**JavaScript**

**JavaScript**: JavaScript is a programming language that’s used to give instructions to a computer and display the output.

Input (code or set of instructions) 🡪 **Computer** 🡪 Display the Output.

**Note**: JavaScript code extension: hello.js

//1.print with console.log();

console.log("Hello");// with double qoutes

console.log('Hello');// with single qoutes

**Print with console:** console.log() is used to log(print) a message to the console.

* + **Syntax:** **console.log ();** 🡪 **console.log("Hello world!");**

1. **console:** A built-in JavaScript object that provides access to the browser's debugging console.
2. **Log():** A method of the console object, that’s used to print messages to the console, for **testing purposes.**

//1. print with console.log(): --> for testing purpose

console.log("Hello JavaScript");// with double qoutes

console.log('Hello JavaScript');// with double qoutes

console.log("Mohammad Al-Amin"); // with double qoutes

console.assert.log('Mohammad Al-Amin'); // with double qoutes

What can Do JavaScript

1. Change html contents.
2. Change html attributes.
3. Change CSS style.
4. Can show and hide html elements.
5. Handle Events
6. Communicate with Servers.
7. Communicate with APIs and so on.

index.html

  <!-- Change HTML Content -->

    <h2>Change HTML Content</h2>

    <p id="contentDemo">Original Content</p>

    <button onclick="changeContent()">Change Content</button>

    <!-- Change HTML Attributes -->

    <h2>Change HTML Attributes</h2>

    <button onclick="changeImageOn()">Change Image: On</button>

    <img id="imageDemo" src="/00. image/off.gif" alt="Old Image" width="100">

    <button onclick="changeImageOff()">Change Image: Off</button>

    <!-- Change CSS Styles -->

    <h2>Change CSS Styles</h2>

    <p id="styleDemo">Lorem ipsum dolor, sit amet consectetur adipisicing elit. Reiciendis saepe suscipit eaque ducimus

        at adipisci odit voluptates veniam exercitationem assumenda.</p>

    <button onclick="changeStyle()">Change Style</button>

    <!-- Hide and Show Elements -->

    <h2>Hide and Show Elements</h2>

    <p id="visibilityDemo">This text can be hidden or shown.</p>

    <button onclick="hideElement()">Hide</button>

    <button onclick="showElement()">Show</button>

    <!-- Handle Events -->

    <h2>Handle Events</h2>

    <button id="eventButton">Click Me!</button>

<!-- Communicate with Servers -->

    <h2>Communicate with Servers</h2>

    <button onclick="fetchData()">Fetch Data</button>

    <p id="serverOutput"></p>

    <!-- Work with APIs -->

    <h2>Work with APIs</h2>

    <button onclick="fetchApiData()">Fetch API Data</button>

    <p id="apiOutput"></p>

    <script src="/1. Introduction/index.js"></script>

index.js

function changeContent() { // Change HTML Content

    document.getElementById("contentDemo").innerHTML = "Hello, World!";

}

function changeImageOn() { // Change HTML Attributes

    document.getElementById("imageDemo").src="/Javascript basic-advance/image/on.gif";

}

function changeImageOff() { // Change HTML Attributes

    document.getElementById("imageDemo").src="/Javascript basic-advance/image/off.gif";

}

function changeStyle() { // Change CSS Style

    const element = document.getElementById("styleDemo");

    element.style.color = "blue";

    element.style.fontSize = "20px";

    element.style.fontWeight = "bold";

}

function hideElement() { // Hide and Show Elements

    document.getElementById("visibilityDemo").style.display = "none";

}

function showElement() {

    document.getElementById("visibilityDemo").style.display = "block";

}

document.getElementById("eventButton").onclick = function() { // Handle Events

    alert("Button clicked!");

};

function fetchData() { // Communicate with Servers

    fetch('https://jsonplaceholder.typicode.com/posts/1') // Example API

        .then(response => response.json())

        .then(data => {

            document.getElementById("serverOutput").innerHTML = `Title: ${data.title}`;

        })

        .catch(error => console.log('Error:', error));

}

function fetchApiData() { // Work with APIs

    fetch('https://jsonplaceholder.typicode.com/users/1') // Example API

        .then(response => response.json())

        .then(data => {

            document.getElementById("apiOutput").innerHTML = `Name: ${data.name}`;

        })

        .catch(error => console.log('Error:', error));

}

**Where to Write JavaScript:**

1. **In the <head> section of HTML**: You can include JavaScript multiple times by using the <script></script> tag within the <head> section.
2. **In the <body> section of HTML**: JavaScript can also be added multiple times within the <body> section using the <script></script> tag.
3. **In an external file**: You can link an external JavaScript file multiple times by using the <script **src="filename.js**"></script> tag in either the <head> or <body> sections.
4. **From an external URL**: You can load JavaScript from an external URL multiple times using the <script **src="URL"></**script> tag, commonly for libraries like jQuery or Bootstrap.

Index.html

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>JavaScript Types Example</title>

    <script>

        // JavaScript in the head section

        console.log('JavaScript from the head section');

    </script>

    <script src="/2. where to write/index.js"></script> <!-- External JavaScript file linked in the head -->

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script> <!-- External URL JavaScript (e.g., jQuery) -->

</head>

<body>

    <h1>JavaScript Display in HTML</h1>

    <p id="demo">This text will change from JavaScript in the body.</p>

    <!-- JavaScript in the body section -->

    <script>

        document.getElementById("demo").innerHTML = "JavaScript changed this text.";  // Writing into an HTML element using innerHTML

        console.log('JavaScript from the body section');  // Displaying output using console.log

        alert('This is a JavaScript alert from the body section');  // Displaying an alert box

        document.write("This is text written using document.write.");   // Using document.write

    </script>

<script src="/2. where to write/index.js"></script> <!-- External JavaScript file linked in the body -->

    <!-- Another script using the external jQuery URL -->

    <script>

        $(document).ready(function () {

            console.log('JavaScript using jQuery from an external URL');

        });

    </script>

Index.js:

// External JavaScript file (index.js)

console.log('JavaScript from an external file (index.js)');

**Display Possibilities of JavaScript**

1. **Writing into an HTML element using innerHTML:**  
   You can update the content of an HTML element by setting its innerHTML property.
2. **Displaying output in the console using console.log():**  
   You can log information to the browser's console for debugging purposes using console.log().
3. **Showing an alert box using window.alert():**  
   You can display a popup alert message to the user with window.alert().
4. **Writing directly to the document using document.write():**  
   You can output content directly to the webpage using document.write(), though it’s rarely used in modern web development.

**Index.html**

   <h1>JavaScript Display Possibilities</h1>

    <!-- Paragraph to display innerHTML content -->

    <p id="demo\_dis1"></p><br><br>

    <!-- Paragraph to display document.write content -->

    <p id="demo\_dis2"></p><br><br>

    <script src="/2. where to write/index.js"></script>

**Index.js:**// 1. Writing into an HTML element using innerHTML

document.getElementById("demo\_dis1").innerHTML = "Mohammad Al-Amin";

// 2. Log the content of the element with id "demo\_dis1" to the console

console.log("Content of 'demo\_dis1':", document.getElementById("demo\_dis1").innerHTML);

// 3. Display an alert box with a message

window.alert("Danger! This is an alert message.");

// 4. Writing directly into the HTML output using document.write (replaced with innerHTML to avoid disrupting the page)

document.getElementById("demo\_dis2").innerHTML = "100 + 200 = " + (100 + 200);

Syntax-Statement-Comment

1. **Computer Program**: A set of instructions written to perform a specific task or solve a problem on a computer.

* Consists of multiple instructions arranged in a logical sequence.
* Each instruction contributes to achieving the overall task, such as calculating numbers or displaying a webpage.

1. **Instruction**: A single operation or command that the computer can execute. Each instruction tells the computer what action to take, like adding two numbers or printing text.

* Executed one by one in a program.
* An instruction could be fetching numbers from memory, performing calculations, and storing results.

1. **Syntax**: The set of rules that define the structure of a computer program. It determines how instructions should be written.

* Every programming language has its own syntax that must be followed exactly.
* Incorrect syntax will result in errors, preventing the program from running correctly.

1. **Statement**: A complete line of code that performs an action, like declaring a variable or making a function call.

