RESEARCH INNOVATION DISCOVERY (RID TECH)
A place for coder, developer & Researcher
RUN BY TWKSAA WELFARE FOUNDATION & ISO: 9001:2015

Reg.No: 048884
Foundation Day
05-09-2024

JavaScript Programming language E-Book

Er. Rajesh Prasad(B.E, M.E)
Founder: RID Organization

- **RIDORGANIZATION**यानि **Research, InnovationandDiscovery**संस्था जिसका मुख्य उदेश्य हैं आने वाले समय में सबसे पहले **NEW(RID, PMS & TLR)**की खोज, प्रकाशन एवं उपयोग भारतकी इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो |
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW(RID, PMS & TLR)**के माध्यम से किया जाये इसके लिए ही मैं राजेश प्रसाद **इसRIDसंस्था** की स्थपना किया हूँ |
- Research, Innovation&Discovery में रूचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियो से मैं आवाहनं करता हूँ की आप सभी **इसRIDसंस्था**से जुड़ें एवं अपने बुध्दि, विवेक एवं प्रतिभा से दुनियां को कुछ नई **(RID, PMS &TLR)**कीखोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें |

"त्वक्सा JavaScript प्रोग्रामिंग लैंग्वेज के इस ई-पुस्तक में आप JavaScript से जुड़ी सभी बुनियादी अवधारणाएँ सीखेंगे। । मुझेआशाहैकिइसई-पुस्तककोपढ़नेकेबादआपकेज्ञानमेंवृद्धिहोगीऔरआपकोकंप्यूटरविज्ञानकेबारेमेंऔरअधिकजाननेमेंरुचिहोगी"

"In this E-Book of JavaScript Programming language you will learn all the basic concepts related to JavaScript. I hope after reading this E-Book your knowledge will be improve and you will get more interest to know more thing about computer Science".

RIDहमें क्यों करना चाहिए?

(Research)

अनुसंधान हमें क्यों करना चाहिए ?

Why should we do research?

1. नई ज्ञान की प्राप्ति(Acquisition of new knowledge)
2. समस्याओं का समाधान(To Solving problems)
3. सामाजिक प्रगति (ToSocial progress)
4. विकास को बढ़ावा देने(To promote development)
5. तकनीकी और व्यापार में उन्नति(Toadvances in technology & business)
6. देश विज्ञान और प्रौद्योगिकी के विकास(To develop the country's science & technology)

(Innovation)

नवीनीकरणहमें क्यों करना चाहिए ?

Why should we do Innovation?

1. प्रगति के लिए(To progress)
2. परिवर्तन के लिए(For change)
3. उत्पादन में सुधार(To Improvement in production)
4. समाज को लाभ(ToBenefit to society)
5. प्रतिस्पर्धा में अग्रणी (To be ahead of competition)
6. देश विज्ञान और प्रौद्योगिकी के विकास।
(To develop the country's science & technology)

(Discovery)

खोजहमें क्यों करना चाहिए?

Why should we doDiscovery?

1. नए ज्ञान की प्राप्ति(Acquisition of new knowledge)
2. अविष्कारों की खोज(ToDiscovery of inventions)
3. समस्याओं का समाधान(To Solving problems)
4. ज्ञान के विकास में योगदान(Contribution to development of knowledge)
5. समाज के उन्नति के लिए (for progress of society)
6. देश विज्ञान और तकनीक के विकास
(To develop the country's science & technology)

❖ Research(अनुसंधान):

- अनुसंधानएक प्रणालीकरण कार्य होता है जिसमें विशेष विषय या विषय की नई ज्ञान एवं समझ को प्राप्त करने के लिए सिद्धांतिक जांच और अध्ययन किया जाता है। इसकी प्रक्रिया में डेटा का संग्रह और विश्लेषण, निष्कर्ष निकालना और विशेष क्षेत्र में मौजूदा ज्ञान में योगदान किया जाता है। अनुसंधान के माध्यम से विज्ञान, प्रोधोगिकी, चिकित्सा, सामाजिक विज्ञान, मानविकी, और अन्य क्षेत्रों में विकास किया जाता है। अनुसंधान की प्रक्रिया में अनुसंधान प्रश्न या कल्पनाएँ तैयार की जाती हैं, एक अनुसंधान योजना डिज़ाइन की जाती है, डेटा का संग्रह किया जाता है, विश्लेषण किया जाता है, निष्कर्ष निकाला जाता है और परिणामों को उचित दर्शाने के लिए समाप्ति तक पहुंचाया जाता है।

❖ Innovation(नवीनीकरण): -

- Innovation एक विशेषता या नई विचारधारा की उत्पत्ति या नवीनीकरण है। यह नए और आधुनिक विचारों, तकनीकों, उत्पादों, प्रक्रियाओं, सेवाओं या संगठनात्मक ढंगों का सृजन करने की प्रक्रिया है जिससे समस्याओं का समाधान, प्रतिस्पर्धा में अग्रणी होने, और उपयोगकर्ताओं के अनुकूलता में सुधार किया जा सकता है।

❖ Discovery (आविष्कार):

- Discovery का अर्थ होता है "खोज" या "आविष्कार"। यह एक विशेषता है जो किसी नए ज्ञान, अविष्कार, या तत्व की खोज करने की प्रक्रिया को संदर्भित करता है। खोज विज्ञान, इतिहास, भूगोल, तकनीक, या किसी अन्य क्षेत्र में हो सकती है। इस प्रक्रिया में, व्यक्ति या समूह नए और अज्ञात ज्ञान को खोजकर समझने का प्रयास करते हैं और इससे मानव सभ्यता और विज्ञान-तकनीकी के विकास में योगदान देते हैं।

नोट : अनुसंधान विशेषता या विषय पर नई ज्ञान के प्राप्ति के लिए सिस्टमैटिक अध्ययन है, जबकि आविष्कार नए और अज्ञात ज्ञान की खोज है।

सुविचार:

1.	**समस्याओं का समाधान करने का उतम मार्ग हैं|→**शिक्षा ,RID,प्रतिभा, सहयोग, एकता एवं समाजिककार्य-
2.	**एक इंसान के लिए जरूरी हैं |→**रोटी, कपड़ामकान, शिक्षा, रोजगार, इज्जत और सम्मान ,
3.	**एक देश के लिए जरूरी हैं** -|→संस्कृतिसभ्यता, भाषा, एकता-, आजादीसंविधान एवं अखंडता,
4.	**सफलता पाने के लिए होना चाहिए |→**लक्ष्यत्याग, इच्छा-शक्ति, प्रतिबद्धता ,, प्रतिभा एवं ,सतता
5.	**मरने के बाद इंसान छोडकर जाता हैं |→**शरीरअन-धन, घर-परिवार, नाम, कर्म एवं विचार ,
6.	**मरने के बाद इंसान को इस धरती पर याद किया जाता हैं उनके**

➔ नाम काम, दान, विचार, सेवा-समपर्ण एवं कर्मो से ,...

आशीर्वाद (बड़े भैया जी)

Mr. RAMASHANKAR KUMAR

मार्गदर्शन एवं सहयोग

Mr. GAUTAM KUMAR

...सोच है जिनकी नई कुछकर दिखाने की, खोज है रीड संस्था को उन सभी इंसानों की...

❖ JavaScript

- JavaScript is a high-level, lightweight, and dynamic programming language.
- It is mainly used to make web pages **interactive and dynamic**.
- Runs in the browser (client-side) and can modify **HTML and CSS**.
- It is **platform-independent** and **object-oriented**.
- JavaScript code is written inside <script> … </script> tags in HTML.

❖ Features of JavaScript

- **High-Level:** Easy to read and understand.
- **Interpreted:** No need for compilation.
- **Dynamic Typing:** Variable types can change at runtime.
- **Weakly Typed:** Automatically converts data types.
- **Client-Side Scripting:** Runs directly in the browser.
- **Cross-Platform:** Works on all browsers and systems.
- **Object-Oriented:** Uses objects and methods.
- **Functions:** Functions can be stored, passed & returned.
- **Asynchronous:** Handles multiple tasks (AJAX, events).
- **Modular:** Supports modules for organized code.
- **Extensible:** Can use libraries and frameworks.
- **Community:** Huge developer support and resources.
- **Scalable:** Used for both frontend and backend (Node.js).

❖ History of JavaScript

- **1990s:** Tim Berners-Lee introduced the web and HTML.
- **1993:** Mosaic browser launched.
- **1994:** Netscape founded by Marc Andreessen.
- **1995:** Brendan Eich created JavaScript at Netscape.
- Initially called **Mocha**, then **LiveScript**, and finally **JavaScript** in **December 1995**.

❖ Applications of JavaScript

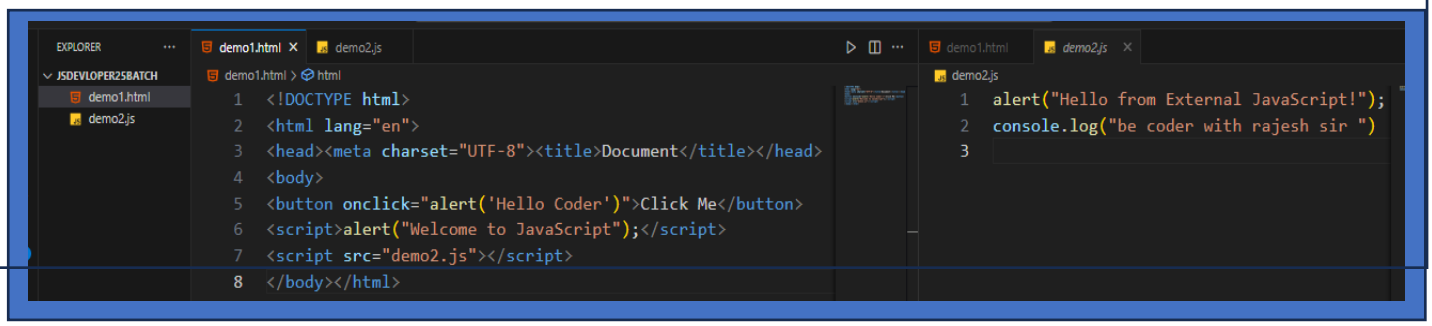
- **Web Development:** Create dynamic web pages.
- **Frontend:** Used with React, Angular, Vue.js.
- **Backend:** With Node.js for servers.
- **Mobile Apps:** Using React Native.
- **Games:** Using HTML5 and Phaser.
- **Web APIs:** Fetch data from external sources.
- **Animations:** With HTML5 Canvas and CSS.
- **Browser Extensions:** For Chrome, Firefox, etc.
- **Data Visualization:** Using D3.js, Chart.js.
- **Real-Time Apps:** Chats, games, collaboration tools.
- **Desktop Apps:** With Electron, NW.js.
- **IoT:** For dashboards and control panels.
- **Automation:** Web scraping, testing (Puppeteer).
- **Machine Learning:** TensorFlow.js for ML in browsers.

❖ Useful extensions for js

- **ESLint:** Checks syntax and code style.
- **Prettier:** Automatically formats code.
- **ES6 Snippets:** Adds quick JavaScript code snippets.
- **Debugger for Chrome:** Debugs code in Chrome browser.
- **Path Intellisense:** Auto-completes file paths.
- **Live Server:** Runs projects with live reloading.
- **GitLens:** Shows Git history and changes.
- **Jest:** Runs and tests code instantly.
- **Code Spell Checker:** Finds spelling mistakes in code.
- **npm Intellisense:** Auto-completes npm package names.

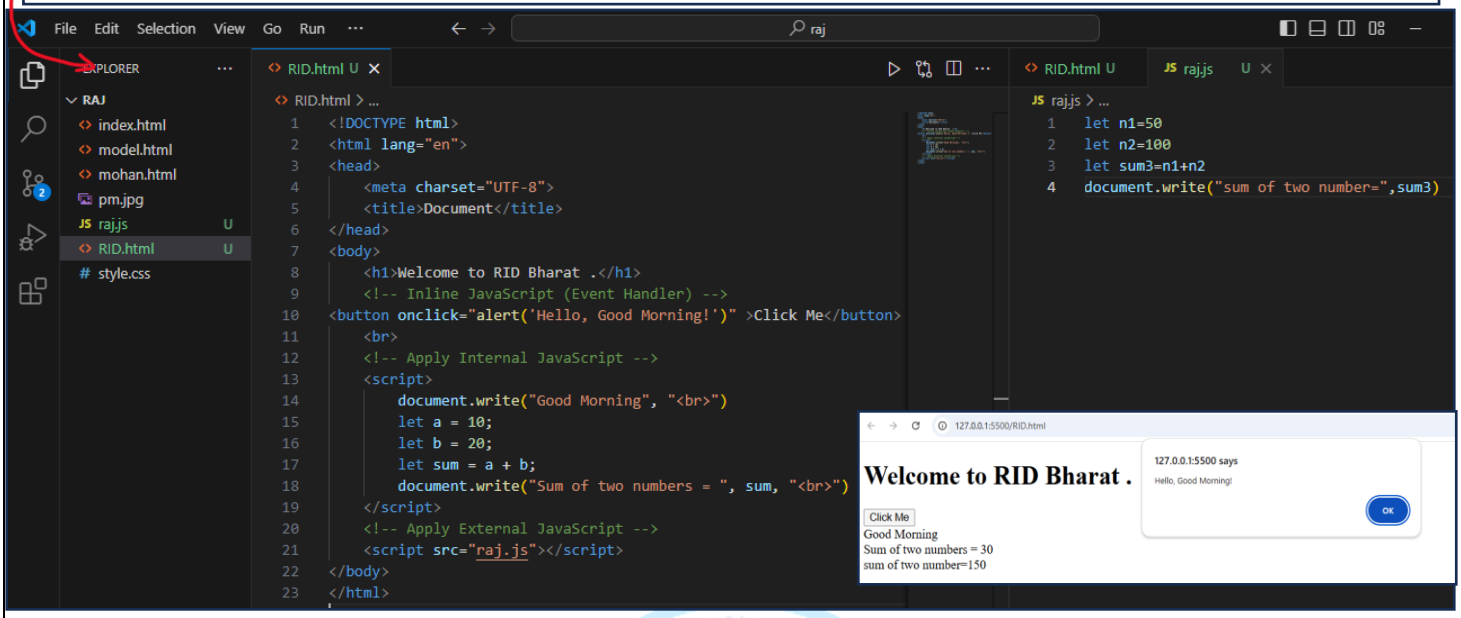
❖ How we can write the JS code in HTML?

1. **Inline JS:** Inside an HTML tag **Example:** <button onclick="alert(RID Bharat!')">Click Me</button>
2. **Internal JS:** Inside <script> tags in HTML
   <script> alert("Welcome to JavaScript!") </script>
3. **External JS:** In a separate .js file and link it **Example:** <script src="demo1.js"></script>
   Example: demo2.js alert("Hello from External JavaScript!");
   console.log("be coder with rajesh sir ")



❖ How many way to Run JavaScript Code?

1. **Browser Console:** Open browser → Press **F12** → Go to **Console** → Type JS code → Press **Enter**.
2. **HTML File:** Write JS inside <script> tags → Save → Open the HTML file in a browser.
3. **Node.js:** Save code in a .js file → Run using node filename.js in the terminal.



How to run the JS code using the Node.js?

Node.js is a JavaScript runtime environment used to run JavaScript code outside the browser.

Step 1: Install Node.js → Go to nodejs.org → Download **LTS version** → Install it.

Step 2: Set Path → **For Windows:**

1. Right-click **This PC → Properties → Advanced System Settings → Edit Environment Variables**
2. Under **System Variables**, find **Path → Edit → New**
3. Add the Node.js installation path, e.g.:
4. C:\Program Files\nodejs\
5. Click **OK** to save.

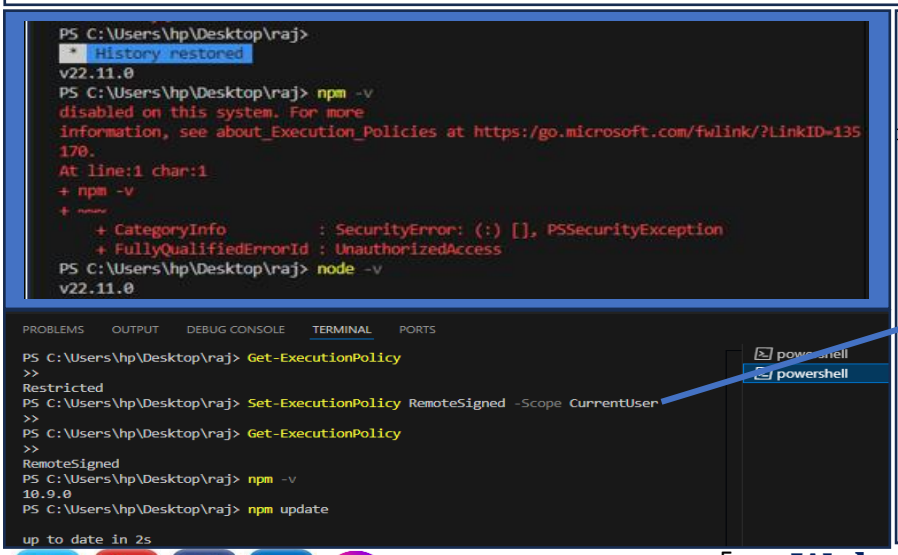
Step 3: Verify Installation → Open Command Prompt & type: **>>>** node -v Enter then check npm **> >>** npm -v

Step 4: Create js file with .js Extention and Run File → Create a file **raj.js**:

Example:- console.log("Welcome To, RID Bharat Org.");

➔ Open Terminal & this command : node raj.js **Output:** Welcome To, RID Bharat Org.

If you are unable to run or view the npm install command properly, please follow the steps below.



Fix PowerShell Script Error:

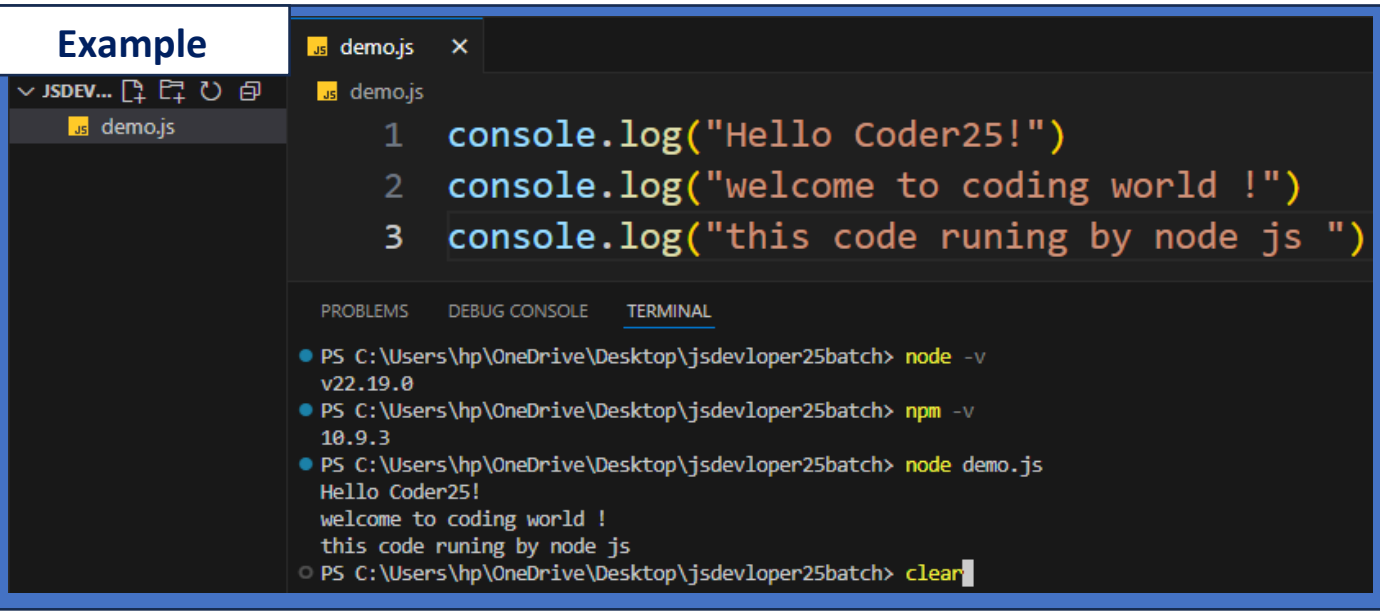
1. **Open PowerShell as Admin:**
   Press **Win + X → Windows PowerShell (Admin)**.
2. **Check Policy:**
3. Get-ExecutionPolicy

(If it shows *Restricted*, scripts are blocked.)

4. **For Allow run this comand:**
   Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

➔ Now running your npm command.

Example



OUTPUT STATEMENT IN JS

We can display data in JavaScript in different ways:

1) **console.log() –** Shows output in the browser console.
2) **innerHTML –** Displays data inside an HTML element.
3) **document.write() –** Writes directly to the webpage.
4) **alert() –** Shows output in a popup box.
5) **.value –** Displays output inside input fields.
6) **innerText –** Displays plain text in an element.
7) **confirm() –** It shows a message with **OK** and **Cancel** buttons. It works as both input and output, returning **true** for OK and **false** for Cancel.

Console Methods:

- console.log() → Normal message
- console.info() → Info message
- console.warn() → Warning message
- console.error() → Error message

raj.html > html > body > input#d3

```html
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Output</title>
</head>
<body>
  <h2>JavaScript Output Methods</h2>
  <p id="d1"></p>
  <p id="d2"></p>
  <input type="text" id="d3" value="">
  <script>
    console.log("Hello Coder ");                                  //console.log()
    document.getElementById("d1").innerHTML = "<b>Welcome</b>";   //.innerHTML
    document.getElementById("d2").innerText = "<b>Welcome</b>";   //.innerText
    document.write("welcome to Home page ");                      //document.write()
    alert("Hello! This is an alert for you.");                    //alert()
    document.getElementById("d3").value = "my name is Sangam";    //.value
    confirm("Do you want to continue?");                          //.confirm()
   let res= confirm("Do you want to continue?");                  // both input & output
   console.log(res)
  </script>
</body></html>
```

Difference between alert() and window.alert():

- No difference; both show the same alert box.
- alert() is a shortcut for window.alert() since window is the global object in browsers.

Example: alert("coder"); // same as →window.alert("code");

Difference between .innerHTML & . innerText:

- **innerHTML** → Displays HTML tags as formatted content (**bold, italic**, etc.).
- **innerText** → Displays text only; HTML tags are shown as plain text.

- ✓ we identify people by their **ID card, name, or group etc.**
- ✓ JavaScript identifies webpage elements by **id, class, tag, or selctor**.

Example:

- **getElementById()** → Like identifying a person by **Aadhaar ID** (unique).
- **getElementsByClassName()** → Like finding all **students in the same class**.
- **getElementsByTagName()** → Like finding all **books of the same type**.
- **getElementsByName()** → Like calling all people with the same **name**.
- **querySelector()** → Like selecting the **first person** who matches a description.
- **querySelectorAll()** → Like selecting **everyone** who matches that description.

There are following js method are used for the access the HTML elements

1. **getElementById()**: Gets a single element by its unique id.
2. **getElementsByClassName():** Gets a live HTMLCollection of elements by class name.
3. **getElementsByTagName()**: Gets a live HTMLCollection of elements by tag name.
4. **getElementsByName()** method is used to select **all elements** in the DOM that have a specified name attribute.
5. **querySelector()**: Gets the first element that matches a CSS selector.
6. **querySelectorAll()**: Gets a NodeList of all elements that match a CSS selector.

Example: Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Document</title>
</head>
<body>
   <h1 class="c c1"> </h1>
   <h2 class="c c2"> </h2>
   <p>hi</p>
   <p>hello</p>
   <pre>...</pre>
   <div name="value"></div>
   <div name="value"></div>
   <h3 class="cc1">...</h3>
   <h3 id="dd1"></h3>
   <p class="qll">...</p>
   <p class="qll">...</p>
   <p class="qll">...</p>
   <p id="Res"></p>
   <script src="demo.js"></script>
</body> </html>
```

Example: demo.js

```
document.getElementsByClassName("c")[0].innerHTML="RID"
document.getElementsByClassName("c")[1].innerHTML="Research"
document.getElementsByClassName("c1")[0].innerHTML="Innovation"
document.getElementsByTagName("p")[0].innerHTML="Discovery"
document.getElementsByTagName("p")[1].innerHTML="TIP"
document.getElementsByName("value")[0].innerHTML="Training"
document.getElementsByName("value")[1].innerHTML="Internship"
document.querySelector(".cc1").innerHTML="Placement"
document.querySelector("#dd1").innerHTML="Workshop"
document.querySelectorAll(".qll")[0].innerHTML="LED Based Skills Center"
document.querySelectorAll(".qll")[1].innerHTML="Learning Earning
Development"
let a = 20
let b = 19
let c = a + b
document.getElementById("Res").innerHTML="sum= "+c
```

OutPut

RID
Research
Innovation
Discovery
TIP
. . .
Training
Internship
Placement
Workshop
LED Based Skills Center
Learning Earning Development
...
sum= 39

Note:[0]: This index is used to select the first (and possibly only) element in that HTMLCollection.

INPUT STATEMENT IN JS

We can take input from users in JavaScript in different ways:

1. **prompt() –** Displays a dialog box to take input from the user.
2. **HTML Form Elements –** Takes user input through form fields like text boxes and buttons.
3. **Event Listeners** – Captures user actions like typing, clicking, or submitting.
4. **Inline Event Handlers** – Takes input directly from HTML tags using events like oninput or onclick.
5. **confirm()** – Used as both input and output; returns **true** if user clicks OK and **false** if Cancel.
6. **.innerText:-** get the data from the normal tag like <h1> <p> <pre> etc.
   Example: <h1 id="d1">This is js class </h1> → html code
   let data=document.getElementById("d1").innerText → Js code

1. **prompt()** – Takes user input in **string format**, so you need to **convert it** into another data type if needed.

Example-1:

raj.html

```
<!DOCTYPE html><html lang="en">
<head><meta charset="UTF-8">
</head>
<body>
    <p id="d1">....</p>
<script>
 let num1=prompt("Enter the number-1")
 let num2=prompt("Enter the number-2")
let sum=num1+num2
document.getElementById("d1").innerHTML="sum of two no="+sum
console.log("sum of two no=",sum)
</script>
</body></html>
```

Type Casting

```
let num1=Number(prompt("Enter the number-1"))
 console.log(typeof(num1))
let num2=Number(prompt("Enter the number-2"))
let sum=num1+num2
```

❖ Using prompt-sync in Node.js

- **prompt()** doesn't work directly in Node.js — use the **prompt-sync** library instead.
  - **Install:** npm install prompt-sync

Example 1: Add Two Numbers

```
const prompt = require('prompt-sync')();
let num1 = parseInt(prompt("Enter Number-1: "));
let num2 = parseInt(prompt("Enter Number-2: "));
let sum = num1 + num2;
console.log("Sum =", sum);
```

Output:

```
Enter Number-1: 10
Enter Number-2: 20
Sum = 30
```

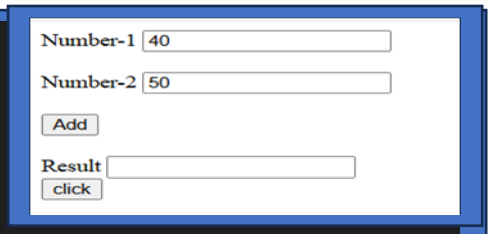
2.HTML Form Elements

Used to take user input through fields like **text boxes**, **buttons**, **checkboxes & dropdowns**.

Syntax:

```
<form>
  <!-- Text Box -->
  <input type="text" placeholder="Enter Name">
  <!-- Checkbox -->
  <input type="checkbox"> Option
  <!-- Dropdown -->
  <select>
    <option>Option 1</option>
    <option>Option 2</option>
  </select>
  <!-- Button -->
  <button type="submit">Submit</button>
</form>
```

Example-1:




```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Form Input Example</title>
</head>
<body>
    <form>
        Number 1: <input type="text" id="num-1"><br><br>

        Number 2: <input type="text" id="num-2"><br><br>

        <button type="button" onclick="add()">Add</button><br><br>
        Result: <input type="text" id="res" readonly>
    </form>
    <br>
    <button type="button" onclick="alert('Hello Everyone!')">Click</button>
    <script>
        function add() {
            let a =Number( document.getElementById("num-1").value);
            let b =Number( document.getElementById("num-2").value);
            let sum = a + b;
            document.getElementById("res").value = sum;
            document.getElementById("num-1").value = ""; // remove the input box-1
            document.getElementById("num-2").value = ""; // remove the input box-2
        }
    </script></body></html>
```

Example-2:

raj.html > html > body > script > showData

```
1   <!DOCTYPE html>
2   <html>
3   <body>
4     <form>
5       Name: <input type="text" id="d1"><br><br> <!-- Text Box -->
6       <input type="checkbox" id="d2"> Subscribe channel <br><br> <!-- Chec
7       Gender: <!-- Radio Buttons -->
8       <input type="radio" id="male" name="gender" value="Male" checked> Ma
9       <input type="radio" id="female" name="gender" value="Female"> Female <br><br>
10      Select City: <!-- Dropdown -->
11      <select id="city">
12        <option>Delhi</option>
13        <option>Mumbai</option>
14        <option>Bhopal</option>
15      </select><br><br>
16      <button type="button" onclick="showData()">Submit</button> <!-- Button -->
17    </form>
18    <p id="res"></p>
19    <script>
20      function showData() {
21        let name = document.getElementById("d1").value;
22        let subscribe = document.getElementById("d2").checked;
23        let gender = document.querySelector('input[name="gender"]:checked').value;
24        let city = document.getElementById("city").value;
25        document.getElementById("res").innerHTML =
26          "Name: " + name +
27          "<br>Subscribed: " + subscribe +       // <br>Subscribed: " + (subscribe ? "Yes" : "No")
28          "<br>Gender: " + gender +
29          "<br>City: " + city;
30      }
31    </script></body></html>
```

Name: Sangam Kumar

Subscribe channel

Gender: Male Female

Select City: Mumbai

Submit

Name: Sangam Kumar
Subscribed: true
Gender: Male
City: Mumbai

- (subscribe ? "Yes" : "No") → this is a **ternary operator** (short form of an if-else statement).
- It checks the value of the variable subscribe.

How it works:

- If subscribe is **true** (checkbox is checked) → it returns "Yes"
- If subscribe is **false** (checkbox is unchecked) → it returns "No"

```
<!DOCTYPE html>
<html>
<body>
 <form>
  Name: <input type="text" id="d1"><br><br> <!-- Text Box -->
  <input type="checkbox" id="d2"> Subscribe channel <br><br> <!-- Checkbox -->
  Gender: <!-- Radio Buttons -->
  <input type="radio" id="male" name="gender" value="Male" checked> Male
  <input type="radio" id="female" name="gender" value="Female"> Female <br><br>
  Select City: <!-- Dropdown -->
  <select id="city">
   <option>Delhi</option>
   <option>Mumbai</option>
   <option>Bhopal</option>
  </select><br><br>
  <button type="button"
onclick="showData()">Submit</button>
 </form>
 <p id="res"></p>
```

```
<script>
 function showData() {
  let name = document.getElementById("d1").value;
  let subscribe = document.getElementById("d2").checked;
  let gender = document.querySelector('input[name="gender"]:checked').value;
  let city = document.getElementById("city").value;
  document.getElementById("res").innerHTML =
   "Name: " + name +
   "<br>Subscribed: " + subscribe +      // <br>Subscribed: " + (subscribe ? "Yes" : "No")
   "<br>Gender: " + gender +
   "<br>City: " + city;
 }
</script></body></html>
```

3.Using Event Listeners:

- Event listeners detect and respond to user actions like clicks, key presses, or form submissions.
- They help handle events such as **click, change, input, or submit**.

```
<!-- HTML -->
<button id="btn">Click Me</button>
<script>
 // JavaScript
 document.getElementById("btn").addEventListener("click", function() {
  alert("Button Clicked!");
 });
```

Example-1: form submission we are using Event listeners

demo1.html > html

```
<!DOCTYPE html>
<html>
<body>
  <form id="sub">
    User Name: <input id="d1" type="text"><br><br>
    Password: <input id="d2" type="text"><br><br>
    <button type="submit">Submit</button>
  </form>
  <h1 id="nameDisplay"></h1>
  <h2 id="passDisplay"></h2>
  <script>
    document.getElementById("sub").addEventListener("submit", function(event) {
      event.preventDefault(); // prevent form from submitting

      let name = document.getElementById("d1").value;
      let pass = document.getElementById("d2").value;

      document.getElementById("nameDisplay").innerText = name;
      document.getElementById("passDisplay").innerText = pass;

      // Clear input fields
      document.getElementById("d1").value = "";
      document.getElementById("d2").value = "";
    });
  </script>
</body>
</html>
```

User Name:

Password:

Submit

Sangam Kumar

123

Example-2:

```
<!DOCTYPE html>
<html>
<body>
  <h3>Press any Arrow Key (↑ ↓ ← →)</h3>
  <p id="result"></p>
  <script>
    document.addEventListener("keydown", function(event) {
      if (event.key === "ArrowUp") {
        document.getElementById("result").innerHTML = "You pressed: ↑ Up Arrow";
      }
      else if (event.key === "ArrowDown") {
        document.getElementById("result").innerHTML = "You pressed: ↓ Down Arrow";
      }
      else if (event.key === "ArrowLeft") {
        document.getElementById("result").innerHTML = "You pressed: ← Left Arrow";
      }
      else if (event.key === "ArrowRight") {
        document.getElementById("result").innerHTML = "You pressed: → Right Arrow";
      }
      else {
        document.getElementById("result").innerHTML = "Press an Arrow Key!";
      }
    });
  </script>
</body>
```

Press any Arrow Key (↑ ↓ ← →)

You pressed: ↑ Up Arrow

Keyboard Event in JavaScript

- keydown → an event that **triggers when any key is pressed** on the keyboard.
- event.key → identifies **which key** was pressed (for example: "ArrowUp", "ArrowDown", "a", "Enter").

How it works in your code:

1. document.addEventListener("keydown", function(event){ ... }) → Listens for any key press on the page.
2. event.key === "ArrowUp" → checks if the **Up Arrow key** is pressed.
3. Similar checks for "ArrowDown", "ArrowLeft", "ArrowRight". 4. Updates the <p> element to show which arrow key was pressed.

4. By Using Inline Event Handlers:

- Inline event handlers are written directly inside the HTML tag using event attributes like onclick, onmouseover, etc. The JavaScript code runs when the specified event happens.
  - **Syntax:** <element event="JavaScript code"></element>
  - **Example Syntax:** <button onclick="functionName()">Click Me</button>
    Or <button onclick="alert('Hello!')">Click Me</button>

Explanation: element → Any HTML tag (like <button>, <p>, , etc.) event → event attribute (like onclick, onmouseover, onchange, etc.) JavaScript code → Code that runs

5. confirm ():

- it is used to display a message box with **OK** &**Cancel** buttons.
- It returns **true** if user clicks **OK** & **false** if user clicks **Cancel**.

Syntax: confirm ("Your message here");

raj.html > ...

```
<!DOCTYPE html>
<html>
<body>
<h3>Example of confirm()</h3>
<button onclick="deleteFile()">Delete File</button>
<script>
  function deleteFile() {
    let result = confirm("Are you sure to delete file?");
    if (result) { alert("File deleted successfully!");}
    else { alert("File deletion canceled.");}
  }
</script></body></html>
```

raj.html > ...

```
<!DOCTYPE html>
<html>
<body>
<h3>Inline Event Handler Example</h3>
<button onclick="alert('Button Clicked!')">Click Me</button>
<button onclick="sayHi()">Say Hi</button>
<script>
  function sayHi() {alert("Hi, Welcome!");}
</script>
</body></html>
```

6.innerText:- get data from the normal tag like <h1> <p> <pre> etc.

Example: <h1 id="d1">This is js class </h1> → html code

let data=document.getElementById("d1").innerText → Js code

This is js class

Sanam Kumar

This is js class

```
<body>
    <h1 id="d1">This is js class </h1>
    <p id="d2">Sanam Kumar </p>
    <pre id="res"> </pre>
    <script>
        let data=document.getElementById("d1").innerText
        let name=document.getElementById("d2").innerText
        console.log(data); console.log(name)
        document.getElementById("res").innerText=data
    </script>
</body>
```

Example: cursor should go one input box to another input box through Enter bottom press from keyword

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Bootstrap demo</title>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.6.0/css/all.min.css"
integrity="sha512-
Kc323vGBEqzTmouAECnVceyQqyqdsSiqLQISBL29aUW4U/M7pSPA/gEUZQqv1cwx4OnYxTxve5UMg5GT6L4JJg=="
crossorigin="anonymous" referrerpolicy="no-referrer" />
  <style>
    button{
      width: 40px;
      height: 40px;
      font-size: 25px;
    }
  </style>
 </head>
 <body>
  <form>
    <label>Number-1</label>
    <input id="d1" type="text" placeholder="Enter the number-1" autocomplete="off"><br><br>
    <label>Number-2</label>
    <input id="d2" type="text" placeholder="Enter the number-2" autocomplete="off"><br><br>
    <button id="addbtm" type="button" onclick="add()"><i class="fa-solid fa-plus"></i></button>
    <button id="subbtm" type="button" onclick="sub()"><i class="fa-solid fa-minus"></i></button>
    <button id="mulbtm" type="button" onclick="mul()"><i class="fa-solid fa-xmark"></i></button>
    <button id="divbtm" type="button" onclick="div()"><i class="fa-solid fa-divide"></i></button>
    <button id="molbtm" type="button" onclick="mol()"><i class="fa-solid fa-percent"></i></button> <br><br>
    <label>Result</label>
    <input type="text" id="d3" >

    <script src="abhi.js"></script>
  </form>
 </body>
</html>
```

Number-1 Enter the number-1

Number-2 Enter the number-2

\+ — × ÷ %

Result

```
// Function to add two numbers
function add() {
   let a = document.getElementById("d1").value;
   let b = document.getElementById("d2").value;
   let sum = parseFloat(a) + parseFloat(b);
   document.getElementById("d3").value = sum;
   document.getElementById("d1").value = "";
   document.getElementById("d2").value = "";
}
```

```
// Function to subtract two numbers
function sub() {
   let n = document.getElementById("d1").value;
   let m = document.getElementById("d2").value;
   let sub = parseFloat(n) - parseFloat(m);
   document.getElementById("d3").value = sub;
   document.getElementById("d1").value = "";
   document.getElementById("d2").value = "";
}
```

```
// Function to divide two numbers
function div() {
   let x = document.querySelectorAll("#d1")[0].value;
   let y = document.querySelectorAll("#d2")[0].value;
   let quotient = parseFloat(x) / parseFloat(y);
   document.querySelector("#d3").value = quotient;
   document.getElementById("d1").value = "";
   document.getElementById("d2").value = "";
}
```

```
// Function to multiply two numbers
function mul() {
   let x = document.querySelectorAll("#d1")[0].value;
   let y = document.querySelectorAll("#d2")[0].value;
   let mul = parseFloat(x) * parseFloat(y);
   document.querySelector("#d3").value = mul;
   document.getElementById("d1").value = "";
   document.getElementById("d2").value = "";
}
```

```
// Event listener for cursor navigation between input fields and buttons
document.getElementById("d1").addEventListener("keypress", function (addmsg) {
   if (addmsg.key === "Enter") {
      addmsg.preventDefault();
      document.getElementById("d2").focus();
   }
});
```

```
// Function to calculate the remainder (modulo) of two numbers
function mol() {
   let x = document.querySelectorAll("#d1")[0].value;
   let y = document.querySelectorAll("#d2")[0].value;
   let rem = parseFloat(x) % parseFloat(y);
   document.querySelector("#d3").value = rem;
   document.getElementById("d1").value = "";
   document.getElementById("d2").value = "";
}
```

```
document.getElementById("d2").addEventListener("keypress", function (msg) {
   if (msg.key === "Enter") {
      msg.preventDefault();
      document.getElementById("addbtm").focus();
   }
});
// Keyboard navigation for buttons
document.getElementById("addbtm").addEventListener("keydown", function (msg1) {
   if (msg1.key === "ArrowRight") {
      msg1.preventDefault();
      document.getElementById("subbtm").focus();
   }
});
document.getElementById("subbtm").addEventListener("keydown", function (msg1) {
   if (msg1.key === "ArrowLeft") {
      msg1.preventDefault();
      document.getElementById("addbtm").focus();
   }
```

```
if (msg1.key === "ArrowRight") {
      msg1.preventDefault();
      document.getElementById("mulbtm").focus();
   }
});
document.getElementById("mulbtm").addEventListener("keydown", function (msg2) {
   if (msg2.key === "ArrowLeft") {
      msg2.preventDefault();
      document.getElementById("subbtm").focus();
   }
   if (msg2.key === "ArrowRight") {
      msg2.preventDefault();
      document.getElementById("divbtm").focus();
   }
});
document.getElementById("divbtm").addEventListener("keydown", function (sadi) {
   if (sadi.key === "ArrowLeft") {
      sadi.preventDefault();
      document.querySelector("#mulbtm").focus();
   }
   if (sadi.key === "ArrowRight") {
      sadi.preventDefault();
      document.getElementById("molbtm").focus();
   }
});
document.getElementById("molbtm").addEventListener("keydown", function (sadi) {
   if (sadi.key === "ArrowLeft") {
      sadi.preventDefault();
      document.getElementById("divbtm").focus();
   }
});

// Event listener to navigate between d1 and d2 using ArrowUp and ArrowDown
document.getElementById("d2").addEventListener("keyup", function (job) {
   if (job.key === "ArrowUp") {
      job.preventDefault();
      document.getElementById("d1").focus();
   }
});
document.getElementById("d1").addEventListener("keyup", function (job) {
   if (job.key === "ArrowDown") {
      job.preventDefault();
      document.getElementById("d2").focus();
   }
});
```

Identifiers

- All JavaScript variables must have **unique names** called **identifiers**.
- Identifiers are used to name **variables, functions, and objects**.

❖ Rules for Naming Identifiers:

1. Names **must begin with a letter**, underscore (_), or dollar sign ($).
2. Names **cannot begin with a number**.
3. Names can contain **letters, digits, underscores, and dollar signs**.
4. **No spaces** are allowed in the name.
5. **JavaScript keywords/reserved words** cannot be used as names (e.g., if, for, let).
6. Identifier names are **case-sensitive** (Name and name are different).
7. Use **meaningful names** to make your code easy to understand.

Note: Use only **letters (A–Z, a–z)**, **digits (0–9)**, **underscore _**, and **dollar sign $**, and **don't start with a number**.

Valid JavaScript Identifiers:

1. name
2. studentName
3. _age
4. $price
5. totalMarks
6. user_id
7. city1
8. myVariable
9. _result2025
10. $salaryAmount

Invalid JavaScript Identifiers:

1. 1name – Cannot start with a number
2. first name – Spaces not allowed
3. for – Reserved keyword
4. let – Reserved keyword
5. @value – Special character @ not allowed
6. student-name – Hyphen (-) not allowed
7. total% – % not allowed
8. 123abc – Cannot begin with a digit
9. class – Reserved keyword
10. name! – ! not allowed

Naming Conventions

1. **camelCase** – The first word starts with a lowercase letter, and each next word starts with a capital letter.
   Example: addOfTwoNumbers
2. **PascalCase** – Every word starts with a capital letter.
   Example: AddOfTwoNumbers
3. **snake_case** – All words are in lowercase and separated by underscores.
   Example: add_of_two_numbers

JavaScript Variable

- A **variable** is a name given to a **memory location** used to store data.

Attributes of a Variable:

1. **Name:** Unique name of the variable.
2. **Data Type:** Defines the type of value it can store (number, string, etc.).
3. **Value:** The actual data stored in the variable.
4. **Reference & Scope:** Defines where the variable can be used (local or global).

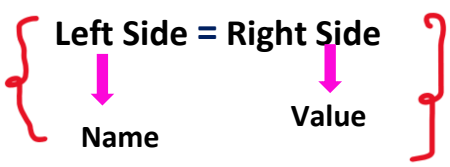
Example:

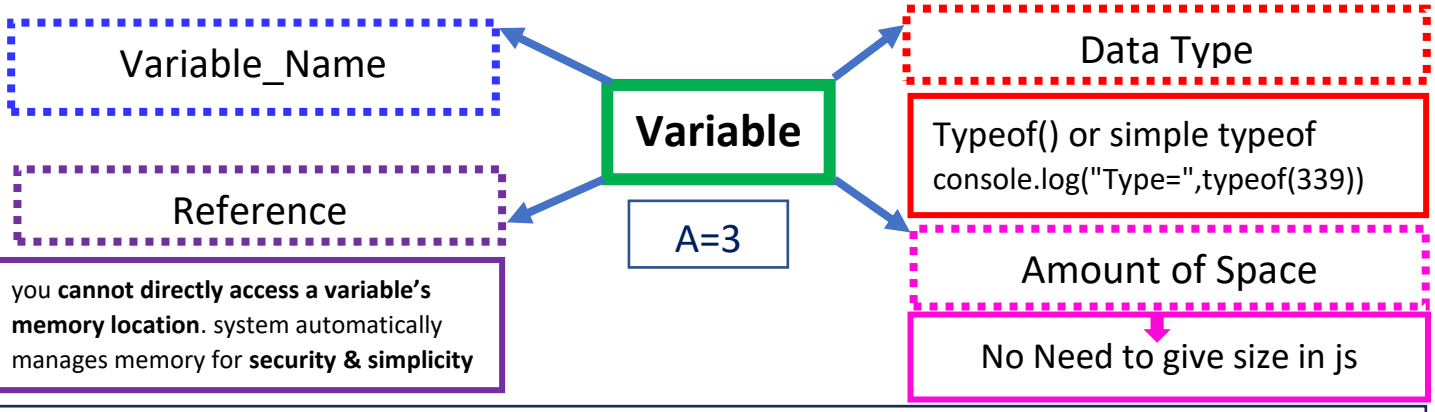
```
let age = 25;
```

Here:

- **age** → Name
- **number** → Data type
- **25** → Value
- **let** → Defines scope (block-level)

Syntax:





- **Scope** means the area or region in a program where a variable can be accessed or used.
- **block** or Area create in JavaScript by using **{ }** .inside {} we can write group of statements.

❖ Types of Variables:

1) Local Variable
2) Global Variable

1) Local Variable

- A local variable is declared inside a function or block.
- It can be used only within that function or block.

Syntax: without function with block{}

```
{
  let variableName = value;  // local variable
}
```

Example-1: without function

```
<script>
{
 let x = 10;   // local variable
 console.log("Local Variable =", x);
}
console.log(x); // Error: x is not defined (outside block)
</script>
```

2). Global Variable

- A **global variable** is declared **outside** any function or block.
- It can be **accessed from anywhere** in the program.

Syntax: without function with block {}

```
let variableName = value;   // global variable
{
  // can use global variable here
}
```

Example: without function.

```
<script>
        let y = 20;   // global variable
        console.log("Global Variable =", y);
        {
          console.log("Access inside block =", y);
        }
</script>
```

Syntax: with function

```
function functionName() {
  let variableName = value;  // local variable
}
```

Example-2: with function

```
<script>
        function test() {
          let x = 10;   // local variable
          console.log("Local variable value =", x);
        }
        test();
</script>
```

Syntax: with function

```
<script>
        let y = 20;   // global variable
        console.log("Global Variable =", y);
        {
          console.log("Access inside block =", y);
        }
</script>
```

Example: with function

```
<script>
        let y = 20;   // global variable
        function showValue() {
          console.log("Access inside function =", y);
        } console.log("Global Variable =", y);
        showValue();
</script>
```

❖ Variable Declaration

➢ **JavaScript Variables can be declared in 4 ways:**

1) **Automatically –** Declared without any keyword (becomes global).
2) **var –** Function-scoped and can be redeclared. (Used before ES6)
3) **let –** Block-scoped and can be reassigned. (Introduced in ES6 – 2015, used now)
4) **const –** Block-scoped and cannot be reassigned. (Introduced in ES6 – 2015, used now)

1) Automatically – Declared without any keyword (becomes global):

- If a variable is assigned a value without using var, let, or const, it becomes a global variable automatically.

Example-1

```
{
 x = 10;   // automatically global
 console.log("Inside block:", x);
}
console.log("Outside block:", x); // still accessible
```

Example-2:

```
x = 10;   // automatically global
console.log(x);  // Output: 10
```

2). var

- **Function-scoped:** Variables declared with var are accessible only within the function where they are declared. If declared outside any function, they become **global**.
- **Hoisting:** var variables are hoisted to the top of their scope and initialized with undefined.
- **Reassignable:** You can change (reassign) the value of a var variable.
- **Redeclarable:** The same variable name can be declared multiple times with var.

Syntax var variableName = value;

Example 1 – Redeclaration:

```
var a = 15;
console.log("Value of a =", a);
var a = 20;
console.log("Value of a =", a);
```

Output:

Value of a = 15
Value of a = 20

Example 2 – Block Scope:

```
var a = 15;
console.log("Outside block =", a);
{
 var a = 20;
 console.log("Inside block =", a);
}
console.log("After block =", a);
```

Example 3 – Declaration without value:

```
var fd;
fd = 25;
console.log(fd);  // 25
Example-4
var name:
name=”sangam kumar”
console.log(name) // sangam kumar
```

❖ Hoisting

- JavaScript में **hoisting** का मतलब है कि variables और functions को **execution से पहले memory में ऊपर (top) पर ले लिया जाता है** — यानी आप उन्हें declare करने से पहले use कर सकते हैं (लेकिन let और const के साथ नहीं)।

Example (with var)

```
console.log(a); // Output: undefined
var a = 10;
```

Explanation: JavaScript internally ऐसे मानती है:

```
var a;
console.log(a);
a = 10;
```

इसलिए output undefined आता है, error नहीं।

Example (with let)

```
console.log(b); //  ReferenceError
let b = 20;
```

Explanation: let और const hoist तो होते हैं, लेकिन **initialize नहीं होते**, इसलिए declare करने से पहले use करने पर error आता है।

Note: Hoisting का मतलब है JS में variable या function को कोड के ऊपर ले जाना, ताकि वो पहले से memory में reserved हो जाए।

❖ **Difference between undefined and ReferenceError:**

- **undefined** → The variable is declared but not assigned a value.
- **ReferenceError** → The variable is not declared at all (does not exist).

3). let

- **Block-scoped:** Variables declared with let are accessible only inside the block {} where they are defined.
- **Hoisting:** let variables are hoisted but **not initialized**, leading to a **ReferenceError** if accessed before declaration.
- **Reassignable:** You can change the value of a let variable.
- **Not Redeclarable:** You cannot redeclare a let variable in the same scope.

Syntax **let** variableName = value;

Example 1 – Redeclaration not allowed:

```
let a = 15;
console.log(a);
let a = 20;
```

Error: 'a' has already been declared

Output:

```
Inside block = 20
Outside block = 15
```

Example 2 – Same variable inside block:

```
let a = 15;
{
  let a = 20;
  console.log ("Inside block =", a);
}
console.log ("Outside block =", a);
```

Example 3 – Declaration without value:

```
let fd;  // allowed
fd = 50;
console.log(fd);
```

Output: 50

4). const

- **Block-scoped:** Like let, const variables are accessible only within the block {} they are declared in.
- **Hoisting:** const variables are hoisted but not initialized, causing a **ReferenceError** if accessed before declaration.
- **Immutability:** Once assigned, a const variable **cannot be reassigned**.
- **Must be initialized:** You must assign a value when declaring a const variable.

Syntax const variableName = value;

Example 1 – Redeclaration not allowed:

```
const a = 15;
const a = 20;
```

Error: 'a' has already been declared

Example 3 – Cannot be reassigned:

```
const fd = 30;
fd = 9;      // Error: Assignment to constant variable
```

Example 4 – Must be initialized:

```
const fd;   //  Error: Missing initializer in const declaration
```

Example 2 – Same variable name inside block:

```
const a = 15;
{
const a = 20;
console.log("Inside block =", a);
}
console.log("Outside block =", a);
```

Difference between var const and let

	var	let	const
Hoisting	Hosted at top of global scope. Can be used before the declaration.	Hosted at top of some private scope and only available after assigning value. Can not be used before the declaration.	Hosted at top of some private scope and only available after assigning value. Can not be used before the declaration.
Scope	Global scope normally. Start to end of the function inside of the function.	Block scoped always. Start to end of the current scope anywhere.	Block scoped always. Start to end of the current scope anywhere.
Redeclaration	Yes, we can redeclare it in the same scope.	No, we can't redeclare it in the same scope.	No, we can't redeclare or reinitialize it.

