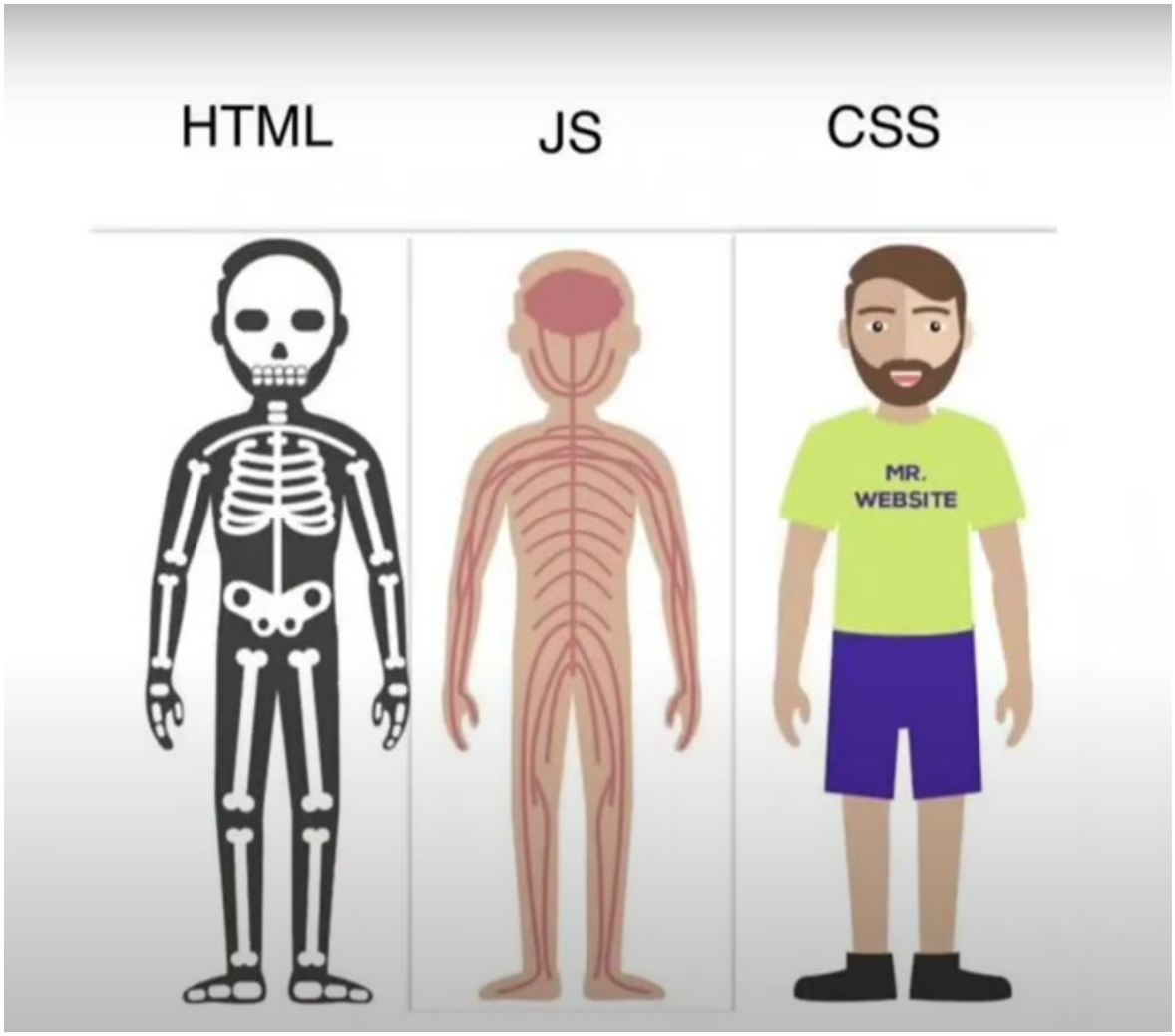
Java Script

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.



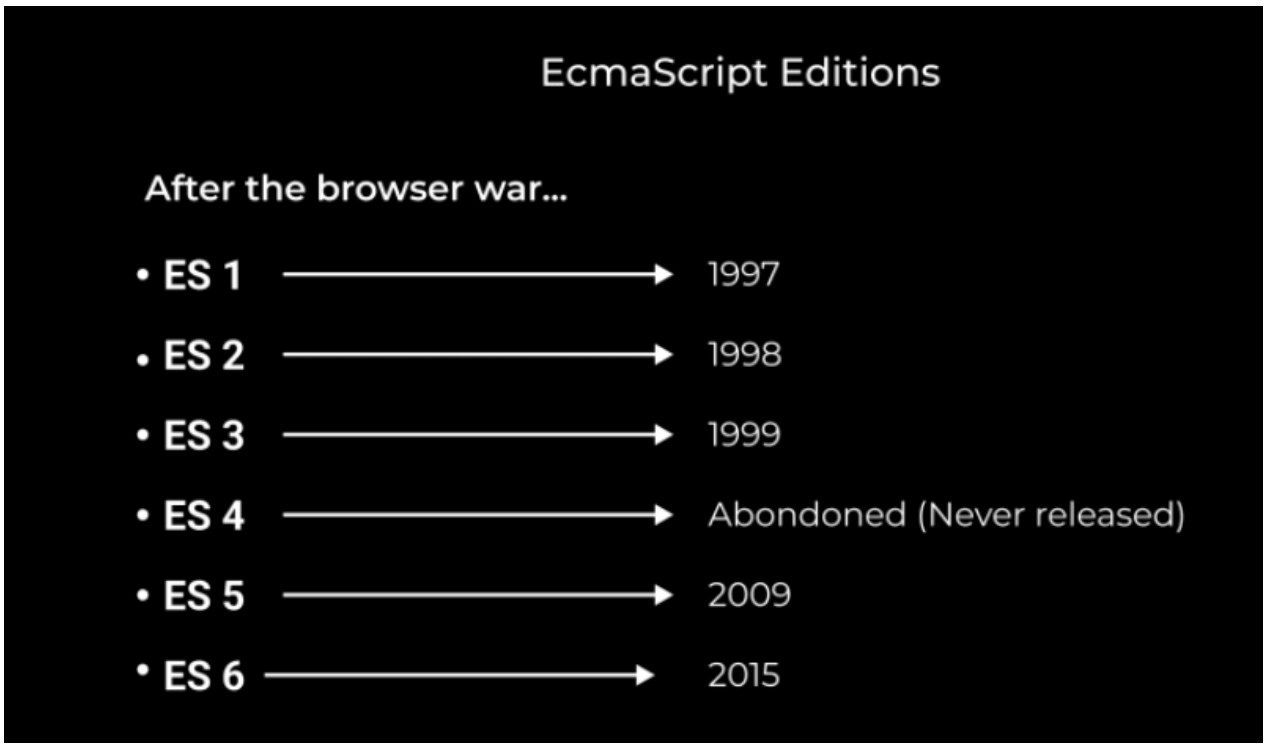
2. History of java script?

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.

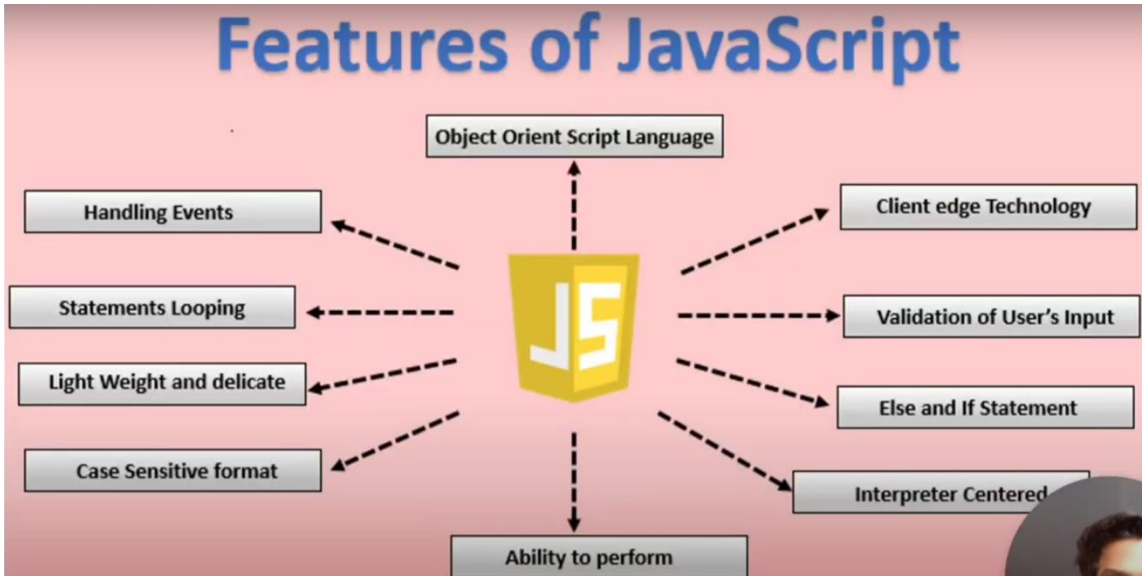


Originally, the name ECMAScript was just the formalization of JavaScript, but now languages like JScript and ActionScript are also based on the ECMAScript standard.

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**

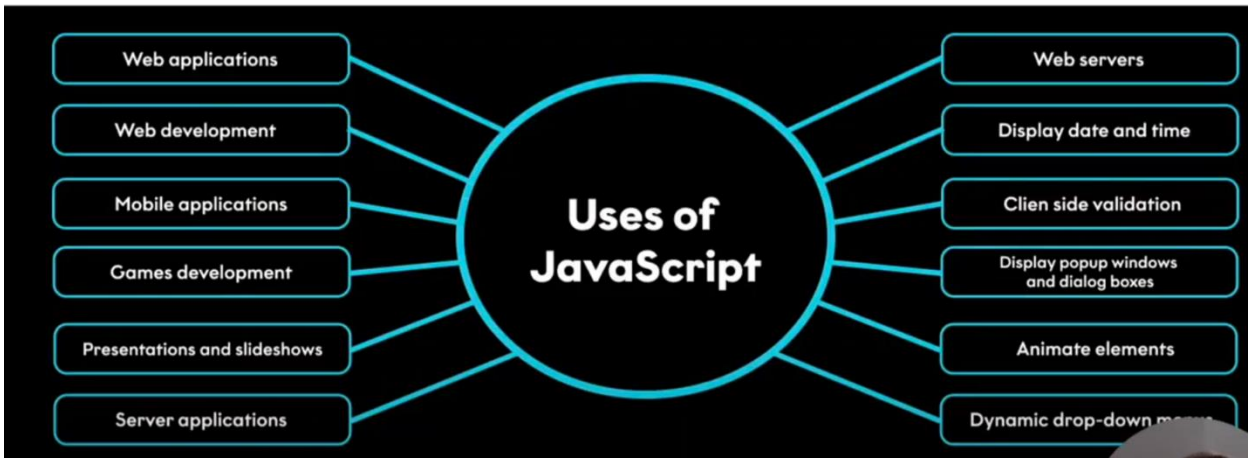
  Synchronous means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

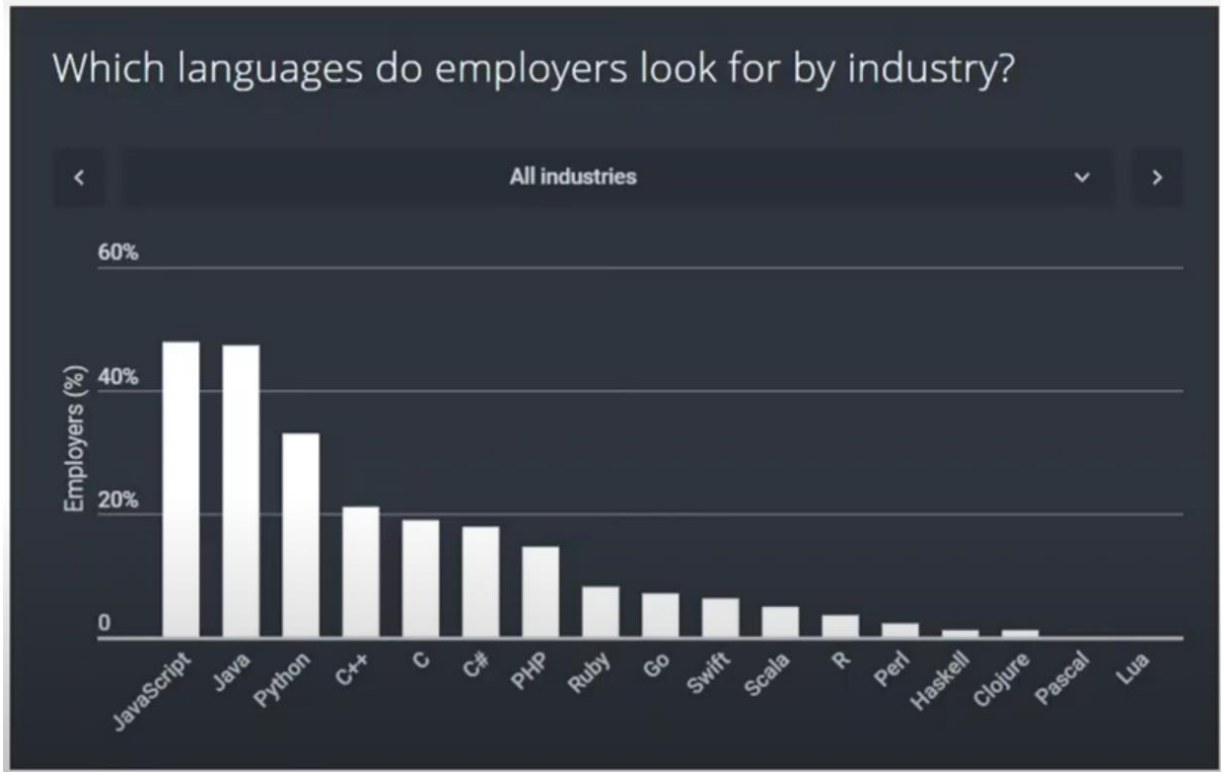
- **Obeject Oriented**

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.