* A complete line of code that performs an action, similar to a sentence in human language.
* Can declare a variable (let x = 5;), perform an operation (x = x + 1;), or call a function (console.log(x);).

1. **Comment**: Text in the code that is not executed by the program, used to explain or annotate the code for humans.

* Used to explain or annotate code for better understanding.
* In JavaScript, // is for single-line comments and /\* \*/ for multi-line comments.
* Comments do not affect the execution of the program.

Index.html:  
 <h1>Computer Program - Instruction - Syntax - Statement - Comment</h1>

   <button onclick="exampleProgram()">Run Program</button>

    <p id="output"></p>

    <script src="/4. Statement-Syntax-Coment/index.js"></script>

Index.js  
// Computer Program: This is the entire code that will be run when the button is clicked.

function exampleProgram() {

    // Instruction: A single task like declaring a variable.

    let num1 = 5; // This is a statement that declares a variable 'num1'.

    let num2 = 10; // Another statement that declares 'num2'.

    // Instruction: Adding two numbers and storing the result.

    let sum = num1 + num2; // This statement adds num1 and num2 and stores the result in 'sum'.

    // Syntax: Ensures that the code follows the rules. If I miss a semicolon here, it could cause an error.

    document.getElementById("output").innerText = "Sum of num1 and num2 is: " + sum;

    // Comment: This line is explaining that the result will be displayed on the webpage.

    // Comments are not executed by the program.

}

**Identifier**

identifier.js

// 1. Variable Declarations

let x = 10; // let keyword: Declares a block-scoped variable x

const pi = 3.14; // const keyword: Declares a block-scoped, read-only constant pi

var name = 'Alice'; // var keyword: Declares a variable name with function or global scope

// 2. Conditional Statements

if (x > 5) { // if keyword: Executes code block if condition is true

  console.log('x is greater than 5'); // Executes if x is greater than 5

} else { // else keyword: Executes code block if condition is false

  console.log('x is 5 or less'); // Executes if x is 5 or less

}

// 3. Loop

for (let i = 0; i < 5; i++) { // for keyword: Creates a loop with initialization, condition, and increment

  console.log(i); // Logs numbers 0 to 4

}

// 4. Function Declaration and Usage

function greet(name) { // function keyword: Defines a function named greet

  return 'Hello, ' + name; // return keyword: Returns a value from the function

}

console.log(greet('Alice')); // Output: Hello, Alice

// 5. Switch Statement

let day = 'Friday';

switch (day) { // switch keyword: Selects code block based on the value of day

  case 'Monday': // case keyword: Matches the value to execute specific block

    console.log('Start of the work week');

    break; // break keyword: Exits the switch block

  case 'Friday': // case keyword: Matches the value to execute specific block

    console.log('End of the work week');

    break; // break keyword: Exits the switch block

  default: // default keyword: Executes if no cases match

    console.log('Midweek');

}

// 6. Error Handling

try { // try keyword: Executes code that may throw an error

  // Code that may throw an error

  let result = x / 0; // This will not throw an error but is an example

  console.log(result);

} catch (error) { // catch keyword: Catches and handles errors thrown in the try block

  // Code to handle the error

  console.error('An error occurred:', error);

} finally { // finally keyword: Executes code regardless of an error

  // Code to run regardless of error

  console.log('Execution completed.');

}

namingConvetion.js

console.log("\n\n");

// let first-name = 'John'; // 1. Hyphens are not allowed in identifiers: SyntaxError: Unexpected token '-'

let first\_name = 'Tahsin',last\_name = 'Ahmed'; // 2. Underscore

console.log(`Underscore: ${first\_name}, ${last\_name}`);

let FirstName = 'John',LastName = 'Doe'; // 3. Upper Camel Case (Pascal Case)

console.log(`Upper Camel Case: ${FirstName}, ${LastName}`);

let firstName = 'John',lastName = 'Doe', masterCard = '1234-5678-9876-5432', interCity = 'Bus';

// 4. Lower Camel Case (ALWAYS use it)

console.log(`Lower Camel Case: ${firstName}, ${lastName}, ${masterCard}, ${interCity}`);

let caseSensitive = 'This is case-sensitive',CaseSensitive = 'This is also valid but different'; // 5. Case Sensitivity

console.log(`Case Sensitivity: ${caseSensitive}, ${CaseSensitive}`);

// 6. Cannot Start with a Digit:  let 1stVariable = 'Invalid'; // SyntaxError: Invalid or unexpected token

let first2ndVariable = 'Valid';

console.log(`Cannot Start with a Digit: ${first2ndVariable}`);

// 7. Keywords: let var = 'Invalid'; // SyntaxError: Unexpected token 'var'

let validVar = 'This is valid';

console.log(`Keywords: ${validVar}`);

keyword.js

// 1. Variable Declarations

let x = 10; // let keyword: Declares a block-scoped variable x

const pi = 3.14; // const keyword: Declares a block-scoped, read-only constant pi

var name = 'Alice'; // var keyword: Declares a variable name with function or global scope

// 2. Conditional Statements

if (x > 5) { // if keyword: Executes code block if condition is true

  console.log('x is greater than 5'); // Executes if x is greater than 5

} else { // else keyword: Executes code block if condition is false

  console.log('x is 5 or less'); // Executes if x is 5 or less

}

// 3. Loop

for (let i = 0; i < 5; i++) { // for keyword: Creates a loop with initialization, condition, and increment

  console.log(i); // Logs numbers 0 to 4

}

// 4. Function Declaration and Usage

function greet(name) { // function keyword: Defines a function named greet

  return 'Hello, ' + name; // return keyword: Returns a value from the function

}

console.log(greet('Alice')); // Output: Hello, Alice

// 5. Switch Statement

let day = 'Friday';

switch (day) { // switch keyword: Selects code block based on the value of day

  case 'Monday': // case keyword: Matches the value to execute specific block

    console.log('Start of the work week');

    break; // break keyword: Exits the switch block

  case 'Friday': // case keyword: Matches the value to execute specific block

    console.log('End of the work week');

    break; // break keyword: Exits the switch block

  default: // default keyword: Executes if no cases match

    console.log('Midweek');

}

// 6. Error Handling

try { // try keyword: Executes code that may throw an error

  // Code that may throw an error

  let result = x / 0; // This will not throw an error but is an example

  console.log(result);

} catch (error) { // catch keyword: Catches and handles errors thrown in the try block

  // Code to handle the error

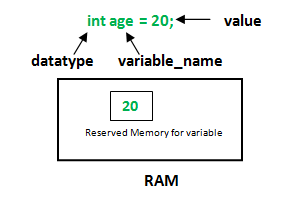
  console.error('An error occurred:', error);

} finally { // finally keyword: Executes code regardless of an error

  // Code to run regardless of error

  console.log('Execution completed.');

Variable and Data Types

**Variables:** Variables are Containers for Storing Data, that can be declared in 4 ways:

**Note:** JavaScript is dynamically typed language

1. **Automatically:** **Global Scope** (can be re-declare and update)

// 1. declare Variables with automatically: Global Scope (can be re-declare and update)

userName ="Mohammad Al-Amin";

userId=101;

userAddress="Dhaka, bangladesh";

userConatact="01745157083";

userDegree="B.Sc in CSE";

userSalary=50000.00;

console.log("userName: ",userName);

console.log("userId: ",userId);

console.log("userAddress: ",userAddress);

console.log("userConatact: ",userConatact);

console.log("userDegree: ",userDegree);

console.log("userSalary: ",userSalary);

userName ="Mohammad Yasin"; // re-declare is possible

console.log("userName: ",userName);

userId=102; //update

console.log("userId: ",userId);

1. **Using-var:** **Global Scope** (can be re-declare and update).

// 2.declare Variables with keyword - var: Global Scope (can be re-decalre and update)

var Number1=100;

var Number2=100.112131323;

console.log("Number1: ",Number1);

console.log("Number2: ",Number2);

var Number1=200; //re-decalre

var Number2=200.222131323; //re-decalre

console.log("Number1: ",Number1);

console.log("Number2: ",Number2);

Number1=300; //update

Number2=300.332131323; //update

console.log("Number1: ",Number1);

console.log("Number2: ",Number2);

1. **Using let:** **Block Scope** (can’t be re-declare but it can be update).

