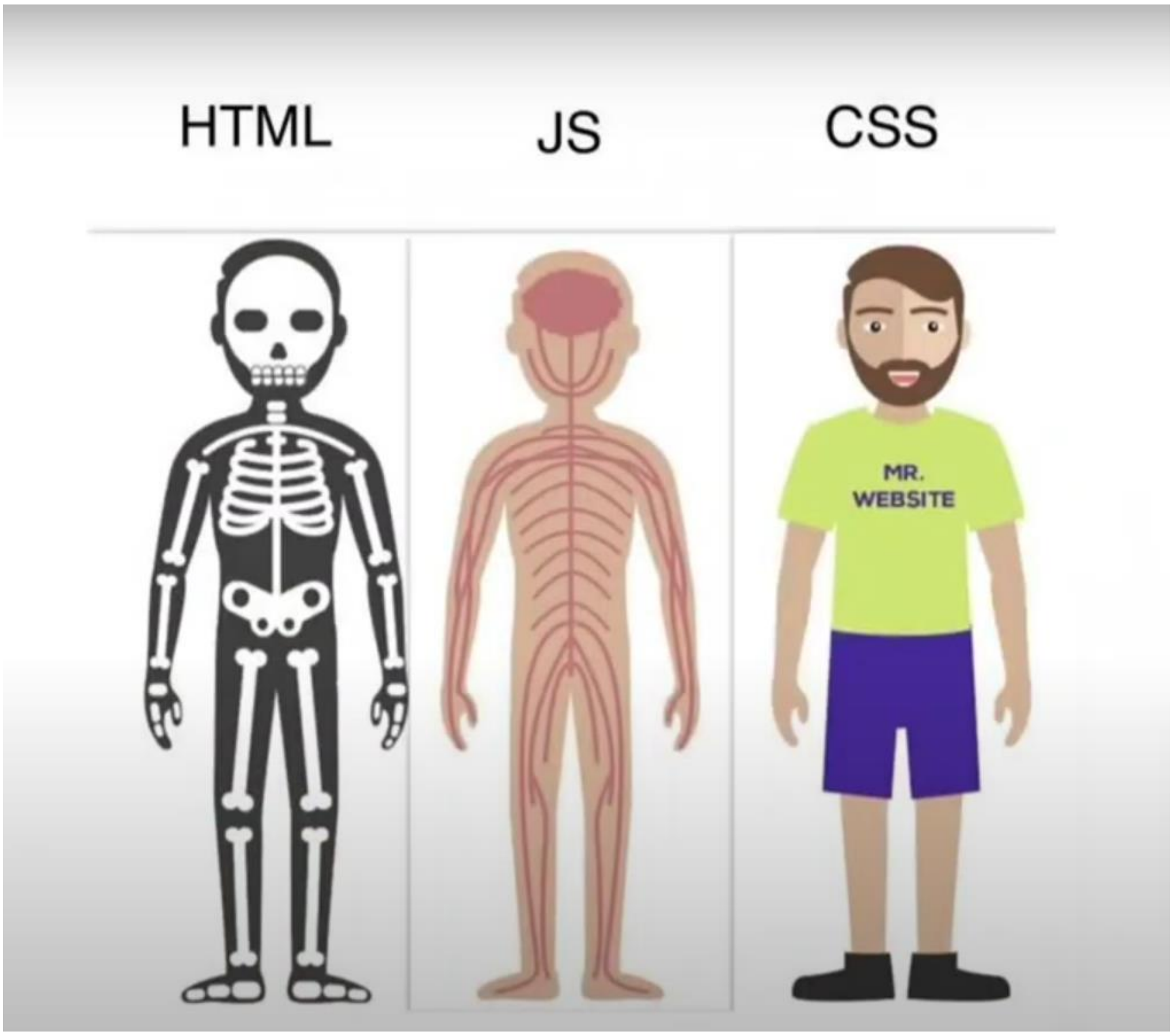
Java Script

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.



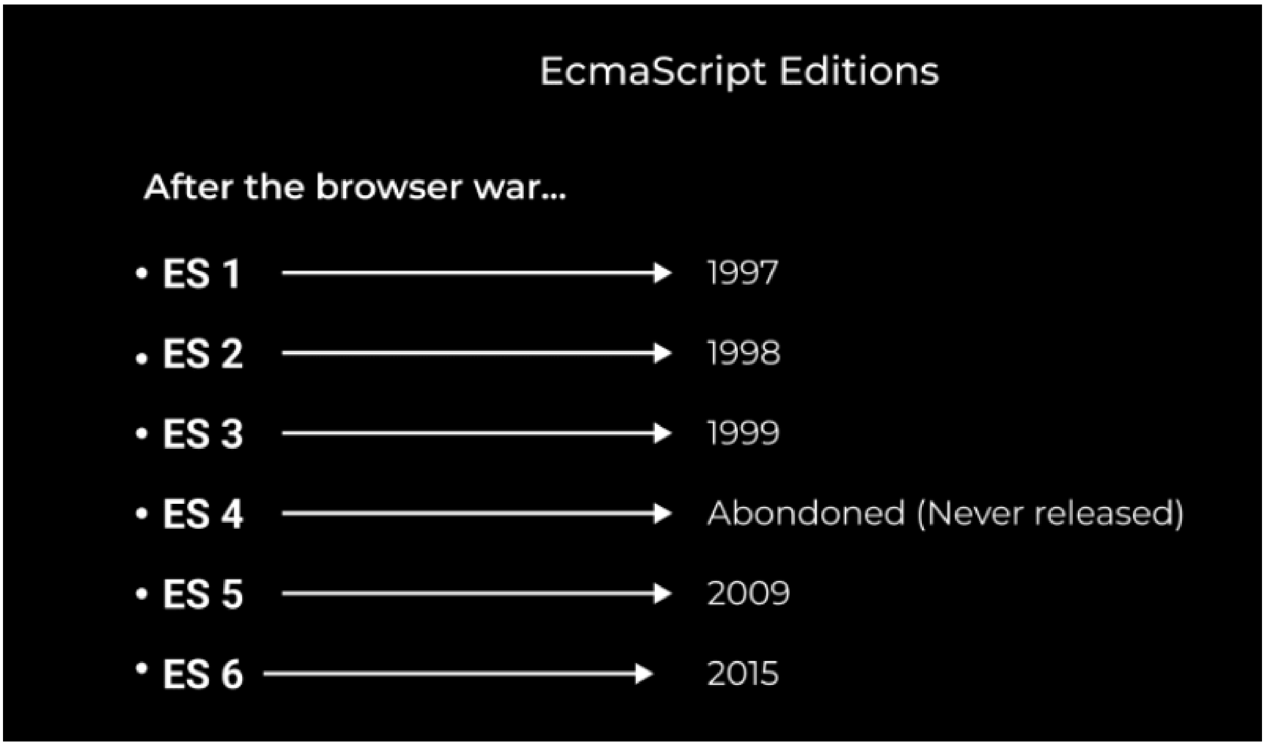
2. History of java script?

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...

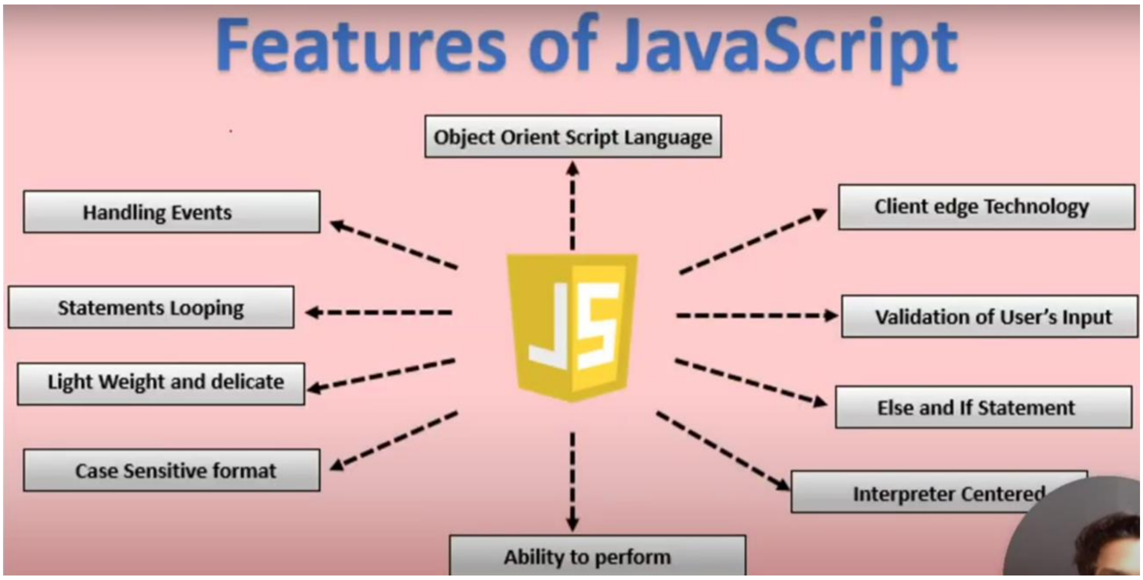
- ES 1 → 1997
- ES 2 → 1998
- ES 3 → 1999
- ES 4 → Abondoned (Never released)
- ES 5 → 2009
- ES 6 → 2015

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.



Features of JavaScript

Object Orient Script Language

Handling Events

Client edge Technology

Statements Looping

Validation of User's Input

Light Weight and delicate

Else and If Statement

Case Sensitive format

Interpreter Centered

Ability to perform

- **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**

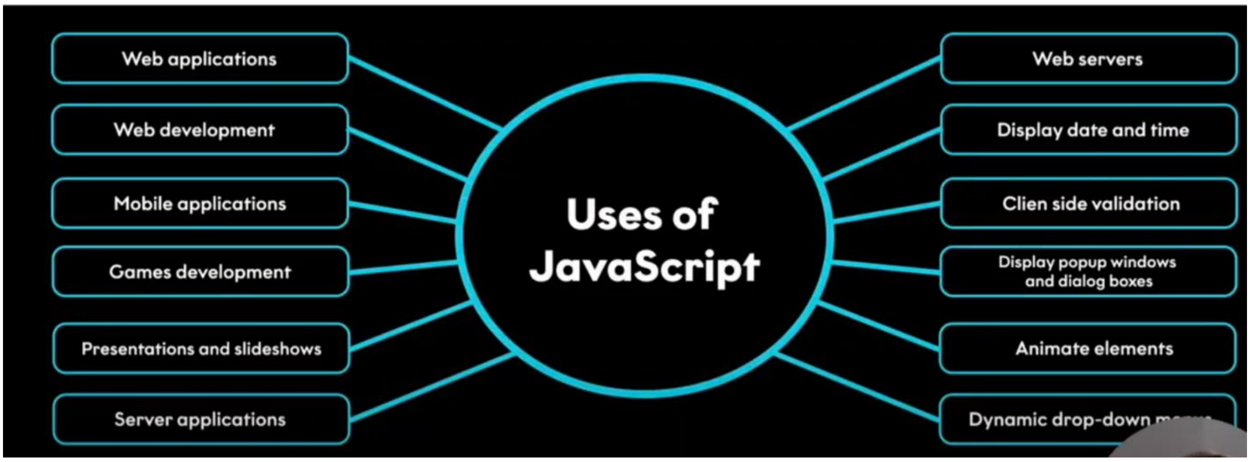
  Synchronous means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

- **Obeject Oriented**

It provides an overview of the basic concepts of OOP.

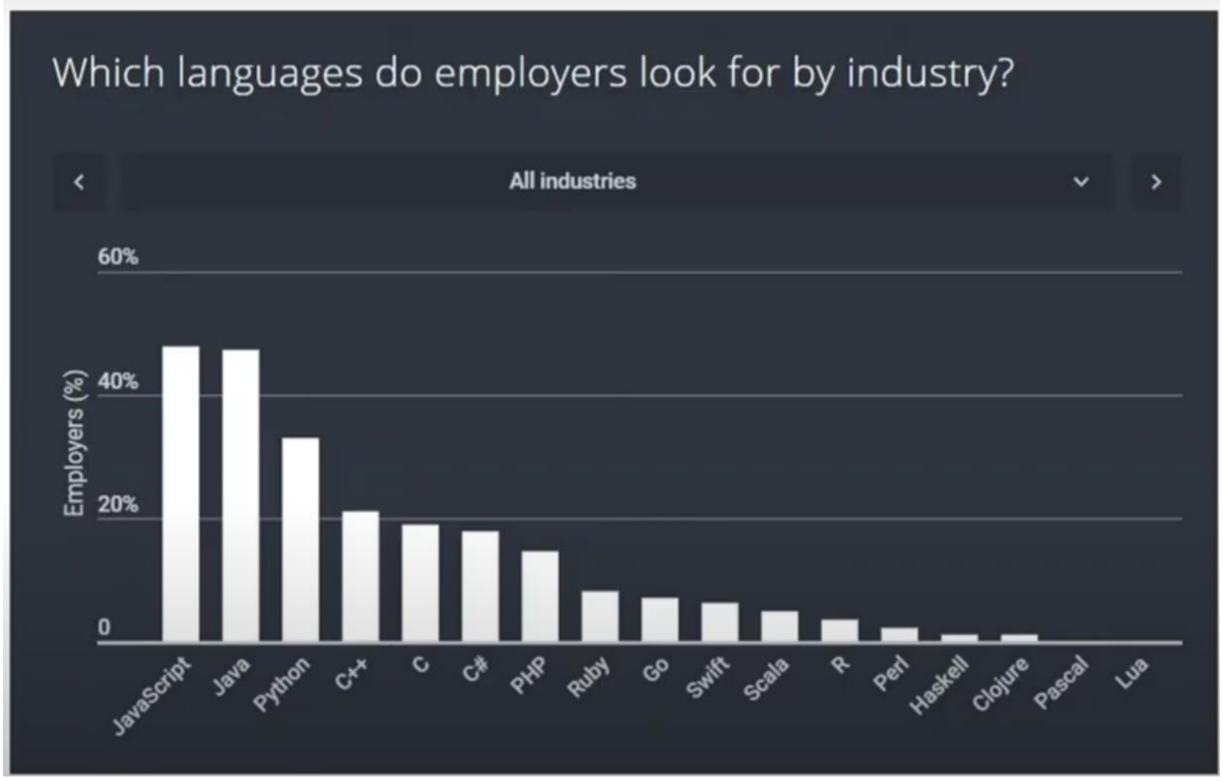
➢ Scripting

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.



