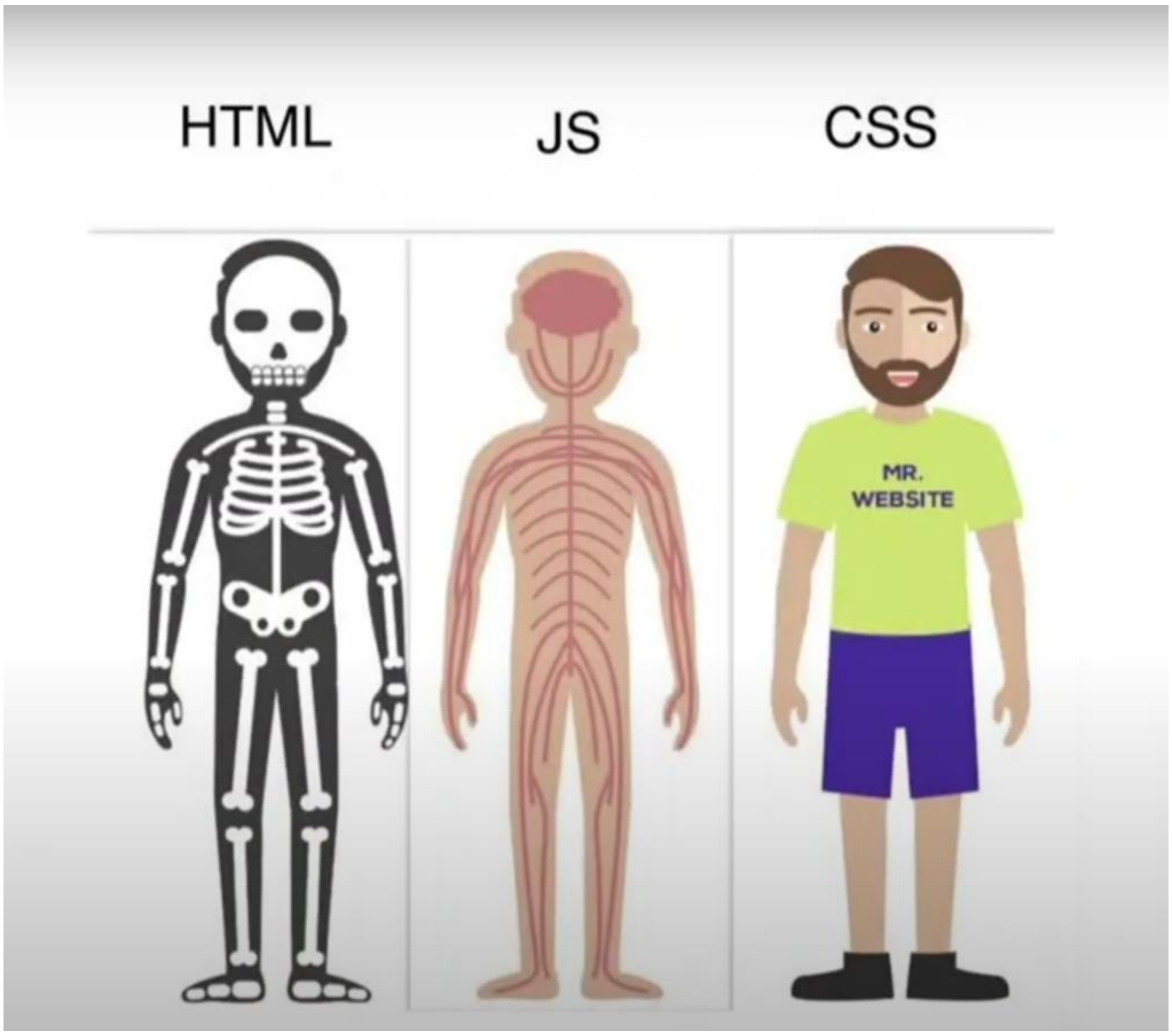
Java Script

1. why we need java script?

Developers use JavaScript in web development to add interactivity and features to improve the user experience and make the internet much more enjoyable.



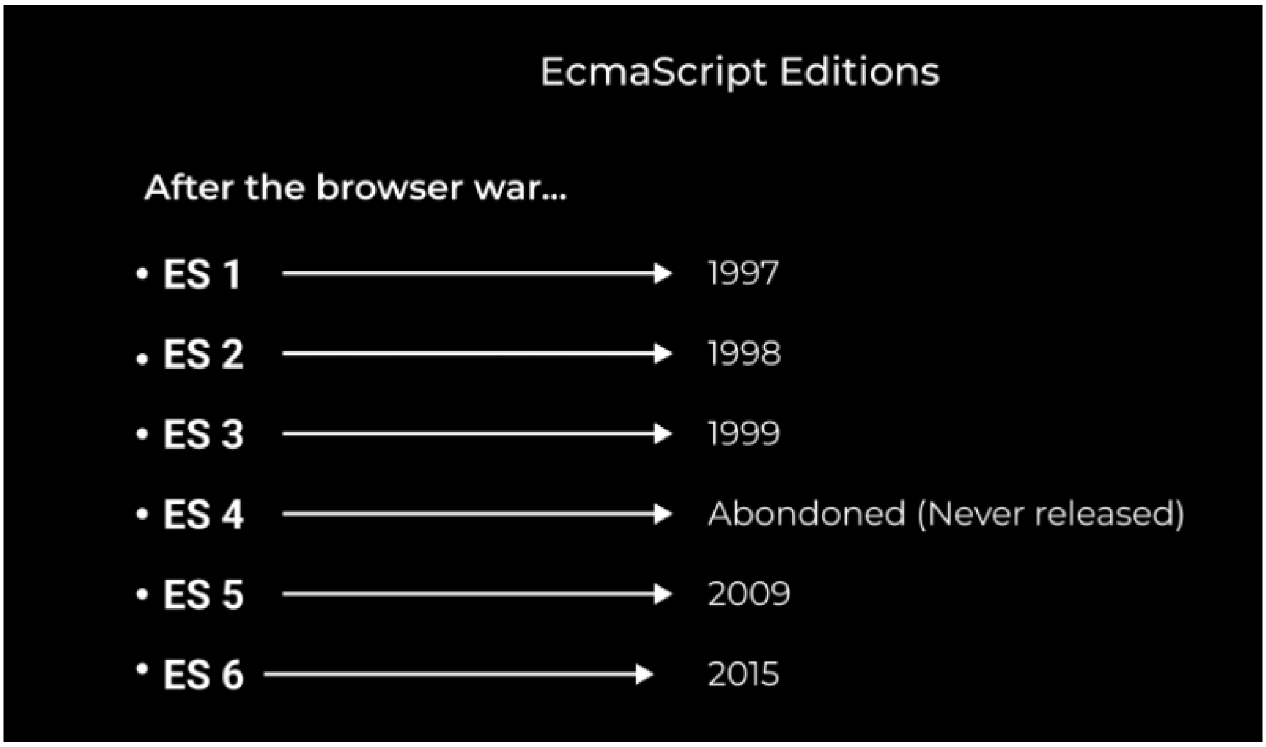
2. History of java script?

Netscape programmer named **Brendan Eich** developed a new scripting language in just **10 days**. It was originally called **Mocha**, but quickly became known as **LiveScript** and, later, **JavaScript**.

What Is ECMAScript?

When JavaScript was first introduced by Netscape, there was a war going on between all the browser vendors on the market at the time. Microsoft and several other browser vendors implemented their own versions of JavaScript (with different names and syntax) in their respective browsers. This created a huge headache for developers, as code that worked fine on one browser was a total waste on another. This went on for a while till they all agreed to use the same language (JavaScript) in their browsers.

As a result, Netscape submitted JavaScript to the European Computer Manufacturers Association (ECMA) for standardization in order to ensure proper maintenance and support of the language. Since JavaScript was standardized by ECMA, it was officially named ECMAScript.



EcmaScript Editions

After the browser war...