OPERATOR

- Operator is symbolic representation of that is used for perform the various operations.
- There is various type of operators

1. Basic Operators

Operator Type	Purpose	Symbols / Example
1. Arithmetic Operators	Perform math operations	+ , - , * , / , % , ++ , -- , **
2. Assignment Operators	Assign or update values	= , += , -= , *= , /= , %= , **=
3. Comparison Operators	Compare two values	== , === , != , !== , > , < , >= , <=
4. Logical Operators	Combine or invert conditions	&& , || , !
5. Ternary Operator	Shorthand for if–else	condition? true: false

2. Advanced Operators (ES6 and newer)

Operator Type	Purpose	Symbols / Example
1. Type Operator	Check data type	typeof x
2. String Operator	Join (concatenate) strings	"RID" + " Bharat"
3. Spread Operator	Expand array/object values	[...arr], {...obj}
4. Rest Operator	Collect remaining arguments in function	function sum(...nums){}
5. Nullish Coalescing Operator	Returns right value if left is null or undefined	x ?? y
6. Optional Chaining Operator	Safely access object properties	obj?.prop
7. Destructuring Assignment	Unpack values from arrays/objects	let [a,b] = arr , let {x,y} = obj

3. Special Operators (Used in specific cases)

Operator Type	Purpose	Symbols / Example
1. Comma Operator	Evaluate multiple expressions	(a = 1, b = 2)
2. Delete Operator	Remove a property from an object	delete obj.key
3. In Operator	Check if property exists in object	"name" in person
4. Instanceof Operator	Check if object is instance of class	x instanceof Array
5. New Operator	Create new instance of object/class	new Date()
6. Void Operator	Evaluate expression without returning value	void(0)
7. Bitwise Operators	Work on binary numbers	`& ,

❖ Ternary Operator (condition ? true : false)

Example 1: Check Age

```
let age = 18;
let result = (age >= 18) ? "You can vote" : "You cannot";
console.log(result);
```

Example 2: Check Even or Odd

```
let num = 7;
let result = (num % 2 === 0) ? "Even Num" : "Odd Num";
console.log(result);
```

❖ Arithmetic Operators: Arithmetic Operators are used to perform arithmetic operation:

Operator	Description	Example
➢ +	Addition	15+20 = 35
➢ -	Subtraction	50-25 = 25
➢ *	Multiplication	3*6 = 18
➢ /	Division	40/8 = 5
➢ %	Modulus (Remainder)	20%10 = 0
➢ ++	Increment	var a=20; a++; Now a = 21
➢ --	Decrement	var a=20; a--; Now a = 19

❖ Assignment Operators: Assignment operators assign values to JavaScript variables.

Operator	Description	Example
➢ =	Assign	10+10 = 20
➢ +=	Add and assign	var a=10; a+=20; Now a = 30
➢ -=	Subtract and assign	var a=20; a-=10; Now a = 10
➢ *=	Multiply and assign	var a=10; a*=20; Now a = 200
➢ /=	Divide and assign	var a=10; a/=2; Now a = 5
➢ %=	Modulus and assign	var a=10; a%=2; Now a = 0

❖ Comparison Operators or Relational:

Operator	Description	Example
➢ ==	Is equal to	10==20 = false
➢ ===	Identical (equal and same data type)	10==20 = false
➢ !=	Not equal to	10!=20 = true
➢ !==	Not Identical	20!==20 = false
➢ >	Greater than	20>10 = true
➢ >=	Greater than or equal to	20>=10 = true
➢ <	Less than	20<10 = false
➢ <=	Less than or equal to	20<=10 = false

❖ Logical Operators:

- The following operators are known as JavaScript logical operators.

Operator	Description	Example
➢ &&	Logical AND	(10==20 && 20==33) = false
➢ ||	Logical OR	(10==20 || 20==33) = false
➢ !	Logical Not	!(10==20) = true

❖ Bitwise Operators:

- bitwise operators perform bitwise operations on operands. bitwise operators are as follows:

Operator	Description	Decimal	Example	Same as	Result
1. &	AND	5 & 1	0101 & 0001	0001	1
2. |	OR	5 | 1	0101 | 0001	0101	5
3. ~	NOT	~ 5	~0101	1010	10
4. ^	XOR	5 ^ 1	0101 ^ 0001	0100	4
5. <<	left shift	5 << 1	0101 << 1	1010	10
6. >>	right shift	5 >> 1	0101 >> 1	0010	2
7. >>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

2.Advanced Operators

1. Type Operator – Used to check the data type of a variable. → **Symbol:** typeof

Syntax: typeof variable

Example: let name = "Raj";
console.log(typeof name); // string

2. String Operator – Used to join two or more strings. → **Symbol:** +

Syntax: str1 + str2

Example: let first = "Hello", last = "World";
console.log(first + " " + last); // Hello World

3. Spread Operator – Used to expand arrays or objects. → **Symbol:** ...

Syntax: [...array]

Example: let num = [1, 2, 3];
let newNum = [...num, 4, 5];
console.log(newNum); // [1, 2, 3, 4, 5]

4. Nullish Coalescing Operator – Returns right value if left is null or undefined. → **Symbol:** ??

Syntax: value = a ?? b;

Example: let name = null;
console.log(name ?? "Guest"); // Guest

5. Optional Chaining Operator – Safely accesses object properties. → **Symbol:** ?.

Syntax: object?.property

Example: let user = { name: "Raj" };
console.log(user?.age); // undefined (no error)

3. Special Operators (Used in specific cases)

1. Comma Operator (,):- Allows multiple expressions to be evaluated in a single statement; returns the last value.

Syntax: (expression1, expression2, expression3)

Example:
```
let result = (a = 1, b = 2, a + b);
console.log(result); // 3
```

Note: all expressions are evaluated, but only the last (a + b) is returned.

2. In Operator (in):- Checks if a property exists in an object or an index exists in an array.

Syntax: "property" in object

Example:
```
let person = { name: "Raj", age: 25 };
console.log("name" in person); // true
console.log("city" in person); // false
```

3. Nullish Coalescing Operator (??):- Returns right-hand value if left-hand value is null or undefined.

Syntax: value = left ?? right;

Example:
```
et username = null;
let user = username ?? "Guest";
console.log(user); // Guest
```

4. Optional Chaining Operator (?.) Safely accesses object properties without throwing an error if the property doesn't exist.

Syntax: object?.property

Example:
```
let user = { name: "Raj" };
console.log(user?.age); // undefined (no error)
```

Note: - Without optional chaining, accessing user.age on an undefined object would cause an error.

raj.html > html > body > script > signup

```
<!-- Project-2: Store the data in localStorage redirect the page after 1 second -->
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8"><title>Login Signup</title>
</head>
<body>
  <h1>Login Signup &  Create Account</h1>
User ID: <input id="uid" type="text"><br><br>
Password:<input id="upwd" type="password">
<button onclick="signup()">Create Account</button><br><br>
  <h2>Login</h2>
  User ID:<input id="lui" type="text"><br><br>
  Password:<input id="lupwd" type="password">
  <button onclick="login()">Login</button>
  <p id="message"></p>
  <script>
    function signup() {
      let userId = document.getElementById("uid").value;
      let userPwd = document.getElementById("upwd").value;
      localStorage.setItem("user_id", userId);
      localStorage.setItem("user_pwd", userPwd);
      document.getElementById("message").innerText = "Account created successfully!";
    }
    function login() {
      let enteredId = document.getElementById("lui").value;
      let enteredPwd = document.getElementById("lupwd").value;
      let storedId = localStorage.getItem("user_id");
      let storedPwd = localStorage.getItem("user_pwd");
      if (enteredId === storedId && enteredPwd === storedPwd) {
        document.getElementById("message").innerText = "Login successful! Redirecting...";
        // Redirect after 1 second
        setTimeout(() => {
          window.location.href = "ebook.html";
        }, 1000);}
       else { document.getElementById("message").innerText = "Invalid user ID or password.";} }
  </script></body></html>
```

&& Operator Project-1

ebook.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>E-BOOK Page</title>
</head>
<body>
<h1>Welcome to ebook Page!</h1>
</body>
</html>
```

Login Signup & Create Account

User ID: raj

Password: Create Account

Login

User ID: raj

Password: Login

Account created successfully!

JavaScript Redirection Methods

1. Redirect in the same tab (replace current page)

- window.location.href = "home.html";

Use for:

- Redirecting after login
- Navigating within your site

Note: Redirects browser to "home.html" in **same tab**.

2. Redirect in a new tab or window

- window.open("home.html", "_blank");

Use for:

- Opening a page without closing the current one

Note: Opens "home.html" in a **new tab** or **new window** (depends on browser settings).

3. Redirect to an external website

- window.location.href = "https://ridbharat.com";

Use for:

- Sending user to another site

4. Redirect to a specific folder or location

- window.location.href = "folder/page.html";

Examples:

window.location.href = "pages/dashboard.html";
window.location.href = "user/profile.html";

5. Redirect after a delay

```
setTimeout(() => {
   window.location.href = "home.html";
}, 2000); // Waits 2 seconds  1000 wait 1 second
```

Use for: Showing a success message before redirect

What is window.location.href = "home.html";?

- This line of JavaScript **redirects the user to another page** — in this case, to a file called home.html.

window

- Represents the **browser window** or **global object** in the browser.
- Everything that happens in the browser — like showing a page, opening a popup, or getting the URL — goes through window.

location

- A property of window.
- It represents the **current URL** (web address) shown in browser's address bar.
- You can **read** it to know the current page, or **set** it to go to a new page.

href

- Stands for **Hypertext REFerence** — basically, a URL.
- window.location.href is the **full URL** of the current page.
- If you **set it**, the browser **navigates** to the new page (like clicking a link).

	Action	Code Example	Opens in
1.	Same tab redirect	window.location.href = "home.html";	Same tab
2.	New tab redirect	window.open("home.html", "_blank");	New tab/window
3.	External site	window.location.href = "https://..."	Same tab
4.	Folder/file redirect	window.location.href = "path/file.html"	Same tab
5.	Delayed redirect	setTimeout(() => { window.location.href = "home.html"; }, 2000);	Same tab after delay

&& and || Operators based RTS Online Examination project-2

Home Exam Singup Login Service Product

Singup

user id:

password:

singup

Already have account Login

Login page

Enetr user id: raj

Enter password: 123

Login

Create new account Singup

4. Which planet is called the Red Planet?

A. Earth

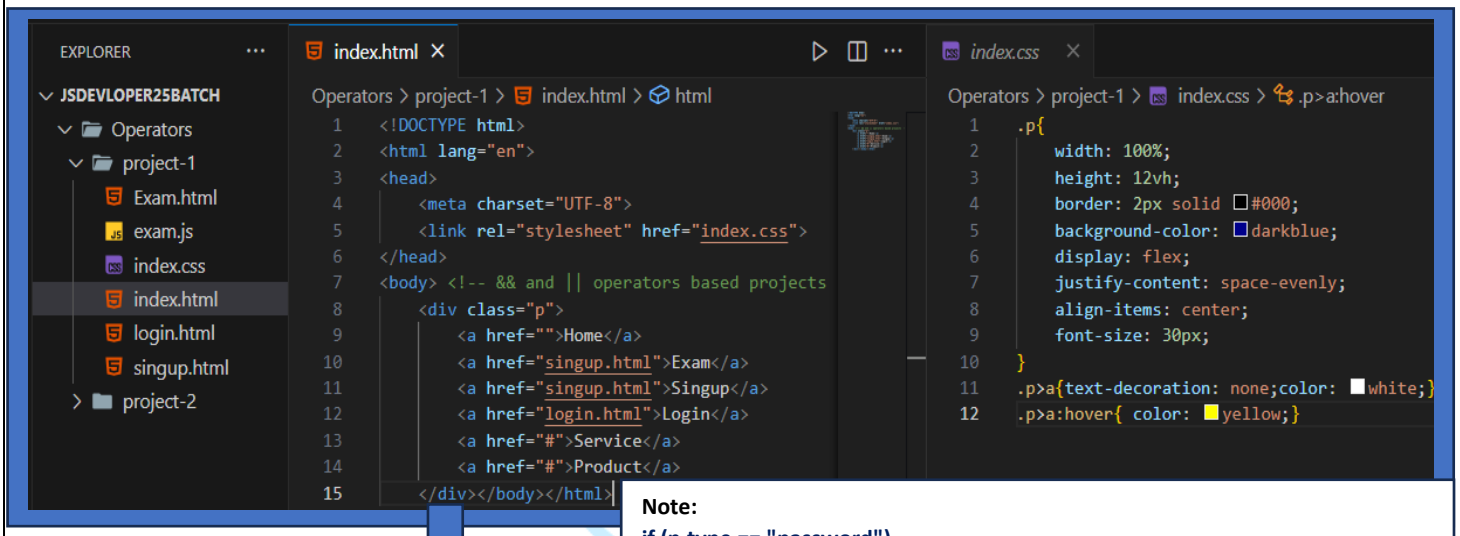
B. Jupiter

C. Mars

D. Venus

Ans:

Submit Check Result



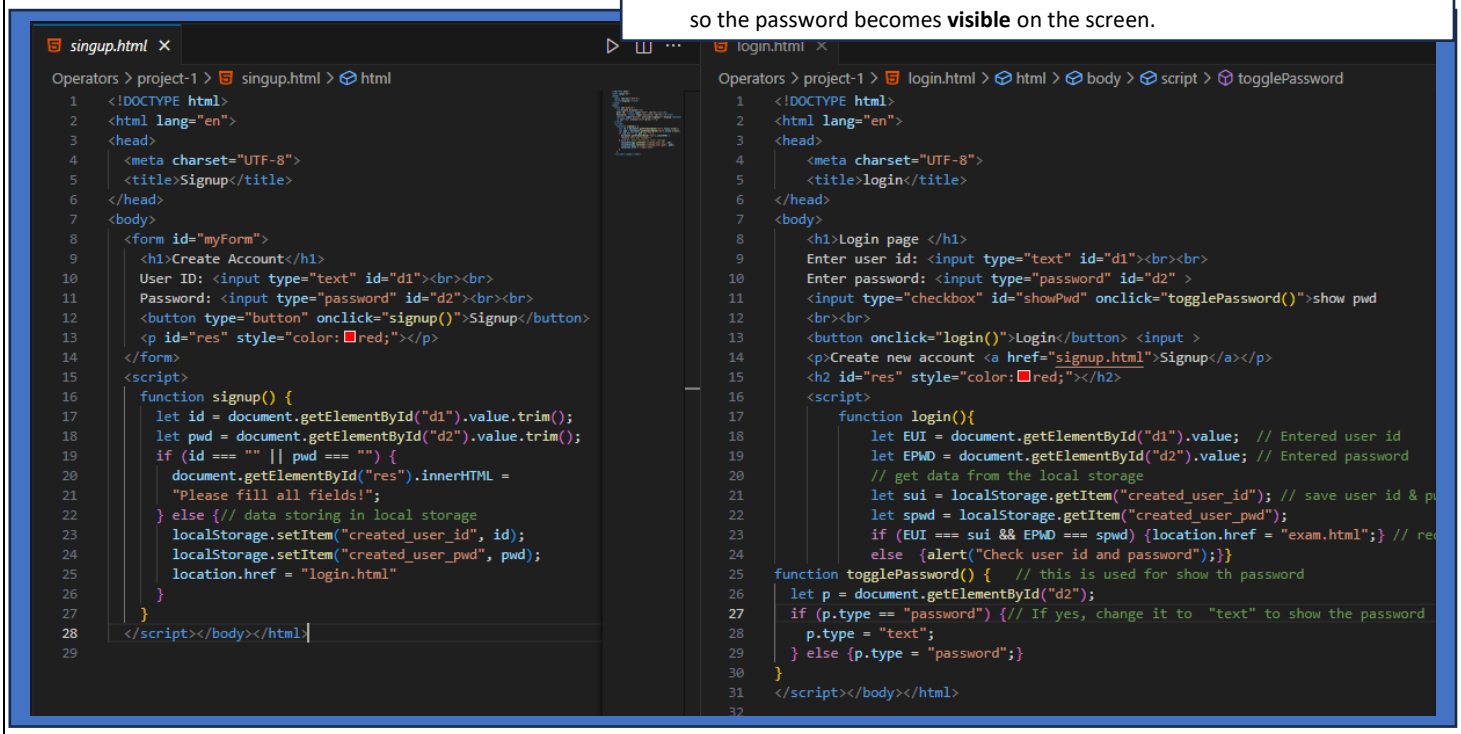

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="index.css">
</head>
<body> <!-- && and || operators based projects
    <div class="p">
        <a href="">Home</a>
        <a href="singup.html">Exam</a>
        <a href="singup.html">Singup</a>
        <a href="login.html">Login</a>
        <a href="#">Service</a>
        <a href="#">Product</a>
    </div></body></html>
```

```
.p{
    width: 100%;
    height: 12vh;
    border: 2px solid #000;
    background-color: darkblue;
    display: flex;
    justify-content: space-evenly;
    align-items: center;
    font-size: 30px;
}
.p>a{text-decoration: none;color: white;}
.p>a:hover{ color: yellow;}
```

Note:

if (p.type == "password")

- This checks the **current type** of the input box.
  If type is "password", that means password is **hidden** (shown as dots or stars).
- **p.type = "text";** If the type was "password", this line **changes it to "text"**, so the password becomes **visible** on the screen.




```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Signup</title>
</head>
<body>
  <form id="myForm">
    <h1>Create Account</h1>
    User ID: <input type="text" id="d1"><br><br>
    Password: <input type="password" id="d2"><br><br>
    <button type="button" onclick="signup()">Signup</button>
    <p id="res" style="color:red;"></p>
  </form>
  <script>
    function signup() {
      let id = document.getElementById("d1").value.trim();
      let pwd = document.getElementById("d2").value.trim();
      if (id === "" || pwd === "") {
        document.getElementById("res").innerHTML =
        "Please fill all fields!";
      } else {// data storing in local storage
        localStorage.setItem("created_user_id", id);
        localStorage.setItem("created_user_pwd", pwd);
        location.href = "login.html"
      }
    }
  </script></body></html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>login</title>
</head>
<body>
    <h1>Login page </h1>
    Enter user id: <input type="text" id="d1"><br><br>
    Enter password: <input type="password" id="d2" >
    <input type="checkbox" id="showPwd" onclick="togglePassword()">show pwd
    <br><br>
    <button onclick="login()">Login</button> <input >
    <p>Create new account <a href="signup.html">Signup</a></p>
    <h2 id="res" style="color:red;"></h2>
    <script>
        function login(){
            let EUI = document.getElementById("d1").value;  // Entered user id
            let EPWD = document.getElementById("d2").value; // Entered password
            // get data from the local storage
            let sui = localStorage.getItem("created_user_id"); // save user id & p
            let spwd = localStorage.getItem("created_user_pwd");
            if (EUI === sui && EPWD === spwd) {location.href = "exam.html";} // re
            else  {alert("Check user id and password");}}
function togglePassword() {   // this is used for show th password
  let p = document.getElementById("d2");
  if (p.type == "password") {// If yes, change it to  "text" to show the password
    p.type = "text";
  } else {p.type = "password";}
}
</script></body></html>
```

Exam.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Exam</title>
</head>
<body>
  <h1>Welcome to RTS</h1>
  <p>1. Who is the Prime Minister of India?</p>   <!-- Question 1 -->
  <p>A. Rahul Gandhi<br>B. Narendra Modi<br>C. Nitish Kumar<br>D. Mamata Ji</p>
  Ans: <input type="text" id="d1"><br><br>
  <p>2. What is the capital of India?</p>   <!-- Question 2 -->
  <p>A. Mumbai<br>B. Kolkata<br>C. New Delhi<br>D. Hyderabad</p>
  Ans: <input type="text" id="d2"><br><br>
  <p>3. What is the National Animal of India?</p>    <!-- Question 3 -->
  <p>A. Lion<br>B. Elephant<br>C. Tiger<br>D. Peacock</p>
  Ans: <input type="text" id="d3"><br><br>
  <p>4. Which planet is called the Red Planet?</p>   <!-- Question 4 -->
  <p>A. Earth<br>B. Jupiter<br>C. Mars<br>D. Venus</p>
  Ans: <input type="text" id="d4"><br><br>
  <button type="button" onclick="submitExam()">Submit</button>
  <button type="button" onclick="result()">Check Result</button>
  <h2 id="submit_msg" style="color: green;"></h2>
  <h1 id="total_marks"></h1>
  <h1 id="wrong_answer" style="color: red;"></h1>
  <script src="exam.js"></script>   <!-- Link external JS -->
</body></html>
```

exam.js

```js
let right_answer = 0;
let wrong_answer = 0;
function submitExam() {
  let a1 = document.getElementById("d1").value.toUpperCase();
  let a2 = document.getElementById("d2").value.toUpperCase();
  let a3 = document.getElementById("d3").value.toUpperCase();
  let a4 = document.getElementById("d4").value.toUpperCase();
  if (a1 === "B")
    {
      right_answer=right_answer+1;
    }
  else {
    wrong_answer=wrong_answer+1;
}
  if (a2 === "C") right_answer++; else wrong_answer++;
  if (a3 === "C") right_answer++; else wrong_answer++;
  if (a4 === "C") right_answer++; else wrong_answer++;
  document.getElementById("submit_msg").innerHTML = "Your answer submitted successfully";
}
function result() {
  document.getElementById("total_marks").innerHTML = "Total Marks = " + (right_answer * 4);
  document.getElementById("wrong_answer").innerHTML = "Wrong Answer = " + wrong_answer;
}
```

Data types

- Data types define the type of value stored in a variable.
- JavaScript supports different data types to store various kinds of values.
- Note: JS is a **dynamic type language**, meaning you don't need to declare type — it's decided automatically.

❖ **Types of Data Types**

1.Primitive Data Types 2. Non-Primitive (Reference) Data Types

1. Primitive data types

- These are **basic/inbuilt data types** in JavaScript.

Type	Description	Example
Number	Represents numbers (integer or decimal)	let age = 25; let marks=90.3
String	Sequence of characters in quotes	let name = "Sangam"; let a= 'rid'
Boolean	Logical values – true or false	let isStudent = true; let b=false
Undefined	Variable declared but not assigned	let x;
Null	Intentional empty or no value	let data = null;
Symbol (ES6)	Unique identifier value	let id = Symbol("id");
BigInt (ES11)	Large integer values	let big = 1234567890123456789 0n;

demo1.js

```js
const age = 15;
let marks = 90.3;
let n = -20;
const name = "Sangam Kumar";
let b = "skills";
const isStudent = true;
let c = false;
let a;
const emptyValue = null;
const uniqueKey = Symbol("unique");
const bigIntValue = 678901234567890n;

console.log(typeof (age));        // number
console.log(typeof marks);        // number
console.log(typeof n);            // number
console.log(typeof name);         // string
console.log(typeof b);            // string
console.log(typeof isStudent);    // boolean
console.log(typeof c);            // boolean
console.log(typeof a);            // undefined
console.log(typeof emptyValue);   // object
console.log(typeof uniqueKey);    // symbol
console.log(typeof bigIntValue); // bigint
```

2. Non -primitive data types

These are reference types, used to store multiple or complex values.

Type	Description	Example
Object	Stores key–value pairs	let student = {name: "Raj", age: 20};
Array	Stores list of values	let colors = ["red", "green"];
Function	Block of reusable code	function greet() { return "Hi"; }
Date	Stores date & time	let today = new Date();
RegExp	Used for pattern matching	let p = /world/;
Map	Stores key–value pairs (any data type)	let map = new Map();
Set	Stores unique values	let set = new Set([1, 2, 3]);

Example Program:

demo1.js > ...

```
1   const person = { name: "Sangam", age: 15 };
2   const numbers = [10, 20, 30];
3   function add(a, b) { return a + b; }
4   const currentDate = new Date();
5   const pattern = /hello/i;
6   const map = new Map([["name", "Raj"], ["age", 20]]);
7   const set = new Set([1, 2, 3]);
8   const uniqueKey = Symbol("unique");
9
10  console.log(typeof person);    // object
11  console.log(typeof numbers);   // object
12  console.log(typeof add);       // function
13  console.log(typeof currentDate); // object
14  console.log(typeof pattern);   // object
15  console.log(typeof map);       // object
16  console.log(typeof set);       // object
17  console.log(typeof uniqueKey); // symbol
```

Difference between Primitive and Non-Primitive Data Types

Feature	Primitive Data Types	Non-Primitive Data Types
Definition	Stores single, immutable (fixed) value.	Stores collections or complex data (multiple values).
Type of Storage	Stored **by value** (copy is made).	Stored **by reference** (points to same memory).
Mutability	**Immutable** – value cannot be changed once created.	**Mutable** – values can be modified.
Memory Allocation	Stored in **stack memory**.	Stored in **heap memory**.
Data Types Included	Number, String, Boolean, Undefined, Null, Symbol, BigInt.	Object, Array, Function, Date, Map, Set, etc.
Copy Behavior	Copy creates a new independent value.	Copy refers to the same original object.
Type Checking	Checked using typeof.	Most return object when checked with typeof.
Example	let a = 10; let b = a; b = 20; → a still 10	let x = [1,2]; let y = x; y[0]=5; → both x and y change

Mini Project Based on Object data types

```
<!DOCTYPE html>
<html lang="en">
<head> <meta charset="UTF-8">
  <title>Student Info</title>
</head>
<body>
  <form>
    <fieldset>
      <legend>Student Info</legend>
      Name: <input id="d1" type="text"><br><br>
      Age: <input id="d2" type="text"><br><br>
      Subject: <input id="d3" type="text"><br><br>
      Marks: <input id="d4" type="text"><br><br>
      DOB: <input id="d5" type="date"><br><br>
      Mobile No: <input id="d6" type="tel" >
<button type="button" onclick="sub()">Submit</button>
      <h1 id="res1" style="color: green;"></h1>
      <h3 id="res2" style="color: green;"></h3>
    </fieldset>
  </form>
  <script>
    function sub() {
      // Create object from input values
      let n=document.getElementById("d1").value
      let student = {
        Name:n,
        Age: document.getElementById("d2").value,
        Subject: document.getElementById("d3").value,
        Marks: document.getElementById("d4").value,
        DOB: document.getElementById("d5").value,
        Mobile_No: document.getElementById("d6").value
      };          // Save object as string in local storage
  localStorage.setItem("studentInfo", JSON.stringify(student));
  document.getElementById("res1").innerText = "Data Submitted Successfully!";
// Display all data as readable text
    let output = `  // backtrick
      Name: ${student_info.Name}<br>
      Age: ${student_info.Age}<br>
      Subject: ${student_info.Subject}
      Marks: ${student_info.Marks}<br>
      DOB: ${student_info.DOB}<br>
Mobile No: ${student_info.Mobile_No} `;
document.getElementById("res2")
.innerHTML = output;}

// Get data back from localStorage and show in res3
let stored = JSON.parse(localStorage.getItem("studentInfo"));
document.getElementById("res3").innerText = JSON.stringify(stored, null, 2);
</script></body></html>
```

Student Info

Name: Sangam Kumar

Age: 20

Subject: Python

Marks: 99

DOB: 14-06-2025

Mobile No: 9202707903

Submit

Data Submitted Successfully!

Name: Sangam Kumar
Age: 20
Subject: Python
Marks: 99
DOB: 2025-06-14
Mobile No: 9202707903

{
"Name": "r",
"Age": "ds",
"Subject": "sd",
"Marks": "sdf",
"DOB": "2025-06-13",
"Mobile_No": "87986+7863"
}

localStorage.getItem("studentInfo")

- This **gets** data stored in local storage under the key "studentInfo".
- data is in **string** format (because localStorage only stores strings).

JSON.parse(...) This **converts** string data back into a normal **JS object**.

- Now you can access it like: storedData.Name, storedData.Age,

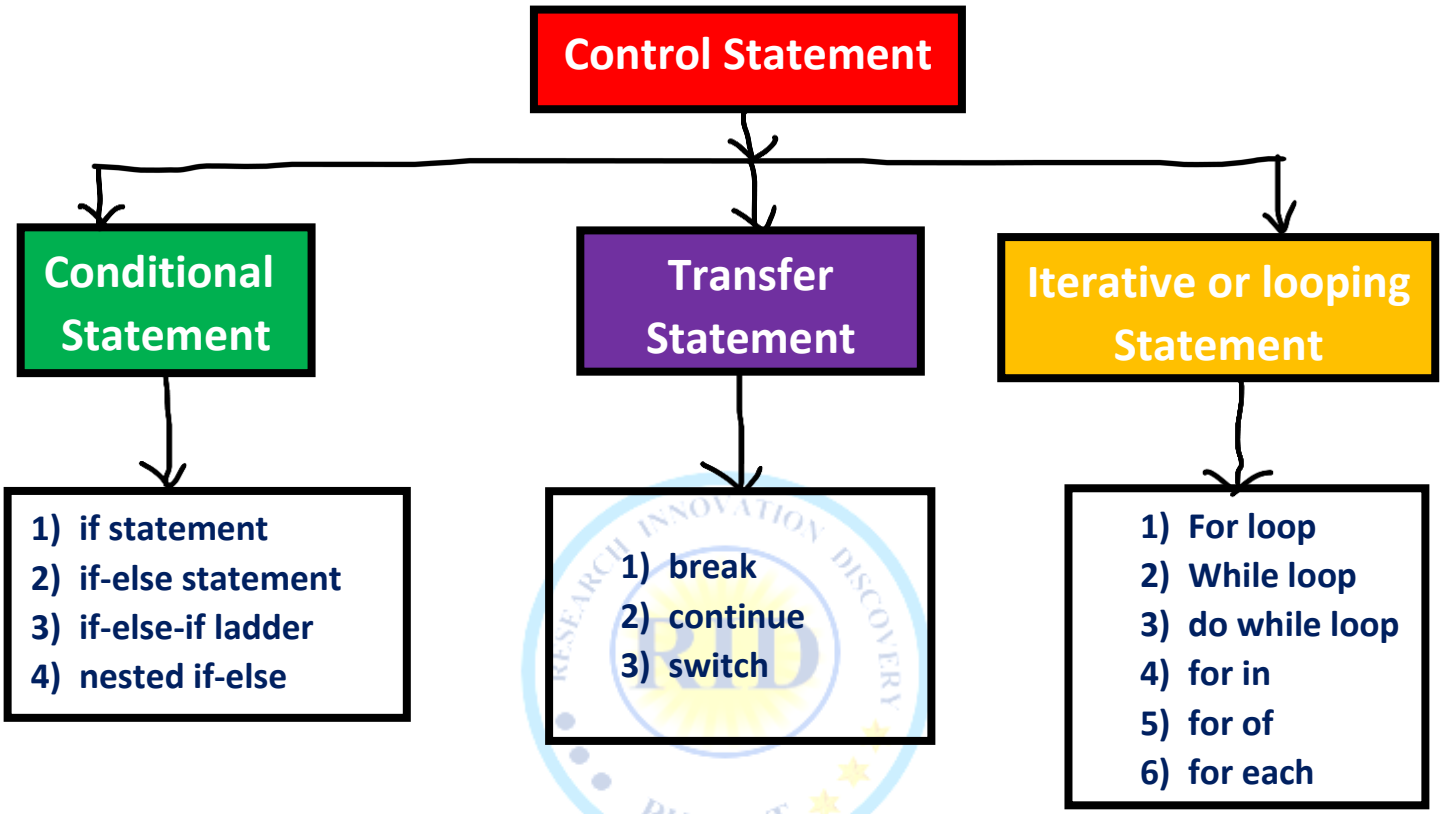
let storedData = … this stores final object in variable storedData.

JSON.stringify(stored, null, 2)

- Converts object back to a **nicely formatted string** so it can be shown on the page.
- null, 2 means it adds spaces/indentation (2 spaces) to make it look like this:

CONTROL STATEMENT

- Control Statement or Flow control describe the order in which statements will be executed at runtime.
- To change the programing follow based on the condition we will used control statement.
- This is used for Decision making.



CONDITIONAL STATEMENTS

- There are three main types of conditional statements:
  1. if statements,
  2. if-else statements,
  3. else-if ladders.
  4. Nested if-else

1) if statement:

- The if statement is used to execute a block of code if the condition is true.

Syntax:

```
if (condition) {
    // code to be executed if condition is true
}
```

Example-1:

```
let age =27
if (age >= 18) {
console.log("You are eligible for vote in Bharat
```

Example-2:

```
let a=10
If(a>0){console.log("+Ve")
If(a<0){console.log("-ve")
```

Example-3:

```
let temperature = 35;
if (temperature > 30) {
  console.log("It's a hot day!");
}
```

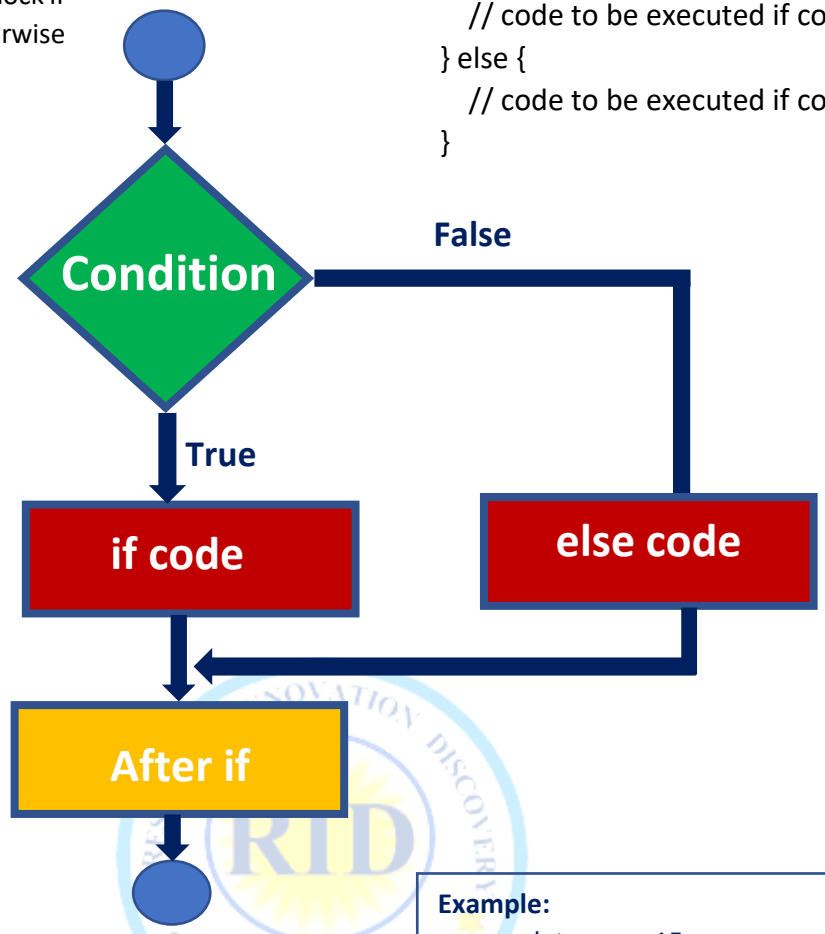
Example-4:

```
let isRaining = true;
if (isRaining) {
  console.log("Take an umbrella!");
}
```

2) if else statement:

- if-else statement also tests the condition. It executes if block if condition is true otherwise else block is executed.

Flow Chart:



Syntax:

```
if (condition) {
   // code to be executed if condition is true
} else {
   // code to be executed if condition is false
}
```

Example-1:

```
let num = 15;
if (num> 0) {
console.log("The number is positive.");
}
else
 {
console.log("The number is not positive.");
}
```

Output:number is positive.

Example:

```
//DYNAMIC INPUT IN node js
//Install the module prompt-sync
// npmi prompt-sync
const prompt = require("prompt-sync")();
let num = parseInt(prompt("Enter the Number:"))
if (num> 0) {
console.log("The number is positive.");
} else {
console.log("The number is not positive.");
}
```

Example:

```
let num = 15;
if (num%2==0) {
console.log("Even Number.");
}
else{ console.log("Odd Number."); }
```

Output:

odd number

Output: Enter the Number:0
The number is not positive.
Enter the Number:-3
The number is not positive.
Enter the Number:15
The number is positive.

Question

1. Check a number is positive or negative
2. Check a number is odd or even
3. Check a number is divisible by 5 or not
4. Check a number is between 1 and 100 (inclusive)

3) if else if ladder statement:

- The if-else-if ladder statement is a series of if-else statements where each 'if' condition is checked sequentially until a true condition is found, or the final else block is executed if none of the conditions are true.

Syntax:

```
if (condition1) {
   // code to execute if condition1 is true
} else if (condition2) {
   // code to execute if condition2 is true
} else if (condition n) {
   // code to execute if condition3 is true
} else {
   // code to execute if none of the
above conditions are true
}
```

Example-1:

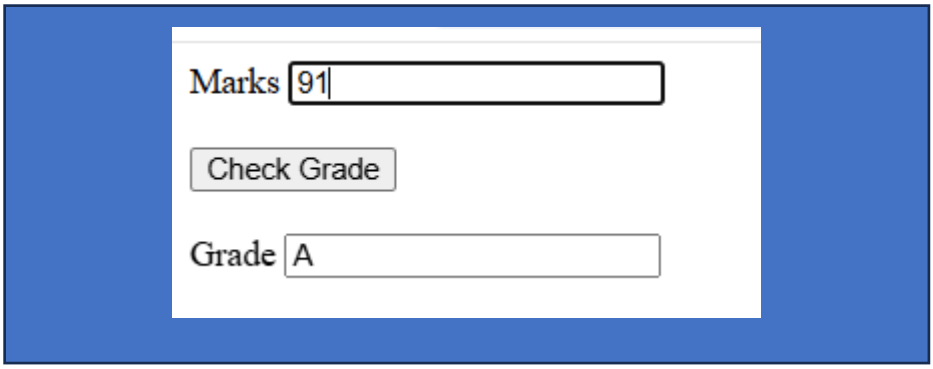
```
//DYNAMIC INPUT IN node js
//Install the module prompt-sync
// npmi prompt-sync
const prompt = require("prompt-
sync")();
let score = parseInt(prompt("Enter
the Marks:"))
if (score >= 90) {
console.log("You got an A Grade.");
}
 else if (score >= 80) {
console.log("You got a B Grad.");
}
else if (score >= 70) {
console.log("You got a C Grad.");
}
else if (score >= 60) {
console.log("You got a D Grad.");
 }
 else if (score >= 50) {
console.log("You got a E Grad.");
 }
else {
 console.log("Fail!!!");
}
```

Output:

```
Enter the Marks:91
You got an A Grade.
Enter the Marks:59
You got a E Grad.
Enter the Marks:45
Fail!!!
```

Example-2:

```
<!DOCTYPE html>
<html lang="en">
<head>
</head>
<body>
  <label for="d1">Marks</label>
  <input id="d1" type="text" placeholder="Enter your marks"><br><br>
  <button type="button" onclick="grade()">Check Grade</button><br><br>
  <label for="d2">Grade</label>
  <input type="text" id="d2">
  <script>
    function grade() {
      let marks = Number(document.getElementById("d1").value);
      let grade;
      if (marks >= 90 && marks <= 100) {
        grade = "A";
      } else if (marks >= 80 && marks < 90) {
        grade = "B";
      } else if (marks >= 70 && marks < 80) {
        grade = "C";
      } else if (marks >= 60 && marks < 70) {
        grade = "D";
      } else if (marks >= 50 && marks < 60) {
        grade = "E";
      } else {
        grade = "Fail";
      }
      document.getElementById("d2").value = grade;
      document.getElementById("d1").value = "";
    }
  </script>
</body></html>
```



Base Conversion project based on if else ladder

Changing a number from one base (like binary, octal, hex) to another.

Main Functions:

1. parseInt(string, base) → Converts string **to decimal**.
2. number.toString(base) → Converts decimal **to another base**.

Convert TO Decimal

```
parseInt("101010", 2); // Binary → 42
parseInt("52", 8);     // Octal → 42
parseInt("2A", 16);    // Hex → 42
```

Convert FROM Decimal

```
(42).toString(2);  // → "101010"  (Binary)
(42).toString(8);  // → "52"      (Octal)
(42).toString(16); // → "2a"      (Hex)
```

Project-1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>RID Base Conversion</title>
</head>
<body>
    <h1>RID Based Conversion Apps</h1>
    <label>Number</label>
    <input id="d1" type="text">
    <label>From</label>
    <select id="from">
   <option value="d">Decimal Number</option>
   <option value="b">Binary Number</option>
   <option value="o">Octal Number</option>
  <option value="h">HexaDeciaml Number</option>
    </select>
    <label>To</label>
    <select id="to">
        <option value="d">Decimal Number</option>
        <option value="b">Binary Number</option>
        <option value="o">Octal Number</option>
        <option value="h">HexaDeciaml Num</option>
    </select>
<button type="button" onclick="convert()">
Convert</button>
    <label>Result</label>
    <input id="res" type="text" >
<script src="raj1.js"> </script>
</body> </html>
```

```
function convert(){
    let num=document.getElementById("d1").value.trim()
    let frominp=document.getElementById("from").value
    let toinp=document.getElementById("to").value
    let res=""
// convert user input into decimal  // parseInt()
    let dec_num
if(frominp==="b"){dec_num=parseInt(num, 2)}
else if(frominp==="o"){dec_num=parseInt(num,8)}
else if(frominp==="h"){dec_num=parseInt(num,16) }
else if(frominp==="d"){dec_num=parseInt(num,10)}
 // convert decimal to any tartget number sytem
// dec_num.toString()
if(toinp==="d"){res= dec_num.toString(10)}
else if(toinp==="b"){ res=dec_num.toString(2)}
else if(toinp==="o"){res=dec_num.toString(8)}
else if(toinp==="h"){res=dec_num.toString(16)}
    document.getElementById("res").value=res
}
```

RID Based Conversion Apps

Number

From Decimal Number To Decimal Number Convert

Result

Task based on if else ladder

1. Student Grade Calculator
2. Electricity Bill Calculator
3. Temperature Converter & Weather Status
4. Voting Eligibility Checker
5. Number Sign Checker (Positive, Negative, Zero)
6. Age Category Finder (Child, Teen, Adult, Senior)
7. Traffic Light Signal Simulator
8. Marks Percentage Evaluator)
9. Discount Calculator Based on Purchase Amount
10. Day Finder (1–7 → Sunday to Saturday)
11. Exam Result Status (Pass/Fail/Grace)
12. Letter Grade Finder (A, B, C, D, F)
13. Traffic Fine Calculator (Speed-Based Fine System)
14. ATM Withdrawal Validator (Check Balance, Limit, Invalid Input)

Project-2

```
Check given character
<!DOCTYPE html>
<html lang="en">
<head>
</head>
<body>
  <br>
  <label for="d1">Character</label>
  <input id="d1" type="text" placeholder="Enter your marks"><br><br>
  <button type="button" onclick="check()">Check Grade</button><br><br>
  <label for="d2">Grade</label>
  <input type="text" id="d2">
  <script>
    function check() {
      let char = document.getElementById("d1").value.trim();
      let res;
      if (char >= 'a' && char <= 'z') { res = "Lower case"; }
      else if (char >= "A" && char <= "Z") { res = "Upper Case"; }
      else if (char >= "0" && char <= "9") { res = "Digit"; }
      else { res = "Special symbol"; }
      document.getElementById("d2").value = res;
      document.getElementById("d1").value = " ";
    }
  </script></body></html>
```

Check given character

Character a

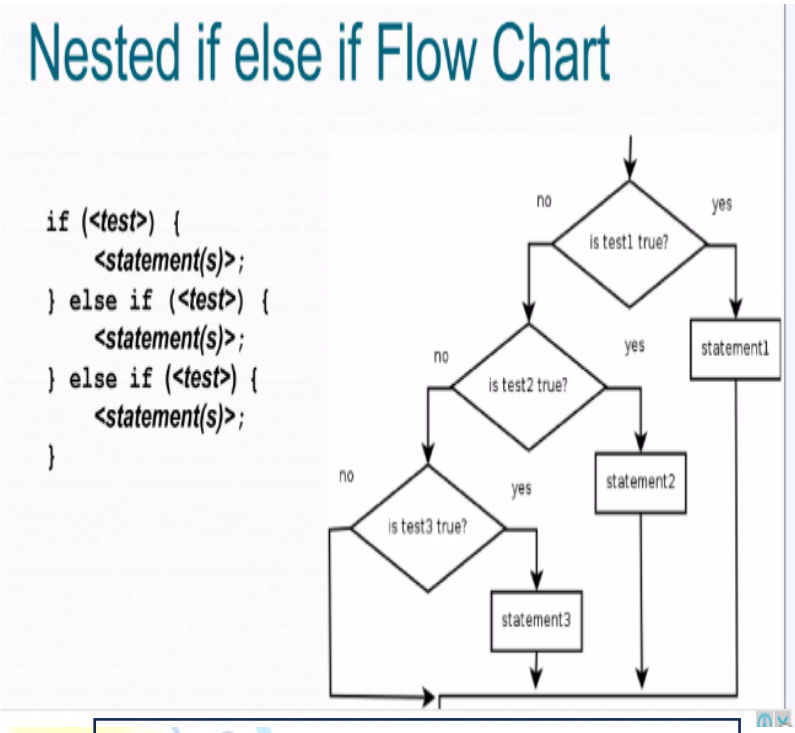
Check Grade

Grade Lower case

4.Nested If else:

- A nested if-else is an if-else structure inside another if-else block, allowing complex decision-making.

```
if(Condition/Expression)   <= Outer If Block
{
    if(Condition/Expression)   <= Inner If Block
    {
        Outer if and Inner If Statements;
    }
    else   <= Inner Else Block
    {
        Outer if and inner else statements;
    }
}
else <= Outer Else Block
{
    if(Condition/Expression)   <= Inner If Block
    {
        Outer else and inner if statements;
    }
    else   <= Inner Else Block
    {
        Outer else and inner else statements;
    }
}
```



1. Grading System

```
let marks = 87;
if (marks >= 50) {
   if (marks >= 85) {
      console.log("Grade: A+");
   } else if (marks >= 70) {
      console.log("Grade: A");
   } else {
      console.log("Grade: B");
   }
} else {
   console.log("Failed."); }
```

2. Login Successful Example

```
let username = "RID";
let password = "1234";
if (username === "RID") {
   if (password === "1234") {
      console.log("Login Successful!");
   } else {
      console.log("Incorrect Password!");
   }
} else {
   console.log("User not found!");
}
```

3. Age and License Eligibility

```
let age = 20;
let hasLearnerPermit = true;
if (age >= 18) {
   if (hasLearnerPermit) {
      console.log("You are eligible to apply for a driving
license.");
   } else {
      console.log("You need a learner's permit first.");
   }
} else {
   console.log("You are not old enough to apply for a license.");
}
```

4. Online Store Purchase Decision

```
let balance = 100;
let itemPrice = 75;
let isInStock = true;

if (balance >= itemPrice) {
   if (isInStock) {
      console.log("Purchase successful!");
   } else {
      console.log("Item is out of stock."); }
} else {
   console.log("Insufficient balance.");
}
```

LOOP STATEMENT

- A **loop** is used to **repeat a block of code** many times.
- Loops run **again and again** until the condition becomes **false**.
- Two types of loops:
  1. **Entry Controlled Loop** – for, while (Condition is checked first, then loop runs)
  2. **Exit Controlled Loop** – do-whil (Loop runs first, then condition is checked)

NOTE: If condition is false → **for/while** will NOT run But **do-while** will run **at least once**

❖ Types of loops

1) **for :-** It repeats the code for a fixed number of times or until a condition becomes false
2) **while :-** It repeats the code as long as the given condition remains true.
3) **do while :-** It runs the code at least once and then repeats it while the condition is true.
4) **for in :-** It is used to loop through the *keys/properties* of an object.
5) **for of :-** t is used to loop through the *values* of an array, string, or iterable.
6) **for each :-** It runs a function on every element of an array, one by one.

1. for Loop :- The for loop repeats the code a specific number of times.

Syntax:
```
for (initialization; condition; change) {
  // code
 }
```

Example:
```
for (let i = 1; i <= 5; i++) {
 console.log(i);}
```

- **initialization** → start value
- **condition** → loop will run until this is true
- **inc/dec** → value increases or decreases in each round
- **code block** → runs again and again

2. while Loop:- The while loop repeats the code as long as the condition is true.

Syntax:
```
while (condition) {
// code
}
```

Example:
```
let i = 1;
while (i <= 5) {
 console.log(i);
 i++; }
```

❖ How to Print output in one line

```
let n=10
let output=""
for (let i=0; i<=n; i++){
  output=output+i+" "; // output+=i+" ";
}
console.log("Number=",output)
```

3. do-while Loop The do-while loop runs the code at least once and then checks the condition.

Syntax:
```
do {
// code
} while (condition);
```

Example:
```
let i = 1;
do {
console.log(i);
 i++;
} while (i <= 5);
```

4. for-in Loop:- it is used to loop through **keys/properties** of an object.

Syntax:
```
for (let key in object) {
// code
}
```

Example:
```
let student = { name: "Raj", age: 16 };
 for (let key in student) {
console.log(key, student[key]); }
```

Example for ++ & -- operator

```
let a=20
console.log(a) //20
console.log(++a) //21
console.log(a++) //21
console.log(a)//22
console.log(--a)//21
console.log(a)//21
console.log(a--)//21
console.log(a)//20
```

5. for-of Loop :- The for-of loop is used to loop through the **values** of arrays or strings.

Syntax: for (let value of array) {
// code
}

Example: let colors = ["red", "blue", "green"];
for (let c of colors) {
console.log(c); }

6. forEach() Loop:- forEach() loop runs a function on every element of an array.

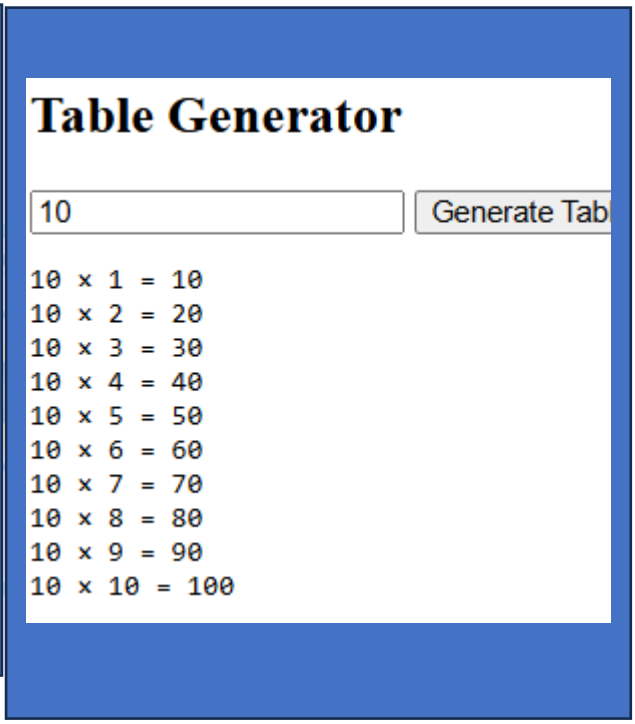
Syntax: array.forEach(function(item) { // code }});

Example: let nums = [5, 10, 15];
nums.forEach(function(n) {
console.log(n);
});

For loop project

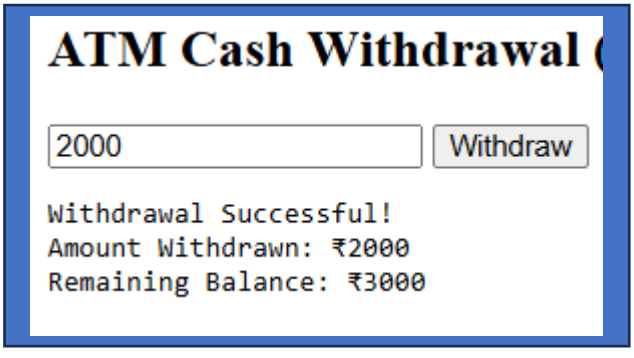
```
<!DOCTYPE html>
<html>
<head>
    <title>Table Generator</title>
</head>
<body>
<h2>Table Generator </h2>
<input type="number" id="num" placeholder="Enter Number">
<button onclick="makeTable()">Generate Table</button>
<pre id="output"></pre>
<script>
function makeTable() {
    let n = parseInt(document.getElementById("num").value);
    let result = "";

    for (let i = 1; i <= 10; i++) {
        result += `${n} x ${i} = ${n * i}\n`;
    }
    document.getElementById("output").innerHTML = result;
}
</script>
</body>
</html>
```



while loop project

```
<!DOCTYPE html>
<html>
<head>
    <title>ATM Simulator</title>
</head>
<body>
<h2>ATM Cash Withdrawal (While Loop)</h2>
<input type="number" id="amount" placeholder="Enter Amount to Withdraw">
<button onclick="withdrawMoney()">Withdraw</button>
<pre id="output"></pre>
<script>
let balance = 5000; // initial ATM balance
function withdrawMoney() {
    let amt = parseInt(document.getElementById("amount").value);
    let result = "";
    while (amt > balance || amt <= 0 || isNaN(amt)) {
        result = "enter a valid value less than or equal to ₹" + balance;
        document.getElementById("output").textContent = result;
        return; // stop function and wait for correct value
    }
    // valid withdrawal
    balance = balance - amt;
    result = `Withdrawal Successful!
Amount Withdrawn: ₹${amt}
Remaining Balance: ₹${balance}`;
    document.getElementById("output").textContent = result;
}
</script>
</body>
</html>
```

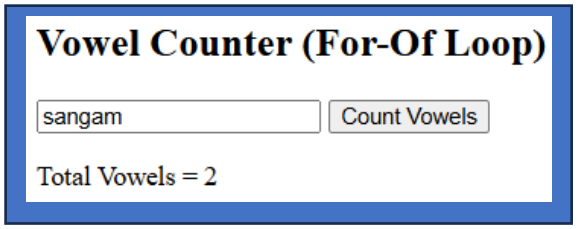


Task based on while loop

- Shopping cart total calculator
- Bus ticket booking system
- Water tank filling simulator
- Electricity bill calculator
- Daily expense tracker

For of loop project

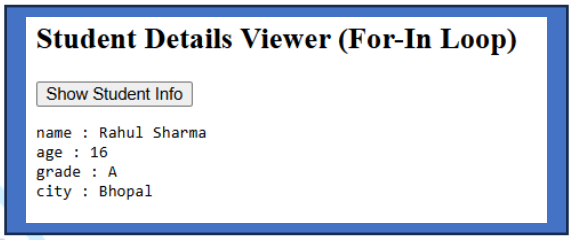
```html
<!DOCTYPE html>
<html>
<head>
    <title>Vowel Counter</title>
</head>
<body>
<h2>Vowel Counter (For-Of Loop)</h2>
<input type="text" id="text" placeholder="Enter any word">
<button onclick="countVowels()">Count Vowels</button>
<p id="result"></p>
<script>
function countVowels() {
    let word = document.getElementById("text").value.toLowerCase();
    let vowels = "aeiou";
    let count = 0;
    for (let ch of word) {
        if (vowels.includes(ch)) {     // includes() to quickly check if a character exists
        //if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
            count++;
        }
    }
    document.getElementById("result").textContent = "Total Vowels = " + count;
}
</script>
</body>
</html>
```



For in loop project

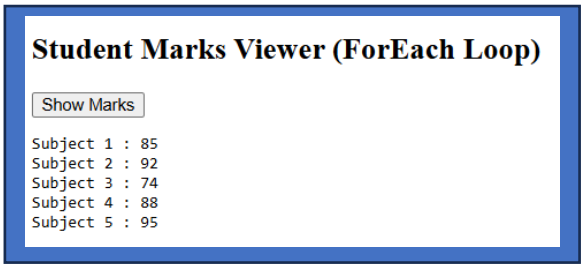
```html
<!DOCTYPE html>
<html>
<head>
    <title>Student Details</title>
</head>
<body>
<h2>Student Details Viewer (For-In Loop)</h2>
<button onclick="showDetails()">Show Student Info</button>
<pre id="info"></pre>
<script>
function showDetails() {
    let student = {
        name: "Rahul Sharma",
        age: 16,
        grade: "A",
        city: "Bhopal"
    };
    let output = "";

    for (let key in student) {
        output += `${key} : ${student[key]}\n`;
    }
    document.getElementById("info").textContent = output;
}
</script>
</body>
</html>
```



forEach loop project

```html
<!DOCTYPE html>
<html>
<head>
    <title>Student Marks</title>
</head>
<body>
<h2>Student Marks Viewer (ForEach Loop)</h2>
<button onclick="showMarks()">Show Marks</button>
<pre id="info"></pre>
<script>
function showMarks() {
    // Array of marks
    let marks = [85, 92, 74, 88, 95];
    let output = "";
    // forEach loop to display each mark
    marks.forEach(function(value, index) {
        output += `Subject ${index + 1} : ${value}\n`;
    });
    document.getElementById("info").textContent = output;
}
</script>
</body>
</html>
```



Transfer statement

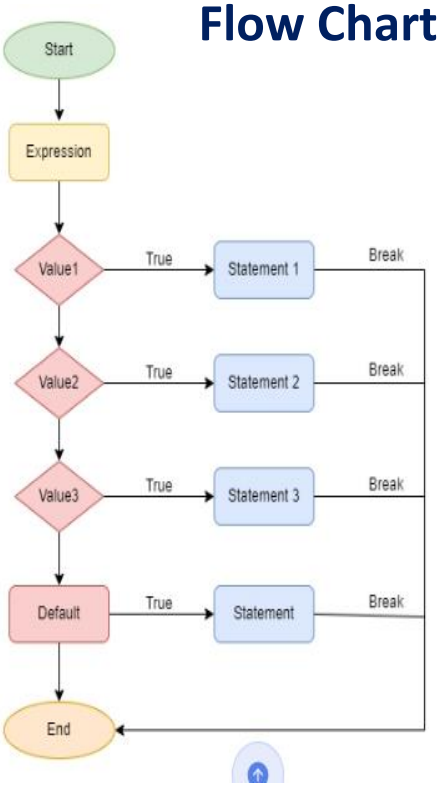
❖ Switch Statement:

- Use the switch statement to select one of many code blocks to be executed.

