<li>${product.title}</li>`)
           )
        })
        endProfiling = performance.now();
        let totalProfileTime = endProfiling - startProfiling;
        console.log(totalProfileTime);
    }

    FetchData();


</script>
```

JavaScript Local Storage

- HTTP is a state less protocol.
- It can't remember information between requests.
- It works with the mechanism
  "Go-Get-Forget"
- Go : Establish connection with server
- Get : Fetch information from server
- Forget: Erase all the traces.
- Web applications use various state management techniques to keep the data available across requests.
  1. Query String
  2. Cookies
  3. Session

Cookie:

- It is a simple text document, that comprises of client details.
- It is appended into browser memory or into client device [HDD]

Day 57 : JS Session Storage – 13/05/23

JavaScript State Management

- Query String 2048
- Local Storage 10MB
- Session Storage 10MB
- Cookie Storage 4MB

Query String

- It transports data by appending it into URL of address bar.
  ?key=value

- It is visible to all users
- It is not safe
- It can be bookmarked
- It is stored in browser logs.
- We have a limit for data "2048chars"
- can't handle complex data like binary

location.search

Local Storage:
- JavaScript allows to store data locally in browser.
- window.localStorage is used to configure data in local storage.
- It will never expire.
- You have to manually remove local storage.

length : total count
clear : removes all local storage
getItem : get any specific local storage
key : verify
removeItem : remove specific
setItem : add new item

Syntax:
```
localStorage.setItem("key", value);
localStorage.getItem("key");
localStorage.removeItem("key");
localStorage.clear();
```

Ex:
login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <script>
        function LoginClick(){
            var username = document.getElementById("UserName").value;
            window.localStorage.setItem("username", username);
            location.href = "home.html";
        }
    </script>
</head>
<body>
    <h2>Login</h2>
    User Name :
    <input type="text" id="UserName"> <button onclick="LoginClick()">Login</button>
</body>
```

```
</html>
```

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Home</title>
   <script>
      function bodyload(){
         var user = window.localStorage.getItem("username");
         if(user==null){
            location.href="login.html";
         } else {
            document.querySelector("p").innerHTML=`Hello ! ${user}`;
         }
      }
      function Signout(){
         window.localStorage.removeItem("username");
         location.href="login.html";
      }
   </script>
</head>
<body onload="bodyload()">
   <h2>Home</h2>
   <p></p>
   <button onclick="Signout()">Signout</button>
</body>
</html>
```

Session Storage
- It is not permanent.
- It is removed when browser is closed.
- It is not accessible to another tab in same browser.

```
length          : total count
clear           : removes all local storage
getItem         : get any specific local storage
key             : verify
removeItem      : remove specific
setItem         : add new item
```

Ex:
login.html

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

```
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Login</title>
   <script>
      function LoginClick(){
         var username = document.getElementById("UserName").value;
         window.sessionStorage.setItem("username", username);
         location.href = "home.html";
      }
   </script>
</head>
<body>
   <h2>Login</h2>
   User Name :
   <input type="text" id="UserName"> <button onclick="LoginClick()">Login</button>
</body>
</html>
```

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Home</title>
   <script>
      function bodyload(){
         var user = window.sessionStorage.getItem("username");
         if(user==null){
            location.href="login.html";
         } else {
            document.querySelector("p").innerHTML=`Hello ! ${user}`;
         }
      }
      function Signout(){
         window.sessionStorage.removeItem("username");
         location.href="login.html";
      }
   </script>
</head>
<body onload="bodyload()">
   <h2>Home</h2>
   <p></p>
   <button onclick="Signout()">Signout</button>
</body>
</html>
```

Cookies
- It is a document object.
- Cookie is a simple text document where client details are stored.
- Cookie can be temporary or persistent.
- Temporary is in-memory which is remove when browser is claose.
- Persitent is permanent with expiry date.
- cookie data is accessible to server.

Ex:
login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Login</title>
   <script>
      function LoginClick(){
         var username = document.getElementById("UserName").value;
         document.cookie = `UserName=${username}; expires=${new Date("2023-05-15
10:30:00AM")}`;
         location.href = "home.html";
      }
   </script>
</head>
<body>
   <h2>Login</h2>
   User Name :
   <input type="text" id="UserName"> <button onclick="LoginClick()">Login</button>
</body>
</html>
```

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Home</title>
   <script>
      function bodyload(){
         var user = document.cookie;
         var [username] = user.split(';');
         var result = username.substring(username.indexOf("=")+1);
         document.querySelector("p").innerHTML= `Hello ! ${result}`;
      }
      function Signout(){
```

```
            document.cookie="";
            location.href="login.html";
        }
    </script>
</head>
<body onload="bodyload()">
    <h2>Home</h2>
    <p></p>
    <button onclick="Signout()">Signout</button>
</body>
</html>
```

Web Components

- A component comprises of
  a) Presentation
  b) Logic
  c) Styles
- Component enables reusability and extensibility.
- You can create custom elements and add to page.
- Presentation is defined by using HTML
- Styles are defined by using CSS
- Logic is defined by using JavaScript

Day 58 : Custom Elements – 15/05/23

What is a component?