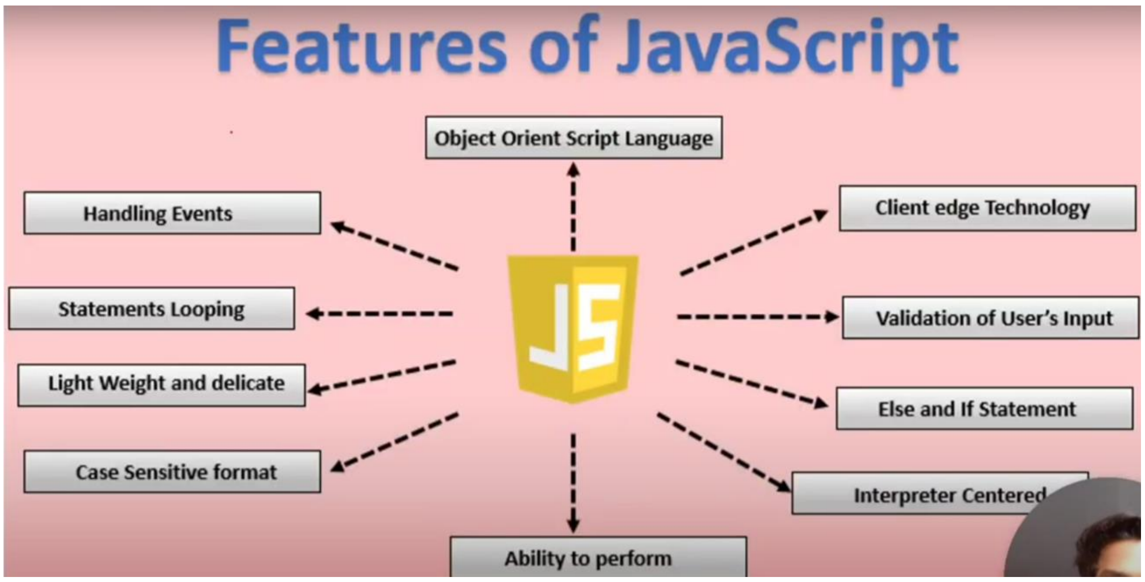
• ES 1 →	1997
• ES 2 →	1998
• ES 3 →	1999
• ES 4 →	Abondoned (Never released)
• ES 5 →	2009
• ES 6 →	2015

Originally, the name ECMAScript was just the formalization of JavaScript, but now languages like JScript and ActionScript are also based on the ECMAScript standard.

3. What is java script?

Java Script is a High level Programing Language that is primarly used to enhance the interactivity and dynamic behaviour of web sites

Java Script is also a light weight, cross platforms, Single threaded and High level Interpreted compiled programming language. It is knowns as Scripting Languages for websites.



➢ **High-Level Language**

High-level languages are programming languages that are used for writing programs or software that can be understood by humans and computers. High-level languages are easier to understand for humans because they use a lot of symbols letters phrases to represent logic and instructions in a program. It contains a high level of abstraction compared to low-level languages.

➢ **Cross-platform**

It is a software or applications that can operate on multiple operating system

➢ **Single-theaded**

It is the only programming language that can run natively in a browser, making it an instrumental part of web development. However, one critical feature of JavaScript is that it is single-threaded. This means that it can only execute one task at a time.

➢ **Asynchronous**

how it can be used to effectively handle potential blocking operations, such as fetching resources from a server.

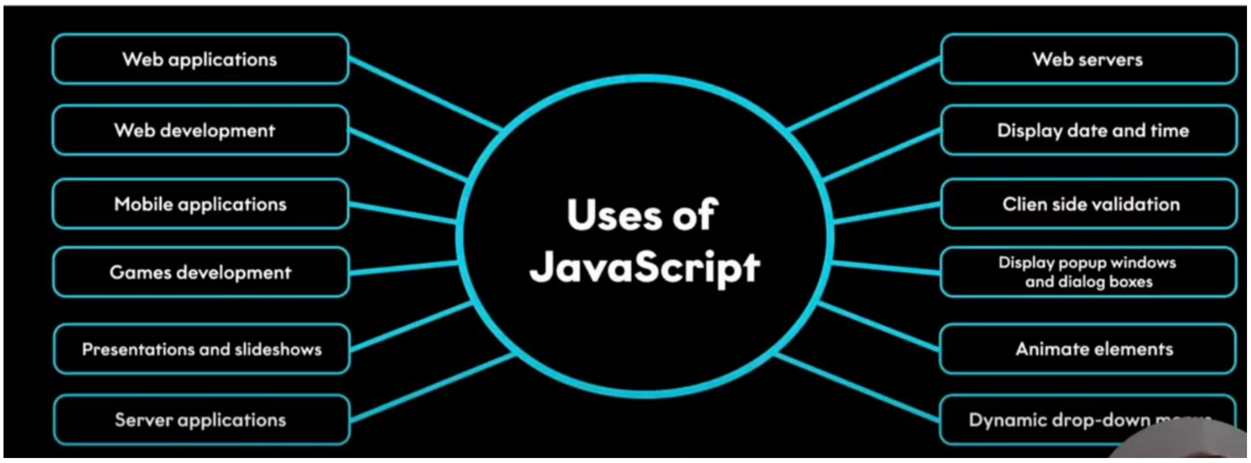
➢ **Synchronous**

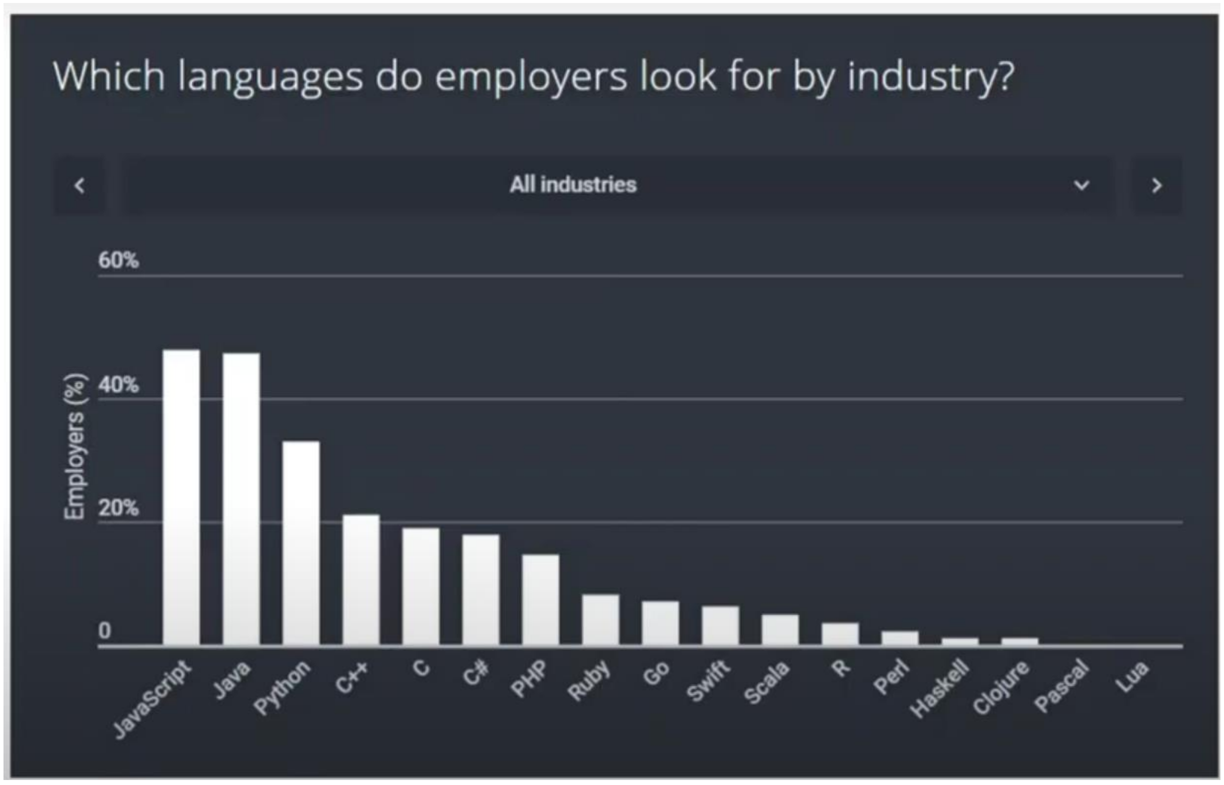
Synchronous means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

➢ **Obeject Oriented**

It provides an overview of the basic concepts of OOP.

- **Scripting**
  Scripting is used to automate tasks on a website. It can respond to any specific event, like button clicks, scrolling, and form submission. It can also be used to generate dynamic content. and JavaScript is a widely used scripting language.





4. What is Vanilla JavaScript

The term vanilla script is used to refer to the pure JavaScript (or we can say plain JavaScript) without any type of additional library. Sometimes people often used it as a joke"nowadays several things can also be done without using any additional JavaScript libraries".

The vanilla script is one of the lightest weight frameworks ever. It is very basic and straightforward to learn as well as to use. You can create significant and influential applications as well as websites using the vanilla script.