// 3.declare Variables with keyword - let: Block Scope (can't be re-decalre but it canbe  update)

let Number3=400;

let Number4=500.552131323;

console.log("Number3: ",Number3);

console.log("Number4: ",Number4);

// let Number3=400; //cant' be re-derclare

// let Number4=500.552131323; //cant' be re-derclare

// console.log("Number3: ",Number3);

// console.log("Number4: ",Number4.toFixed(2));

Number3=600; //update

Number4=700.772131323; //update

console.log("Number3: ",Number3);

console.log("Number4: ",Number4);

1. **Using const: Block Scope** (can’t be re-declare and update).

// 4.declare Variables with keyword - const: Block Scope (can't be re-decalre and update)

const Number5=800;

const Number6=900.992131323;

console.log("Number5: ",Number5);

console.log("Number6: ",Number6);

// const Number5=800;//cant' be re-derclare

// const Number6=900.992131323;//cant' be re-derclare

// console.log("Number5: ",Number5);

// console.log("Number6: ",Number6);

// Number5=1000; // can't be update

// Number6=1000.1992131323; // can't be update

// console.log("Number5: ",Number5);

// console.log("Number6: ",Number6);

**Variable naming Rules:**

1. Variable names are case sensitive; “a” & “A” is different.

//1. Variable names are case sensitive; “a” & “A” is different.

let fullName="Tahsen Ahmed Al-Amin";

let FullName="Mohammad Imran";

console.log("Full Name: ",fullName );

console.log("Full Name: ",FullName );

1. Only letters (a-z / A-Z), digits (1…), underscore( \_ ) and $ is allowed. (not even space).

//2. Only letters (a-z / A-Z), digits (1…), underscore( \_ ) and $ is allowed. (not even space)

let numbeR=1000.12323;

console.log("Number: ", numbeR);

let number200=2000.12323;

console.log("Number: ", number200);

let \_number=3000.3123312;

console.log("Number: ", \_number);

let $number=4000.3243458947;

console.log("Number: ", $number);

1. Only a letter, underscore (\_) or $ should be 1st character.
2. Reserved words and digit cannot be variable names.

// 3)Reserved words and  digit  cannot be variable names.

// 12number=5000.3432434; // digit can't be the 1st character

// console.log("Number: ", 12number);

// let new=15000.3432434; // digit can't be the 1st character

// console.log("Number: ", new);

1. Variable names can’t contain spaces.

// 4). Variable names can’t contain spaces.

// let full name="Mohammad Yasin- Imran"; //cant't contain space

// console.log("Name: ", full name );

**Different type of variables name cases:**

**fullName**: Camel Case (Generally max. time use it),**fullname**: Lowercase,**FULLNAME**: Uppercase (or CONSTANT Case), **full\_name**: Snake Case, **full-name**: Kebab Case (or Hyphen Case), **FullName**: Pascal Case.

**Datatypes: 1. primitive (7 types), 2. non-primitive (objects🡪 Arrays, Functions etc.)**

**1. primitive-data types: number, string, Boolean, undefined, null, symbol, BigInt**

// 1. primitive-data types: string, number, boolean, undefined, null,  Bigint,symbol

let str="Amin";//string

let num1=100.12331; //number

let bool=true; // boolean

let num2; //undefinde

let num3=null; //null--> it's a object, null means absence of a objects

let Bigint=BigInt("123"); // BinInt() ->>1 to n

let symbol=Symbol("AlAmin95");

console.log("str: ", str, typeof(str));

console.log("num1: ",num1, typeof(num1));

console.log("bool: ",bool, typeof(bool));

console.log("num2: ",num2, typeof(num2));

console.log("num3: ",num3, typeof(num3));

console.log("Bigint: ", Bigint, typeof(Bigint));

console.log("symbol:  ", symbol, typeof(symbol));

1. **non-primitive data types: (objects**🡪 **Arrays, Functions etc.)**

* **Objects: it’s a collection of values , that is the key value pairs.**

**Example: create object name: students🡪 name; string, age: number; marks: numbers, isPass: Boolean.**

const student={ // object

    fullName: "Al-Amin", //key

    age:20,

    cgpa: 3.09,

    isPAss: true

}

console.log(student, typeof(student))

console.log(student["fullName"], student['age']); // access individual keys of student

console.log(student.fullName, student.age, student.cgpa); // access individual keys of student

// update name and age

student.fullName="Tahsin";

student.age=25;

console.log(student["fullName"], student['age']); // access individual keys of student

**JavaScript Comment: Part of code which is not executed.**

//Single line comment

/\*

1. Multiple line comment

2.

3.

4.

…

\*/

**Operator’s**

JavaScript operators are used to perform different types of mathematical and logical computations.

**Types:** 1. Arithmetic Operators, 2. Assignment Operators, 3. Comparison Operators, 4. String Operators**,** 5**.** Logical Operators,

1. Bitwise Operators, 7. Ternary Operators, 8. Type Operators.

**Arithmetic Operators**

**Arithmetic Operators:** a. summation (+), Subtraction (-), Multiplication (\*), Division (/), Modulus (%), Exponentiation (\*\*)

b. Unary Operator: Increment (++), Decrement (--)

arithmaticOperator.html:  
<h1>Assignment Operator in JavaScript</h1>

    <h3 id="sum"></h3>

    <h3 id="sub"> </h3>

    <h3 id="mul"></h3>

    <h3 id="div"></h3>

    <h3 id="rem"></h3>

    <h3 id="exp"></h3> <br><br>

    <h3 id="number2"></h3>

    <h3 id="preIncrement"></h3>

    <h3 id="postIncrement1"></h3>

    <h3 id="postIncrement2"></h3>

    <h3 id="preDecrement"></h3>

    <h3 id="postDecrement1"></h3>

    <h3 id="postDecrement2"></h3>

    <script src="/6. Operrators/assignmenOperator.js"></script>

arithmaticOperator.js:  
let number1 = 2;

let number2 = 3;

let sum = number1 + number2; // summation

let sub = number1 - number2; // subtraction

let mul = number1 \* number2; // multiplication

let div = number1 / number2; // division

let rem = number1 % number2; // remainder (modulus)

let exp = number1 \*\* number2; // exponential (power: 2^3=8)

document.getElementById("sum").innerText = "Summation: " + sum;

document.getElementById("sub").innerText = "Subtraction: " + sub;

document.getElementById("mul").innerText = "Multiplication: " + mul;

document.getElementById("div").innerText = "Division: " + div;

document.getElementById("rem").innerText = "Remainder: " + rem;

document.getElementById("exp").innerText = "Exponential: " + exp;

//print in console

console.log(sum);

console.log(sub);

console.log(mul);

console.log(div);

console.log(rem);

console.log(exp);

// Pre-increment and post-increment

// Initial Number

document.getElementById("number2").innerHTML = "Initial Number: " + number2;

console.log("Before increment-decrement: ", number2);

// Pre-increment

let preIncrement = ++number2; // first increment then print

document.getElementById("preIncrement").innerHTML = "Pre increment - first increment then print: " + preIncrement;

console.log("\nAfter pre increment (number2): ", number2);

console.log("After pre increment (preIncrement): ", preIncrement);

// Post-increment

let postIncrement = number2++; // first print then increment

document.getElementById("postIncrement1").innerHTML = "Post increment - first print then increment: " + postIncrement;

document.getElementById("postIncrement2").innerHTML = "Post increment (after operation): " + number2;

console.log("\nAfter postIncrement (postIncrement): ", postIncrement);

console.log("After postIncrement (number2): ", number2);

// Pre-decrement

let preDecrement = --number2; // first decrement then print

document.getElementById("preDecrement").innerHTML = "Pre decrement - first decrement then print: " + preDecrement;

console.log("\nAfter pre decrement (number2): ", number2);

console.log("After pre decrement (preDecrement): ", preDecrement);

// Post-decrement

let postDecrement = number2--; // first print then decrement

document.getElementById("postDecrement1").innerHTML = "Post decrement - first print then decrement: " + postDecrement;

document.getElementById("postDecrement2").innerHTML = "Post decrement (after operation): " + number2;

console.log("\nAfter postDecrement (postDecrement): ", postDecrement);

console.log("After postDecrement (number2): ", number2);

**Assignment Operators**

**Assignment Operators:** Assignment operators assign values to JavaScript variable,

* (=, +=,-=, \*=, /=, %=, \*\*=)

assignmentOperator.html:

 <h1>Assignment Operator</h1>

    <h3 id="sum"></h3>

    <h3 id="sub"></h3>

    <h3 id="mul"></h3>

    <h3 id="div"></h3>

    <h3 id="rem"></h3>

    <h3 id="exp"></h3>

    <script src="/6. Operrators/assignmentOperator.js"></script>

assignmentOperator.js:

let number1 = 2;

let number2 = 3;

let sum = number1 + number2; // summation

let sub = number1 - number2; // subtraction

let mul = number1 \* number2; // multiplication

let div = number1 / number2; // division

let rem = number1 % number2; // remainder (modulus)

let exp = number1 \*\* number2; // exponential (power: 2^3=8)

document.getElementById("sum").innerText = "Summation: " + sum;

document.getElementById("sub").innerText = "Subtraction: " + sub;

document.getElementById("mul").innerText = "Multiplication: " + mul;

document.getElementById("div").innerText = "Division: " + div;

document.getElementById("rem").innerText = "Remainder: " + rem;

document.getElementById("exp").innerText = "Exponential: " + exp;

//print in console

console.log(sum);

console.log(sub);

console.log(mul);

console.log(div);

console.log(rem);

console.log(exp);

// Pre-increment and post-increment

// Initial Number

document.getElementById("number2").innerHTML = "Initial Number: " + number2;

console.log("Before increment-decrement: ", number2);

// Pre-increment

let preIncrement = ++number2; // first increment then print

document.getElementById("preIncrement").innerHTML = "Pre increment - first increment then print: " + preIncrement;

console.log("\nAfter pre increment (number2): ", number2);

console.log("After pre increment (preIncrement): ", preIncrement);

// Post-increment

let postIncrement = number2++; // first print then increment

document.getElementById("postIncrement1").innerHTML = "Post increment - first print then increment: " + postIncrement;

document.getElementById("postIncrement2").innerHTML = "Post increment (after operation): " + number2;

console.log("\nAfter postIncrement (postIncrement): ", postIncrement);

console.log("After postIncrement (number2): ", number2);

// Pre-decrement

let preDecrement = --number2; // first decrement then print

document.getElementById("preDecrement").innerHTML = "Pre decrement - first decrement then print: " + preDecrement;

console.log("\nAfter pre decrement (number2): ", number2);

console.log("After pre decrement (preDecrement): ", preDecrement);

// Post-decrement

let postDecrement = number2--; // first print then decrement

document.getElementById("postDecrement1").innerHTML = "Post decrement - first print then decrement: " + postDecrement;

document.getElementById("postDecrement2").innerHTML = "Post decrement (after operation): " + number2;

console.log("\nAfter postDecrement (postDecrement): ", postDecrement);

console.log("After postDecrement (number2): ", number2);

**Shift Assignment Operators**

shiftAssignmentOperators.html:  
 <h1>Shift Assignment Operators</h1>

    <p id="leftShift"></p>

    <p id="rightShift"></p>

    <p id="unsignedRightShift"></p>

    <script src="/6. Operrators/shiftAssignmentOperators.js"></script>

shiftAssignmentOperators.js:

// Initial values and their binary representation

let x = 2; // Binary: 00000000000000000000000000000010

let y = 3; // Binary: 00000000000000000000000000000011

let z = 4; // Binary: 00000000000000000000000000000100

x <<= 2; // Left shift by 2:  Binary after shift:  00000000000000000000000000001000 (8 in decimal)

y >>= 2; // Right shift by 2: Binary after shift:  00000000000000000000000000000000 (0 in decimal)

z >>>= 2; // Unsigned right shift by 2: Binary after shift:  00000000000000000000000000000001 (1 in decimal)

document.getElementById("leftShift").innerHTML = "Left Shift: " + x;

console.log("x: ", x);

document.getElementById("rightShift").innerHTML = "Right Shift: " + y;

console.log("y: ", y);

document.getElementById("unsignedRightShift").innerHTML = "Unsigned Right Shift: " + z;

console.log("z: ", z);

**Bitwise Assignment Operators**

bitwiseAssignmentOperators.html:

<h2>Bitwise Assignment Operators Example</h2>

    <p id="andOperation"></p>

    <p id="orOperation"></p>

    <p id="xorOperation"></p>

    <p id="leftShiftOperation"></p>

    <p id="rightShiftOperation"></p>

    <p id="unsignedRightShiftOperation"></p>

    <!-- Link to the JavaScript file -->

    <script src="/6. Operrators/bitwiseAssignmentOperator.js"></script>

bitwiseAssignmentOperators.js:

let a = 5;  // 00000101 in binary

let b = 3;  // 00000011 in binary

// Bitwise AND Assignment

let andValue = a;

andValue &= b; // andValue = andValue & b

document.getElementById('andOperation').textContent = `Bitwise AND Assignment: ${andValue} (binary: ${andValue.toString(2)})`;

// Bitwise OR Assignment

let orValue = a;

orValue |= b; // orValue = orValue | b

document.getElementById('orOperation').textContent = `Bitwise OR Assignment: ${orValue} (binary: ${orValue.toString(2)})`;

// Bitwise XOR Assignment

let xorValue = a;

xorValue ^= b; // xorValue = xorValue ^ b

document.getElementById('xorOperation').textContent = `Bitwise XOR Assignment: ${xorValue} (binary: ${xorValue.toString(2)})`;

// Bitwise Left Shift Assignment

let leftShiftValue = a;

leftShiftValue <<= 1; // leftShiftValue = leftShiftValue << 1

document.getElementById('leftShiftOperation').textContent = `Bitwise Left Shift Assignment: ${leftShiftValue} (binary: ${leftShiftValue.toString(2)})`;

// Bitwise Right Shift Assignment

let rightShiftValue = a;

rightShiftValue >>= 1; // rightShiftValue = rightShiftValue >> 1

document.getElementById('rightShiftOperation').textContent = `Bitwise Right Shift Assignment: ${rightShiftValue} (binary: ${rightShiftValue.toString(2)})`;

// Bitwise Unsigned Right Shift Assignment

let unsignedRightShiftValue = a;

unsignedRightShiftValue >>>= 1; // unsignedRightShiftValue = unsignedRightShiftValue >>> 1

document.getElementById('unsignedRightShiftOperation').textContent = `Bitwise Unsigned Right Shift Assignment: ${unsignedRightShiftValue} (binary: ${unsignedRightShiftValue.toString(2)})`;

**Comparison Operator**

it compares between two values.

* Equal to (==)🡪**check only value**, Equal to & type (===)🡪 **check value and data types**, Not equal to (! =), Not equal to &type (! ==), greater than (>), Less than(<), Greater than or equal(>=), Less than or equal(<=)

**comparisonOperator.html:**

<h1>Comparison Operator</h1>

    <h2 id="demo1"></h2>

    <h2 id="demo2"></h2>

    <h2 id="demo3"></h2>

    <h2 id="demo4"></h2>

    <h2 id="demo5"></h2>

    <h2 id="demo6"></h2>

    <h2 id="demo7"></h2>

    <h2 id="demo8"></h2>

    <h2 id="demo9"></h2>

    <script src="/6. Operrators/comparisonOperator.js"></script>

**comparisonOperator.JS:**

let num1 = 100;

let num2 = 200;

document.getElementById("demo1").innerHTML = "equal to: " + (num1==num2); // check only the value

console.log(num1 == num2);

document.getElementById("demo2").innerHTML = "equal value and type to: " + (num1===num2); // check only the value

console.log(num1 === num2);

document.getElementById("demo3").innerHTML = "check  not equal to: " + (num1!=num2); // check if only not equal  then true, otherwise false

console.log(num1 != num2);

document.getElementById("demo4").innerHTML = "num1 is greater than num2: " + (num1>num2); // check  only num1 is greater than num2

console.log(num1 > num2);

document.getElementById("demo5").innerHTML = "num1 is less than num2: " + (num1<num2); // check  only num1 is less than num2

console.log(num1 < num2);

document.getElementById("demo6").innerHTML = "num1 is greater than or equal num2: " + (num1>=num2); // check  only num1 is greater than or equal num2

console.log(num1 > num2);

document.getElementById("demo7").innerHTML = "num1 is less than or equal num2: " + (num1<=num2); // check  only num1 is less than or equal num2

console.log(num1 < num2);