Uses of JavaScript

- Web applications
- Web development
- Mobile applications
- Games development
- Presentations and slideshows
- Server applications
- Web servers
- Display date and time
- Clien side validation
- Display popup windows and dialog boxes
- Animate elements
- Dynamic drop-down m



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.

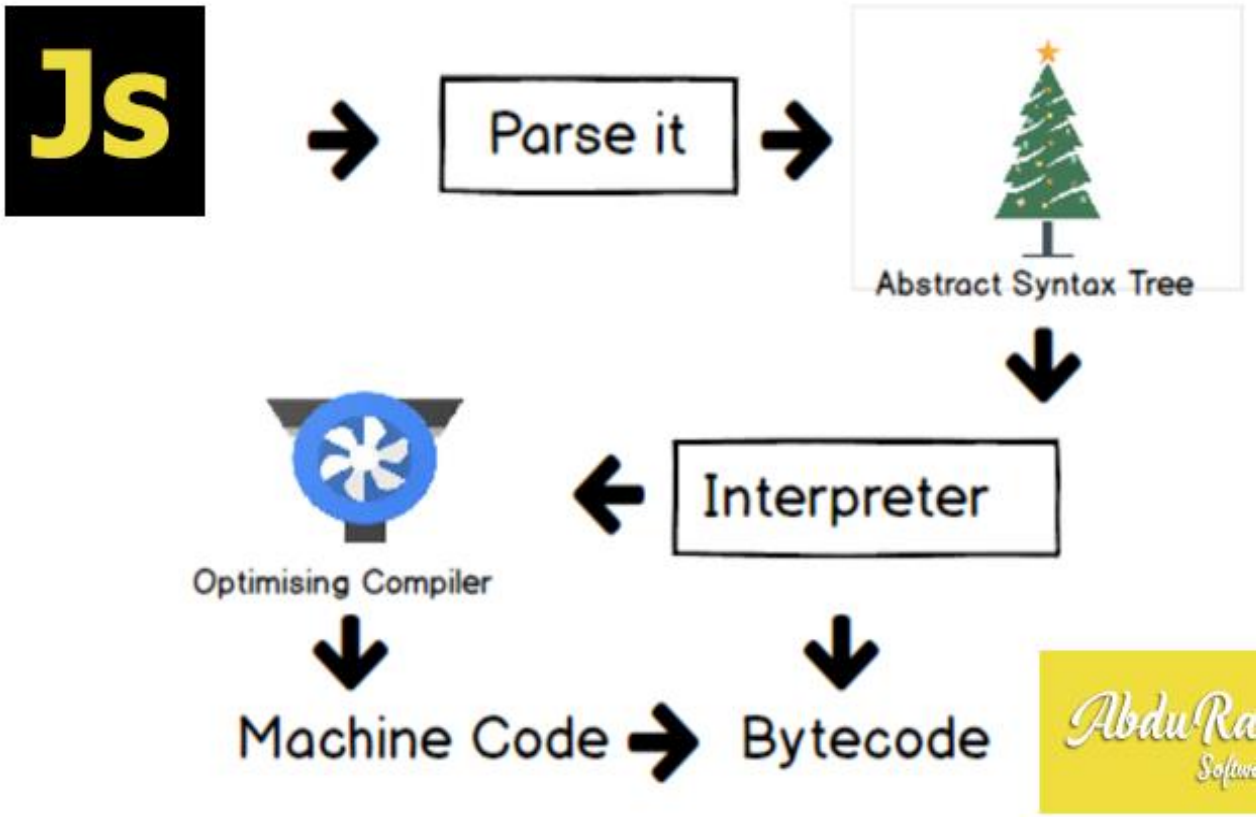
The team of developers that created the vanilla JavaScript is continuously working on it to improve it and make it more useful for the web-developers.

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:

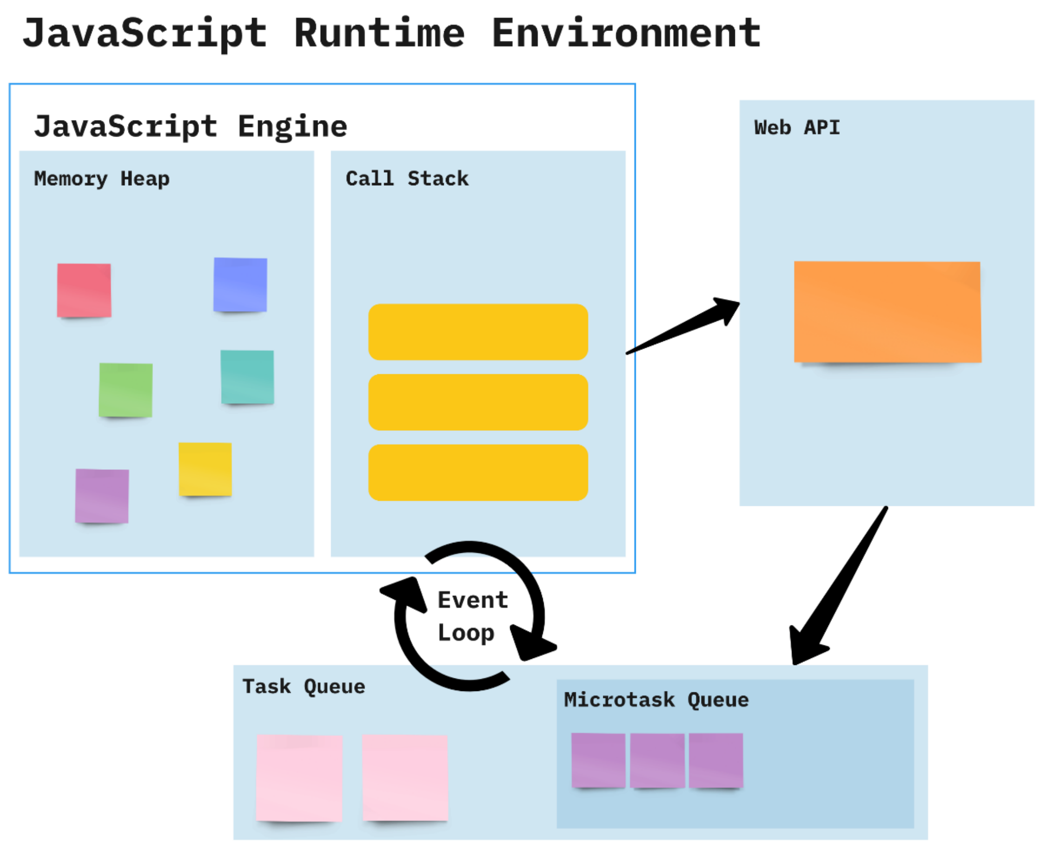
A Web API (Application Programming Interface) is a set of rules and tools that allows different software applications to communicate with each other over the web. It acts as an intermediary, enabling one application to interact with another application's data or functionality using standard web protocols,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

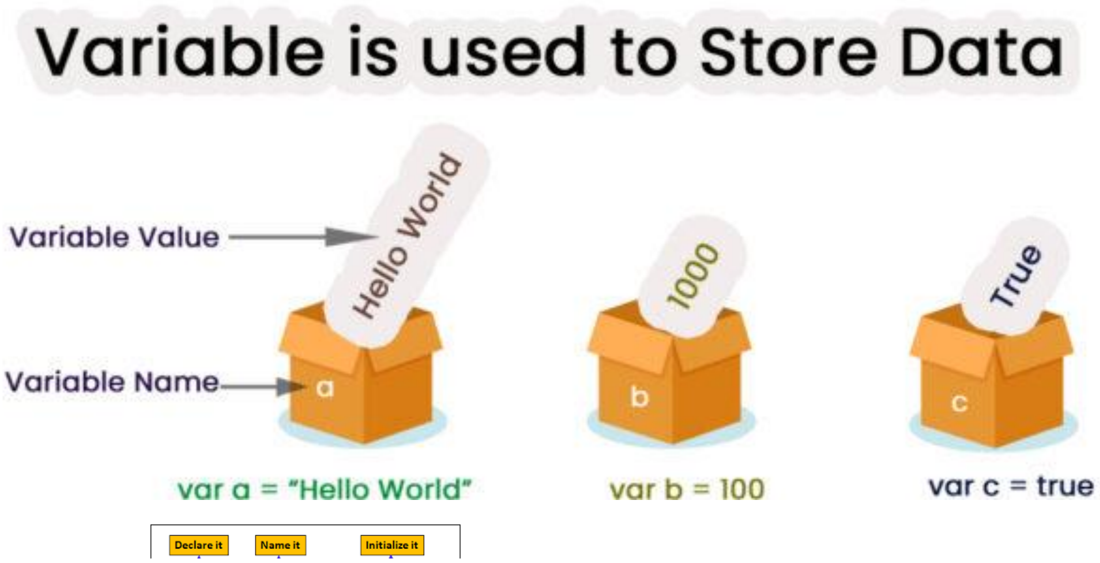
   By creating a js file with .js extention we can insert js file into the html document. That js file is linked in the script tag.