Syntax:switch(expression) {

```
case x:
  // code block
  break;
case y:
  // code block
  break;
default:
  // code block}
```

Flow Chart



❖ How it's works:

- The switch expression is evaluated once.The value of the expression is compared with the values of each case.If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Question: Program to print the correct week name as per given week number

1-Monday, 2-Tuesday, 3-Wednesday, 4-Thursday, 5-Friday, 6-Saturday, 7-Sunday

Program:

```
const prompt = require("prompt-sync")();
let week = parseInt(prompt("Enter Your Choice:"))
        switch(week){
          case 1:  console.log("Monday")
             break
          case 2:  console.log("Tuesday")
             break
          case 3:  console.log("Wednesday")
             break
          case 4:  console.log("Thursday")
             break
          case 5: console.log("Friday")
             break
          case 6:  console.log("Saturday")
             break
          case 7:  console.log("Sunday")
             break
          default:
        console.log("Wrong input")}
```

Output: Enter Your Choice:3

Wednesday

Example: Calculator

```
const prompt = require("prompt-sync")();
let num1 = parseFloat(prompt("Enter the first number: "));
let num2 = parseFloat(prompt("Enter the second number: "));
let operator = prompt("Enter an operator (+, -, *, /): ");
switch (operator) {
   case '+':
console.log(`Result: ${num1 + num2}`);
      break;
   case '-':
console.log(`Result: ${num1 - num2}`);
      break;
   case '*':
console.log(`Result: ${num1 * num2}`);
      break;
   case '/':
      if (num2 !== 0) {
console.log(`Result: ${num1 / num2}`);
      } else {
console.log("Error: Division by zero");
      }
      break;
   default:
console.log("Invalid operator");}
```

Output: Enter the first number: 10

Enter the second number: 20

Enter an operator (+, -, *, /): +

Result: 30

Problem: write a program to print the grade according to given marks:

```
let n = 91;
switch (true) {
  case (n >= 90 && n <= 100):
    console.log("A");
    break;
  case (n >= 80 && n < 90):
    console.log("B");
    break;
  case (n >= 70 && n < 80):
    console.log("C");
    break;
  case (n >= 60 && n < 70):
    console.log("D");
    break;
  default:
console.log("not correct");
}
```

❖ Break and Continue

- break statement in JavaScript is used to terminate a loop or switch statement prematurely.
- break statement in JavaScript exits a loop or switch block immediately, transferring control to the code following it.

Example:

```
for (let i = 1; i <= 5; i++) {
   if (i === 3) {
      break; // Exit the loop when i equals 3
   } console.log(i);
}
```

Output: 1, 2

❖ Continue:

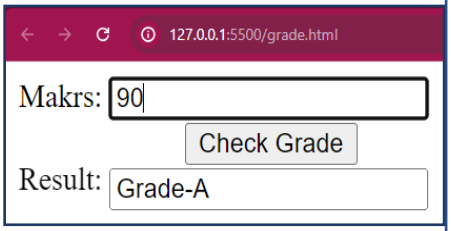
- continue statement in JavaScript skips the current iteration of the loop and proceeds with the next iteration.

Example:

```
for (let i = 1; i <= 5; i++) {
   if (i === 3) {
      continue; // Skip the rest of the code for this iteration
   }
   console.log(i);
}
```

Output: 1, 2, 4, 5

```
<!DOCTYPE html>
<html lang="en"><head>
<meta charset="UTF-8">
<title>Document</title>
</head><body>
<form onsubmit="sub(event)">
<label for="d1">Makrs:</label>
<input type="text" placeholder="Enter Marks" id="d1">
<!--<button type="submit" onclick="check()">Check
Grade</button><br><br> -->
<button type="submit">Check Grade</button><br><br>
</form>
<label for="d2">Result:</label>
<input type="text" id="d2">
<script>
      function sub(event){
event.preventDefault()
check()}
      function check() {
         let marks = parseFloat(document.getElementById("d1").value)
document.getElementById("d1").value = ""
         let result =""
         if(!isNaN(marks)){
         switch (true) {
            case (marks >= 90 && marks <= 100):
               result = "Grade-A"
               break
            case (marks >= 80 && marks < 90):
               result = "Grade-B"
               break
            case (marks >= 70 && marks < 80):
               result = "Grade-C"
               break
            case (marks >= 60 && marks < 70):
               result = "Grade-D"
               break
            case (marks >= 50 && marks < 60):
               result = "Grade-E"
               break
            default:
               result = "Fail"}}
else{
         result="Invalid Number"}
document.getElementById("d2").value = result}
</script>
</body>
</html>
```

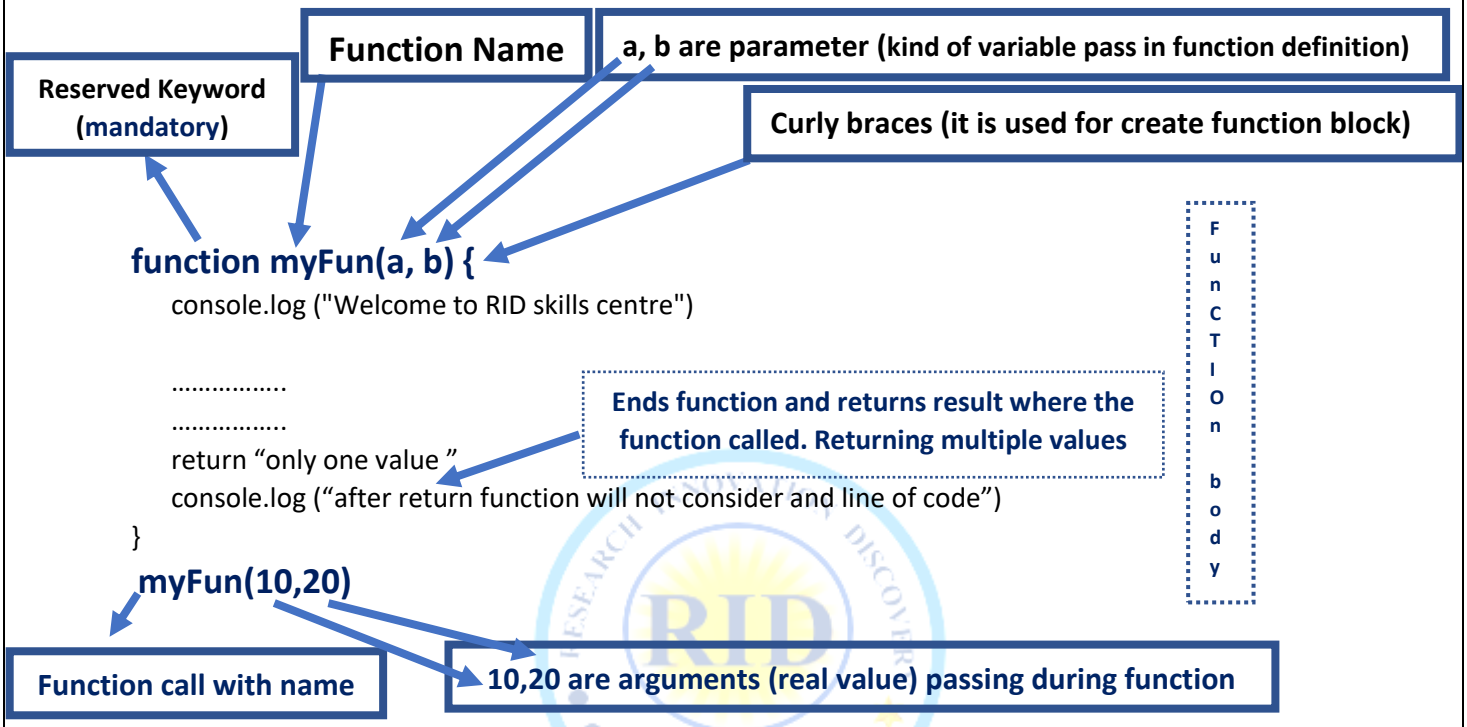


FUNCTIONS

- Function is a block of code that run when we call.
- A **function** is a **block of reusable code** that performs a specific task.
- Functions help us organize code, make it clean, modular, and easy to maintain.
- A function runs only when it is called.

Advantages of Functions: Code Reusability & Less Coding – No need to repeat same code again

Syntax



Note:- JavaScript returns ONLY ONE value, but that value can be a container (array or object) holding many values.

1. **Function** – A block of code that performs a specific task when called.
2. **Function Name** – The unique name given to a function to identify and call it.
3. **Function Definition** – The way to create a function using the def keyword.
4. **Parameter** – A variable listed inside the function definition to receive values.
5. **Curly Braces** – {} is used to define block of code inside the function.
6. **Function Body** – The group of statements written inside a function block.
7. **Return** – Sends a value back from the function to the caller.
8. **Function Call** – The action of executing a function using its name and parentheses.
9. **Arguments** – The actual values passed to the function when it is called.
10. **Scope** – The area where a variable is accessible (like inside or outside a function).

❖ **Example-1:**

```
function add(a,b){
   let c=a+b ;
   return c ;
}
let sum= add(10,20) ;
console.log("sum of two number=",sum) ;
```

a and b are parameters

C return value to add function

10 and 20 are arguments

Output:

sum of two number=30

Types of Functions:

1. **User-defined Function** – A custom function created by the user using the function keyword.
2. **Built-in Function** – Functions provided by JavaScript (e.g., alert(), console.log(), parseInt()).
3. **Anonymous Function** – A function without a name, often used inside callbacks.
4. **Arrow Function** – A short and modern function written using =>.
5. **Callback Function** – A function passed as an argument to another function to be executed later.
6. **Higher-Order Function** – A function that accepts another function as a parameter or returns a function.
7. **Constructor Function** – A special function used to create objects using the new keyword.
8. **Recursive Function** – A function that calls itself to solve smaller parts of a problem.
9. **Nested Function** – A function defined inside another function.
10. **Async Function** – A function defined with async, used for asynchronous programming with await.

1. User-defined Function:- A custom function created by the user using the function keyword

Syntax
```
function functionName() {
  // code
}
```

Example:
```
function greet() {
  console.log("Hello!");
}
```

Example-2:
```
greet();
function add(a, b) {
  console.log(a + b);
}
add(5, 3);
```

2. Built-in Function:- Functions provided by JavaScript (e.g., alert(), console.log(), parseInt()).

Examples
```
console.log("Welcome");
let x = parseInt("40");
console.log(x);
```

3. Anonymous Function:- A function without a name, often used inside callbacks

Syntax
```
let x = function() {
  // code
};
```

Examples
```
let show = function () {
  console.log("Anonymous function running");
};
show();
```

Example:
```
setTimeout(function () {
  console.log("Inside anonymous callback");
}, 1000);
```

4. Arrow Function:- A short and modern function written using =>.

Syntax
```
let fn = () => {
  // code
};
```

Examples
```
let hello = () => console.log("Hello Arrow!");
hello();
```

Example:
```
let mul = (a, b) => console.log(a * b);
mul(4, 2);
```

5. Callback Function:- A function passed as an argument to another function to be executed later.

Syntax functionName(callback);

Examples
```
function calculate(a, b, callback) {
  callback(a, b);   // calling the callback function
}
function add(x, y) {
  console.log(x + y);
}
calculate(5, 3, add);
```

- **calculate()** receives 3 things: **a**, **b**, and a **callback function**
- **callback(a, b)** means → *run the callback function using a and b*
- We created a simple **add()** function and passed it as callback.

Example:-

```
function greet(name, callback) {
  console.log("Hello " + name);
  callback();  // calling the second function
}
function bye() {
  console.log("Goodbye!");
}
greet("Rahul", bye);
```

- greet() prints "Hello Rahul"
- Then it calls the callback function bye()
- Output:

  Hello Rahul
  Goodbye!

6. Higher-Order Function:- A function that accepts another function as a parameter or returns a function.

Syntax

```
function hof(fn) {
  fn();
}
```

Example-1:

```
function operate(fn) {
  fn();  // calling the function passed to it
}
function sayHello() {
  console.log("Hello!");
}
operate(sayHello);
```

Example-2:

```
function doSomething(action) {
  action();  // calling the passed function
}
function showMessage() {
  console.log("Task Completed!");
}
```

7. Constructor Function:- A special function used to create objects using the new keyword

Syntax

```
function FunctionName() {
  this.property = value;
}
```

Examples-1

```
function Student(name) {
  this.name = name;
}
let s1 = new Student("Rahul");
console.log(s1.name);
```

Example-2:

```
function Car(model) {
  this.model = model;
}
let c1 = new Car("BMW");
console.log(c1.model);
```

8. Recursive Function:- A function that calls itself to solve smaller parts of a problem.

Syntax

```
function func() {
  func();
}
```

Examples

```
function count(n) {
  if (n == 0) return;
  console.log(n);
  count(n - 1);
}
count(3);
```

Example:-2

```
function fact(n) {
  if (n == 1) return 1;
  return n * fact(n - 1);
}
console.log(fact(5));
```

9. Nested Function:- A function defined inside another function.

Syntax:

```
function outer() {
  function inner() {}
}
```

Examples

```
function outer() {
  function inner() {
    console.log("Inner function");
  }
  inner();
```

10. Async Function:- A function defined with async, used for asynchronous programming with await.

```
async function functionName() {
  await task;
}
async function getData() {
  let data = await fetch("https://jsonplaceholder.typicode.com/posts/1");
  let res = await data.json();
  console.log(res);
```

❖ Higher-Order Function (HOF)

- A higher-order function is a function that accepts another function as an argument OR returns a function.
- Callback functions are usually used inside HOFs.

❖ Example of Higher-Order Function

```
function operate(fn, x, y) {
   fn(x, y);   // directly calling the callback
}
// Callback Function
function multiply(a, b) {
   console.log(a * b);   // printing result
}
// Calling the HOF
operate(multiply, 4, 5);   // Output: 20
```

- powerOf is a **Higher-Order Function** because it returns another function.
- square becomes a function that calculates n^2.
- cube becomes a function that calculates n^3.

Example: HOF that returns a function

```
function powerOf(x) {
   return function(n) {
      console.log(n ** x);   // n raised to the power x
   };
}
let square = powerOf(2);   // prepares a function for square
square(5);                 // 25
let cube = powerOf(3);     // prepares a function for cube
cube(4);                   // 64
```

Example: Callback used inside a HOF

```
// Higher-Order Function
function calculate(callback) {
   callback(10);   // using the callback inside HOF
}
// Callback Function
function showDouble(num) {
   console.log(num * 2);
}
// Call the HOF with the callback
calculate(showDouble);   // Output: 20
```

Explanation

- calculate is a **Higher-Order Function** because it takes a function as a parameter.
- showDouble is a **callback function**.
- The HOF **uses the callback inside** its body (callback(10)).

cut copy clear project

Number: Even Print Table

Result: Copy Copy Table Cut Cut Table Clear

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

<body>
    <h1>cut copy clear project </h1>
    Number: <input type="text" id="d1" autocomplete="off" />
    <button type="button" onclick="isEven()">Even</button>
    <button type="button" onclick="table()">Print Table</button><br><br>
    Result: <input type="text" id="res" />
    <button type="button" onclick="copyText()">Copy</button>
    <button type="button" onclick="copyTable()">Copy Table</button>
    <button type="button" onclick="cutText()">Cut</button>
    <button type="button" onclick="cutTable()">Cut Table</button>
    <button type="button" onclick="clearInputs()">Clear</button>
    <pre id="output"></pre>
    <script>
        function getNum() {return Number(document.getElementById("d1").value);}
        function isEven() {    // EVEN CHECK
            let num = getNum();
            if (num % 2 === 0) {document.getElementById("res").value = num + " is Even";}
            else {document.getElementById("res").value = num + " is Odd";}} // PRINT TABLE
        function table() {let num = getNum();
            document.getElementById("res").value = "";
            let out = "";
            for (let i = 1; i <= 10; i++) {out += `${num} x ${i} = ${num * i}\n`;
            }document.getElementById("output").textContent = out;}// COPY result input
        function copyText() {
            let r = document.getElementById("res");
            r.select();
            document.execCommand("copy");}// CUT result input
        function cutText() {
            let r = document.getElementById("res");
            r.select();
            document.execCommand("copy"); r.value = "";}// COPY TABLE (from <pre>)
        function copyTable() {
            //→ Gets the text written inside the <pre id="output"> (the table).
            let text = document.getElementById("output").textContent;
            navigator.clipboard.writeText(text); //→ Copies that text to the clipboard.
        }// CUT TABLE (copy + clear)
        function cutTable() {
            let out = document.getElementById("output"); //out = the <pre id="output"> element.
            navigator.clipboard.writeText(out.textContent);  //→ Copies the table text to the clipboard.
            out.textContent = "";  }// CLEAR
        function clearInputs() {
            document.getElementById("d1").value = "";
            document.getElementById("res").value = "";
            document.getElementById("output").textContent = "";
            document.getElementById("pdata").value = "";}</script></body></html>
```

Differences Between Types of Functions in JavaScript

Feature	Normal Function	Arrow Function	Higher-Order Function	Anonymous Function	Callback Function
Definition	Standard `function` keyword	Uses `=>` syntax	Takes or returns a function	No name	Passed to another function
`this` Binding	Dynamic (based on caller)	Lexical (from outer scope)	Depends on implementation	Same as normal or arrow	Same as normal or arrow
Hoisting	Yes	No	N/A	No	N/A
Usage	General-purpose functions	Shorter syntax, callbacks	Functional programming	Inline or short functions	Passed to be executed later
Example	`function fn() {}`	`const fn = () => {}`	`fn(() => {})`	`() => {}` or `function()`	`fn(callback)`

↓

Synchronous vs Asynchronous in JavaScript

Synchronous (Blocking Code)

- Code runs **line by line**.
- Next line **waits** until the previous line finishes.
- If a task is slow, everything stops (blocked).

Example:

```
console.log("Start");
for (let i = 0; i < 1e9; i++) {}  // long task
console.log("End");
```

➔ **"End" prints only after the long loop finishes.**

Asynchronous (Non-Blocking Code)

- Code **does not wait** for long tasks.
- Slow operations run **in the background**.
- Uses **callbacks, promises, async/await**.

Example:

```
console.log("Start");
setTimeout(() => {
  console.log("Async task done");
}, 2000);
console.log("End");
```

Key Differences

Feature	Synchronous	Asynchronous
Execution	Tasks are executed in order.	Tasks can run concurrently.
Blocking	Code execution is blocked.	Code execution is non-blocking.
Use Case	Simple tasks.	Time-consuming tasks like API calls.
Examples	Loops, mathematical operations.	Timers, AJAX, file reading.

Object

What is an Object in JavaScript?

1. **Object is a collection of key–value pairs**
2. **Objects can store multiple data types** like String, number, array, function, another object, etc.
3. **Objects represent real-world entities** Example: student, car, user, product.
4. **Objects are mutable (changeable)** means You can add, update, or delete properties anytime.
5. **Objects use dot notation and bracket notation** Example: vobj.name or obj["name"].

Syntax key: value

Syntax of Object with Key–Value Pairs

```
let obj = {
   key1: value1,
   key2: value2
};
```

What is a Key?

- A **key** is the *property name* inside an object.
- It must be **unique** (no two keys with same name).
- Keys are always **strings**, even if you write them as numbers.
- Keys can contain: letters, numbers, underscores _, strings
- If the key has spaces, you must use **quotes**.

What is a Value?

- A **value** is the data stored inside the key.
- Value can be **any data type**: string, number, Boolean, array, object, function (called method)

Example 1:

```
let student = {
   name: "Tanmay",
   age: 16,
   grade: "A"
};
```

Example 2 (Number as Key + String Key with Spaces)

```
let product = {
   101: "Laptop",        // number key (converted to string internally)
   "product name": "Dell", // key with spaces → use quotes
   price: 45000
};
```

Important Rules:

1. **Keys must be unique.** Example: You cannot write two name keys.
2. **Keys are always strings internally.** Even 101: "data" becomes "101".
3. **Keys without quotes must follow JS naming rules.**
   Example: first_name first name (must use quotes)
4. **Values can be anything (string, number, array, object, function).**
5. **Use dot or bracket notation to access keys.**
   - obj.name
   - obj["product name"]

Example:

```
let student = {
   name: "Sangam",
   age: 16
};
console.log(student.name); // sangam
console.log(student.age);  // 16
```

How to create objects

1. **Object Literal**
2. **Constructor Function**
3. **ES6 Class**
4. **Object.create()**

1. Object Literal :- The simplest and most common way to create objects using {}.
You directly define key-value pairs inside curly braces.

Syntax

```
let obj = {
   key: value,
   key2: value2
};
```

Example 1

```
let car = {
   brand: "Honda",
   model: "City"
};
console.log(car.brand);
```

Example 2

```
let student = {
   name: "Sangam",
   age: 16
};
console.log(student.age);
```

2. Constructor Function: it is used with new keyword to create multiple objects with same structure.

Syntax

```
function Person(name, age) {
   this.name = name;
   this.age = age;
}
let p = new Person("A", 10);
```

Example 1

```
function Mobile(company, price) {
   this.company = company;
   this.price = price;
}
let m1 = new Mobile("Samsung", 15000);
```

Example 2

```
function Book(title, author) {
   this.title = title;
   this.author = author;
}
let b1 = new Book("Maths", "R.D Sharma");
console.log(b1.title);
```

3. ES6 Class :- A cleaner and modern way to write constructor functions using class keyword.

Syntax

```
class Person {
   constructor(name, age) {
      this.name = name;
      this.age = age;
   }
}
```

Example 1

```
class Student {
   constructor(name, roll) {
      this.name = name;
      this.roll = roll;
   }
}
let s1 = new Student("Ravi", 21);
console.log(s1.name);
```

Example 2

```
class Car {
   constructor(brand, year) {
      this.brand = brand;
      this.year = year;
   }
}
let c1 = new Car("Tata", 2024);
console.log(c1.year);
```

4. Object.create()

Creates a new object using an existing object as its **prototype**.
Syntax let obj = Object.create(prototypeObject);

Example 1

```
let proto = { greet() { console.log("Hello!"); } };
let person = Object.create(proto);
person.greet();
```

Example 2

```
let animal = { type: "Dog" };
let pet = Object.create(animal);
console.log(pet.type);
```

How to Access Data from Object

1. **Dot Notation**
2. **Bracket Notation**
3. **Object Destructuring**
4. **for...in Loop**

5.Object Methods

- Object.keys()
- Object.values()
- Object.entries()

1. Dot Notation: Used to access properties with a dot (.) Works when key has **no spaces or special characters**. **Syntax:** object.property

Example 1

```
let user = { name: "Ram", age: 20 };
console.log(user.name);
console.log(user.age);
```

Example 2

```
let car = { brand: "Honda", model: "City" };
console.log(car.brand);
console.log(car.model);
```

2. Bracket Notation:- Used when property name is **dynamic**, has spaces, or special characters.

Syntax: object["property"]

Example 1

```
let student = { "full name": "Sangam Kumar", age: 15 };
console.log(student["full name"]);
console.log(student["age"]);
```

Example 2

```
let laptop = { brand: "Dell", price: 45000 };
console.log(laptop["brand"]);
console.log(laptop["price"]);
```

3. Object Destructuring :- Extract multiple values from an object into variables.

Syntax let { prop1, prop2 } = object;

Example 1

```
let user = { name: "Sangam", age: 15 };
let { name, age } = user;
console.log(name, age);
```

Example 2

```
let car = { brand: "Tata", model: "Nexon" };
let { brand, model } = car;
console.log(brand, model);
```

4. for...in Loop:- Loops through **all keys** of an object.

Syntax :

```
for (let key in object) {
  object[key];
}
```

Example-1:

```
let user = { name: "Ram", age: 20 };
for (let key in user) {
  console.log(key, user[key]); }
```

Example 2

```
let book = { title: "JS Guide", pages: 200 };
for (let k in book) {
  console.log(k + ": " + book[k]);
}
```

5. Object Methods : Useful built-in methods to get keys, values or key-value pairs.

a) Object.keys() :- Syntax:- Object.keys(object)

- **Example-1:**
```
let user = { name: "Sangam", age: 15 };
console.log(Object.keys(user));
```
- **Example-2:**
```
let car = { brand: "BMW", price: 5000000 };
console.log(Object.keys(car));
```

b) Object.values():- Syntax Object.values(object)

- **Example 1 :-** console.log(Object.values({ name: "Ram", age: 20 }));
- **Example 2:-** console.log(Object.values({ brand: "Apple", model: "iPhone" }));

c) Object.entries():- Syntax Object.entries(object)

- **Example 1:-** console.log(Object.entries({ a: 1, b: 2 }));
- **Example 2:-** console.log(Object.entries({ name: "Sangam", age: 15 }));

Adding New Properties to an Object

Used to insert new key–value pairs into an existing object.

1. Dot Notation
2. Bracket Notation
3. Add Multiple Properties

1. Dot Notation:- Used to add a new property to an object using a dot (.).

Syntax:- object.property = value;

Example-1

```
let student = { name: "Rahul" };
student.age = 18;
console.log(student);
```

Example-2

```
let car = { model: "Swift" };
car.year = 2022;
console.log(car);
```

2. Bracket Notation:

Used to add a property using square brackets — helpful when the key is dynamic or contains spaces.

Syntax: - object["property"] = value;

Example-1

```
let student = { name: "Rahul" };
student["city"] = "Delhi";
console.log(student);
```

Example-2

```
let car = { model: "Swift" };
car["color"] = "Red";
console.log(car);
```

3. Add Multiple Properties (Object.assign)

- Used to insert multiple new key–value pairs into an object at the same time.

Syntax: - Object.assign(object, { key1: value1, key2: value2 });

Example-1

```
let student = { name: "Rahul" };
Object.assign(student, { age: 18, city: "Delhi" });
console.log(student);
```

Example-2

```
let car = { model: "Swift" };
Object.assign(car, { color: "Black", price: 600000 });
console.log(car);
```

Example-3:

```
let student={

   // empty object
}
// adding new properties and value in student
object
student.name="sangam"
student.age="15"
student.city="Patna"
student.state="Bihar"
student.Country="Bharat"
student.Branch="CSE"
console.log("Object student=",student)
for(let key in student){
console.log(key+": "+student[key])
}
```

❖ How to add multiple key and value at time

```
let money={
 //Empty Object
}
console.log(money)
const prompt=require('prompt-sync')()
let n=Number(prompt("Enter the number of key: "))
for(let i=1; i<=n; i++){
  keys1=prompt("Enter your key: ")
  value=prompt("Enter the value: ")
    money[keys1]=value
}console.log(money)
```

Note: npm install prompt-sync

```
PS C:\Users\hp\Desktop\RID ONLINE\function> node obj1.js
Enter the number of key: 3
Enter your key: count
Enter the value: 100
Enter your key: Note
Enter the value: yes
Enter your key: Coin
Enter the value: No
{ count: '100', Note: 'yes', Coin: 'No' }
PS C:\Users\hp\Desktop\RID ONLINE\function>
```

Deleting Properties from Object

Used to remove an existing key–value pair from an object.

1. **delete Operator**: Removes a property from an existing object

Syntax:- delete object.property;

or

delete object["property"];

Example 1 — Dot Notation

```
let student = { name: "Rahul", age: 18 };
delete student.age;
console.log(student);
```

Example 2 — Bracket Notation

```
let car = { model: "Swift", color: "Red" };
delete car["color"];
console.log(car);
```

2. Destructuring (to delete by creating a NEW object)

Creates a new object without specific properties.

Example

```
let data = { a: 1, b: 2, c: 3 };
let { b, ...newData } = data;
console.log(newData);
// { a: 1, c: 3 }
```

3. Loop + delete (For deleting multiple keys)

- When you want to delete many properties at once.

Example:

```
let obj = { name: "Amit", age: 20, city: "Delhi" };
let removeKeys = ["age", "city"];
removeKeys.forEach(key => delete obj[key]);
console.log(obj);
```

4. Reassigning a New Object: Remove all data by replacing object with empty object.

Example

```
let user = { a: 10, b: 20, c: 30 };
user = {};
console.log(user);
```

Delete Multiple Properties Using a For Loop

Example-1

```
let student = { name: "Rahul", age: 18, city: "Delhi" };
let keysToDelete = ["age", "city"];
for (let i = 0; i < keysToDelete.length; i++) {
   delete student[keysToDelete[i]];
}
console.log(student);
```

Output: { name: "Rahul" }

Example-2

```
let car = { model: "Swift", color: "Red", price: 600000 };
let remove = ["color", "price"];
for (let i = 0; i < remove.length; i++) {
   delete car[remove[i]];
}
console.log(car);
```

Output: - { model: "Swift" }

❖ Nested Objects:

- Nested objects in JavaScript are objects within other objects. They allow you to create complex data structures that can represent real-world entities more accurately.

Example-1 :

```
const student={
   marks: {
      maths: 100,
      Science: 98,
      English: 56,
      Hindi: 78
   },
   food_name: ["panipuri","momos","Burger"],
   College_name: "Oxford Business Collge",
   Birthday: {
      party: "yes",
      Gift: "pen",
      Cake: "3kg"
   },
   age: 25
}
console.log(marks.English)
for (let i in student){
console.log(i)
} console.log(student["Birthday"]["Gift"]) // gift is key of nested object
   console.log(student["food_name"][0])  // 0 is a index of array
//  console.log(student["food_name"])
b=student.food_name
for (let i of b){
  console.log(i)
}
for (let j in b){
   console.log(j)
   console.log(j,":",b[j])
}
```

```
momos
PS C:\Users\hp\Desktop\RID ONLINE\function> node obj2.js
pen
panipuri
0
1
2
PS C:\Users\hp\Desktop\RID ONLINE\function> node obj2.js
pen
panipuri
0 : [ '0' ]
1 : [ '1' ]
2 : [ '2' ]
PS C:\Users\hp\Desktop\RID ONLINE\function> node obj2.js
pen
panipuri
0 : panipuri
1 : momos
2 : Burger
PS C:\Users\hp\Desktop\RID ONLINE\function>
```

Example-2:

```
let person={
name:"Sangam",
   age:25,
fav:["red","blue","green"],
marks:{
      maths:50,
      sci:70,
      hindi:90
   }
}

console.log(person)
console.log(person["name"])//Access value in object via key- method 1
console.log(person.name)//Access value in object via key- method 2
console.log(person.fav[2],person.fav[0],person.fav[1])
console.log(person.marks.sci)
console.log(person["marks"]["sci"])
//Nested Objects, Values in an object be in another object
```

Output:

```
{
  name: 'Sangam',
  age: 25,
  fav: [ 'red', 'blue', 'green' ],
  marks: { maths: 50, sci: 70, hindi: 90 }
}
Sangam
Sangam
green red blue
70
70
```

Example-3:

```
const person = {
  name: "Sangam Kumar",
  age: 15,
  address: {
    street: "123 Main St",
    city: "Patna",
    country: "Bihar"
  }};
console.log(person.name);
console.log(person.address.street);
console.log(person.address.city);
```

Output: Sangam Kumar
123 Main St
Patna

Example-4

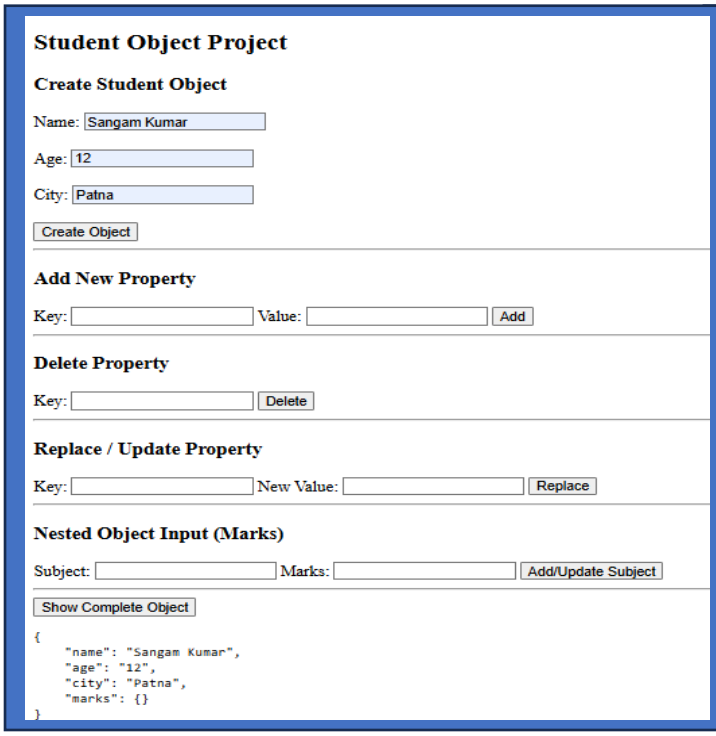
```
const book = {
  title: "JavaScript Basics",
  pages: 250,
  author: {
firstName: "Jane",
lastName: "Smith",
    nationality: "Canadian"
  }};
// Accessing nested object properties
console.log(book.title); // Output: JavaScript Basics
console.log(book.author.firstName); // Output: Jane
console.log(book.author.nationality); // Output: Canadian
```

❖ Nested Arrays and Objects:

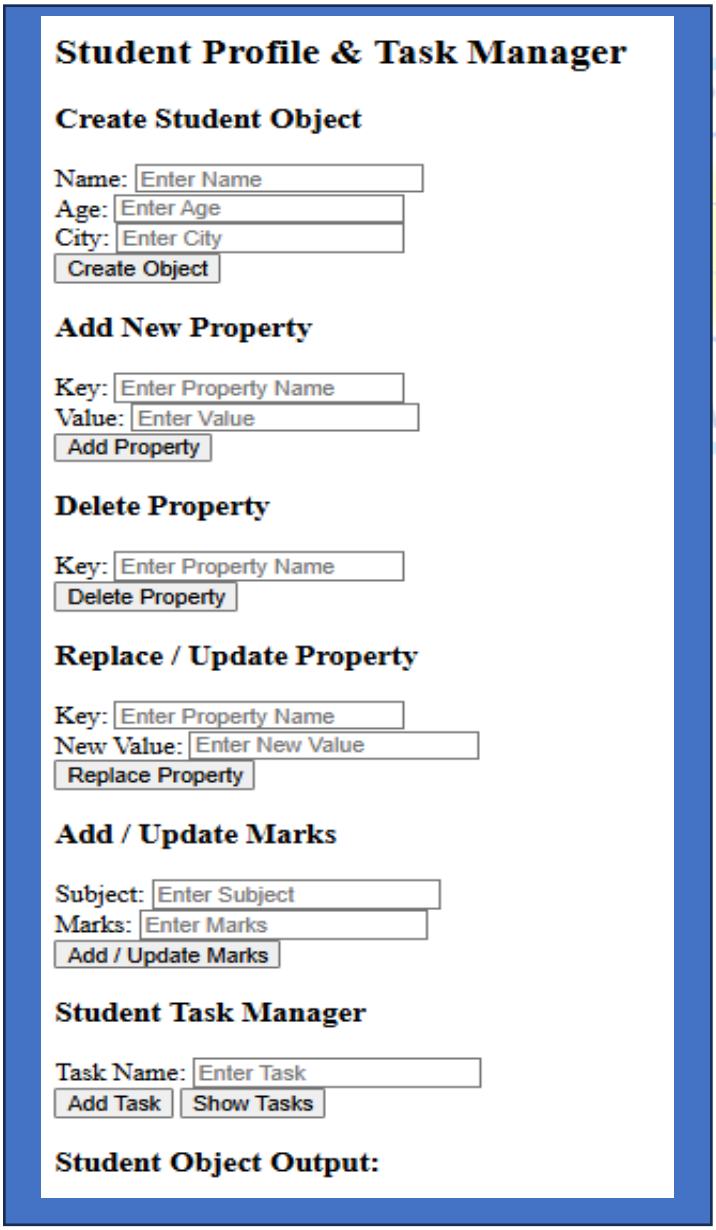
- Values in objects can be arrays, and values in arrays can be objects:

Example:

```
const school = {
  name: "Delhi Public School",
  address: {
    street: "45 MG Road",  city: "Delhi",  zip: "110001"
  },
  classes: [  {
      name: "Math 101",
      teacher: "Mr. Sharma",
      students: [
        { name: "Aarav", age: 14, grade: "A" },
        { name: "Ananya", age: 15, grade: "B" }
      ]   },
    {
      name: "Science 102",
      teacher: "Ms. Mehta",
      students: [
        { name: "Vivaan", age: 14, grade: "A" },
        { name: "Isha", age: 15, grade: "B+" }
      ]  }
  ]  }; // Accessing nested values
console.log("School Name:", school.name); // Delhi Public School
console.log("City:", school.address.city); // Delhi
console.log("First Class Teacher:", school.classes[0].teacher); // Mr. Sharma
console.log("Second Student Name in Math 101:", school.classes[0].students[1].name);
school.classes[1].students.push({ name: "Aryan", age: 16, grade: "A-" });
console.log("Updated Students in Science 102:", school.classes[1].students);
school.classes.forEach((classObj) => {
  console.log(`Class: ${classObj.name}, Teacher: ${classObj.teacher}`);
  classObj.students.forEach((student) => {
    console.log(` - Student: ${student.name}, Grade: ${student.grade}`);    }); });
```




```
<!DOCTYPE html>
<html>
<head> <title>Interactive Object Project</title>
</head> <body>
<h2>Student Object Project</h2>
<h3>Create Student Object</h3> <!-- Create Object -->
Name: <input id="name"><br><br>
Age: <input id="age"><br><br>
City: <input id="city"><br><br>
<button onclick="createObject()">Create Object</button> <hr>
<h3>Add New Property</h3> <!-- Add Property -->
Key: <input id="addKey">
Value: <input id="addValue">
<button onclick="addProperty()">Add</button> <hr>
<h3>Delete Property</h3> <!-- Delete Property -->
Key: <input id="delKey">
<button onclick="deleteProperty()">Delete</button> <hr>
<h3>Replace / Update Property</h3> <!-- Replace/Update Property -->
Key: <input id="replaceKey">
New Value: <input id="replaceValue">
<button onclick="replaceProperty()">Replace</button> <hr>
<h3>Nested Object Input (Marks)</h3> <!-- Nested Object -->
Subject: <input id="subName">
Marks: <input id="subMarks">
<button onclick="addNested()">Add/Update Subject</button><hr>
<button onclick="showObject()">Show Complete Object</button>
<pre id="output"></pre>
```



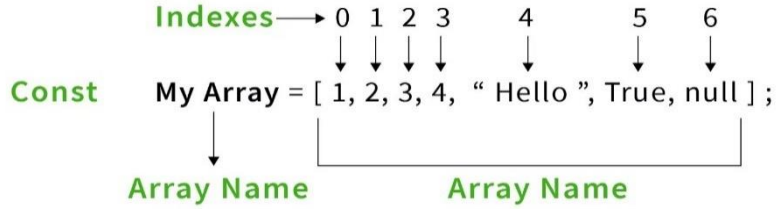

```
<script>
let student = {};   // empty object created
function createObject() { // 1) Create Object from User
    student = {
        name: document.getElementById("name").value,
        age: document.getElementById("age").value,
        city: document.getElementById("city").value,
        marks: {}
    };
    showObject();
}// 2) Add New Property
function addProperty() {
    let key = document.getElementById("addKey").value;
    let value = document.getElementById("addValue").value;
    student[key] = value; // add new key
    showObject();
}// 3) Delete Property
function deleteProperty() {
    let key = document.getElementById("delKey").value;
    delete student[key]; // delete key
    showObject();
}// 4) Replace / Update Property
function replaceProperty() {
    let key = document.getElementById("replaceKey").value;
    let value = document.getElementById("replaceValue").value;
    student[key] = value;
    showObject();
} //5) Add/Update Nested Property (marks)
```

```
function addNested() {
    let subject = document.getElementById("subName").value;
    let marks = document.getElementById("subMarks").value;
    student.marks[subject] = marks;
    showObject();
}// 6) Show Complete Object
function showObject() {
        document.getElementById("output").innerText =
        JSON.stringify(student, null, 4);}
</script>
</body>
</html>
```

ARRAY

1.What is array?

- Array is a non-primitive data type that stores multiple values in a single variable.
- Arrays can hold elements of different data types means homogenous and heterogenous data .
- arrays are indexed, ordered, and mutable, meaning you can add, remove, and modify elements.



2.How to Creating Arrays?

- You can create an array in JavaScript using in this following way.
  1. **Array Literals –** Creates an array using square brackets [].
  2. **Array Constructor –** Creates an array using new Array().
  3. **Array.of –** Creates an array from the provided arguments.
  4. **Array.from –** Creates an array from iterable objects like strings or sets.
  5. **.split(' ') –** Splits a string by spaces into an array of words.

1. Array Literals: - Used to create an array using square brackets [].

- **Syntax:-** let arr = [element1, element2, element3];
- **Example: -**

```
let fruits = ['Apple', 'Banana', 'Mango'];
console.log(fruits); // ['Apple', 'Banana', 'Mango']
let numbers = [1, 2, 3, 4];
console.log(numbers); // [1, 2, 3, 4]
```

2. Array Constructor: - Used to create an array using new Array().

- **Syntax:-** let arr = new Array(element1, element2, ...);
- **Example**

```
let fruits = new Array('Apple', 'Banana', 'Mango');
console.log(fruits); // ['Apple', 'Banana', 'Mango']
let numbers = new Array(5); // creates empty array of length 5
console.log(numbers); // [ <5 empty items> ]
```

3. Array.of : Creates an array from provided arguments. Each argument becomes an array element.

- **Syntax:-** let arr = Array.of(element1, element2, ...);
- **Example**

```
let nums = Array.of(1, 2, 3);
console.log(nums); // [1, 2, 3]
let fruits = Array.of('Apple', 'Banana');
console.log(fruits); // ['Apple', 'Banana']
```

4. Array.from:- Creates an array from **iterable objects** like strings, sets, or array-like objects.

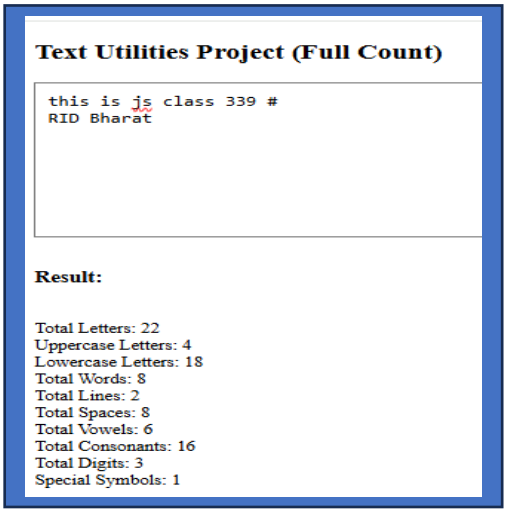
- **Syntax: -** let arr = Array.from(iterable);
- **Example:**

```
let chars = Array.from('Mohan');
console.log(chars); // ['M', 'o', 'h', 'a', 'n']
let nums = Array.from([1,2,3], x => x*2);
console.log(nums); // [2, 4, 6]
```

5. .split(' '):- Splits a string at **every space**, creating an array of words.

- **Syntax:** let arr = string.split(' ');
- **Example:**

```
let str = 'this is Mohan';
let words = str.split(' ');   console.log(words); // ['this', 'is', 'Mohan']
```



1. Regular Expressions (Regex)

Regular expressions are patterns used to **match character combinations** in strings.

Used in the project:

1. **/[a-zA-Z]/g** → Matches all **letters** (upper + lower case).
2. **/[A-Z]/g** → Matches all **uppercase letters**.
3. **/[a-z]/g** → Matches all **lowercase letters**.
4. **/[aeiouAEIOU]/g** → Matches all **vowels**.
5. **/[bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ]/g** → Matches all **consonants**.
6. **/\d/g** → Matches all **digits** (0–9).
7. **/ /g** → Matches all **spaces**.
8. **/[^a-zA-Z0-9\s]/g** → Matches all **special symbols** (anything not a letter, digit, or space).

match()

- Returns an array of all matches of a regex pattern in a string.
- Ex: "hello123".match(/\d/g);
- Output: ['1','2','3']

split()

- Splits a string into an array based on a separator.
- Ex: "this is js".split(" ");
- Output: ['this', 'is', 'js']
- "apple orange".match(/[aeiou]/g); Output: ['a','e','o','a','e']

length

- Counts number of elements in an array or characters in a string.
- [1,2,3].length; // Output: 5

addEventListener('input', function)

- Allows **live updating** as the user types in the textarea.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Text Utilities Project (Full Count)</title>
    <style>
        textarea {
            width: 100%;
            height: 150px;
            padding: 10px;
            font-size: 16px;
            margin-bottom: 10px;
        }
    </style>
</head>
<body>
<h2>Text Utilities Project (Full Count)</h2>
<textarea id="d1" placeholder="write paragraph here..."></textarea>
<h3>Result:</h3>
<div id="res" class="result"></div>
<script>
const textInput = document.getElementById("d1");
textInput.addEventListener("input", analyzeText);
function analyzeText() {
    const text = textInput.value;
    // Total letters (upper + lower case)
    const lettersArray = text.match(/[a-zA-Z]/g) || [];
    const numLetters = lettersArray.length;
    // Uppercase letters
    const upperCaseArray = text.match(/[A-Z]/g) || [];
    const numUpperCase = upperCaseArray.length;
    // Lowercase letters
    const lowerCaseArray = text.match(/[a-z]/g) || [];
    const numLowerCase = lowerCaseArray.length;
    // Words
    const wordsArray = text.trim().split(' ');   //.split(/\s+/)
    const numWords = wordsArray.length;
    // Lines
    const linesArray = text.split(/\n/);
    const numLines = linesArray.length;
    // Spaces
    const numSpaces = (text.match(/ /g) || []).length;
    // Vowels
    const vowelsArray = text.match(/[aeiouAEIOU]/g) || [];
    const numVowels = vowelsArray.length;
    // Consonants
    const consonantsArray = text.match(/[bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ]/g) || [];
    const numConsonants = consonantsArray.length;
    // Digits
    const digitsArray = text.match(/\d/g) || [];
    const numDigits = digitsArray.length;
    // Special symbols (everything except letters, digits, space)
    const specialArray = text.match(/[^a-zA-Z0-9\s]/g) || [];
    const numSpecial = specialArray.length;
    // Show result
    document.getElementById("res").innerText = `
Total Letters: ${numLetters}
Uppercase Letters: ${numUpperCase}
Lowercase Letters: ${numLowerCase}
Total Words: ${numWords}
Total Lines: ${numLines}
Total Spaces: ${numSpaces}
Total Vowels: ${numVowels}
Total Consonants: ${numConsonants}
Total Digits: ${numDigits}
Special Symbols: ${numSpecial}
`;
}
</script></body></html>
```

3.How to access data from the array

- **These following ways to access data from an array**

1. **Using Index**: - accessing array elements directly using square brackets [] and index.
2. **Using Loops** (for, for...of, for in)
   2.1 **for: -** it is used to access array elements using index, find length of array using length
   2.2 **for...of: -**it is used to directly access the **values** of array elements one by one.
   2.3 **for...in: -**it is used to access the **indexes (keys)** of array elements.
   2.4 **While:** - It is used to access array elements repeatedly using a condition.
   2.5 **for each:** It is used to execute a function for each array element and directly access values
3. **slice (): -** it is used to extract a part of a string or array without changing the original one.
4. **Using Array.find :-** it is used to get the **first element** from an array that matches a given condition
5. **Using Array.filter:-** it is used to get **all elements** from an array that match a given condition.

1. Using Index:- Accessing array elements directly using square brackets [] and index.

- **Syntax:** array[index]
- **Example:**
```
const arr = [10, 20, 30];
console.log(arr[2]);
```
Output: 30

2.1 for Loop: - It is used to access array elements using index and find length of array using length.

- **Syntax:**
```
for (let i = 0; i < array.length; i++) {
  // array[i]
}
```
- **Example:**
```
const arr = [10, 20, 30];
for (let i = 0; i < arr.length; i++)
{ console.log(arr[i]);
}
```

Example: arr.join(" ") prints all elements **horizontally in one line** separated by spaces
```
const arr = [10, 20, 30];
for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
console.log(arr.join(" "));
```

2.2 for...of Loop:- It is used to directly access the values of array elements one by one.

- **Syntax:**
```
for (let value of array) {
// value
}
```
- **Example:**
```
const arr = [10, 20, 30];
for (let value of arr) { console.log(value); }
```

2.3 for...in Loop:- It is used to access the indexes (keys) of array elements.

- **Syntax:**
```
for (let index in array) {
  // array[index]
}
```
- **Example:**
```
const arr = [10, 20, 30];
for (let index in arr) { console.log(index, arr[index]); }
```

2.4 while Loop: - It is used to access array elements repeatedly using a condition.

- **Syntax:**
```
let i = 0;
while (i < array.length) {
  // array[i]
i++; }
```

Example:
```
const arr = [10, 20, 30];
let i = 0;
while (i < arr.length) { console.log(arr[i]);
  i++;  }
```

2.5 forEach():- It is used to execute a function for each array element and directly access values.

- **Syntax:**
```
array.forEach(function(value, index) {
  // value   });
```

Example:
```
const arr = [10, 20, 30];
arr.forEach(function(value) {
  console.log(value);
});}
```

3. slice():- It is used to extract a part of a string or array without changing the original one.

- **Syntax:** array.slice(startIndex, endIndex)
- **Example:**
```
const arr = [10, 20, 30, 40];
const result = arr.slice(1, 3);
console.log(result); Output: [20, 30]
```
- **How to remove the [] Method 1: Using join()**
```
const arr = [10, 20, 30, 40];
const result = arr.slice(1, 3);
console.log(result.join(", ")); output: 20, 30
```
Method 2: Using Spread Operator
```
const arr = [10, 20, 30, 40];
const result = arr.slice(1, 3);
console.log(...result);
```

4. Using Array.find(): It is used to get **first element** from an array that matches a given condition.

- **Syntax:**
```
array.find(function(element) {
  // condition
});
```
- **Example:**
```
const arr = [10, 20, 30, 40];
const result = arr.find(num => num > 25);
console.log(result); Output: 30
```

5. Using Array.filter():- It is used to get **all elements** from an array that match a given condition.

- **Syntax:**
```
array.filter(function(element) {
  // condition
});
```
- **Example:**
```
const arr = [10, 20, 30, 40];b v
const result = arr.filter(num => num > 20);
console.log(result);
```

Output: [30, 40]

How to remove the square brackets and print only values

Method 1: Using join():
```
const arr = [10, 20, 30, 40];
const result = arr.filter(num => num > 20);
console.log(result.join(", "));
```
Output: 30, 40

Method 2: Using forEach()
```
const arr = [10, 20, 30, 40];
const result = arr.filter(num => num > 20);
result.forEach(num => {
  console.log(num);
});
```
Output:30
40

Method 3: Using Spread Operator
```
const arr = [10, 20, 30, 40];
const result = arr.filter(num => num > 20);
console.log(...result);
```
Output: 30 40

4.How to add the dynamic data inside array

Arrays are mutable, which means you can change their content by adding, removing.