- It is a template that provides reusable content for web application.
- It comprises of 3 basic elements
  a) Presentation
  b) Styles
  c) Logic

- You can create custom components are you can use built-in components.
- There are also reffered as custom elements
- Custom Elements are 2 types
  a) Standalone
  b) Extended
- Standalone is creating a new HTML element manually and defining functionality

```
document.createElement("div");
```

- Extended is creating a new HTML element by extending the HtmlElement class.

Syntax:

```
class MyElement extends HTMLElement
{

}
```

- Every element must call super constructor.

```
constructor() {
   super();
}
```

- The attributes of element defined within the constructor by using "this".

```
constructor(){
   super();
   this.attributeName = value;
}
```

- The properties of class are accessed by using "accessors"
- HTML element provides an accessor
  a) get()
  b) set()
- observedAttributes() is an accessor used to access any property or attribute of element.

- attributeChangedCallBack(property, old, new) is an accessor used to set value into property.

Syntax:

```
static get  observedAttributes()
```

- static refers to contineous memory
- memory allocated for first object will be same across other objects.
- Configure shadow root for element.
- HTMLElement provides a method "connectedCallback()" , which executes automatically when element is created successfully.
- You can configure shadow root with content to render.

Syntax:

```
connectedCallBack()
{
  let shadow =  this.shadowRoot({mode: 'open | close'});
}
```

- You can render child elements using "innerHTML".

- Every element can have inner styles or external styles.

- Inner style is defined directly inside the shadow root.
- external style is defined in general HTML head or body.

Syntax:

```
let shadow = this.attachShadow({ mode : 'open' });
shadow.innerHTML = `
   <style> </style>
`
```

Ex:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
       class MyDialog extends HTMLElement
      {
         constructor(){
            super();
            this.caption = "some caption";
         }
         static get observedAttributes(){
            return ['caption'];
         }
         attributeChangedCallback(property, oldValue, newValue) {
            if(oldValue==newValue) {
               return;
            }
            this[property] = newValue;
         }
         connectedCallback(){
            let shadow = this.attachShadow({mode: 'open'});
            shadow.innerHTML = `
               <style>
                  h2 {
                     text-align: center;
                     font-family: 'Arial'
                  }
               </style>
               <h2>${this.caption}</h2>
            `;

         }
      }
      customElements.define("my-dialog", MyDialog);
   </script>
   <style>
      my-dialog {
         box-shadow: 2px 2px 2px gray;
         border:1px solid red;
      }
   </style>
</head>

<body>
   <h2>Custom Elements</h2>

      <my-dialog caption="My Dialog Ready"></my-dialog>
```

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

Day 59 : Web Components and JQuery – 16/05/23

Web Components | Custom Elements

Phases of Component
- Component is derived from "HtmlElement"
- Component is created using
    constructor()
- Component constructor must call super constructor
- Component properties are defined in constructor
- Component properties are accessed by using

    static get observerdAttributes() { }

- Component properties are changed by using

    attributesChangedCallback() { }

- Component is created and added to page

    connectedCallback() { }

- Component can attach shadow member

    attachShadow() { }

- Component must be registered

    customComponent.define("tagName" , className)

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script>
      class Login extends HTMLElement
      {
         constructor(){
            super();
            this.logintitle = "Some Title";
         }
         static get observedAttributes(){
```

```
            return ['logintitle'];
        }
        attributeChangedCallback(property, oldValue, newValue){
            if(oldValue===newValue) {
                return;
            }
            this [property] = newValue;
        }
        connectedCallback(){
            var shadow = this.attachShadow({mode:"open"});
            shadow.innerHTML = `
                <dl>
                    <h3>${this.logintitle}</h3>
                    <dt>User Name</dt>
                    <dd><input type="text"></dd>
                    <dt>Password</dt>
                    <dd><input type="password"></dd>
                </dl>
                <button>Login</button>
                <button>Cancel</button>
            `;
        }
    }
    customElements.define("my-login", Login);
  </script>
</head>
<body>
  <my-login logintitle="Admin Login"></my-login>
  <hr>
  <my-login logintitle="User Login"></my-login>
</body>
</html>
```

jQuery

- It is a JavaScript library for building UI.
- Library provides pre-defined functions and components.
- jQuery introduced in 2006 by John Resig
- "Write Less - Do More"

