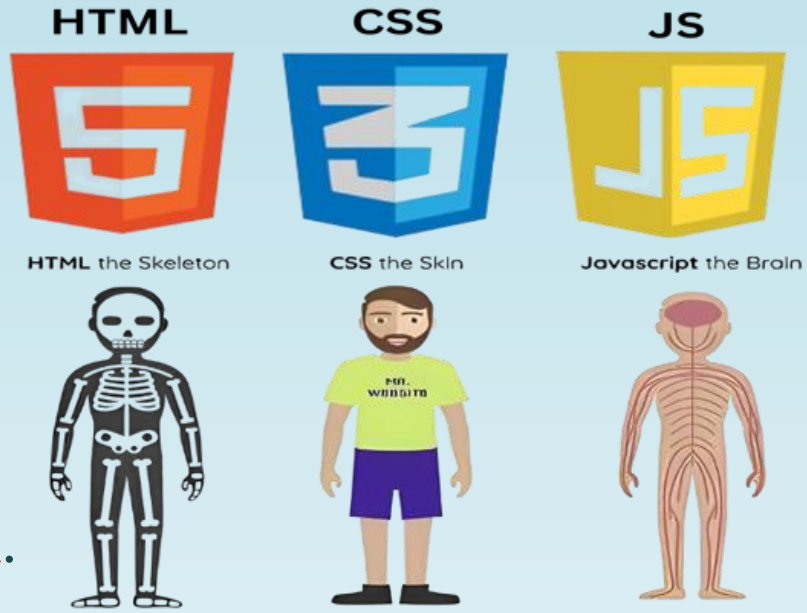
JAVASCRIPT (JS)

- Client-side Scripting Language.
- World's most popular programming language.
- Provide interactivity to the webpage.
- Originally designed to run in browsers.
- One of the 3 languages all web developers must learn.



FIRST JAVASCRIPT PROGRAM

- inserted between <script> and </script> tags when used in an HTML document.
- can be placed inside the body or the head section of an HTML page.

```
<script>
    console.log("Welcome");
    console.error("Error");
    console.warn("Warning");      // Will display the output in console
    document.write("Hello world");       // Will display the output in browser
</script>
```

FIRST JAVASCRIPT PROGRAM

- document.write("<h1>Hello world</h1>");
- alert("Login Successful"); // for getting the popup box message.

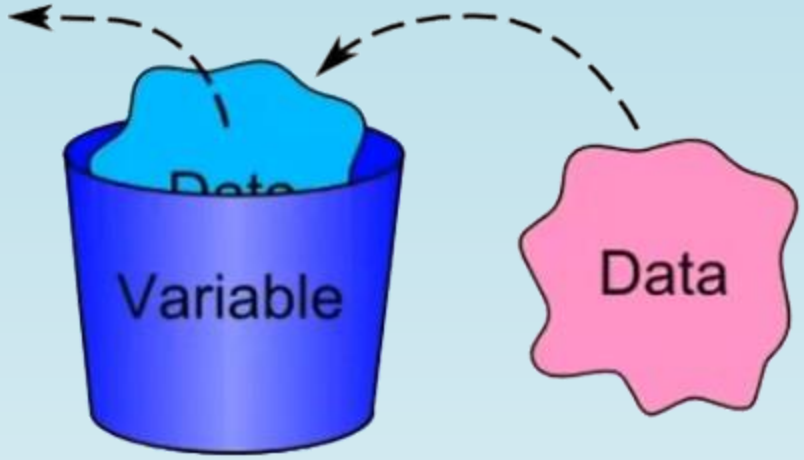
- External javascript file: index.js

<script src="index.js"></script> //for linking index.js inside HTML file

JAVASCRIPT VARIABLES

- Variables are containers for storing data.
- Can change or vary its value.
- Example:

  name="John";



- Variables can be declared in 3 ways:
  - Using var
  - Using let
  - Using const

JAVASCRIPT VARIABLES

- Block scope and Global scope
- var keyword
  - variables declared with the “var” always have Global Scope
  - Redeclaration possible.
  - Reassigning possible.

- let keyword
  - variables declared with the “let” always have Block Scope
  - Redeclaration not possible.
  - Reassigning possible.

JAVASCRIPT VARIABLES

- const keyword
  - variables declared with the “const” will be constant (value cannot be changed)
  - Redeclaration not possible.
  - Reassigning not possible.