➢ **We can add the data in following way**

A. **Push():-** method is used for add the elements in **end** of the array array
B. **unshift():-** method is used for add the elements in **beginning** of the array array
C. **[index]=value:-** using add data in specific place it replaces the old value with new value
D. **splice() :-** it is used add multiple elements at a **specific position** (index) in an array.

Note: If you want to add multiple elements at a **specific position**, you cannot directly use push() or unshift().you can use these methods along with **splice()** to insert elements at a specific position.

A. push() Method

- Adds one or more elements to end of the array and returns new length of the array.
- **Syntax**: arrayName.push(element1, element2, ...);
- **Example**:

```
const arr = [10, 20, 30];
arr.push(40, 50);
console.log(arr); // Output: [10, 20, 30, 40, 50]
```

B. unshift() Method

- Adds one or more elements to beginning of the array and returns new length array.
- **Syntax**: arrayName.unshift(element1, element2, ...);
- **Example**:

```
const arr = [10, 20, 30];
arr.unshift(0, 5);
console.log(arr); // Output: [0, 5, 10, 20, 30]
```

Example:

```
const fruits = ["apple", "banana", "cherry"];
console.info("Before adding element in array")
console.log("array=",fruits)
fruits.push("orange"); // Adds "orange" to the end
fruits.unshift("grape"); // Adds "grape" to the beginning
console.info("after adding element in array")
console.log("array=",fruits)
```

Output:

```
Before adding element in array
array= [ 'apple', 'banana', 'cherry' ]
after adding element in array
array= [ 'grape', 'apple', 'banana', 'cherry', 'orange' ]
console.log("array=",fruits
```

C. Updating elements in specific index.

- By using the index value, we can add the element on specific position in array

Syntax: arrayName[index] = newValue;

Example:

```
const arr = [10, 20, 30, 40];
arr[2] = 35; // Update element at index 2 (change 30 to 35)
console.log(arr); // Output: [10, 20, 35, 40]
```

Note 1: Updating an Existing Element

- If the index already exists in the array, the value at that index is **updated**.

Example:

```
const arr = [10, 20, 30];
arr[1] = 25;
console.log(arr); // [10, 25, 30]
```

Note 2: Adding an Element at a New Index

- If index is **greater than array length**, JavaScript **extends array** and adds empty slots.

Example:

```
const arr = [10, 20, 30];
```

```
arr[5] = 40;
console.log(arr);       // [10, 20, 30, <2 empty items>, 40]
console.log(arr.length); // 6
```

Note 3: Adding Elements Indirectly

- Assigning a value to a higher index places the value **only at that index**.
- Missing positions remain **empty**.

Example:

```
const arr = [1, 2];
arr[4] = 10;
console.log(arr); // [1, 2, <2 empty items>, 10]
```

Important Points:

1. The array **grows** when you assign a value to a higher index.
2. Empty positions create a **sparse array**. Direct assignment does **not** add elements step-by-step; it places the value only at the given index.

D). Adding Elements at a Specific Index using splice()

- The splice() method is used to **add elements at any position** in an array.
- It can insert **one or more elements**.

Syntax: array.splice(index, deleteCount, element1, element2, ...);

- **index** → position where elements are added
- **deleteCount** → number of elements to remove (0 means remove nothing)
- **elements** → values to be inserted

Example: Add Elements at a Specific Index

```
const arr = [10, 20, 30, 40];
arr.splice(2, 0, 25, 27);
console.log(arr); // [10, 20, 25, 27, 30, 40]
```

1. Add Elements at the Beginning

- Use index 0.

```
const arr = [30, 40, 50];
arr.splice(0, 0, 10, 20);
console.log(arr); // [10, 20, 30, 40, 50]
```

2. Add Elements in the Middle

- **Use the required index.**

```
const arr = [10, 20, 30, 40, 50];
arr.splice(2, 0, 25, 27);
console.log(arr); // [10, 20, 25, 27, 30, 40, 50]
```

3. Add Elements at the End

- **Use arr.length as the index.**

```
const arr = [10, 20, 30];
arr.splice(arr.length, 0, 40, 50, 60);
console.log(arr); // [10, 20, 30, 40, 50, 60]
```

❖ **Key Points:** splice() can add elements at **start, middle, or end**. Setting deleteCount to 0 means **no elements are removed**.Multiple values can be added at the same time.

5.How to Remove the elements from array.

A. **Pop():-** method is used for **remove** the elements in **end** of the array.
B. **shift():-** method is used for **remove** the elements in **beginning** of the array.
C. **splice():-** Removes elements at a specific position and can remove multiple elements.
D. **Delete:-** Deletes an element at a specific index, leaving an empty slot (undefined).

A. pop() Method:

- Removes last element from array and returns it. This method modifies original array.
- **Note:** Removes one element at a time. Use a loop or splice() for multiple removals.
- **Syntax**: arrayName.pop();
- **Example**: const arr = [10, 20, 30, 40];

```
const removedElement = arr.pop();
console.log(arr); // Output: [10, 20, 30]
console.log(removedElement); // Output: 40
```

B. shift() Method

- Removes first element from array and returns it. This method modifies original array.
- **Note**: it can only remove **one element at a time** from the beginning of the array.
- **Syntax**: arrayName.shift();
- **Example**: const arr = [10, 20, 30, 40];

```
const removedElement = arr.shift();
console.log(arr); // Output: [20, 30, 40]
console.log(removedElement); // Output: 10
```

Example:

```
const fruits = ["apple", "banana", "cherry"];
console.info("Before remove element in array")
console.log("array=",fruits)
fruits.pop("orange"); // remove "orange" to the end
fruits.shift("grape"); // remove "grape" to the beginning
console.info("after remove element in array")
console.log("array=",fruits)
```

Outout:

```
Before remove element in array
array= [ 'apple', 'banana', 'cherry' ]
after remove element in array
array= [ 'banana' ]
```

C. splice ():- Removes elements at a specific position and can remove multiple elements.

- **Syntax**: array.splice(index, deleteCount);
  - ✓ **index:** The position where removal starts.
  - ✓ **deleteCount:** The number of elements to remove.
- **Example-1**:

```
const arr = [10, 20, 30, 40];
arr.splice(2, 1); // Removes the element at index 2
console.log(arr); // Output: [10, 20, 40]
```

Example -2: Removing Multiple Consecutive Elements

```
const arr = [10, 20, 30, 40, 50, 60]; // Remove 3 elements starting from index 2 (30, 40, 50)
arr.splice(2, 3);
console.log(arr); // Output: [10, 20, 60]
```

Example-3: Removing Multiple Elements from the End

```
const arr = [100, 200, 300, 400, 500]; // Remove 2 elements starting from index 3 (400, 500)
arr.splice(3, 2);
console.log(arr); // Output: [100, 200, 300]
```

splice(2, 3):
- ✓ **2** is the index where the removal starts.
- ✓ **3** is the number of elements to remove (30, 40, 50).

splice(3, 2):

- **3** is the index to start removing from (starting at 400).
- **2** is the number of elements to remove (400, 500).

Example-4: Removing More Elements Than Exist in the Array

```
const arr = [10, 20, 30];
// Try to remove 5 elements starting from index 1 (only 2 elements left after index 1)
arr.splice(1, 5);
console.log(arr); // Output: [10]
```

splice(1, 5):

- **1** is the index where removal starts (from 20).
- **5** specifies that we want to remove 5 elements, but there are only 2 elements left (20, 30). So, only those 2 elements will be removed.

D.delete:- Deletes an element at a specific index, leaving an empty slot (undefined).

- **Syntax**: delete array[index];
- **Example**:

```
const arr = [10, 20, 30, 40];
delete arr[2]; // Removes the element at index 2
console.log(arr); // Output: [10, 20, <1 empty item>, 40]
```

Difference Between `delete` and `splice()`

Feature	`delete`	`splice()`
Purpose	Removes an element at a specific index from an array.	Removes or adds elements at a specific index in an array.
Effect on Array	Leaves an empty slot (`undefined`) at the removed index.	Removes elements and shifts the rest of the array to fill the gap.
Return Value	`undefined`	The removed elements as an array.
Array Length	Does not change the array's length; only creates empty slots.	Changes the array's length by removing or adding elements.
Usage	Primarily for removing an element without affecting other elements.	Primarily for modifying the array by removing or adding elements.

Example 1: Using delete

- delete removes the value **but does not change the array size**.
- It leaves an **empty slot** at that index.

```
const arr = [10, 20, 30, 40];
delete arr[2];
console.log(arr);       // [10, 20, <empty>, 40]
console.log(arr.length); // 4
```

Key Point: Array length stays the same. Elements are **not shifted**.

Example 2: Using splice()

- splice() removes the element **and shifts remaining elements**.
- The array **length is updated**.

```
const arr = [10, 20, 30, 40];
arr.splice(2, 1);
console.log(arr);       // [10, 20, 40]
console.log(arr.length); // 3
```

Key Point: Element is removed completely. Array size becomes smaller.

- **delete** → leaves empty space, length unchanged
- **splice()** → removes element properly, length change

6.Mathematical method in array

1. **reduce()** → sum, product
2. **map()** → square, cube, power
3. **filter()** → even, odd, positive, negative
4. **sort()** → ascending, descending
5. **reverse()**
6. **Math.max()**
7. **Math.min()**

➢ **reduce()** → Performs a function on array elements to return a single value (sum, product, etc.).

Syntax: array.reduce((accumulator, current) => operation, initialValue)

➢ **Example (Sum):**

```
const arr = [10, 20, 30];
const sum = arr.reduce((a, b) => a + b, 0);
console.log(sum); // 60
```

- **a** → **accumulator**: it stores the **accumulated value** from the previous iteration.
- **b** → **current**: it is the **current element** of the array being processed.

➢ **Step by Step:**

1. **Initial value of a** = 0 (because of the second argument in reduce()).
2. Iteration 1: a = 0, b = 10 → a + b = 10
3. Iteration 2: a = 10, b = 20 → a + b = 30
4. Iteration 3: a = 30, b = 30 → a + b = 60

Final result: 60

- a = accumulated value so far
- b = current element of the array

Example (Product):

```
const product = arr.reduce((a, b) => a * b, 1);
console.log(product); // 6000
```

2. map() → Creates a new array by applying a function to each element.

- **Syntax:** array.map(element => operation)
- **Example (Square):**

```
const arr = [2, 3, 4];
console.log(arr.map(n => n * n)); // [4, 9, 16]
```

- **Example (Cube):** console.log(arr.map(n => n ** 3)); // [8, 27, 64]

3. filter() → Creates a new array containing elements that pass a condition.

- **Syntax:** array.filter(element => condition)
- **Example (Even Numbers):**

```
const arr = [1, 2, 3, 4];
console.log(arr.filter(n => n % 2 === 0)); // [2, 4]
```

- **Example (Odd Numbers):** console.log(arr.filter(n => n % 2 !== 0)); // [1, 3]

4. sort() →Sorts array elements numerically.

Syntax:

```
array.sort((a, b) => a - b) // ascending
array.sort((a, b) => b - a) // descending
```

- **a and b** are **two elements from the array** that the sort() method compares during sorting.
- The function (a, b) => a - b is called a **compare function**.

How it works:

1. **If the result is negative (a - b < 0)** → a comes **before** b.
2. **If the result is positive (a - b > 0)** → b comes **before** a.
3. **If the result is 0** → positions of a and b remain unchanged.

Example:
```
const arr = [30, 10, 20];
arr.sort((a, b) => a - b); // Ascending
console.log(arr); // [10, 20, 30]
arr.sort((a, b) => b - a); // Descending
console.log(arr); // [30, 20, 10]
```

5. reverse():- Reverses the order of elements in an array.

- **Syntax:** array.reverse()
- **Example:**
```
const arr = [10, 20, 30];
console.log(arr.reverse()); // [30, 20, 10]
```

6. Math.max():- Returns the largest number from an array.

- **Syntax:** Math.max(...array)
- **Example:**
```
const arr = [10, 20, 30];
console.log(Math.max(...arr)); // 30
```

7.Math.min():- Returns the smallest number from an array.

- **Syntax:** Math.min(...array)
- **Example:**
```
const arr = [10, 20, 30];
console.log(Math.min(...arr)); // 10
```

7.Special methods in Array

1. join(): - Joins all elements of an array into a string, separated by a specified delimiter.

- **Syntax:** array.join(separator)
- **Example:**
```
const arr = [10, 20, 30];
console.log(arr.join(", ")); Output: 10, 20, 30
```

2. includes ():- Checks if an array contains a specified element. Returns true or false.

- **Syntax:** array.includes(element)
- **Example:**
```
const arr = [10, 20, 30];
console.log(arr.includes(20)); Output: true
```

3. indexOf():- Returns index of the first occurrence of a specified element. Returns -1 if not found.

- **Syntax:** array.indexOf(element)
- **Example:**
```
const arr = [10, 20, 30];
console.log(arr.indexOf(20)); Output: 1
```

4. lastIndexOf():- Returns the index of the last occurrence of a specified element.

- **Syntax:** array.lastIndexOf(element)
- **Example:**
```
const arr = [10, 20, 30, 20];
console.log(arr.lastIndexOf(20)); Output: 3
```

5. toString() :- Converts an array to a comma-separated string.

- **Syntax:** array.toString()
- **Example:**
```
const arr = [10, 20, 30];
console.log(arr.toString()); Output: 10,20,30
```

6. reverse():- Reverses the order of elements in an array.

- **Syntax:** array.reverse()
- **Example:**
```
const arr = [10, 20, 30];
console.log(arr.reverse()); Output: [30, 20, 10]
```

7. concat() :- Joins two or more arrays into one.

- **Syntax:** array1.concat(array2, array3, ...)

Example:
```
const a = [1, 2];
const b = [3, 4];
console.log(a.concat(b));
```
Output: [1, 2, 3, 4]

STRING

- A string is a sequence of characters used to represent text.
- Strings are enclosed within single quotes (' '), double quotes (" "), or backticks (` `).
- " " is for single-line strings only, whereas ` ` supports multi-line strings and ${} interpolation.
- Strings are **immutable**, meaning once created, their content cannot be changed.
- Each character in a string is stored at a specific **index**, starting from index 0.
- Strings support many built-in methods like length, toUpperCase(), and slice().
- Strings are a **primitive data type** in JSt and widely used to store names, messages, and text data.
- **Double quotes (" ") do not allow multi-line strings**, while **backticks (``) allow multi-line strings** without using special characters.

Example-1:
```
let str1 = 'Research, Innovation!& Discovery';
let str2 = "JavaScript is awesome!";
```

Example-2:
```
let Name1 = "wit"  // Double quotes
let Name2 = 'RID'  // Single quotes
```

➢ You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example-3:
```
let answer1 = "It's alright";
let answer2 = "He is called 'Hi'"
let answer3 = 'He is called "Hello"'
```

Example-4: The string is enclosed using **backticks ( )**.

- Backticks are used for **template literals** in JS.
- They allow multi-line strings and variable interpolation using ${}.

Example:
```
let org = "RID Bharat";
let message = `Welcome to ${org}.
We focus on Research, Innovation,
and Discovery.`;
console.log(message);
```

❖ Escape Character:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash
\b		Backspace
\n		New Line
\t		Horizontal Tabulator

Example:

1. **Single Quote (\'):**Used to include a single quote within a string delimited by single quotes.
   ```
   let a= 'He said \'Hello\'';
   console.log(a);
   ```
   Output: He said 'Hello'
2. **Double Quote (\"):**Used to include double quotes within a string delimited by double quotes.
   ```
   let a = "She said \"Hi\"";
   console.log(a);
   ```
   Output: She said "Hi"
3. **Backslash (\\):**Used to include a backslash character in a string.
   ```
   let a = 'This is a backslash: \\';
   console.log(a);
   ```
   Output: This is a backslash: \
4. **Backspace (\b):**Moves the cursor one character back.
   ```
   let b = 'Hello\bWorld';
   console.log(b);
   ```
   Output: HellWorld
5. **New Line (\n):** Inserts a new line.
   ```
   let n = 'Hello\nWorld';
   console.log(n);
   ```
   Output: Hello
   World
6. **Horizontal Tabulator (\t):**Inserts a horizontal tab.
   ```
   let h = 'Hello\tWorld';
   console.log(h);
   ```
   Output: Hello World

❖ String Methods:

1) **String length** – Used to find the total number of characters in a string.
2) **String slice()** – Used to extract a part of a string using start and end index.
3) **String substring()** – Used to get characters between two index positions (no negative index).
4) **String substr()** – Used to extract characters from a string starting at a position with given length.
5) **String replace()** – Used to replace the first matching value in a string.
6) **String replaceAll()** – Used to replace all matching values in a string.
7) **String toUpperCase()** – Used to convert all characters of a string into uppercase.
8) **String toLowerCase()** – Used to convert all characters of a string into lowercase.
9) **String concat()** – Used to join two or more strings together.
10) **String trim()** – Used to remove spaces from both start and end of a string.
11) **String trimStart()** – Used to remove spaces only from the beginning of a string.
12) **String trimEnd()** – Used to remove spaces only from the end of a string.
13) **String padStart()** – Used to add characters at beginning of a string until it reaches a given length.
14) **String padEnd()** – Used to add characters at the end of a string until it reaches a given length.
15) **String charAt()** – Used to get the character at a specific index position.
16) **String charCodeAt()** – Used to get the Unicode value of a character at a given index.
17) **String split()** – Used to convert a string into an array based on a separator.

1. String length (length property): Returns the length of the string.

Syntax: string.length

Example:
```
const str = "Research, Innovation";
console.log("lenght of string=",str.length);
```
Output: lenght of string= 20

2. String slice(start, end): Extracts a portion of the string and returns it as a new string.

- Extracts a portion of the string from start **(inclusive)** to end **(exclusive). By default endindex will last**
- Supports negative indices (counts from the end of the string if negative).
- Does not swap start and end if start > end.

Syntax: string.slice(startIndex, endIndex)

Example:
```
let str = "RidBharat";
console.log(str.slice(0, 4));
console.log(str.slice(2, -3));
console.log(str.slice(2,3));
console.log(str.slice(7, 4));
```
Output:
```
RidB
dBha
d
<empy-string>
```

Ex-1
```
let str = "ridbharat";
console.log(str[-1]); // Output: undefined (negative indexing is not supported)
```
Ex-2
```
let str = "ridbhart";
console.log(str.slice(-1)); // Output: "t" (last character)
console.log(str.slice(-4, -1)); // Output: "har" (4th to last to 2nd to last character)
```

```
let sentence = "JavaScript is a versatile language.";
Extracting the word "versatile"
let word = sentence.slice(16, 25);
console.log(word); Output: "versatile"
Using negative indices to extract the word "language"
let wordFromEnd = sentence.slice(-9, -1);
console.log(wordFromEnd); Output: "language"
Extracting the substring "JavaScript" using only the start index
let startOnly = sentence.slice(0);
console.log(startOnly); Output: "JavaScript is a versatile language."
```

3. **String substring(startindx, endinx):**

- Extracts a portion of the string from start (inclusive) to end (exclusive).
- **Does not support negative indices**; negative values are treated as 0.
- Automatically swaps start and end if start > end.

Syntax: string.substring(startIndex, endIndex)

Example:

```
let str = "JavaScript";
console.log(str.substring(0, 4)); Output: "Java"
console.log(str.substring(4, 0)); Output: "Java" (start and end swapped)
console.log(str.substring(4, -3)); Output: "Java" (-3 treated as 0)
```

Output: Java
Java
Java

Example: let quote = "The quick brown fox jumps over the lazy dog";

Extracting the word "quick"

```
let word1 = quote.substring(4, 9);
console.log(word1); Output: "quick"
```

Extracting from the start when `end` is omitted

```
let fromStart = quote.substring(10);
console.log(fromStart); Output: "brown fox jumps over the lazy dog"
```

Swapping `start` and `end` automatically

```
let swapped = quote.substring(9, 4);
console.log(swapped); Output: "quick" (start and end are swapped)
```

Using negative index (treated as 0)

```
let negativeIndex = quote.substring(-5, 9);
console.log(negativeIndex); Output: "The quick" (-5 is treated as 0)
```

Extracting the word "lazy"

```
let word2 = quote.substring(35, 39);
console.log(word2); // Output: "lazy"
```

Key Differences:

Feature	`slice(start, end)`	`substring(start, end)`
Negative Indices	Supported	Treated as `0`
Swaps `start` & `end`	No	Yes
Behavior When `start > end`	Returns empty string	Swaps the indices

4. String substr(start, length):

- Extracts a specified number of characters from a string, starting at the specified index.

Syntax: string.substr(startIndex, length)

Example-1:

```
const str = "Hello, World!";
const s = str.substr(7, 5);
console.log(s);
```

Output: "World"

Example-2:

```
const text = "ResearchInnovation";
const result = text.substr(8, 10);
console.log(result);
```

Output: Innovation

5. String replace(oldValue, newValue):- Replaces the first occurrence of oldValue with newValue.

Syntax: string.replace(searchValue, replaceValue)

Example:
```
let a="RID BHARAT BHOPAL"
B=a.replace("BHOPAL", "Patna")
console.log(B)
```

Output: RID BHARAT Patna

6. String replaceAll(oldValue, newValue) (Available in ES2021 and later):

- Replaces all occurrences of oldValue with newValue.

Syntax: string.replaceAll(searchValue, replaceValue)

Example:
```
const s="abababbababaababababaab"
let ns=s.replaceAll("a","A")
console.log(ns)
```

Output: AbAbAbbAbAbAAbAbAAb

7. String toUpperCase():

- The toUpperCase() method converts all the characters in a string to uppercase. It does not modify the original string; instead, it returns a new string with all letters converted to uppercase.

Syntax: string.toUpperCase()

Example:
```
const s="rid bharat bhopal"
let ns=s.toUpperCase()
console.log(ns)
```

Output: RID BHARAT BHOPAL

8. String toLowerCase():

- toLowerCase() method converts all the characters in a string to lowercase. Like toUpperCase(), it does not modify the original string but returns a new string with all letters converted to lowercase.

Syntax: string.toLowerCase()

Example:
```
const s="RID BHARAT BHOPAL"
let ns=s.toLowerCase()
console.log(ns)
```

Output: rid bharat Bhopal

Note-1: How to write first letter of a string in capital letter remaing should be lowerCase

Example:
```
function CFL(s) {
    if (!s) {
       return s;
    }
    else{
return s.charAt(0).toUpperCase() + s.slice(1).toLowerCase();
    }}
const user_str= "rid bharat Bhopal";
const res = CFL(user_str);
console.log(res);
```
Output: Rid bharat bhopal

charAt()

- charAt() method is used to retrieve the character at a specified index in a string. The index is zero-based, meaning the first character is at index 0.

Syntax: string.charAt(index)

Example
```
const str = "JavaScript";
const firstChar = str.charAt(0);
// Gets the first character
const fourthChar = str.charAt(3);
// Gets the fourth character
```

Note-2: How to write each first letter of a word in a string in capital letter remains should be lowerCase

Example:
```
function capitalizeWords(str) {
  const words = str.split(' ').map(word => word[0]?.toUpperCase() + word.slice(1).toLowerCase());
  return words.join(' ');
} console.log(capitalizeWords("hello, WORLD! welcome to JAVASCRIPT."));
```

Output: "Hello, World! Welcome To Javascript."

Explanation of the Simplified Code:

1. str.split(' '): Splits the string into an array of words.
2. map(): For each word:
   - word[0]?.toUpperCase(): Converts the first character to uppercase. The ?. ensures it works even if the word is empty.
   - word.slice(1).toLowerCase(): Converts the rest of the word to lowercase.
3. join(' '): Combines the words back into a single string with spaces.

2nd Method:

```
function capitalizeWords(str) {
  const words = str.split(' ').map(word => {
    const firstChar = word[0]?.toUpperCase();
    const restChars = word.slice(1).toLowerCase();
    return firstChar + restChars;
  });
  return words.join(' ');
} console.log(capitalizeWords("hello, WORLD! welcome to JAVASCRIPT."));
```

Note: If you remove the ?, the function will still work correctly, provided there are no empty strings in the array or edge cases.

9. String concat(): Combines one or more strings and returns a new string.

Syntax: string.concat(string1, string2, ..., stringN)

Example:
```
const str1 = "Hello, ";
const str2 = "World!";
const combinedStr = str1.concat(str2);
console.log(combinedStr);
```
Output: "Hello, World!"

10. String trim(): Removes whitespace from the beginning and end of the string.

Syntax: string.trim()

Example:
```
const str = "   Hello, World!   ";
const trimmedStr = str.trim();
console.log("String trim:", trimmedStr);
```
Output: "Hello, World!"

11. String trimStart() or String trimLeft(): Removes whitespace from beginning of string.

Syntax: string.trimStart()

Example:
```
const str = "   Hello, World!   ";
const trimmedStartStr = str.trimStart();
console.log("String trimStart:", trimmedStartStr);
```
Output: "Hello, World!   "

12. String trimEnd() or String trimRight(): Removes whitespace from the end of the string.

Syntax: string.trimEnd()

Example:
```
const str = "   Hello, World!   ";
consttrimmedEndStr = str.trimEnd();
console.log("String trimEnd:", trimmedEndStr);
```
Output: "   Hello, World!"

❖ How to replace the space between the world as well as letter

1. Using replaceAll() (ES2021+)

- **replaceAll() method replaces all occurrences of a substring or pattern in the string.**

Example:
```
let str = "H e l l o   W o r l d";
let result = str.replaceAll(" ", ""); // Replace all spaces
console.log(result);
```
Output: "HelloWorld"

2. Remove the space between word

Example:
```
let str = "This   is a   string   with   multiple   spaces.";
```

Split the string into an array of words → let wordsArray = str.split(' ');
Filter out empty strings→ let filteredArray = wordsArray.filter(word => word !== '');
Join the array back into a string with a single space between words

let newStr = filteredArray.join(' ');
console.log(newStr); **Output:** "This is a string with multiple spaces."

13. String padStart(targetLength, padString):

- Pads the string with a specified character (or space) until it reaches the desired length.

Syntax: string.padStart(targetLength, padString)

Example: const str = "5";
const paddedStartStr = str.padStart(3, "0");
console.log("String padStart:", paddedStartStr); **Output:** "005"

14. String padEnd(targetLength, padString):

- Pads string from the end with a specified character (or space) until it reaches the desired length.

Syntax: string.padEnd(targetLength, padString)

Example: const str = "5";
const paddedEndStr = str.padEnd(3, "0");
console.log("String padEnd:", paddedEndStr); **Output:** "500"

15. String charAt(index): Returns the character at the specified index. charAt is useful for accessing individual characters

Syntax: string.charAt(index)

Example: const str = "Hello, World!";
const char = str.charAt(7);
console.log("String charAt:", char); **Output:** "W"

16. String charCodeAt(index):

- Returns the Unicode value (integer) of the character at the specified index.

Syntax: string.charCodeAt(index)

Example: const str = "Hello, World!";
constcharCode = str.charCodeAt(7);
console.log("String charCodeAt:", charCode); **Output:** 87

17. String split(separator):

- Splits the string into an array of substrings based on the specified separator.

Syntax: string.split(separator, limit)

Example: const str = "apple,banana,cherry";
const fruits = str.split(",");
console.log("String split:", fruits); **Output:** ["apple", "banana", "cherry"]

Example 2: Splitting a URL into Components
const url = "https://www.example.com/path/to/page";
const components = url.split("/");
console.log("String split:", components);
Output: ["https:", "", "www.example.com", "path", "to", "page"]

Example 1: Splitting a Sentence into Words
const sentence = "The quick brown fox jumps over the lazy dog";
const words = sentence.split(" ");
console.log("String split:", words);
Output: ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]

This Keyword

- **this** is a special keyword that refers to the **current object**.
- Its value depends on **how a function is called**.

Uses of this:

- In an **object method** → this refers to the object
- In the **global scope** → this refers to the global object
- In a **normal function** → this refers to the global object
- In **strict mode** → this is undefined
- In an **event** → this refers to the element that triggered the event
- Using **call(), apply(), bind()** → this can refer to any object

1. Global Context:- Outside any function or object, this refers to the global object.

```
console.log(this === window); // true (Browser)
```

2. Function Context :- In a normal function, this depends on how the function is called.

```
function greet() {
  console.log(this.name);
}
const person = { name: "Sangam" };
person.greet = greet;
person.greet(); // Sangam
greet(); // undefined
```

3. Arrow Functions:- Arrow functions **do not have their own this**.
They use this from their surrounding scope.

```
const person = {
  name: "Sangam",
  greet() {
    setTimeout(() => {
      console.log(this.name);
    }, 1000);
  }
};
person.greet(); // Sagam
```

4. Constructor Function:- With new, this refers to the **new object created**.

```
function Person(name) {
  this.name = name;
}
const p = new Person("Raj");
console.log(p.name); // Raj
```

5. Event Handler:- In events, this refers to the **HTML element**.

```
button.onclick = function () {
  console.log(this);
};
```

6. Strict Mode:- In strict mode, this in a function is undefined.

```
"use strict";
function test() {
  console.log(this);
}
test(); // undefined
```

HOISTING

Hoisting means **JavaScript moves declarations to the top** before execution.

Variable Hoisting

var

```
console.log(a); // undefined
var a = 10;
```

let and const

```
console.log(a); // Error
let a = 5;
```

Function Hoisting

Function Declaration (Hoisted)

```
hello();
function hello() {
  console.log("Hello");
}
```

Function Expression (Not Hoisted)

```
test(); // Error
var test = function () {
  console.log("Hi");
};
```

Key Points

- this depends on **how a function is called**
- Arrow functions **inherit this**
- var is hoisted (undefined)
- let and const are hoisted but **not accessible**
- Function declarations are **fully hoisted**

DATE OBJECT

Date object is used to work with dates and times. It allows you to represent and manipulate dates, perform date arithmetic, and format dates for display. It helps us **get**, **set**, and **format** date and time values like day, month, year, hours, minutes, etc.

❖ Get Date Parts

- **getDate() →** returns day (1–31)
- **getMonth() →** returns month (0–11)
- **getFullYear() →** returns year (YYYY)

Note: Month starts from **0** (January = 0)

Example:

```
let d = new Date("2023-09-30");
console.log(d.getDate());     // 30
console.log(d.getMonth()+1);  // 9
console.log(d.getFullYear()); // 2023
```

Note: - JavaScript does NOT support Indian date format (dd-mm-yyyy) directly

JavaScript understands only:

- **yyyy-mm-dd**
- **mm/dd/yyyy**

- JavaScript months start from 0, so we subtract 1 while setting month and add 1 while getting month.

❖ How to convert Indian format

Example:

```
let input = "25-12-2025";
let parts = input.split("-");    // ["17","12","2025"]
let d = new Date(parts[2], parts[1]-1, parts[0]);
console.log(d)
console.log(d.getDate());        // 17
console.log(d.getMonth()+1);     // 12
console.log(d.getFullYear());
```

```
PROBLEMS 2   DEBUG CONSOLE   TERMINAL

PS C:\Users\hp\OneDrive\Desktop\Javascript> node "c:\Users\hp\OneDrive\Desktop\Javascrip
2025-12-24T18:30:00.000Z
25
12
2025
```

Example:

```
let input = "25-12-2025";
let parts = input.split("-");    // ["17","12","2025"]
let d = new Date(parts[2], parts[1]-1, parts[0]);
console.log(d)
console.log(d.getDate());      // 17
console.log(d.getMonth()+1);   // 12
console.log(d.getFullYear());
```

Why parts[1] - 1 and getMonth() + 1?

In JavaScript, **months start from 0** (January = 0, December = 11).

parts[1] - 1

- Indian date uses months **1–12**
- JavaScript needs months **0–11**
- So we subtract **1** when setting the month

parts[1] - 1 // 12 → 11

getMonth() + 1

- getMonth() returns **0–11**
- Humans understand **1–12**
- So we add **1** while displaying

d.getMonth() + 1 // 11 → 12

- **Minus 1 when setting month**
- **Plus 1 when getting month**

❖ Get Time Parts

- getHours() → hours (0–23)
- getMinutes() → minutes (0–59)
- getSeconds() → seconds (0–59)
- getMilliseconds() → milliseconds (0–999)

Example:

```
let now = new Date();
console.log(now.getHours());
console.log(now.getMinutes());
console.log(now.getSeconds());
```

❖ How to check the AM & PM

```
let now = new Date();  // create Date object
let hours = now.getHours();
let ampm = hours >= 12 ? "PM" : "AM";
hours = hours % 12 || 12;
console.log(hours, ampm);
```

How to print day name using getDay()

```
// Create Date object
let now = new Date();
let dayNum = now.getDay(); // Get day
// Convert number to day name
let dayName;
if (dayNum === 0) {
  dayName = "Sunday";
} else if (dayNum === 1) {
  dayName = "Monday";
} else if (dayNum === 2) {
  dayName = "Tuesday";
} else if (dayNum === 3) {
  dayName = "Wednesday";
} else if (dayNum === 4) {
  dayName = "Thursday";
} else if (dayNum === 5) {
  dayName = "Friday";
} else {
  dayName = "Saturday";
}
// Time with AM/PM
let hours = now.getHours();
let ampm = hours >= 12 ? "PM" : "AM";
hours = hours % 12 || 12;
// Output
console.log("Day Number:", dayNum);
console.log("Day Name:", dayName);
console.log("Time:", hours, ampm);
```

Create Date object

```
let now = new Date();
let dayNum = now.getDay(); // Get day number (0–6)
```

Convert number to day name

```
let dayName;
if (dayNum === 0) {dayName = "Sunday";}
else if (dayNum === 1) {dayName = "Monday";}
else if (dayNum === 2) {dayName = "Tuesday";}
else if (dayNum === 3) {dayName = "Wednesday";}
else if (dayNum === 4) {dayName = "Thursday";}
else if (dayNum === 5) {dayName = "Friday";}
else {dayName = "Saturday";}
```

Time with AM/PM

```
let hours = now.getHours();
let ampm = hours >= 12 ? "PM" : "AM";
hours = hours % 12 || 12;
```

Output

```
console.log("Day Number:", dayNum);
console.log("Day Name:", dayName);
console.log("Time:", hours, ampm)
```

Time: 2:59:57

```
<!DOCTYPE html>
<html lang="en">
<head>
</head>
<body>
    <div id="timer"></div>
<script>
let time = 3 * 60 * 60 * 1000; // 3 hours
setInterval(() => {
  let h = Math.floor(time / (1000*60*60));
  let m = Math.floor((time % (1000*60*60)) / (1000*60));
  let s = Math.floor((time % (1000*60)) / 1000);
  document.getElementById("timer").innerText = h+":"+m+":"+s;
  time -= 1000;
}, 1000);
</script>
</body>
</html>
```

Timer needs milliseconds to work correctly

1000 :- milliseconds in 1 second

setInterval() runs code **again and again** after a fixed time.

- **time = 3 * 60 * 60 * 1000**
  Converts 3 hours into milliseconds (JS uses milliseconds).

```
let h = Math.floor(time / (1000 * 60 * 60));
```

- Divide total milliseconds by
  1000 × 60 × 60 = milliseconds in 1 hour
- Math.floor() removes decimal value 3.8 hours → 3 hours

let m = Math.floor((time % (1000 * 60 * 60)) / (1000 * 60));

Why used? To calculate minutes.

- % removes the hours part
- Remaining time is converted into minutes

Example: 3 hours 25 minutes → get only 25

```
let s = Math.floor((time % (1000 * 60)) / 1000);
```

To calculate **seconds**. % removes minutes Remaining milliseconds → seconds

❖ Convert Date to String

- **toString():-** To get **full date + time + timezone** as a string.
- **toDateString():-**To get **only the date** in readable form.
- **toTimeString():-** To get **only time + timezone**.

Example:

```
let now = new Date();
console.log(now.toString());
console.log(now.toDateString());
console.log(now.toTimeString());
```

Output: Wed Dec 17 2025 11:30:45 GMT+0530
Output : Wed Dec 17 2025
Output : 11:30:45 GMT+0530 (India Standard Time)

Local Date & Time (User Location Based)

- **toLocaleString():-** To show **date & time in local format** (India, US, etc.)
- **toLocaleDateString():-** To show **only local date**.
- **toLocaleTimeString():-** To show **only local time**.

❖ toLocaleString().

➢ console.log(now.toLocaleString());

Output example (India): 17/12/2025, 11:30:45 am

❖ toLocaleDateString()

➢ console.log(now.toLocaleDateString());

Output example: 17/12/2025

❖ toLocaleTimeString()

➢ console.log(now.toLocaleTimeString());

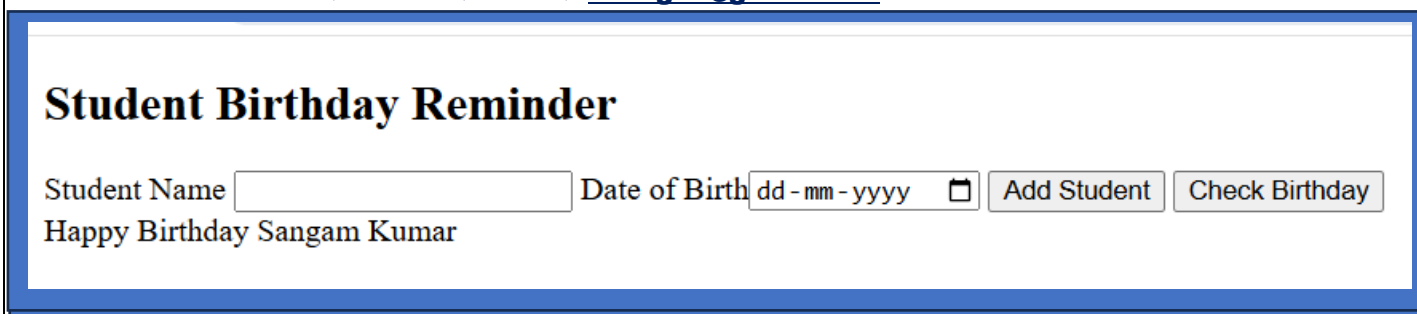
Output example: 11:30:45 am

Timestamp

getTime():- To get **milliseconds from 1 Jan 1970** (used in timers, comparisons).

➢ console.log(now.getTime());

Output example: 1765951245123




```html
<!DOCTYPE html><html>
<head><title>Student Birthday Reminder</title></head>
<body>
<div class="box">
  <h2>Student Birthday Reminder</h2>
Student Name <input type="text" id="name">
Date of Birth<input type="date" id="dob">
  <button onclick="addStudent()">Add Student</button>
  <button onclick="checkBirthday()">Check Birthday</button>
  <div class="out" id="result"></div>
</div>
<script>
let students = []; // store student data
function addStudent(){
  let name = document.getElementById("name").value;
  let dob = document.getElementById("dob").value;
  if(name == "" || dob == ""){
    alert("Please enter name and DOB");
    return;}
  students.push({name:name, dob:dob});
  document.getElementById("result").innerText =
    "Student Added: " + name;
  document.getElementById("name").value = "";
  document.getElementById("dob").value = "";
}
function checkBirthday(){
  let today = new Date();
  let todayDate = today.getDate();
  let todayMonth = today.getMonth(); // 0-11
  let msg = "";
  for(let i=0; i<students.length; i++){
    let d = new Date(students[i].dob);
    let dobDate = d.getDate();
    let dobMonth = d.getMonth();
    if(dobDate === todayDate && dobMonth === todayMonth){
      msg += " Happy Birthday " + students[i].name + "<br>";
    }}
  if(msg === ""){msg = "No birthdays today "; }
  document.getElementById("result").innerHTML = msg;
}</script></body></html>
```

MATH IN JAVASCRIPT

- math object provides several constants and methods to perform mathematical operation.

1) **Math.abs(x):** Returns the absolute value of a number x.
2) **Math.acos(x):** Returns the arccosine (in radians) of a number x, where x is in the range [-1, 1].
3) **Math.asin(x):** Returns the arcsine (in radians) of a number x, where x is in the range [-1, 1].
4) **Math.atan(x):** Returns the arctangent (in radians) of a number x.
5) **Math.cbrt(x):** Returns the cube root of a number x.
6) **Math.ceil(x):** Returns the smallest integer greater than or equal to a number x.
7) **Math.cos(x):** Returns the cosine of a number x (assumed to be in radians).
8) **Math.cosh(x):** Returns the hyperbolic cosine of a number x.
9) **Math.exp(x):** Returns the exponential value of a number x.
10) **Math.floor(x):** Returns the largest integer less than or equal to a number x.
11) **Math.hypot(...args):** Returns the square root of the sum of squares of its arguments, effectively calculating the hypotenuse of a right triangle.
12) **Math.log(x):** Returns the natural logarithm (base e) of a number x.
13) **Math.max(...args):** Returns the maximum value among a list of numbers passed as arguments.
14) **Math.min(...args):** Returns the minimum value among a list of numbers passed as arguments.
15) **Math.pow(x, y):** Returns x raised to the power of y.
16) **Math.random():** Returns a random number between 0 (inclusive) and 1 (exclusive).
17) **Math.round(x):** Returns the value of a number x rounded to the nearest integer.
18) **Math.sign(x):** Returns sign of a number x (1 for +ve, -1 for negative, 0 for zero, and NaN for NaN).
19) **Math.sin(x):** Returns the sine of a number x (assumed to be in radians).
20) **Math.sinh(x):** Returns the hyperbolic sine of a number x.
21) **Math.sqrt(x):** Returns the square root of a number x.
22) **Math.tan(x):** Returns the tangent of a number x (assumed to be in radians).
23) **Math.tanh(x):** Returns the hyperbolic tangent of a number x.
24) **Math.trunc(x):** Returns the integer part of a number x (removing the decimal part).

1. Math.abs(x):- Returns the absolute value of a number x.

- **Syntax: Math.abs(x)**
- **Example:** console.log(Math.abs(-10)); **Output:** 10
  console.log(Math.abs(3.14)); **Output:** 3.14

2. Math.acos(x):- Returns the arccosine (in radians) of a number x, where x is in the range [-1, 1].

- **Syntax: Math.acos(x)**
- **Example:** console.log(Math.acos(1)); **Output:** 0
  console.log(Math.acos(0)); **Output:** 1.5707963267948966 (π/2)

3. Math.asin(x):- Returns the arcsine (in radians) of a number x, where x is in the range [-1, 1].

- **Syntax:** Math.asin(x)
- **Example:** console.log(Math.asin(1)); **Output:** 1.5707963267948966 (π/2)
  console.log(Math.asin(0)); **Output:** 0

4. Math.atan(x):- Returns the arctangent (in radians) of a number x.

- **Syntax:** Math.atan(x)
- **Example:** console.log(Math.atan(1)); **Output:** 0.7853981633974483 (π/4)
  console.log(Math.atan(0)); **Output:** 0

5. Math.cbrt(x):- Returns the cube root of a number x.

- **Syntax:** Math.cbrt(x)
- **Example:** console.log(Math.cbrt(27)); **Output:** 3
  console.log(Math.cbrt(-8)); **Output:** -2

1. Math.ceil(x):- Returns the smallest integer greater than or equal to a number x.

Syntax: Math.ceil(x)

Example: console.log(Math.ceil(4.2)); **Output:** 5
console.log(Math.ceil(-1.7)); **Output:** -1

7. Math.cos(x):- Returns the cosine of a number x (assumed to be in radians).

- **Syntax:** Math.cos(x)
- **Example:** console.log(Math.cos(0)); **Output:** 1
  console.log(Math.cos(Math.PI)); **Output:** -1

8. Math.cosh(x):- Returns the hyperbolic cosine of a number x.

- **Syntax:** Math.cosh(x)
- **Example:** console.log(Math.cosh(0)); **Output:** 1
  console.log(Math.cosh(1)); **Output:** 1.5430806348152437

9. Math.exp(x):- Returns the exponential value of a number x, i.e., e^x.

- **Syntax:** Math.exp(x)
- **Example:** console.log(Math.exp(1)); **Output:** 2.718281828459045 (e)
  console.log(Math.exp(0)); **Output:** 1

10. Math.floor(x):- Returns the largest integer less than or equal to a number x.

- **Syntax:** Math.floor(x)
- **Example:** console.log(Math.floor(4.7)); **Output:** 4
  console.log(Math.floor(-1.2)); **Output:** -2

11. Math.hypot(...args):- Returns the square root of the sum of squares of its arguments, effectively calculating the hypotenuse of a right triangle.

- **Syntax:** Math.hypot(...args)
- **Example:** console.log(Math.hypot(3, 4)); **Output:** 5 (sqrt(3^2 + 4^2))
  console.log(Math.hypot(1, 2, 3)); **Output:** 3.74165738 (sqrt(1^2 + 2^2 + 3^2))

12. Math.log(x):- Returns the natural logarithm (base e) of a number x.

- **Syntax:** Math.log(x)
- **Example:**console.log(Math.log(1)); **Output:** 0 (ln(1))
  console.log(Math.log(Math.E)); **Output:** 1 (ln(e))

13. Math.max(...args)

- **Syntax:** Math.max(...args):- Returns maximum value among a list of numbers passed as arguments.
- **Example:** console.log(Math.max(1, 2, 3, 4, 5)); **Output:** 5
  console.log(Math.max(-1, -2, -3)); **Output:** -1

14. Math.min(...args):- Returns minimum value among a list of numbers passed as arguments.

- **Syntax:** Math.min(...args)
- **Example:** console.log(Math.min(1, 2, 3, 4, 5)); **Output:** 1
  console.log(Math.min(-1, -2, -3)); **Output:** -3

15. Math.pow(x, y):- Returns x raised to the power of y. **Syntax:** Math.pow(x, y)

- **Example:** console.log(Math.pow(2, 3)); **Output:** 8 (2^3)
  console.log(Math.pow(5, 0)); **Output:** 1 (5^0)

16. Math.random():- Returns a random number between 0 (inclusive) and 1 (exclusive).

- **Syntax:** Math.random()
- **Example:** console.log(Math.random()); **Output:** A random number between 0 and 1
- **Math.floor(Math.random() * (max - min + 1)) + min**
  Random number from 5 to 10
  let num = Math.floor(Math.random() * (10 - 5 + 1)) + 5;
  console.log(num)

17. Math.round(x):- Returns the value of a number x rounded to the nearest integer.

- **Syntax:** Math.round(x)
- **Example:**console.log(Math.round(4.7)); **Output:** 5
  console.log(Math.round(4.4)); **Output:** 4

18. Math.sign(x):- Returns sign of a number x (1 for positive, -1 for negative, 0 for zero, and NaN for NaN).

- **Syntax:** Math.sign(x)
- **Example:** console.log(Math.sign(10)); **Output:** 1
  console.log(Math.sign(-10)); **Output:** -1
  console.log(Math.sign(0)); **Output:** 0
  console.log(Math.sign(NaN)); **Output:** NaN

19. Math.sin(x):- Returns the sine of a number x (assumed to be in radians).

- **Syntax:** Math.sin(x)
- **Example:** console.log(Math.sin(0)); // Output: 0
  console.log(Math.sin(Math.PI / 2)); **Output:** 1

20. Math.sinh(x):- Returns the hyperbolic sine of a number x.

- **Syntax:** Math.sinh(x)
- **Example:** console.log(Math.sinh(0)); **Output:** 0
  console.log(Math.sinh(1)); **Output:** 1.1752011936438014

21. Math.sqrt(x):- Returns the square root of a number x.

- **Syntax:** Math.sqrt(x)
- **Example:** console.log(Math.sqrt(9)); **Output:** 3
  console.log(Math.sqrt(16)); **Output:** 4

22. Math.tan(x):- Returns the tangent of a number x (assumed to be in radians).

- **Syntax:** Math.tan(x)
- **Example:** console.log(Math.tan(0)); **Output:** 0
  console.log(Math.tan(Math.PI / 4)); **Output:** 1

23. Math.tanh(x):- Returns the hyperbolic tangent of a number x.

- **Syntax:** Math.tanh(x)
- **Example:** console.log(Math.tanh(0)); **Output:** 0
  console.log(Math.tanh(1)); **Output:** 0.7615941559557649

24. Math.trunc(x):- Returns the integer part of a number x (removing the decimal part).

- **Syntax:** Math.trunc(x)
- **Example:** console.log(Math.trunc(4.9)); **Output:** 4
  console.log(Math.trunc(-4.9)); **Output:** -4

NUMBER OBJECT

- The JavaScript number object enables you to represent a numeric value.

1) **Number.isNaN(value):** Checks if a value is NaN.
2) **Number.isFinite(value):** Checks if a value is finite (not NaN, Infinity, or -Infinity).
3) **Number.parseInt(string, radix):** Parses a string and returns an integer.
4) **Number.parseFloat(string):** Parses a string and returns a floating-point number.
5) **Number.toFixed(digits):** Converts a number to a string with a fixed number of decimal places.
6) **Number.toPrecision(precision):** Converts a number to a string with a specified number of significant digits.
7) **Number.toString(base):** Converts a number to a string in a specified base.
8) **Number.isInteger(value):** Checks if a value is an integer.
9) **Number.MAX_VALUE:** Represents the maximum numeric value.
10) **Number.MIN_VALUE:** Represents the smallest positive numeric value.
11) **Number.POSITIVE_INFINITY:** Represents positive infinity.
12) **Number.NEGATIVE_INFINITY**: Represents negative infinity.

1. Number.isNaN(value):- Checks if a value is NaN (Not-a-Number).

- **Syntax:** Number.isNaN(value)
- **Example:** console.log(Number.isNaN(NaN)); **Output:** true
  console.log(Number.isNaN(123)); **Output:** false

2. Number.isFinite(value):- Checks if a value is finite (not NaN, Infinity, or -Infinity).

- **Syntax:** Number.isFinite(value)
- **Example:** console.log(Number.isFinite(123)); **Output:** true
  console.log(Number.isFinite(Infinity)); **Output:** false

3. Number.parseInt(string, radix):- Parses a string and returns an integer. The radix is an integer between 2 and 36 that represents the base of the string.

- **Syntax:** Number.parseInt(string, radix)
- **Example:** console.log(Number.parseInt('100', 10)); **Output:** 100
  console.log(Number.parseInt('11', 2)); **Output:** 3

4. Number.parseFloat(string):- Parses a string and returns a floating-point number.

- **Syntax:** Number.parseFloat(string)
- **Example:** console.log(Number.parseFloat('3.14')); **Output:** 3.14
  console.log(Number.parseFloat('123abc')); **Output:** 123

5. Number.toFixed(digits):- Converts a number to a string with a fixed number of decimal places.

- **Syntax:** number.toFixed(digits)
- **Example:** let num = 3.14159;
  console.log(num.toFixed(2)); **Output:** '3.14'
  console.log(num.toFixed(4)); **Output:** '3.1416'

6. Number.toPrecision(precision):- Converts a number to a string with a specified number of significant digits

- **Syntax:** number.toPrecision(precision)
- **Example:** let num = 3.14159;
  console.log(num.toPrecision(2)); **Output:** '3.1'
  console.log(num.toPrecision(4)); **Output:** '3.142'

7. Number.toString(base): Converts a number to a string in a specified base. The base is an integer between 2 and 36.

- **Syntax:** number.toString(base)
- **Example:** let num = 255;
  console.log(num.toString(16)); **Output:** 'ff'
  console.log(num.toString(2)); **Output:** '11111111'

8. Number.isInteger(value):- Checks if a value is an integer.

- **Syntax:** Number.isInteger(value)
- **Example:** console.log(Number.isInteger(123)); **Output:** true
  console.log(Number.isInteger(123.45)); **Output:** false

9. Number.MAX_VALUE:- Represents maximum numeric value that can be represented in JavaScript.

- **Syntax:** Number.MAX_VALUE
- **Example:** console.log(Number.MAX_VALUE); **Output:** 1.7976931348623157e+308
  console.log(Number.MAX_VALUE > 1e308); **Output:** true

10. Number.MIN_VALUE:- Represents smallest positive numeric value that can be represented in JavaScript.

- **Syntax:** Number.MIN_VALUE
- **Example:** console.log(Number.MIN_VALUE); **Output:** 5e-324
  console.log(Number.MIN_VALUE < 1e-323); **Output:** true

11. Number.POSITIVE_INFINITY:- Represents positive infinity

- **Syntax:** Number.POSITIVE_INFINITY
- **Example:** console.log(Number.POSITIVE_INFINITY); **Output:** Infinity
  console.log(1 / 0); **Output:** Infinity

12. Number.NEGATIVE_INFINITY

- **Syntax:** Number.NEGATIVE_INFINITY
- **Description:** Represents negative infinity.
- **Example:** console.log(Number.NEGATIVE_INFINITY); **Output:** -Infinity
  console.log(-1 / 0); **Output:** -Infinity

OOPS
Object Oriented Programming System

- Oops is a methodology or paradigm to design a program using **classes** and **objects.**
- The programming paradigm where everything is represented as an **object** is known as truly **object-oriented programming** language. Smalltalk is considered as the first truly object-oriented programming language.
- It simplifies the software development and maintenance by providing following concepts.