Install jQuery for Project

```
>npm install jquery --save
```

Link jQuery to your Page

```
<script src="../node_modules/jquery/dist/jquery.js"> </script>
```

Load jQuery Library:

```
 <script>
   $(function(){
```

```
        ... your jquery code...
    })
</script>
              (or)
<script>
   $(document).ready(function(){

    });
 </script>
```

- jQuery can select HTML elements by using CSS selectors

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script src="../node_modules/jquery/dist/jquery.js"></script>
   <script>
      $(function(){
         $("h1").text("jQuery");
         $("#subtitle").html(`<b><i>Write Less - Do More</i></b>`);
         $(".txt").html("jQuery is a JavaScript library for building effective UI");
      })
   </script>
</head>
<body>
   <h1></h1>
   <p id="subtitle"></p>
   <div class="txt"></div>
</body>
</html>
```

jQuery DOM Methods

```
      html()          innerHTML
      text()          innerText
      val()           value
      attr()          attribute
      prop()          property
      append()
      appendTo()
      prepend()
      prependTo()
      before()
      after()
      css()
      $.each()
```

jQuery DOM Events

all JavaScript events are same

You have to use event listeners

```
<button> Insert </button>

$("button").click(function(event){
    event.clientX
    event.keyCode
    event.target.id
    event.target.name
    event.target.className
})
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script>
        $(function(){
            $("button").click(()=>{
                $("p").html(`Hello ! ${$("#UserName").val()}`);
            })
        })
    </script>
</head>
<body>
   Your Name : <input type="text" id="UserName"> <button>Submit</button>
   <p></p>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script>
        var categories = ["Electronics", "Footwear", "Fashion"];
        $(function(){
```

```
            categories.map((value)=>{
                $(`<li>${value}</li>`).appendTo("ol");
                $(`<option>${value}</option>`).appendTo("select");
            })
        })
    </script>
</head>
<body>
    <ol></ol>
    <select></select>
</body>
</html>

Ex:
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script>

        $(function(){
            $.ajax({
                method: "get",
                url: "http://fakestoreapi.com/products&quot;,
                success: (data)=>{
                    data.map((item)=>{
                        $(`<tr>
                            <td>${item.title}</td>
                            <td><img src=${item.image} width="100" height="100"></td>
                            <td>${item.price}</td>
                           </tr>`).appendTo("tbody");
                    })
                }
            })
        })
    </script>
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.css">
</head>
<body class="container-fluid">
    <table class="table table-hover">
        <thead>
            <tr>
                <th>Title</th>
                <th>Image</th>
                <th>Price</th>
            </tr>
        </thead>
```

```
        <tbody>

        </tbody>
    </table>
</body>
</html>
```

jQuery UI

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="../node_modules/jquery-ui/jquery-ui.css">
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script src="../node_modules/jquery-ui/jquery-ui.js"></script>
    <script>
        $(function(){
            $("#dept").datepicker();
        })
    </script>
</head>
<body>
    Departure
    <input type="text" id="dept">
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="../node_modules/jquery-ui/jquery-ui.css">
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script src="../node_modules/jquery-ui/jquery-ui.js"></script>
    <script>
        $(function(){
            $("#faqs").accordion();
        })
    </script>
</head>
<body>
  <div id="faqs">
    <h2>What is Netflix?</h2>
```

```
      <div>
        <p>something about netflix</p>
      </div>
      <h2>How to access Netflix?</h2>
      <div>
        <p>something..</p>
      </div>
    </div>
</body>
</html>
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="../node_modules/jquery-ui/jquery-ui.css">
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script src="../node_modules/jquery-ui/jquery-ui.js"></script>
    <script>
        $(function(){
            $("ol").sortable();
        })
    </script>
</head>
<body>
    <ol>
      <li>JavaScript</li>
      <li>CSS</li>
      <li>Bootstrap</li>
      <li>HTML</li>
    </ol>
</body>
</html>
```

Day 60 : JS DSA Link – 20/05/23

https://drive.google.com/file/d/1O4qlxp9l5ULpxcJuqnkQZvFWPHt3TQtw/view?usp=drive_web&authuser=0