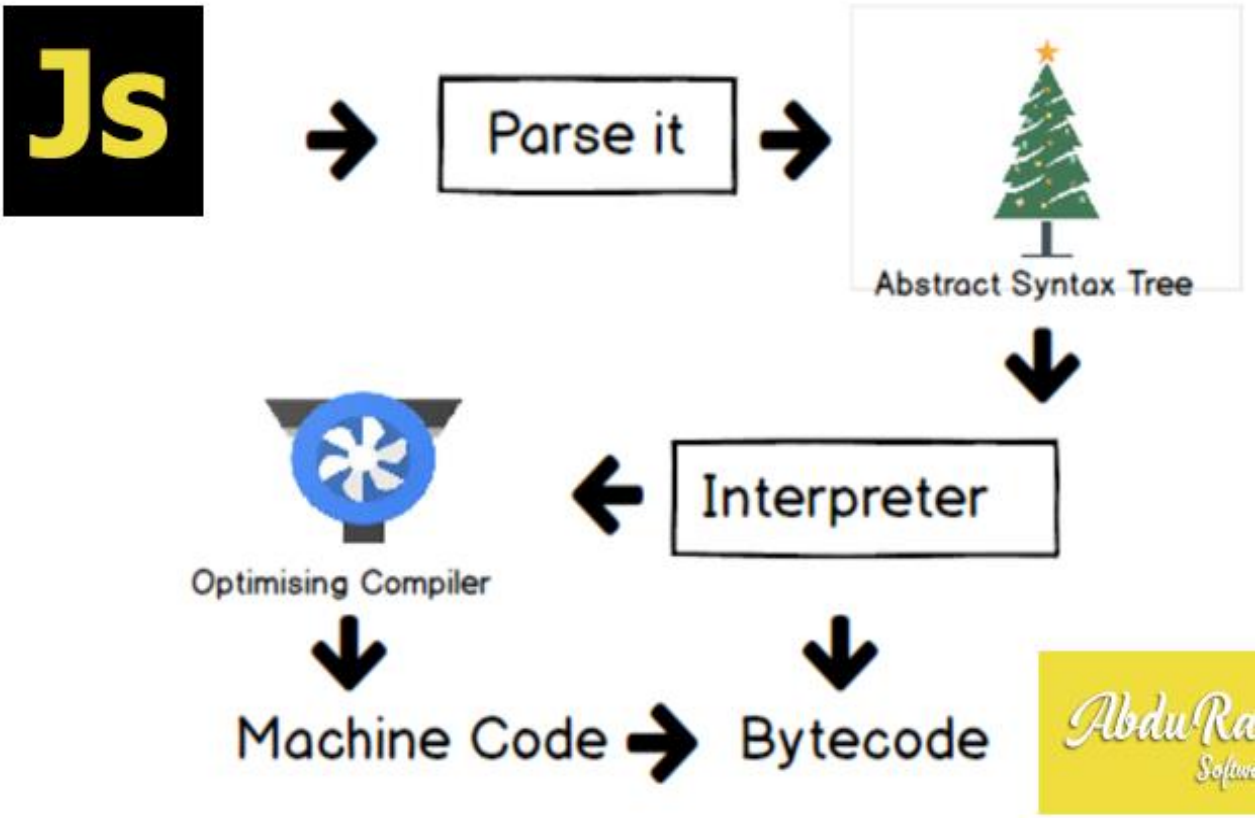
The team of developers that created the vanilla JavaScript is continuously working on it to improve it and make it more useful for the web-developers.

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:

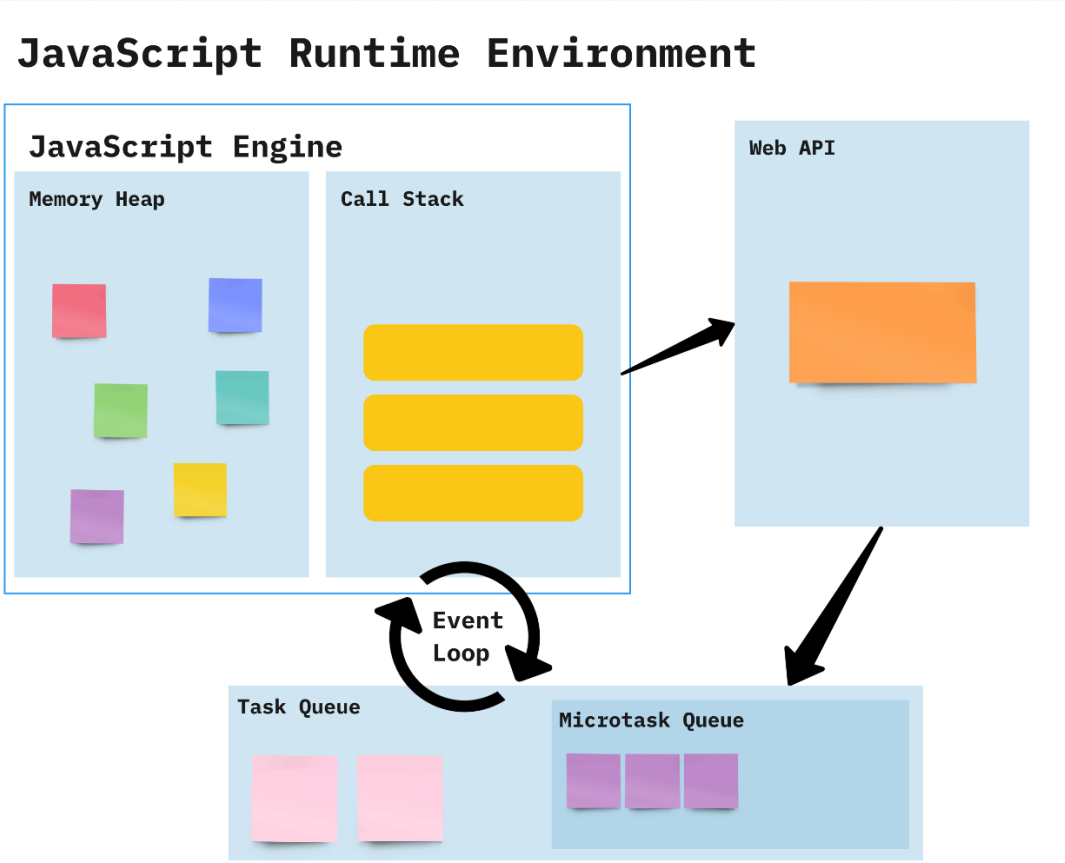
A Web API (Application Programming Interface) is a set of rules and tools that allows different software applications to communicate with each other over the web. It acts as an intermediary, enabling one application to interact with another application's data or functionality using standard web protocols,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

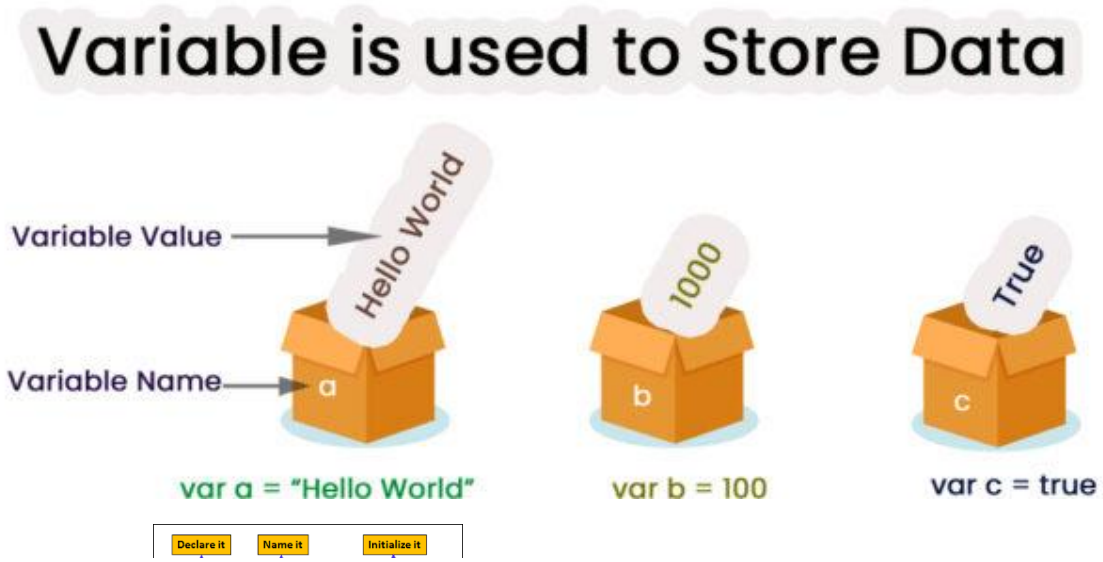
   By creating a js file with .js extention we can insert js file into the html document. That js file is linked in the script tag.

   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**

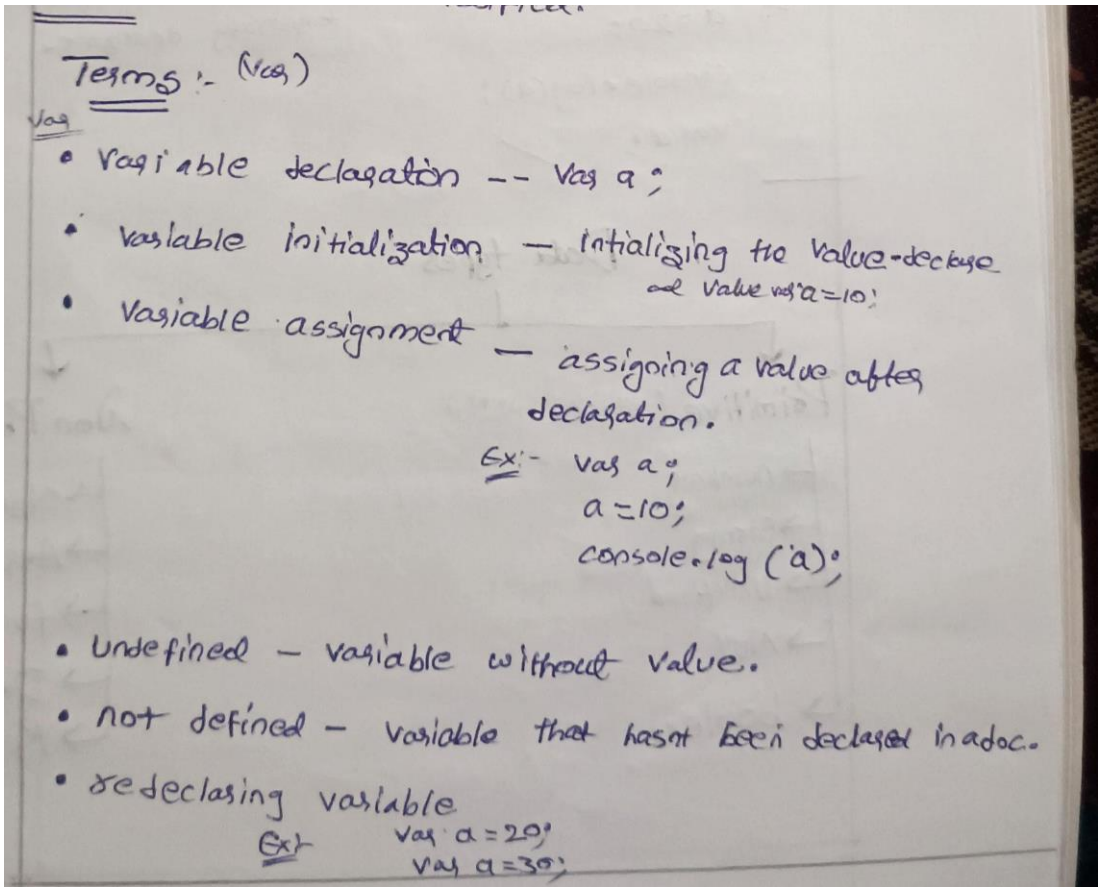
Example:

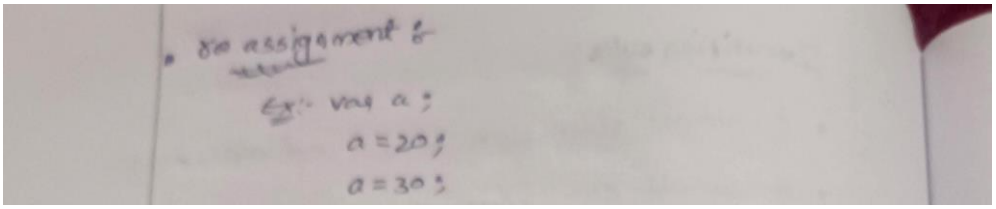
```
a=10;
Var b=20;
let c=5;
Const d=15;
console.log(a);     //10
console.log(a);    //20
console.log(a);   //5
console.log(a);   //15
```

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:





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

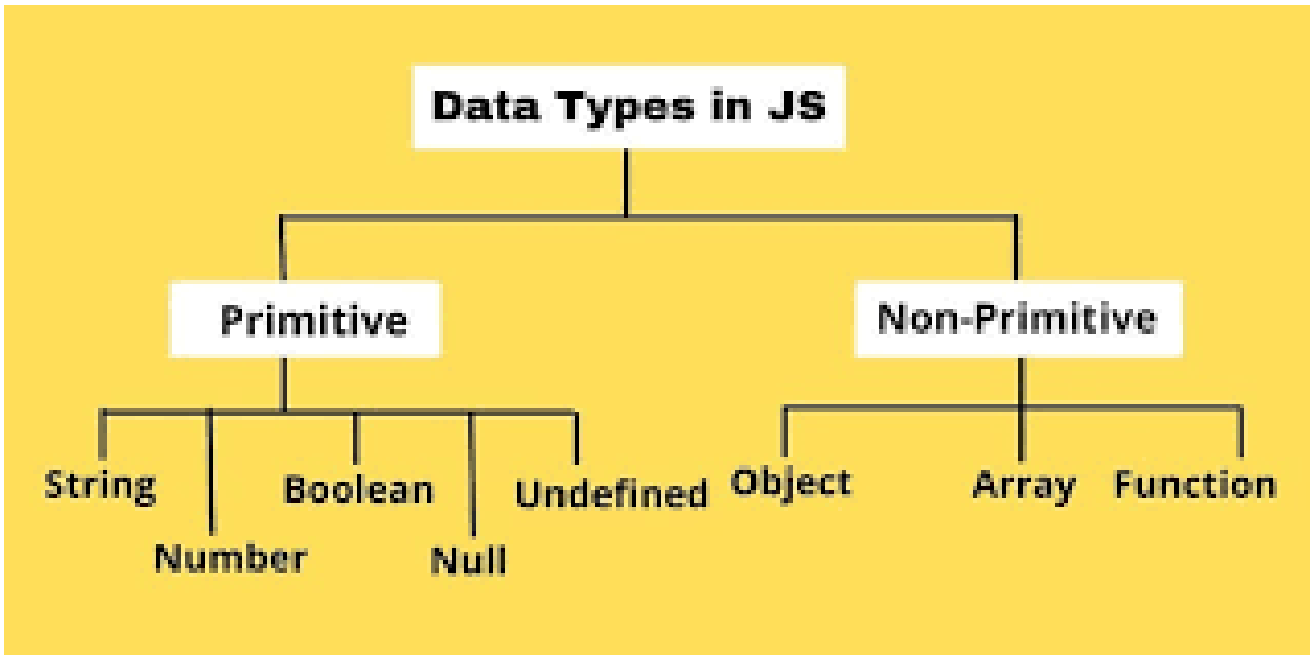
- It is a behaviour where the declaration of the variable and functions are moved to the top even before the execution.
- Only Declaration is hoisted not the Initialization
- Only works in **var** remaining **let** and **const** goes to temporary deadzone

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 Data 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 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

- Numbers :- 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 similar 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
```

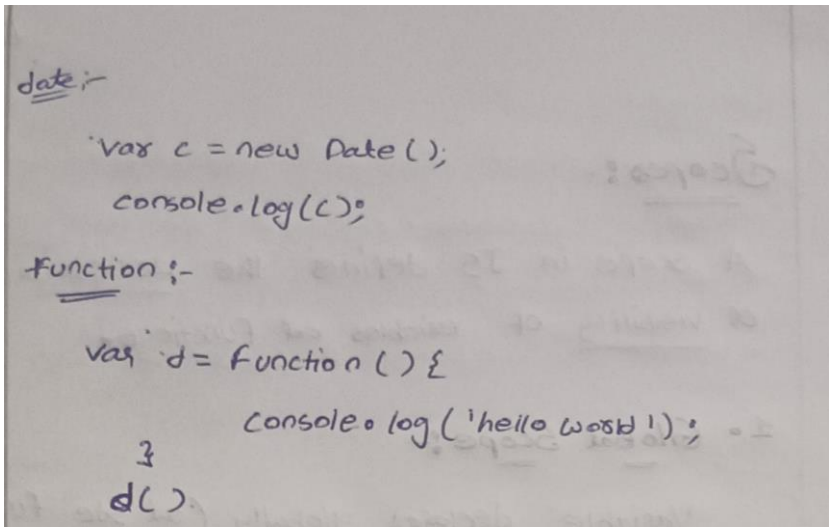
non primitive

arrays (number index)

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

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

Console.log(a[-1]);
```

objects (named index)

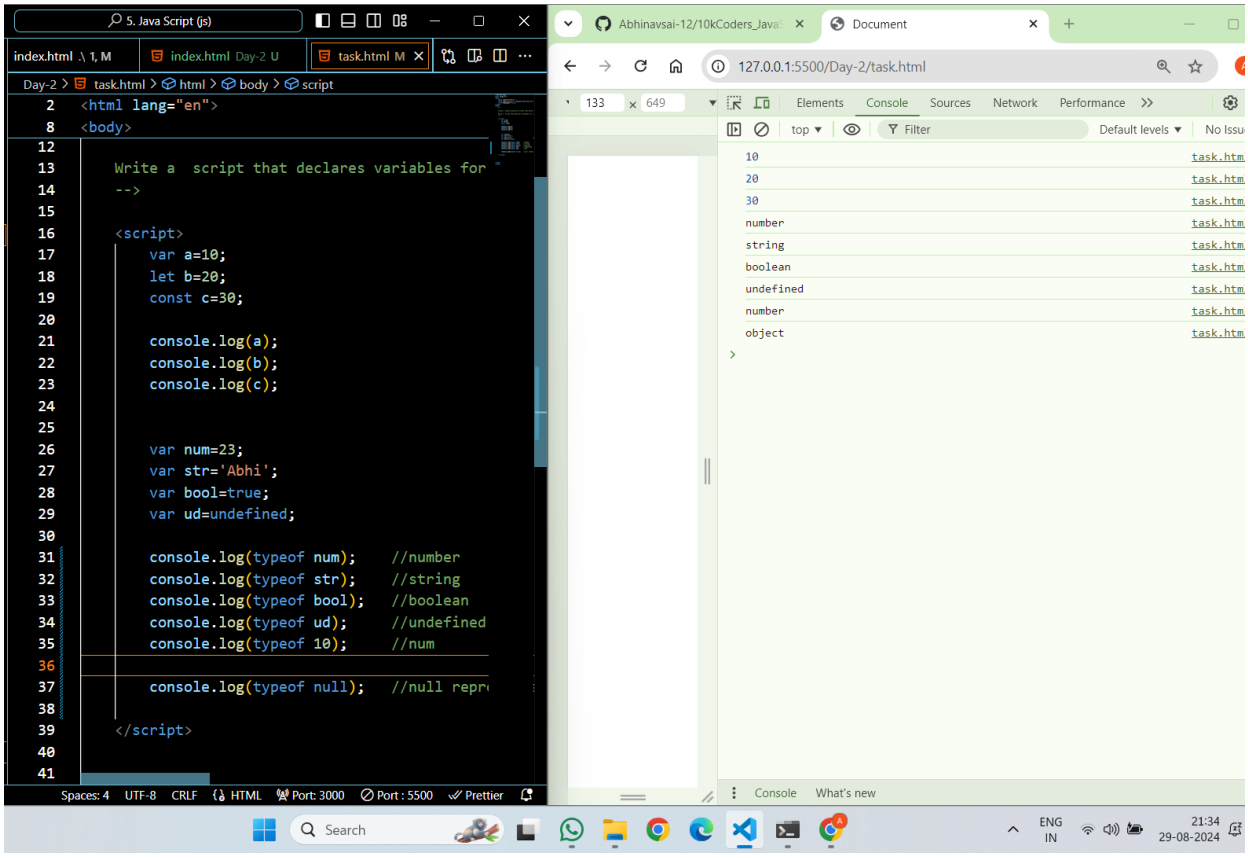
```
var b={
        id:1201,
        name: 'abhi',
        age: 22
    }
Consde.log(b.age);  //22
```



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".



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.

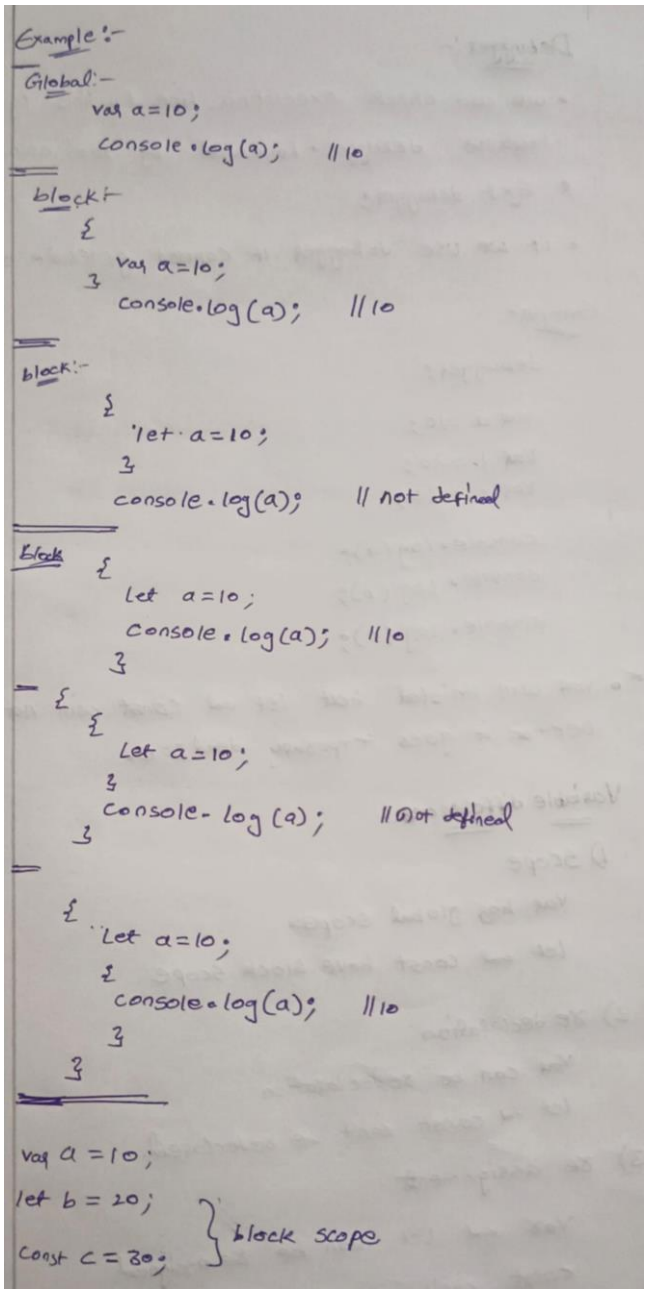
2. Block Scope:

   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.



Debugger

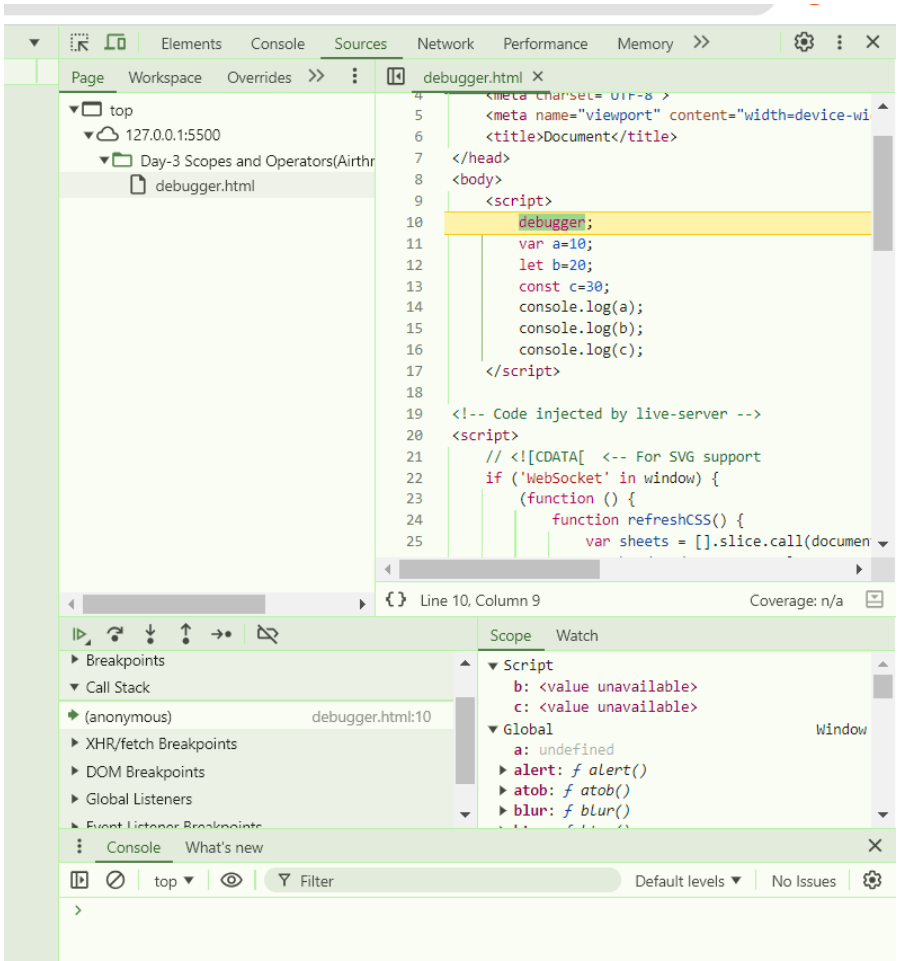
We can check Execution line by line by using keyword debugger followed by semi colon (:)

syntax:

```
debugger;
```

Example:

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



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:

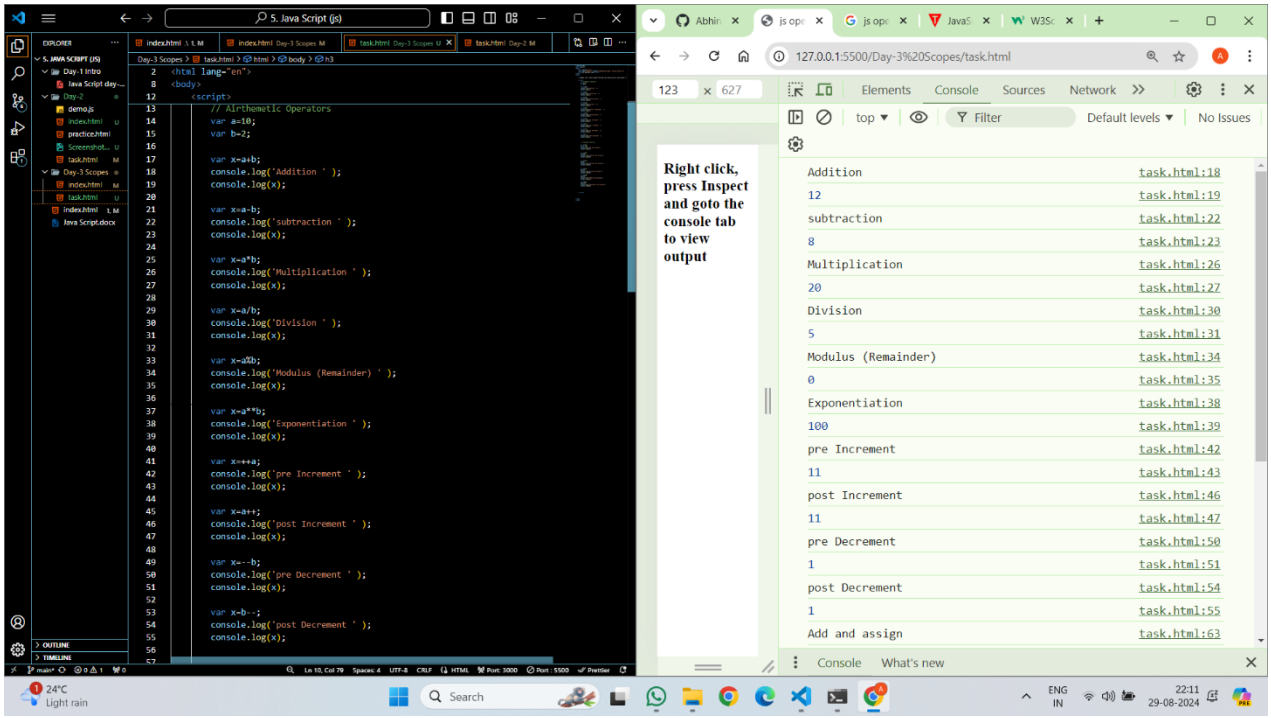
1. Airthmetic Operators
2. Assignment Operator
3. Comparision Operator
4. Logical Operator
5. Ternary Operator
6. Bitwise Oberator
7. String Operator
8. Typeof Operator

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:

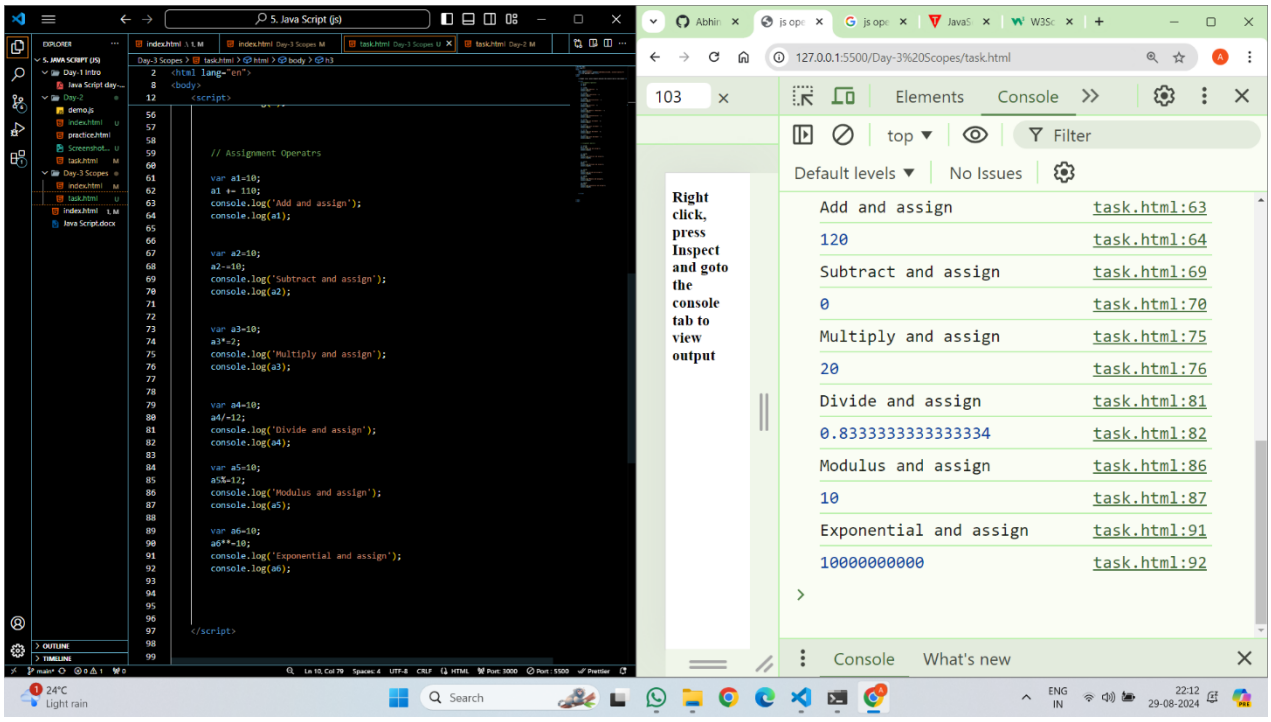


Assignment Operators:

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

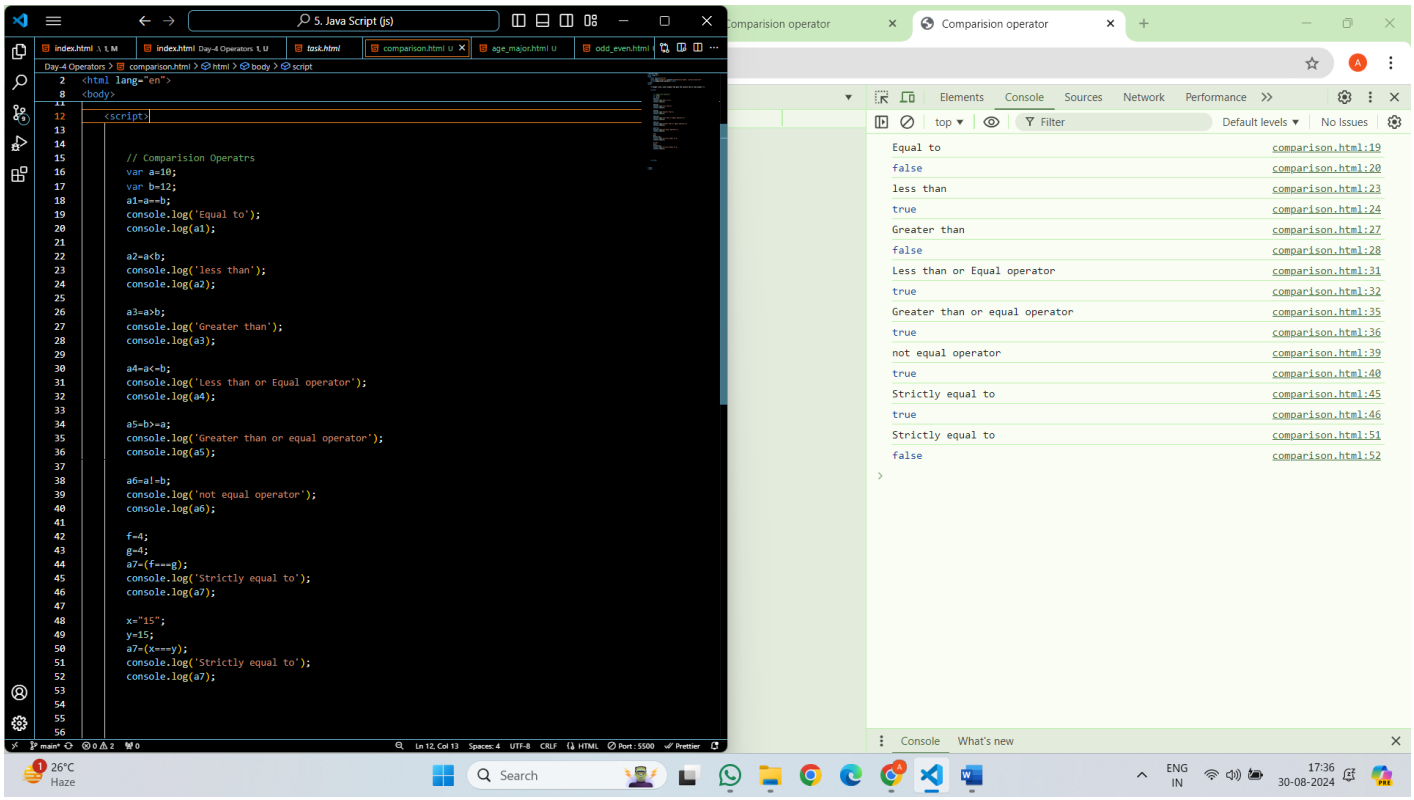
Example:



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`

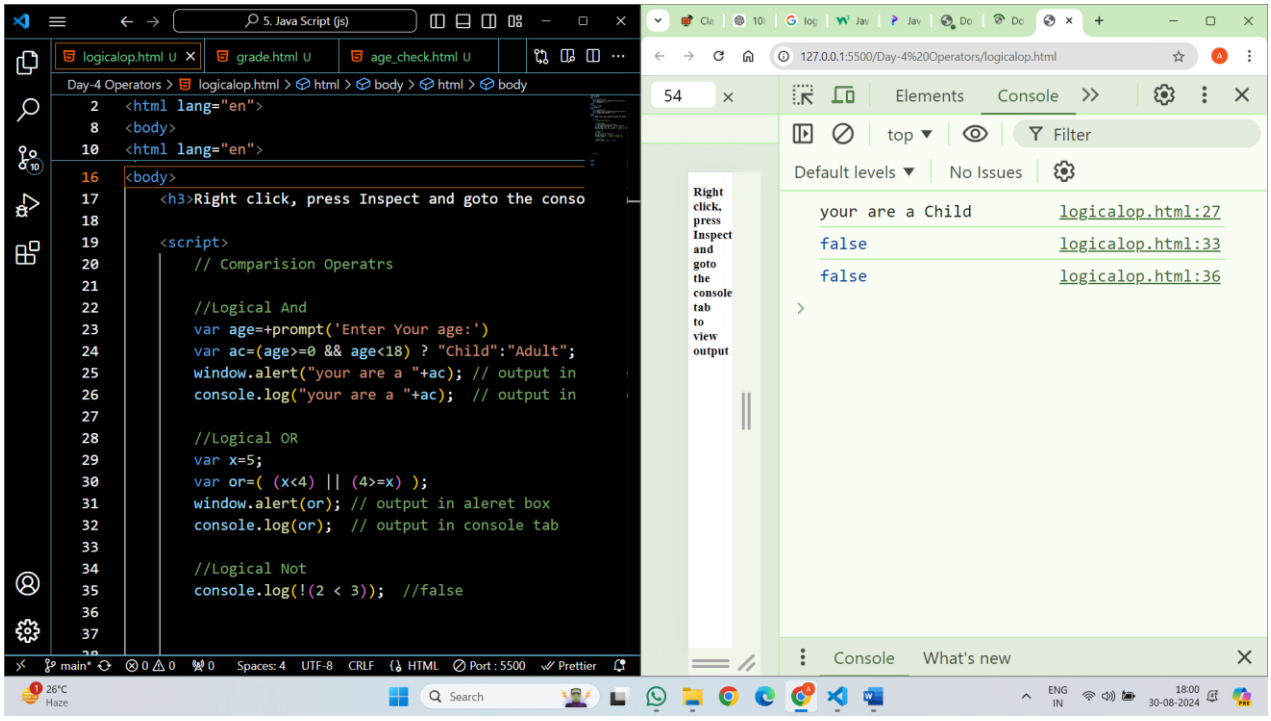


Logical Operator:

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



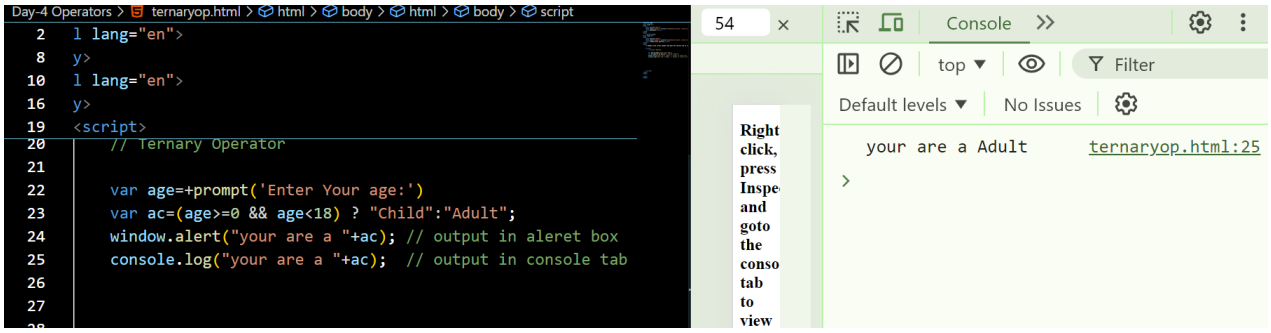
Ternary operator:

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:



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>
```

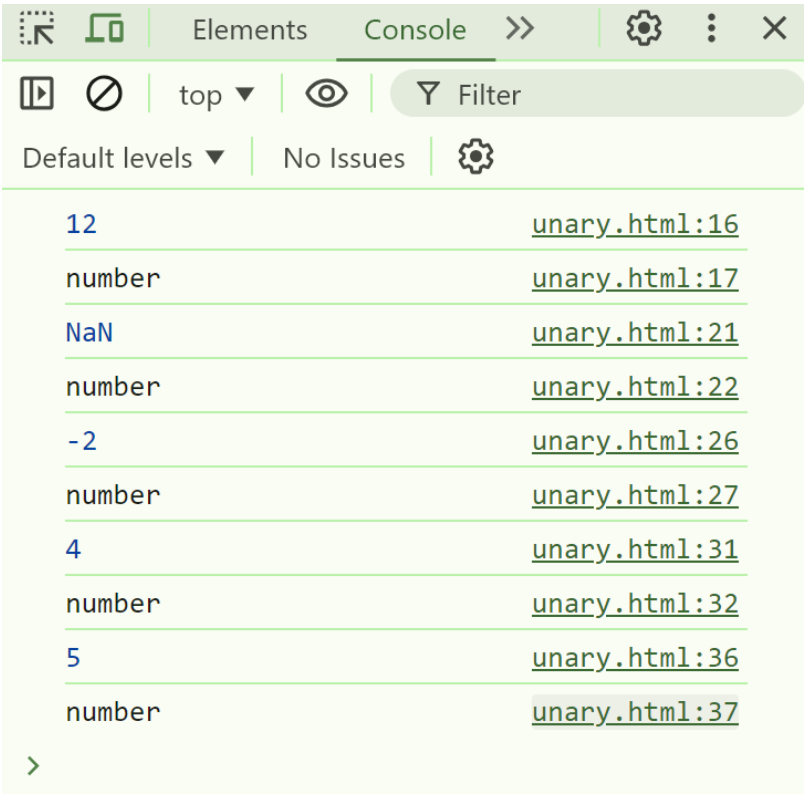
Unary Operator:

- 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>
```



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"
```

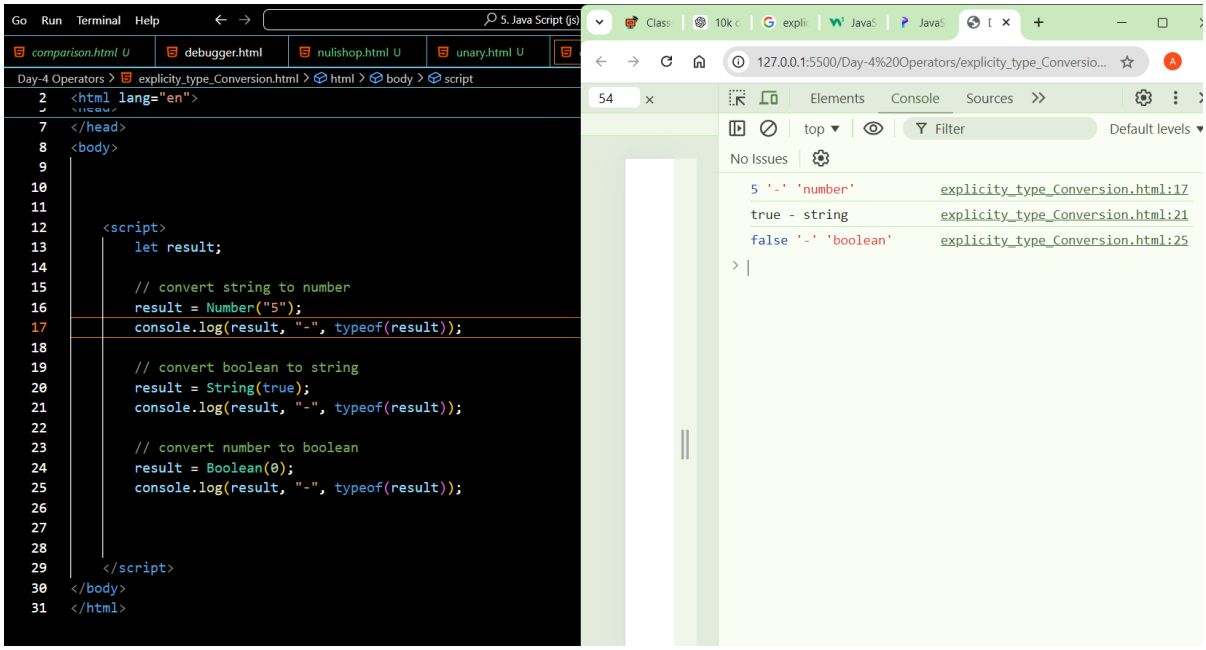
2. **Number()**: Converts a value to a number.

```
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:



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:

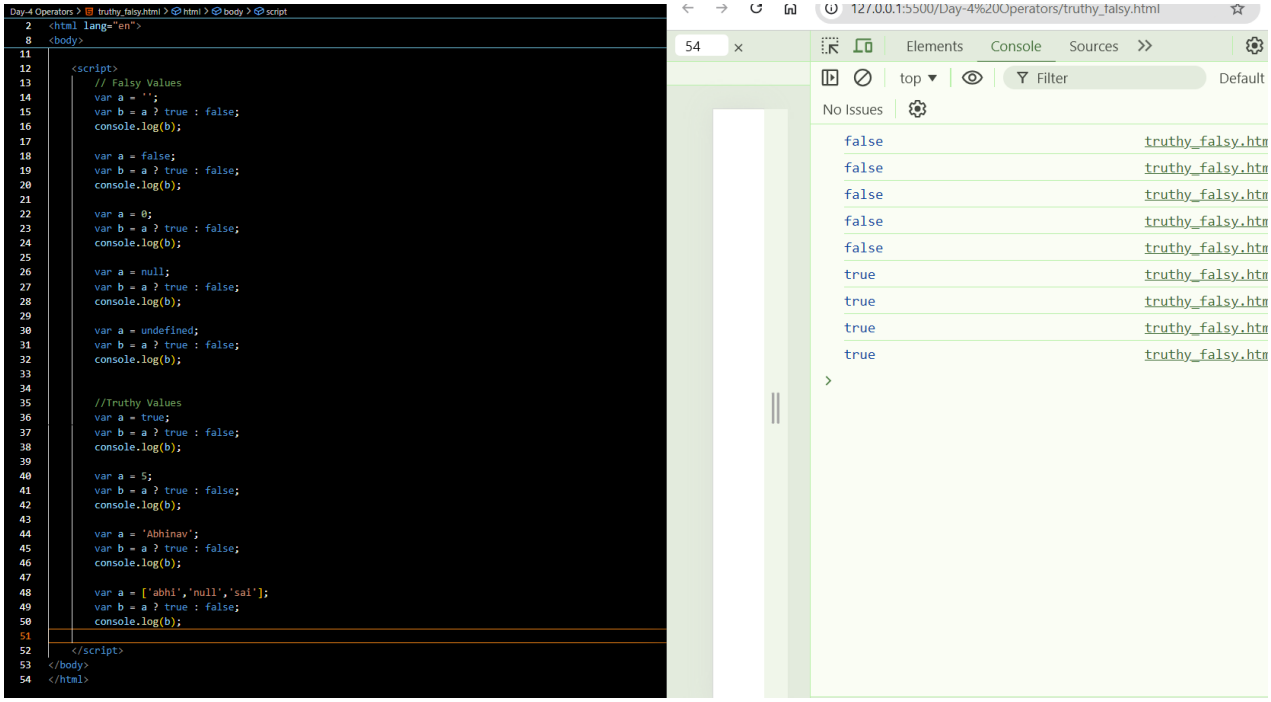
1. **false**: The boolean value false itself.
2. **0**: The number zero.
3. **""**: Empty string.
4. **null**: The absence of any value.
5. **undefined**: A variable that has not been assigned a value or a property that does not exist.
6. **NaN**: Not-a-Number.

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:

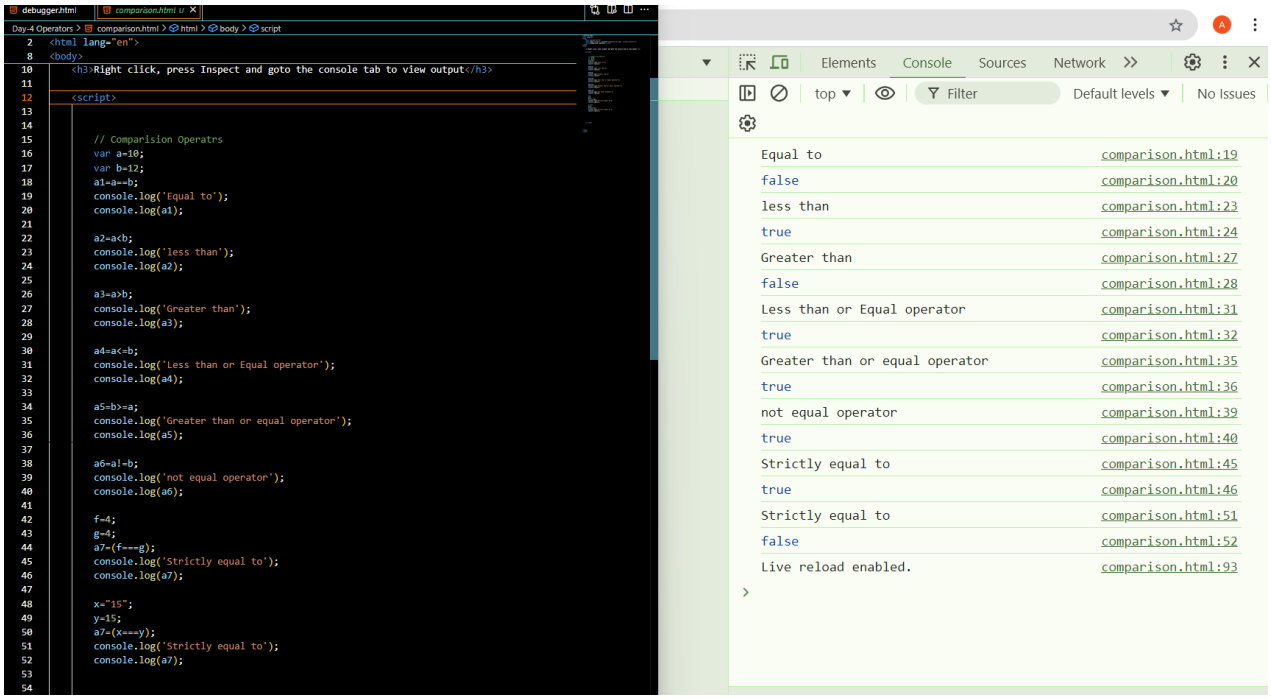


```
<html lang="en">
<body>

    <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:

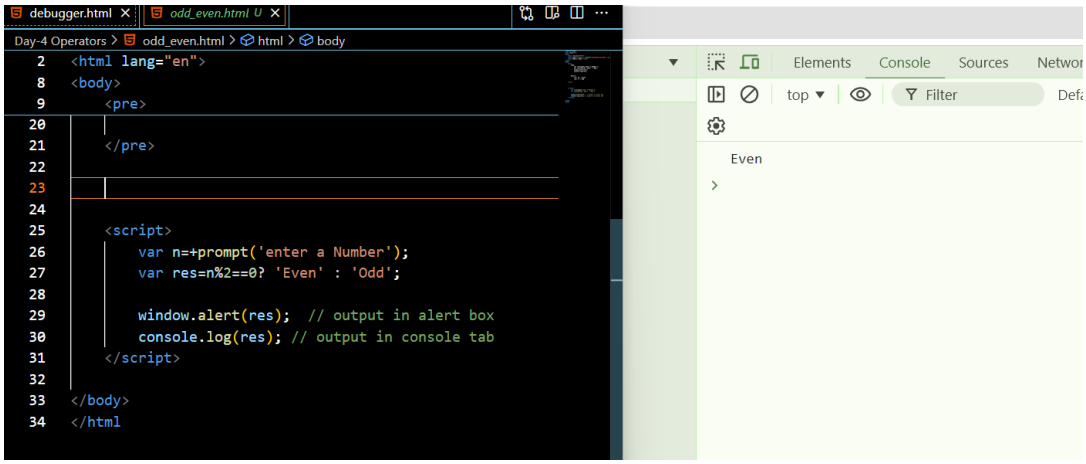
1. Write a JavaScript script that compares two variables using different comparison operators (==, ===, !=, !==, >, <, >=, <=) and prints the results.



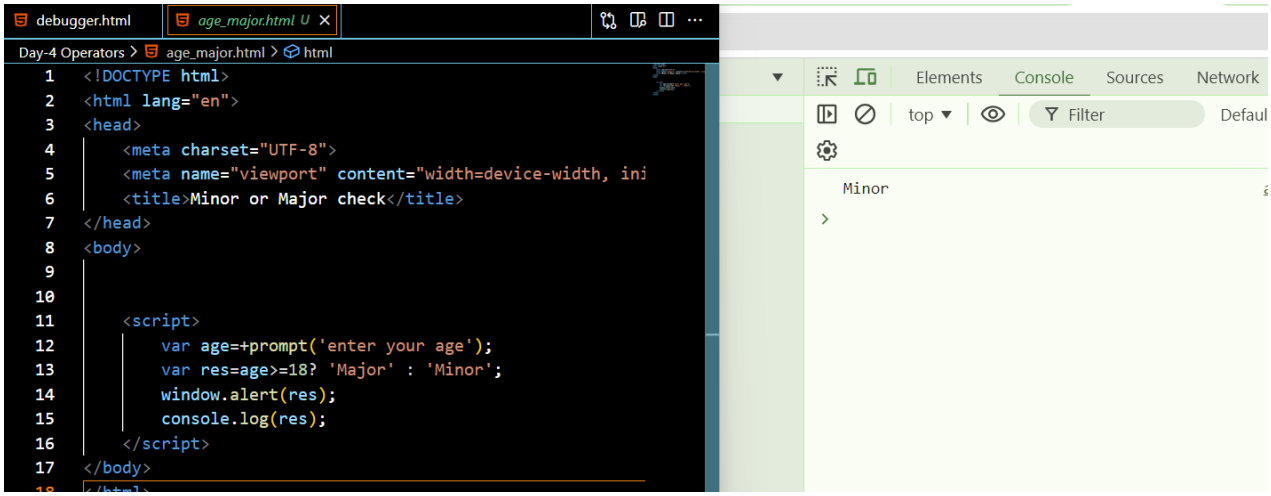
```
<html lang="en">
<body>
    <h3>Right click, press Inspect and goto the console tab to view output</h3>
    <script>