   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**

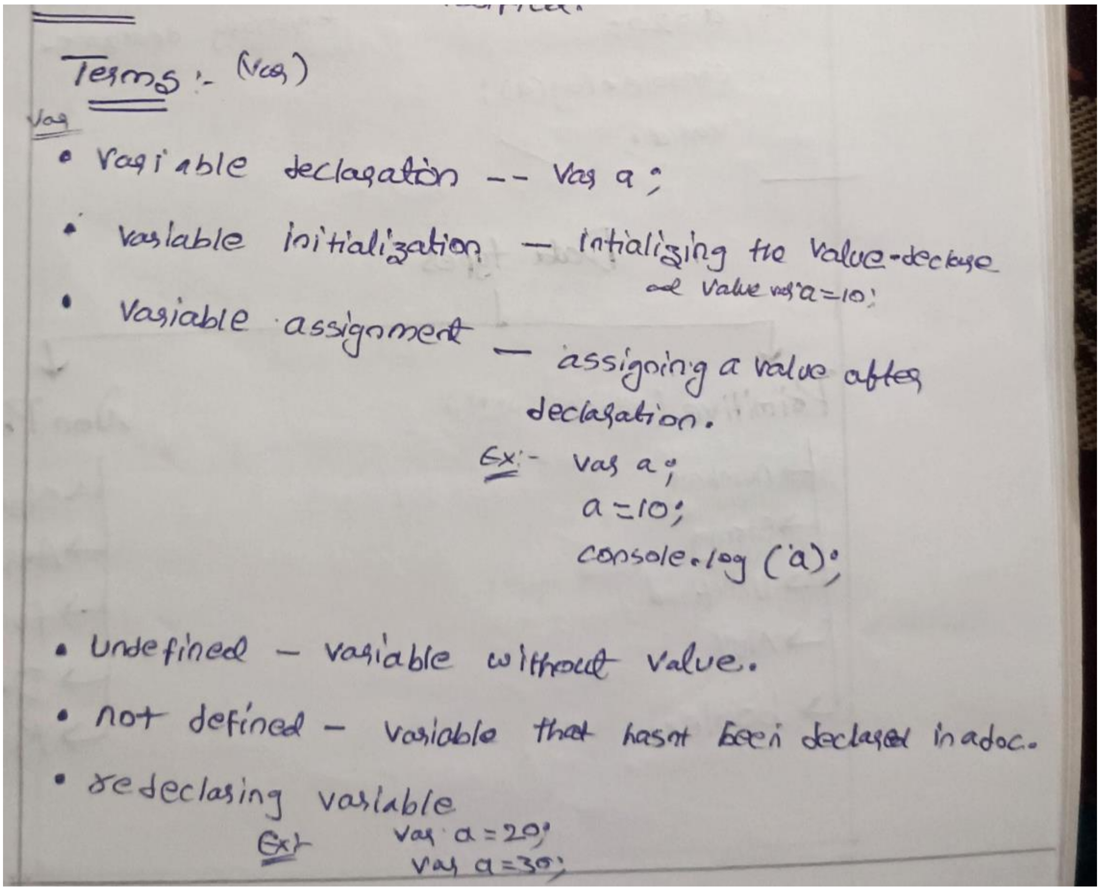
Example:

```
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)

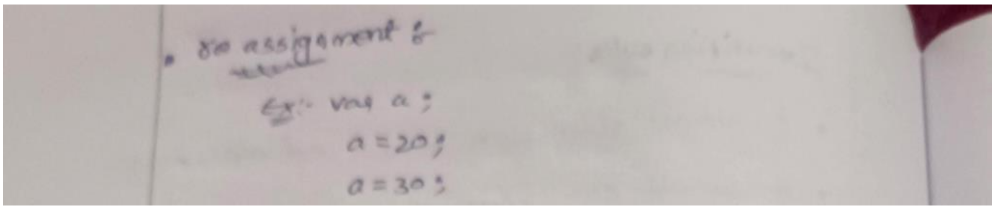
var

- 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 hasnt 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

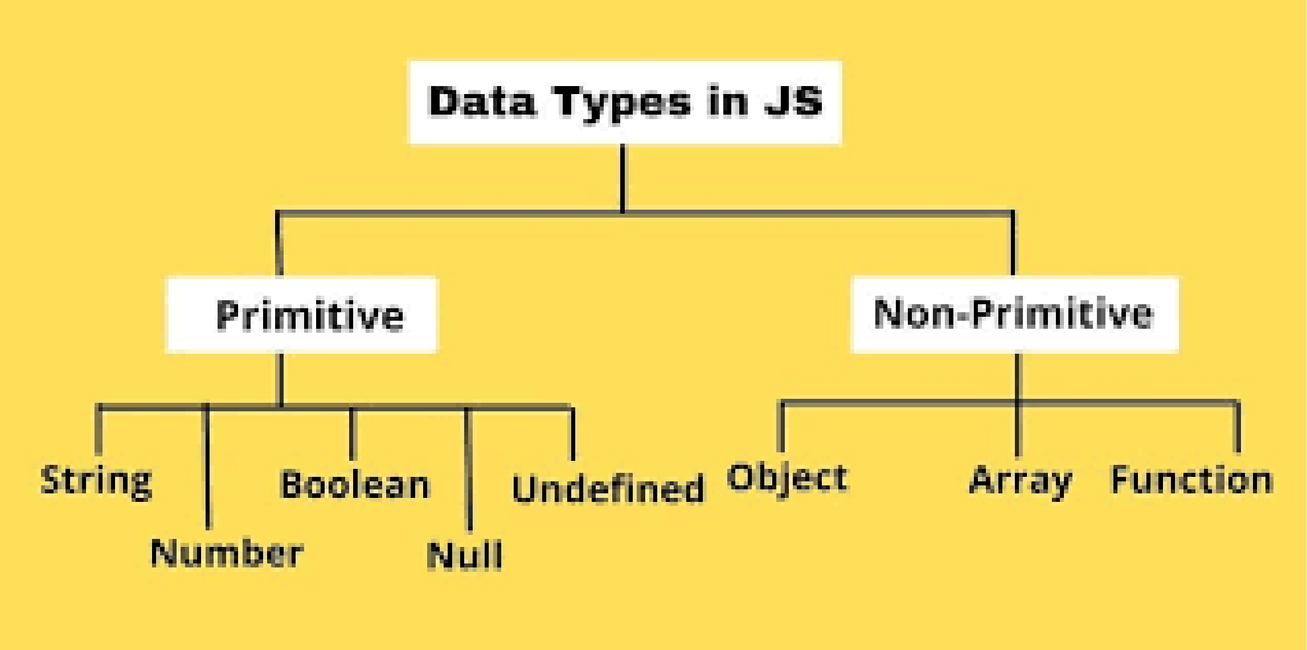
- It is a behaviour where the declaration of the variable and functions are moved to the top even before the execution.
- Only Declaration is hoisted not the Initialization
- Only works in **var** remaining **let** and **const** goes to temporary deadzone

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)
- Un defined :- variable with out value (declared but not assigned value)

non primitive

- array :- represents group of siminal Elements
- Objects :- represents instance through which we can access members.
- functions :- it is a block of code to perform Particular ta ~~a~~ 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
```

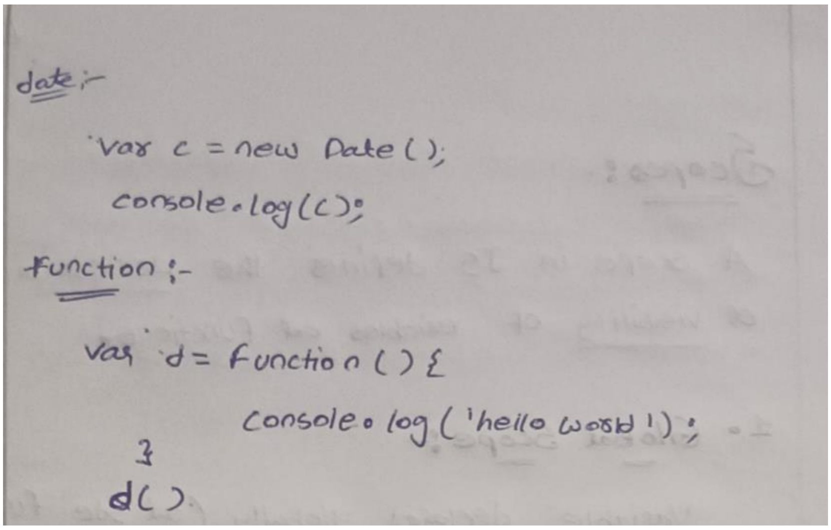
non primitive

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);
```

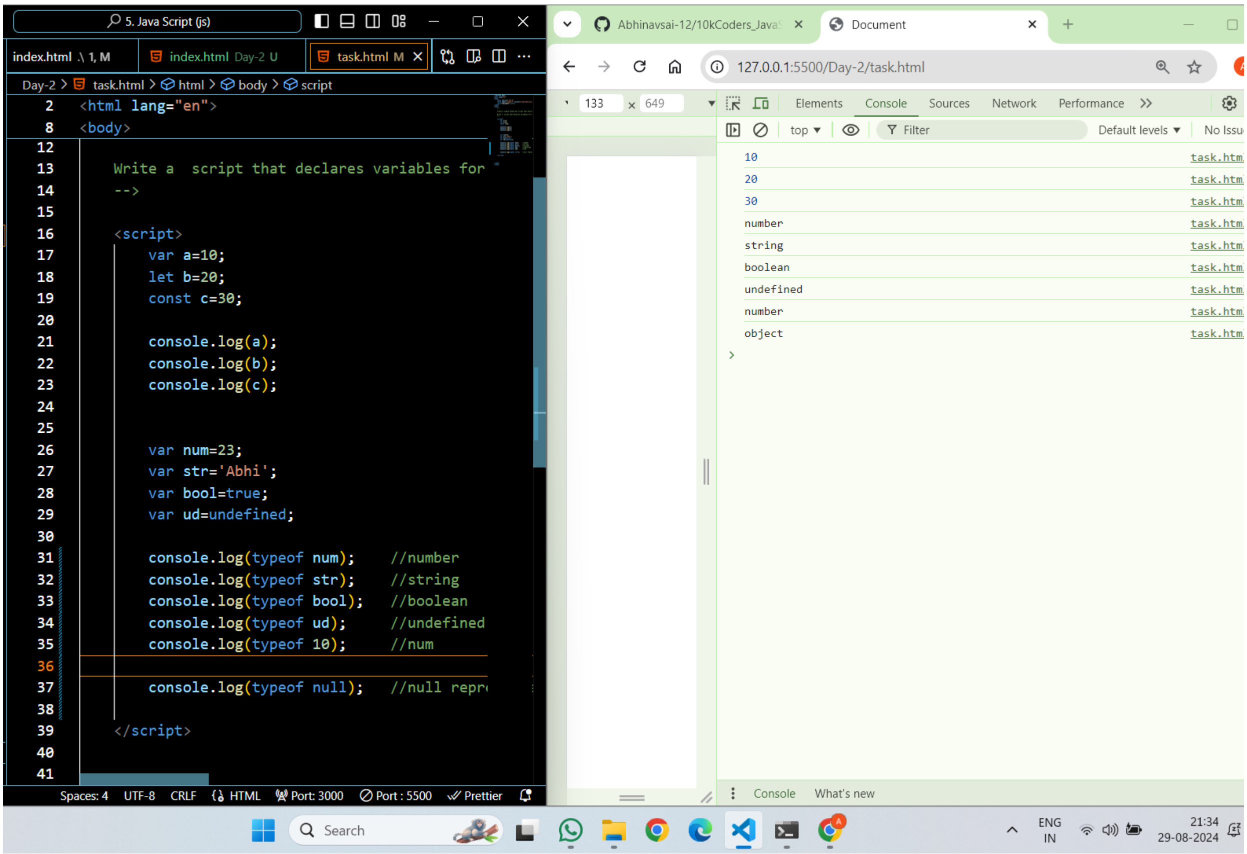
Function :-

```
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.

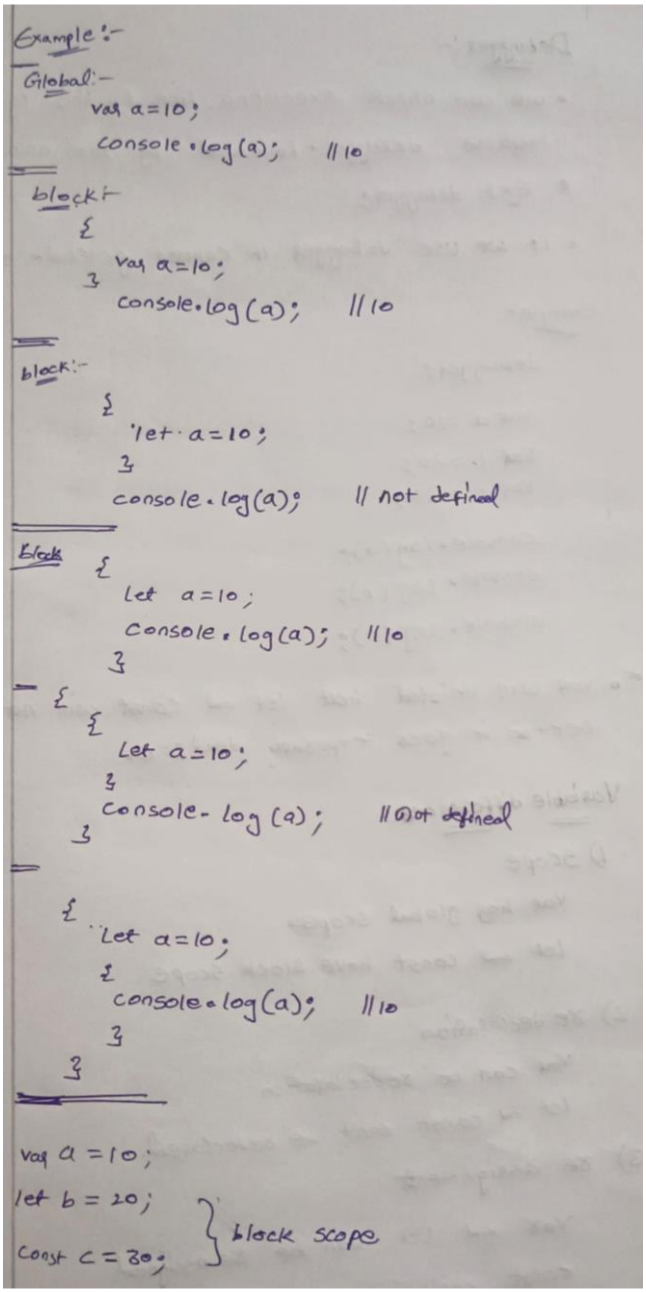
2. Block Scope:

   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

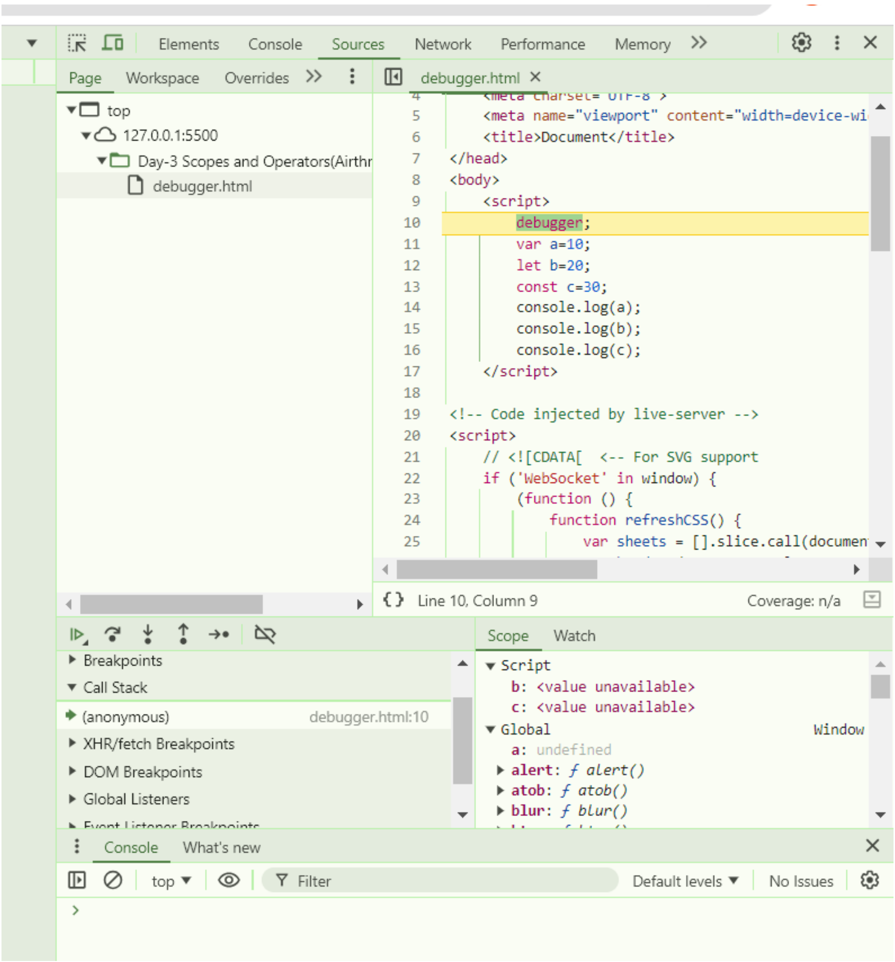
We can check Execution line by line by using keyword debugger followed by semi colon (:)