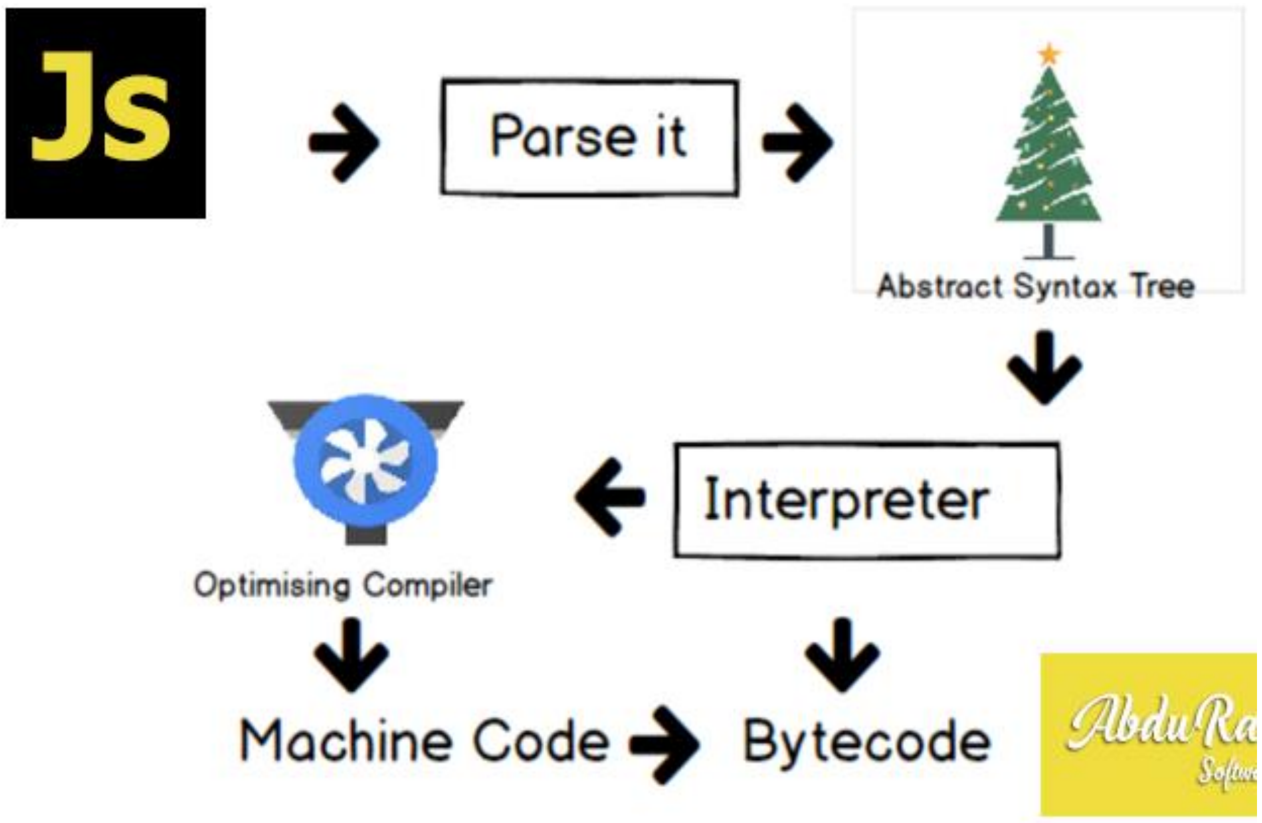
The team of developers that created the vanilla JavaScript is continuously working on it to improve it and make it more useful for the web-developers.

5. why JavaScript programming language?

It supplies objects relevant to running JavaScript on a server. For if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is **node.js**.

6. What is a JavaScript Engine?

A JavaScript engine is a program that compiles JavaScript code and executes it. It is a software that runs inside a web browser or on a server that interprets JavaScript code and executes it. The engine is responsible for parsing the JavaScript code, compiling it into machine code, and executing it.



Parsing

The first stage of a JavaScript engine is parsing. It is the process of breaking down the source code into its individual components, such as keywords, variables, and operators. The parser creates a tree structure called the Abstract Syntax Tree (AST), which represents the structure of the code.

Compilation

The next stage is compilation. The compiler takes the AST and converts it into machine code. The machine code is optimized to run efficiently on the target platform. The compilation process includes several optimizations, such as inlining, loop unrolling, and dead code elimination.

Execution

The final stage is execution. The compiled code is executed by the JavaScript engine. The engine executes the code line by line, keeping track of variables and function calls. It also manages memory allocation and deallocation.

7. Java script Run time environment?

JavaScript works on a environment called **JavaScript Runtime Environment**. To use JavaScript you basically install this environment and than you can simply use JavaScript.

So in order to use JavaScript you install a **Browser** or **NodeJS** both of which are JavaScript Runtime Environment.

call stack:

The call stack is used to store information about function calls, including local variables, parameters, and the point of execution.

Heap:

The heap is a region of memory used for dynamic memory allocation. It stores objects, arrays, and other complex data structures that are created and managed at runtime.

Web API:

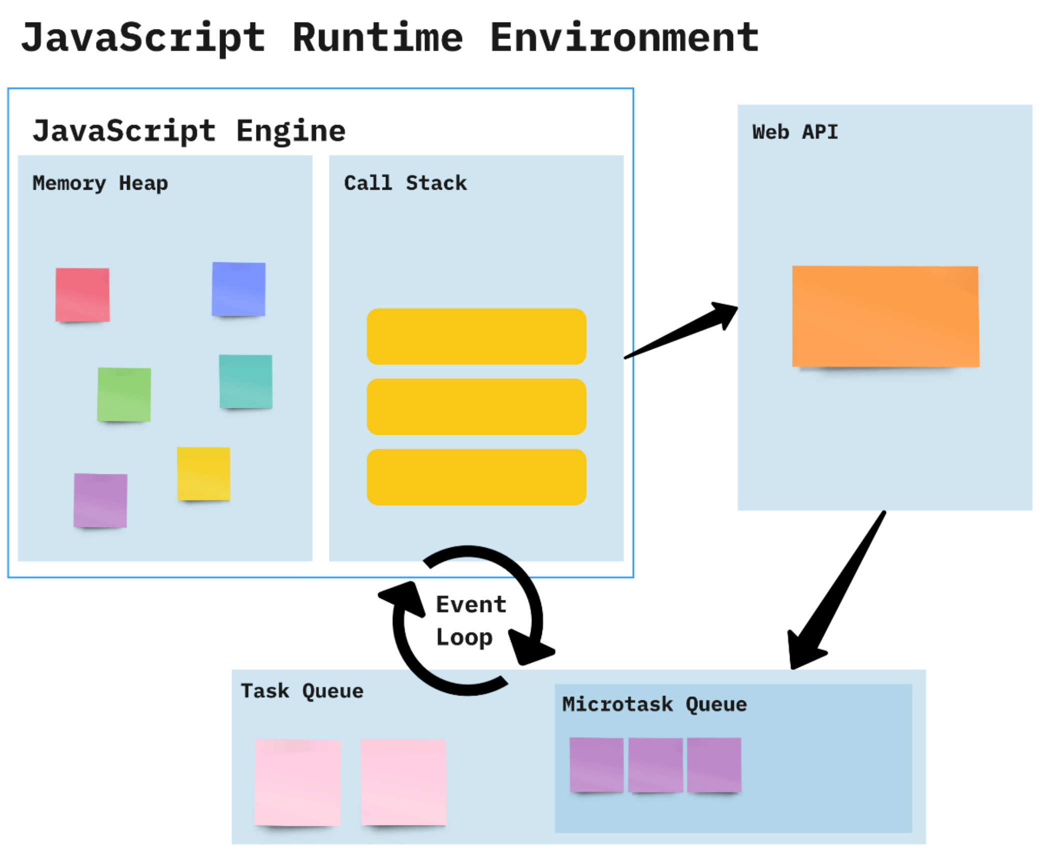
A Web API (Application Programming Interface) is a set of rules and tools that allows different software applications to communicate with each other over the web. It acts as an intermediary, enabling one application to interact with another application's data or functionality using standard web protocols, usually HTTP.

Event loop:

The event loop is a fundamental concept in asynchronous programming, especially in environments like JavaScript. It enables non-blocking operations, allowing code to execute asynchronously while ensuring that tasks are handled in an orderly manner. Here's a closer look at how synchronous and asynchronous operations interact with the event loop:

Synchronous vs. Asynchronous

1. **Synchronous:**
   Synchronous operations are executed sequentially, one after the other. Each operation must complete before the next one begins.
2. **Asynchronous:**
   Asynchronous operations allow code to execute without waiting for previous operations to complete. This is useful for tasks that involve waiting, such as network requests or timers.