JAVASCRIPT DATATYPES

- Two types of data:
    - Primitive datatypes
    - Non-primitive datatypes

PRIMITIVE DATATYPES

- Also called built-in datatypes/ value types.
- set of basic data types from which all other data types are constructed.
- Primitive datatypes are:
  - String — let name="manu";
  - Number — let a=5;
  - Boolean — let isAvailable=true;
  - Null — let value=Null;
  - Undefined — let num;

NON PRIMITIVE DATATYPES

- Also called derived datatypes/ reference datatypes.
- set of data types derived from primitive datatypes.

- Non-Primitive datatypes are:

  - Objects
  - Arrays
  - Functions

OBJECTS

- Is a collection of key-value pairs, separated by commas.
- Combination of different datatypes.
- Example:

```
let student={
name: “ram”,
age: 22,
isPresent: true
}
```

- Inorder to print the output
  - console.log(student);
  - console.log(student.age); OR document.write(student.age);

ARRAYS

- Collection of data of same datatypes.
- Example:
- Let days=['monday','tuesday','wednesday'];

- Inorder to print the output

  - console.log(days); OR document.write(days);
  - console.log(days[1]); OR document.write(days[1]);

FUNCTIONS

- Is a block of code designed to perform a particular task.
- Function is executed only when we invoke or call it.
- Example:

```
function welcome(){                                    //function definition
console.log("welcome to javascript class");
}

welcome();          //function call
```

FUNCTIONS

- Example 2:

```
function display(name){                    //function definition      //name-parameter
document.write ("Welcome"+name);
}

display("John");                           //function call        //"John"-argument
```

JAVASCRIPT OPERATORS

- There are different types of JavaScript operators:

  - Arithmetic Operators
  - Assignment Operators
  - Comparison Operators
  - String Operators
  - Logical Operators
  - Ternary Operators

ARITHMETIC OPERATORS

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

ASSIGNMENT OPERATORS

=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

COMPARISON OPERATORS

==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

STRING OPERATORS

- **String Comparison**

```
let text1 = "A";
let text2 = "B";
let result = text1 < text2;
document.write("Is A less than B? " + result);
```

- **String Addition**

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

LOGICAL OPERATORS

LOGICAL

- && logical and
- || logical or
- ! logical not

BUILT-IN FUNCTIONS

- Typeof - Returns the type of a variable.

- Example:

```
let name="ragu";
document.write(typeof (name));
OR
document.write(typeof name);
```

JavaScript User Interaction

- Interact with the user and respond accordingly.

- User-interface functions:
  - alert()
  - prompt()
  - confirm()

- **Alert() Method**
- Syntax: alert('text');
- Example: alert("Welcome to Javascript");

JavaScript User Interaction

- **Prompt() Method**
- Most used interface
- Can ask the user to input something
- And then use that input to build something.

- Syntax: prompt('text', default value);

- Example: let age = prompt('How old are you?', 50);
  alert(`You are ${age} years old!`);

JavaScript User Interaction

- **Confirm() Method**
- window with a question and two buttons ‘OK’ and ‘CANCEL’.

- Syntax: confirm('question’);

- Example: let isHappy = confirm(‘Are you happy?');

  alert(`You are ${isHappy}`);

CONTROL STATEMENTS

- is used to control the execution of a program based on a specific condition.

- 2 types:
  - Iterative Statements
  - Conditional Statements

Iterative statements OR JavaScript Loops

- for performing repetitive tasks efficiently.
- execute a block of code again and again while the condition is true.

- Different loops are:
- For loop
- While loop
- Do-while loop

For Loop

- Entry-controlled loop
- Three statements:
  - Initialization statement
  - Condition Statement
  - Increment statement

- Syntax:

```
for(initialization ; condition ; increment ){
//block of code
}
```

- Example:

```
for (let i = 0; i < 4; i++) {
 console.log(i);
}
```

While Loop

- Entry-controlled loop

- Syntax:

```
while (condition) {
  // code block to be executed
}
```

- Example:

```
let i = 0;
while (i < 6) {
  console.log(i);
  i++;
}
```

Do-While Loop

- Exit-controlled loop
- similar to a while loop
- block of code is executed at least once, even if the condition is false.

- Syntax:

```
do {
  // code block to be executed
}
while (condition);
```

- Example:

```
let i = 0;
do {
  console.log(i);
  i++;
}
 while (i < 6);
```

Conditional Statements

- execute specific blocks of code based on conditions.