1. **Object:** Real-world entity
2. **Class:** Blueprint of object
3. **Inheritance:** Acquiring properties of another class
4. **Polymorphism:** One name, many forms
5. **Abstraction:** Hiding details
6. **Encapsulation:** Binding data and methods;

Advantages of OOPS

1. Reusable code saves time
2. Easy to understand and manage
3. High productivity
4. Easy to expand programs
5. Supports multiple objects
6. Easy team work
7. Real-world representation
8. Data security Less code duplication

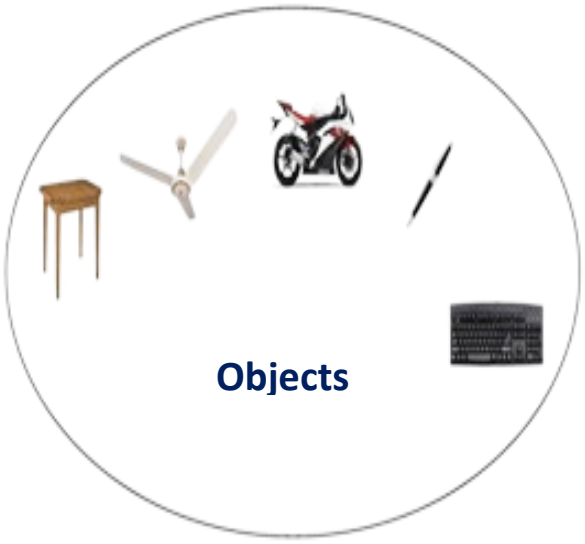
Disadvantages of OOPS:

1. Large and slow programs
2. Not suitable for all problems
3. Complex design
4. Difficult for beginners
5. Requires object-based thinking

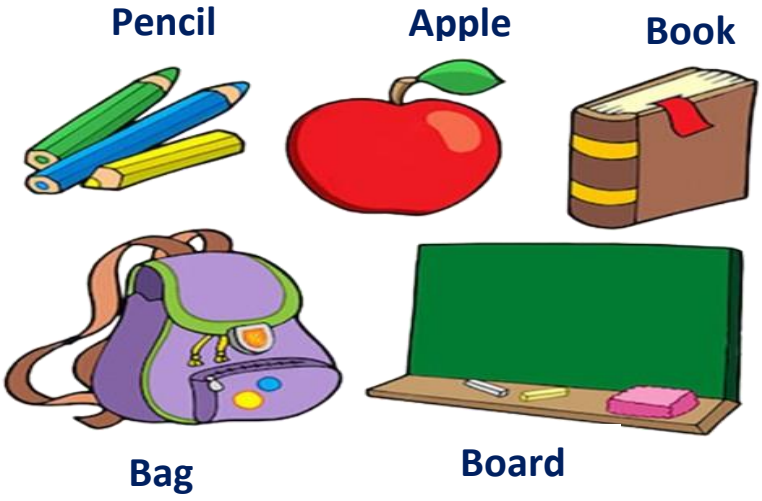
1. Object:

- Object means a real word entity such as pen, chair, table etc.
- Any entity that has **state** and **behavior** is known as an object.
- For example: chair, pen, table, keyboard, bike etc. It can be **physical** and **logical.**
- **State** tells us how object looks or what properties it has. **Behavior** tells us what object does.
- object is a specific **instance** of a class.
- **instance** is a specific realization of any objects.

Example:-

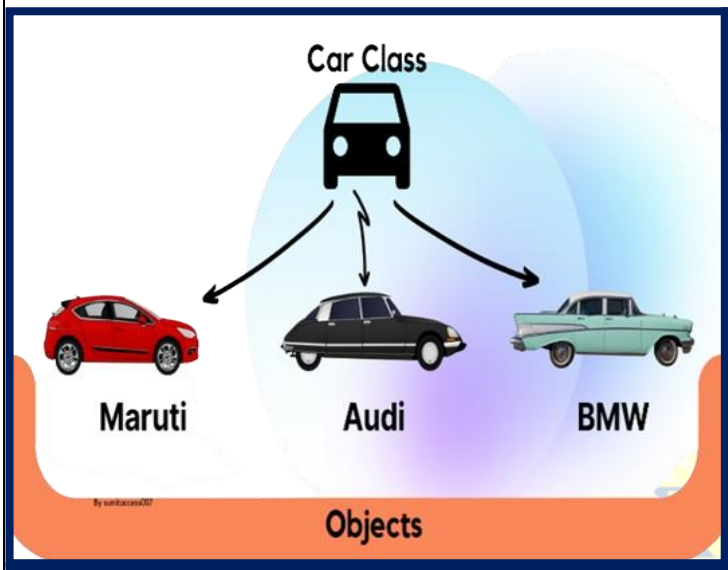


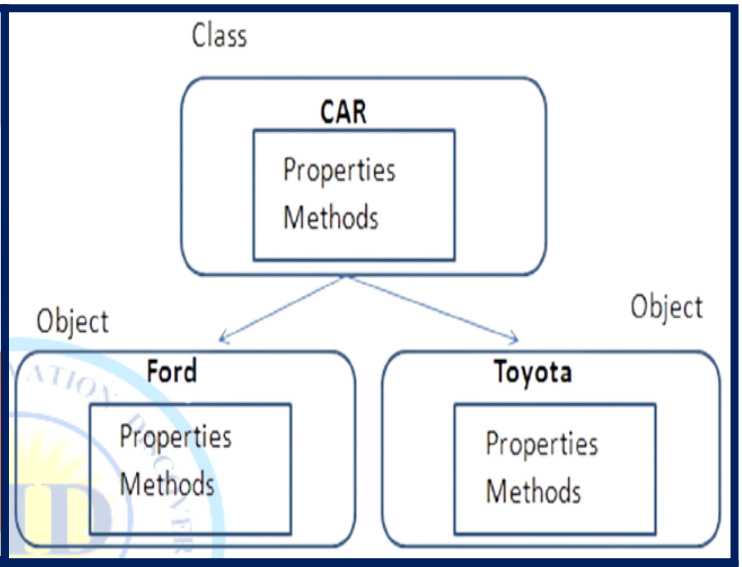
Objects: Real World Example

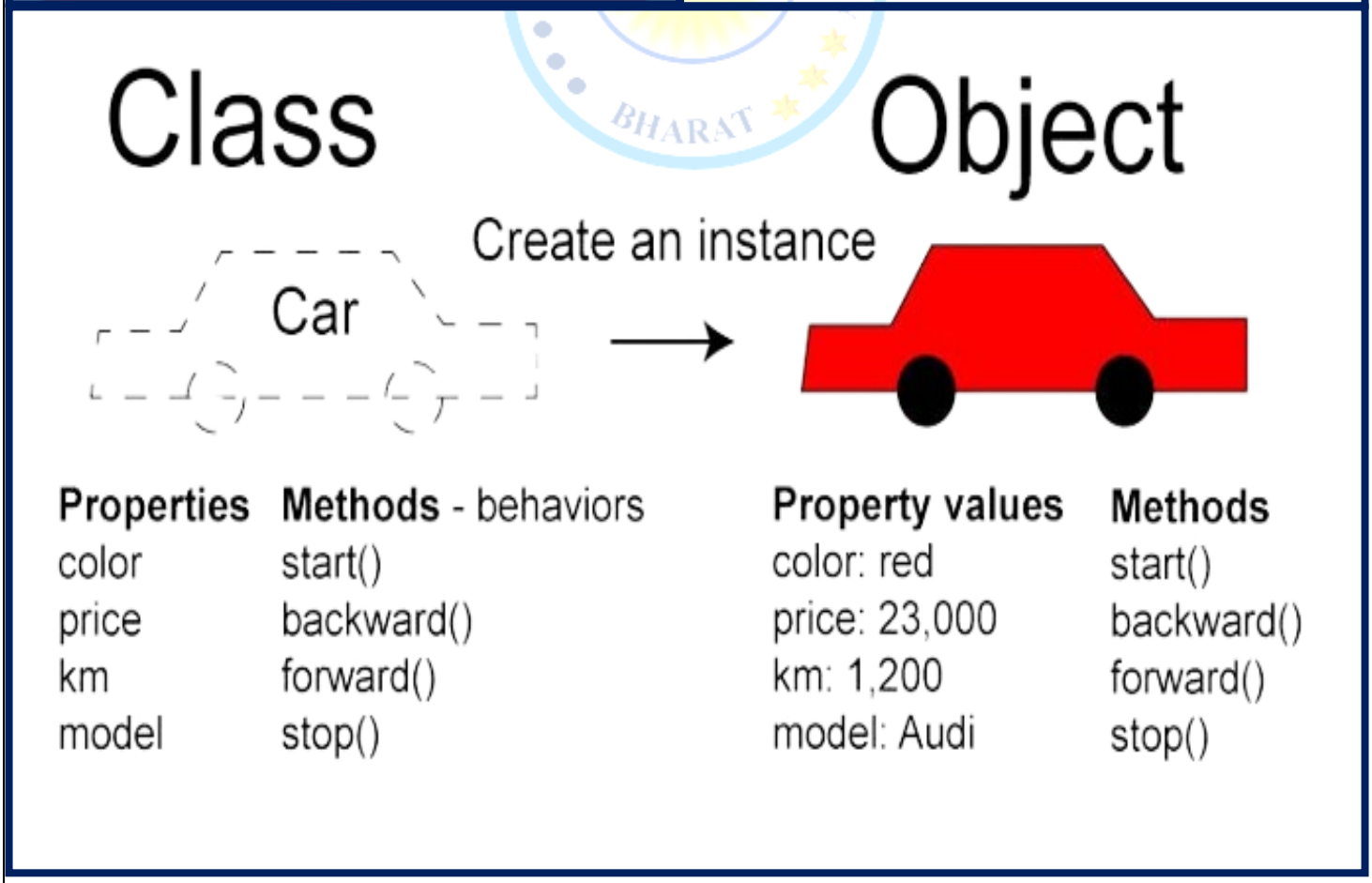


2. Class:

- Collection of **objects** is called class. It is a logical entity.
- To create objects, we required some model or plan or template or blue print, which is nothing but class.
- We can write a class to represent **properties(attributes)** and **actions (behaviour)** of object.
- **Properties** can be represented by **variable**
- **Action** can be represented by **methods.**
- Hence class contains both **variables** and **method.**
- a class is a template definition of the methods and variables in a particular kind of object. Thus, an object is a specific **instance of a class**.

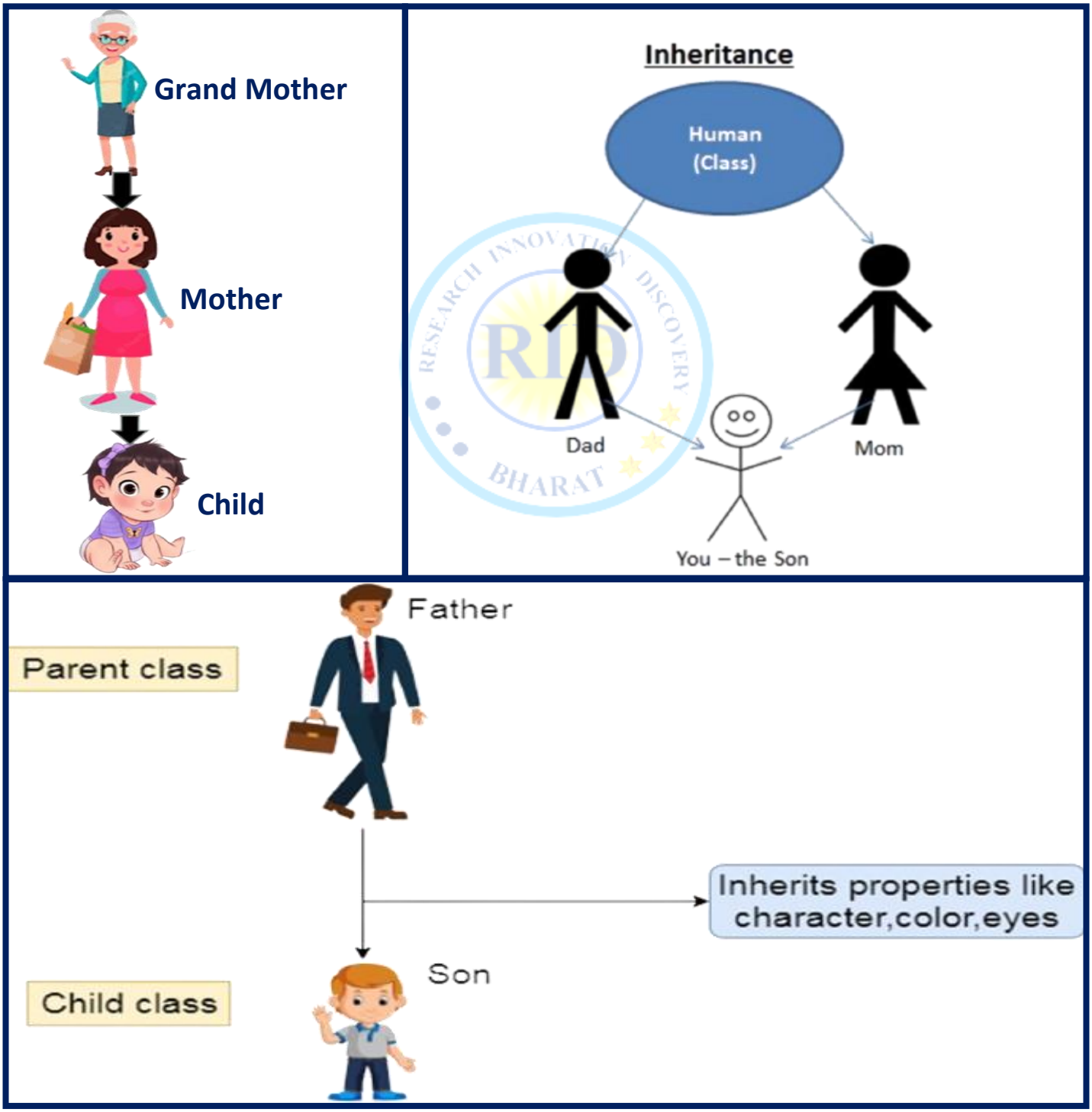






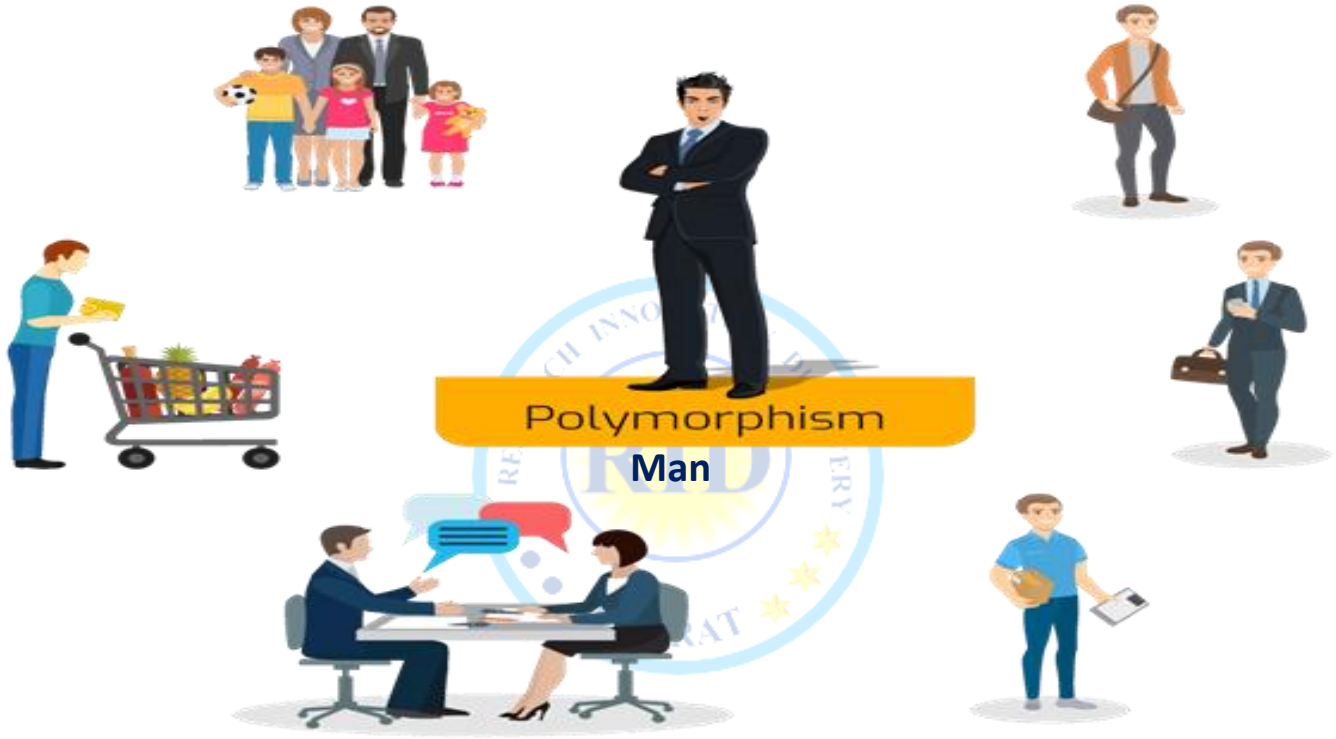
3. Inheritance:

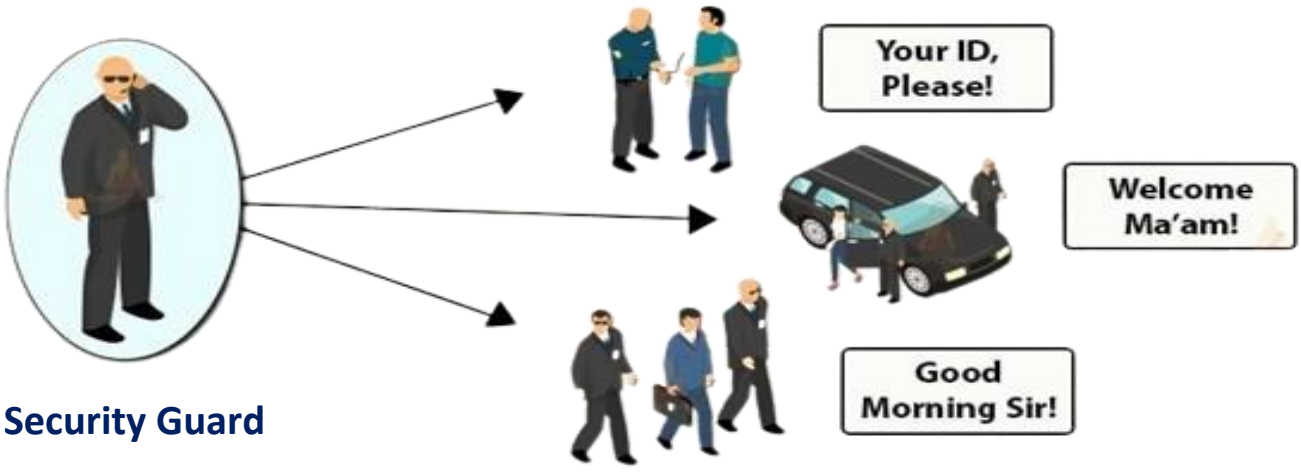
- When one object acquires all the **properties** and **behaviours** of **parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime **polymorphism.**
- **Sub class -** Subclass or Derived Class refers to a class that receives properties from another class.
- **Super class -** The term "Base Class" or "Super Class" refers to the class from which a subclass inherits its properties.
- **Reusability -** when we wish to create a new class, but an existing class already contains some of the code we need, we can generate our new class from the old class that is inheritance.



1. Polymorphism:

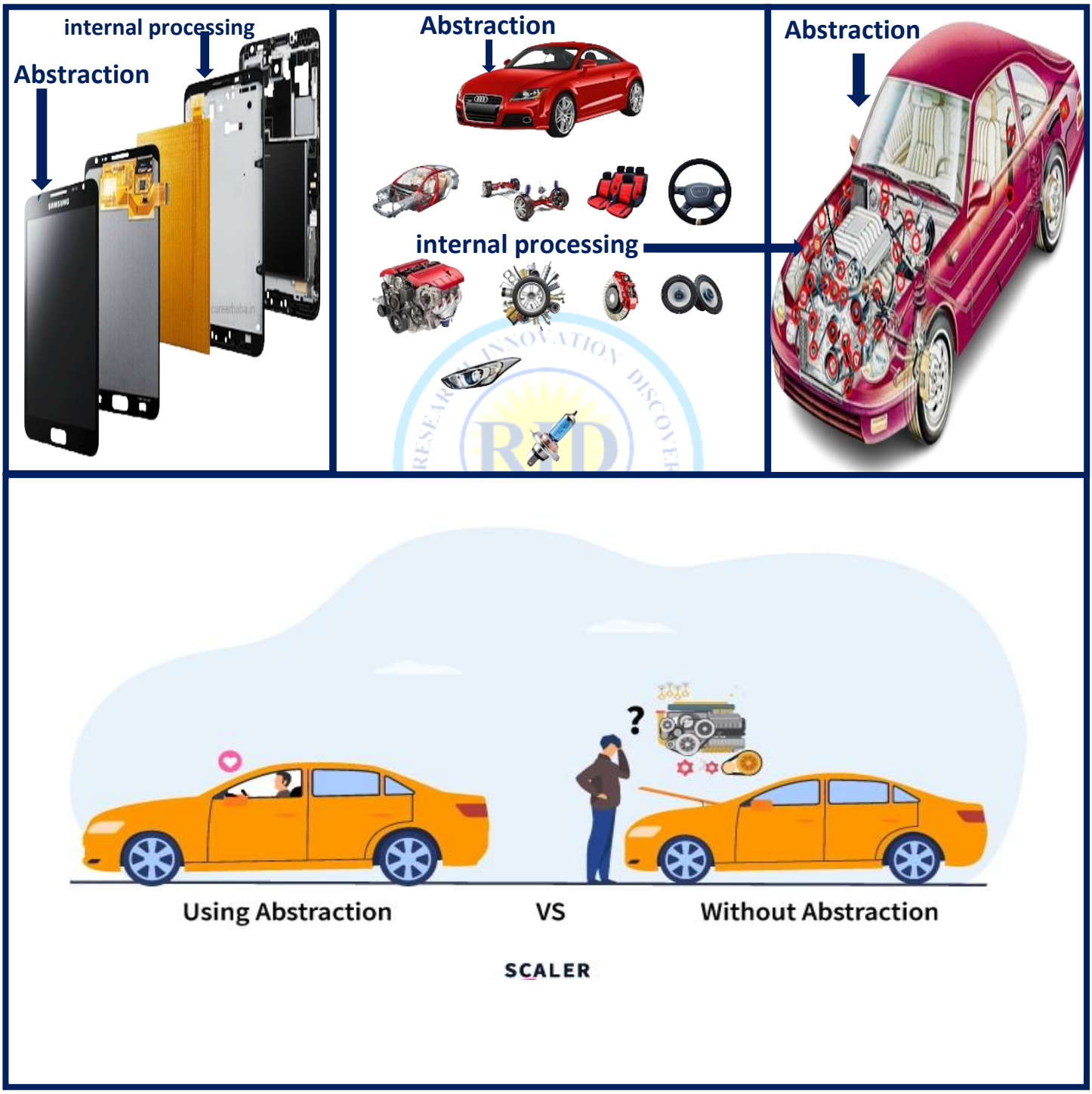
- When one task is performed by different ways i.e. known as polymorphism.
- Polymorphism means "many forms"
- Imagine a man named Sangam. Sangam has different roles depending on the context:
  - ✓ At work, Sangam is an Engineer.
  - ✓ At home, Sangam is a father.
  - ✓ In a social gathering, Sangam is a Friend.
  - ✓ In flight, Sangam is passenger.





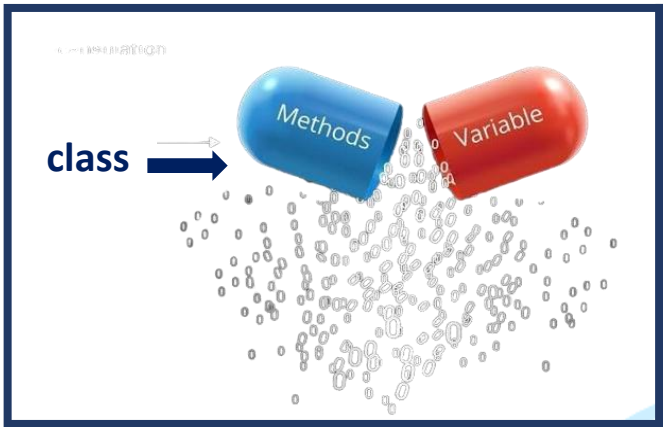
2. Abstraction:

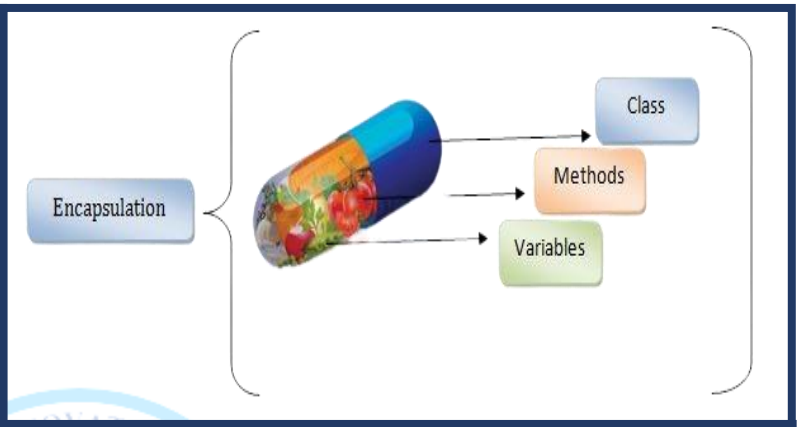
- Hiding **internal** details and showing **functionality** is known as abstraction.
- Data abstraction is the process of exposing to the outside world only the information that is absolutely necessary while concealing implementation or background information. For example: phone call and Car. we don't know the internal processing.
- n C++, we use abstract class and interface to achieve abstraction.

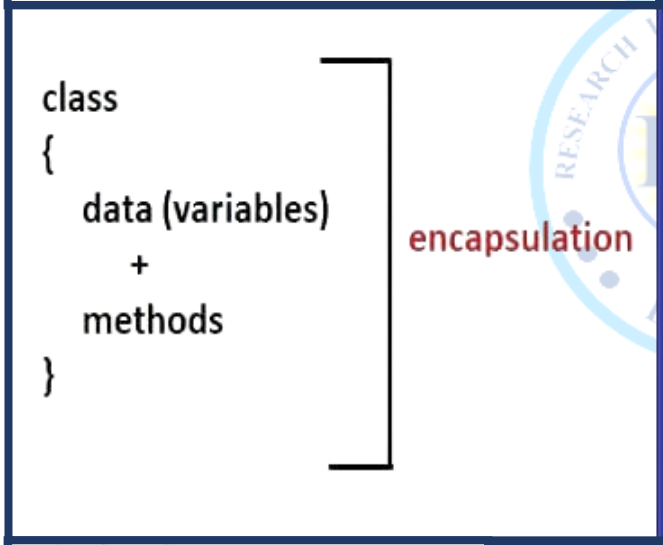


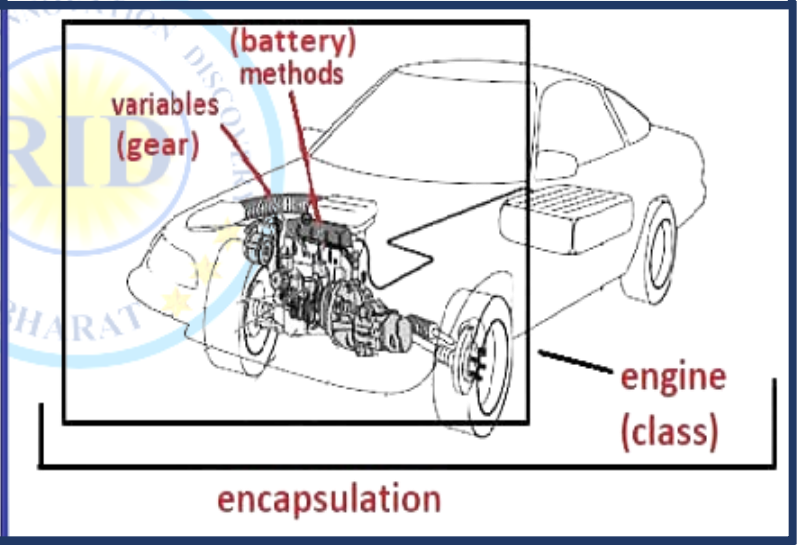
3. Encapsulation:

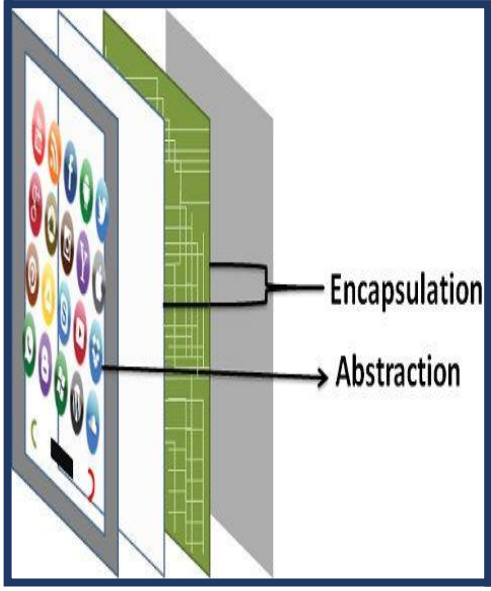
- **Binding** (or wrapping) **code** and **data** together into a single unit is known as encapsulation.
- encapsulation is described as the process of combining code (**methods or functions**) and data (**variables or attributes**) into a single unit.
- **Methods** or functions represent **executable** code, while data or variables represent **stored** information within a program.

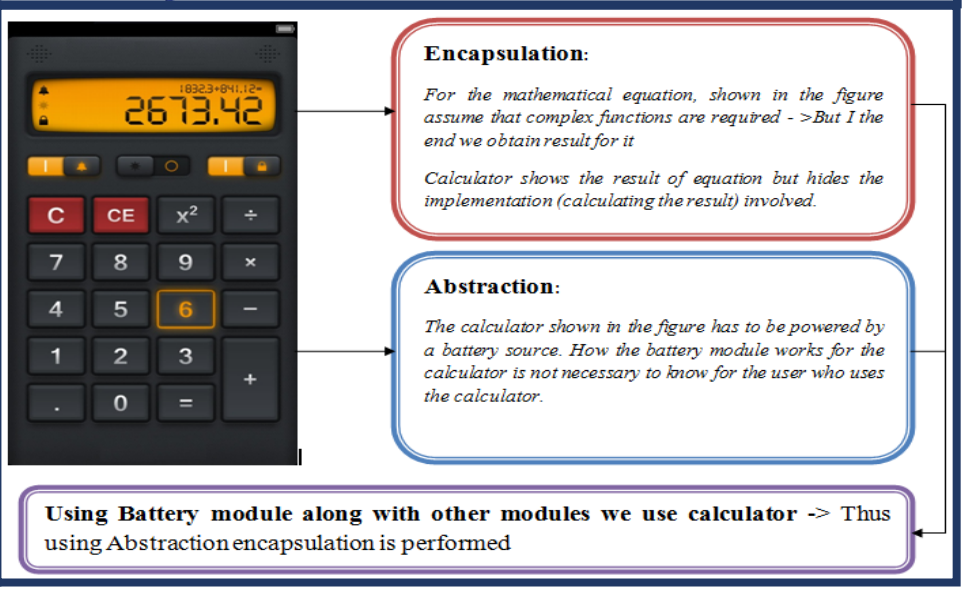












Object

An object is a **real-world entity** that contains **properties (data)** and **methods (functions)**.

→ An object is an **instance of a class**.

➢ **Example (Real Life):** A **car** is an object.

- **Properties:** color, model, speed
- **Actions:** drive, stop

❖ **Ways to Create Object in JavaScript**

1. Object Literal: - Object literal is the **easiest way** to create an object using { } with key–value pairs.

Example:

```
let person = {
  name: "Sangam Kumar",
  age: 28,
  contact: 900500506,
  getDetails: function () {
    console.log(this.name, this.age, this.contact);
  }
};
person.getDetails();
```

2. Object Constructor

(a) Built-in Object Constructor → let obj = new Object();

→ Creates an empty object (same as { }).

(b) Custom Object Constructor A constructor function is used to create **multiple objects with same structure**.

Example:

```
function Person(name, age, address) {
  this.name = name;
  this.age = age;
  this.address = address;
}
let p1 = new Person("Jai Kumar", 30, "Patna");
console.log(p1.name, p1.age, p1.address);
```

Class

A class is a **blueprint** used to create objects. Collection of object

Syntax: - to create a class in JavaScript (ES6):

```
class ClassName {
  constructor() {
    // initialize properties
  }
  methodName() {
    // class method
  }
}
```

- **class**: Used to create a class
- **constructor()**: Initializes object data
- **Methods**: Functions inside a class
- **this**: Refers to current object
- **new**: Creates object from class

Example:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  show() { console.log(this.name, this.age);
  }
}
let p1 = new Person("Sangam", 25);
p1.show();
```

Output: Sangam 25

❖ constructor(name, age) — Scope Explained

```
constructor(name, age) {
 this.name = name;
 this.age = age;
}
```

➢ **name and age (parameters)**

- name and age are **local variables** of the constructor.
- Their **scope is only inside the constructor**.
- They receive values when an object is created using new.

Example: new Person("Amit", 20);
Here, "Amit" goes to name and 20 goes to age.

➢ **this.name = name;**

- this refers to the **current object**.
- Left side this.name → **object property**
- Right side name → **constructor parameter**
- This line **stores the parameter value into the object**.

➔ It means: **Object's name = value passed to constructor**

➢ **this.age = age;**

- Same logic as above.
- Creates an **object property age** and assigns value to it.

➢ **this.name and this.age (object properties)**

- These are **object-level variables**.
- They can be accessed anywhere in the class using this.
- They exist as long as the object exists.

Example:

```
show() {
 console.log(this.name, this.age);
}
```

Note:

- Constructor parameters are **local**
- this.variable creates **object properties**
- this refers to the **current object**

Note:

- Inside a class, use **this** to access **properties** and **methods** of the object.
- Example: this.name, this.age, this.show()

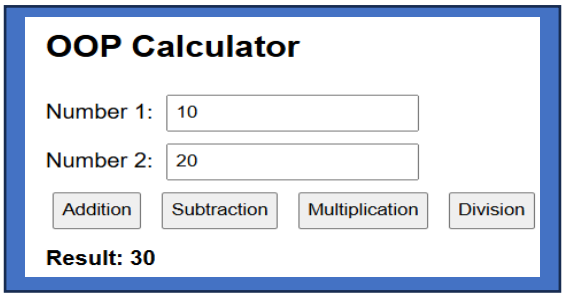
Access Inside a Method

```
class Person {
 constructor(name, age) {
  this.name = name;
  this.age = age;
 }
 show() {
  console.log(this.name, this.age);
 }
}
```

Note: **Access One Method from Another**

```
this.show();
```

```
<body>
  <h2>OOP Calculator</h2>
  Number 1: <input type="number" id="num1"><br>
  Number 2: <input type="number" id="num2"><br>
  <button onclick="calculate('add')">Addition</button>
  <button onclick="calculate('sub')">Subtraction</button>
  <button onclick="calculate('mul')">Multiplication</button>
  <button onclick="calculate('div')">Division</button>
  <div class="result" id="result"></div>
  <script>  // Calculator class
    class Calculator {
      constructor(a, b) {
        this.a = a;
        this.b = b;
      }
      add() { return this.a + this.b; }
      sub() { return this.a - this.b; }
      mul() { return this.a * this.b; }
      div() { return this.a / this.b; }
    }
    function calculate(operation) {
      let n1 = parseFloat(document.getElementById("num1").value);
      let n2 = parseFloat(document.getElementById("num2").value);
      if (isNaN(n1) || isNaN(n2)) {
        document.getElementById("result").innerText = "Please enter valid numbers!";
        return;
      }
      let calc = new Calculator(n1, n2);
      let res;
      switch(operation) {
        case 'add': res = calc.add(); break;
        case 'sub': res = calc.sub(); break;
        case 'mul': res = calc.mul(); break;
        case 'div': res = calc.div(); break;
        default: res = "Invalid operation";
      }
      document.getElementById("result").innerText = "Result: " + res;
    }</script></body></html>
```




```
<!DOCTYPE html>
<html>
<head>
 <title>OOP Calculator</title>
 <style>
  body { font-family: Arial; padding: 20px; }
  input, button { margin: 5px; padding: 5px; }
  .result { margin-top: 10px; font-weight: bold; }
 </style>
</head>
<body>
 <h2>OOP Calculator</h2>
 Number 1: <input type="number" id="num1"><br>
 Number 2: <input type="number" id="num2"><br>
 <button onclick="calculate('add')">Addition</button>
 <button onclick="calculate('sub')">Sub</button>
 <button onclick="calculate('mul')">Mul</button>
 <button onclick="calculate('div')">Division</button>
 <div class="result" id="result"></div>
 <script>  // Calculator class
  class Calculator {
   constructor(a, b) {
    this.a = a;
    this.b = b;
   }
   add() { return this.a + this.b; }
   sub() { return this.a - this.b; }
   mul() { return this.a * this.b; }
   div() { return this.a / this.b; }
  }
```

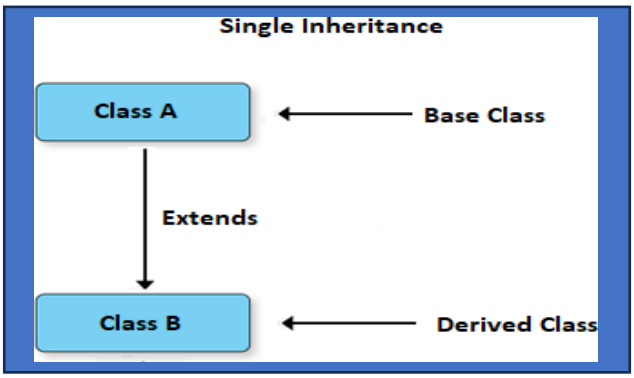
```
  function calculate(operation) {
   let n1 = parseFloat(document.getElementById("num1").value);
   let n2 = parseFloat(document.getElementById("num2").value);
   if (isNaN(n1) || isNaN(n2)) {
    document.getElementById("result").innerText = "Please enter valid
numbers!";
    return;
   }
   let calc = new Calculator(n1, n2);
   let res;
   switch(operation) {
    case 'add': res = calc.add(); break;
    case 'sub': res = calc.sub(); break;
    case 'mul': res = calc.mul(); break;
    case 'div': res = calc.div(); break;
    default: res = "Invalid operation";
   }
   document.getElementById("result").innerText = "Result: " + res;
  }</script></body></html>
```

Inheritance

- Inheritance means acquiring properties and methods from another class or object. It helps in code reuse, better organization, and hierarchical structure.

Types Of Inheritance

1. Single Inheritance: - When a child object or class inherits from a single parent object or class.



Example:

```
Define the Father Class (Parent Class)
class father{ constructor(fname){
this.fname=fname
}
print(){
console.log("Father name is ", this.fname)
}}
// Define the Son Class (Child Class)
class son extends father{
// Son class: Inherits from the Father class using the extends keyword.
constructor(fname, sname){
super(fname) // Uses super keyword to call the parent class's constructor, passing the fname argument.
this.sname=sname
}
print(){
console.log("Father's Name is",this.fname) console.log("Child name is ", this.sname)
} }
// Create an Instance of Son and Call the print() Method
let obj2=new son("Mohan Kumar", "Sohan Kumar") obj2.print()
```

Output:

```
Father's Name is Mohan Kumar
Child name is Sohan Kumar
```

Key Concepts Demonstrated

1. **Class-based Inheritance:**
   - The Son class inherits properties and methods from the Father class using the extends keyword.
2. **super Keyword:**
   - Used in the Son class constructor to call the parent class (Father) constructor and initialize properties (fname).
3. **Method Overriding:**
   - The Son class overrides the print() method of the Father class to provide additional functionality.
4. **Polymorphism**:
   - Both Father and Son have a print() method, but the behavior changes depending on the class instance.

2. Multilevel Inheritance:

- A type of inheritance where a child class inherits from a parent class, and then a subclass inherits from that child class. This forms a chain of inheritance.
- **Supported in JavaScript**: Yes, JavaScript supports multilevel inheritance using class and prototype.

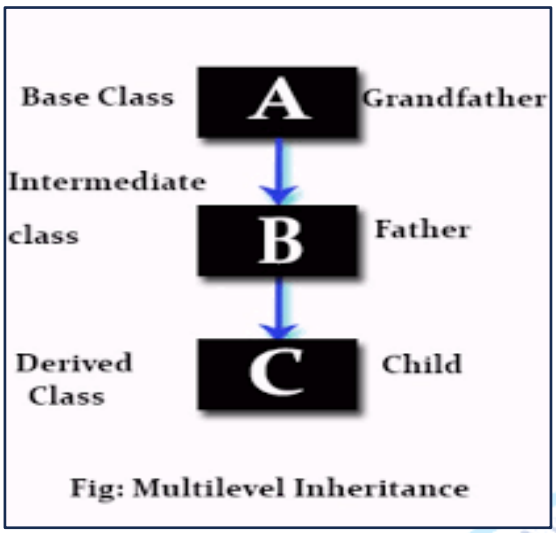
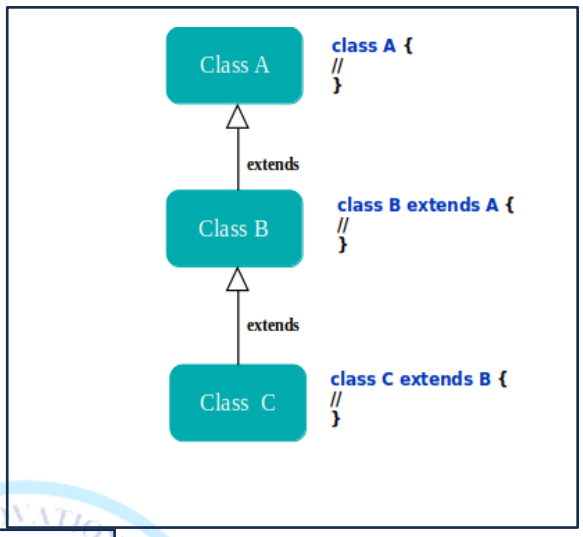


Fig: Multilevel Inheritance

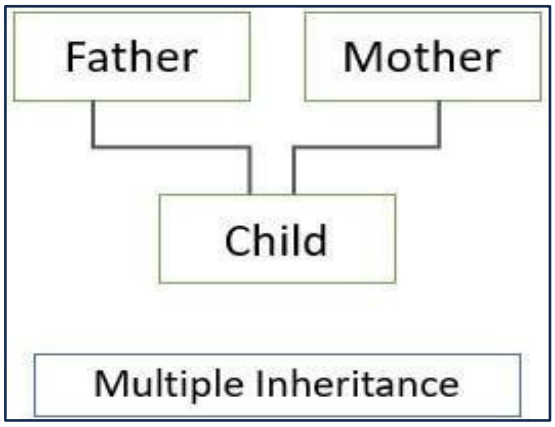


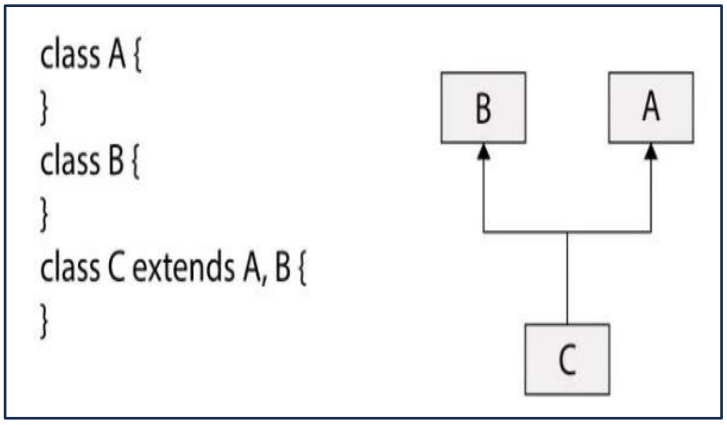
Example 1. GrandFather Class (Base Class)

```
class GrandFather {
  constructor(gname) {
    this.gname = gname;
  }
  disp() {
    console.log("Grandfather name is:", this.gname);
  }}// 2. Father Class (Child of GrandFather)
class Father extends GrandFather {
  constructor(gname, fname) {
    super(gname);
    this.fname = fname;
  }
  disp() {
    console.log("Grandfather name is:", this.gname);
    console.log("Father name is:", this.fname);
  }}// 3. Son Class (Child of Father)
class Son extends Father {
  constructor(gname, fname, sname) {
    super(gname, fname);
    this.sname = sname;
  }
  disp() {
    console.log("Grandfather name is:", this.gname);
    console.log("Father name is:", this.fname);
    console.log("Son name is:", this.sname);
  }} // Object creation and method call
let obj1 = new Son("Mohan", "Sohan", "Johan");
obj1.disp();
```

3. Multiple Inheritance:

- A type of inheritance where a class can inherit from multiple parent classes at the same time.
- Supported in js: No direct support. However, it can be achieved through mixins or composition.
- Prototype: A mechanism in JavaScript that allows objects to share properties and methods.





1.Multiple Inheritance (Without Prototype – Using Composition)
2.Multiple Inheritance (Using Mixins

Example-1

```
//Multiple Inheritance (Without Prototype
// - Using Composition)
class CanWalk { // Feature 1
 walk() {console.log("I can walk");}
}
// Feature 2
class CanTalk {
  talk() {console.log("I can talk");}
}
// Main class uses both features
class Person {
  constructor() {
    this.walkFeature = new CanWalk();
    this.talkFeature = new CanTalk();
  }
  walk() {this.walkFeature.walk();}
  talk() {this.talkFeature.talk();}
}
// Create object
let p1 = new Person();
p1.walk();
p1.talk();
```

Example-2

```
//Multiple Inheritance (Using Mixins
// Mixins (features)
let canWalk = {
  walk() {console.log("I can walk");}
};
let canTalk = {
  talk() {console.log("I can talk");}
};
// Class
class Person {
  constructor(name) {
    this.name = name;   // FIXED
    console.log(this.name);
  }
}
// Apply mixins
Object.assign(Person.prototype, canWalk, canTalk);
// Create object
let p1 = new Person("Mohan");
p1.walk();
p1.talk();
```

- Object.assign(): Used to copy properties and methods from one object to another.
- Person.prototype: Target object where methods will be added.
- canWalk: Source object that contains walk() method.
- canTalk: Source object that contains talk() method.
- Why we use it: To add multiple features to a class and achieve
- multiple inheritance using mixins.

4.Hierarchical Inheritance

→ When multiple child objects inherit from the same parent object

Example: 1. The Animal Class (Parent Class)

class Animal { speak() {
console.log("Animal makes a sound.");
} }

Purpose: This method is intended to represent a general behavior common

2. The Dog Class (Child of Animal)

class Dog extends Animal { bark() {
console.log("Dog barks.");
}
}

//Inheritance: The Dog class extends the Animal class using the extends keyword.

// Purpose: Represents a specific type of animal (Dog) with unique behavior (bark())

The Cat Class (Child of Animal)

class Cat extends Animal { meow() {
console.log("Cat meows.");
}
}

// Inheritance: The Cat class also extends the Animal class. Purpose: Represents another specific type of animal (Cat) with unique behavior (meow()).

4.Creating Instances and Calling Methods

let myDog = new Dog(); let myCat = new Cat();

5.Calling the speak() Method

myDog.speak(); // Output: Animal makes a sound. myCat.speak(); // Output: Animal makes a sound.

```
// Parent class
class Student {
  constructor(name) {
    this.name = name;
  }
  study() {console.log(this.name + " studies");}
}
// Child class 1
class SchoolStudent extends Student {
  constructor(name) {
    super(name);   // call parent constructor
  }
  attendSchool() {console.log(this.name + " attends school"); }
}
// Child class 2
class CollegeStudent extends Student {
  constructor(name) {
    super(name);   // call parent constructor
  }
  attendCollege() {console.log(this.name + " attends college");}
}
// Create objects
let s1 = new SchoolStudent("Amit");
let s2 = new CollegeStudent("Rahul"); // Call methods
s1.study();
s1.attendSchool();
s2.study();
s2.attendCollege();
```

```
// Parent class
class Student {
  study() {console.log("Student studies");}
}
// Child class 1
class SchoolStudent extends Student {
  attendSchool() {console.log("School student attends school");}
}
// Child class 2
class CollegeStudent extends Student {
  attendCollege() {console.log("College student attends college");}
}
// Create objects
let s1 = new SchoolStudent();
let s2 = new CollegeStudent();
// Call methods
s1.study();          // from Student
s1.attendSchool();   // from SchoolStudent
s2.study();          // from Student
s2.attendCollege();  // from CollegeStudent
```

Output:

Amit studies
Amit attends school
Rahul studies
Rahul attends college

❖ Polymorphism:

→ Polymorphism in JavaScript refers to the ability of different objects to respond to the same method or function call in their own way.

1.Polymorphism with Classes and Method Overriding

→ When a subclass overrides a method from its parent class to provide a specific implementation.

1.Method overriding---inheritance is mandatory

→ When a child class provides its own version of a parent class method to change or extend its behavior.

```
// Parent class
class Student {
  info() {console.log("This is a student.");}
}
// Child class 1
class SchoolStudent extends Student {
  info() {console.log("This is a school student.");}
}
// Child class 2
class CollegeStudent extends Student {
  info() {console.log("This is a college student.");}
}
// Create objects
let s1 = new Student();
let s2 = new SchoolStudent();
let s3 = new CollegeStudent();
// Call methods
s1.info();  // Output: This is a student.
s2.info();  // Output: This is a school student.
s3.info();  // Output: This is a college student.
// Create objects and call method
// const students = [new Student(), new SchoolStudent(), new CollegeStudent()];
// students.forEach(s => s.info());
```

```
// Parent class
class Student {
  constructor(name) {
    this.name = name;
  }
  info() {console.log(this.name + " is a student");}
}
// Child class 1
class SchoolStudent extends Student {
  info() {console.log(this.name + " is a school student");}
}
// Child class 2
class CollegeStudent extends Student {
  info() {console.log(this.name + " is a college student");}
}
// Create objects and call method
const students = [
  new Student("Amit"),
  new SchoolStudent("Rohit"),
  new CollegeStudent("Priya")
];
students.forEach(s => s.info());
```

2. Polymorphism with Interfaces (Duck Typing)

- Different objects can implement the same method and behave differently.

Example:

```
class Circle {
 draw() {
  console.log("Drawing a Circle.");
 } }
class Square {
 draw() {
  console.log("Drawing a Square.");
 } }
class Triangle {
 draw() {
  console.log ("Drawing a Triangle.");
 }}
const shapes = [new Circle(), new Square(), new Triangle()];
shapes.forEach(shape => shape.draw());
```

Output:

```
Drawing a Circle.
Drawing a Square.
Drawing a Triangle.
```

❖ Function Overloading:

- JavaScript does not support function overloading natively (like in some other languages), but it can be simulated using default parameters or by checking the arguments.

Example-1: by checking the arguments

```
function greet(name, age) {
 if (age !== undefined) {
  console.log(`Hello ${name}, you are ${age} years old.`);
 } else {
  console.log(`Hello ${name}.`);
 }
}
// Calling the function with different arguments
greet("Sangam");        // Output: Hello Sangam.
greet("Bob", 25);       // Output: Hello Bob, you are 25 years old.
```

Example-1: Using Default Arguments

```
function greet(name, age = null) {
 if (age !== null) {
  console.log(`Hello ${name}, you are ${age} years old.`);
 } else {
  console.log(`Hello ${name}.`);
 }
}
// Calling the function with different arguments
greet("Alice");        // Output: Hello Alice.
greet("Bob", 25);      // Output: Hello Bob, you are 25 years old.
```

❖ Operator Overloading:

JavaScript does not allow direct operator overloading, but you can achieve a similar effect by overriding behavior of object methods like toString or using valueOf method for custom operations.

Example: Overloading the + Operator

```
class ComplexNumber {
 constructor(real, imaginary) {
  this.real = real;
  this.imaginary = imaginary;
 }
 // Custom addition for complex numbers
 add(other) {
  return new ComplexNumber(
   this.real + other.real,
   this.imaginary + other.imaginary
  );
 }
 toString() {
  return `${this.real} + ${this.imaginary}i`;
 } }
const num1 = new ComplexNumber(3, 4);
const num2 = new ComplexNumber(1, 2);
const result = num1.add(num2);
console.log(result.toString()); // Output: 4 + 6i
```

Key Notes:

1. **Function Overloading:** Simulated using conditions or default arguments.
2. **Operator Overloading:** Achieved by defining custom methods (e.g., add or overriding toString/valueOf).

❖ Abstraction

- Abstraction is the concept of hiding implementation details and showing only the essential features of an object or function. In JavaScript, abstraction can be achieved using:
  1. **Classes and Methods**: Encapsulating implementation details inside methods.
  2. **Closures**: Using private variables that are accessible only through specific functions.

Comparison with Public Methods

Aspect	Private Method (`#igniteEngine`)	Public Method (`start`)
Access Scope	Only accessible within the class.	Accessible from anywhere.
Purpose	Internal implementation detail.	Public interface for the class.
Syntax	Prefixed with `#`.	No prefix required.

Example 1: Abstraction with Classes

Using methods to hide complex implementation details and expose only what's necessary.