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 charset="UTF-8">
    <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
```

Line 10, Column 9   Coverage: n/a

(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)

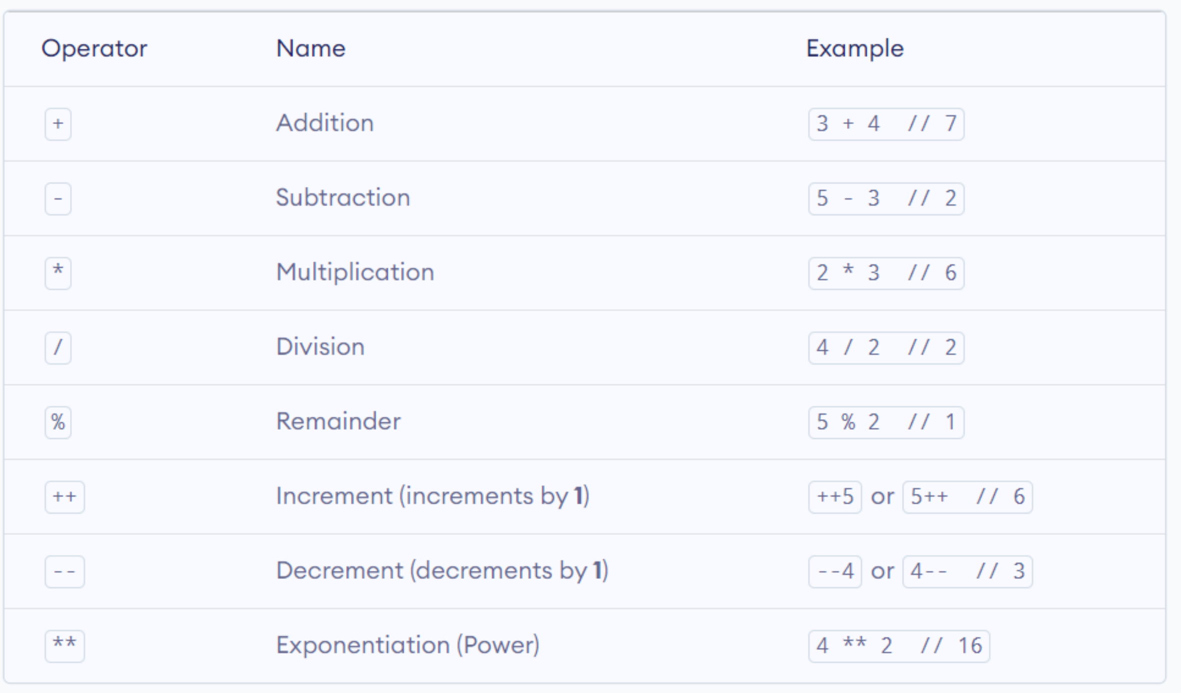
In JavaScript, an **operator** is a symbol that performs an operation on one or more operands, such as variables or values, and returns a result. Let us take a simple expression **4 + 5** is equal to 9. Here 4 and 5 are called **operands**, and '+' is called the **operator**.

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

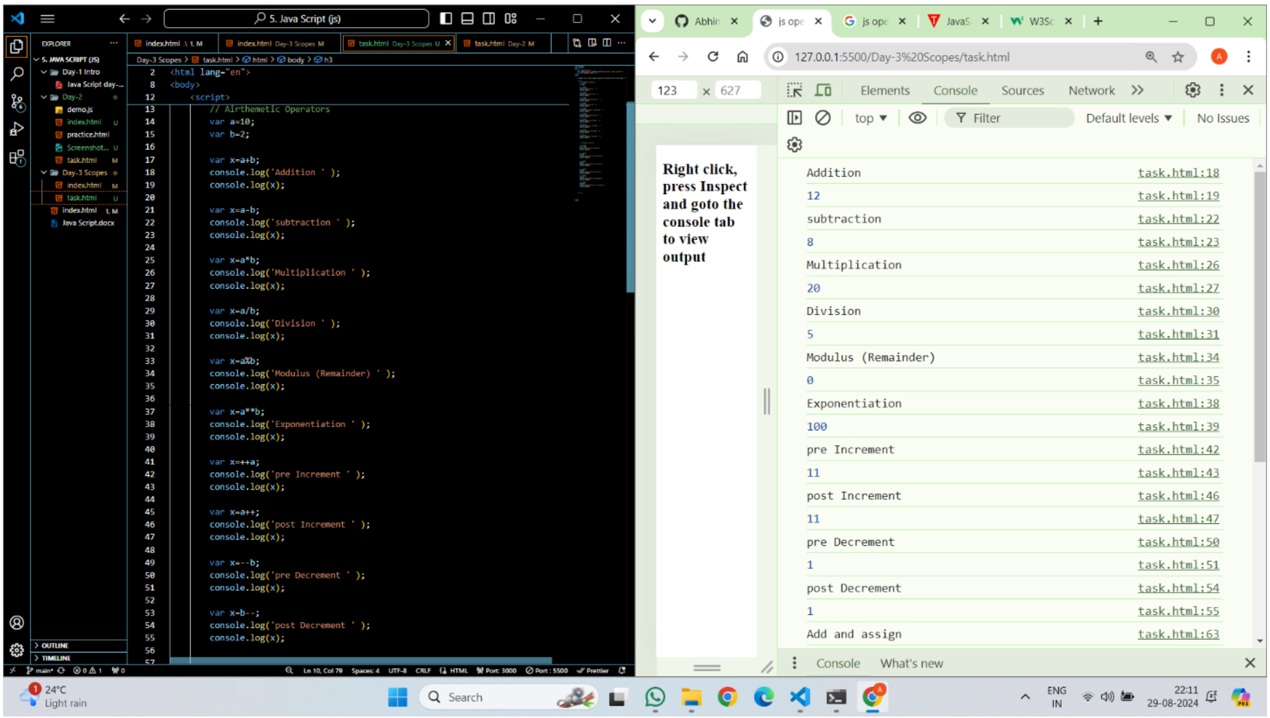
Arithmetic operators

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:



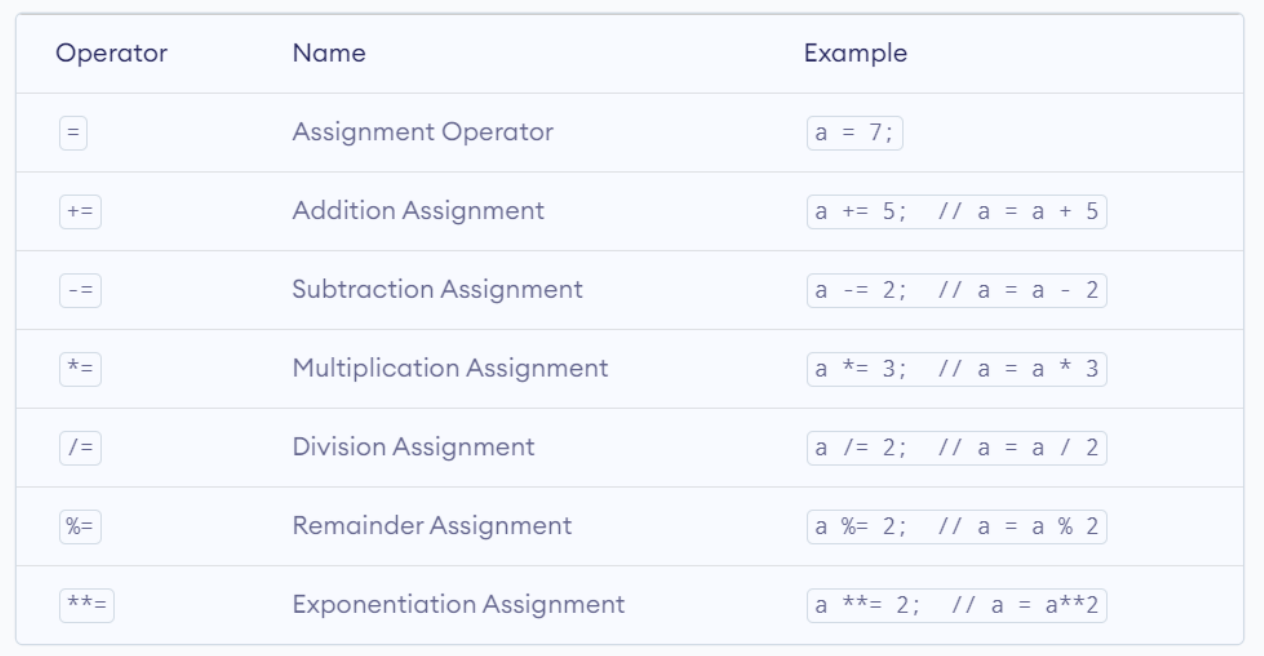
```
<html lang="en">
<body>
    <script>
        // Airthemetic Operators
        var a=10;
        var b=2;

        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
```

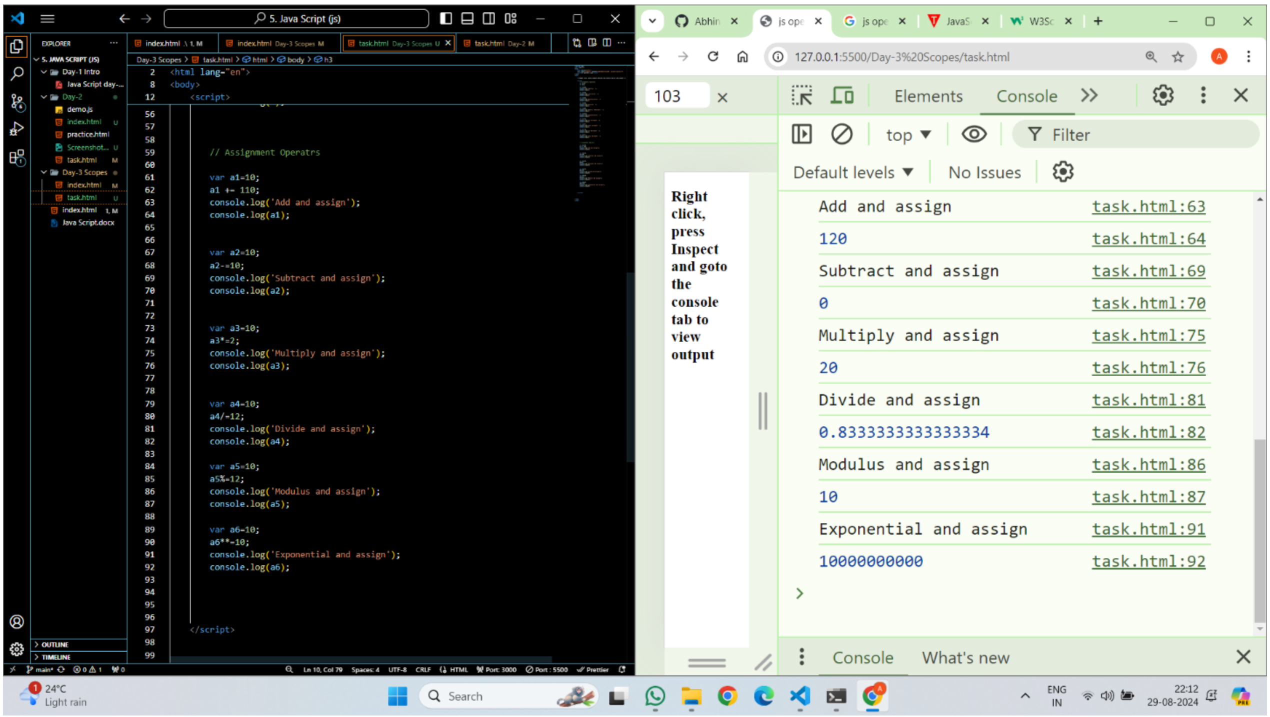
Assignment Operators:

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:



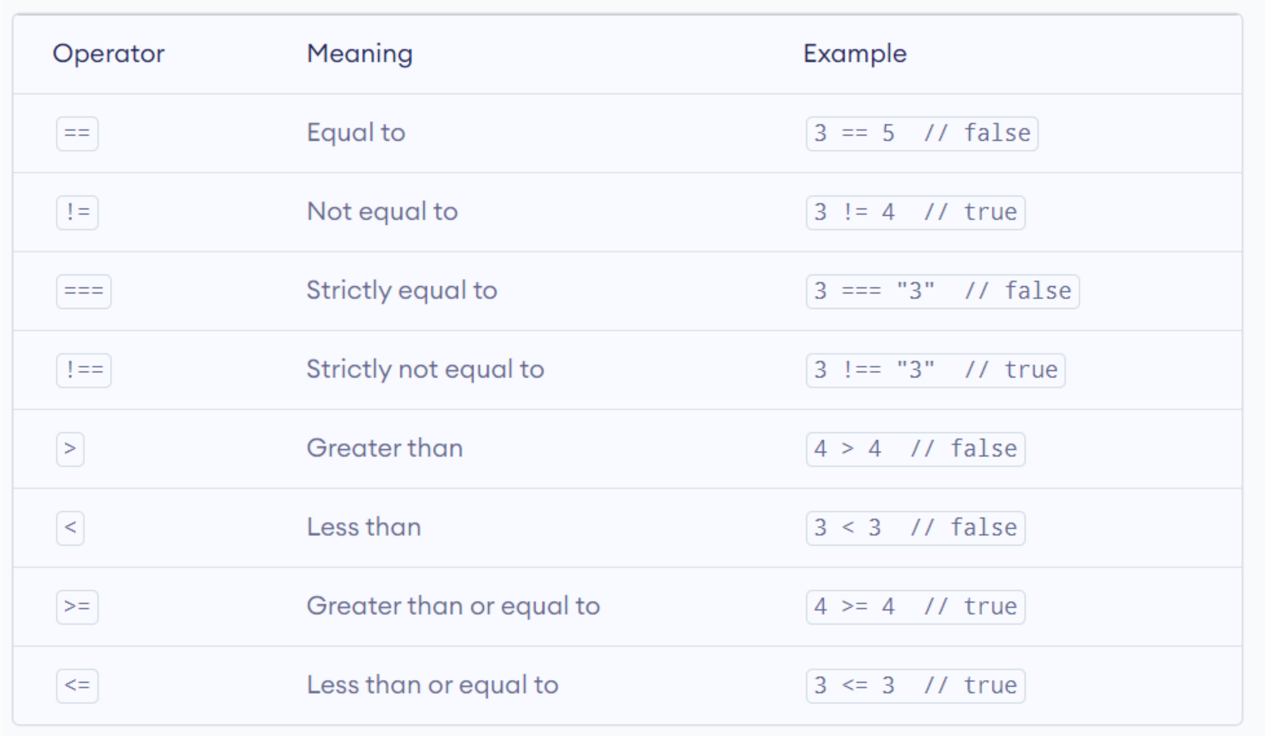
```
<html lang="en">
<body>
    <script>