How it Works:

- Constantly checks whether or not the call stack is empty
- When the call stack is empty, all queued up Microtasks from Microtask Queue are popped onto the callstack
- If both the call stack and Microtask Queue are empty, the event loop dequeues tasks from the Task Queue and calls them
- Starved event loop

Difference Between Compiler and Interpreter

Compiler	**Interpreter**
Steps of Programming: • Program Creation. • Analysis of language by the compiler and throws	**Steps of Programming:** • Program Creation. • Linking of files or generation of Machine Code is not required by Interpreter.

Compiler	**Interpreter**
errors in case of any incorrect statement. • In case of no error, the Compiler converts the source code to Machine Code. • Linking of various code files into a runnable program. • Finally runs a Program.	• Execution of source statements one by one.
The compiler saves the Machine Language in form of Machine Code on disks.	The Interpreter does not save the Machine Language.
Compiled codes run faster than Interpreter.	Interpreted codes run slower than Compiler.
The compiler generates an output in the form of (.exe).	The interpreter does not generate any output.
Errors are displayed in Compiler after Compiling together at the current time.	Errors are displayed in every single line.

How Many Ways To Insert JS

JavaScript, also known as JS, is one of the scripting (client-side scripting) languages, that is usually used in web development to create modern and interactive web-pages. The term "script" is used to refer to the languages that are not standalone in nature and here it refers to JavaScript which run on the client machine.

1. **internal JS:**

   By using script tag at the bottom of the document

   Syntax:

   ```
    <body>
          <script>
          Console.log(“hello world”);
         </script>
   </body>
   ```

2. **inline JS:**

   we can apply inline js within the element.

   Syntax:

   ```
    <body>
          <button onclick=”alert(‘hello world’)”>click</button>
   </body>
   ```

3. **external JS:**

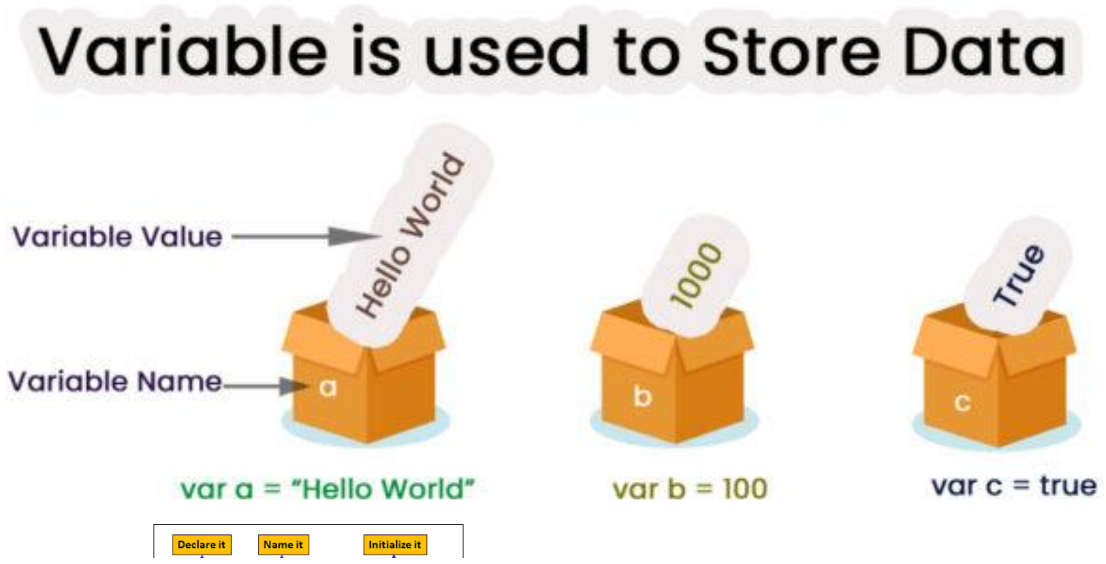
   By creating a js file with .js extention we can insert js file into the html document. That js file is linked in the script tag.

   Syntax:

   ```
   <body>
          <script src=”js file with .js extention”></script>
    </body>
   ```

Variables:

Variables are used to store data in JavaScript. Variables are used to store reusable values. The values of the variables are allocated using the assignment operator(“=”).



JavaScript Variables can be declared in 4 ways:

- **Automatically**
- **Using var**
- **Using let**
- **Using const**

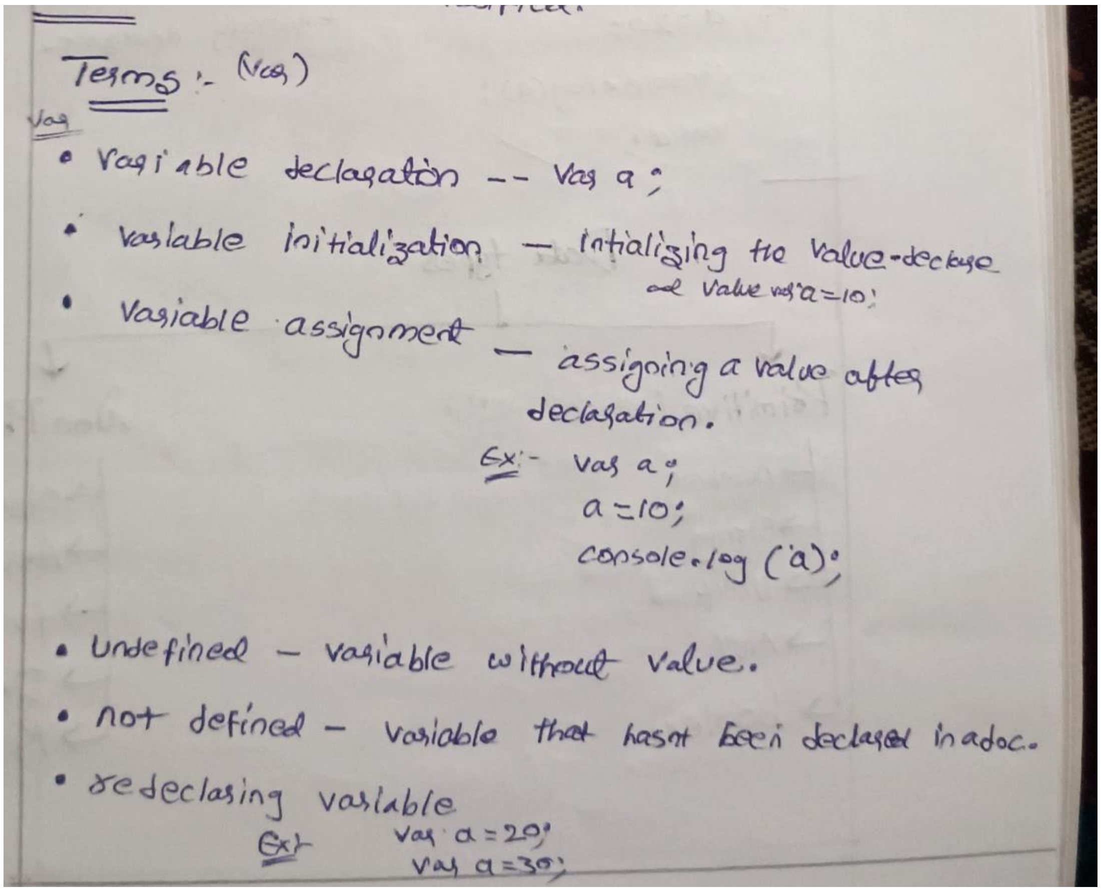
Example:

```
a=10;
Var b=20;
let c=5;
Const d=15;
console.log(a);    //10
console.log(a);   //20
console.log(a);  //5
console.log(a);  //15
```

Rules for Identifiers:

- Names can contain letters, digits, underscores, and dollar signs
- Identifier should not start with number
- Names must begin with a letter or _ or $
- Names are case-sensitive
- Reserved words cannot be used as Identifier.

Terms:



Terms :- (Var)

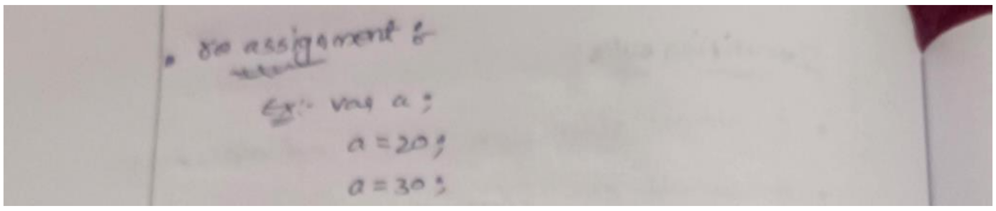
Var

- Variable declaration -- Var a;
- variable initialization — Intializing the value-declare and value var a=10;
- Variable assignment — assigning a value after declaration.

```
Ex:- Var a;
      a=10;
      console.log('a');
```

- Undefined — variable without value.
- not defined — Variable that hasn't been declared in a doc.
- redeclaring variable

```
Ex:-    Var a=20;
        Var a=30;
```



- re assignment :

```
Ex:- Var a;
     a=20;
     a=30;
```

Dynamic typing:

Js is a dynamically typed, meaning you do not have to specify the datatype of the variable when declared. The Data type of the variable is determined automatically in a runtime.