```
class Car {
  constructor(make, model) {
    this.make = make;
    this.model = model;
  }
// Public method to start the car
  start() {
    this.#raj();
    console.log(`${this.make} ${this.model} is starting...`);
  }   // Private method (implementation detail)
  #raj() {
    console.log("Igniting the engine...");
  }   }
const myCar = new Car("Toyota", "TATA");
myCar.start();
// Output: Igniting the engine... \n Toyota TATA is starting...
// myCar.#raj(); // Error: Private field '#igniteEngine' must be
// declared in an enclosing class
```

Abstraction means **showing only what is necessary and hiding internal details**.
In this example, the Car class exposes only the **public method** start() to the user.
The **private method** #raj() hides the internal process of starting the engine.

Why we use abstraction:

- Hides implementation details (#raj() is private)
- Improves security (private methods cannot be accessed outside)
- Makes code simple and easy to use
- Reduces complexity for the user

Conclusion:

The user only calls start(), while the internal engine logic (#raj()) remains hidden — this is

Example-2: Student Class with Abstraction

```
class Student {
 constructor(name, rollNumber) {
  this.name = name; // Public property
  this.rollNumber = rollNumber; // Public property
 } // Public method to display student details and final grade
 displayDetails() {
  const grade = this.#calculateGrade();
  console.log(`Student Name: ${this.name}`);
  console.log(`Roll Number: ${this.rollNumber}`);
  console.log(`Final Grade: ${grade}`);
 } // Private method (implementation detail)
 #calculateGrade() {
  // Simulate complex grade calculation logic
  console.log("Calculating grade based on performance...");
  return "A"; // Simplified grade logic for demonstration
 } } // Using the Student class
const student1 = new Student("Ankit Kumar", 101);
student1.displayDetails();
```

Output:

```
Calculating grade based on performance...
Student Name: Ankit Kumar
Roll Number: 101
Final Grade: A
// student1.#calculateGrade(); // Error: Private field '#calculateGrade' must be declared in an enclosing class
```

```
if (this.marks >= 90 && this.marks <= 100) { return "A"; } else if (this.marks >= 80 && this.marks < 90) { return "B"; } else if (this.marks >= 70 && this.marks < 80) { return "C"; } else if (this.marks >= 60 && this.marks < 70) { return "D"; } else if (this.marks < 60) { return "Fail"; } else { return "Invalid Marks"; }
```

Example 3: Abstraction with Closures

Using closures to create private variables and methods.

```
function createCounter() {
 let count = 0; // Private variable
 return {
  increment() {
   count++;
   console.log(`Count: ${count}`);
  },
  decrement() {
   count--;
   console.log(`Count: ${count}`);
  },
  getCount() {
   return count; // Exposing the value through a method
  },};}
const counter = createCounter();
counter.increment(); // Output: Count: 1
counter.increment(); // Output: Count: 2
console.log(counter.getCount()); // Output: 2
// console.log(counter.count); // Undefined (private variable)
```

❖ Encapsulation:

- it is the practice of bundling data (properties) and methods (functions) into a single unit, typically a class, and restricting direct access to some of the object's components.

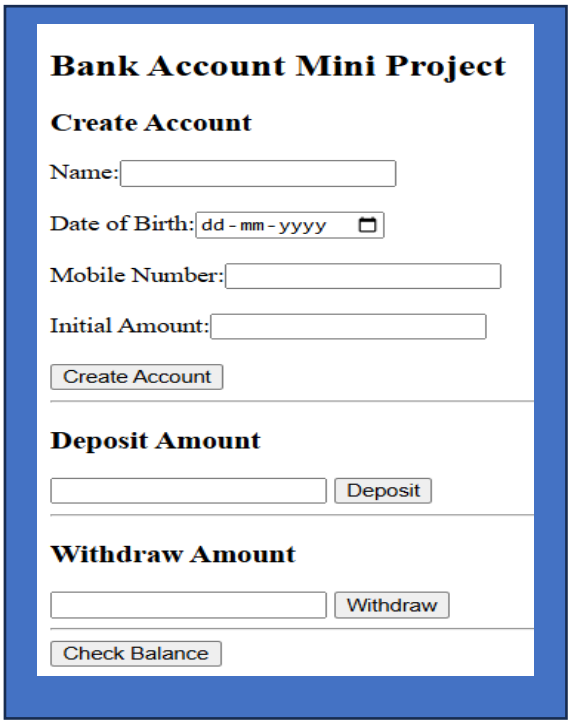
Key Features of Encapsulation

1. **Data Hiding:**
   - Internal details of an object (e.g., private variables) are hidden from the outside.
   - Access is controlled through getter and setter methods or private fields.
2. **Controlled Access:**
   - Public methods provide controlled access to the hidden data.
   - Prevents unintended interference or misuse.
3. **Improved Maintainability:**
   - Internal logic can be changed without affecting external code that interacts with the object.

Example: Bank Account with Encapsulation

```
class BankAccount { constructor(owner, balance) {
this.owner = owner; // Public property
this.#balance = balance; // Private property
} // Getter for balance (read-only access)
getBalance() {
return `The balance for ${this.owner} is $${this.#balance}.`;
} // Method to deposit money (controlled modification)
deposit(amount) { if (amount > 0) {
this.#balance += amount; console.log(`$${amount} deposited successfully.`);
} else {
console.log("Deposit amount must be greater than 0.");
} }// Method to withdraw money (controlled modification)
withdraw(amount) {
if (amount > 0 && amount <= this.#balance) { this.#balance -= amount;
console.log(`$${amount} withdrawn successfully.`);
} else {
console.log("Insufficient balance or invalid amount.");
} } // Private property
#balance;
}// Using the BankAccount class
const account = new BankAccount("Sangam", 1000);
// Public access through methods
console.log(account.getBalance()); // Output: The balance for Sangam is $1000.
account.deposit(500); // Output: $500 deposited successfully. console.log(account.getBalance()); //
Output: The balance for Sangam is $1500. account.withdraw(300); // Output: $300 withdrawn
successfully. console.log(account.getBalance()); // Output: The balance for Sangam is $1200.
// Trying to directly access the private property
// console.log(account.#balance); // Error: Private field '#balance' must be declared in an enclosing
class
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Bank Account Mini Project</title>
</head>
<body>
  <h2>Bank Account Mini Project</h2>
  <h3>Create Account</h3>
  Name:<input type="text" id="name"><br><br>
  Date of Birth:<input type="date" id="dob"><br><br>
  Mobile Number:<input type="number" id="mobile"><br><br>
  Initial Amount:<input type="number" id="iamount"><br><br>
  <button onclick="createAccount()">Create Account</button>
  <hr>
  <h3>Deposit Amount</h3>
  <input type="number" id="depositAmount">
  <button onclick="depositMoney()">Deposit</button><hr>
  <h3>Withdraw Amount</h3>
  <input type="number" id="withdrawAmount">
  <button onclick="withdrawMoney()">Withdraw</button><hr>
  <button onclick="checkBalance()">Check Balance</button>
  <p id="output"></p>
  <script>
```




```
  <script>
    // BankAccount class (Abstraction)
    class BankAccount {
      #balance; // private property
      constructor(owner, initialAmount) {
        this.owner = owner;
        this.#balance = initialAmount;
      }
      getBalance()
      {
    return `Balance of ${this.owner} is ₹${this.#balance}`;
    }
      deposit(amount) {
        if (amount > 0) {
          this.#balance += amount;
          return `₹${amount} deposited successfully.`;
        }
        return "Invalid deposit amount.";
      }
      withdraw(amount) {
        if (amount > 0 && amount <= this.#balance) {
          this.#balance -= amount;
          return `₹${amount} withdrawn successfully.`;
        }
        return "Insufficient balance or invalid amount.";
      }
    }
    let account = null;
```

```
  <script>
    let account = null;
    function createAccount() {
      const name = document.getElementById("name").value;
      const dob = document.getElementById("dob").value;
      const mobile = document.getElementById("mobile").value;
      const initialAmount = Number(document.getElementById("iamount").value);
      if (name === "" || dob === "" || mobile === "" || initialAmount <= 0) {
        document.getElementById("output").innerText ="fill all details ";
        return;}
      account = new BankAccount(name, initialAmount);
      document.getElementById("output").innerText =
        `Account created for ${name} with initial balance ₹${initialAmount}`;
    }
    function depositMoney() {
      if (!account) { alert("Create account first");  return;}
      const amt = Number(document.getElementById("depositAmount").value);
      document.getElementById("output").innerText =account.deposit(amt);
    }
    function withdrawMoney() {
      if (!account) {alert("Create account first");return;}
      const amt = Number(document.getElementById("withdrawAmount").value);
      document.getElementById("output").innerText =account.withdraw(amt);
    }
    function checkBalance() {
      if (!account) {alert("Create account first"); return;}
      document.getElementById("output").innerText =account.getBalance();}
  </script>
</body></html>
```

OOPS Concepts Used (In Short)

- **Class:** BankAccount is used to represent a bank account structure.
- **Object:** account object represents a real bank account.
- **Encapsulation:** Data and methods are wrapped inside the class.
- **Abstraction:** User interacts only with methods like deposit() and withdraw().
- **Private Data Member:** #balance hides actual balance from outside access.
- **Constructor:** Initializes account details and initial balance.

OOPS CONCEPTS USED IN THIS PROJECT

❖ Class

- What is used:
- class BankAccount { ... }

Why used:

- A class is a blueprint to create bank accounts.
- It groups data (owner, balance) and methods (deposit, withdraw) together.
- Makes the program organized and reusable.

Object

- What is used:
- account = new BankAccount(name, initialAmount);

Why used:

- An object represents a real bank account.
- Each object has its own balance and owner.
- Allows multiple accounts in real applications.

Encapsulation

- What is used:
- #balance;

Why used:

- Encapsulation means binding data and methods together.
- The balance is protected inside the class.
- Balance cannot be changed directly from outside the class.

Benefit: Prevents unauthorized access to account balance.

Abstraction

- What is used:
- deposit(), withdraw(), getBalance()

Why used:

- User does not know how balance is stored or updated.
- User only uses simple methods.
- Internal logic is hidden.

Benefit: Reduces complexity and improves security.

Private Data Members

- What is used:
- #balance

Why used:

- Ensures balance cannot be accessed directly like:

account.balance

- Balance is modified only through methods.

Benefit: Data safety and controlled access.

Constructor

- What is used:
- constructor(owner, initialAmount)

Why used:

- Automatically initializes account details.
- Sets initial balance at account creation.
- Ensures account starts in a valid state.

Browser Object Model (BOM)

- **Browser Object Model (BOM) is used to interact with the browser using JavaScript.**
- The default browser object is window.
- All browser functions can be called using window or directly without it.
- Example: <script> alert("Welcome to T3 Skills Center"); </script>

Note: window.alert() and alert() both are the same.

❖ Window Object

- The window object represents the browser window.
- It is automatically created by the browser.
- window is a browser object, not a JavaScript object. JavaScript objects are: String, Array, Date, etc.

S.no	Method	Description
1	alert()	displays the alert box containing message with ok button.
2	confirm()	displays the confirm dialog box containing message with ok and cancel button.
3	prompt()	displays a dialog box to get input from the user.
4	open()	opens the new window
5	close()	closes the current window.
6	setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

1. alert() Method: The alert() method displays a dialog box with a specified message and an **OK** button. It is commonly used to **inform users**, show notifications, or display important messages. **Syntax:** alert(message);

- **message**: The text or value (string/variable) to be displayed in the alert box.

Example 1: Simple Alert alert("This is an alert box!");

Example 2: Alert Using a Variable

```
<!DOCTYPE html>
<html> <body>
<script "> function msg() {
let userName = "Sangam Kumar"; alert("Welcome, " + userName + "!");
}
</script>
<input type="button" value="Click Here" onclick="msg()" />
</body> </html>
```

2. confirm() Method:

- confirm() method displays a **confirmation dialog box** with a specified message and two buttons: **OK** and **Cancel**. It is commonly used when the application needs **user**

➢ **confirmation before performing an important action**.

- Returns **true** if the user clicks **OK** → Returns **false** if the user clicks **Cancel**

Syntax: - let result = confirm(message); **message:** text string displayed in confirmation dialog box.

Example 1:

```
let userConfirmed = confirm("Do you want to proceed?"); if (userConfirmed) {
// User clicked OK
} else { // User clicked Cancel }
```

Note: Useful for **delete, logout, exit, or critical operations**.

Example 2: Confirmation with Action

```
<!DOCTYPE html>
<html> <body>
<input type="button" value="Delete Record" onclick="msg()" />
<script>
function msg() {
let v = confirm("Are you sure you want to delete this record?"); if (v === true) {
alert("Your record has been deleted successfully!");
} else {
alert("Operation cancelled!");
} }
</script>
</body></html>
```

3.prompt() Method:

- **The prompt()** method displays a dialog box that asks the user to enter a value.
- It returns the entered value as a string. If the user clicks Cancel, the method returns null.
- **Syntax: -** let result = prompt (message, default);
- **message:** The text displayed in the prompt dialog box.
- **default (optional):** A default value shown in the input field.

Example 1: Basic Prompt Usage

```
let userInput = prompt("Please enter your name:", "John Doe"); if (userInput !== null) {
console.log("Hello, " + userInput);
}
```

Example 2: Prompt with Button Click

```
<!DOCTYPE html>
<html>
<body>
<input type="button" value="Click Here" onclick="msg()" />
<script type="text/javascript"> function msg() {
let v = prompt("Who are you?"); alert("I am " + v);
}
</script>
</body></html>
```

4.open() Method:

- **window.open()** method is used to open a new browser window or a new tab. It can open an external website or your own HTML file, and also allows you to control the size, position, and appearance of the new window.
- **Syntax:-** window.open(URL, name, features);

Parameters

1.URL

- The web address or file path to open.
- If omitted, a blank window is opened.

2.name (Optional)

- Specifies the name or target of the window.
- Common values:

- blank → opens in a new window/tab
- _self → opens in the same window
- _parent, _top
- Or a custom name (used to reuse the same window)

3.features (Optional)

- A comma-separated string that defines window properties such as:
- width, height
- left, top
- scrollbars, toolbar, etc.

Example 1: Open an External Website

```
window.open( "https://www.ridbharat.com", "_blank", "width=500,height=500"
);
```

Example 2: Open a Website and Your Own HTML File

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Window Open Example</title>
</head>
<body>
<!-- Button to open an external website -->
<button type="button" onclick="openWebsite()">Open Website</button>
<!-- Button to open your own HTML file -->
<button type="button" onclick="openOwnFile()">Open My File</button>
<script>
// Opens an external website in a new window
function openWebsite() { window.open(
"https://www.ridtech.in", "_blank",
"width=300,height=300,left=100,top=50"
);
}
// Opens your own HTML page stored in the same folder
function openOwnFile() { window.open(
"onlinetest.html", // Your HTML file name "OnlineTest",          // Window name
"width=600,height=400,left=200,top=100"
);
}
</script>
</body>
</html>
```

5.close() Method:- The window.close() method is used to **close the current browser window**. For security reasons, modern browsers allow this method to close **only those windows or tabs that were opened using window.open()** through JavaScript.

Syntax: -window.close(); or referenceToWindow.close();

Example 1:- Basic Usage

```
window.open('https://example.com', '_blank');
// Opens a new window window.close(); //
Attempts to close the current window
```

Note: This will work only if the current window was opened by JavaScript.

Example 2: Open and Close a Window Using Buttons

project-3 > index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Window Close Example</title>
</head>
<body><!-- Button to open a new window -->
  <button type="button" onclick="openWebsite()">Open Website</button>
  <!-- Button to close the opened window -->
  <button type="button" onclick="closeWebsite()">Close Window</button>
  <script>
    let myWin; // Variable to store reference of the opened window
    function openWebsite() {
      myWin = window.open(
        "https://www.ridtech.in","_blank",
        "width=300,height=300,left=100,top=150"
      );}
    function closeWebsite() {
      if (myWin) {myWin.close();}
    }
  </script>
</body>
</html>
```

6.setTimeout(): this Method executes a function **after a specified delay (in milliseconds)**. It is used to **delay actions**, such as showing messages, alerts, or loading data after some time.

Syntax: → setTimeout(function, milliseconds);

- **function**: Code to be executed after the delay
- **milliseconds**: Time delay (1000 ms = 1 second)

Example: Execute Code After Delay and Stop It

Example-1:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <button type="button" class="c1">display data</button>
  <button  type="button" class="c2">stop</button>
  <h1 id="d1"></h1>
  <h1 id="d2"></h1>
<script>
```

```
const dpd=document.querySelector(".c1")
let setdata
 dpd.addEventListener("click", ()=>{
  setdata=setTimeout(()=>{
     document.querySelector("#d1").innerHTML="After 3 second data will display"
  }, 3000)
 })
 const stp=document.querySelector(".c2")
 stp.addEventListener("click", ()=>{
  clearTimeout(setdata)
  document.querySelector("#d2").innerHTML="your data will not display"
 })
</script>
</body>
</html>
```

DOCUMENT OBJECT MODEL(DOM)

❖ What is the DOM?

- The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content dynamically.

❖ The Virtual DOM:

- React introduces the concept of a "virtual DOM." Instead of directly manipulating the actual DOM, React creates and maintains a lightweight virtual representation of it in memory. This virtual DOM is a tree-like structure that mirrors the actual DOM's structure.

❖ Tree-like Structure:

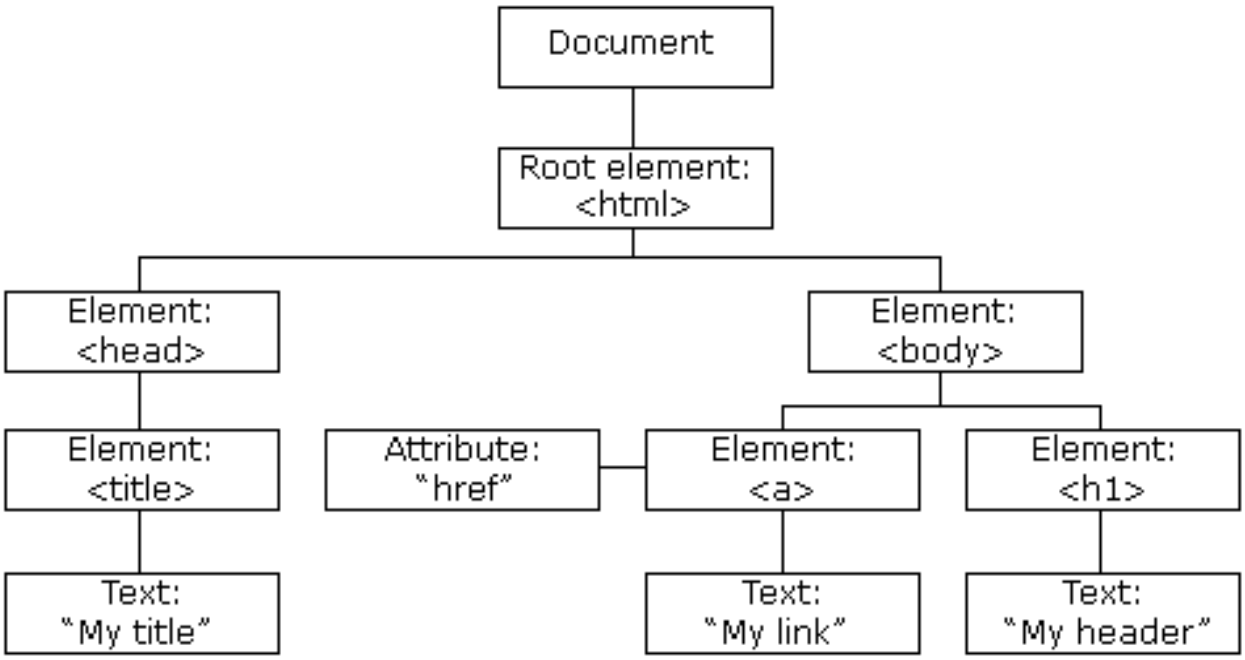
- The DOM represents an HTML or XML document as a hierarchical tree structure. Each element in the document, such as HTML tags, text, attributes, and comments, is represented as a node in this tree. These nodes are organized in a parent-child relationship, where the document itself is the root node.

❖ Node Types:

➢ There are several types of nodes in the DOM, including:

- **Element Nodes:** Represent HTML elements like , <p>, or <a>.
- **Text Nodes:** Contain text within an element.
- **Attribute Nodes:** Store attributes of elements.
- **Comment Nodes:** Contain comments within the HTML.
- **Document Node:** Represents the entire HTML document.

HTML DOM Tree of Objects



❖ What is the HTML DOM?

➢ The HTML DOM is a standard object model and programming interface for HTML. It defines:

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

In other words: HTML DOM is a standard for how to get, change, add, or delete HTML elements.

DOM METHODS

- HTML DOM methods are actions. HTML DOM properties are values

Note: The HTML DOM can be accessed with JavaScript (and with other programming languages).

➢ The programming interface is the properties and methods of each object.

- A **property is a value** that you can get or set (like changing the content of an HTML element).
- A **method is an action** you can do (like add or deleting an HTML element).

❖ Methods for Selecting Elements:

1. **document.getElementById(id):** Selects an element by its id attribute.
2. **document.getElementsByClassName(className):** Returns a live HTMLCollection of elements with the specified class name.
3. **document.getElementsByTagName(tagName):** Returns a live HTMLCollection of elements with the specified tag name.
4. **document.querySelector(selector):** Selects first element that matches the CSS selector.
5. **document.querySelectorAll(selector):** Selects all elements that match the CSS selector.

```html
<!DOCTYPE html>
<html>
<body>
  <!-- Elements for selection -->
  <h2 id="myId">ID Selector</h2>
  <p class="myClass">Class Selector 1</p>
  <p class="myClass">Class Selector 2</p>
  <span>Tag Selector</span>
  <div id="output"></div>
  <script>
    // 1. getElementById
    let idEl = document.getElementById("myId");
    idEl.style.color = "blue";
    // 2. getElementsByClassName
    let classEls = document.getElementsByClassName("myClass");
    classEls[0].style.color = "green";
    // 3. getElementsByTagName
    let tagEls = document.getElementsByTagName("span");
    tagEls[0].innerHTML = "Tag Selected";
    // 4. querySelector
    let firstPara = document.querySelector(".myClass");
    firstPara.style.fontWeight = "bold";
    // 5. querySelectorAll
    let allParas = document.querySelectorAll(".myClass");
    allParas[1].style.textDecoration = "underline";
```

ID Selector

Class Selector 1

Class Selector 2

Tag Selected

New Element Created

Note: [0] is used with selectors that return multiple elements such as getElementsByClassName(), getElementsByTagName(), and querySelectorAll() to access the first element.

Example

```
let items = document.getElementsByClassName("myClass");
items[0].style.color = "red";
```

✓ items[0] → first element

✓ items[1] → second element

Note-1:- 0] is an index used to access the first element from a group of elements returned by a selector.

Changing Element Content

There are following method are used for the changing the element

1) Set or get the plain text content of an
   - **element.element.textContent,**
   - **element.innerHTML,**
   - **element.innerText**
2) Changing Element Styles with **style.property = value**
3) Creating and Appending New Elements in the DOM: **document.createElement(tagName)**
4) Modifying Element Attributes with **setAttribute(name, value)**
5) Removing Elements from the DOM Using **element.remove():**
6) Replacing Elements in the DOM Using **element.replaceWith(newElement):**

1. Set or Get Plain Text Content of an Element

- JavaScript provides **three main properties** to **read (get)** or **change (set)** the content of an HTML element:
  - **A. textContent**
  - **B. innerHTML**
  - **C. innerText**

A. element.textContent

- textContent is used to **set or get only text** inside an element.
- It **does not read HTML tags**, only plain text.

Syntax

- element.textContent = value; → // set
- element.textContent; → // get

Example :- document.getElementById("box1").textContent = "Hello TextContent";

B. element.innerHTML

- innerHTML is used to **set or get text with HTML tags**.
- It allows inserting **HTML elements** inside another element.

Syntax

- element.innerHTML = value; → // set
- element.innerHTML; → // get

Example document.getElementById("box2").innerHTML = "Hello InnerHTML";

C. element.innerText

- innerText is used to **set or get only visible text**.
  It ignores hidden text and does not show HTML tags.

Syntax

- element.innerText = value; → // set
- element.innerText; → // get

Example:- document.getElementById("box3").innerText = "Hello InnerText";

```
project-3 > index.html > ...
<!DOCTYPE html>
<html>
<body>
  <h3 id="t1">Research </h3>
  <h3 id="t2"><b>Innovation</b></h3>
  <h3 id="t3">Discovery</h3>
  <button onclick="getData()">Get Data</button>
  <button onclick="setData()">Set Data</button>
  <p id="output"></p>
  <script>
    function getData() {
      let a = document.getElementById("t1").textContent;
      let b = document.getElementById("t2").innerHTML;
      let c = document.getElementById("t3").innerText;
      document.getElementById("output").innerHTML =
        "textContent: " + a + "<br>" +
        "innerHTML: " + b + "<br>" +
        "innerText: " + c;
    }
    function setData() {
      document.getElementById("t1").textContent = "TextContent Changed: Workshop";
      document.getElementById("t2").innerHTML = "<i>InnerHTML Changed: Internship</i>";
      document.getElementById("t3").innerText = "InnerText Changed: Training ";
    }
  </script>
</body>
</html>
```

TextContent Changed: Workshop

InnerHTML Changed: Internship

InnerText Changed: Training

Get Data | Set Data

textContent: Research
innerHTML: **Innovation**
innerText: Discovery

Research

Innovation

Discovery

Get Data | Set Data

textContent: Research
innerHTML: **Innovation**
innerText: Discovery

2. Changing Element Styles in JavaScript

- style property is used to **change the CSS style of an HTML element dynamically**.
- Each CSS property is written in **camelCase** in JavaScript.

Example: background-color → backgroundColor
font-size → fontSize

```
<!DOCTYPE html>
<html>
<body>
  <h2 id="title">JavaScript Style Change Example</h2>
  <p id="text">Click the buttons below to change my style dynamically.</p>
  <button onclick="changeColor()">Change Color</button>
  <button onclick="changeSize()">Change Size</button>
  <button onclick="changeBackground()">Change Background</button>
  <button onclick="resetStyle()">Reset</button>
  <script>
    function changeColor() {document.getElementById("text").style.color = "red";}
    function changeSize() {document.getElementById("text").style.fontSize = "22px"; }
    function changeBackground() {document.getElementById("text").style.backgroundColor = "yellow";}
    function resetStyle() {
      let el = document.getElementById("text");
      el.style.color = "black";
      el.style.fontSize = "16px";
      el.style.backgroundColor = "transparent";
    }
  </script>
</body>
</html>
```

JavaScript Style Change Example

Click the buttons below to change my style dynamically.

Change Color | Change Size | Change Background | Reset

```
<!DOCTYPE html>
<html>
<head>
    <title>Change Element Style</title>
    <style>
        #example {
            width: 200px;
            height: 100px;
            background-color: lightblue;
            color: black;
            text-align: center;
            line-height: 100px;
        }
    </style>
</head>
<body>
    <div id="example">Original Style</div>
    <button onclick="change()">Change Style</button>
    <script>  // Define the change function
        function change() {
            let element = document.getElementById("example");   // Select the element
            element.style.backgroundColor = "yellow";  // Modify CSS styles using style.property
            element.style.color = "red";
            element.style.border = "2px solid green";
            element.style.fontSize = "20px";
            element.style.padding = "10px";
            // Output the modified element to the console
            console.log(element.outerHTML);
        }
    </script></body></html>
```

Original Style

Change Style

Original Style

Change Style

How to create new tag and add dynamic content

➔ **document.createElement(), document.createTextNode()**, and **appendChild()** are used together to create and display HTML content dynamically. First, an element is created, then text is added to it using a text node, and finally the element is inserted into the webpage using appendChild().

1. document.createElement(tagName)

- This method is used to create a new HTML element dynamically using JavaScript.
- **Syntax:** document.createElement("tagName");

2. document.createTextNode(text)

- This method is used to create a text node containing the specified text. It is mainly used to add text safely inside an HTML element.
- **Syntax:** document.createTextNode("text");

3. appendChild(node)

- appendChild() method is used to add a new element or text node as the **last child** of a specified parent element.

Syntax: parentElement.appendChild(childNode);

Combine Syntax:

```
let element = document.createElement("tagName");
let text = document.createTextNode("Your text here");
element.appendChild(text);
parentElement.appendChild(element);
```

➢ Append to <body> Ex:- document.body.appendChild(element);

Syntax (Append to a specific parent element)

```
<div id="d3"></div>
<script>
let element = document.createElement("tagName");
let text = document.createTextNode("Your text here");
element.appendChild(text);
let parentElement = document.getElementById("d3");
parentElement.appendChild(element);
</script>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>DOM Example</title>
</head>
<body>
    <h1>This is DOM</h1>
    <div id="d2"></div>
    <script>
        let newele = document.createElement("h2"); // Create a new h2 element
        let text = document.createTextNode("This is  Sangam "); // Create a text node
        newele.appendChild(text); // Append text to h2
        document.body.appendChild(newele); // Append h2 to the body
        // Create the p tag and add inside the div
        let name = "Sangam";
        let ptag = document.createElement("p");
        let data = document.createTextNode("Hi " + name);
        ptag.appendChild(data);
        document.getElementById("d2").appendChild(ptag);
    </script>
</body></html>
```

This is DOM

Hi Sangam

This is Sangam

```html
<!DOCTYPE html>
<html>
<body>
<h2>Dynamic List Example</h2>
<button onclick="addListItem()">Add List Item</button>
<ul id="myList"></ul>
<script>
    // Array declared outside so values are remembered
    let names = ["Mohan", "Raj", "Sohan", "Rohan", "Sangam"];
    function addListItem() {
        // Check if array is not empty
        if (names.length > 0) {
            let li = document.createElement("li");
            // Get last value and remove it from array
            li.textContent = names.pop();
            // Append li to ul
            document.getElementById("myList").appendChild(li);
        } else {
            alert("No more names in the array!");
        }
    }
</script>
</body></html>
```

```html
<!DOCTYPE html>
<html>
<body>
<h2>Dynamic List Example</h2>
<button onclick="addListItem()">Add List Item</button>
<ul id="myList"></ul>
<script>
    // Original array (will NOT change)
    const names = ["Mohan", "Raj", "Sohan", "Rohan", "Sangam"];
    // Index to track current position
    let index = 0;
    function addListItem() {
        if (index < names.length) {
            let li = document.createElement("li");
            li.textContent = names[index];
            document.getElementById("myList").appendChild(li);
            index++; // move to next value
        } else {
            alert("No more names in the array!");
        }
    }
</script>
</body></html>
```

Dynamic List Example

Add List Item

- Mohan
- Raj
- Sohan
- Rohan
- Sangam

Dynamic Table Example

Add Row

Name	Subject	Marks
Mohan	Math	85
Raj	Science	90

```html
<!DOCTYPE html>
<html>
<body>
<h2>Create Button Example</h2>
<div id="buttonContainer"></div>
<button onclick="createButton()">Create Button</button>
<script>
function createButton() {
  // Create a new button element
  let btn = document.createElement("button");
  // Add text to the button
  btn.textContent = "Click Me!";
  // Add a click event
  btn.onclick = function() {
    alert("Button Clicked!");
  };// Append button to the container
  document.getElementById("buttonContainer").appendChild(btn);
}
</script>
</body>
</html>
```

```html
<!DOCTYPE html><html><body>
<h2>Dynamic Table Example</h2>
<button onclick="createTable()">Add Row</button><br><br>
<div id="tableContainer"></div>
<script>
    const headings = ["Name", "Subject", "Marks"]; // Heading array
    const tableData = [  // Table data array (unchanged)
        ["Mohan", "Math", 85],
        ["Raj", "Science", 90],
        ["Sohan", "English", 78],
        ["Rohan", "Computer", 88]];
    let index = 0;          // tracks row position
    let table = null;       // stores table reference
    function createTable() {
        // Create table only once
        if (table === null) {
            table = document.createElement("table");
            table.border = "1";
            table.cellPadding = "8";
            // Create heading row
            let headerRow = document.createElement("tr");
            for (let head of headings) {
                let th = document.createElement("th");
                th.textContent = head;
                headerRow.appendChild(th);
            }
            table.appendChild(headerRow);
            document.getElementById("tableContainer").appendChild(table);
        }// Add one row per click
        if (index < tableData.length) {
            let tr = document.createElement("tr");
            for (let cell of tableData[index]) {
                let td = document.createElement("td");
                td.textContent = cell;
                tr.appendChild(td);
            }
            table.appendChild(tr);
            index++;    // move to next row
        } else {alert("No more data available!"); }
    }</script></body></html>
```

Create Input

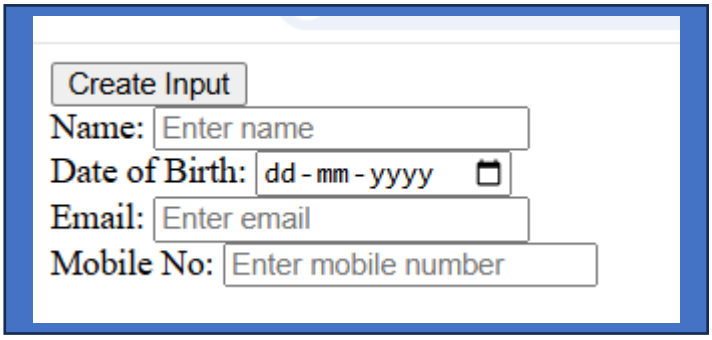
Name: Enter name

Name: Enter name

Name: Enter name

Name: Enter name

```html
<!DOCTYPE html>
<html>
<body>
<button onclick="createInput()">Create Input</button>
<div id="box"></div>
<script>
function createInput() {
    // Create label
    let label = document.createElement("label");
    label.textContent = "Name: ";
    // Create input
    let input = document.createElement("input");
    input.type = "text";
    input.placeholder = "Enter name";
    // Append label and input
    document.getElementById("box").appendChild(label);
    document.getElementById("box").appendChild(input);
    document.getElementById("box").appendChild(document.createElement("br"));
}
</script>
</body>
</html>
```




```html
<!DOCTYPE html>
<html>
<body>
<button onclick="createInput()">Create Input</button>
<div id="box"></div>
<script>// Field details array
    const fields = [
        { label: "Name", type: "text", placeholder: "Enter name" },
        { label: "Date of Birth", type: "date" },
        { label: "Email", type: "email", placeholder: "Enter email" },
        { label: "Mobile No", type: "tel", placeholder: "Enter mobile number" }];
    let index = 0;
    function createInput() {
        if (index < fields.length) {
            let label = document.createElement("label"); // Create label
            label.textContent = fields[index].label + ": ";
            let input = document.createElement("input"); // Create input
            input.type = fields[index].type;
            input.placeholder = fields[index].placeholder || "";
            let box = document.getElementById("box"); // Append to div
            box.appendChild(label);
            box.appendChild(input);
            box.appendChild(document.createElement("br"));
            index++; // move to next field
        } else {alert("All input fields are created!");}
    }
</script></body></html>
```

Explanation: Dynamic Input Fields Step-by-Step

1. **Array of fields:**
   fields stores the input details: label, type, and optional placeholder.
2. **Index tracker:**
   index keeps track of which field should be created next.
3. **Button click function:**
   - Checks if there are remaining fields (index < fields.length)
   - Creates a **label** and an **input** element dynamically
   - Sets input type and placeholder
   - Appends them to the
   - Adds a
 for spacing
   - Increments index for the next click
4. **End of array:**
   When all fields are created, an **alert** appears: "All input fields are created!"

3.Modifying Element Attributes with setAttribute(name, value)

- The setAttribute method is used to change the value of an existing attribute or to add a new attribute to an HTML element. This method is versatile and works for any valid attribute.
- **Syntax:** element.setAttribute(attributeName, attributeValue);
  - ✓ **`attributeName`**: name of the attribute to modify or add (e.g., "class", "id", "src").
  - ✓ **`attributeValue`**: The value to assign to the attribute.
  - ✓ **outerHTML:** Returns the **entire HTML element**, including the element itself and its content (opening and closing tags, along with child elements or text).
- **Example:**

```
<!DOCTYPE html>
<html>
<head>
  <title>setAttribute Example</title>
</head>
<body>
  <img id="image" src="logo.jpg" alt=" Image">
  <script>
    // Select the image element
    let a = document.getElementById("image");
    console.log(a.outerHTML)
    console.log(a.innerHTMLHTML)
    // Change the 'src' attribute
    a.setAttribute("src", "example.jpg");
    // Add or modify the 'alt' attribute
    a.setAttribute("alt", "Updated Image Description");
    // Add a new 'title' attribute
    a.setAttribute("title", "Hover text for the image");
    console.log(a.outerHTML);
  </script>
</body>
</html>
```

Output:

```
<img id="image" src="example.jpg"
alt="Updated Image Description" title="Hover text for image">
```

console.log(a.outerHTML)

Why we use it:

- To view the **entire HTML of an element**
- To check the element along with its tag, attributes, and content
- Helpful for debugging structure and attributes

What it shows:

- Opening tag, Attributes,Inner content
- Closing tag

console.log(a.innerHTML)

Why we use it:

- To view **only the content inside an element**
- To check what HTML/text is present **between the opening and closing tags**
- Useful when debugging dynamic content changes

What it shows:

- Child elements and text
- Does **not** include the element's own tag

Example meaning:

console.log(a.innerHTML);

Shows *what is inside* element a

innerHTML Shows only inner content

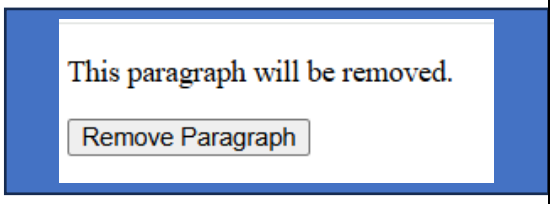
outerHTML Shows full element with tag

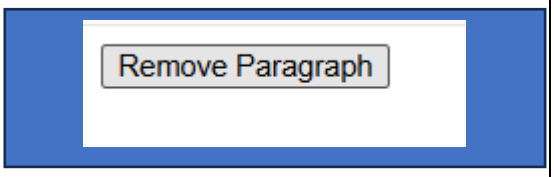
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        #d1 {
            background-color: red;
            color: yellow;
        }
    </style>
</head>
<body>
    <h1>This is JS class</h1>
    <p>Hello Everyone</p>
    <script>
        // getElementsByTagName returns a collection
        let v = document.getElementsByTagName("h1")[0];
        //  setAttribute
        v.setAttribute("id", "d1");
    </script>
</body>
</html>
```

4.Removing Elements from the DOM Using element.remove():

- The element.remove() method is used to remove an element from the DOM. Once removed, the element and its children are no longer part of the document.
- **Syntax:** element.remove(); :- **element**: The DOM element you want to remove.
- **Example:**

```html
<!DOCTYPE html>
<html>
<head>
    <title>Remove Element Example</title>
</head>
<body>
    <div id="container">
<p id="d1">This paragraph will be removed.</p>
<button onclick="removeElement()">Remove Paragraph</button>
    </div>
    <script>
        function removeElement() {
            // Select the element to remove
            let paragraph = document.getElementById("d1");
            // Remove the element from the DOM
            paragraph.remove();
            console.log("Paragraph removed!");
        }
    </script>
</body>
</html>
```

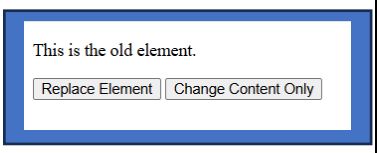


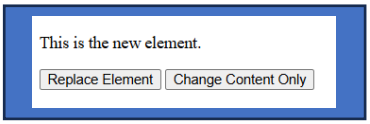


5.Replacing Elements in the DOM Using element.replaceWith(newElement):

- element.replaceWith(newElement) method replaces an existing element in the DOM with a new element (newElement). The original element is removed, and the new element takes its place.
- **Syntax:** element.replaceWith(newElement);
  - ✓ **element**: The DOM element you want to replace.
  - ✓ **newElement**: The new DOM element that will replace the original one.
- **Example:**

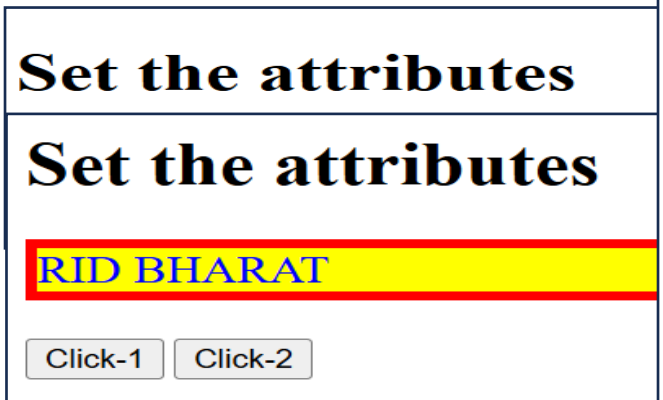
```html
<!DOCTYPE html>
<html>
<head>
    <title>Replace Element Example</title>
</head>
<body>
    <div id="container">
        <p id="d1">This is the old element.</p>
        <button onclick="replaceElement()">Replace Element</button>
        <button onclick="changeContent()">Change Content Only</button>
    </div>
    <script>
        function replaceElement() {
            // Step 1: Select the element to be replaced
            let oldElement = document.getElementById("d1");
            // Step 2: Create the new element
            let newElement = document.createElement("p");
            newElement.id = "d1"; // Set an ID for the new element
            newElement.textContent = "This is the new element.";
            // Step 3: Replace the old element with the new element
            oldElement.replaceWith(newElement);
            console.log("Element replaced!");
        }
        // NEW FUNCTION: Change only the content
        function changeContent() {
            let element = document.getElementById("d1");
            element.textContent = "This is the updated content only.";
            console.log("Content changed only!");
        }
    </script>
</body>
</html>
```





Example-2: Dom_setatributes.html

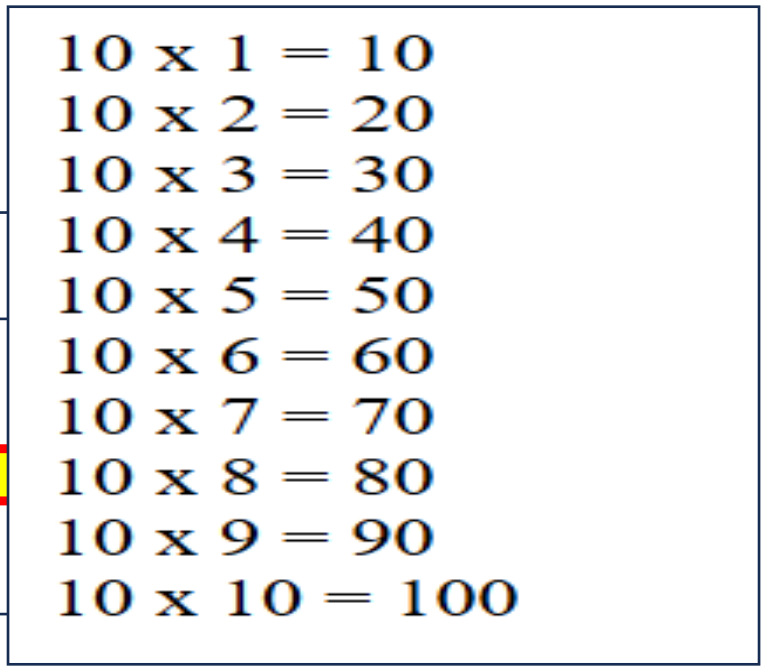
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style>
    .a {
      border: 5px solid red;
    }
    #two {
      background-color: yellow;
      color: blue;
      font-size: 20px;
    }
  </style>
</head>
<body>
  <h1>Set the attributes </h1>
  <p id="one">RID BHARAT </p>
  <button onclick="myFun()">Click-1</button>
  <button onclick="myFun1()">Click-2</button>
  <script>
    function myFun() {
      let element =
document.getElementById("one")
      element.setAttribute("class", "a")
    }
    function myFun1() {
      let element =
document.getElementById("one")
      element.setAttribute("id", "two")
    }
  </script>
</body> </html>
```



Example-3:

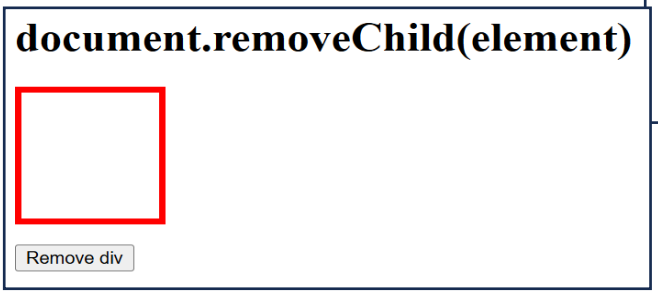
- **dom_write.html:**

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Document</title>
</head>
<style>
</style>
<body>
<h1 id="one"></h1>
<h4 id="two">
<button onclick="myFun()">Click to display</button>
</h4>
<script>
    let a=prompt("Enter the number ")
document.getElementById("one").innerText="Multiplic
ation table of "+a;
    let n=a;
    function myFun(){
for(let i=1;i<10;i++){
document.write(n," x ",i," = ",n*i,"<br>")
    } }
</script> </body> </html>
```



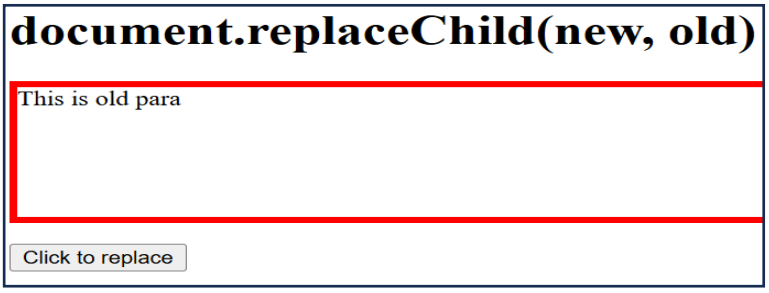
Example-4: REMOVE Element

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<style>
  p {
    height: 100px;
    width: 100px;
    border: 5px solid red;
  }
</style>
<body>
  <h1>document.removeChild(element)</h1>
  <p>
  </p>
  <button onclick="myFun()">Remove
div</button>
  <script>
    let a =
document.getElementsByTagName("p")[0]
    function myFun() {
      if (a) {
        a.parentNode.removeChild(a)
      }
    }
  </script>
</body>
</html>
```



Example-5: dom-replacechild.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-
width, initial-scale=1.0">
  <title>Document</title>
</head>
<style>
  #one {
    height: 100px;
    width: auto;
    border: 5px solid red;
  }
  #two {
    height: 100px;
    width: auto;
    border: 5px solid blue;
  }
</style>
<body>
  <h1>document.replaceChild(new, old)</h1>
  <p id="one">This is old para </p>
  <button onclick="myFun()">Click to
replace</button>
  <script>
    function myFun() {
      let a = document.getElementById("one")
      let b = document.createElement("p")
      b.id = "two"
      let txt = document.createTextNode("This is
new para")
      b.appendChild(txt)
      a.parentNode.replaceChild(b, a)
    }
  </script> </body>  </html>
```



Example-6: Dom.addevent.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<h1>Adding Events Handlers</h1>
<p>Syntax:<br>
document.getElementById(id).onclick = function(){code}
</p>
<button id="one">Click</button>
<script>
  //
document.getElementById("one").onclick = function(){
  //   alert("Button clicked successfully")
  // }
  let btn = document.getElementById("one")
btn.onclick = function(){
alert("Click event added")
  }
</script>
</body>
</html>
```

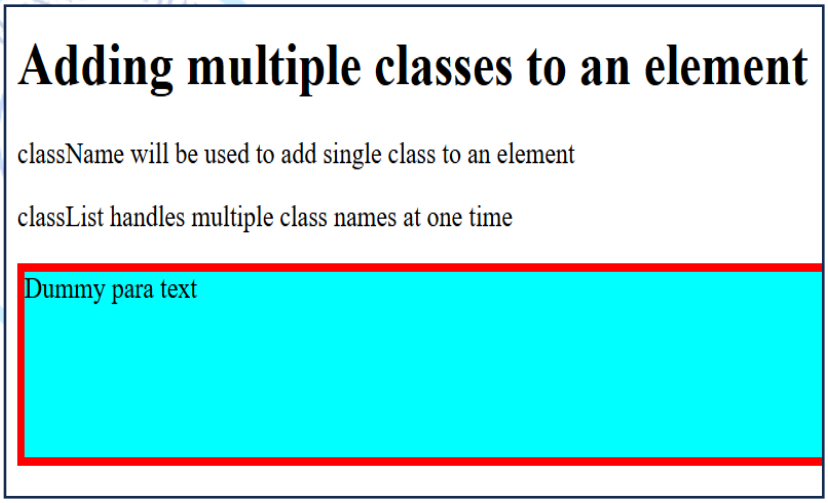
Example-7: create.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
</head>
<style>
  img {
    height: 200px;
    width: 200px;
  }
  table {
    width: 100%;
    text-align: center;
  }
</style>
<body>
  <table border="1">
    <thead>
      <th>Name</th>
      <th>Age</th>
      <th>Image</th>
    </thead>
    <tbody id="t-body">
    </tbody>
  </table>
  <script>
    let data = [
{ "name": "Raj", age: 20, image:
"https://i.postimg.cc/qMdDpVSH/one.jpg" },
{ "name": "Aryan", age: 30, image:
"https://i.postimg.cc/j28D3rJn/two.jpg" },
{ "name": "Kazi", age: 40, image:
"https://i.postimg.cc/Pq2nwrGD/three.jpg" },
{ "name": "Peter", age: 31, image:
"https://i.postimg.cc/L6t9sSst/four.jpg"  ]
    let t = document.getElementById("t-body")
    for (let i = 0; i < data.length; i++) {
      let row = t.insertRow(i)
      let cell1 = row.insertCell(0)
      let cell2 = row.insertCell(1)
      let cell3 = row.insertCell(2)
      let pic = document.createElement("img")
      pic.src = data[i].image
      cell1.innerText = data[i].name
      cell2.innerText = data[i].age
      cell3.appendChild(pic)
    }
  </script>  </body> </html>
```

Example-08:

- **Add_class .html**

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>classList</title>
</head>
<style>
   .one{
      border:5px solid red;}
.two{
      height: 100px;
      background-color: aqua;}
</style>
<body>
<h1>Adding multiple classes to an element</h1>
<p>className will be used to add single class to an element</p>
<p>classList handles multiple class names at one time</p>
<script>
      let p = document.createElement("p")
p.innerText="Dummy para text"
      // p.className="one"
p.classList.add("one")
p.classList.add("two")
document.body.appendChild(p)
</script></body>  </html>
```



Example-10: Important

- **Todo_index.html**

```
<!DOCTYPE html>
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>To-DO List</title>
</head>
<style>
   olli {
      font-size: 30px;
   }
</style>
<body>
   <h1>ToDo List</h1>
   <div id="container">
      <input type="text" id="inp">
      <button id="add">Add</button>
      <ol id="todos"></ol></div>
```

```
<script>
   let inpF = document.getElementById("inp")
   let btn = document.getElementById("add")
   let todo = document.getElementById("todos")
   btn.addEventListener("click", () => {
      let res = document.createElement("li")
      res.innerText = inpF.value
      todo.appendChild(res)
      inpF.value = "" //clear the input field
      res.addEventListener("click", () => {
         res.style.textDecoration = "line-through"
      })
      res.addEventListener("dblclick", () => {
         todo.removeChild(res)
      }) })
</script> </body>  </html>
```

Example-10

Todo_index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Todo Apps </title>
   <style>
      li button:hover {
         cursor: pointer;
         background-color: aqua;
      }
   </style>
</head>
<body>
   <div>
      <input type="text" id="inp" placeholder="Enter your task">
      <button type="button" id="add">Add</button>
      <ol type="1" id="todos"></ol>
   </div>
   <script>
      let inpf = document.getElementById("inp");
      let btn = document.getElementById("add");
      let todo = document.getElementById("todos");
      let v = null;
      // Function to handle adding or editing todos
      function addTodo() {  // Edit functionality
         if (v) {
            v.innerHTML = inpf.value.trim();
            inpf.value = "";
            v = null;
            btn.innerHTML = "Add";
            return;  }
```

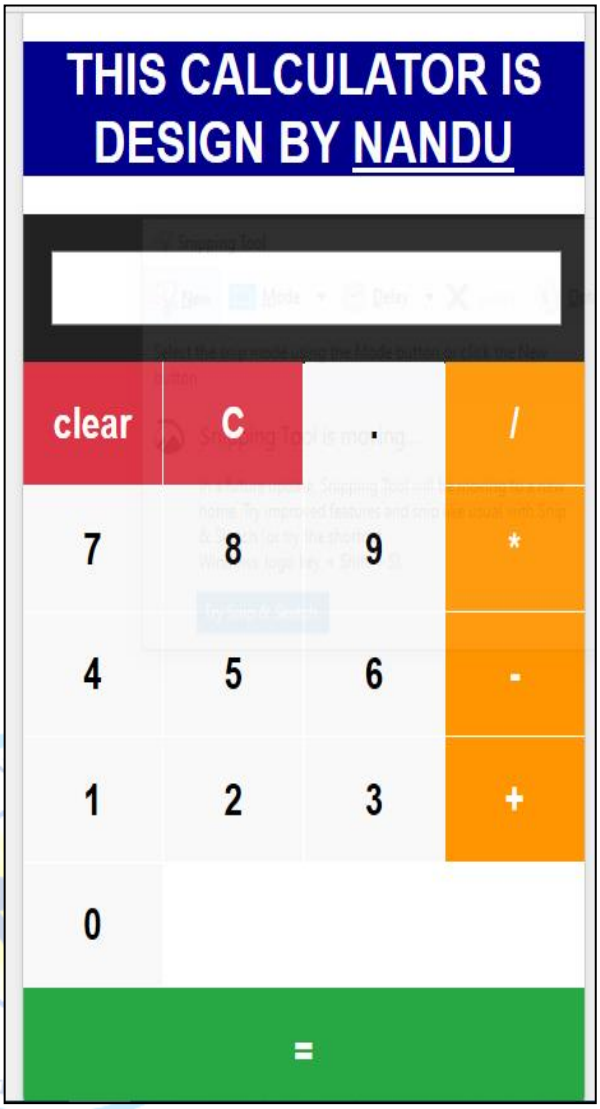
```
// Add functionality
if (inpf.value.trim() !== "") {
  let res = document.createElement("li");
  let sp = document.createElement("span");
  sp.innerHTML = inpf.value;
  res.style.lineHeight = "2em";
  res.append(sp);  // Create buttons
  let btn1 = document.createElement("button");
  let btn2 = document.createElement("button");
  let btn3 = document.createElement("button");
  let btn4 = document.createElement("button");  // Button properties
  btn1.innerHTML = "Mark";
  btn1.style.marginLeft = "10px";
  btn2.innerHTML = "Update";
  btn2.style.marginLeft = "10px";
  btn3.innerHTML = "Remove";
  btn3.style.marginLeft = "10px";
  btn4.innerHTML = "Complete";
  btn4.style.marginLeft = "10px";  // Append buttons to the list item
  res.append(btn1, btn2, btn3, btn4);
  todo.appendChild(res);  // Clear input field
  inpf.value = ""  // Button functionalities
  btn1.addEventListener("click", () => {
    res.style.color = "green";
    res.style.textDecoration = "underline";
  });
  btn2.addEventListener("click", () => {
    inpf.value = sp.innerHTML;
    v = sp;
    btn.innerHTML = "Edit";
  });
  btn3.addEventListener("click", () => {
    todo.removeChild(res);
  });
  btn4.addEventListener("click", () => {
    alert("Your data is updated successfully!");
  });
} else {
  alert("Input cannot be empty!");
}  }
```

```
// Event listener for the Add button
    btn.addEventListener("click", addTodo);
    // Event listener for the Enter key
    inpf.addEventListener("keydown", (event) => {
      if (event.key === "Enter") {
        addTodo();
      }
    });
  </script>
</body>
</html>
```

Calculator project

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport"
  <title>Calculator</title>
  <style>
    body {
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      background-color: #f0f0f0;
      font-family: Arial, sans-serif;}
    h1 {text-align: center;}
    .calculator {
      border: 1px solid #ccc;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0, 0, 0,
0.1);
      overflow: hidden;
      width: 400px;
      background-color: white;}
    .display {
      background-color: #222;
      color: white;
      text-align: center;
      padding: 20px;
      font-size: 2em;}
    .display input {
      width: 100%;
      font-size: 30px;
      font-weight: bold;}
    .buttons {
      display: grid;
      grid-template-columns: repeat(4, 1fr);
      gap: 1px;}
    .buttons button {
      padding: 20px;
      font-size: 1.5em;
      font-weight: bold;
      border: none;
      background-color: #f9f9f9;
      cursor: pointer;
      transition: background-color 0.3s;}
    .buttons button:hover {
      background-color: #e0e0e0;}
background-color: #e0e0e0;}
.buttons button:active {
background-color: #d0d0d0;}
.buttons .operator {
background-color: #ff9500;
color: white;}
.buttons .operator:hover {
background-color: #ff8500;}
.buttons .operator:active {
background-color: #ff7500;}
.buttons .equals {
background-color: #28a745;
color: white;
grid-column: span 4;}
.buttons .equals:hover {
background-color: #218838;}
.buttons .equals:active {
background-color: #1e7e34;}
.buttons .clear {
background-color: #dc3545;
color: white;}
.buttons .clear:hover {
background-color: #c82333;}
.buttons .clear:active {
background-color: #bd2130;}
</style></head> <body>
  <div class="calculator">
 <h1 style="background-color: darkblue; color: white;">THIS
CALCULATOR IS DESIGN BY <strong><U> NANDU</strong></h1>
    <div class="display"><input type="text" readonly /></div>
    <div class="buttons">
      <button class="clear" onclick="clr()">clear</button>
      <button class="clear" onclick="backspace()">C</button>
      <button onclick="fun(this)">.</button>
      <button class="operator" onclick="fun(this)">/</button>
      <button onclick="fun(this)">7</button>
      <button onclick="fun(this)">8</button>
      <button onclick="fun(this)">9</button>
      <button class="operator" onclick="fun(this)">*</button>
      <button onclick="fun(this)">4</button>
      <button onclick="fun(this)">5</button>
      <button onclick="fun(this)">6</button>
      <button class="operator" onclick="fun(this)">-</button>
      <button onclick="fun(this)">1</button>
      <button onclick="fun(this)">2</button>
      <button onclick="fun(this)">3</button>
      <button class="operator" onclick="fun(this)">+</button>
      <button onclick="fun(this)">0</button>
```

```
<button class="equals"
onclick="res()">=</button>
      </div>
   </div>
   <script>
      let inp =
document.getElementsByTagName("input")[0
];
      function fun(e) {
         inp.value += e.innerText;
      }
      function res() {
         try {
            inp.value = eval(inp.value);
         } catch (e) {
            inp.value = "Error";
         }
      }
      function clr() {
         inp.value = "";
      }
      function backspace() {
         inp.value = inp.value.slice(0, -1);
      }
   </script>
</body>
</html>
```



JavaScript HTML DOM Elements

- JavaScript HTML DOM (Document Object Model) elements are the representation of HTML elements within a web page, accessible and manipulable through JavaScript.