- Conditional statements are:
  - if statement
  - else statement
  - switch statement
  - ternary operator (conditional operator)
  - nested if else statement

If Statement

- to specify a block of JavaScript code to be executed if a condition is true.
- Syntax:
- if (condition) {
- // block of code to be executed if the condition is true
- }

Else Statement

- Comes with if statement.
- to specify a block of code to be executed if the condition is false.
- Syntax:
- if (condition) {
- // block of code to be executed if the condition is true
- }
- else{
- // block of code to be executed if the condition is false
- }

Example for If-else

```
 number=2;
if (number >= 0)
{
    document.write("Number is positive");
}
 else
 {
 document.write("Number is negative");
}
```

Switch statement

- to select one of many code blocks to be executed.

- dealing with many conditions, the switch statement may be a more preferred option.

```
switch (expression) {
   case value1:
      statement1;
      break;
   case value2:
      statement2;
      break;
   . . .
default:
      statementDefault;
};
```

Example for Switch

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
document.write("Today is " + day);
```

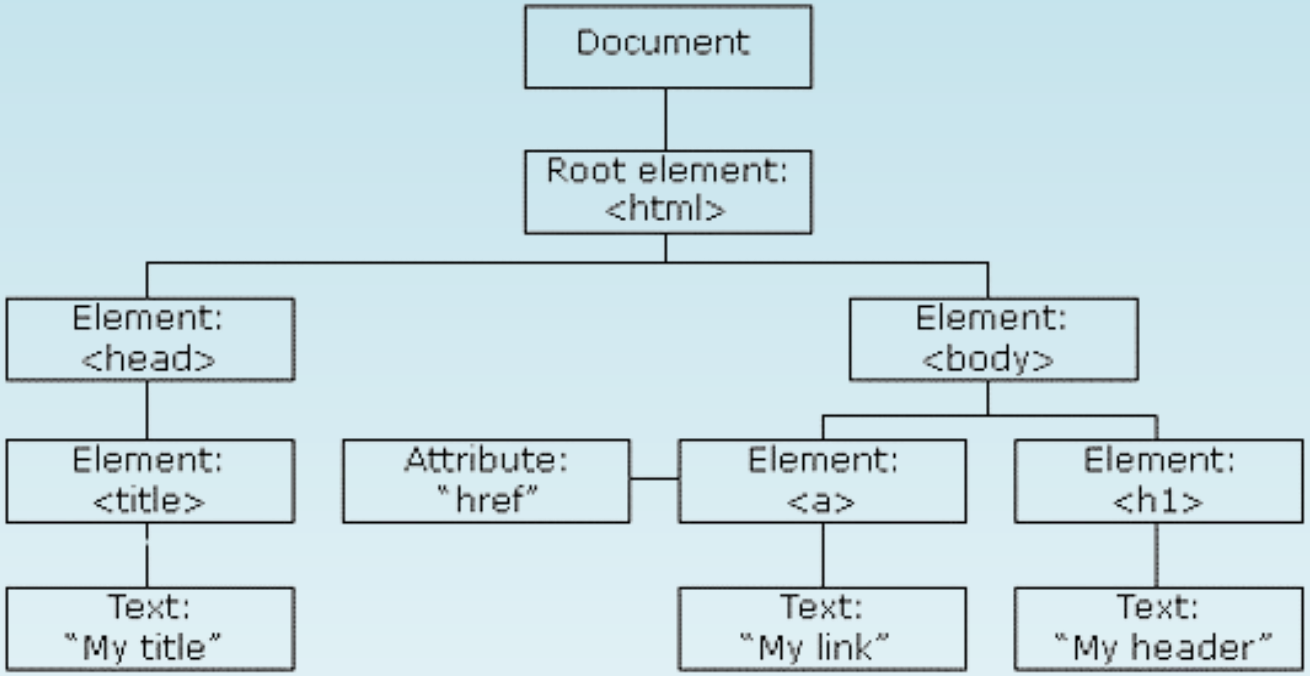
Nested if else statement

- allow us to create complex conditional logic
- by checking multiple conditions in a hierarchical manner.

```
if (condition1) {
    // Code block 1
    if (condition2) {
        // Code block 2
    } else {
        // Code block 3
    }
} else {
    // Code block 4
}
```

DOM (Document Object Model)

- interface that treats an HTML document as a tree structure.
- wherein each node is an object representing a part of the document.



DOM (Document Object Model)

- When an HTML file is loaded into the browser,
- the JavaScript can not understand the HTML document directly.

- So it interprets and interacts with the Document Object Model (DOM),
- which is created by the browser based on the HTML document.

DOM(Document Object Model)

- When a web page is loaded, browser creates the HTML DOM of the page.
- access and update the content, structure, and style of a document.

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

HTML DOM

- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

Finding HTML Elements

- Finding HTML element by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors

Finding HTML elements by id

- returns the element of specified id.
- Syntax: document.getElementById(“id-name”)
- Example:

- HTML FILE