Hoisting

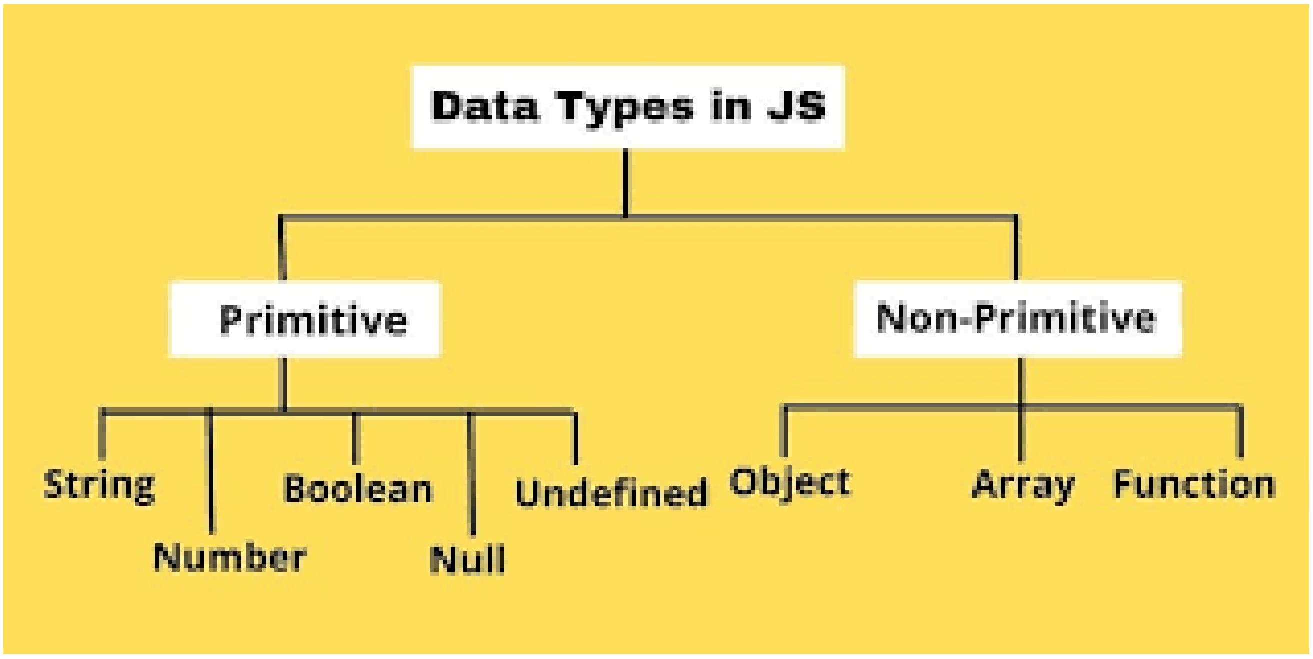
- It is a behaviour where the declaration of the variable and functions are moved to the top even before the execution.
- Only Declaration is hoisted not the Initialization
- Only works in **var** remaining **let** and **const** goes to temporary deadzone

Example:

```
a=20;

Console.lob(a);     //20

var a;
```

Data types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.



Primitive Date Types:-

Primitive data types are the fundamental building blocks used to represent single values.

- Primitive data types which is stored in stack. (call by value and pass by ~~reference~~ value)
- which are immutable. we can access the data but cannot change.

Non Primitive Data types:- (or) Composite data type

- Used to represent multiple values
- non primitive data types are stored in heap.. (call by reference and Pass by reference)
- which are mutable (we can change data).

Primitive

- number :- represents numeric values

  Eg:- var a=100;

- String :- represents Group of characters

  eg:- var b = "Hello"

- Boolean :- represents boolean value either

  true - 1 or false - 0

- null :- represents null. ie. no value at all (intentionally Empty value)
- Undefined :- variable with out value (declared but not assigned value)

non primitive

- array :- represents group of siminasl Elements
- Objects :- represents instance through which we can access members.
- functions :- it is a block of code to perform Particular task & it is reusable.
- date
- reg exp :- represents regular expressions.

Examples :-

primitive

```
var a = 20;  //number
var b = 'hello';  //string
var c = true;  //boolean
var d = False
var e = null
var f = undefined

console.log(c+f);  //NaN
```

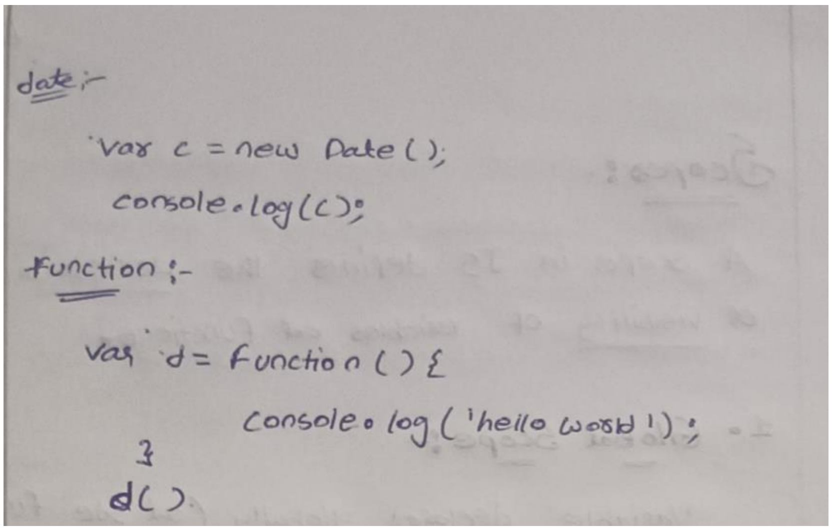
non primitive

arrays (number index)

```
var a = [20, "hello", true];

a[0] = 30;
console.log(a);  //[30, 'hello', true]
Console.log(a[-1]);
```

objects (named index)

```
var b = {
        id : 1201,
        name: 'abhi',
        age: 22
   }
console.log(b.age);  //22
```



date:-

```
var c = new Date();
console.log(c);
```

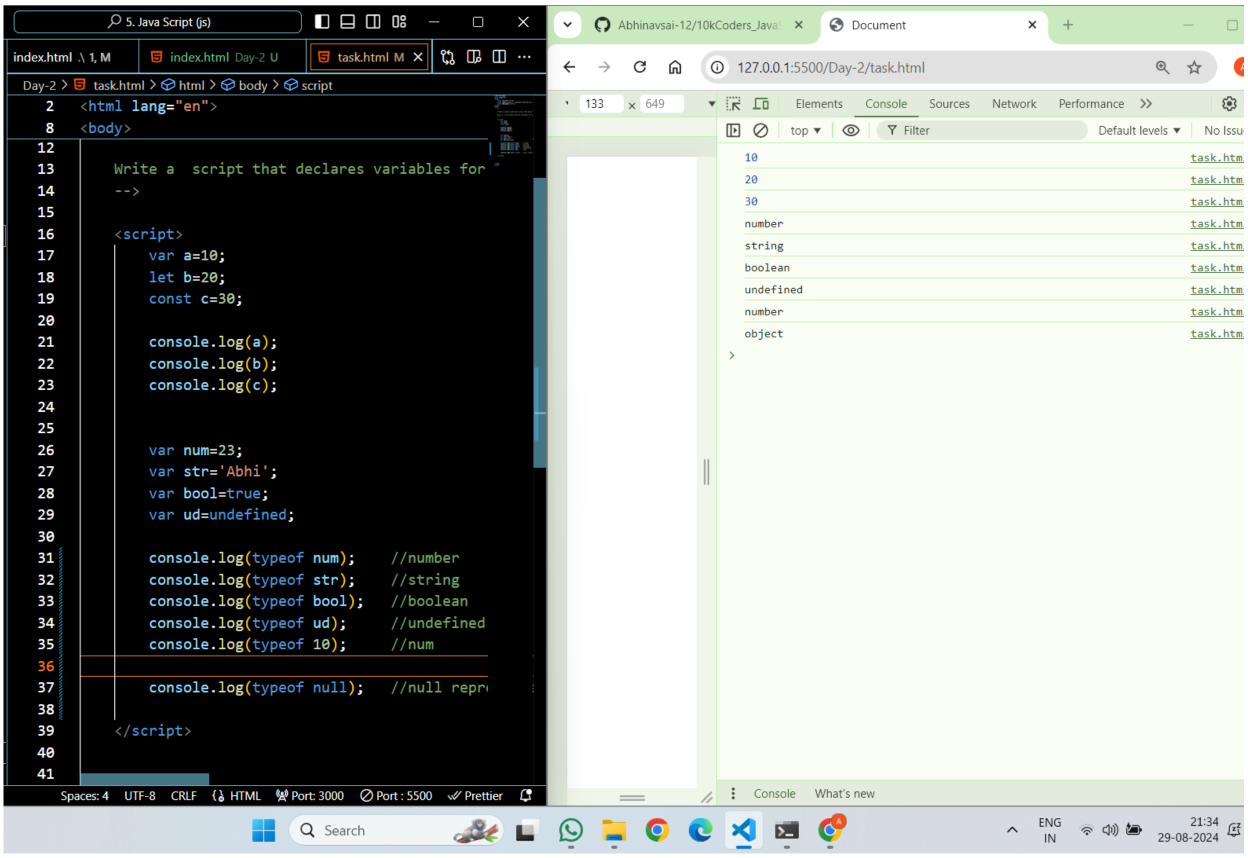
Function :-

```
var d= function (){
        console.log('hello word!');
  }
d()
```