1. Accessing DOM Elements:

- **getElementById:** This method allows you to select an element by its unique id attribute.
- **getElementsByClassName:** Select elements based on their class names.
- **getElementsByTagName:** Select elements by their HTML tag name (e.g., "div," "p," "a").
- **querySelector and querySelectorAll:** Use CSS-style selectors to select elements.
- **Traversing the DOM:** Navigate the DOM tree using properties like parentNode, childNodes, nextSibling, and previousSibling to find elements relative to a known reference point.

2. Properties and Methods:

- **textContent:** Gets or sets the text content of an element.
- **innerHTML:** Gets or sets the HTML content of an element.
- **attributes:** Access and manipulate attributes of an element.
- **style:** Access and manipulate CSS styles of an element.
- **addEventListener:** Attach event listeners to respond to user interactions.
- **removeEventListener:** Remove previously attached event listeners.
- **appendChild and removeChild:** Add and remove child elements within parent elements.
- **classList:** Access and manipulate the class attribute to add or remove CSS classes.
- **getAttribute and setAttribute:** Get and set attributes of elements.
- **value:** Access or change the value of form elements like input fields and textareas.
- **parentNode and childNodes:** Access a node's parent and child elements.

3. Common DOM Element Properties:

- **tagName:** Returns the tag name of the element (e.g., "DIV" for a element).
- **id:** Returns the id attribute of the element.
- **className:** Returns the value of the class attribute.
- **href:** Returns the href attribute for anchor elements (<a>).
- **src:** Returns the src attribute for image elements (<img>).
- **innerHTML:** Returns or sets the HTML content of the element.

4. Events:

- JavaScript can be used to attach event listeners to DOM elements to respond to user interactions such as clicks, mouse movements, keyboard inputs, etc.
- Common events include click, mouseover, keydown, submit, and more.Event listeners can be added using addEventListener() and removed using removeEventListener().

5. Modifying DOM Elements:

- JavaScript can dynamically modify the content, structure, and style of HTML elements in response to user actions or other events.
- Elements can be created, removed, replaced, and manipulated as needed.

6. Security Considerations:

- Care should be taken when working with DOM to prevent cross-site scripting (XSS) attacks.
- Always sanitize and validate input and avoid injecting untrusted data into the DOM.

HTML DOM Events

- **HTML DOM Events** are actions or interactions that happen on a web page and can be handled using JavaScript to make websites **interactive and dynamic**.

❖ What are Events?

- Events occur due to **user actions** (click, type, submit) or **browser actions** (page load, unload).

➢ Common Event Types

- **Mouse Events:** click, mousemove
- **Keyboard Events:** keydown, keyup
- **Form Events:** input, change, submit
- **Focus Events:** focus, blur
- **Document Events:** DOMContentLoaded, load

➢ Event Handling

- Event handling means **writing a function that runs when an event occurs**.
- element.addEventListener("click", myFunction);

➢ Event Object

- When an event happens, JavaScript creates an **event object** that gives details like:
  1. event type
  2. target element

➢ Event Propagation

- Events move in the DOM in two ways:
  1. **Capturing**
  2. **Bubbling** (most common)

➢ Control using:

- event.stopPropagation();
- event.preventDefault();

➢ Event Delegation

- Use **one event listener on a parent** to handle events of many child elements.

➢ Removing Events

- Remove events when not needed:
- element.removeEventListener("click", myFunction);

❖ Why Events are Important?

- Handle button clicks
- Validate forms
- Respond to user input
- Create interactive UI

Note: - JavaScript DOM events allow web pages to respond to user and browser actions dynamically.

❖ DOM EVENTS

1. **Mouse Events:**
   - **click:** Triggered when an element is clicked.
   - **dblclick:** Triggered on a double-click.
   - **mouseover:** Triggered when the mouse enters an element.
   - **mouseout:** Triggered when the mouse leaves an element.
2. **Keyboard Events:**
   - **keydown:** Triggered when a keyboard key is pressed down.
   - **keyup:** Triggered when a keyboard key is released.
   - **keypress:** Triggered when a character key is pressed down and released.
3. **Form Events:**
   - **submit:** Triggered when a form is submitted.
   - **reset:** Triggered when a form is reset.
   - **change:** Triggered when the value of a form element (input, select, etc.) changes.
   - **input:** Triggered when the value of an input element changes (more responsive than change).
   - **focus:** Triggered when an element receives focus.
   - **blur:** Triggered when an element loses focus.
   - **select:** Triggered when text is selected within an input or textarea element.
4. **Window and Document Events:**
   - **load:** Triggered when the document or a resource (e.g., image) finishes loading.
   - **unload:** Triggered when the user navigates away from the page.
   - **resize:** Triggered when the browser window is resized.
   - **scroll:** Triggered when the user scrolls within an element.
   - **DOMContentLoaded:** Triggered when the HTML document is fully loaded and parsed.
   - **beforeunload:** Triggered before the user leaves the page (for confirmation).
5. **Drag-and-Drop Events:**
   - **dragstart:** Triggered when an element is dragged.
   - **dragend:** Triggered when the element's drag operation is completed.
   - **dragover:** Triggered when an element is being dragged over a valid drop target.
   - **drop:** Triggered when an element is dropped onto a valid drop target.
6. **Media Events:**
   - **play:** Triggered when media (audio/video) starts playing.
   - **pause:** Triggered when media is paused.
   - **ended:** Triggered when media playback reaches the end.
7. **Network Events:**
   - **online:** Triggered when the browser detects an internet connection.
   - **offline:** Triggered when the browser loses its internet connection.
8. **Custom Events:**
   - Developers can create and dispatch custom events using the CustomEvent constructor.
   - **Touch Events** (for mobile and touch-enabled devices):
   - **touchstart:** Triggered when a touch point is placed on the screen.
   - **touchmove:** Triggered when a touch point moves along the screen.
   - **touchend:** Triggered when a touch point is removed from the screen.

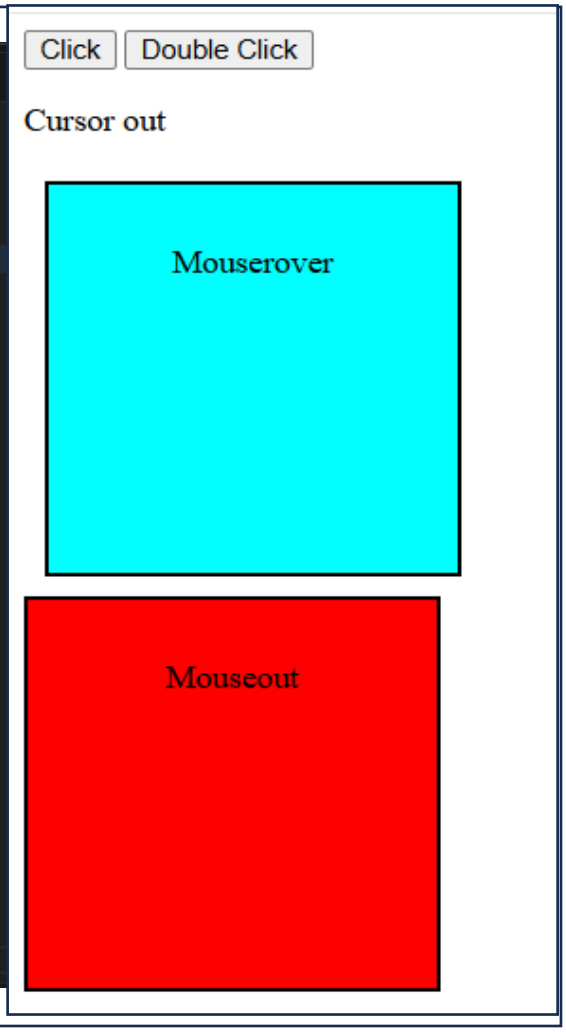
❖ DOM Mouse Events:

- **onclick / click**: Triggered when an element is clicked.
- **mouseover**: Triggered when the mouse pointer moves over an element.
- **mousemove**: Triggered when the mouse pointer moves within an element.
- **mouseout**: Triggered when the mouse pointer leaves an element.
- **dblclick**: Triggered when an element is double-clicked.

➢ **raj.html**

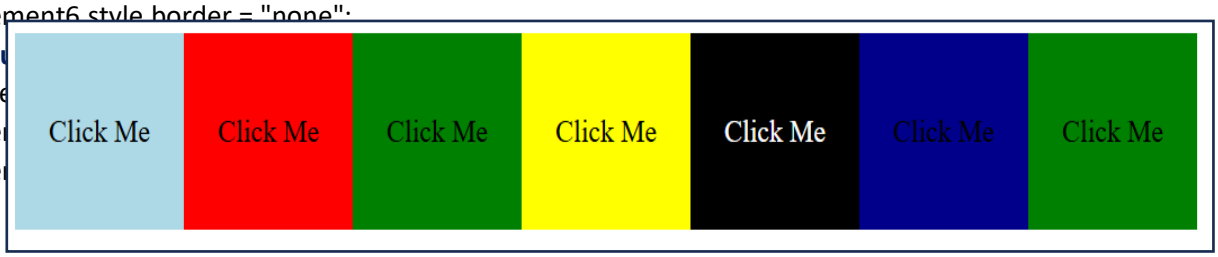
demo2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Mouse Event</title>
    <style>...
    </style>
</head>
<body>
    <button onclick="add()">Click</button>
    <button ondblclick="dadd()">Double Click</button>
    <p></p>
    <div class="c c1">Mouseover</div>
    <div class="c c2">Mouseout</div>
    <script>
        function add() {alert("Click button is successful!");}
        function dadd() {alert("Double click successful!");}
        let div1 = document.querySelector(".c1");
        let div2 = document.querySelector(".c2");
        const message = document.querySelector("p");
        div1.addEventListener("mousemove", () => {
            message.innerHTML = "You are visiting this div or my photo";
            console.log("user seeing my photo ")
        });
        div1.addEventListener("mouseout", () => {
            message.innerHTML = "Cursor out";
        });
        div2.addEventListener("mouseout", () => {
            message.innerHTML = "Cursor out";
        });
    </script></body></html>
```



```
<!DOCTYPE html>
<html>
<head>
  <title>Mouse Events Example</title>
  <style>
    .p{
      display: inline-flex;
      width: auto;
      height: auto;
    }
    #myElement1, #myElement2,#myElement3,#myElement3,#myElement4,#myElement5, #myElement6,
#myElement7 {
      width: 100px;
      height: 100px;
      background-color: lightblue;
      text-align: center;
      line-height: 100px;
      cursor: pointer;
    }
  </style>
</head>
```

```
<body>
  <div class="p">
  <div id="myElement1">Click Me</div>
  <div id="myElement2">Click Me</div>
  <div id="myElement3">Click Me</div>
  <div id="myElement4">Click Me</div>
  <div id="myElement5">Click Me</div>
  <div id="myElement6">Click Me</div>
  <div id="myElement7">Click Me</div>
</div>
  <script>
    let myElement1 = document.getElementById("myElement1");
    let myElement2 = document.getElementById("myElement2");
    let myElement3 = document.getElementById("myElement3");
    let myElement4 = document.getElementById("myElement4");
    let myElement5 = document.getElementById("myElement5");
    let myElement6 = document.getElementById("myElement6");
    let myElement7 = document.getElementById("myElement6");
    // Click event
    myElement1.onclick = function () {
      myElement1.style.backgroundColor = "lightgreen";
      myElement1.textContent = "Clicked!";
    }; // Mousedown event
    myElement2.onmousedown = function () {
      myElement2.style.backgroundColor = "lightcoral";
      myElement2.textContent = "Mouse Down";
    }; // Mouseup event
    myElement3.onmouseup = function () {
      myElement3.style.backgroundColor = "lightblue";
      myElement3.textContent = "Mouse Up";
    }; // Mousemove event
    myElement4.onmousemove = function () {
      myElement4.style.backgroundColor = "lightpink";
    }; // Mouseover event
    myElement5.onmouseover = function () {
      myElement5.style.border = "2px solid red";
    }; // Mouseout event
    myElement6.onmouseout = function () {
      myElement6.style.border = "none";
    }; // Dou
    myEleme
      myEle
      myEle
    };
  </script>
</body>
</html>
```



Example:

❖ keyboard events:

- **keydown**: triggers immediately when you press the key.
- **Keypress**: (if supported) triggers after keydown (only for printable characters).
- **keyup** triggers when you release the key.

Key Differences at a Glance

Feature	keydown	keyup	keypress (Deprecated)
When Triggered	When the key is pressed down	When the key is released	After pressing a key (printable characters only)
Supports All Keys	Yes	Yes	No (only printable characters)
Recommended?	Yes	Yes	No (use `keydown` or `keyup`)

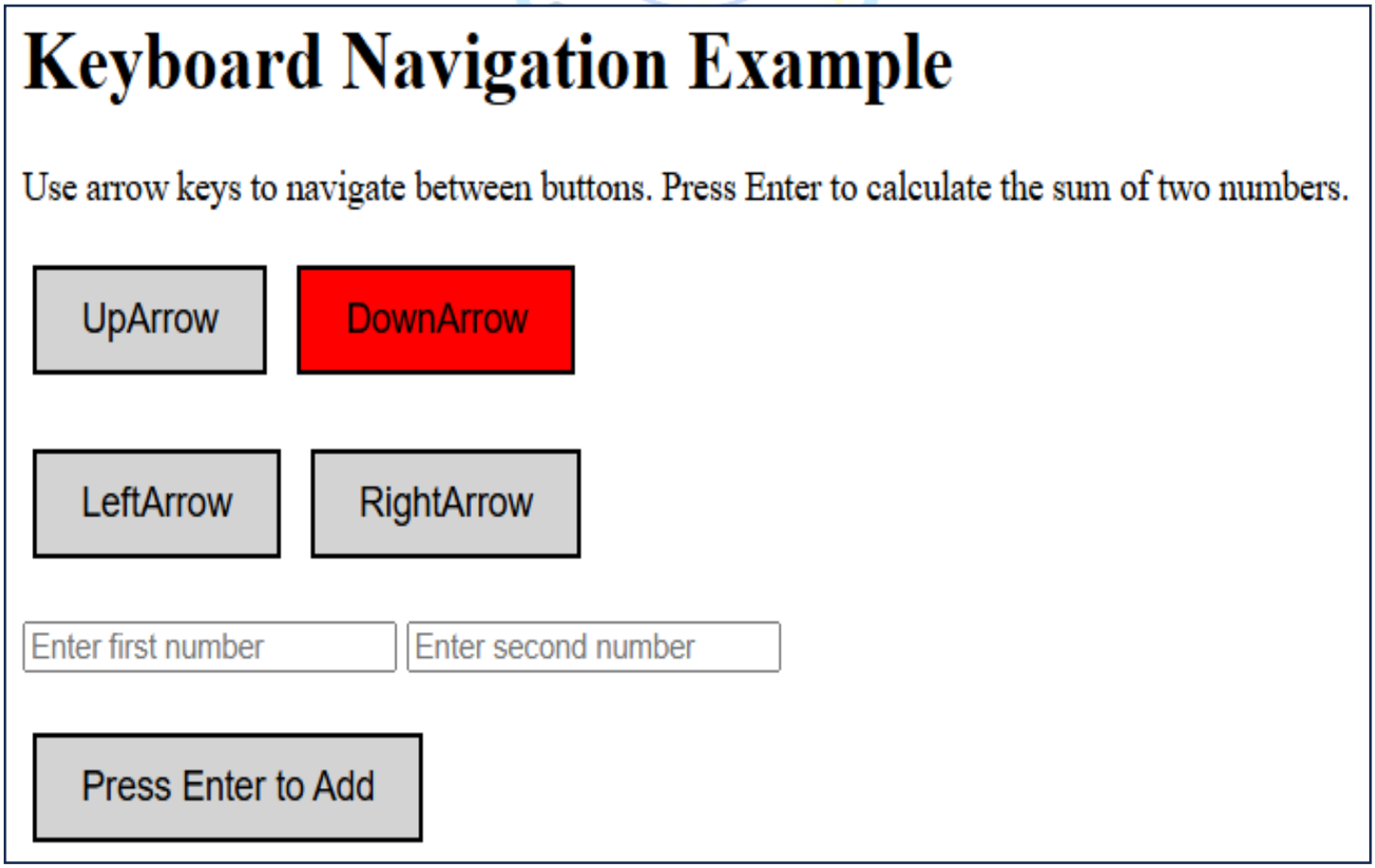
➢ Raj.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Keyboard Navigation and Addition</title>
  <style>
    button {
      background-color: lightgray;
      border: 2px solid black;
      padding: 10px 20px;
      font-size: 16px;
      margin: 5px;
      transition: background-color 0.3s;
    }
    button:focus {
      background-color: green;
      outline: none;
    }
  </style>
</head>
<body>
  <h1>Keyboard Navigation Example</h1>
<p>Use arrow keys to navigate. Press Enter to calculate the sum of two numbers.</p>
  <button type="button" id="d1">UpArrow</button>
  <button type="button" id="d2">DownArrow</button><br><br>
  <button type="button" id="d3">LeftArrow</button>
  <button type="button" id="d4">RightArrow</button><br><br>
```

```
<!-- Input fields for number addition -->
  <input type="number" id="num1" placeholder="Enter first number">
  <input type="number" id="num2" placeholder="Enter second number"><br><br>
  <button type="button" id="calculateBtn">Press Enter to Add</button>
  <p id="result"></p>

  <script>
    // Function to handle focus changes
    function updateFocus(target) {
      document.querySelectorAll("button").forEach((btn) => {
        btn.style.backgroundColor = "lightgray";
      });
      target.style.backgroundColor = "red";
    }
    // Keyboard navigation for buttons
    document.addEventListener("keydown", (event) => {
      // Identify the currently focused element
      const focusedElement = document.activeElement;  // built-in property
       // Switch cases to determine the navigation logic
      switch (event.key) {
        case "ArrowRight":
          if (focusedElement.id === "d1") {
              // Move focus to d2
            document.getElementById("d2").focus();
            updateFocus(document.getElementById("d2")); // this used for color change
          } else if (focusedElement.id === "d3") {
            document.getElementById("d4").focus();
            updateFocus(document.getElementById("d4"));    // this used for color change
          }
          break;
        case "ArrowLeft":
          if (focusedElement.id === "d2") {
            document.getElementById("d1").focus();
            updateFocus(document.getElementById("d1"));
          } else if (focusedElement.id === "d4") {
            document.getElementById("d3").focus();
            updateFocus(document.getElementById("d3"));
          }
          break;
        case "ArrowDown":
          if (focusedElement.id === "d1") {
            document.getElementById("d3").focus();
            updateFocus(document.getElementById("d3"));
          } else if (focusedElement.id === "d2") {
            document.getElementById("d4").focus();
            updateFocus(document.getElementById("d4"));
          }
```

```
            break;
        case "ArrowUp":
            if (focusedElement.id === "d3") {
                document.getElementById("d1").focus();
                updateFocus(document.getElementById("d1"));
            } else if (focusedElement.id === "d4") {
                document.getElementById("d2").focus();
                updateFocus(document.getElementById("d2"));
            }
            break;
        case "Enter":
            // Perform addition when Enter key is pressed
            const num1 = parseFloat(document.getElementById("num1").value);
            const num2 = parseFloat(document.getElementById("num2").value);
            if (!isNaN(num1) && !isNaN(num2)) {
                const result = num1 + num2;
                document.getElementById("result").innerText = `Result: ${result}`;
            } else {
                document.getElementById("result").innerText = "Please enter valid numbers!";
            }
            break;
    }
});
</script>
</body>
</html>
```



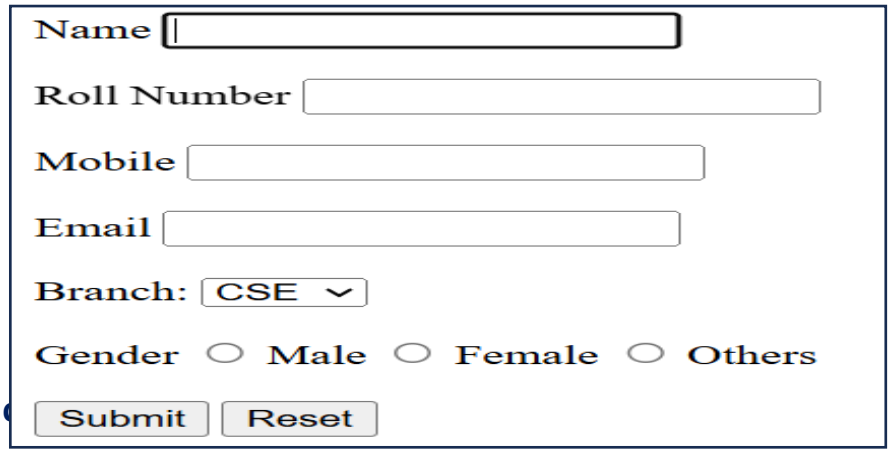
Name	Description
`submit`	Triggered when a form is submitted. Use it to handle or validate form data before submission.
`reset`	Triggered when a form is reset. Often used to perform actions when clearing form inputs.
`change`	Triggered when the value of a form element (input, select, checkbox, etc.) changes and loses focus.
`input`	Triggered in real-time when the value of an input or textarea changes.
`focus`	Triggered when a form element gains focus.
`blur`	Triggered when a form element loses focus.
`invalid`	Triggered when a form element does not meet its validation constraints (e.g., `required`).
`select`	Triggered when text is selected in an input or textarea element.

Example-1 submit and reset Event.

- **Raj.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  <form id="fsubmit">
    <label>Name </label>
    <input type="text" id="d1" required>  <br><br>
    <label>Roll Number </label>
    <input type="text" id="d2" required> <br><br>
    <label>Mobile </label>
    <input type="tel" id="d3" maxlength="10" required> <br><br>
    <label>Email  </label>
    <input type="email" id="d4" required> <br><br>
    <label>Branch: </label>
    <select id="bn">
      <option>CSE</option>
      <option>AIML</option>
      <option>CSIT</option>
    </select><br><br>
    <label>Gender</label>
    <input id="gn1" type="radio" name="gender" value="Male" required> Male
    <input id="gn2" type="radio" name="gender" value="Female"> Female
    <input id="gn3" type="radio" name="gender" value="Others"> Others
    <br><br>
    <button type="submit">Submit</button>
    <button type="reset">Reset</button>
  </form>
  <script>
    document.getElementById("fsubmit").addEventListener("submit", function(event) {
```

```
        event.preventDefault(); // Prevent form submission and page reload
        // Collect form data
        const name = document.getElementById("d1").value;
        const rollNumber = document.getElementById("d2").value;
        const mobile = document.getElementById("d3").value;
        const email = document.getElementById("d4").value;
        const branch = document.getElementById("bn").value;
        // Get selected gender
        let gender = "";
        if (document.getElementById("gn1").checked) {
          gender = "Male";
        } else if (document.getElementById("gn2").checked) {
          gender = "Female";
        } else if (document.getElementById("gn3").checked) {
          gender = "Others";
        }
        // Store data in an object
        const studentData = {
          Name: name,
          RollNumber: rollNumber,
          Mobile: mobile,
          Email: email,
          Branch: branch,
          Gender: gender
        };
        // Display data using alert
        alert(`Student Information:\n
          Name: ${studentData.Name}\n
          Roll Number: ${studentData.RollNumber}\n
          Mobile: ${studentData.Mobile}\n
          Email: ${studentData.Email}\n
          Branch: ${studentData.Branch}\n
          Gender: ${studentData.Gender}`);
      });
document.getElementById("fsubmit").addEventListener("reset", function() {
 alert("The form has been reset."); });
  </script>
</body>
</html>
```



Example-2: Input event and c

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Change and Input Events</title>
</head>
<body>
    <h2>Change and Input Events Example</h2>
    <form id="simpleForm">
        <label for="username">Name:</label>
        <input type="text" id="username" >
        <p id="nameDisplay">Name: </p>  <br><br>
        <label for="age">Age:</label>
        <input type="number" id="age" >
        <p id="ageDisplay">Age: </p>  <br><br>
        <label for="gender">Gender:</label>
        <select id="gender">
            <option value="">Select Gender</option>
            <option value="Male">Male</option>
            <option value="Female">Female</option>
            <option value="Other">Other</option>
        </select>
        <p id="genderDisplay">Gender: </p>  <br><br>
        <button type="submit">Submit</button>
    </form>
```

```
    </form>
    <script> // Handling the input event for the Name field
        let a= document.getElementById("username")
            a.addEventListener("input", function () {
            let b= document.getElementById("nameDisplay")
            b.innerText = "Name: " + this.value;
        });// Handling the change event for the Age field
        let a1= document.getElementById("age")
        a1.addEventListener("change", function () {
            let b1=document.getElementById("ageDisplay")
            b1.innerText = "Age: " + this.value;
        });// Handling the change event for the Gender field
        let a2=document.getElementById("gender")
        a2.addEventListener("change", function () {
            let b2=document.getElementById("genderDisplay")
            b2.innerText = "Gender: " + this.value;
        });// Handling the submit event on the form
        document.getElementById("sf").addEventListener("submit", function (event) {
            event.preventDefault();
            alert("Form submitted successfully!");
        });
    </script></body></html>
```

Change and Input Events Example

Name: rid bharat

Name: rid bharat

Age: 20

Age: 20

Gender: Male

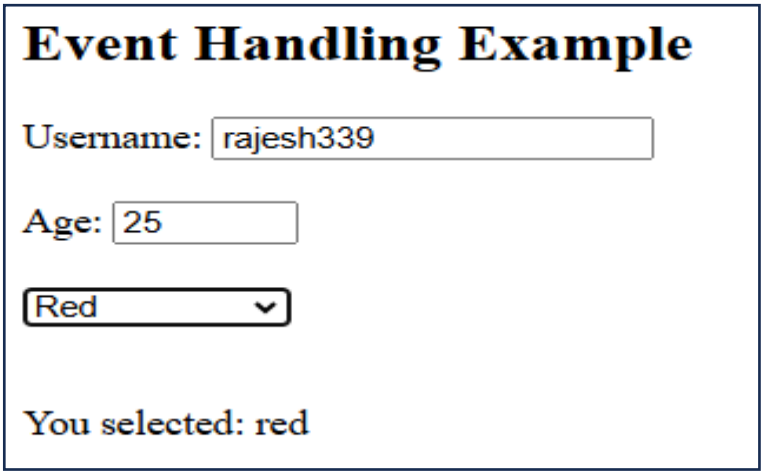
Gender: Male

Submit

- Inside the function, this refers to that **input element** (<input id="username">).
- this.value gets the **current text** typed by the user.
- "input" is the **event type**.
- It fires **every time the user types, deletes, or changes** the content of the input field.
- **event** is an action or occurrence (like a click, input, or keypress) that a web page can respond to using JavaScript.

Example-3: Input event and change event

```
<!DOCTYPE html>
<html>
<head>
    <title>Event Handling</title>
</head>
<body>
    <h2>Event Handling Example</h2>
    <form>
        <label for="username">Username:</label>
        <input type="text" id="username" required>
        <br><br>
        <label for="age">Age:</label>
        <input type="number" id="age" min="18" max="100">
        <br><br>
        <select id="color">
            <option value="">Select a color</option>
            <option value="red">Red</option>
            <option value="green">Green</option>
            <option value="blue">Blue</option>
        </select>
        <br><br>
        <p id="result"></p>
    </form>
    <script>
        // Focus event
        document.getElementById("username").addEventListener("focus", function() {
            document.getElementById("result").innerText = "Username field is focused";
        });
        // Invalid event
        document.getElementById("username").addEventListener("invalid", function() {
            document.getElementById("result").innerText = "Please enter a valid username";
        });
        // Select event
        document.getElementById("color").addEventListener("change", function() {
            var selectedColor = this.value;
            document.getElementById("result").innerText = "You selected: " + selectedColor;
        });
    </script>
</body>
</html>
```

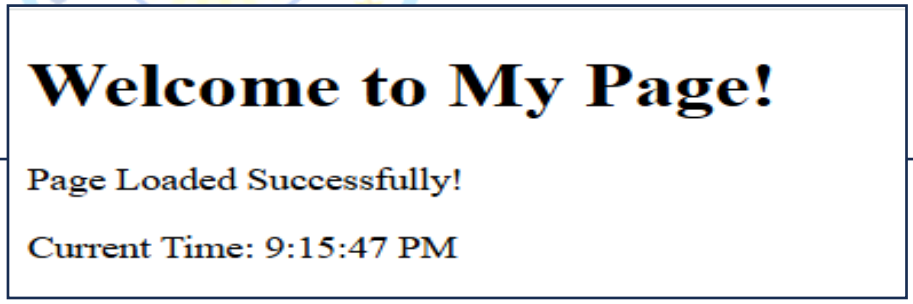


❖ Window and Document Events:

1) **load**: Triggered when a resource or webpage fully loads.
2) **unload**: Triggered when a webpage is being unloaded or closed.
3) **beforeunload**: Triggered before the webpage is unloaded, allowing prompts or warnings.
4) **scroll**: Triggered when the user scrolls within an element or the webpage.
5) **resize**: Triggered when the browser window or an element is resized.

Example: load Event:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Load Event Example</title>
</head>
<body>
 <h1>Welcome to My Page!</h1>
 <p id="status">Loading content...</p>
 <p id="time"></p>
 <script>
  window.addEventListener('load', () => {
   document.getElementById('status').textContent = "Page Loaded Successfully!";
   const now = new Date();
   document.getElementById('time').textContent = `Current Time: ${now.toLocaleTimeString()}`;
  });
 </script>
</body>
</html>
```

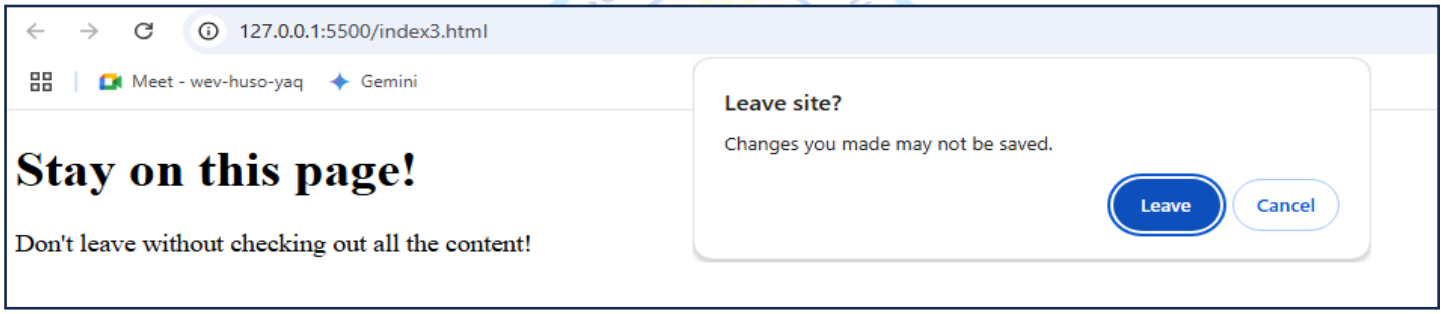


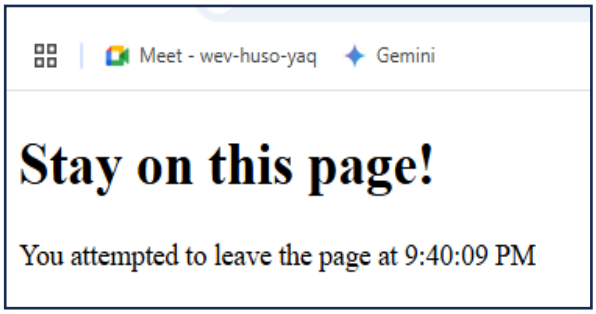
Example: unload and beforeunload events

Feature	`beforeunload`	`unload`
When It Triggers	Before the page is about to unload.	After the page has been unloaded.
Purpose	Warn the user or prevent navigation.	Perform cleanup tasks.
Allows Prevent Navigation?	Yes, using `event.returnValue`.	No. Navigation cannot be stopped.
Dialog/Message	Can trigger a browser confirmation dialog.	No dialog or message is possible.
Use Case	Warn about unsaved changes or confirm exit.	Send data or clean up connections.
Browser Restrictions	Custom messages in dialogs are ignored.	Often limited to asynchronous tasks.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Unload Event Example</title>
</head>
<body>
  <h1>Stay on this page!</h1>
  <p id="leaveMessage">Don't leave without checking out all the content!</p>
  <script>
    window.addEventListener('beforeunload', (event) => {
      event.preventDefault();
      event.returnValue = '';
      document.getElementById('leaveMessage').textContent =
        'You attempted to leave the page at ' + new Date().toLocaleTimeString();
    });
  </script>
</body>
</html>
```



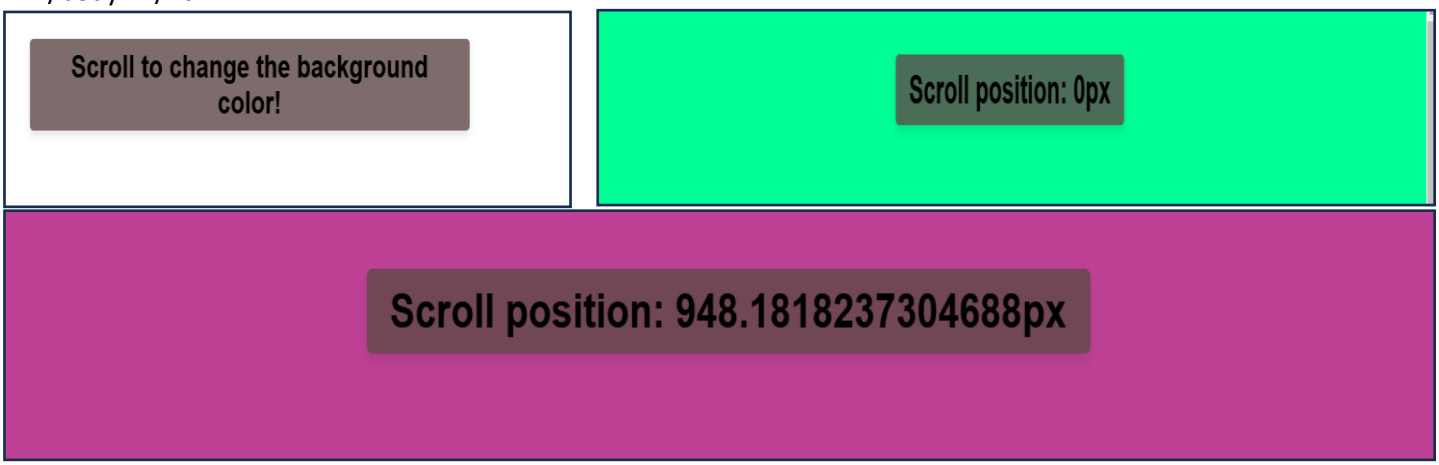


Example: Scroll and resize:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Scroll Event Example</title>
 <style>
  body {
   height: 1500px; /* To make the page scrollable */
   margin: 0;
   font-family: Arial, sans-serif;
   text-align: center;
   transition: background-color 0.5s ease;
  }
  h1 {
   position: fixed;
   top: 20px;
   left: 50%;
   transform: translateX(-50%);
   background: rgba(94, 71, 71, 0.8);
   padding: 10px 20px;
   border-radius: 5px;
   box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  }
 </style>
</head>
<body>
 <h1 id="scrollInfo">Scroll to change the background color!</h1>
 <script>
  window.addEventListener('scroll', () => {
   const scrollY = window.scrollY; // Get vertical scroll position
   const colorValue = Math.min(255, scrollY / 5); // Calculate color intensity based on scroll
   document.body.style.backgroundColor = `rgb(${colorValue}, ${255 - colorValue}, ${150})`;
   document.getElementById('scrollInfo').textContent = `Scroll position: ${scrollY}px`;
  });
 </script>
</body> </html>
```

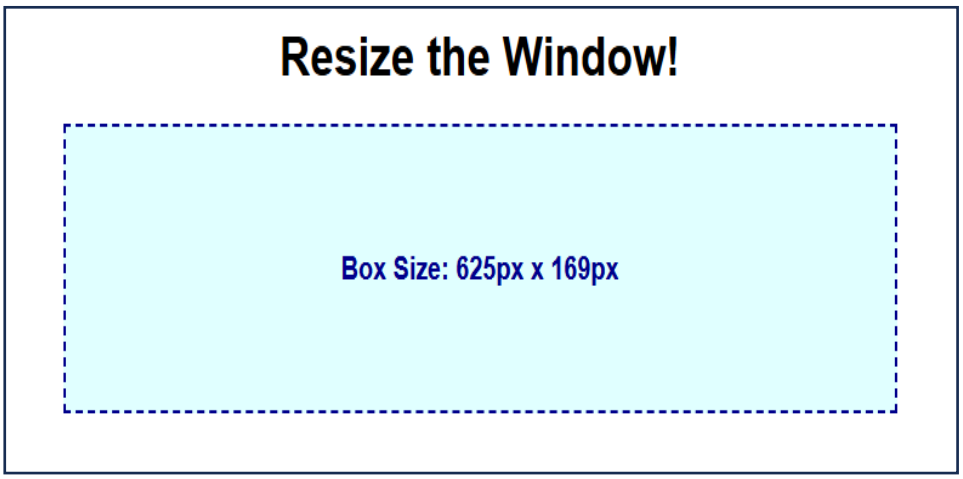
How It Works:

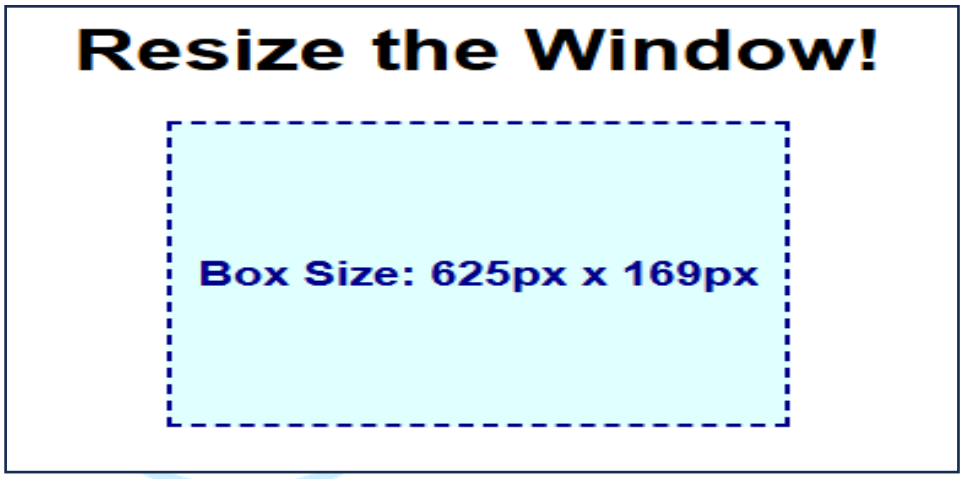
- As the user scrolls down, the background color gradually changes.
- The RGB values are dynamically adjusted based on the scroll position.
- The scroll position is displayed in the heading.



Example of Resize:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Resize Event Example</title>
 <style>
  body {
   font-family: Arial, sans-serif;
   text-align: center;
   margin: 0;
  }
  .box {
   margin: 20px auto;
   width: 50vw;
   height: 30vh;
   border: 2px dashed darkblue;
   display: flex;
   align-items: center;
   justify-content: center;
   background: lightcyan;
   transition: all 0.3s ease;
  }
  .info {
   font-size: 18px;
   font-weight: bold;
   color: darkblue;
  }
 </style>
</head>
<body>
 <h1>Resize the Window!</h1>
 <div class="box">
  <p class="info" id="boxInfo">Box Size: Loading...</p>
 </div>
 <script>
  const updateBoxSize = () => {
   const box = document.querySelector('.box');
   const boxWidth = box.offsetWidth; //Get the current width of the box
   const boxHeight = box.offsetHeight; //Get the current height of the box
   document.getElementById('boxInfo').textContent = `Box Size: ${boxWidth}px x ${boxHeight}px`;
  };
  // Initial size update
  updateBoxSize();
  // Add event listener for resize
  window.addEventListener('resize', updateBoxSize);
 </script>
</body>
</html>
```





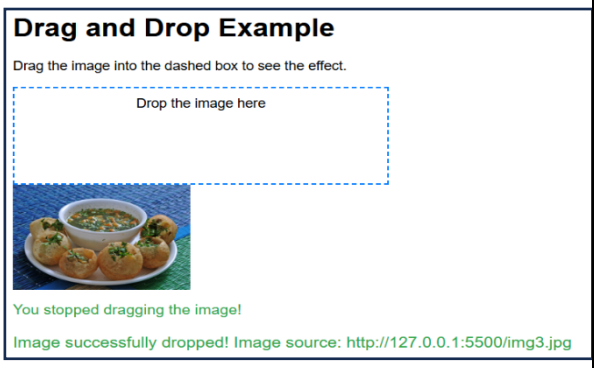
❖Drag-and-Drop Events:

1) **dragstart:** Triggered when the user starts dragging an element.
2) **dragend**: Triggered when the user stops dragging an element.
3) **dragover**: Triggered when a dragged element is over a valid drop target.
4) **drop**: Triggered when a dragged element is dropped onto a valid drop target.

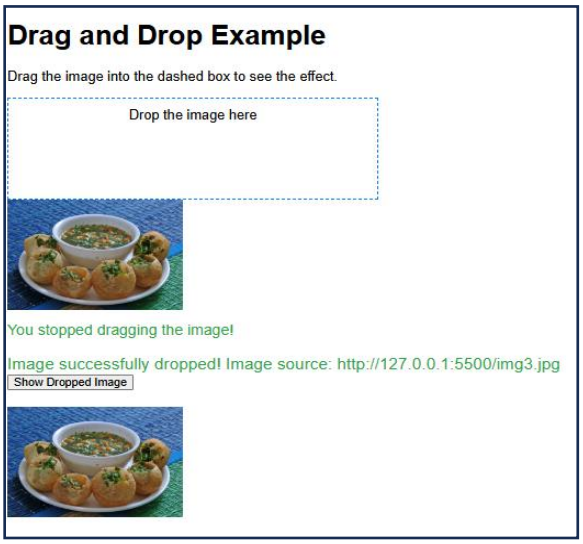
➢ **raj.html:**

```html
<!DOCTYPE html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Drag and Drop Example</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 10px; }
#da { border: 2px dashed #007bff; padding: 10px; text-align: center; m
    img { width: 200px; cursor: grab; }
    #dm { margin-top: 20px; font-size: 1.2em; }
    #sm { margin-top: 10px; color: #007bff; font-size: 1.1em; }
  </style>
</head>
<body>
  <h1>Drag and Drop Example</h1>
  <p>Drag the image into the dashed box to see the effect.</p>
  <div id="da">Drop the image here</div>
  <img id="di" src="image1.png" alt="Drag Me!" draggable="true">
  <div id="sm"></div>
  <div id="dm"></div>
  <script>
    const dragImage = document.getElementById('di');
    const dragArea = document.getElementById('da');
    const startMessage = document.getElementById('sm');
    const dropMessage = document.getElementById('dm');
    let droppedImage = null; // Variable to store the dropped image
    // Handle dragstart (on the image)
    dragImage.addEventListener('dragstart', () => {
      startMessage.textContent = 'You started dragging the image!';
      startMessage.style.color = '#007bff';
    });
    // Handle dragend (on the image)
    dragImage.addEventListener('dragend', () => {
      startMessage.textContent = 'You stopped dragging the image!';
      startMessage.style.color = '#28a745';
    });

    // Handle dragover (on the drop area)
    dragArea.addEventListener('dragover', (event) => {
      event.preventDefault();
    });
    // Handle drop (on the drop area)
    dragArea.addEventListener('drop', (event) => {
      event.preventDefault();
      droppedImage = dragImage.src; // Store the image source in the variable
      dropMessage.textContent = `Image successfully dropped! Image source: ${droppedImage}`;
      dropMessage.style.color = '#28a745';
    });
  </script>
</body>
</html>
```

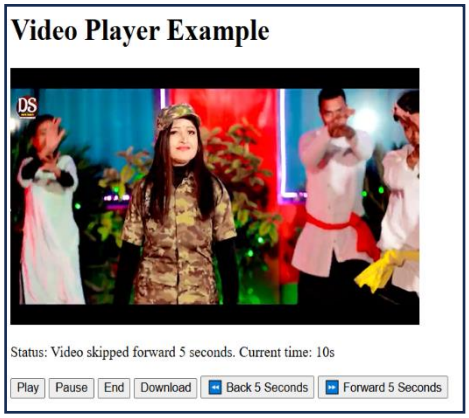


```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Drag and Drop Example</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 10px; }
    #da { border: 2px dashed #007bff; padding: 10px; text-align: center;
    img { width: 200px; cursor: grab; }
    #dm { margin-top: 20px; font-size: 1.2em; }
    #sm { margin-top: 10px; color: #007bff; font-size: 1.1em; }
    #imageContainer { margin-top: 20px; }
  </style>
</head>
<body>
  <h1>Drag and Drop Example</h1>
  <p>Drag the image into the dashed box to see the effect.</p>
  <div id="da">Drop the image here</div>
  <img id="di" src="image1.png" alt="Drag Me!" draggable="true">
  <div id="sm"></div>
  <div id="dm"></div>
  <button id="showImageBtn">Show Dropped Image</button>
  <div id="imageContainer"></div>
  <script>
    const dragImage = document.getElementById('di');
    const dragArea = document.getElementById('da');
    const startMessage = document.getElementById('sm');
    const dropMessage = document.getElementById('dm');
    const showImageBtn = document.getElementById('showImageBtn');
    const imageContainer = document.getElementById('imageContainer');
    let droppedImage = null; // Variable to store the dropped image  // Har
    dragImage.addEventListener('dragstart', () => {
      startMessage.textContent = 'You started dragging the image!';
      startMessage.style.color = '#007bff';
    });  // Handle dragend (on the image)
    dragImage.addEventListener('dragend', () => {
      startMessage.textContent = 'You stopped dragging the image!';
      startMessage.style.color = '#28a745';
    });  // Handle dragover (on the drop area)
    dragArea.addEventListener('dragover', (event) => {
      event.preventDefault();
    });  // Handle drop (on the drop area)
    dragArea.addEventListener('drop', (event) => {
      event.preventDefault();
      droppedImage = dragImage.src; // Store the image source in the variable
      dropMessage.textContent = `Image successfully dropped! Image source: ${droppedImage}`;
      dropMessage.style.color = '#28a745';
    });
  // Show dropped image when the button is clicked
  showImageBtn.addEventListener('click', () => {
      if (droppedImage) {
        const img = document.createElement('img');
        img.src = droppedImage;
        img.alt = 'Dropped Image';
        img.style.width = '200px';
        imageContainer.innerHTML = '';  // Clear previous images
        imageContainer.appendChild(img); // Display the dropped image
      } else {
        imageContainer.innerHTML = 'No image dropped yet!';
      }
    });
  </script>
</body>
</html>
```



❖ DOM Media Events:

- **play:** Triggered when the video or audio starts playing.
- **pause**: Triggered when the video or audio is paused.
- **ended**: Triggered when the video or audio playback finishes.



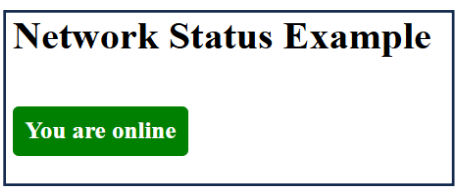
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Video Events with Skip Buttons</title>
</head>
<body>
  <h1>Video Player Example</h1>
  <!-- Video Element -->
  <video id="myVideo" width="480">
    <source id="videoSource" src="video1.mp4" type="video/mp4">
  </video>
  <!-- Status Display -->
  <p id="status">Status: Video is not playing</p>
  <!-- Buttons -->
  <button id="playBtn">Play</button>
  <button id="pauseBtn">Pause</button>
  <button id="endBtn">End</button>
  <button id="downloadBtn">Download</button>
  <button id="leftBtn">⏪ Back 5 Seconds</button>
  <button id="rightBtn">⏩ Forward 5 Seconds</button>
  <!-- Script for Event Handlers -->
  <script>
    const video = document.getElementById('myVideo');
    const videoSource = document.getElementById('videoSource'); // Get the video source
    const status = document.getElementById('status');
    const playBtn = document.getElementById('playBtn');
    const pauseBtn = document.getElementById('pauseBtn');
    const endBtn = document.getElementById('endBtn');
    const downloadBtn = document.getElementById('downloadBtn');
    const leftBtn = document.getElementById('leftBtn'); // Backward Button
    const rightBtn = document.getElementById('rightBtn'); // Forward Button
    // Play Button Event
    playBtn.addEventListener('click', () => {
      video.play();
      status.textContent = 'Status: Video is playing';
    });// Pause Button Event
    pauseBtn.addEventListener('click', () => {
      video.pause();
      status.textContent = 'Status: Video is paused';
    });// End Button Event
    endBtn.addEventListener('click', () => {
      video.currentTime = video.duration;
      status.textContent = 'Status: Video has ended';
    });// Download Button Event
    downloadBtn.addEventListener('click', () => {
      const videoURL = videoSource.src; // Get the video source URL
      const anchor = document.createElement('a'); // Create a temporary anchor element
      anchor.href = videoURL;
      anchor.download = 'video.mp4'; // Set the download filename
      anchor.click(); // Trigger the download
    });// Backward Button Event
    leftBtn.addEventListener('click', () => {
      video.currentTime = Math.max(0, video.currentTime - 5);
// Skip 5 seconds back, ensure it doesn't go below 0
      status.textContent = `Status: Video skipped back 5 seconds. Current time: ${Math.floor(video.currentTime)}s`;
    });// Forward Button Event
    rightBtn.addEventListener('click', () => {
      video.currentTime = Math.min(video.duration, video.currentTime + 5);
// Skip 5 seconds forward, ensure it doesn't exceed video duration
      status.textContent = `Status: Video skipped forward 5 seconds. Current time: ${Math.floor(video.currentTime)}s`;
    });
  </script></body></html>
```

❖ Network Events:

- **Online**: Triggered when the device regains internet connection.
- **Offline**: Triggered when the device loses internet connection.

raj.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Network Events Example</title>
 <style>
  #status {
   font-size: 20px;
   font-weight: bold;
   color: white;
   padding: 10px;
   border-radius: 5px;
   display: inline-block;
  }
  .online { background-color: green  }
  .offline { background-color: red;  }
 </style>
</head>
<body>
 <h1>Network Status Example</h1>
 <p id="status" class="online">You are online</p>
 <script>
  const statusElement = document.getElementById('status');
  // Function to update the status
  const updateStatus = () => {
   if (navigator.onLine) {
    statusElement.textContent = 'You are online';
    statusElement.className = 'online';
   } else {
    statusElement.textContent = 'You are offline';
    statusElement.className = 'offline';
   } };
  // Event listeners for network events
  window.addEventListener('online', updateStatus);
// Triggered when the user goes online
  window.addEventListener('offline', updateStatus);
// Triggered when the user goes offline
  // Initial status check
  updateStatus();
 </script>
</body> </html>
```



Regular expressions in JavaScript

- regular expression (regex)* is a sequence of characters that forms a search pattern. It is used to find, match, or replace patterns in strings. Regular expressions are powerful tools for string manipulation, including tasks like validation, extraction, and replacement of text.