**Logical Operators**

Logical operators are used to combine two or more conditions and return a boolean value (**true or false**). In JavaScript, there are three main logical operators:

* **logical AND (&&):** if two conditions are true then return true otherwise return false.
* **logical OR (||):** if two conditions are false then return false otherwise return true.
* **logical NOT (!):** it’s made false value to true vice-versa.

logicalOperator.html

<h1>Logical Operators in JavaScript</h1>

    <h3 id="AND"></h3>

    <h3 id="OR"></h3>

    <h3 id="NOT"></h3>

    <h3 id="combination1"></h3>

    <h3 id="combination2"></h3>

    <h3 id="combination3"></h3>

    <script src="/6. Operrators/logicalOperator.js"></script>

logicalOperator.js

// Initializing variables

let x = 200,y = 200,a = 10,b = 10;

let AND = x && y; // Both are non-zero (truthy), so result will be 200

let OR = x || y; // Both are truthy, so result will be 200

let NOT = !(x == b); // x is 200 and b is 10, so x == b is false. NOT false = true

let combination1 = ((a == b) && !(a === null)); // a == b is true, and a is not null, so true && true = true

let combination2 = ((a === b) || !(b === null)); // a === b is true, and !(b === null) is true, so true || true = true

let combination3 = ((a == b) == !(a === null)); // a == b is true, and !(a === null) is true, so true == true = true

// Logging results to console

console.log("AND: ", AND); // Logs: 200

console.log("OR: ", OR);   // Logs: 200

console.log("NOT: ", NOT); // Logs: true

console.log("Combination 1: ", combination1); // Logs: true

console.log("Combination 2: ", combination2); // Logs: true

console.log("Combination 3: ", combination3); // Logs: true

// Displaying results in the HTML

document.getElementById("AND").innerHTML = "AND: " + AND;

document.getElementById("OR").innerHTML = "OR: " + OR;

document.getElementById("NOT").innerHTML = "NOT: " + NOT;

document.getElementById("combination1").innerHTML = "Combination 1: " + combination1;

document.getElementById("combination2").innerHTML = "Combination 2: " + combination2;

document.getElementById("combination3").innerHTML = "Combination 3: " + combination3;

**Bitwise Operators**

1. **AND (&)**: Compares each bit of two numbers.
   * **if both bits are same from both, set=1, otherwise, set=0.**
   * Example: 5 & 3 (binary: 0101 & 0011 = 0001), result is 1.
2. **OR (|)**: Compares each bit of two numbers.
   * **if at least one bit is 1 from both, sets=1, otherwise sets=0.**
   * Example: 5 | 3 (binary: 0101 | 0011 = 0111), result is 7.
3. **XOR (^)**: Compares each bit of two numbers.
   * **if bits are different from both, set=1, and otherwise, set= 0**
   * Example: 5 ^ 3 (binary: 0101 ^ 0011 = 0110), result is 6.
4. **NOT (~)**: Inverts the bits of the number.

* **Change each 1 to 0 and each 0 to 1.**
* Example: ~5 (binary: ~0101), result is -6 (because inverting the bits of 5 gives a negative number in signed representation).

1. **Zero fill left shift (<<):**  This operator shifts the bits of the number to the left by a specified number of positions. **The empty positions on the right are filled with 0s.** Bits that are shifted out on the left are discarded.
   * **Zeros are shifted in from the right, and the leftmost bits fall off.**
   * Example: 5 << 1 (binary: 0101 << 1 = 1010), result is 10.
2. **Signed right shift (>>)**: This operator shifts the bits of the number to the right. **For positive numbers, the leftmost bits are filled with 0s,** and **for negative numbers, they are filled with 1s to preserve the sign**. Bits that are shifted out on the right are discarded.
   * **The leftmost bits are filled by copying the sign bit (0 for positive numbers, 1 for negative numbers).**
   * Example: 5 >> 1 (binary: 0101 >> 1 = 0010), result is 2.
3. **Zero fill right shift (>>>)**: This operator shifts the bits of the number to the right, just like the signed right shift, but the **leftmost bits are always filled with 0s,** regardless of the sign of the number. Bits that are shifted out on the right are discarded.

* **Zeros are shifted in from the left, regardless of the sign of the number.**
* Example: 5 >>> 1 (binary: 0101 >>> 1 = 0010), result is 2.

bitwiseOperator.html

 <h1>Bitwise Operator</h1>

    <h3 id="AND"></h3>

    <h3 id="OR"></h3>

    <h3 id="XOR"></h3>

    <h3 id="NOT\_a"></h3>

    <h3 id="NOT\_b"></h3>

    <h3 id="zeroFilledLeftShift"></h3>

    <h3 id="signedRightShift1"></h3>

    <h3 id="signedRightShift2"></h3>

    <h3 id="zeroFilledRightShift"></h3>

    <script defer src="/6. Operrators/practice.js"></script>

bitwiseOperator.js

let a = 5;  // (binary 0101)

let b = 3;  // (binary 0011)

let AND = a & b; // AND operation: 5: 0101 & 3: 0011 => 0101 & 0011 = 0001 = 1

let OR = a | b; // OR operation: 5: 0101 | 3: 0011 => 0101 | 0011 = 0111 = 7

let XOR = a ^ b; // XOR operation: 5: 0101 ^ 3: 0011 => 0101 ^ 0011 = 0110 = 6

let NOTa = ~a; // NOT operation on 'a': ~5: 0101 => 1010 = -6 (in two's complement)

let NOTb = ~b; // NOT operation on 'b': ~3: 0011 => 1100 = -4 (in two's complement)

let zeroFillLeftShift = a << 1; // Zero fill left shift: 5: 0101 << 1 => 1010 = 10

let signedRightShift1 = a >> 1; // Signed right shift: 5: 0101 >> 1 => 0010 = 2

let signedRightShift2 = b >> 1;  // Signed right shift (negative shift is not valid, fixed to 1): 3: 0011 >> 1 => 0001 = 1

let zeroFillRightShift = b >>> 1; // Zero fill right shift: 3: 0011 >>> 1 => 0001 = 1

// Print results to the console and into HTML document

console.log("AND: ", AND);

console.log("OR: ", OR);

console.log("XOR: ", XOR);

console.log("NOTa: ", NOTa);

console.log("NOTb: ", NOTb);

console.log("zeroFillLeftShift: ", zeroFillLeftShift);

console.log("signedRightShift1: ", signedRightShift1);

console.log("signedRightShift2: ", signedRightShift2);

console.log("zeroFillRightShift: ", zeroFillRightShift);

document.getElementById("AND").innerHTML = "AND: " + AND;

document.getElementById("OR").innerHTML = "OR: " + OR;

document.getElementById("XOR").innerHTML = "XOR: " + XOR;

document.getElementById("NOT\_a").innerHTML = "NOTa: " + NOTa;

document.getElementById("NOT\_b").innerHTML = "NOTb: " + NOTb;

document.getElementById("zeroFilledLeftShift").innerHTML = "zeroFilledLeftShift: " + zeroFillLeftShift;

document.getElementById("signedRightShift1").innerHTML = "signedRightShift1: " + signedRightShift1;

document.getElementById("signedRightShift2").innerHTML = "signedRightShift2: " + signedRightShift2;

document.getElementById("zeroFilledRightShift").innerHTML = "zeroFilledRightShift: " + zeroFillRightShift;

**Typed Operators**

In JavaScript, a primitive value is a single value with no properties or methods, and JavaScript has 7 primitive data types: "string", "number", "boolean", "object", "bigint", "symbol", "undefined"

* **Purpose**: Used to determine the data type of a variable or expression.

**String Type:** Represents text values enclosed in single (') or double (") quotes.

* **Example**: "Hello, World!"
* **Output**: typeof returns "string" for string values.

**Number Type:** Represents numeric values (integers or floating-point numbers).

* **Example**: 42 or 3.14
* **Output**: typeof returns "number" for numeric values.

**Boolean Type:** Represents logical values: true or false. Often used in conditions and logical operations.

* **Example**: true, false
* **Output**: typeof returns "boolean" for boolean values.

**BigInt Type:** Represents large integers beyond the safe range of the number type, allowing for operations on very large numbers.