Typeof:

The typeof operator returns the data type of a variable.

The JavaScript typeof operator returns the data type of a variable or expression. It's a unary operator placed before its operand and returns a string indicating the data type, such as "number", "string", "boolean", "object", "undefined", "function", or "symbol".



```
<html lang="en">
<body>

    Write a  script that declares variables for
    -->

    <script>
        var a=10;
        let b=20;
        const c=30;

        console.log(a);
        console.log(b);
        console.log(c);


        var num=23;
        var str='Abhi';
        var bool=true;
        var ud=undefined;

        console.log(typeof num);     //number
        console.log(typeof str);     //string
        console.log(typeof bool);    //boolean
        console.log(typeof ud);      //undefined
        console.log(typeof 10);      //num

        console.log(typeof null);    //null repr

    </script>
```

```
10
20
30
number
string
boolean
undefined
number
object
```

Scopes

A Scope in JS defines the Accessibility or life or Visibility of Variables and Functions.

1. Global Scope:

   Variable declared globally (out side function) have globally. Scope means can be access from any where.

   Var have global Scope and Function Scope.

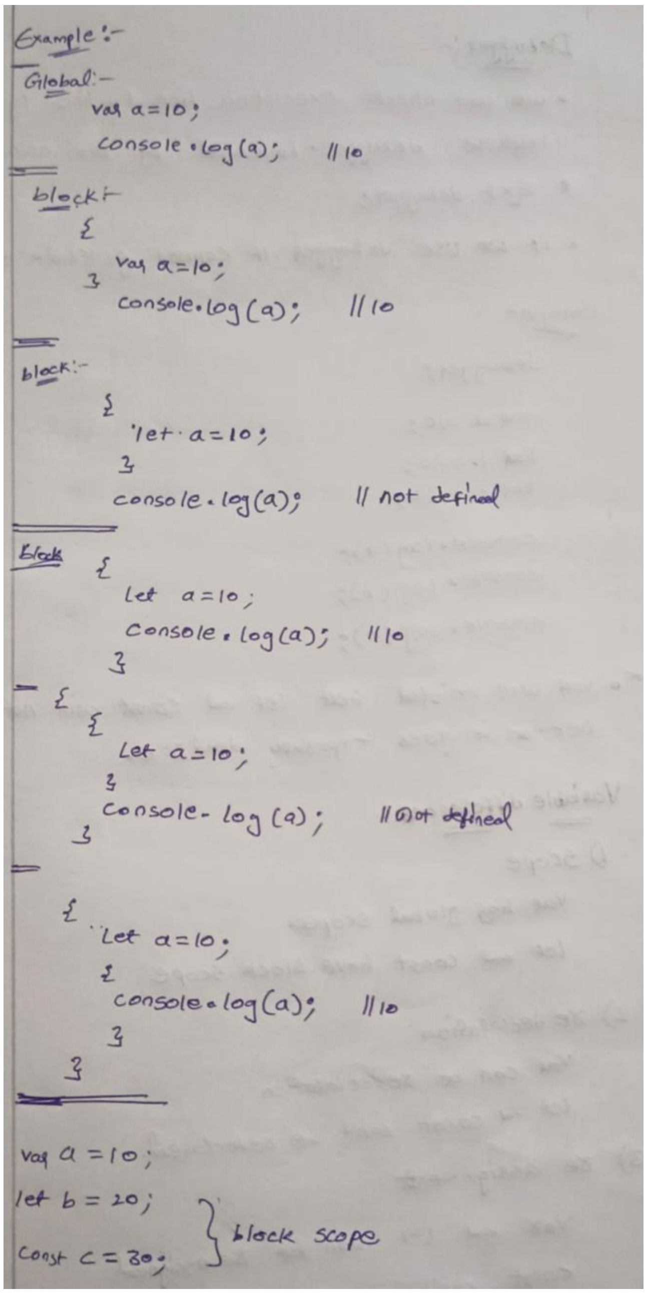
2. Block Scope:

   Variables declared in a block have block scope means that can't be accessed outside of the block.

   Only var have global scope the remaining let and const have block scope.

3. Local Scope:

   Variables declared within the function have local scope. They can only be accessed with in the function.



Example :-

Global:-

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

block:-

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

block:-

```
      {
        let a=10;
      }
      console.log(a);     // not defined
```

block

```
      {
        let  a=10;
        console.log(a);   // 10
      }
```

```
  {
     {
       Let a=10;
     }
     console.log(a);     // not defined
  }
```

```
  {
     Let a=10;
     {
      console.log(a);    // 10
     }
  }
```

```
var a =10;
let b = 20;   } block scope
const c=30;
```

Debugger

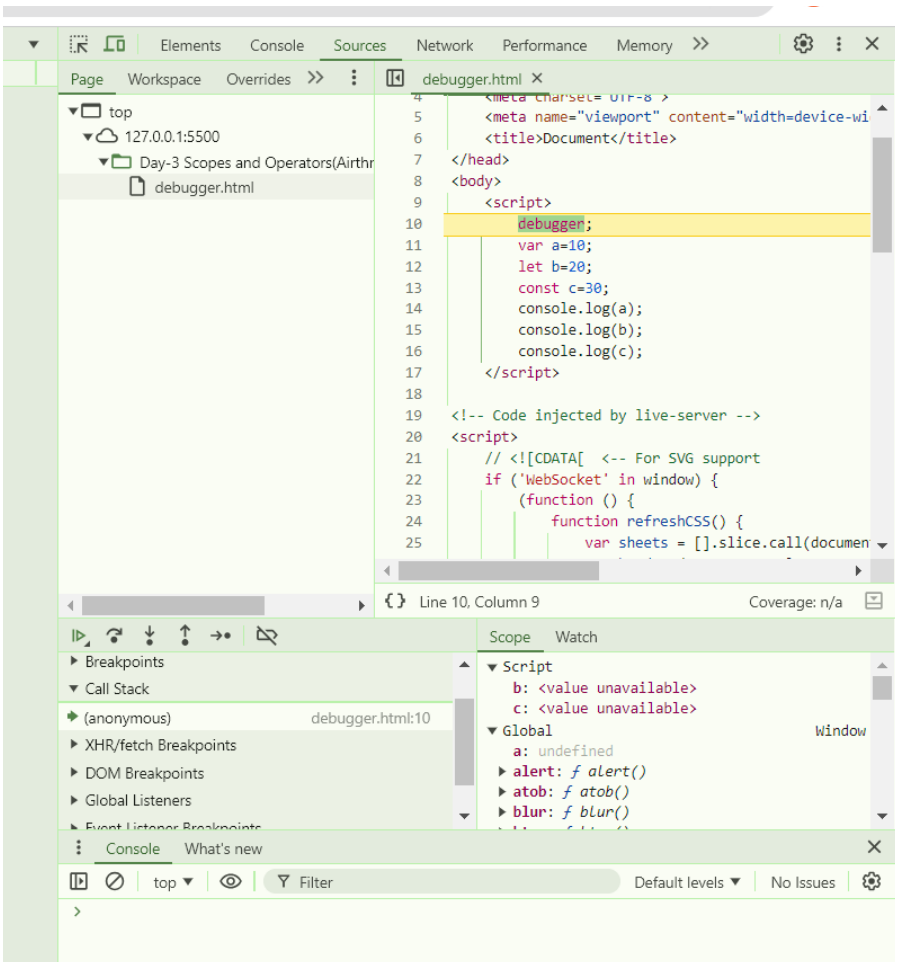
We can check Execution line by line by using keyword debugger followed by semi colon (:)