1. Matching Digits

- **Match any digit**: \d
- **Match any non-digit**: \D

2. Matching Words and Characters

- **Match any word character** (letter, digit, underscore): \w
- **Match any non-word character**: \W

3. Matching Whitespace

- **Match any whitespace character** (space, tab, newline): \s
- **Match any non-whitespace character**: \S

4. Matching Specific Characters

- **Match a specific character** (e.g., a): a
- **Match a character range** (e.g., a to z): [a-z]
- **Match any character except those in range**: [^0-9] (matches non-digits)

5. Quantifiers (Repetition)

- **Match zero or more times**: *
- **Match one or more times**: +
- **Match zero or one time**: ?
- **Match exactly n times**: {n}
- **Match n or more times**: {n,}
- **Match between n and m times**: {n,m}

6. Anchors

- **Match the start of a line or string**: ^
- **Match the end of a line or string**: $

7. Grouping and Capturing

- **Create a capturing group**: (pattern)
- **Refer to captured groups**: \1, \2, etc.

8. Alternation

- **Match one of several patterns**: pattern1|pattern2

9. Word Boundaries

- **Match at the beginning of a word**: \b
- **Match at the end of a word**: \b

10. Quantifier Shortcuts

- **Match exactly one digit**: \d{1}
- **Match exactly two digits**: \d{2}

11. Email Validation

- **Basic email validation (not exhaustive)**:

^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$

12. URL Validation

- **Basic URL validation (not exhaustive)**:

^(https?|ftp)://[^\s/$.?#].[^\s]*$

❖ Types of method.

- There are following types of method in regural expresion:
    1. **test()**
    2. **match()**
    3. **matchAll()**
    4. **replace()**
    5. **replaceAll()**
    6. **exec()**
    7. **search()**
    8. **split()**

1. test() Method

- Tests whether a regular expression matches a string. Returns true or false.
- **Syntax**: regex.test(string)

1. Test for a Word

```
let regex = /hello/;
let str1 = "hello world";
let str2 = "Hi there!";
console.log(regex.test(str1)); // Output: true
console.log(regex.test(str2)); // Output: false
```

Explanation: Checks if word "hello" is present in the string. Returns true for str1 but false for str2.

2. Case-Insensitive Matching

```
let regex = /hello/i;      // `i` makes it case-insensitive
let str1 = "HELLO world";
let str2 = "Goodbye!";
console.log(regex.test(str1)); // Output: true
console.log(regex.test(str2)); // Output: false
```

Expln: i flag makes search case-insensitive. It matches "HELLO" in str1 but finds nothing in str2.

3. Test for Digits

```
let regex = /\d+/;     // Matches one or more digits
let str1 = "I have 2 apples";
let str2 = "No numbers here";
console.log(regex.test(str1)); // Output: true
console.log(regex.test(str2)); // Output: false
```

Exp: \d+ checks for one or more digits. It matches number 2 in str1 but finds no digits in str2.

4. Check for Start of String

```
let regex = /^JavaScript/;     // `^` ensures the string starts with "JavaScript"
let str1 = "JavaScript is fun";
let str2 = "I love JavaScript";
console.log(regex.test(str1)); // Output: true
console.log(regex.test(str2)); // Output: false
```

Explanation: ^ ensures that "JavaScript" is at the beginning of the string. It matches str1 but not str2.

5. Test for Whitespace

```
let regex = /\s/;       // Matches any whitespace character (space, tab, etc.)
let str1 = "Hello World";
```

```
let str2 = "NoSpacesHere";
console.log(regex.test(str1)); // Output: true
console.log(regex.test(str2)); // Output: false
```

Explanation: The \s matches any whitespace. It detects the space between "Hello" and "World" in str1 but finds no spaces in str2.

❖ Flags (Modifiers)

1) **i:** Case-insensitive matching
2) **g:** Global match (find all matches)
3) **m:** Multi-line matching

1. i - Case-Insensitive Matching i flag makes search case-insensitive, meaning it doesn't differentiate between uppercase and lowercase letters.

Example:

```
let regex = /hello/i;     // Case-insensitive matching
let str1 = "HELLO world!";
let str2 = "Hi, Hello!";
let str3 = "No match here.";
console.log(regex.test(str1)); // Output: true (matches "HELLO")
console.log(regex.test(str2)); // Output: true (matches "Hello")
console.log(regex.test(str3)); // Output: false
```

2. g - Global Match

- The g flag allows you to find all matches in a string instead of stopping after the first match. Although test() does not return all matches (only true or false), you can see the flag's behavior.

Example:

```
let regex = /hello/g;       // Global match
let str1 = "hello world! hello again!";
let str2 = "No match here.";
console.log(regex.test(str1));    // Output: true (finds "hello")
console.log(regex.test(str2));    // Output: false (no match)
```

3. m - Multi-Line Matching

- m flag allows the ^ (start of line) and $ (end of line) anchors to work across multiple lines.

Example:

```
let regex = /^hello/m;    // Match "hello" at the start of any line
let str1 = "hello world!\nHello again!";
let str2 = "Hi there!\nhello";
console.log(regex.test(str1)); // Output: true (matches "hello" in the first line)
console.log(regex.test(str2)); // Output: true (matches "hello" in the second line)
```

Combined Example: i, g, and m Flags

- You can combine flags to use multiple functionalities together.

```
let regex = /hello/img;   // Case-insensitive, global, and multi-line match
let str = "HELLO world!\nhello again!\nHi, Hello!";
console.log(regex.test(str));     // Output: true (matches "HELLO", "hello" across lines)
```

2. match() method

- match() method in JavaScript is used to extract matches for a regular expression from a string. It returns an **array** of matches or null if no match is found.

Example:

1. Simple Match

```
let str = "I love JavaScript!";
let regex = /love/;
let result = str.match(regex);
console.log(result);
```

// Output: ['love']
// ['love', index: 2, input: 'I love JavaScript!',
groups: undefined]

Explanation:

1. **'love'**:
   This is the part of the string that matched the regular expression /love/.
2. **index: 2**:
   The starting position of the match in the string. In this case, the word 'love' starts at the 2nd index (0-based indexing) in the string "I love JavaScript!".
3. **input: 'I love JavaScript!'**:
   The full input string that was searched.
4. **groups: undefined**:
   - The groups property is used to store any **named capturing groups** from the regular expression.
   - In your example, the regular expression /love/ does not have any capturing groups (e.g., (?<name>...)), so groups is undefined.
   - If you had a named capturing group, it would contain an object with the group names and their matched values.

Example with a Named Capturing Group:

```
let str = "I love JavaScript!";
let regex = /(?<emotion>love)/;
// Named capturing group
let result = str.match(regex);
console.log(result);
```

Output:

```
[
'love',
'love',
index: 2,
input: 'I love JavaScript!',
 groups: { emotion: 'love' }
]
```

2. Case-Insensitive Matching

```
let str = "I LOVE JavaScript!";
let regex = /love/i;    // `i` makes it case-insensitive
let result = str.match(regex);
console.log(result); // Output: ['LOVE']
```

Explanation: The i flag allows the regex to match "LOVE" regardless of case.

3. Global Match

```
let str = "I love JavaScript! I also love Python.";
let regex = /love/g;  // `g` finds all matches
let result = str.match(regex);
console.log(result);   // Output: ['love', 'love']
```

Explanation: The g flag returns all occurrences of "love" as an array.

4. Match Digits

```
let str = "I have 2 apples and 3 bananas.";
let regex = /\d+/g; // `\d+` matches one or more digits
let result = str.match(regex);
console.log(result);   // Output: ['2', '3']
```

Explanation: The regex matches all numbers (2 and 3) in the string.

5. Match Words at the Start

```
let str = "JavaScript is fun!";
let regex = /^JavaScript/;   // `^` ensures the match starts at the beginning
let result = str.match(regex);
console.log(result);   // Output: ['JavaScript']
```

Explanation: The regex matches "JavaScript" only if it's at the start of the string. **Return Value:**

1. If a match is found:

- Returns an array:
  - The first element is the full match.
  - The remaining elements contain captured groups (if any).

2. If no match is found: Returns null.

Example of No Match:

```
let str = "I love coding.";
let regex = /python/;
let result = str.match(regex);
console.log(result); // Output: null
```

3. matchAll() Method

- matchAll() method in JavaScript returns an iterator of all matches of a regular expression in a string, including detailed information about each match like capturing groups and positions. It requires the **global flag (g)** in the regex.

Syntax: string.matchAll(regex)

- **string**: The string to search within.
- **regex**: The regular expression to match against, which must include the g (global) flag.

Example: Find All Numbers in a String

```
let str = "I have 2 apples, 3 bananas, and 10 oranges.";
let regex = /\d+/g;   // Matches all numbers
let matches = str.matchAll(regex);
for (const match of matches) {
  console.log(match[0]);   // The matched number
}
```

Output: 2, 3,10

Example 1: Find All Vowels in a String

```
let str = "Hello, how are you?";
let regex = /[aeiou]/g; // Matches vowels (a, e, i, o, u)
let matches = str.matchAll(regex);
for (const match of matches) {
  console.log(match[0]); // The matched vowel
}
```

Output: e, o, o, a, e, o

Example 2: Find All Words in a Sentence

```
let str = "JavaScript is fun!";
let regex = /\b\w+\b/g;  // Matches complete words
let matches = str.matchAll(regex);
for (const match of matches) {
  console.log(match[0]);   // The matched word  }
```

Output: JavaScript
is
fun

Example 1:

- The regular expression /[aeiou]/g looks for all vowels in the string.
- The loop prints each vowel one by one.

Example 2:

- The regular expression /\b\w+\b/g matches entire words in the string.
- \b ensures it matches word boundaries, and \w+ matches one or more word characters.

Aspect	`match()`	`matchAll()`
Output	Returns an array of matches or `null`.	Returns an iterator with detailed match info.
Multiple Matches	Needs `g` flag to return all matches.	Always returns all matches.
Details Provided	No capturing groups or match index.	Includes match index, input, and groups.
Use Case	When only matched strings are needed.	When detailed match info is required.

4. replace() Method

- replace() method is used to replace a specific substring or pattern (defined by a regular exp..) in a string with a new string. It replaces only **first match** unless global flag (g) is used.

Syntax: string.replace(pattern, replacement)

- pattern: The substring or regular expression to match.
- replacement: The string or function to replace the matched substring.

Example 1: Replacing a Word Replace the word "apple" with "orange" in a string.

```
let str = "I like apple pie.";
let newStr = str.replace(/apple/, "orange");
console.log(newStr);   // Output: "I like orange pie."
```

Explanation: This replaces the first occurrence of "apple" with "orange".

Example 2: Replacing All Occurrences (Global Match) Use the g flag to replace all occurrences of "apple" with "orange".

Example:

```
let str = "I have an apple and another apple.";
let newStr = str.replace(/apple/g, "orange");
console.log(newStr); // Output: "I have an orange and another orange."
```

Explanation: The g flag ensures that both occurrences of "apple" are replaced.

Example 3: Case-Insensitive Replacement Use the i flag to replace "Apple" (case-insensitive) with "orange".

Example:

```
let str = "I love Apple pie!";
let newStr = str.replace(/apple/i, "orange");
console.log(newStr); // Output: "I love orange pie!"
```

Explanation: The i flag makes the search case-insensitive, so it matches "Apple" regardless of case.

Example 4: Replacing Digits with a Symbol Replace any number in the string with the # symbol.

Example:

```
let str = "My house number is 1234.";
let newStr = str.replace(/\d+/g, "#");
console.log(newStr); // Output: "My house number is #."
```

Explanation: The regular expression \d+ matches any number, and the g flag ensures that all numbers in the string are replaced.

Example 5: Removing Extra Spaces Remove multiple spaces between words and replace them with a single space.

Example:

```
let str = "This   is   a   sentence.";
let newStr = str.replace(/\s+/g, " ");
console.log(newStr); // Output: "This is a sentence."
```

Explanation: The regular expression \s+ matches one or more whitespace characters, and the g flag ensures that all extra spaces are replaced with a single space.

5. replaceAll() Method

- **replaceAll()** method replaces **all occurrences** of a substring or pattern in a string with a new string. Unlike replace(), it does not require the global flag (g) and replaces all matches by default.

Syntax: string.replaceAll(pattern, replacement)

Example: Replacing All Occurrences of a Word:

```
let str = "I like apples, apples are tasty.";
let result = str.replaceAll("apples", "oranges");
console.log(result);   // Output: "I like oranges, oranges are tasty."
```

Example: Replacing Using Regular Expression:

```
let str = "1, 2, 3, 4";
let result = str.replaceAll(/\d/g, "#");
console.log(result); // Output: "#, #, #, #"
```

Example: Replacing All Spaces:

```
let str = "Hello World! How are you?";
let result = str.replaceAll(" ", "-");
console.log(result); // Output: "Hello-World!-How-are-you?"
```

Key Differences Between `replace()` and `replaceAll()`:

Aspect	`replace()`	`replaceAll()`
Default Behavior	Replaces the first occurrence only (unless `g` is used).	Replaces all occurrences by default.
Regex Requirement	Can use a regex with the `g` flag.	No need for the `g` flag to replace all.
Availability	Available in most browsers.	Available in modern browsers only.

6. exec() Method

The exec() method executes a regular expression on a string and returns the first match as an array with detailed information (e.g., index, input). If no match is found, it returns null.

Syntax: regex.exec(string)

Example-1: Find a Word

```
let regex = /cat/;
let str = "The cat is on the mat.";
let result = regex.exec(str);
console.log(result);
```

// Output: ['cat', index: 4, input: 'The cat is on the mat.']

Example-2: Find Numbers

```
let regex = /\d+/;
let str = "There are 123 apples.";
console.log(regex.exec(str));
```

// Output: ['123', index: 10, input: 'There are 123 apples.']

Example-3: No Match

```
let regex = /cat/;
let str = "The dog is barking.";
console.log(regex.exec(str));
// Output: null
```

7. search() Method

The search() method searches for a match of a regular expression in a string and returns the index of the first match. If no match is found, it returns -1.

Syntax: string.search(regex)

Example-1: Find the First Match

```
let str = "I love JavaScript!";
let index = str.search(/JavaScript/);
console.log(index);   // Output: 7
```

Example 2: Find the First Match

```
let str = "I love rid bharat rid patna !";
console.log(str.search(/rid/));  // Output: 2
```

Example 3: Case-Insensitive Search

```
let str = "I like Cats.";
console.log(str.search(/cats/i));   // Output: 7
```

Example 3: No Match Found

```
let str = "Hello world!";
console.log(str.search(/dog/)); // Output: -1
```

8. split() Method

The split() method splits a string into an array of substrings using a specified delimiter, which can be a substring or regular expression.

Syntax: string.split(separator)

Example 1: Split by Comma

```
let str = "apple,banana,orange";
console.log(str.split(",")); // Output: [ 'apple', 'banana', 'orange' ]
```

Example 2: Split by Space

```
let str = "JavaScript is fun";
console.log(str.split(" ")); // Output: [ 'JavaScript', 'is', 'fun' ]
```

Example 3: Split by a Regular Expression

```
let str = "1. First 2. Second 3. Third";
console.log(str.split(/\d\./)); // Output: [ '', ' First ', ' Second ', ' Third' ]
```

❖ When to Use Regular Expressions?

- Validating input (e.g., email, phone numbers).
- Searching and replacing text.
- Extracting specific data (e.g., numbers, words).
- Splitting strings by patterns.

❖ Special Characters:

- **.**: Matches any character (except newline).
- **^**: Matches the beginning of a string.
- **$**: Matches the end of a string.
- **\d**: Matches any digit (0-9).
- **\w**: Matches word characters (letters, digits, underscore).
- **\s**: Matches whitespace.
- **+**: Matches one or more of the preceding characters.
- *****: Matches zero or more of the preceding characters.
- **?**: Makes the preceding character optional.

JAVASCRIPT VALIDATION

- Form Validation
- Email Validation

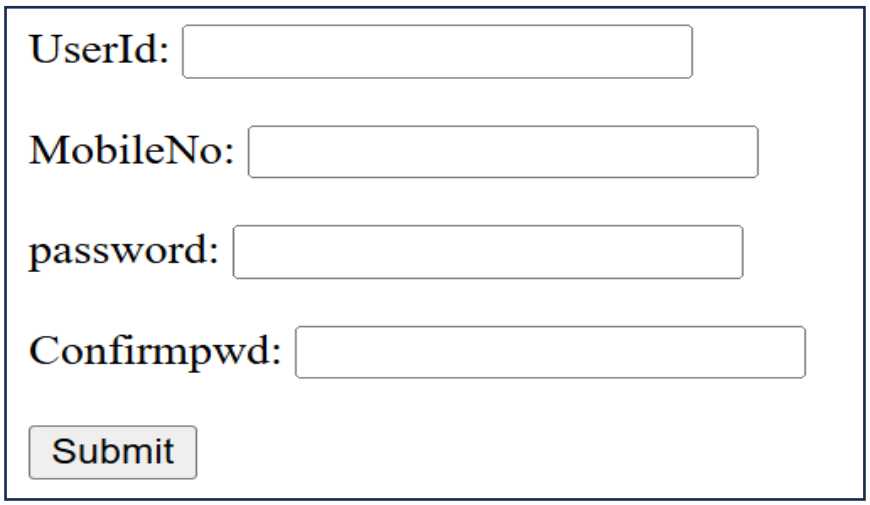
❖ Example of Form Validation:

➢ index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Form validation</title>
</head>
<body>
  <form onsubmit="return fsubmit();" action="home.html">
    UserId: <input id="d1" type="text"><br><br>
    MobileNo: <input id="d2" type="tel"><br><br>
    password: <input id="d3" type="password"><br><br>
    Confirmpwd: <input id="d4" type="password"><br><br>
    <input type="submit" value="Submit">
  </form>
  <script>
    function fsubmit() {
      const a = document.getElementById("d1").value;
      const b = document.getElementById("d2").value;
      const c = document.getElementById("d3").value;
      const d = document.getElementById("d4").value;
      if (a === "" || b === "" || c === "" || d === "") {
        alert("All fields are mandatory");
        return false;
      }
      else if (b.length<10 || b.length>10) {
        alert("Please Enter 10 digits");
        return false;
      }
      else if (isNaN(b)) {
        alert("Please Enter only digits");
        return false;
      }
      else if (c!==d) {
        alert("Password is not same !");
        return false;
      }
      else { return true;
      } }
  </script>
</body>
</html>
```

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>data submit successfully </h1>
</body>
</html>
```



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Form Validation Example</title>
</head>
<body>
<h1>Registration Form</h1>
<form onsubmit="return fsubmit()" action="home.html">
    Name:<input type="text" id="name">
    <span id="nameError"></span><br><br>
    Email:<input type="text" id="email">
    <span id="emailError"></span><br><br>
    Password:<input type="password" id="password">
    <button type="button" onclick="togglePassword()">Show</button>
    <span id="passwordError"></span><br><br>
    Confirm Password:<input type="password" id="cp">
    <span id="cpError"></span><br><br>
    <button type="submit">Register</button>
</form>
<script>
function fsubmit() {
    const name = document.getElementById("name").value.trim();
    const email = document.getElementById("email").value.trim();
    const password = document.getElementById("password").value;
    const cp = document.getElementById("cp").value;

    document.getElementById("nameError").innerText = "";
    document.getElementById("emailError").innerText = "";
    document.getElementById("passwordError").innerText = "";
    document.getElementById("cpError").innerText = "";

    // Name validation (only letters and spaces)
    const namePattern = /^[A-Za-z\s]+$/;
    if (name === "") {
document.getElementById("nameError").innerText = "Enter name";
        return false;
    }
    if (!namePattern.test(name)) {
        document.getElementById("nameError").innerText =
            "Name can only contain letters and spaces";
        return false;
    }
    // Simple email validation
    if (email === "" || !email.includes("@") || !email.includes(".")) {
        document.getElementById("emailError").innerText = "Enter valid email";
        return false;
    }
    // Simple password validation
    let hasNumber = false;
    let hasSpecial = false;
    const specialChars = "!@#$%^&*";
    for (let i = 0; i < password.length; i++) {
        if (!isNaN(password[i])) hasNumber = true;
        if (specialChars.includes(password[i])) hasSpecial = true;
    }
    if (password.length < 6 || !hasNumber || !hasSpecial) {
        document.getElementById("passwordError").innerText =
            "Password must be 6+ chars, 1 number & 1 special symbol";
        return false;
    }
    // Confirm password validation
    if (password !== cp) {
        document.getElementById("cpError").innerText = "Passwords do not match";
        return false;
    }
    return true;
}
// Show / Hide password
function togglePassword() {
    const pwd = document.getElementById("password");
    pwd.type = pwd.type === "password" ? "text" : "password";
}
</script>
</body>
</html>
```

regular expression ^[A-Za-z\s]+$

- **^:** Matches the beginning of the string.
- **[A-Za-z]:** Matches any uppercase letter (A to Z) or lowercase letter (a to z).
- **\s:** Matches any whitespace character (spaces, tabs, newlines, etc.).
- **+:** Matches one or more occurrences of the preceding character or group. In this case, it means "one or more letters or whitespace characters".
- **$:** Matches the end of the string. **So, put together**:
- **^[A-Za-z\s]+$:** Matches a string that starts at the beginning (^), contains one or more (+) letters (A-Z, a-z) or whitespace characters (\s), and ends at the end of the string ($).

❖ **Email validation regular expression ^[^\s@]+@[^\s@]+\.[^\s@]+$**

- **^:** Matches the beginning of the string.
- **[^\s@]:** Matches any character that is *not* a whitespace character (\s) or an @ symbol. The ^ inside the square brackets [] means "not".
- **+:** Matches one or more occurrences of the preceding character or group.
- **@:** Matches the literal @ symbol.
- **\.:** Matches the literal . symbol. The backslash \ is used to escape the dot, as the dot has a special meaning in regular expressions (it matches any character except a newline).
- **$:** Matches the end of the string.

Web APIs

What is a Web API?

- **API** stands for **Application Programming Interface**.
- A **Web API** allows web applications to interact with browsers or servers.
- **Browser APIs** add extra features to browsers.
- **Server APIs** help servers provide services to web apps.

Browser APIs

- Browsers provide built-in Web APIs to perform complex tasks easily.
- They help access device features like location, storage, camera, etc.

Example: Geolocation API

- The **Geolocation API** gives the current latitude and longitude of the user.

Note: - When the button is clicked, the browser fetches the user's location and displays it.

Third-Party APIs

- These APIs are **not built into the browser**.
- They are provided by external companies and must be included from the web.

Examples:

- **YouTube API** – Show videos on a website **Twitter API** – Display tweets
- **Facebook API** – Access Facebook data

JavaScript Validation API

- Also called the **Constraint Validation API**.
- Used to **validate form data in the browser** before submitting to the server.
- It improves user experience and reduces server load.

Key Features of Validation API

1. HTML5 Validation Attributes

- Used directly in input fields.
- Common attributes: required, min, max, pattern, minlength, maxlength.

Example: <input type="text" required minlength="3" maxlength="20">

2. Validation Methods

- checkValidity() – checks if input is valid
- setCustomValidity() – sets custom error message
- reportValidity() – shows validation message

3. Validity Properties

- validity.valid – input is valid or not
- validity.valueMissing – required field empty
- validity.tooShort / tooLong – length error

4. Validation Messages

- Browser shows default messages.
- Custom messages can also be added.

5. Styling Invalid Fields

- Invalid inputs can be styled using CSS.

```
input:invalid {
  border-color: red;
}
```

6. Prevent Form Submission

- If form is invalid, submission is automatically blocked.

7. Event Handling

- Validation can be checked during submit, input, or change events.

JavaScript AJAX

What is AJAX?

- **AJAX** stands for **Asynchronous JavaScript and XML**.
- It allows data exchange with the server **without reloading the page**.
- Mostly uses **JSON** instead of XML.

Key Concepts of AJAX

1. XMLHttpRequest Object

- Used to send and receive data from the server.

2. Asynchronous Requests

- Page continues working while data is loading.
- Improves performance and user experience.

3. Data Formats

- XML (old)
- **JSON (most common)**
- Text or HTML

4. AJAX Workflow

1. Create request object
2. Set callback function
3. Open server connection
4. Send request
5. Receive response

5. Same-Origin Policy

- Requests allowed only from the same domain.
- **CORS** is used for cross-domain access.

6. AJAX Libraries

- jQuery AJAX
- Axios
- Fetch API

7. Uses of AJAX

- Dynamic page updates
- Form submission without reload
- Live search
- Chat applications
- API data fetching

8. Benefits

- Faster websites
- Better user experience
- Less server load

JavaScript JSON

What is JSON?

- **JSON (JavaScript Object Notation)** is a lightweight data format.
- Used to exchange data between server and client.
- Easy to read and write.

JSON Syntax

- Data is stored as **key–value pairs**
- Keys are in double quotes
- Uses {} for objects and [] for arrays

Example:

```
{
 "name": "Sangam",
 "age": 30,
 "city": "Patna"
}
```

JSON in JavaScript

Convert Object to JSON

- JSON.stringify()

Convert JSON to Object

- JSON.parse()

Why JSON is Popular?

- Lightweight
- Fast
- Easy to use
- Works well with AJAX and APIs

JavaScript (JS) vs jQuery

JavaScript (JS)

- JavaScript is the **core programming language of the web**.
- It works in **all modern browsers**.
- Used to **create interactive web pages**.
- Provides **full control** over HTML, CSS, and browser behavior.
- Used with frameworks like **React, Angular, Vue**.
- Slightly **harder for beginners**, but very powerful.
- Gives **better performance** when optimized.

➔ Best for large, modern, and complex applications.

jQuery

- jQuery is a **JavaScript library**.
- It **simplifies JavaScript code**, especially DOM manipulation.
- Requires **less code** to do common tasks.
- Handles **cross-browser issues** automatically.
- Easy to learn and **beginner-friendly**.
- Good for **small projects and quick development**.

➔ Best for simple websites and fast prototyping.

JavaScript Graphics

JavaScript allows us to **draw graphics and animations** on web pages using **Canvas** and **SVG**.

HTML5 Canvas (Raster Graphics)

- Uses the <canvas> element.
- Works with **pixels** (raster graphics).
- Good for **games, animations, and drawings**.
- Graphics are drawn using JavaScript.

Key Features:

- Draw shapes, text, and images
- Supports animations
- Handles mouse and keyboard events

Canvas Example (Simple)

- Draws a rectangle and text using JavaScript.

SVG (Vector Graphics)

- SVG stands for **Scalable Vector Graphics**.
- Uses **XML-based tags**.
- Graphics do **not lose quality when resized**.
- Best for **icons, charts, and diagrams**.

Key Features:

- Scalable without blur
- Easy to style with CSS
- Supports animation and interaction
- Accessible and screen-reader friendly

SVG Example: - Draws a rectangle and text directly in HTML.

Exception Handling

- **Exception handling** is used to **handle errors** without stopping the program.
- It helps prevent script crashes and shows **meaningful error messages**.

Types of Errors in JavaScript

(a) Syntax Error

- Happens when code rules are broken. Code will **not run**.

Example: if (x == 10 { // Missing)

(b) Runtime Error (Exception) Occurs while the program is running.

Example: let x = y + 5; // y is not defined

(c) Logical Error

- Code runs but gives **wrong output**. No error message.

Example: let sum = 10 - 5; // Logic mistake

try...catch Statement

- Used to catch and handle runtime errors.

Syntax:
```
try {
  // risky code
} catch (error) {
  // error handling code
}
```

Example:
```
try {
  let x = y + 10;
} catch (error) {
  console.log("Error occurred");
}
```

Error Object:- The error object stores error details. Common property: error.message

Example:
```
try {
  let a = b + 5;
} catch (error) {
  console.log(error.message);
}
```

finally Block

- Always runs whether error occurs or not.
- Used for cleanup tasks.

Syntax:
```
try {
  // code
} catch (error) {
  // handle error
} finally {
  // always execute
}
```

Example:
```
try {
  let x = 10 / 0;
} catch (error) {
  console.log("Error");
} finally {
  console.log("Program ended");
```

```
}
```

Throwing Custom Errors

- Use throw to create your own error.

Syntax: throw new Error("Message");

Example:

```
function divide(a, b) {
  if (b === 0) {
    throw new Error("Division by zero not allowed");
  }
  return a / b;
}
try {
  divide(10, 0);
} catch (error) {
  console.log(error.message);
}
```

Built-in Error Types

- JavaScript provides built-in errors:
  - Error, TypeError, ReferenceError, SyntaxError

Example:

```
try {
  let num = 5;
  num.toUpperCase();
} catch (error) {
  if (error instanceof TypeError) {
    console.log("Type Error occurred");
  }}
```

Asynchronous Exception Handling

- Errors inside async code must be handled **inside** it.

Example with setTimeout:

```
setTimeout(() => {
  try {
    let x = y + 10;
  } catch (error) {
    console.log("Async Error");
  }
}, 1000);
```

Global Error Handling→ Catches unhandled errors globally.

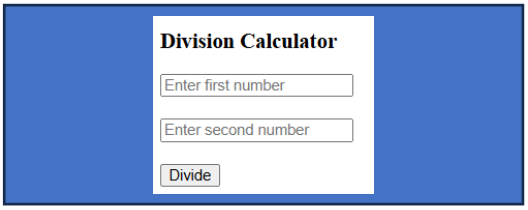
Syntax:

```
window.onerror = function(message, source, line, column, error) {
  // handle error
};
```

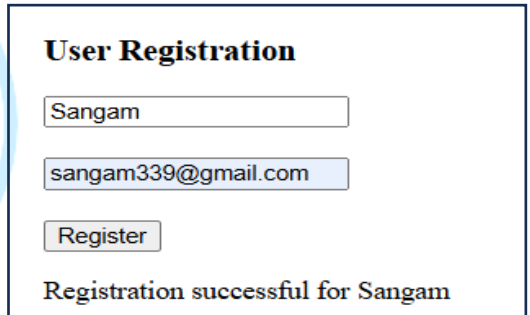
Example:

```
window.onerror = function(message, source, line, column, error) {
  console.log("Global Error:", message);
  return true;
};
```

```html
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8"> <title>JS Exception Handling Project</title>
</head>
<body>
<div class="box">
    <h3>Division Calculator</h3>
    <input type="text" id="num1" placeholder="Enter first number"><br><br>
    <input type="text" id="num2" placeholder="Enter second number"><br><br>
    <button onclick="divide()">Divide</button>
    <p id="result"></p>
</div>
<script>
function divide() {
    try {
        let a = document.getElementById("num1").value;
        let b = document.getElementById("num2").value;
        // Check empty input
        if (a === "" || b === "") { throw new Error("Please enter both numbers");
        }
        a = Number(a);
        b = Number(b);
        // Check invalid number
        if (isNaN(a) || isNaN(b)) { throw new Error("Only numbers are allowed");}
        // Division by zero check
        if (b === 0) {throw new Error("Division by zero is not allowed");}
        let result = a / b;
        document.getElementById("result").innerHTML =
            "Result: " + result;
    } catch (error) {
        document.getElementById("result").innerHTML =
            "Error: " + error.message;
    } finally {console.log("Calculation attempt completed");}
}
</script></body></html>
```




```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>JS Exception Handling - Non-Math Example</title>
</head>
<body>
<div class="box">
    <h3>User Registration</h3>
    <input type="text" id="username" placeholder="Enter username"><br><br>
    <input type="text" id="email" placeholder="Enter email"><br><br>
    <button onclick="register()">Register</button>
    <p id="result"></p>
</div>
<script>
function getUserInput() {
    let name = document.getElementById("username").value;
    let email = document.getElementById("email").value;
    if (name === "" || email === "") {
        throw new Error("All fields are required");
    }
    if (name.length < 3) {
        throw new Error("Username must be at least 3 characters");
    }
    if (!email.includes("@")) {
        throw new Error("Invalid email format");
    }
    return { name, email };
}
function register() {
    try {
        const user = getUserInput();
        document.getElementById("result").innerHTML =
            "Registration successful for " + user.name;
    } catch (error) {
        document.getElementById("result").innerHTML =
            "Error: " + error.message;
    } finally { console.log("Registration process completed");}
}
</script></body></html>
```



Sample Inputs and Outputs

Case 1: Valid Input

Input: Username: Sangam
Email: rahul@gmail.com

Output: Registration successful for Rahul

Case 2: Empty Fields

Input: Username: Email:

Output: Error: All fields are required

Case 3: Short Username

Input: Username: Ab
Email: ab@gmail.com

Output: Error: Username must be at least 3 characters

Case 4: Invalid Email

Input: Username: Aman
Email: amangmail.com

Output: Error: Invalid email format

Local Storage in JavaScript

What is Local Storage?

- **Local Storage** is a browser feature that stores data as **key–value pairs**.
- Data is saved **permanently** (not deleted on page refresh).
- Data is stored in the **user's browser**.

1. Storing Data

- Use setItem(key, value) Both key and value must be **strings**.

Example:
```
localStorage.setItem("username", "raj");
localStorage.setItem("email", "ra393@.com");
```

2. Retrieving Data→ Use getItem(key)

Example:
```
let a = localStorage.getItem("username");
let b = localStorage.getItem("email");
console.log(a);
console.log(b);
```

3. Updating Data→ Store again using the same key.

Example:
```
localStorage.setItem("username", "Sangam");
localStorage.setItem("username", "Sangam123"); // updated value
let a = localStorage.getItem("username");
console.log(a);
```

4. Checking if Data Exists

- getItem() returns **null** if data does not exist.
- Use comparison to get **true / false**.

Example:
```
let a = localStorage.getItem("email") !== null;
let b = localStorage.getItem("username") !== null;
console.log(a); // true or false
console.log(b); // true or false
```

5. Removing Specific Data→ Use removeItem(key)

Example:
```
localStorage.removeItem("email");
let a = localStorage.getItem("username");
let b = localStorage.getItem("email");
console.log(a);
console.log(b); // null
```

6. Clearing All Data→ Removes **all stored data**.

Example:→
```
localStorage.clear();
```

Storage Limit

- Storage size is about **5–10 MB** (browser dependent).
- Exceeding the limit may cause errors.

7. Storing Objects and Arrays

- Local Storage stores **only strings**. Use JSON.stringify() to store data.
- Use JSON.parse() to retrieve data.

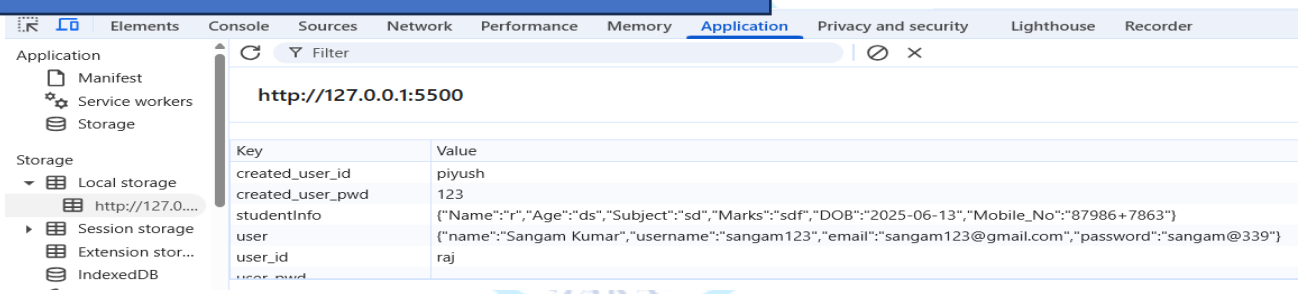
Example:
```
const user = {
    name: "sangam",
    email: "sangam@example.com"
};
localStorage.setItem("user", JSON.stringify(user));
const retrievedUser = JSON.parse(localStorage.getItem("user"));
console.log(retrievedUser);
```

```html
<!DOCTYPE html>
<html lang="en">
<head> <meta charset="UTF-8">
    <title>Local Storage Example</title>
</head>
<body>
<h2>User Details (Local Storage)</h2>
<button onclick="showData()">Show Stored Data</button>
<p id="output"></p>
<script>
    // User object
    const user = {
        name: "Sangam Kumar",
        username: "sangam123",
        email: "sangam123@gmail.com",
        password: "sangam@339"
    };
    // Store data in localStorage
    localStorage.setItem("user", JSON.stringify(user));
    // Function to show data on button click
    function showData() {
        // Get data from localStorage
        const data = JSON.parse(localStorage.getItem("user"));
        // Display data in HTML
        document.getElementById("output").innerHTML =
            "Name: " + data.name + "<br>" +
            "Username: " + data.username + "<br>" +
            "Email: " + data.email + "<br>" +
            "Password: " + data.password;
    }
</script></body></html>
```

User Details (Local Storage)

Show Stored Data

Name: Sangam Kumar
Username: sangam123
Email: sangam123@gmail.com
Password: sangam@339




```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Local Storage User Input</title>
</head>
<body>
<h2>Local Storage User Input Example</h2>
<input type="text" id="username" placeholder="Enter Username"><br><br>
<input type="email" id="email" placeholder="Enter Email"><br><br>
<button onclick="saveData()">Save Data</button>
<button onclick="getData()">Get Data</button>
<button onclick="clearData()">Clear Data</button>
<p id="output"></p>
<script>
  function saveData() {
    let user = document.getElementById("username").value;
    let mail = document.getElementById("email").value;
    localStorage.setItem("username", user);
    localStorage.setItem("email", mail);
    document.getElementById("output").innerHTML = "Data Saved Successfully";
  }
  function getData() {
    let user = localStorage.getItem("username");
    let mail = localStorage.getItem("email");
    document.getElementById("output").innerHTML =
      "Username: " + user + "<br>Email: " + mail;
  }
  function clearData() {
    localStorage.clear();
    document.getElementById("output").innerHTML = "All Data Cleared";
  }
</script></body></html>
```

Local Storage – User Input

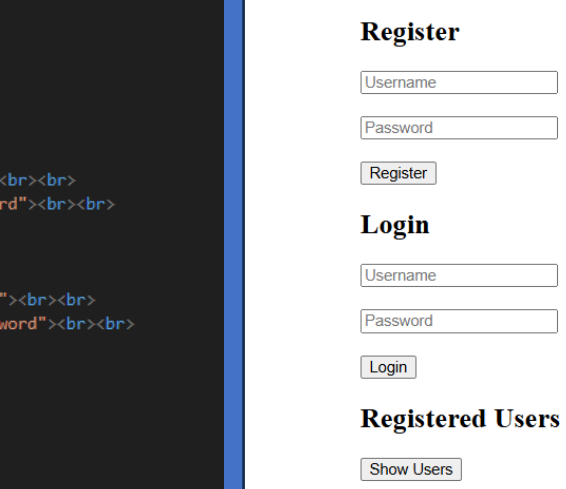
Sangam Kumar

sangam339@gmail.com

Save Data | Get Data | Clear Data

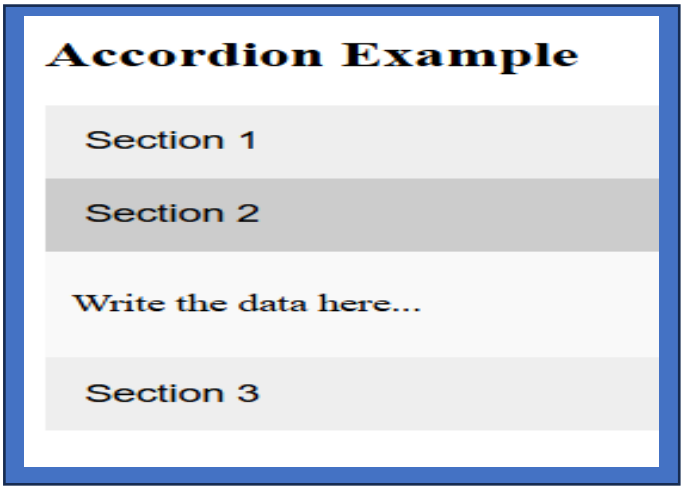
Username: Sangam Kumar
Email: sangam339@gmail.com

```html
<!DOCTYPE html>
<html>
<head>
    <title>User Login/Registration</title>
</head>
<body>
    <h2>Register</h2>
    <input type="text" id="regUsername" placeholder="Username"><br><br>
    <input type="password" id="regPassword" placeholder="Password"><br><br>
    <button onclick="registerUser()">Register</button>
    <p id="regMessage"></p>
    <h2>Login</h2>
    <input type="text" id="loginUsername" placeholder="Username"><br><br>
    <input type="password" id="loginPassword" placeholder="Password"><br><br>
    <button onclick="loginUser()">Login</button>
    <p id="loginMessage"></p>
    <h2>Registered Users</h2>
    <button onclick="displayUsers()">Show Users</button>
    <ul id="userList"></ul>
    <script>
        function registerUser() {
            const username = document.getElementById('regUsername').value.trim();
            const password = document.getElementById('regPassword').value.trim();
            const regMessage = document.getElementById('regMessage');
            if (!username || !password) {
                regMessage.textContent = "Please enter both username and password.";
                return;
            }
            let users = JSON.parse(localStorage.getItem('users')) || {};
            if (users[username]) {
                regMessage.textContent = "Username already exists.";
                return;
            }
            users[username] = password;
            localStorage.setItem('users', JSON.stringify(users));
            regMessage.textContent = "Registration successful!";
            document.getElementById('regUsername').value = "";
            document.getElementById('regPassword').value = "";
            displayUsers();
        }
        function loginUser() {
            const username = document.getElementById('loginUsername').value.trim();
            const password = document.getElementById('loginPassword').value.trim();
            const loginMessage = document.getElementById('loginMessage');
            let users = JSON.parse(localStorage.getItem('users')) || {};
            if (users[username] === password) {
                loginMessage.textContent = "Login successful!";
                document.getElementById('loginUsername').value = "";
                document.getElementById('loginPassword').value = "";
            } else {
                loginMessage.textContent = "Invalid credentials.";
            } }
        function displayUsers() {
            const userList = document.getElementById('userList');
            userList.innerHTML = '';
            let users = JSON.parse(localStorage.getItem('users')) || {};
            if (Object.keys(users).length === 0){
                userList.innerHTML = "No users registered yet.";
                return;
            }
            for (const username in users) {
                const li = document.createElement('li');
                li.textContent = `Username: ${username}, Password: ${users[username]}`;
                userList.appendChild(li);
            }}
    </script></body></html>
```



Example for accordion

```html
<!DOCTYPE html>
<html>
<head>
<title>Accordion Example</title>
<style>
.accordion {
  width: 100%;
  padding: 15px;
  background: #eee;
  border: none;
  text-align: left;
  cursor: pointer;
  font-size: 16px;
}
.accordion.active { background: #ccc;}
.panel {
  display: none;
  padding: 10px;
  background: #f9f9f9;
}
</style>
</head>
<body>
<h2>Accordion Example</h2>
<button class="accordion">Section 1</button>
<div class="panel"><p>Write the data here...</p></div>
<button class="accordion">Section 2</button>
<div class="panel"><p>Write the data here...</p></div>
<button class="accordion">Section 3</button>
<div class="panel"><p>Write the data here...</p></div>
<script>
const acc = document.querySelectorAll(".accordion");
acc.forEach((btn) => {
  btn.addEventListener("click", () => {
    acc.forEach((item) => {
      if (item !== btn) {
        item.classList.remove("active");
        item.nextElementSibling.style.display = "none";
      }});
    btn.classList.toggle("active");
    const panel = btn.nextElementSibling;
    panel.style.display =
      panel.style.display === "block" ? "none" : "block";
  });});
</script></body></html>
```



Example for Carousel

```
<!DOCTYPE html>
<html>
<head>
<title>Image Slider</title>
<style>
.slider {
  width: 400px;
  height: 250px;
  margin: auto;
  overflow: hidden;
}
.slide {
  display: none;
  width: 100%;
  height: 100%;
  object-fit: cover;
}
.dots {text-align: center;}
.dot {
  height: 12px;
  width: 12px;
  margin: 5px;
  background: #bbb;
  border-radius: 50%;
  display: inline-block;
  cursor: pointer;
}
.active {background: black;}
</style>
</head>
<body>
<h2 align="center">Image Slider</h2>
<div class="slider">
  <img class="slide" src="rpsir.jpg">
  <img class="slide" src="image.png">
  <img class="slide" src="rpsir.jpg">
</div>
<div class="dots">
  <span class="dot" onclick="currentSlide(1)"></span>
  <span class="dot" onclick="currentSlide(2)"></span>
  <span class="dot" onclick="currentSlide(3)"></span>
</div>
<script>
let index = 0;
showSlides();
function showSlides() {
  let slides = document.getElementsByClassName("slide");
  let dots = document.getElementsByClassName("dot");
  for (let i = 0; i < slides.length; i++) {
    slides[i].style.display = "none";
    dots[i].classList.remove("active");
  }
  index++;
  if (index > slides.length) {
    index = 1;
  }
  slides[index - 1].style.display = "block";
  dots[index - 1].classList.add("active");
  setTimeout(showSlides, 2000);
}
function currentSlide(n) {
  index = n - 1;
  showSlides();
}
</script>
</body>
</html>
```

JavaScript Interview Questions & Answers

1. What is JavaScript?

→ JavaScript is a **client-side scripting language** used to create **dynamic and interactive web pages**.

2. Is JavaScript compiled or interpreted?

→ JavaScript is an **interpreted language** (modern engines use Just-In-Time compilation).

3. What are data types in JavaScript?

→ Primitive: String, Number, Boolean, Null, Undefined, BigInt, Symbol

→ Non-primitive: Object, Array, Function

4. Difference between var, let, and const?

- var → function scope
- let → block scope
- const → block scope, cannot be reassigned

5. What is hoisting?

→ Hoisting moves **variable and function declarations** to the top of their scope before execution.

6. What is undefined vs null?

- undefined → variable declared but not assigned
- null → intentional empty value

7. What is an array?

→ An array stores **multiple values in a single variable**.

→ let arr = [1, 2, 3];

8. What is an object?

→ An object stores data in **key–value pairs**.

→ let user = { name: "Raj", age: 20 };

9. What is a function?

A function is a **block of reusable code**.

```
function add(a, b) {
  return a + b;
}
```

10. What is arrow function?

→ Shorter syntax for functions.

→ const add = (a, b) => a + b;

11. Difference between == and ===?

- == → compares value only
- === → compares value and type

12. What is DOM?

→ DOM (Document Object Model) represents the **HTML structure** as objects.

13. How to select an element in DOM?

→ document.getElementById("id");

➔ document.querySelector(".class");

14. What is event handling?

→ Responding to user actions like click, keypress, mouseover.

15. What is this keyword?

→ this refers to the **current object**.

16. What is callback function?

→ A function passed as an argument to another function.

17. What is closure?

→ A function that **remembers its outer scope variables**.

18. What is promise?
→ Promise handles **asynchronous operations**.
→ Promise.then().catch();
19. What is async / await?
→ Used to write asynchronous code in a **synchronous style**.
20. What is setTimeout()?
→ Executes code **after a delay**.
→ setTimeout(() => console.log("Hi"), 2000);
21. What is setInterval()?
→ Executes code **repeatedly after a fixed time**.
22. What is localStorage?
→ Stores data **permanently in browser**.
→ localStorage.setItem("name", "Raj");
23. Difference between localStorage and sessionStorage?

- localStorage → permanent
- sessionStorage → cleared after tab close

24. What is JSON?
→ JavaScript Object Notation used for **data exchange**.
25. What is exception handling?
→Handling errors using try...catch.
26. What is NaN?
→ Represents **Not a Number**.
27. What is isNaN()?
→ Checks whether value is not a number.
28. What is event bubbling?
→ Event flows from **child to parent**.
29. What is event capturing?
→ Event flows from **parent to child**.
30. What is JavaScript used for?

- Form validation
- Dynamic UI
- Web apps
- Games
- APIs

21. What is an IIFE?
→IIFE (Immediately Invoked Function Expression) runs **immediately after creation**.

```
(function () {
  console.log("Hello");
})();
```

22. What is typeof operator?
→ Used to check the **data type** of a variable.
→ typeof 10; // number
23. What is NaN?
→NaN means **Not a Number**.
24. What is isNaN()?
→ Checks whether a value is **NaN or not**.
25. What is map()?
→ Creates a **new array** by applying a function to each element.

26. What is filter()?
→ Returns elements that **match a condition**.
27. What is reduce()?
→ Reduces array to **single value**.
28. What is forEach()?
→ Loops through array elements (no return).
29. Difference between map() and forEach()?

- map() → returns new array
- forEach() → returns nothing

30. What is spread operator (...)?
→ Used to **copy or merge** arrays/objects.
31. What is rest parameter?
→ Collects **multiple arguments** into an array.
32. What is destructuring?
→ Extracts values from arrays/objects.
33. What is call(), apply(), bind()?
→Used to change the value of this.
34. What is prototype?
→Mechanism by which objects **inherit properties**.
35. What is event delegation?
→ Handling events using **parent element**.
36. What is preventDefault()?
→ Stops default browser behavior.
37. What is stopPropagation()?
→ Stops event bubbling.
38. What is strict mode?
Enforces **clean and secure JavaScript**.
"use strict";
39. What is setTimeout() vs setInterval()?

- setTimeout() → runs once
- setInterval() → runs repeatedly

40. What is window object?
→ Global object in the browser.
41. What is the difference between null and undefined?

- undefined → Variable declared but **not assigned**
- null → **Intentional empty value**

```
let a;
let b = null;
```

42. What is the Event Loop?
→ The Event Loop handles **asynchronous tasks** and moves them from the queue to the call stack.
43. What is currying in JavaScript?
→ Currying is a technique where a function takes **one argument at a time**.

```
function add(a) {
  return function(b) {
    return a + b;
  };
}
```

What is RID Organization (RID संस्था क्या है)?

- **RID Organization** यानि **Research, Innovation and Discovery Organization एक** संस्था हैं जो TWF (TWKSAA WELFARE FOUNDATION) **NGO** द्वारा RUN किया जाता है | जिसका मुख्य उदेश्य हैं आने वाले समय में सबसे पहले **NEW (RID, PMS & TLR)** की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो |
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW (RID, PMS & TLR)** के माध्यम से किया जाये इसके लिए ही **इस RID Organization** की स्थपना 30.09.2023 किया गया है | जो TWF द्वारा संचालित किया जाता है |
- TWF (TWKSAA WELFARE FOUNDATION) NGO की स्थपना 26-10-2020 में बिहार की पावन धरती सासाराम में Er. RAJESH PRASAD एवं Er. SUNIL KUMAR द्वारा किया गया था जो की भारत सरकार द्वारा मान्यता प्राप्त संस्था हैं |
- Research, Innovation & Discovery में रूचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियो से मैं आवाहनं करता हूँ की आप सभी **इस RID संस्था** से जुड़ें एवं अपने बुध्दि, विवेक एवं प्रतिभा से दुनियां को कुछ नई **(RID, PMS & TLR)** की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें |

MISSION, VISSION & MOTIVE OF "RID ORGANIZATION"	
मिशन	हर एक ONE भारत के संग
विजन	TALENT WORLD KA SHRESHTM AB AAYEGA भारत में और भारत का TALENT भारत में
मकसद	NEW (RID, PMS, TLR)

MOTIVE OF RID ORGANIZATION NEW (RID, PMS, TLR)		
NEW (RID)		
R	I	D
Research	Innovation	Discovery
NEW (TLR)		
T	L	R
Technology, Theory, Technique	Law	Rule
NEW (PMS)		
P	M	S
Product, Project, Production	Machine	Service

RID रीड संस्था की मिशन, विजन एवं मकसद को सार्थक हमें बनाना हैं |
भारत के वर्चस्व को हर कोने में फैलना हैं |
कर के नया कार्य एक बदलाव समाज में लाना हैं |
रीड संस्था की कार्य-सिध्दांतों से ही, हमें अपनी पहचान बनाना हैं |

Er. Rajesh Prasad (B.E, M.E)
Founder:
TWF & RID Organization

JavaScript के इस **E-Book** में अगर मिलती त्रुटी मिलती है तो कृपया हमें सूचित करें | **WhatsApp's No: 9202707903 or**

Email Id: ridorg.in@gmail.com

|| धन्यवाद ||