* **Example**: 9007199254740991n (numbers ending with an n)
* **Output**: typeof returns "bigint" for BigInt values.
* **Range**: BigInt supports integers larger than **(2^53 - 1) and less than -(2^53 - 1)** and has no practical limit.

**Symbol Type:**  Used to create unique and immutable identifiers, typically used as object keys to ensure uniqueness.

* **Example**: Symbol("id")
* **Output**: typeof returns "symbol" for symbol values.

**Null Type:** Represents an intentional absence of any object value. It indicates that a variable is empty or doesn't point to any object.

* **Output**: typeof returns "object" for null (this is a known bug in JavaScript).

**Undefined Type:** Represents a variable that has been declared but not assigned a value yet.

* **Example**: A variable declared without initialization.
* **Output**: typeof returns "undefined" for uninitialized variables.

**typeofOperator.html:**

    <h1>typeof(): Operator's</h1>

    <h3 id="demo1"></h3>

    <h3 id="demo2"></h3>

    <h3 id="demo3"></h3>

    <h3 id="demo4"></h3>

    <h3 id="demo5"></h3>

    <h3 id="demo6"></h3>

    <h3 id="demo7"></h3>

    <h3 id="demo8"></h3>

typeofOperator.j:

// Declare variables with different data types

let var1 = "Mohmmad Al-Amin"; // String: A sequence of characters (text).

let var2 = 1000.132; // Number: A numeric value (can be an integer or a floating-point number).

let var3 = true; // Boolean: A logical value representing true or false.

let var4 = BigInt(11434432312434); // BigInt: Used for representing large integers beyond the safe limit of number type.

let var5 = Symbol("a unique symbol"); // Symbol: A unique identifier, often used as keys in objects.

let var6 = null; // Null: An intentional absence of value, often set to represent an empty object reference.

let var7; // Undefined: A variable declared but not assigned a value yet, so its value is undefined.

// Display each variable's value and its data type in the HTML

document.getElementById("demo1").innerHTML = "var1 : " + var1 + " , data type is: " + typeof(var1); // Displays the value of var1 and its type (string).

document.getElementById("demo2").innerHTML = "var2 : " + var2 + " , data type is: " + typeof(var2); // Displays the value of var2 and its type (number).

document.getElementById("demo3").innerHTML = "var3 : " + var3 + " , data type is: " + typeof(var3); // Displays the value of var3 and its type (boolean).

document.getElementById("demo4").innerHTML = "var4 : " + var4 + " , data type is: " + typeof(var4); // Displays the value of var4 and its type (bigint).

document.getElementById("demo5").innerHTML = "var5 : " + var5.toString() + " , data type is: " + typeof(var5); // Displays the symbol as a string and its type (symbol).

document.getElementById("demo6").innerHTML = "var6 : " + var6 + " , data type is: " + typeof(var6); // Displays the value of var6 (null) and its type (object due to a known bug in JavaScript).

document.getElementById("demo7").innerHTML = "var7 : " + var7 + " , data type is: " + typeof(var7); // Displays the value of var7 (undefined) and its type (undefined).

// Console log the data type of each variable for debugging purposes

console.log("var1 data type is: ", typeof(var1)); // Logs the type of var1 (string).

console.log("var2 data type is: ", typeof(var2)); // Logs the type of var2 (number).

console.log("var3 data type is: ", typeof(var3)); // Logs the type of var3 (boolean).

console.log("var4 data type is: ", typeof(var4)); // Logs the type of var4 (bigint).

console.log("var5 data type is: ", typeof(var5)); // Logs the type of var5 (symbol).

console.log("var6 data type is: ", typeof(var6)); // Logs the type of var6 (null, but shown as "object" due to a JS quirk).

console.log("var7 data type is: ", typeof(var7)); // Logs the type of var7 (undefined).

**String Operator’s**

A **string operator** in JavaScript allows you to perform operations on strings, which are sequences of characters. String operators enable actions like concatenating, comparing, and evaluating strings within your code.

**String operators work with:**

1. **Concatenation (+ operator):**
   * The **+** operator combines two or more strings into one. It joins the strings while keeping the original strings intact.
2. **Comparison (==, === operators):**
   * The (**==**) operators compare two strings to check if they are equal.
   * The (**===**) operator checks for both value and type equality.
3. **String Length (.length property):**
   * The **.length** property is used to determine the how many number of characters in a string.
4. **Other Common String Methods:**
   * **.toUpperCase()**: Converts all characters in a string to uppercase.
   * **.toLowerCase()**: Converts all characters in a string to lowercase.
   * **.includes()**: Checks if a **specific substring exists within a string.**
   * **.slice()**: Extracts a section of a string.

**Where to use string operators:** String operators are commonly used when working with textual data. Some practical applications include:

* **Concatenating strings**: Useful for constructing sentences, messages, or creating dynamic content.
* **Comparing strings**: Often applied in validating user inputs, like password matching or performing case-insensitive comparisons.
* **Extracting and analyzing data**: Employed when processing text, such as searching for keywords or modifying specific portions of a string.

String operators are essential in web development a.

stringOperator.html:

    <h1>String Operators in JavaScript</h1>

    <!-- These elements will display the output of our JavaScript operations -->

    <h3 id="demo1"></h3>

    <h3 id="demo2"></h3>

    <h3 id="demo3"></h3>

    <h3 id="demo4"></h3>

    <h3 id="demo5"></h3>

    <h3 id="demo6"></h3>

    <h3 id="demo7"></h3>

    <h3 id="demo8"></h3>

    <h3 id="demo9"></h3>

    <h3 id="demo10"></h3>

    <!-- Link to external JavaScript file -->

    <script defer src="/6. Operrators/stringOperator.js"></script>

stringOperator.js:

// Define two string variables for demonstration

let string1 = "Mohammad Al-Amin";

let string2 = "Graduate from East West University";

//Concatenation of two strings using the + operator

let concatenated1 = string1 + "   " + string2; // JavaScript ignores multiple spaces between strings

let concatenated2 = string1 + "   Department: CSE"; // Another example of concatenation

// Convert the string to uppercase and lowercase using string methods

let toupper1 = string1.toUpperCase(); // Converts all characters in string1 to uppercase.

let lowercase1 = string1.toLowerCase(); // Converts all characters in string1 to lowercase.

// Use the includes() method to check if a substring exists in string1

let include1 = string1.includes("A"); // Checks if 'Z' is present in string1.

let include2 = string1.includes("A", 6); // Starts the search for 'Z' from index 6 in string1.

// Displaying the length of the strings

document.getElementById("demo1").innerHTML = "Length of string1 is: " + string1.length; // Displays length of string1

document.getElementById("demo2").innerHTML = "Length of string2 is: " + string2.length; // Displays length of string2

// Displaying the concatenated strings and their lengths

document.getElementById("demo3").innerHTML = "After concatenation of string1 and string2: " + concatenated1; // Displays concatenated1

document.getElementById("demo4").innerHTML = "Length after concatenation: " + concatenated1.length; // Displays the length of concatenated1

document.getElementById("demo5").innerHTML = "After second concatenation: " + concatenated2; // Displays concatenated2

document.getElementById("demo6").innerHTML = "Length after second concatenation: " + concatenated2.length; // Displays the length of concatenated2

// Displaying the results of uppercase and lowercase conversions

document.getElementById("demo7").innerHTML = "Uppercase of string1: " + toupper1; // Displays string1 in uppercase

document.getElementById("demo8").innerHTML = "Lowercase of string1: " + lowercase1; // Displays string1 in lowercase

// Displaying the results of the includes() method

document.getElementById("demo9").innerHTML = "Does string1 contain 'Z'? " + include1; // Checks if 'Z' is present in string1

document.getElementById("demo10").innerHTML = "Does string1 contain 'Z' starting from index 6? " + include2; // Checks for 'Z' in string1 starting from index 6

// Logging outputs to the browser console for debugging

console.log("Length of string1: " + string1.length); // Logs the length of string1

console.log("Length of string2: " + string2.length); // Logs the length of string2

console.log("Concatenated result: " + concatenated1); // Logs the result of concatenating string1 and string2

console.log("Uppercase of string1: ", toupper1); // Logs string1 in uppercase

**Ternary Operator/ Conditional Operator**

**Syntax:** condition?True output: false output

**ternaryOperator.html:**

<h1>Ternary Operator</h1>

    <label for="ageInput">Age: </label>

    <input type="text" id="ageInput">

    <button onclick="myFunction()">Try It</button>

    <p id="demo1"></p>

<script src="/6. Operrators/ternaryOperator.js"></script>

**ternaryOperator.js:**

function myFunction() {

    // Get the value from the input field

    let age = document.getElementById("ageInput").value;

    // Use the ternary operator to determine if the user is eligible to vote

    let eligible = (age >= 18) ? "Eligible" : "Not Eligible";

    // Display appropriate message based on eligibility

    if (age >= 18) {

        document.getElementById("demo1").innerHTML = eligible + " - You are eligible to vote.";

    } else {

        document.getElementById("demo1").innerHTML = eligible + " - You are not eligible to vote.";

    }

}

**Conditional Statements (if - else if - else, switch)**

It controls the flow of code based on conditions.