```
<h1 id=“head">HEADING</h1>
```

- JS FILE

```
var heading=document.getElementById(‘head’);
console.log(heading);
```

innerHTML, InnerText, Style

innerHTML:

Will change the entire tag internally

innerText:

Will change only the text content inside the tag

style.css property:

Will change the css properties of the tag

InnerHTML Example

- HTML FILE

```
<p id="para">This is a boy</p>
```

- JS FILE

```
var para=document.getElementById('para');
para.innerHTML="<h1>Welcome</h1>";
```

InnerText Example

- HTML FILE

```
<p id=“para">This is a boy</p>
```

- JS FILE

```
var para=document.getElementById(‘para’);
para.innerText=“Welcome";
```

Style Example

- HTML FILE

```
<p id=“para">This is a boy</p>
```

- JS FILE

```
var para=document.getElementById(‘para’);
para.style.color=“red";
```

Finding HTML elements by Tagname

- returns all the elements of specified tag name.
- Syntax: document.getElementsByTagName(“tagname") .
- Example:

- HTML FILE

```
<h1 >HEADING</h1>
```

- JS FILE

```
var heading=document.getElementsByTagName(‘h1’);
heading[0].innerText=“WELCOME”;
```

Finding HTML elements by ClassName

- returns all the elements of specified class name.
- Syntax: document.getElementsByClassName("classname");
- Example:

- HTML FILE

```
<h1 class="head">HEADING</h1>
```

- JS FILE

```
var heading=document.getElementsByClassName('head');
heading[0].innerText="WELCOME";
```

Finding HTML elements by CSS selectors

- 2 methods
  - querySelector()- returns the first element that matches a CSS selector
  - querySelectorAll()-return all matches (not only the first)
- Syntax:

```
document.querySelector(“selectorname");
```

- Example 1: querySelector

```
<p id="para”>This is an apple </p>
var x=document.querySelector(“#para");
x.style.color="red";
```

Finding HTML elements by CSS selectors

- Example 2: querySelectorAll

```
<p class="para“>This is an apple </p>
<p class="para“>This is a mango</p>

var x=document.querySelectorAll(“.para");
x[0].style.color="red";
x[1].style.color=“blue";
```

JAVASCRIPT EVENTS

- JavaScript Events are actions or occurrences that happen in the browser.
- “Things" that happen to HTML elements.
- JavaScript can "react" on these events.

JAVASCRIPT EVENTS

- onclick - clicks an HTML element
- onmouseover - moves the mouse over an HTML element
- onmouseout - moves the mouse away from an HTML element
- onkeydown - user pushes a keyboard key
- onkeyup - when a key is released
- onchange-An HTML element has been changed
- onload-The browser has finished loading the page
- onsubmit- when a form is submitted.

JS FUNCTIONS

- block of code designed to perform a particular task.
- Code Reusability.
- JavaScript function is executed when “something” invokes it (calls it).

- Example:

```
function myFunction(x, y) {
    return x / y;
}
const value = myFunction(8, 2); // Calling the function
console.log(value);
```

JS FUNCTIONS

- Arguments and Parameters

- Variables passed along the function call – Arguments
- Variables passed along the function definition – Parameters

JS EVENT HANDLING

- Step1: Add Event Attribute to the element
  - <h1 id="text" onclick="">Good Morning</h1>
- Step2: Write scripts for actions to be done in case the event occurs
  - <h1 id="text" onclick="document.getElementById('text').style.color='red' ">Good Morning</h1>

JS EVENT HANDLING USING FUNCTIONS

- **In HTML page,**

```
<h1 id="head">Welcome</h1>
<button onclick="clicked()">Click me</button>
```

- **In JS page,**

```
function clicked() {
 document.getElementById("head").style.backgroundColor="red";
}
```

JavaScript Form Validation

- Types of Form Validation
    - Client side validation
    - Server side validation

Client side validation

- performed by the browser, before input is sent to a web server.
- This is done before the form is submitted.

- Data processing will be faster in server side validation.
- simple validations like required fields, format, and range.

Server side validation

- is performed by the server, after input has been sent to the server.
- complex validations like uniqueness, existence, and logic.

Client side validation

- Two types:
  - Basic validation : checking whether all the fields are filled.
  - Format validation : checking the format of each fields.

Basic Validation

- Checking fields empty
- Required attribute and email type
- Checking for pswd length

Format Validation