        // 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

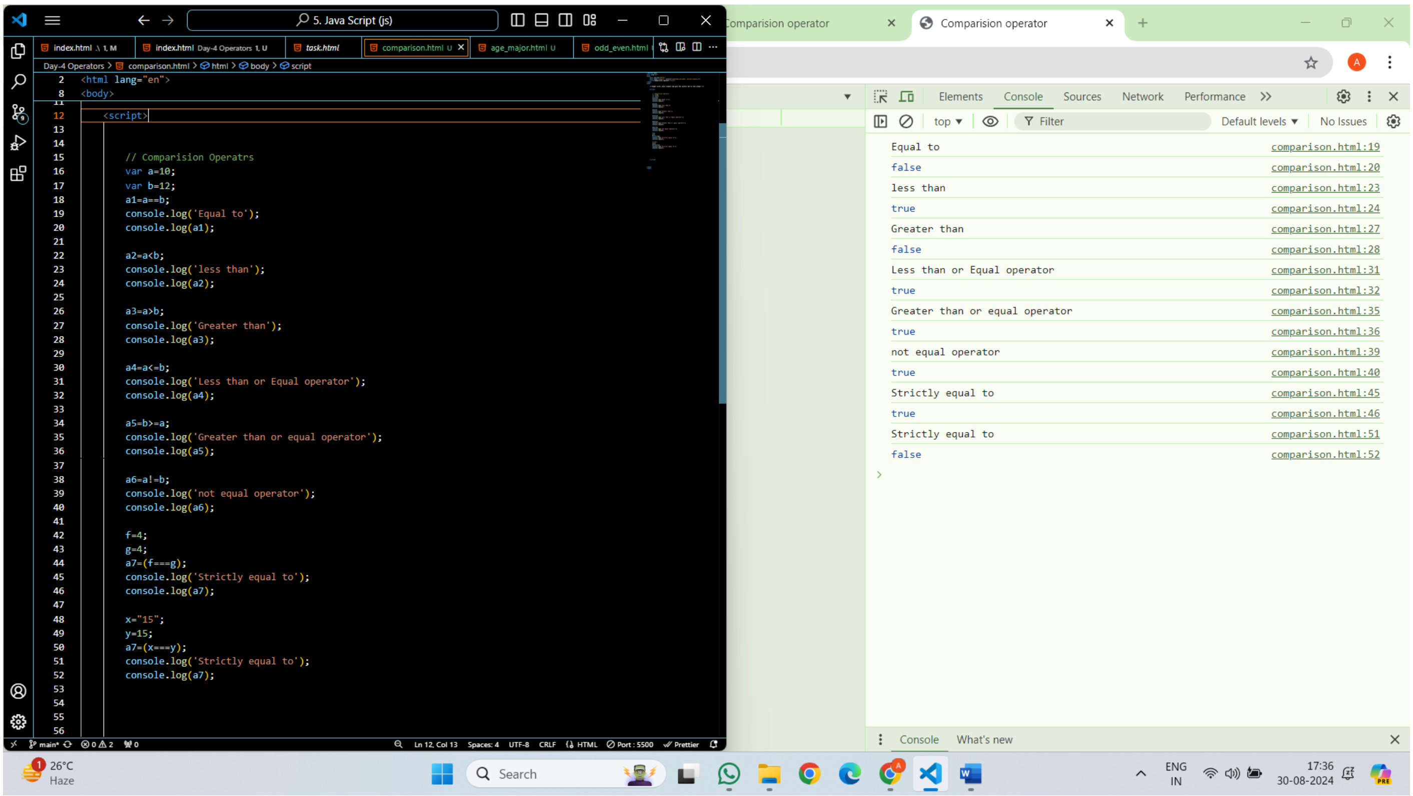
```
Add and assign              task.html:63
120                         task.html:64
Subtract and assign         task.html:69
0                           task.html:70
Multiply and assign         task.html:75
20                          task.html:76
Divide and assign           task.html:81
0.8333333333333334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

Comparision Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.



Operator	Meaning	Example
`==`	Equal to	`3 == 5  // false`
`!=`	Not equal to	`3 != 4  // true`
`===`	Strictly equal to	`3 === "3"  // false`
`!==`	Strictly not equal to	`3 !== "3"  // true`
`>`	Greater than	`4 > 4  // false`
`<`	Less than	`3 < 3  // false`
`>=`	Greater than or equal to	`4 >= 4  // true`
`<=`	Less than or equal to	`3 <= 3  // true`



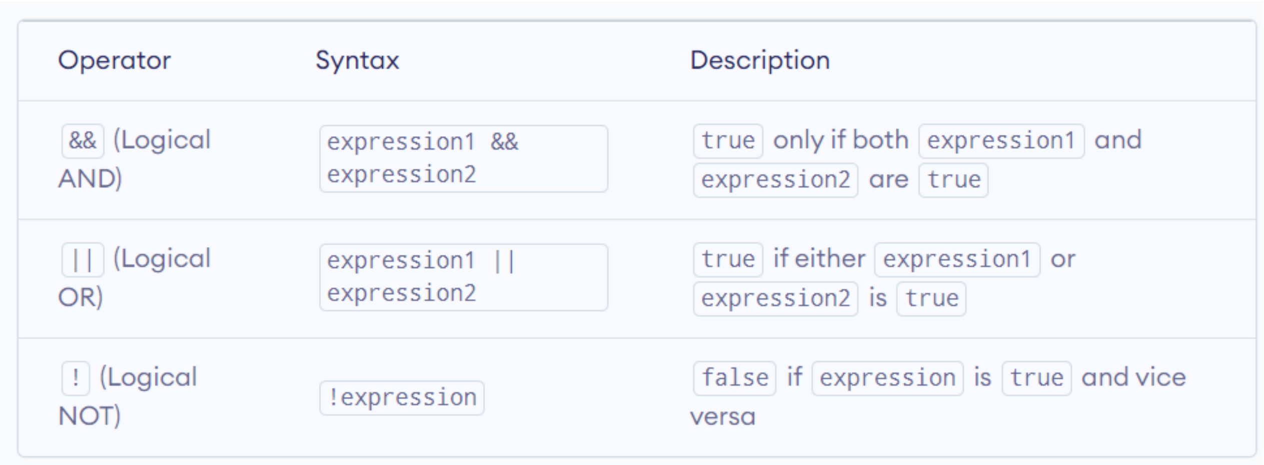
```
<html lang="en">
<body>

    <script>


        // Comparision Operatrs
        var a=10;
        var b=12;
        a1=a==b;
        console.log('Equal to');
        console.log(a1);

        a2=a<b;
        console.log('less than');
        console.log(a2);

        a3=a>b;
        console.log('Greater than');
        console.log(a3);

        a4=a<=b;
        console.log('Less than or Equal operator');
        console.log(a4);

        a5=b>=a;
        console.log('Greater than or equal operator');
        console.log(a5);

        a6=a!=b;
        console.log('not equal operator');
        console.log(a6);

        f=4;
        g=4;
        a7=(f===g);
        console.log('Strictly equal to');
        console.log(a7);

        x="15";
        y=15;
        a7=(x===y);
        console.log('Strictly equal to');
        console.log(a7);
```

```
Equal to                              comparison.html:19
false                                 comparison.html:20
less than                             comparison.html:23
true                                  comparison.html:24
Greater than                          comparison.html:27
false                                 comparison.html:28
Less than or Equal operator           comparison.html:31
true                                  comparison.html:32
Greater than or equal operator        comparison.html:35
true                                  comparison.html:36
not equal operator                    comparison.html:39
true                                  comparison.html:40
Strictly equal to                     comparison.html:45
true                                  comparison.html:46
Strictly equal to                     comparison.html:51
false                                 comparison.html:52
```

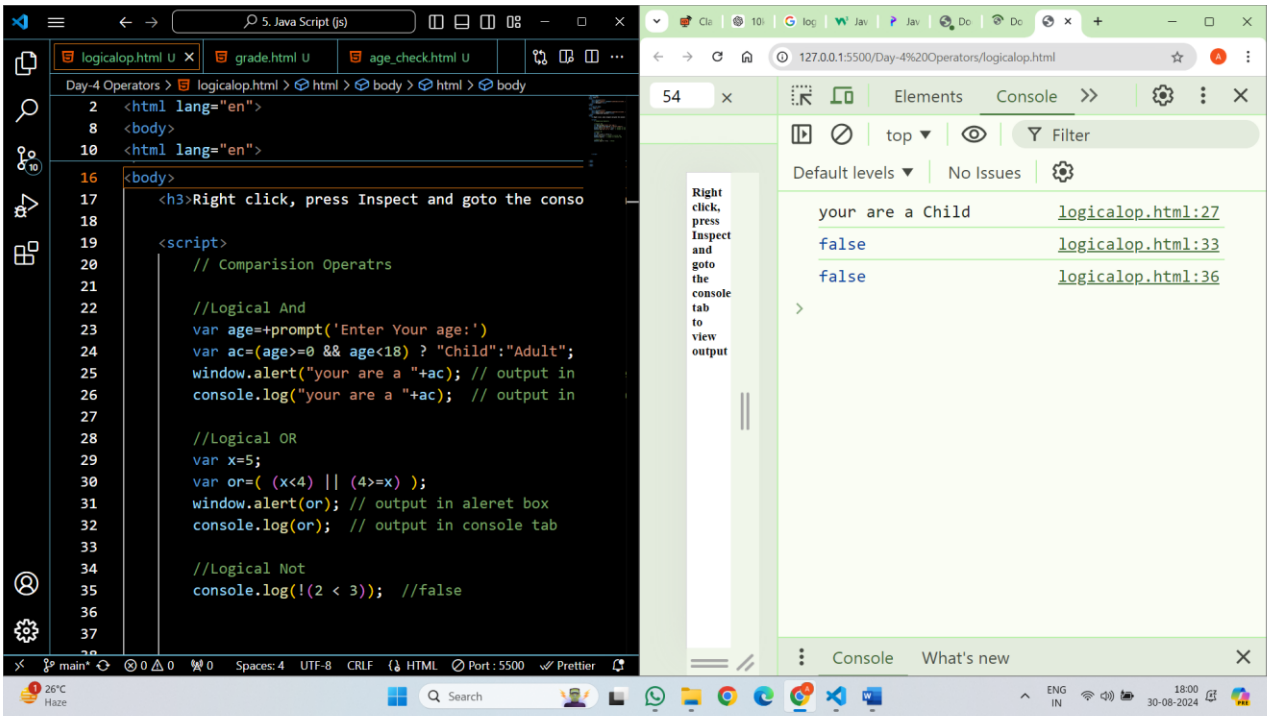
Logical Operator:

Logical operators return a boolean value by evaluating boolean expressions.

1. Logical And Operator: The logical AND operator **&&** returns true if both the expressions are true.
2. Logical OR Operator: The logical OR operator || returns true if at least one expression is true.
3. Logical Not Operator: The logical NOT operator **!** returns true if the specified expression is false and vice versa.



Operator	Syntax	Description
`&&` (Logical AND)	`expression1 && expression2`	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	`expression1 || expression2`	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	`!expression`	`false` if `expression` is `true` and vice versa



5. Java Script (js)
logicalop.html U
grade.html U
age_check.html U
Day-4 Operators > logicalop.html > html > body > html > body
main* ⊗ 0 ⚠ 0 Spaces: 4 UTF-8 CRLF HTML Port : 5500 Prettier
127.0.0.1:5500/Day-4%20Operators/logicalop.html
Elements Console
top Filter
Default levels No Issues
Console What's new