1. **if**: Executes a block of code if the specified condition is true.
2. **else**: Executes a block of code if the same condition is false.
3. **else if**: Tests a new condition if the first condition is false.
4. **switch**: Allows you to specify multiple alternative blocks of code to be executed based on different conditions.

Syntax:

if (test expression)

    {// code

    }

else if (test expression)

    {// code-- depend on if

    }

else

    {// code-- depend on if

    }  
Note:

1. if can run on its own because it checks a condition and executes code if the condition is true.

3. else and else if cannot run without an if because they depend on the if condition to determine what to do if the if condition is false.

**if - else if – else**

conditionStatement.html:  
 <label for="score">Enter Your Score: </label>

    <input type="text" name="score" id="score">

    <br><br>

    <button onclick="calculateGrade()">Grade</button>

    <h1 id="result"></h1>

    <!-- Link to the JavaScript file -->

    <script src="/6. Operrators/conditionStatement.js"></script>

conditionStatement.js:  
function calculation(score) {

    if (score >= 90) {

        return "A";

    } else if (score >= 85) {

        return "A-";

    } else if (score >= 83) {

        return "B+";

    } else if (score >= 80) {

        return "B";

    } else if (score >= 77) {

        return "B-";

    } else if (score >= 73) {

        return "C+";

    } else if (score >= 70) {

        return "C";

    } else if (score >= 60) {

        return "D";

    } else {

        return "F";

    }

}

function calculateGrade() {

    let userInput = document.getElementById("score").value;     // Get the user's input

    let score = Number(userInput);     // Convert the input to a number

    if (isNaN(score) || score < 0 || score > 100) { // Ensure that the input is a valid number

        document.getElementById("result").innerHTML = "Please enter a valid score between (0 to 100)";

        return;

    }

    let grade = calculation(score); // Calculate the grade

    document.getElementById("result").innerHTML = "Your Grade is: " + grade;     // Display the grade

}

**Switch Statement**

conditionStatement2.html:  
 <label for="score">Enter Your Score: </label>

    <input type="text" name="score" id="score">

    <br><br>

    <button onclick="calculateGrade()">Grade</button>

    <h1 id="result"></h1>

    <!-- Link to the JavaScript file -->

    <script src="/6. Operrators/conditionStatement2.js"></script>

conditionStatement2.js:

function calculation(score) {

    let grade;

    switch (true) {

        case (score >= 90):

            grade = "A";

            break;

        case (score >= 80):

            grade = "B"; // Corrected the letter case

            break;

        case (score >= 70):

            grade = "C";

            break;

        case (score >= 60):

            grade = "D";

            break;

        case (score <= 59):

            grade = "F";

            break;

        default:

            grade = "Invalid"; // To handle unexpected cases

            break;

    }return grade;

}

function calculateGrade() {

    let userInput = document.getElementById("score").value;     // Get the user's input

    let score = Number(userInput);   // Convert the input to a number

    if (isNaN(score) || score < 0 || score > 100) { // Validate the score input

        document.getElementById("result").innerHTML = "Please enter a valid score between 0 and 100.";

        return;

    }let grade = calculation(score);    // Calculate the grade

    document.getElementById("result").innerHTML = "Your grade is: " + grade;    // Display the grade

}

**JavaScript Data Types**

JavaScript has 8 primary data types, categorized into two groups: primitive and non-primitive types. Each type serves a specific purpose in programming, influencing how data is stored, manipulated, and processed.

**Total Number of Data Types:** JavaScript includes a total of 8 data types:

1. **Primitive Data Types (7)**: String, Number, BigInt, Boolean, Undefined, Null, Symbol
2. **Non-Primitive Data Type (1)**: Object

**Primitive Data Types:** Primitive data types are basic types that represent single values and are **immutable**. They include:

1. **String**
   * Represents textual data.
   * Stored as a sequence of characters enclosed in quotes (single, double, or backticks).
   * Used for manipulating and displaying text.
2. **Number**
   * Represents numeric values, including both integers and floating-point numbers.
   * Stored as 64-bit floating-point values.
   * Used for mathematical operations and numerical calculations.
3. **BigInt**
   * Represents integers that are too large for the Number type.
   * Allows for high-precision arithmetic with very large numbers.
   * Useful for applications requiring precise calculations with large integers.
   * Range= -(2^53 - 1) to 2^53 - 1 (-9,007,199,254,740,991 to 9,007,199,254,740,991).
4. **Boolean**
   * Represents a logical value: true or false.
   * Used in conditional statements and logical operations to control program flow.
5. **Undefined**
   * Indicates that a variable has been declared but not yet assigned a value.
   * Default value for variables that have not been initialized.
6. **Null**
   * Represents the intentional absence of any object value.
   * Used to explicitly denote that a variable should not hold any value.
7. **Symbol**
   * Represents a unique and immutable value primarily used as property keys in objects.
   * Ensures that property keys are unique and avoids name collisions.

**Non-Primitive Data Type:** Non-primitive data types are more complex and can hold collections of values or other objects. They include:

1. **Object**

* object data type can contain both **built-in objects**, and **user defined objects**:
* **Built-in object types can be:** objects, arrays, dates, maps, sets, intarrays, floatarrays, promises, and more.

**Note:**Immutable: it’s refers to values or objects that cannot be altered once created. Here's a concise explanation:

* **Unchangeable:** Cannot be modified after creation.
* **Consistency:** Value remains constant.
* **Predictability**: Avoids unintended changes.
* **Safe Operations:** Prevents accidental modifications.

In JavaScript, primitive data types like strings and numbers are immutable, meaning after operations create new values instead of changing the original.

**Primitive Data Types**

primritiveDataTypes.html:

<h1>Practice of DataTypes In JavaScript</h1><br><br><br>

    <!-- find data type  -->

    <h1 id="demo1"></h1>

    <h1 id="demo2"></h1>

    <h1 id="demo3"></h1>

    <h1 id="demo4"></h1>

    <h1 id="demo5"></h1>

    <h1 id="demo6"></h1>

    <h1 id="demo7"></h1>

    <h1 id="demo8"></h1>

    <h1 id="demo9"></h1>

    <script src="/5. Variable/practice.js"></script>

primritiveDataTypes.js:

let var1 = "Mohammad Al-Amin";//string: o   Stored as a sequence of characters enclosed in quotes (single, double, or backticks).

let var2 = 'Tahsin Ahmed'; //string: o  Stored as a sequence of characters enclosed in quotes (single, double, or backticks).

let var3 = 100;;// integer number, type: number(Stored as 64-bit floating-point values)

let var4 = 13212312.2434123;// float number, type: number(Stored as 64-bit floating-point values)

let var5 = true;// boolean: true or false,  type: boolean

let var6 = false;// boolean: true or false, type: boolean

let var7; // undefined:

let var8 = null; // null: intentionally don't assign any value on this variable, type: object(this is known bug of javaScript)