syntax:

```
debugger;
```

Example:

```
<script>
    debugger;
    var a=10;
    let b=20;
    const c=30;
    console.log(a);
    console.log(b);
    console.log(c);
</script>
```



```
    <meta name="viewport" content="width=device-wi
    <title>Document</title>
</head>
<body>
    <script>
        debugger;
        var a=10;
        let b=20;
        const c=30;
        console.log(a);
        console.log(b);
        console.log(c);
    </script>

<!-- Code injected by live-server -->
<script>
    // <![CDATA[  <-- For SVG support
    if ('WebSocket' in window) {
        (function () {
            function refreshCSS() {
                var sheets = [].slice.call(documen
```

```
(anonymous)    debugger.html:10
```

```
▼ Script
    b: <value unavailable>
    c: <value unavailable>
▼ Global                                Window
    a: undefined
  ▶ alert: ƒ alert()
  ▶ atob: ƒ atob()
  ▶ blur: ƒ blur()
```

Variable Difference:

1) Scope
   var has global scope
   Let and const have block scope

2) Re declaration
   Var can be re declared
   Let and const can’t be re declared

3) Re assignment
   Var and let can be re assigned
   Const can’t be re assigned

Operators

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

(or)

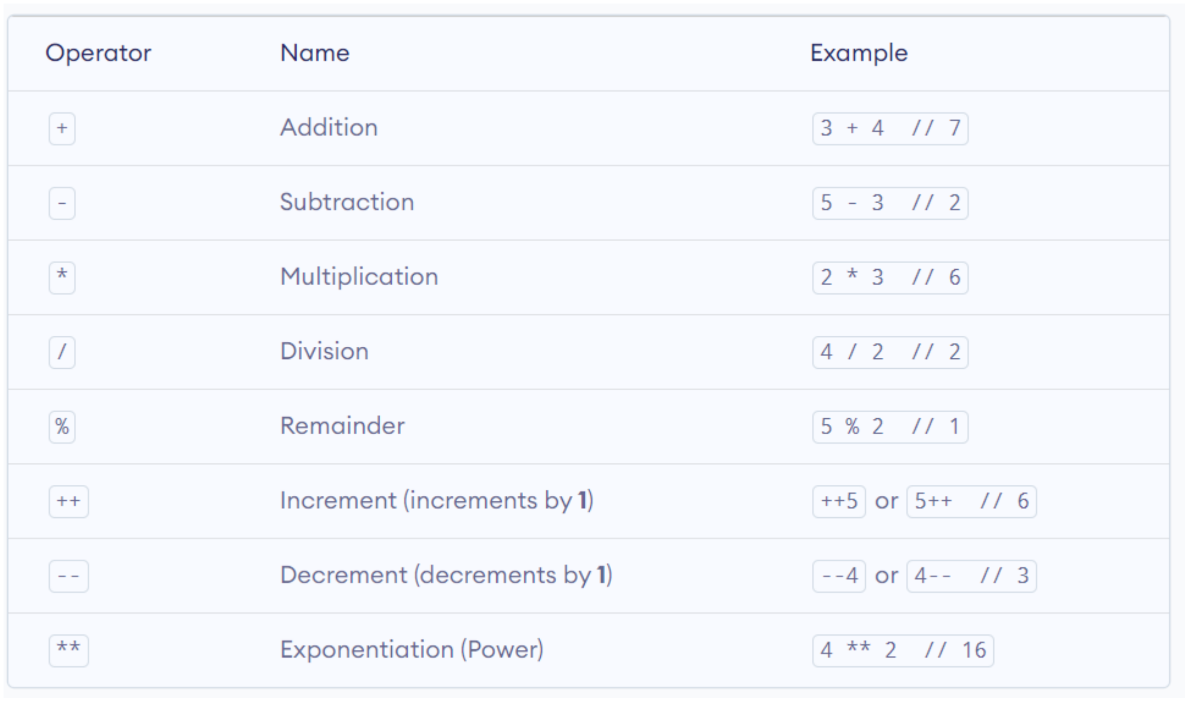
In JavaScript, an **operator** is a symbol that performs an operation on one or more operands, such as variables or values, and returns a result. Let us take a simple expression **4 + 5** is equal to 9. Here 4 and 5 are called **operands**, and ‘+’ is called the **operator**.

Types:

1. Airthmetic Operators
2. Assignment Operator
3. Comparision Operator
4. Logical Operator
5. Ternary Operator
6. Bitwise Oberator
7. String Operator
8. Typeof Operator

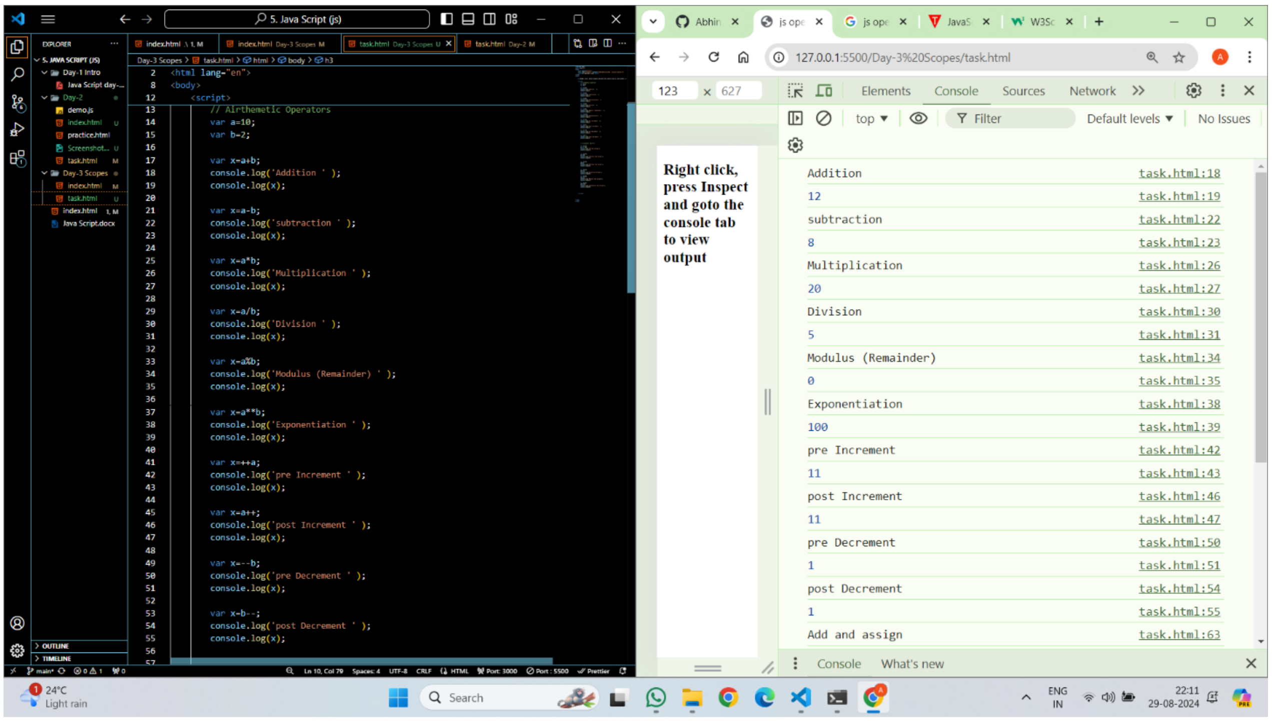
Arithmetic operators

Arithmetic operators are used to perform **arithmetic operations** between variables or values.