```
<html lang="en">
<body>
<html lang="en">
<body>
    <h3>Right click, press Inspect and goto the conso

    <script>
        // Comparision Operatrs

        //Logical And
        var age=+prompt('Enter Your age:')
        var ac=(age>=0 && age<18) ? "Child":"Adult";
        window.alert("your are a "+ac); // output in
        console.log("your are a "+ac);  // output in

        //Logical OR
        var x=5;
        var or=( (x<4) || (4>=x) );
        window.alert(or); // output in aleret box
        console.log(or);  // output in console tab

        //Logical Not
        console.log(!(2 < 3));  //false
```

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

```
your are a Child        logicalop.html:27
false                   logicalop.html:33
false                   logicalop.html:36
```

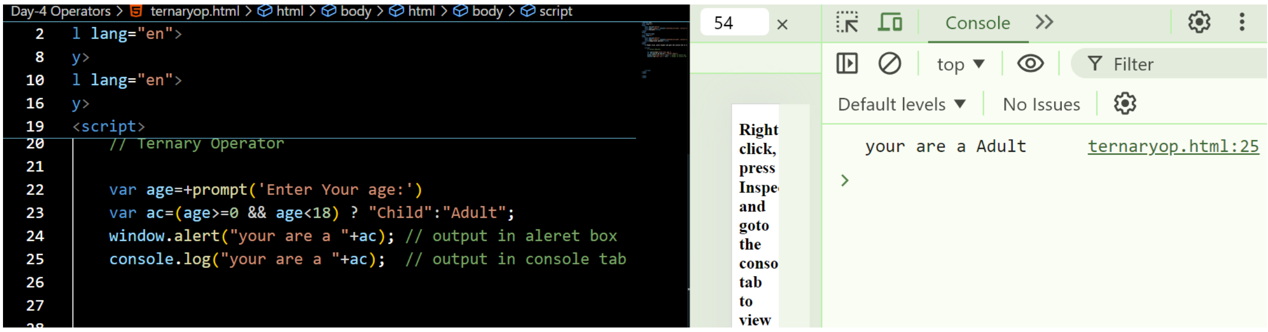
Ternary operator:

The Ternary Operator in JavaScript is a shortcut for writing simple if-else statements. It’s also known as the Conditional Operator because it works based on a condition. The ternary operator allows you to quickly decide between two values depending on whether a condition is true or false.

Syntax:

condition ? trueExpression : falseExpression

Example:


Day-4 Operators > ternaryop.html > html > body > html > body > script
54
Console
top
Filter
Default levels
No Issues

```
l lang="en">
y>
l lang="en">
y>
<script>
    // Ternary Operator

    var age=+prompt('Enter Your age:')
    var ac=(age>=0 && age<18) ? "Child":"Adult";
    window.alert("your are a "+ac); // output in aleret box
    console.log("your are a "+ac);  // output in console tab
```

```
your are a Adult        ternaryop.html:25
>
```

Right click, press Inspe and goto the conso tab to view

Nullish coalescing operator (??)

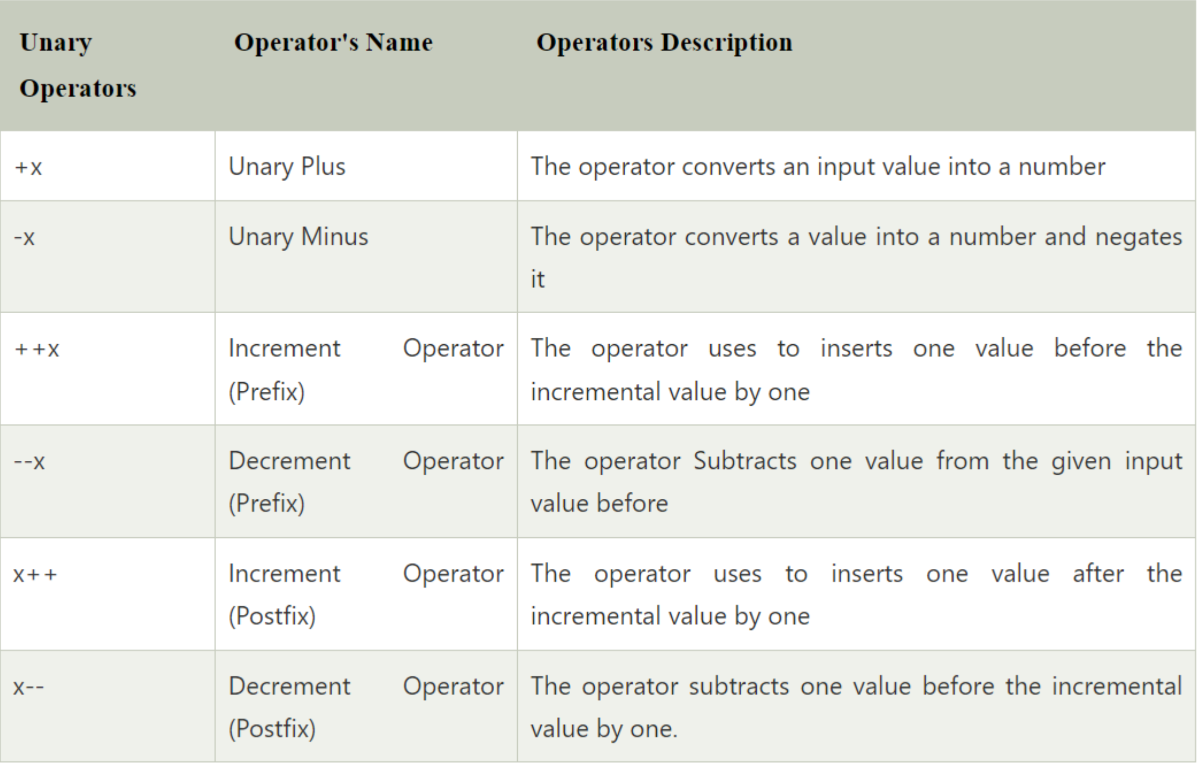
is a logical operator that returns its right-hand side operand when its left-hand side operand is null or undefined, and otherwise returns its left-hand side operand. It’s commonly used to provide default values for variables.

Example:

```
<script>
      //Nulish Coalescing Operator
       var a=null;
       var b=a ?? "Some Content";
       console.log(b);     // Some Content
   </script>
```

Unary Operator:

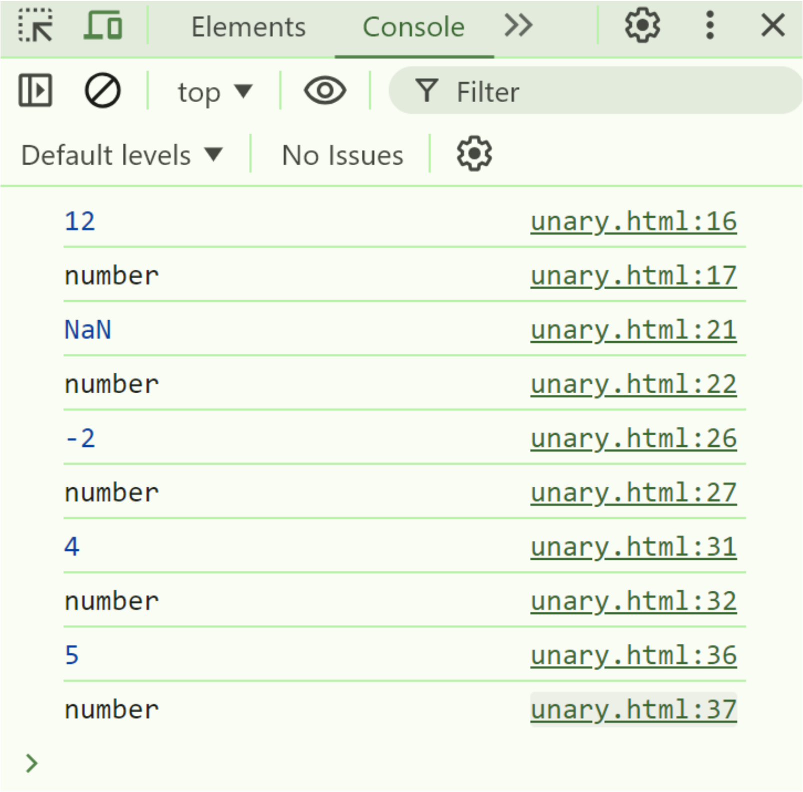
- Unary operators in JavaScript are unique operators that consider a single input and carry out all possible operations.
- The Unary plus, unary minus, prefix increments, postfix increments, postfix decrements, and prefix decrements are examples of these operators. These operators are either put before or after the operand.
- The unary operators are more effective in executing functions than JavaScript; they are more popular. Unary operators are flexible and versatile since they cannot be overridden.



Unary Operators	Operator's Name	Operators Description
+x	Unary Plus	The operator converts an input value into a number
-x	Unary Minus	The operator converts a value into a number and negates it
++x	Increment Operator (Prefix)	The operator uses to inserts one value before the incremental value by one
--x	Decrement Operator (Prefix)	The operator Subtracts one value from the given input value before
x++	Increment Operator (Postfix)	The operator uses to inserts one value after the incremental value by one
x--	Decrement Operator (Postfix)	The operator subtracts one value before the incremental value by one.

Example:

```
  <script>
        // Using unary plus to convert string to number
        let str1 = "12";
        let num = +str1;
        console.log(num);
        console.log(typeof (num))  // Here we are using typeof operator

        // "Abhinav" cannot be converted to a number
        let str2 = +"Abhinav";
        console.log(str2);
        console.log(typeof (str2))

        let s1='2'
        let n1 = -s1;
        console.log(n1);
        console.log(typeof (n1))

        let s2='3'
        let n2 = ++s2;
        console.log(n2);
        console.log(typeof (n2))

        let s3='5'
        let n3 = s3++;
        console.log(n3);
        console.log(typeof (n3))
  </script>
```



```
12          unary.html:16
number      unary.html:17
NaN         unary.html:21
number      unary.html:22
-2          unary.html:26
number      unary.html:27
4           unary.html:31
number      unary.html:32
5           unary.html:36
number      unary.html:37
>
```

Type Coercion

Type coercion refers to the automatic or implicit conversion of values from one data type to another.

In programming, type conversion is the process of converting data of one type to another. For example, converting string data to number.

There are two types of type conversion in JavaScript:

- **Implicit Conversion** - Automatic type conversion.
- **Explicit Conversion** - Manual type conversion.

Explicit Type Conversion

JavaScript type conversion, allowing you to convert values from one data type to another.

1. **String()**: Converts a value to a string.

```
let num = 123;
let str = String(num);
console.log(str);
// Output: "123"
```

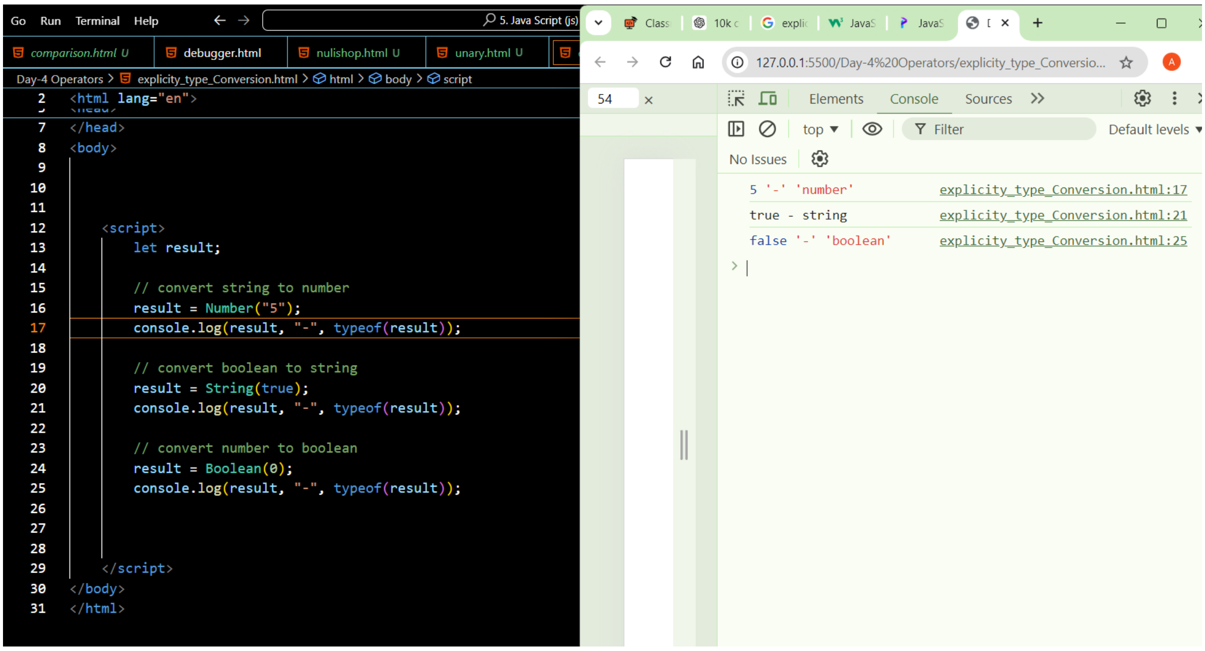
2. **Number()**: Converts a value to a number.