// find data types

document.getElementById("demo1").innerHTML = var1 + ", type = " + typeof (var1);

document.getElementById("demo2").innerHTML = var2 + ", type = " + typeof (var2);

document.getElementById("demo3").innerHTML = var3 + ", type = " + typeof (var3);

document.getElementById("demo4").innerHTML = var4 + ", type = " + typeof (var4);

document.getElementById("demo5").innerHTML = var5 + ", type = " + typeof (var5);

document.getElementById("demo6").innerHTML = var6 + ", type = " + typeof (var6);

document.getElementById("demo7").innerHTML = var7 + ", type = " + typeof (var7);

document.getElementById("demo8").innerHTML = var8 + ", type(this is known bug of JavaScript) = " + typeof (var8);

console.log(var1 + ", type = " + typeof (var1));

console.log(var2 + ", type = " + typeof (var2));

console.log(var3 + ", type = " + typeof (var3));

console.log(var4 + ", type = " + typeof (var4));

console.log(var5 + ", type = " + typeof (var5));

console.log(var6 + ", type = " + typeof (var6));

console.log(var7 + ", type = " + typeof (var7));

console.log(var8 + ", type = " + typeof (var8));

nonPrirmitiveDataType.html:  
   <h1>Practice of DataTypes In JavaScript</h1><br><br><br>

    <h1 id="demo9"></h1>

    <h1 id="demo10"></h1>

    <h1 id="demo11"></h1>

    <script src="/5. Variable/practice.js"></script>

nonPrirmitiveDataType.js:  
// find non-primirive data types

let ob = {

    firstName: "Mohammad",

    lastName: "Imran"

}; // object

document.getElementById("demo9").innerHTML = "Non-primitive data: object= [object Object], Type= " + typeof (ob);

console.log("Non-primitive data: object= ", ob, ", Type= " + typeof (ob));

let ar = ["ab", "ac", "ad"]; // array object

document.getElementById("demo10").innerHTML = "Non-primitive data: array= " + ar + ", Type= " + typeof (ar);

console.log("Non-primitive data: array= " + ar + ", Type= " + typeof (ar));

let date = new Date(); // date object

document.getElementById("demo11").innerHTML = "Non-primitive data: date= " + date + ", Type= " + typeof (date);

console.log("Non-primitive data: date= " + date + ", Type= " + typeof (date));

dataTypesOperation.html:

 <h1>DataTypes Operation in JavaScript</h1><br><br><br>

    <h1>When adding a number and a string, JavaScript will treat the number as a string:</h1>

    <h3 id="demo1"></h3><br>

    <h1>JavaScript Types are Dynamic:</h1>

    <h3 id="demo2"></h3>

    <h3 id="demo3"></h3>

    <h3 id="demo4"></h3><br><br>

    <h1>Exponential Notation:</h1>

    <h3 id="demo5"></h3>

    <h3 id="demo6"></h3><br><br>

    <h1>JavaScript Strings</h1>

    <h3 id="demo7"></h3>

    <h3 id="demo8"></h3><br><br>

    <h1>JavaScript Numbers</h1>

    <h3 id="demo9"></h3>

    <h3 id="demo10"></h3><br><br>

    <h1>JavaScript BigInt</h1>

    <h3 id="demo11"></h3>

    <h3 id="demo12"></h3><br><br>

    <h1>JavaScript Booleans: </h1>

    <h3 id="demo13"></h3>

    <h3 id="demo14"></h3><br><br>

    <h1>JavaScript Arrays:</h1>

    <h3 id="demo15"></h3>

    <h3 id="demo16"></h3>

    <h1>JavaScript Objects: </h1>

    <h3 id="demo17"></h3><br><br>

    <h1>Empty Values: </h1>

    <h3 id="demo18"></h3>

    <script src="/5. Variable/dataTypesOperation.js"></script>

dataTypesOperation.js:

// When adding a number and a string, JavaScript will treat the number as a string.

let var1 = "Tahsin" + 95;

document.getElementById("demo1").innerHTML = "After concatetion var1=  " + var1 + ",   Type = " + typeof (var1);

console.log("After concatetion var1= "+var1+", Type = " + typeof (var1));

// JavaScript Types are Dynamic

let var2; //type  is undefinde

document.getElementById("demo2").innerHTML = "var2=  " + var2 + ",   Type = " + typeof (var2);

console.log("\n\nvar2-type: ", typeof (var2));

var2 = "Mohammad Al-Amin";// type is string;

document.getElementById("demo3").innerHTML = "var2=  " + var2 + ",   Type = " + typeof (var2);

console.log("var2-type: ", typeof (var2));

var2 = 500; //type is number

document.getElementById("demo4").innerHTML = "var2=  " + var2 + ",   Type = " + typeof (var2);

console.log("var2-type: ", typeof (var2));

// Exponential Notation: Javascript numbers are always one type: double (64-bit floating point).

let var3 = 125e5;

let var4 = 125e-5;

document.getElementById("demo5").innerHTML = " Exponential Notation of 125e5 =  " + var3 + ", Type: " + typeof (var3);

document.getElementById("demo6").innerHTML = " Exponential Notation of 125e5 =  " + var4 + ", Type: " + typeof (var4);

console.log("\n\nExponential Notation of 125e5 =  " + var3 + ", Type: " + typeof (var3));

console.log("Exponential Notation of 125e5 =  " + var4 + ", Type: " + typeof (var4));

// JavaScript Strings

let var5 = "Mohammad Imran"; // use double qoutattion

let var6 = 'Mohammad Imran'; // use single qoutattion

document.getElementById("demo7").innerHTML = "String with double qooutation: " + var5 + ", Type: " + typeof (var5);

document.getElementById("demo8").innerHTML = "String with single qooutation: " + var6 + ", Type: " + typeof (var6);

console.log("String with double qooutation: " + var5 + ", Type: " + typeof (var5));

console.log("String with single qooutation: " + var6 + ", Type: " + typeof (var6));

// JavaScript Numbers

let var7 = 2000; // with decimal

let var8 = 100.123123; // without decimal

document.getElementById("demo9").innerHTML = "Number var7: " + var7 + ", type = " + typeof (var7);

document.getElementById("demo10").innerHTML = "Number var8: " + var8 + ", type = " + typeof (var8);

console.log("\n\nNumber var7: " + var7 + ", type = " + typeof (var7));

console.log("Number var8: " + var8 + ", type = " + typeof (var8));

//JavaScript BigInt : range= -(2^53 - 1) to 2^53 - 1

let var9 = BigInt("12124334343243243"); // Using a string for better precision

let var10 = 12124334343243243n;

document.getElementById("demo11").innerHTML = "BinInt- var9: " + var9 + ", type = " + typeof (var9);

document.getElementById("demo12").innerHTML = "BinInt- var10: " + var10 + ", type = " + typeof (var10);

console.log("\n\nBinInt- var9: " + var9 + ", type = " + typeof (var9));

console.log("BinInt- var9: " + var9 + ", type = " + typeof (var10));

// JavaScript Booleans : return true/false

let var11 = 50;

let var12 = 50;

let var13 = (var11 == var12);

let var14 = (var11 === var12);

document.getElementById("demo13").innerHTML = "Boolean- var14: " + var13 + ", Type: " + typeof (var13);

document.getElementById("demo14").innerHTML = "Boolean- var15: " + var14 + ", Type: " + typeof (var14);

console.log("\n\nBoolean- var14: " + var13 + ", Type: " + typeof (var13));

console.log("Boolean- var14: " + var14 + ", Type: " + typeof (var14));

// JavaScript Arrays

let ar1 = [100, 200, 300];

let ar2 = ['asd', 'adsa', 'adsds'];

document.getElementById('demo15').innerHTML = "Arrays: " + ar1 + ", Type: " + typeof (ar1);

document.getElementById('demo16').innerHTML = "Arrays: " + ar2 + ", Type: " + typeof (ar2);

console.log("\n\nArrays: " + ar1 + ", Type: " + typeof (ar1));

console.log("Arrays: " + ar2 + ", Type: " + typeof (ar2));

//JavaScript Objects

let obj = {

    fullName: "Mohammad AL-Amin",

    dept: "CSE",

    studentId: "2019-1-65-003",

}

document.getElementById('demo17').innerHTML = "Name: " + obj .fullName+", Dept: "+obj.dept+", student ID:  "+obj.studentId+ ", Type: " + typeof (obj);

console.log("\n\nName: " + obj.fullName + ", Dept: " + obj.dept + ", student ID:  " + obj.studentId + ", Type: " + typeof (obj));

// Empty Values

let var15 = "";

document.getElementById("demo18").innerHTML = "var15: " + var15 + ", type=" + typeof (var15);

console.log("\n\nvar15: " + var15 + ", type=" + typeof (var15));

**Due more details of data types:** string, number, BigInt, Boolean, array, object.