Operator	Name	Example
+	Addition	3 + 4 // 7
-	Subtraction	5 - 3 // 2
*	Multiplication	2 * 3 // 6
/	Division	4 / 2 // 2
%	Remainder	5 % 2 // 1
++	Increment (increments by 1)	++5 or 5++ // 6
--	Decrement (decrements by 1)	--4 or 4-- // 3
**	Exponentiation (Power)	4 ** 2 // 16

Example:



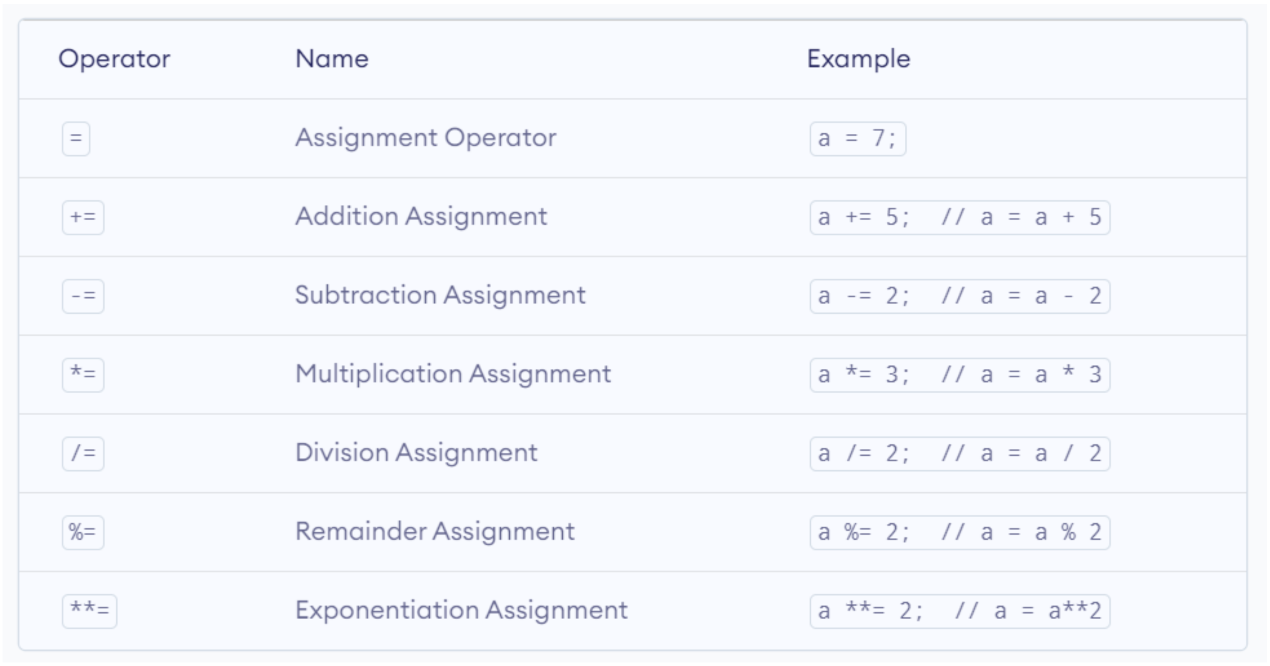
```
<html lang="en">
<body>
    <script>
        // Airthemetic Operators
        var a=10;
        var b=2;

        var x=a+b;
        console.log('Addition ' );
        console.log(x);

        var x=a-b;
        console.log('subtraction ' );
        console.log(x);

        var x=a*b;
        console.log('Multiplication ' );
        console.log(x);

        var x=a/b;
        console.log('Division ' );
        console.log(x);

        var x=a%b;
        console.log('Modulus (Remainder) ' );
        console.log(x);

        var x=a**b;
        console.log('Exponentiation ' );
        console.log(x);

        var x=++a;
        console.log('pre Increment ' );
        console.log(x);

        var x=a++;
        console.log('post Increment ' );
        console.log(x);

        var x=--b;
        console.log('pre Decrement ' );
        console.log(x);

        var x=b--;
        console.log('post Decrement ' );
        console.log(x);
```

Right click, press Inspect and goto the console tab to view output

```
Addition                task.html:18
12                      task.html:19
subtraction             task.html:22
8                       task.html:23
Multiplication          task.html:26
20                      task.html:27
Division                task.html:30
5                       task.html:31
Modulus (Remainder)     task.html:34
0                       task.html:35
Exponentiation          task.html:38
100                     task.html:39
pre Increment           task.html:42
11                      task.html:43
post Increment          task.html:46
11                      task.html:47
pre Decrement           task.html:50
1                       task.html:51
post Decrement          task.html:54
1                       task.html:55
Add and assign          task.html:63
```

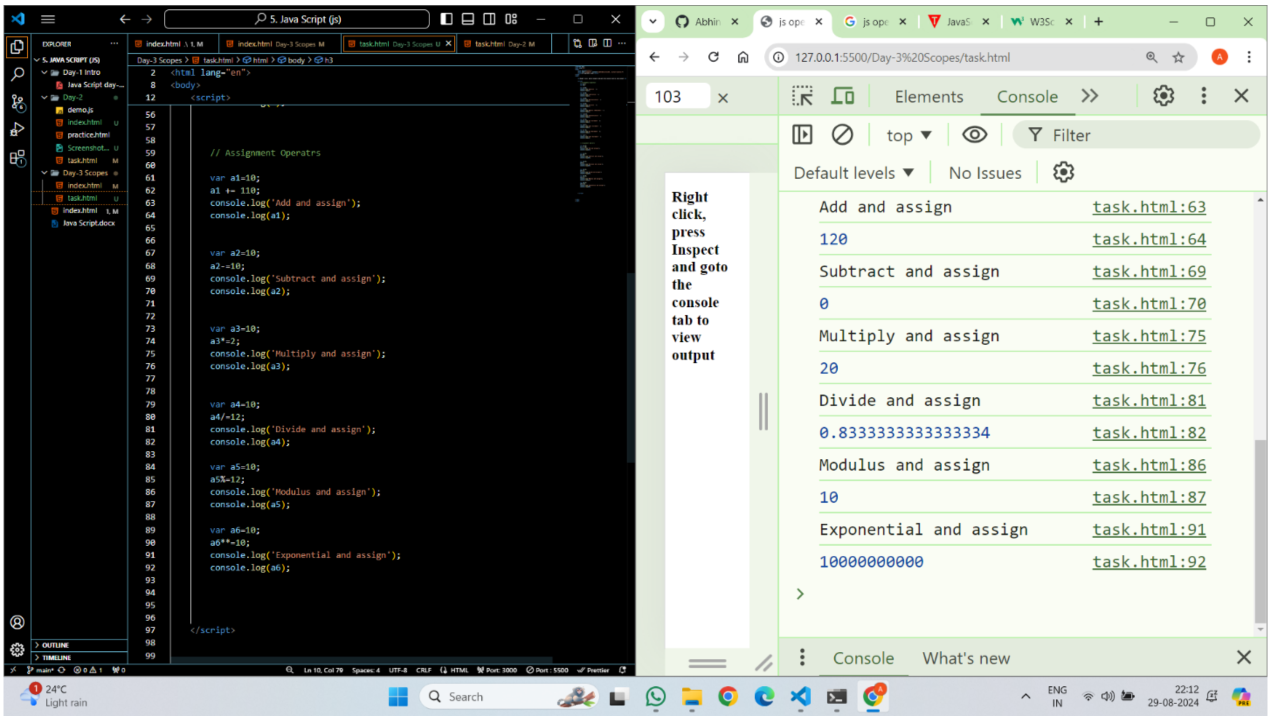
Assignment Operators:

We use assignment operators to **assign** values to variables.



Operator	Name	Example
=	Assignment Operator	a = 7;
+=	Addition Assignment	a += 5; // a = a + 5
-=	Subtraction Assignment	a -= 2; // a = a - 2
*=	Multiplication Assignment	a *= 3; // a = a * 3
/=	Division Assignment	a /= 2; // a = a / 2
%=	Remainder Assignment	a %= 2; // a = a % 2
**=	Exponentiation Assignment	a **= 2; // a = a**2

Example:



```
<html lang="en">
<body>
    <script>

        // Assignment Operatrs

        var a1=10;
        a1 += 110;
        console.log('Add and assign');
        console.log(a1);


        var a2=10;
        a2-=10;
        console.log('Subtract and assign');
        console.log(a2);


        var a3=10;
        a3*=2;
        console.log('Multiply and assign');
        console.log(a3);


        var a4=10;
        a4/=12;
        console.log('Divide and assign');
        console.log(a4);

        var a5=10;
        a5%=12;
        console.log('Modulus and assign');
        console.log(a5);

        var a6=10;
        a6**=10;
        console.log('Exponential and assign');
        console.log(a6);



    </script>
```

Right click, press Inspect and goto the console tab to view output

```
Add and assign                task.html:63
120                           task.html:64
Subtract and assign           task.html:69
0                             task.html:70
Multiply and assign           task.html:75
20                            task.html:76
Divide and assign             task.html:81
0.8333333333333334            task.html:82
Modulus and assign            task.html:86
10                            task.html:87
Exponential and assign        task.html:91
10000000000                   task.html:92
```