```
let str = "123";
let num = Number(str);
console.log(num); // Output: 123
```

3. **Boolean()**: Converts a value to a boolean.

```
let num = 0;
let bool = Boolean(num);
console.log(bool); // Output: false
```

Example:



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



    <script>
        let result;

        // convert string to number
        result = Number("5");
        console.log(result, "-", typeof(result));

        // convert boolean to string
        result = String(true);
        console.log(result, "-", typeof(result));

        // convert number to boolean
        result = Boolean(0);
        console.log(result, "-", typeof(result));



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

```
5 '-' 'number'          explicity_type_Conversion.html:17
true - string           explicity_type_Conversion.html:21
false '-' 'boolean'     explicity_type_Conversion.html:25
```

How to take or get input from Users:

Var a= +prompt(‘Enter Your Data’);

In JavaScript, values are categorized as either "truthy" or "falsy"

Falsy Values:

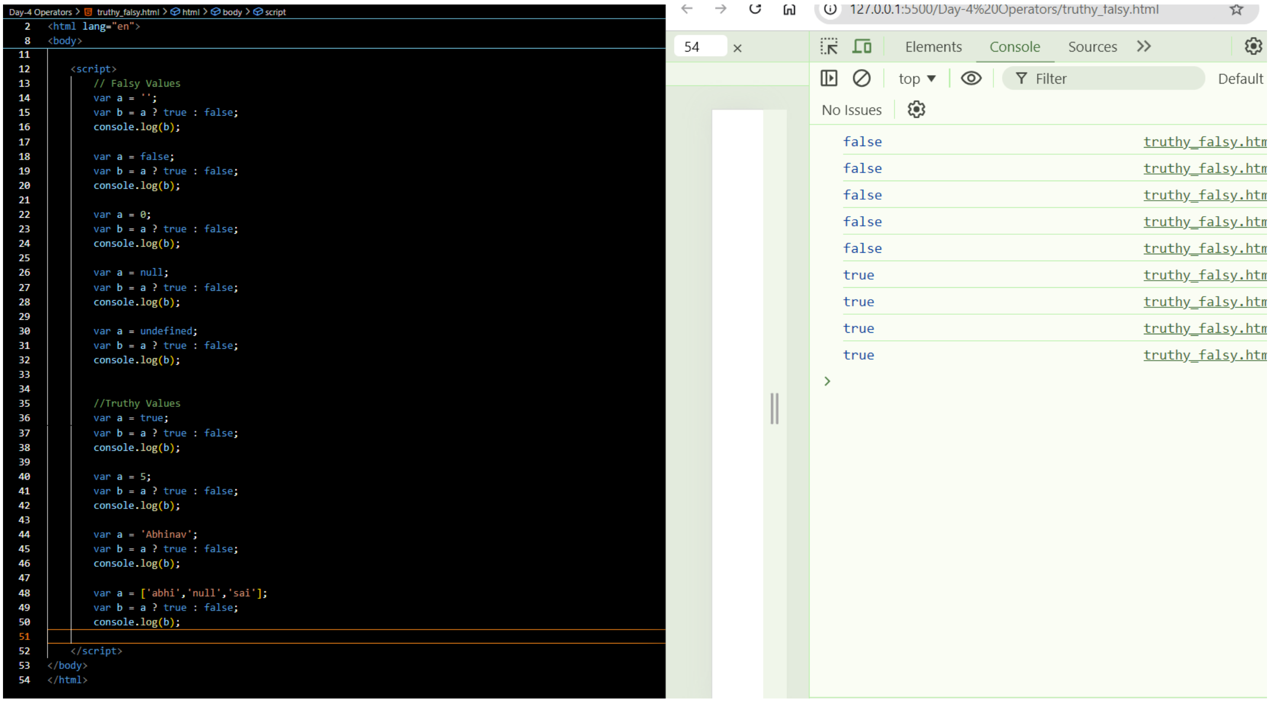
1. **false**: The boolean value false itself.
2. **0**: The number zero.
3. **""**: Empty string.
4. **null**: The absence of any value.
5. **undefined**: A variable that has not been assigned a value or a property that does not exist.
6. **NaN**: Not-a-Number.

Truthy Values:

1. **true**: The boolean value true itself.
2. **Non-zero numbers**: Any number other than 0 (including negative numbers and decimals).
3. **Non-empty strings**: Any string with at least one character.
4. **Non-empty arrays**: Arrays with at least one element.

5. **Objects**: Any object (including functions and arrays) is truthy, even if it's empty.
6. **Functions**: Any function is truthy, even if it doesn't return anything.

Check Truthy, Falsy values using ternary operator:

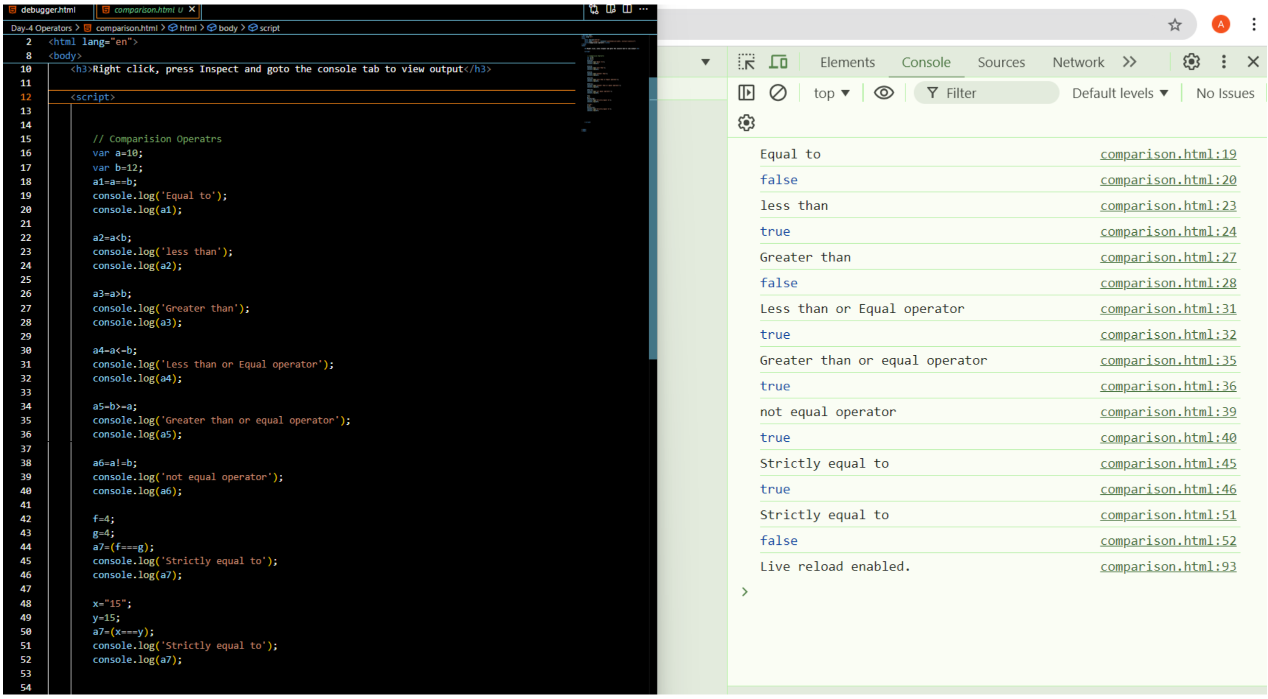

Day-4 Operators > truthy_falsy.html > html > body > script
127.0.0.1:5500/Day-4%20Operators/truthy_falsy.html
54
Elements
Console
Sources
top
Filter
Default
No Issues

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

    <script>
        // Falsy Values
        var a = '';
        var b = a ? true : false;
        console.log(b);

        var a = false;
        var b = a ? true : false;
        console.log(b);

        var a = 0;
        var b = a ? true : false;
        console.log(b);

        var a = null;
        var b = a ? true : false;
        console.log(b);

        var a = undefined;
        var b = a ? true : false;
        console.log(b);


        //Truthy Values
        var a = true;
        var b = a ? true : false;
        console.log(b);

        var a = 5;
        var b = a ? true : false;
        console.log(b);

        var a = 'Abhinav';
        var b = a ? true : false;
        console.log(b);

        var a = ['abhi','null','sai'];
        var b = a ? true : false;
        console.log(b);

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

```
false                                        truthy_falsy.htm
false                                        truthy_falsy.htm
false                                        truthy_falsy.htm
false                                        truthy_falsy.htm
false                                        truthy_falsy.htm
true                                         truthy_falsy.htm
true                                         truthy_falsy.htm
true                                         truthy_falsy.htm
true                                         truthy_falsy.htm
>
```

Tasks:

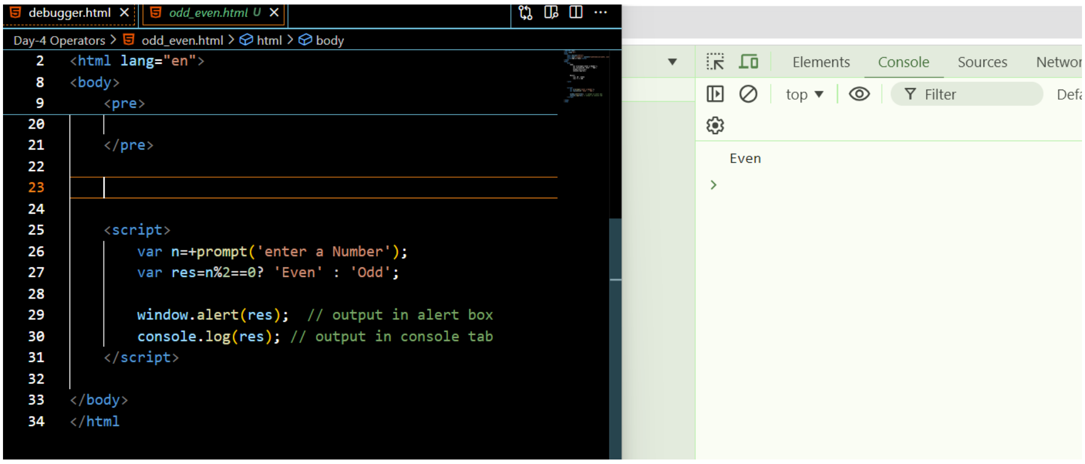
1. Write a JavaScript script that compares two variables using different comparison operators (==, ===, !=, !==, >, <, >=, <=) and prints the results.



```
<html lang="en">
<body>
    <h3>Right click, press Inspect and goto the console tab to view output</h3>

    <script>


        // Comparision Operatrs
        var a=10;
        var b=12;
        a1=a==b;
        console.log('Equal to');
        console.log(a1);

        a2=a<b;
        console.log('less than');
        console.log(a2);

        a3=a>b;
        console.log('Greater than');
        console.log(a3);

        a4=a<=b;
        console.log('Less than or Equal operator');
        console.log(a4);

        a5=b>=a;
        console.log('Greater than or equal operator');
        console.log(a5);

        a6=a!=b;
        console.log('not equal operator');
        console.log(a6);

        f=4;
        g=4;
        a7=(f===g);
        console.log('Strictly equal to');
        console.log(a7);

        x="15";
        y=15;
        a7=(x===y);
        console.log('Strictly equal to');
        console.log(a7);

```

```
Equal to                                 comparison.html:19
false                                    comparison.html:20
less than                                comparison.html:23
true                                     comparison.html:24
Greater than                             comparison.html:27
false                                    comparison.html:28
Less than or Equal operator              comparison.html:31
true                                     comparison.html:32
Greater than or equal operator           comparison.html:35
true                                     comparison.html:36
not equal operator                       comparison.html:39
true                                     comparison.html:40
Strictly equal to                        comparison.html:45
true                                     comparison.html:46
Strictly equal to                        comparison.html:51
false                                    comparison.html:52
Live reload enabled.                     comparison.html:93
>
```

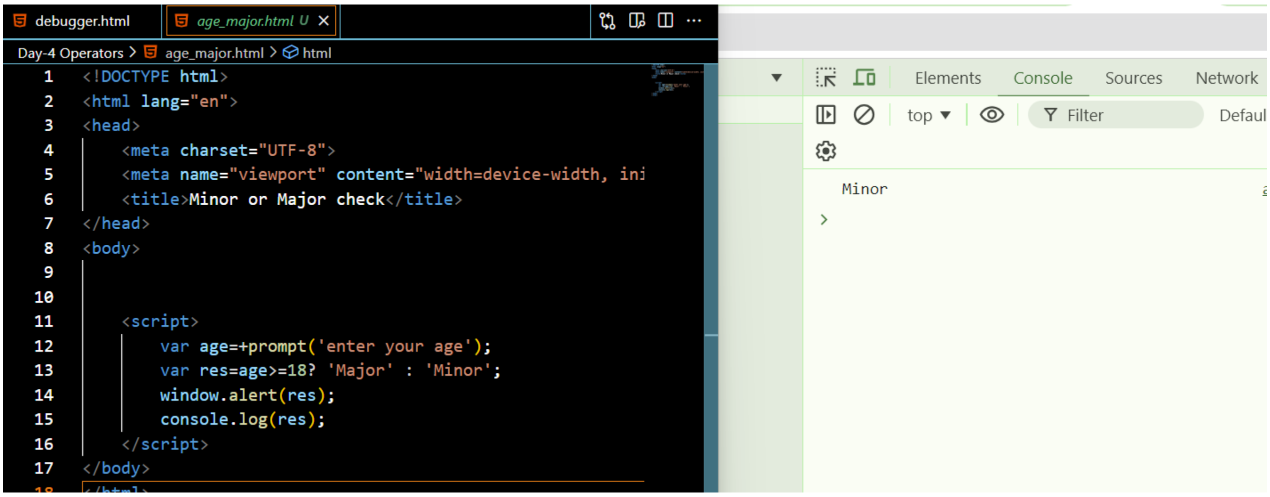
2. Write a JavaScript script that uses the ternary operator to determine if a number is even or odd.



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

    </pre>


    <script>
        var n=+prompt('enter a Number');
        var res=n%2==0? 'Even' : 'Odd';

        window.alert(res);  // output in alert box
        console.log(res); // output in console tab
    </script>

</body>
</html>
```

```
Even
```

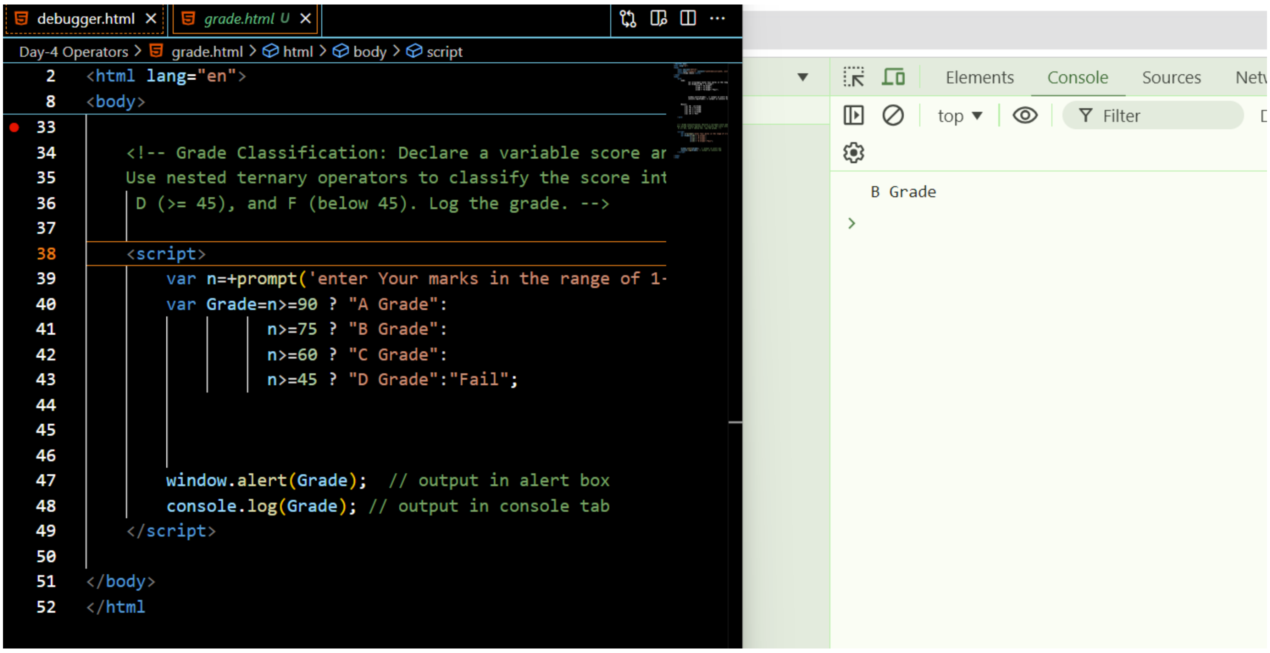
3. Expand the script to include a ternary operation that checks if a user is an adult (18+) or a minor.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, ini
    <title>Minor or Major check</title>
</head>
<body>


    <script>
        var age=+prompt('enter your age');
        var res=age>=18? 'Major' : 'Minor';
        window.alert(res);
        console.log(res);
    </script>
</body>
</html>
```

```
Minor
```

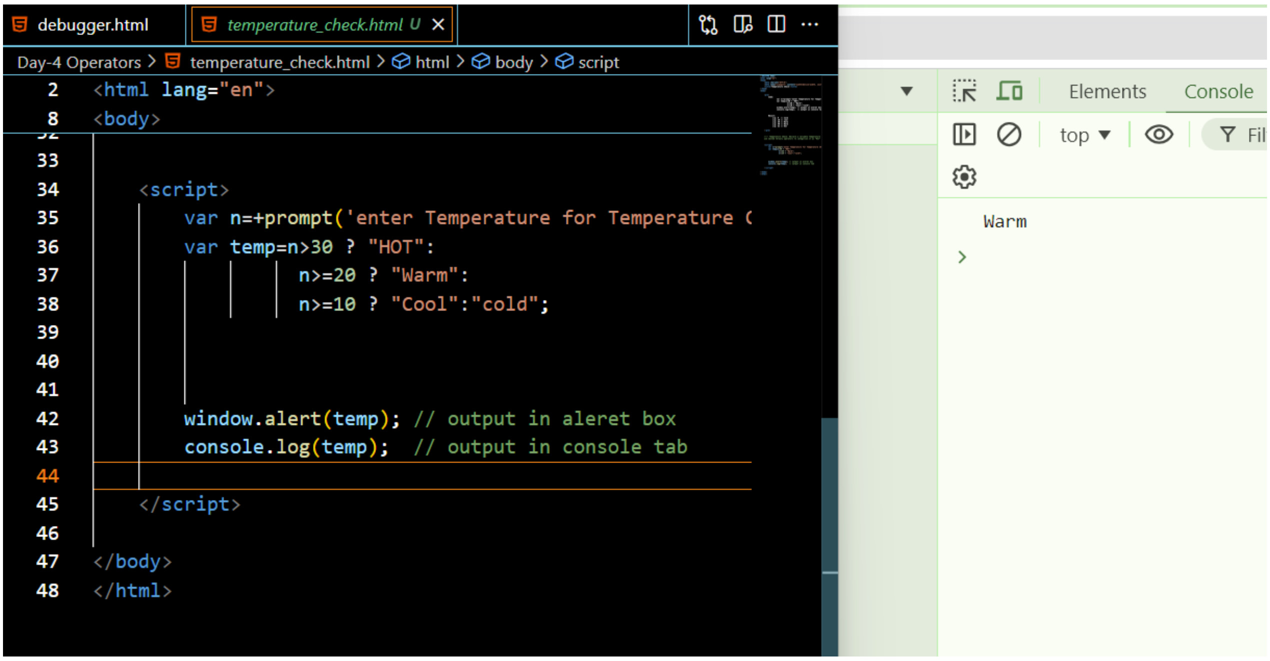
4. Grade Classification: Declare a variable score and set it to a value between 0 and 100. Use nested ternary operators to classify the score into grades: A (>= 90), B (>= 75), C (>= 60), D (>= 45), and F (below 45). Log the grade.



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

    <!-- Grade Classification: Declare a variable score ar
    Use nested ternary operators to classify the score int
     D (>= 45), and F (below 45). Log the grade. -->

    <script>
        var n=+prompt('enter Your marks in the range of 1-
        var Grade=n>=90 ? "A Grade":
                    n>=75 ? "B Grade":
                    n>=60 ? "C Grade":
                    n>=45 ? "D Grade":"Fail";



        window.alert(Grade);  // output in alert box
        console.log(Grade); // output in console tab
    </script>

</body>
</html>
```

```
B Grade
```

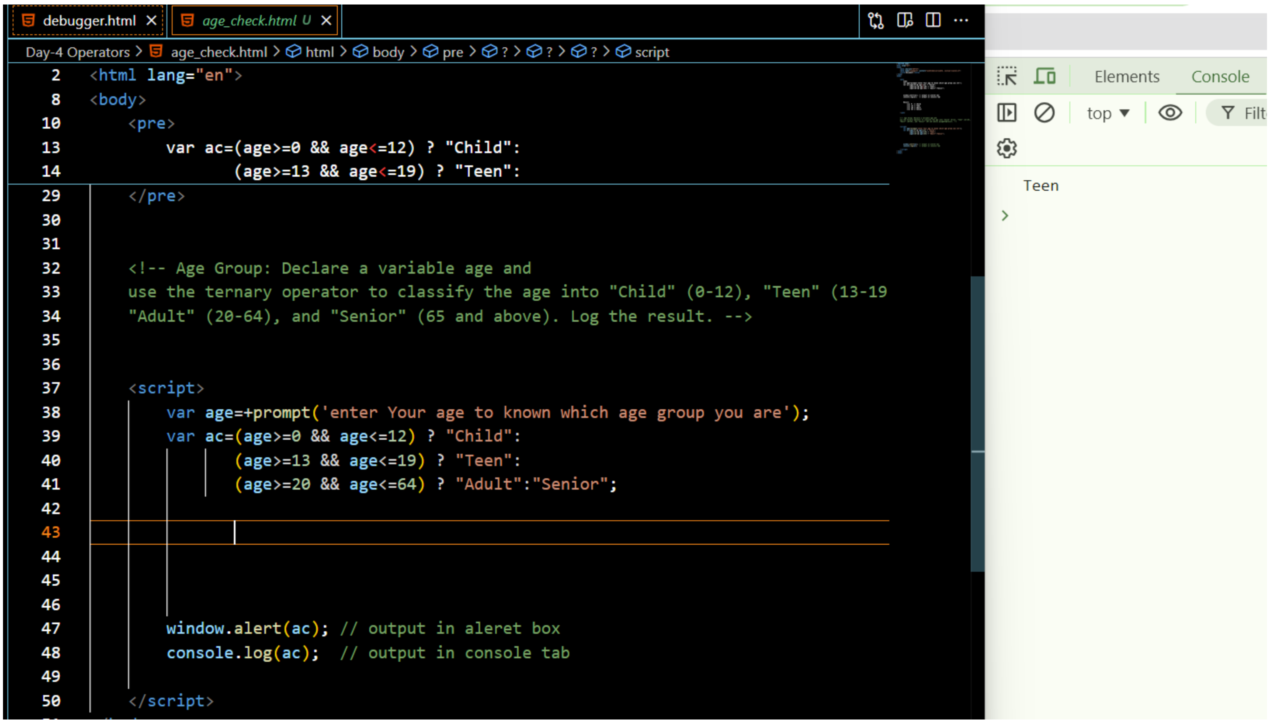
5. Temperature Check: Declare a variable temperature and use nested ternary operators to categorize it as "Hot" (above 30), "Warm" (20-30), "Cool" (10-19), and "Cold" (below 10). Log the result.


debugger.html
temperature_check.html U
Day-4 Operators > temperature_check.html > html > body > script
Elements
Console
top
Fil

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

    <script>
        var n=+prompt('enter Temperature for Temperature (
        var temp=n>30 ? "HOT":
                   n>=20 ? "Warm":
                   n>=10 ? "Cool":"cold";



        window.alert(temp); // output in alerет box
        console.log(temp);  // output in console tab

    </script>

</body>
</html>
```

```
Warm
```

6. Age Group: Declare a variable age and use the ternary operator to classify the age into "Child" (0-12), "Teen" (13-19), "Adult" (20-64), and "Senior" (65 and above). Log the result.


debugger.html
age_check.html U
Day-4 Operators > age_check.html > html > body > pre > ? > ? > ? > script
Elements
Console
top
Filt

```
<html lang="en">
<body>
    <pre>
        var ac=(age>=0 && age<=12) ? "Child":
               (age>=13 && age<=19) ? "Teen":
    </pre>


    <!-- Age Group: Declare a variable age and
    use the ternary operator to classify the age into "Child" (0-12), "Teen" (13-19
    "Adult" (20-64), and "Senior" (65 and above). Log the result. -->


    <script>
        var age=+prompt('enter Your age to known which age group you are');
        var ac=(age>=0 && age<=12) ? "Child":
               (age>=13 && age<=19) ? "Teen":
               (age>=20 && age<=64) ? "Adult":"Senior";




        window.alert(ac); // output in alerет box
        console.log(ac);  // output in console tab

    </script>
```

```
Teen
```