        // 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
```

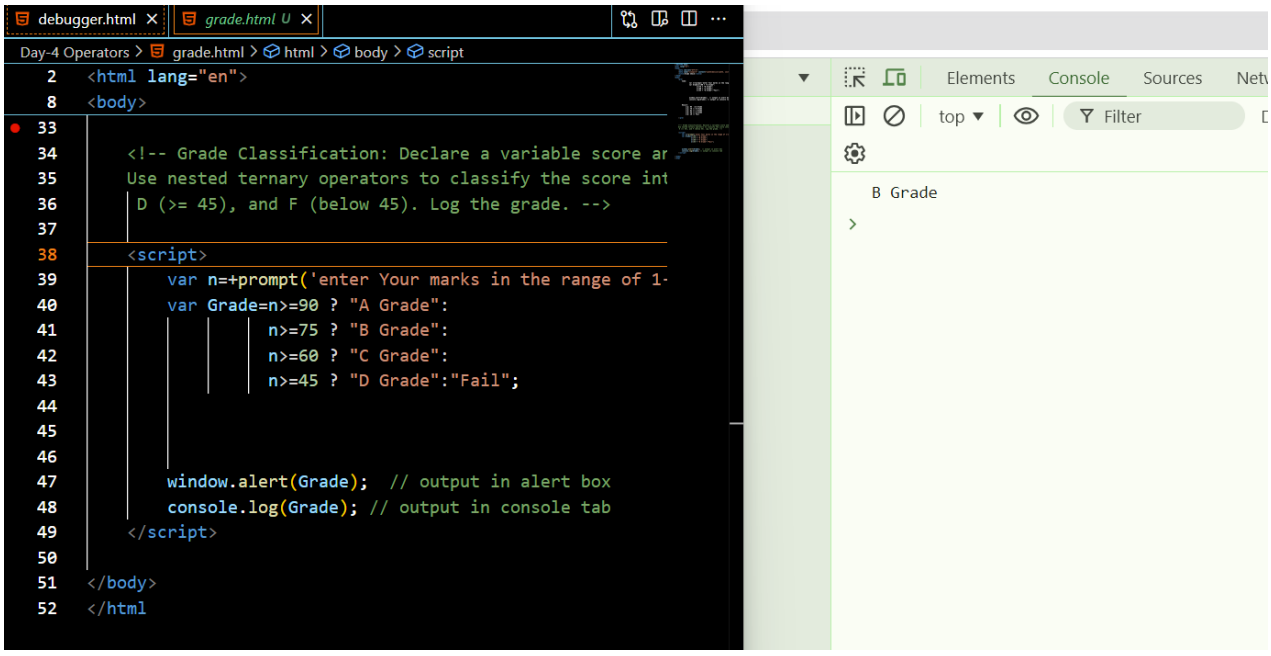
2. Write a JavaScript script that uses the ternary operator to determine if a number is even or odd.



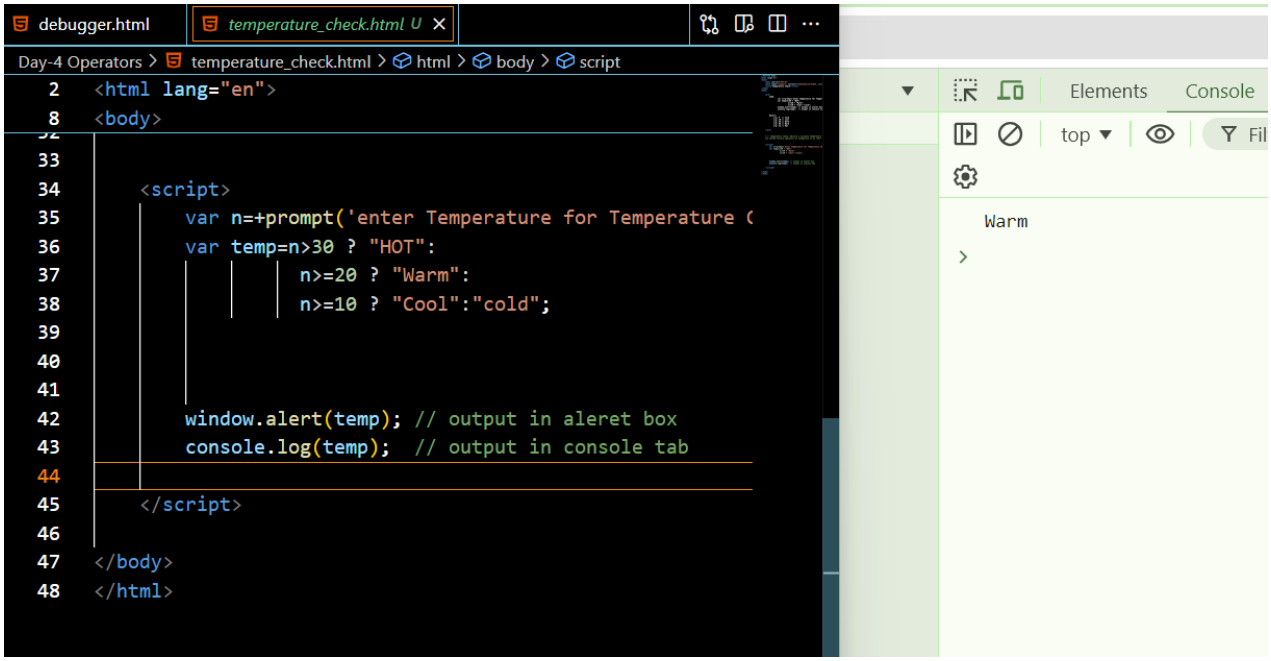
3. Expand the script to include a ternary operation that checks if a user is an adult (18+) or a minor.



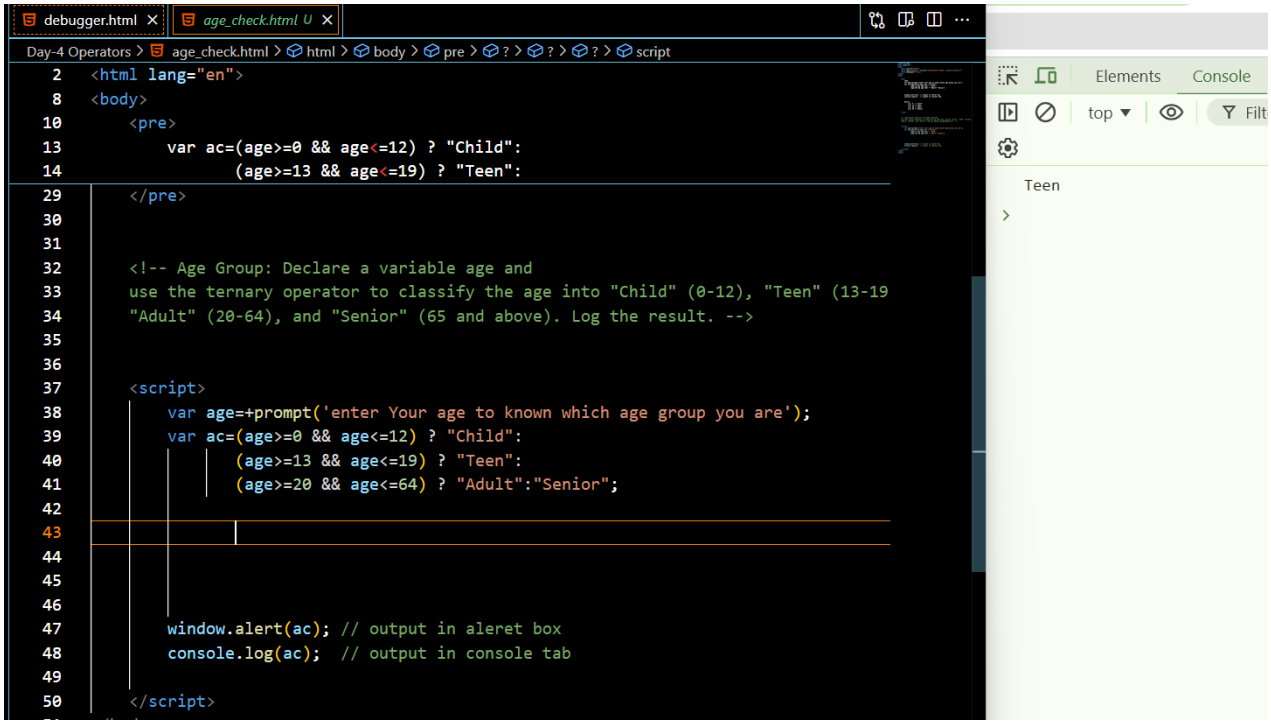
4. Grade Classification: Declare a variable score and set it to a value between 0 and 100. Use nested ternary operators to classify the score into grades: A (>= 90), B (>= 75), C (>= 60), D (>= 45), and F (below 45). Log the grade.



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.

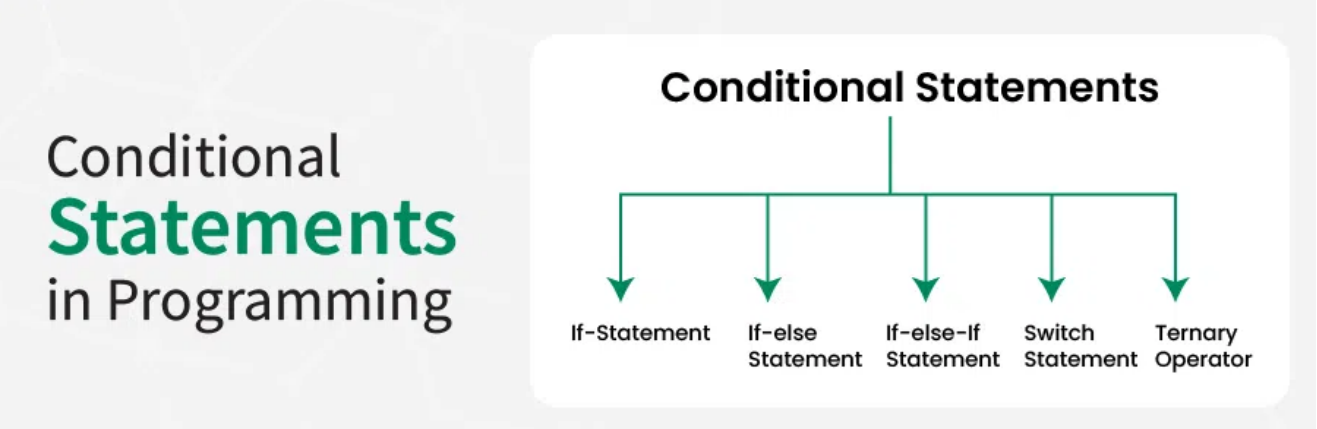


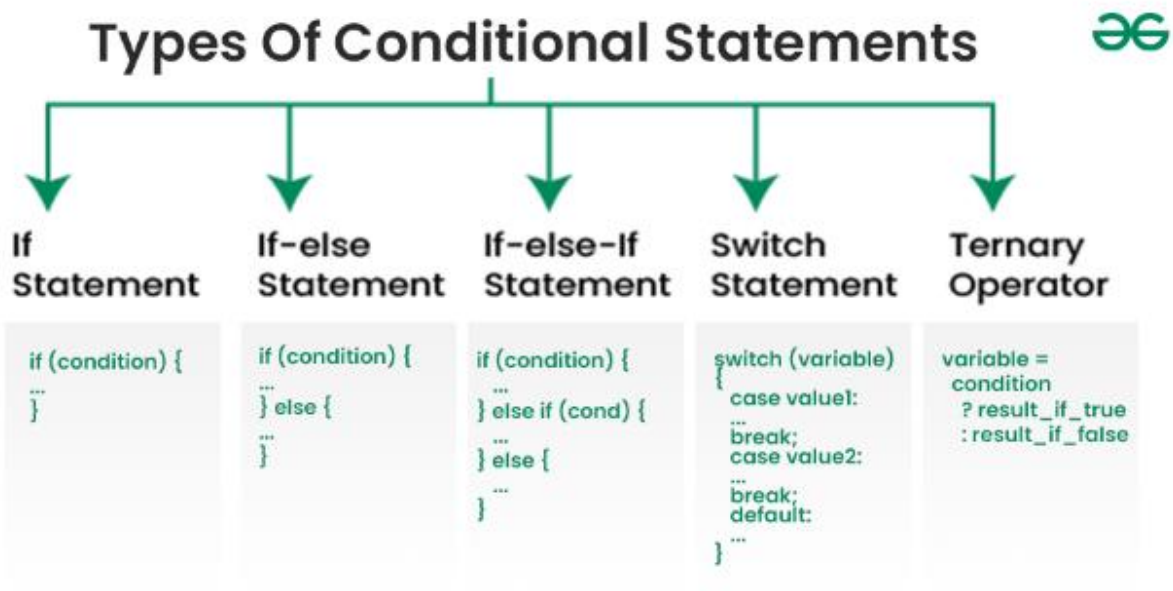
6. Age Group: Declare a variable age and use the ternary operator to classify the age into "Child" (0-12), "Teen" (13-19), "Adult" (20-64), and "Senior" (65 and above). Log the result.



Conditional Statements

Conditional statements in programming are used to **control the flow of a program** based on certain conditions. These statements allow the execution of different code blocks depending on whether a specified condition evaluates to true or false, providing a fundamental mechanism for **decision-making** in algorithms. In this article, we will learn about the basics of Conditional Statements along with their different types.





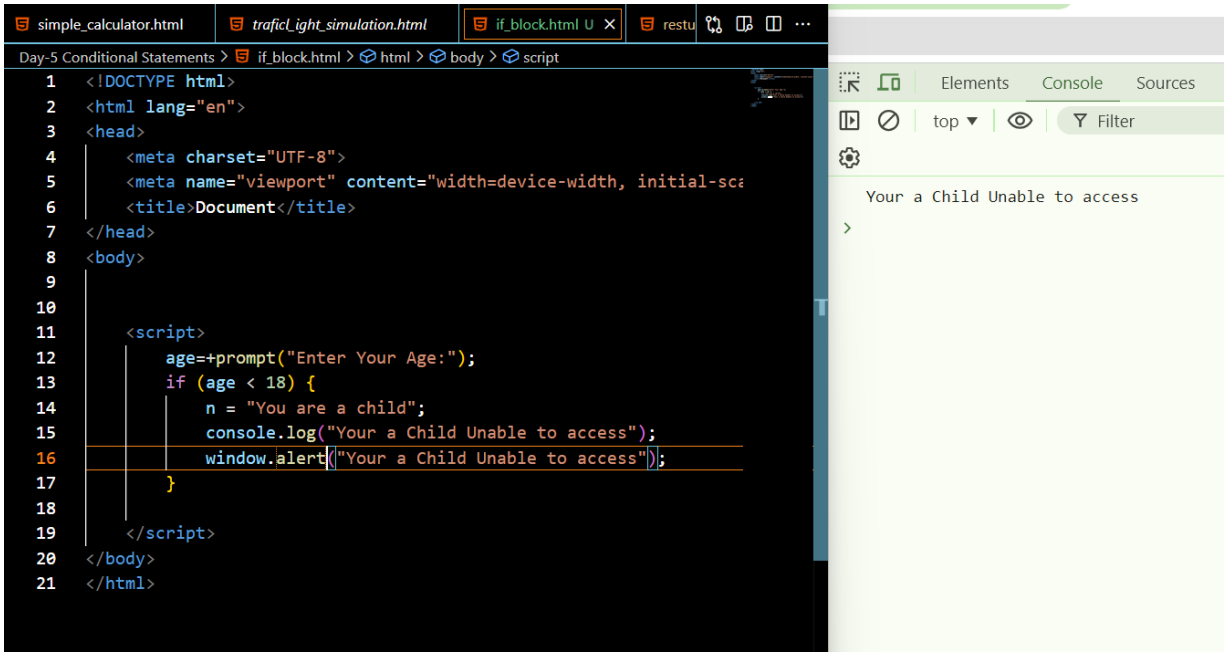
1. if Statement:

The **if** statement executes a block of code if a specified condition is true.

Syntax:

```
if (condition) {
    // Code to execute if condition is true
}
```

Example:



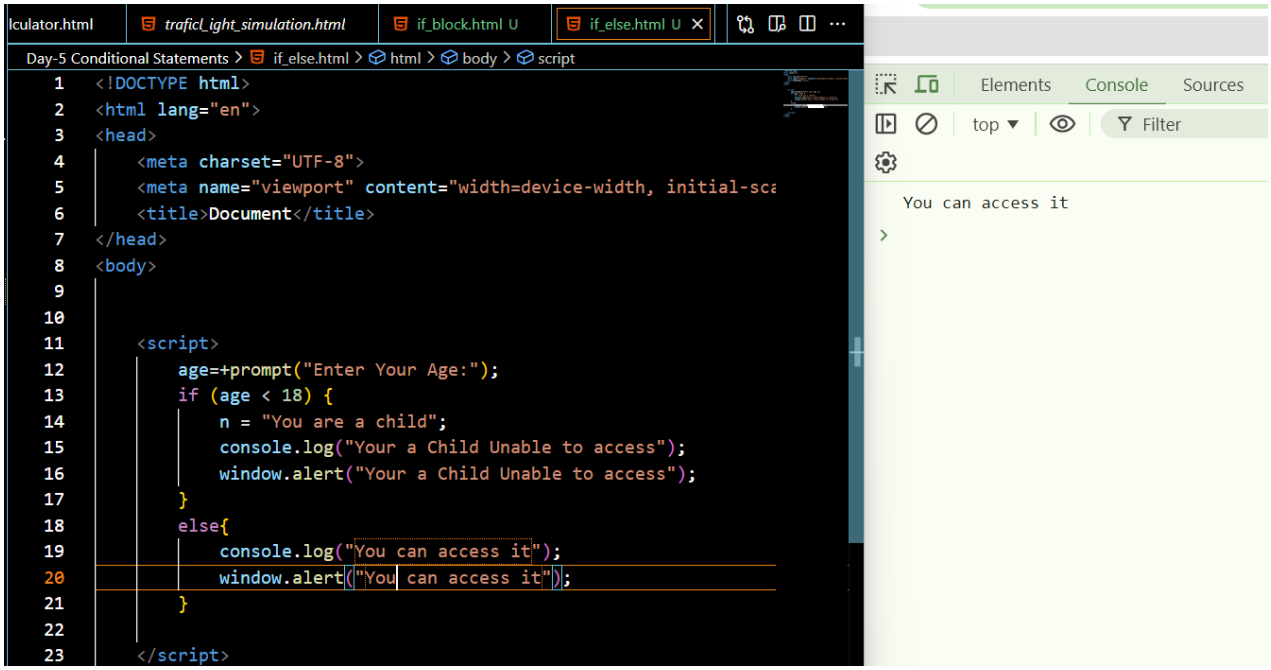
2. if...else Statement:

The **if...else** statement executes one block of code if a specified condition is true and another block if the condition is false.

Syntax:

```
if (condition) {
   // Code to execute if condition is true
} else {
   // Code to execute if condition is false
}
```

Example:



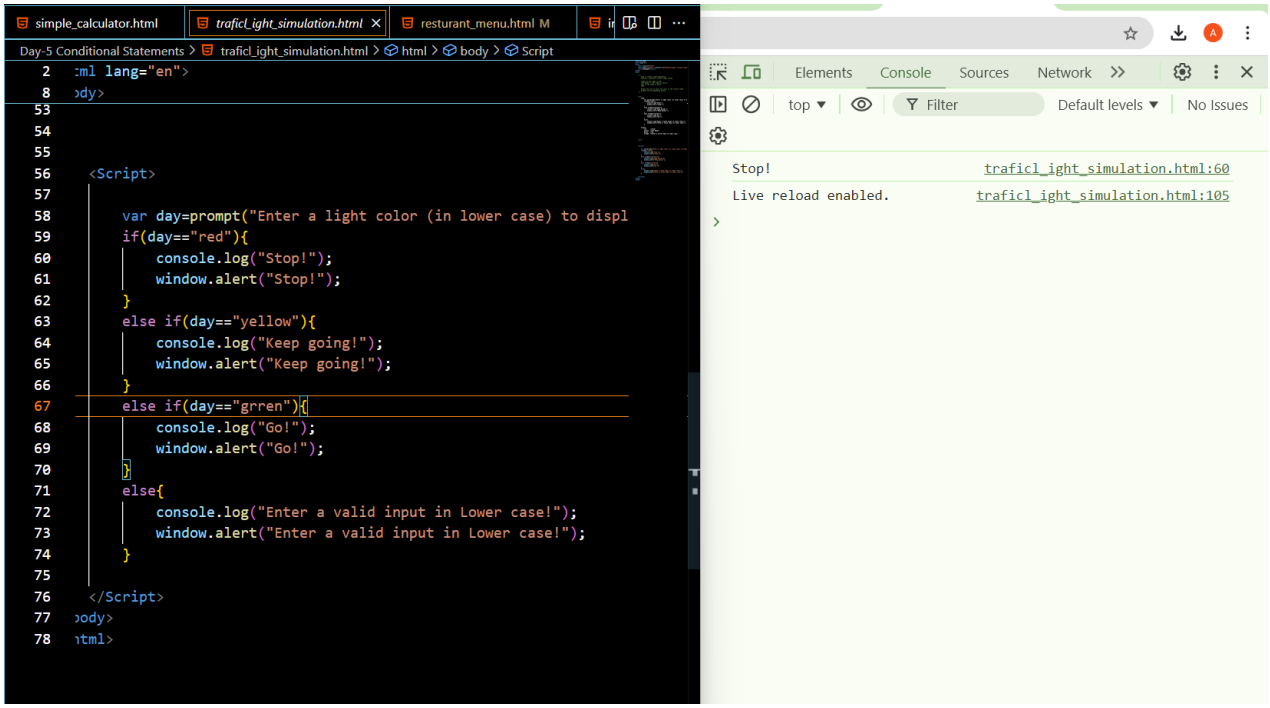
3)if...else if...else Statement:

The **if...else if...else** statement allows you to specify multiple conditions and execute different code blocks based on the outcome of those conditions.

Syntax:

```
if (condition1) {
   // Code to execute if condition1 is true
} else if (condition2) {
   // Code to execute if condition2 is true
} else {
   // Code to execute if none of the conditions are true
}
```

Example:



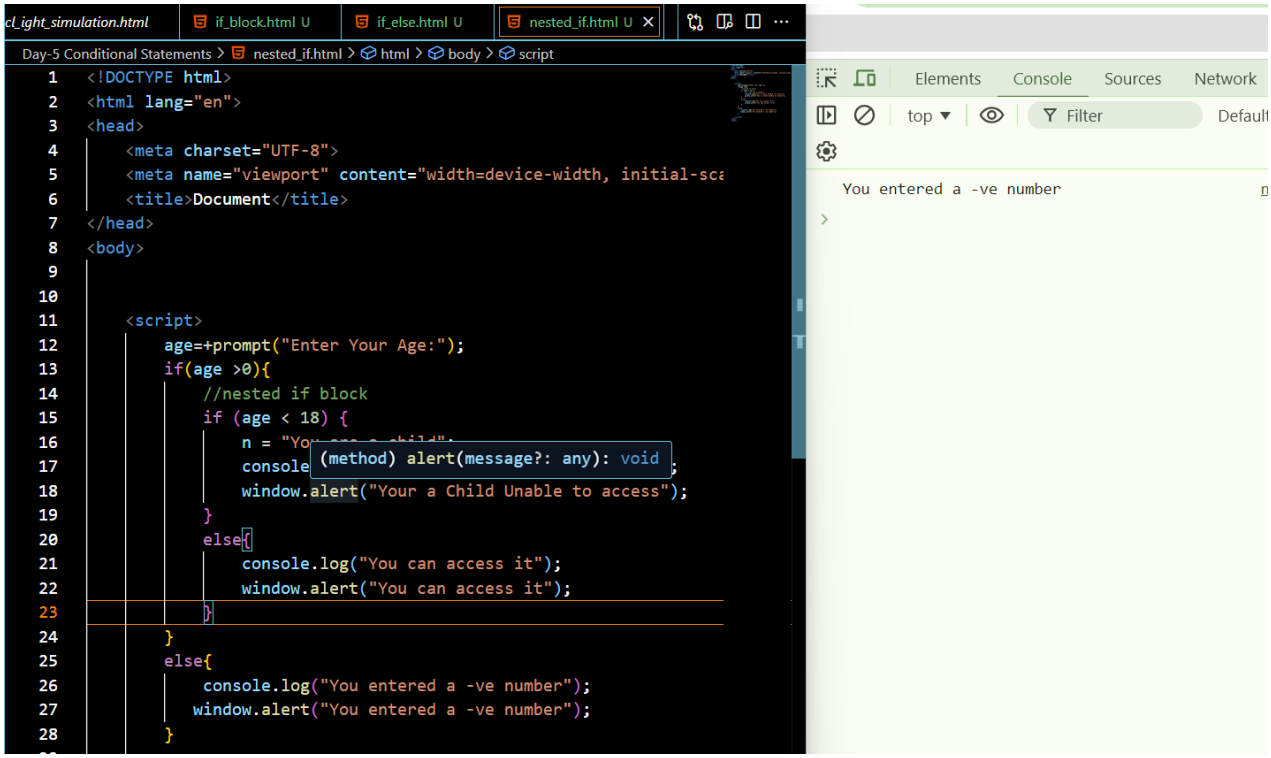
4)Nested if:

You can have if statements inside if statements, this is called a nested if.

Syntax

```
if condition1 {
  // code to be executed if condition1 is true
 if condition2 {
    // code to be executed if both condition1 and condition2 are true
 }
}
```

Example:



Switch statements

A switch statement in JavaScript is a control flow statement that allows you to execute a block of code among many options based on the value of an expression.

Key Points

1. **Expression Evaluation**: The **expression** inside the switch statement is evaluated once.

2. **Case Matching**: The result of the expression is compared with the values specified in each **case** clause using strict equality (**===**).

3. **Code Execution**: If a match is found, the code block associated with that **case** is executed.

4. **Break Statement**: The **break** statement is used to terminate the switch statement. If omitted, execution will continue to the next **case** clause (fall-through behavior).

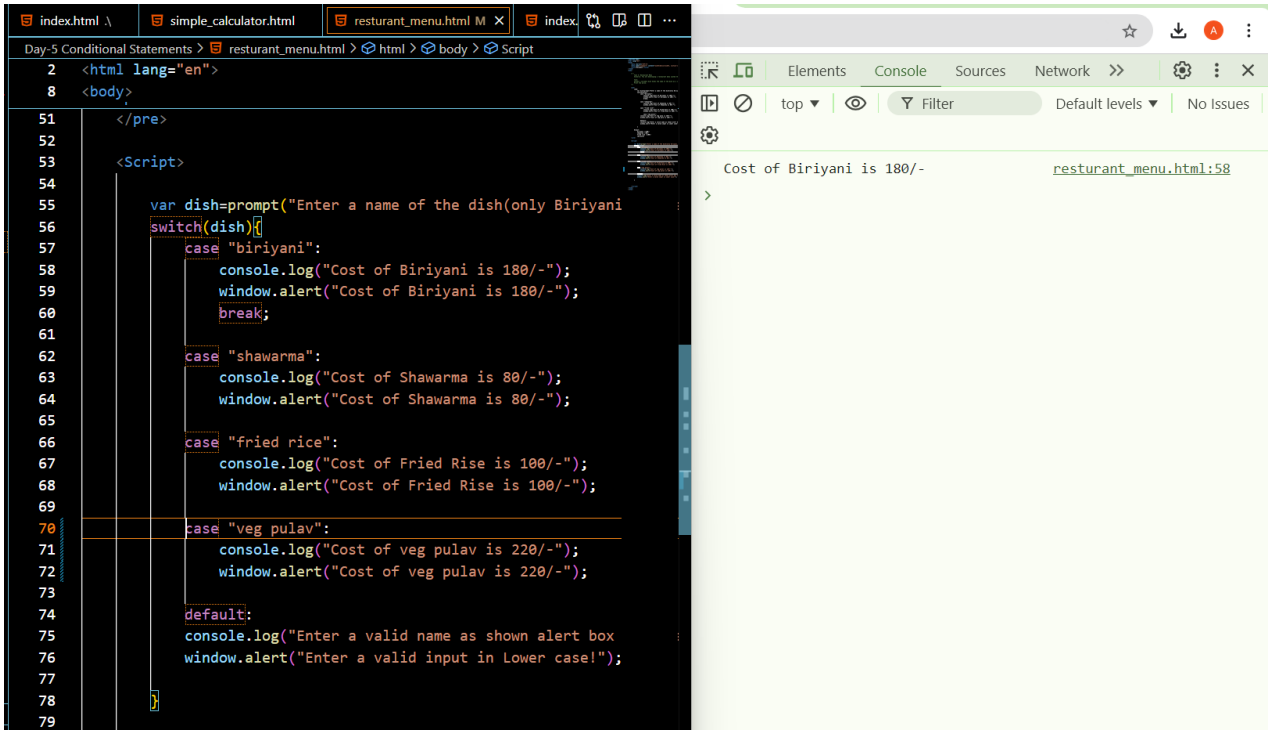
5. **Default Case**: The **default** clause is optional and executes if no matching **case** is found. It acts like the **else** in an if-else structure.

Syntax:

```
switch (expression) {
  case value1:
    // Code to run if expression === value1
    break;
  case value2:
    // Code to run if expression === value2
    break;
```

```
  // More cases...
  default:
    // Code to run if no case matches
}
```

Example:



TASKS

Task 1: Day of the Week Message

Scenario: Develop a webpage that displays a special message based on the current day of the week.

“Start your week strong!” for Monday.

“Keep going!” for Tuesday.

“Halfway there!” for Wednesday.

“Almost the weekend!” for Thursday.

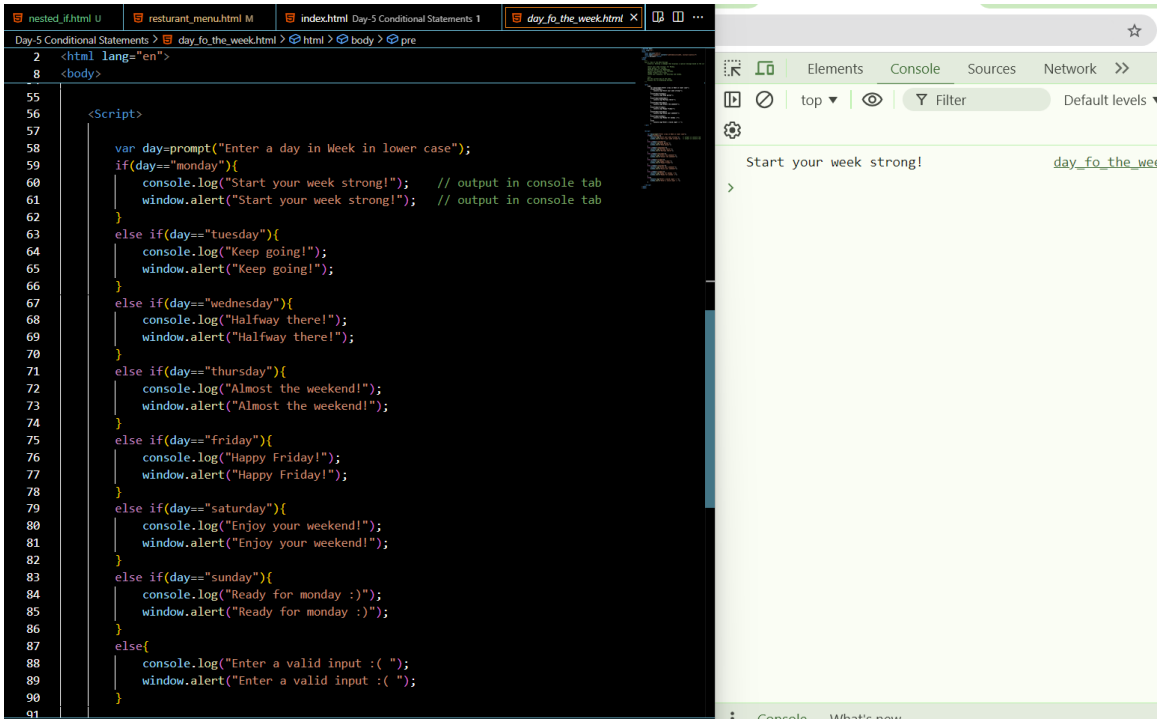
“Happy Friday!” for Friday.

“Enjoy your weekend!” for Saturday and Sunday.

Task:

Get the current day of the week.

Display the corresponding message.



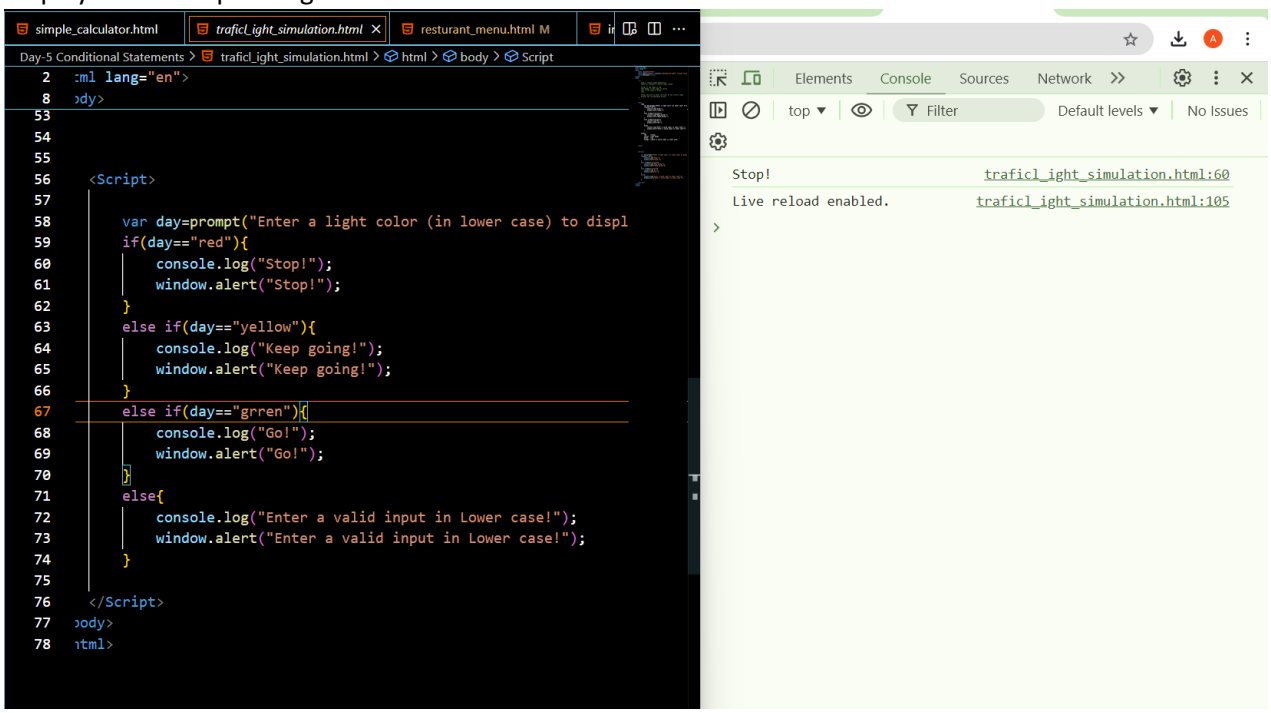
Task 2: Traffic Light Simulation

Scenario: Simulate a traffic light system.

“Stop” if the light is red.
“Get Ready” if the light is yellow.
“Go” if the light is green.

Task:

Prompt the user to enter the color of the traffic light.
Display the corresponding action.

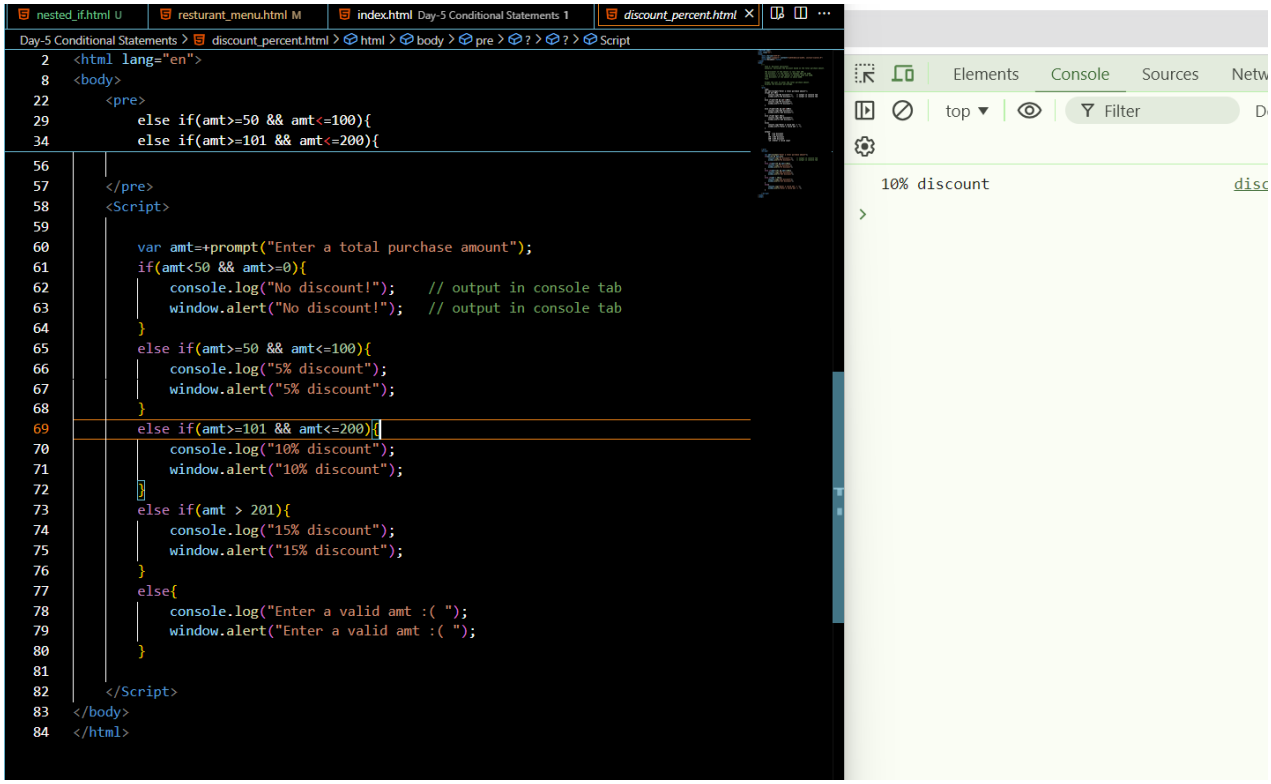


Task 3: Discount Calculator

Scenario: Calculate the discount based on the total purchase amount.

“No discount” if the amount is less than $50.
“5% discount” if the amount is between $50 and $100.
“10% discount” if the amount is between $101 and $200.
“15% discount” if the amount is above $200.

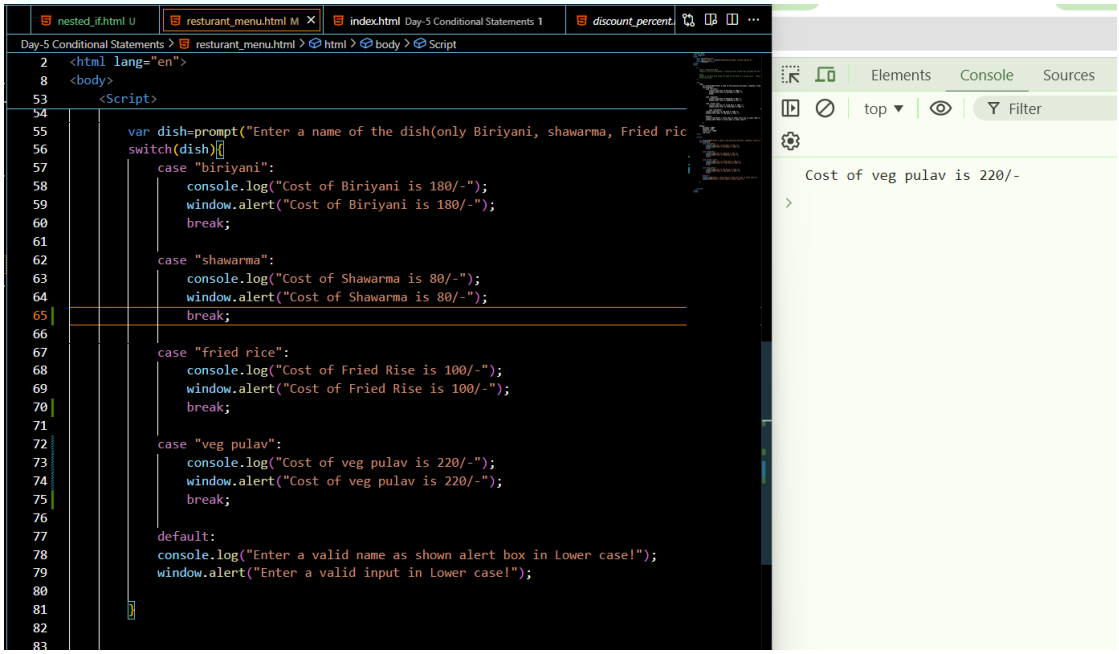
Task:
Prompt the user to enter the total purchase amount.
Display the discount percentage.



Task 4: Restaurant Menu

Scenario: You are developing a restaurant menu system that provides the price of a dish based on the dish name.

Task:
Assume a variable dish holds the name of the dish as a string (e.g., "Biriyani", "shawarma", "Fried rice", "veg pula").
Print the price.



Task 5: Simple Calculator

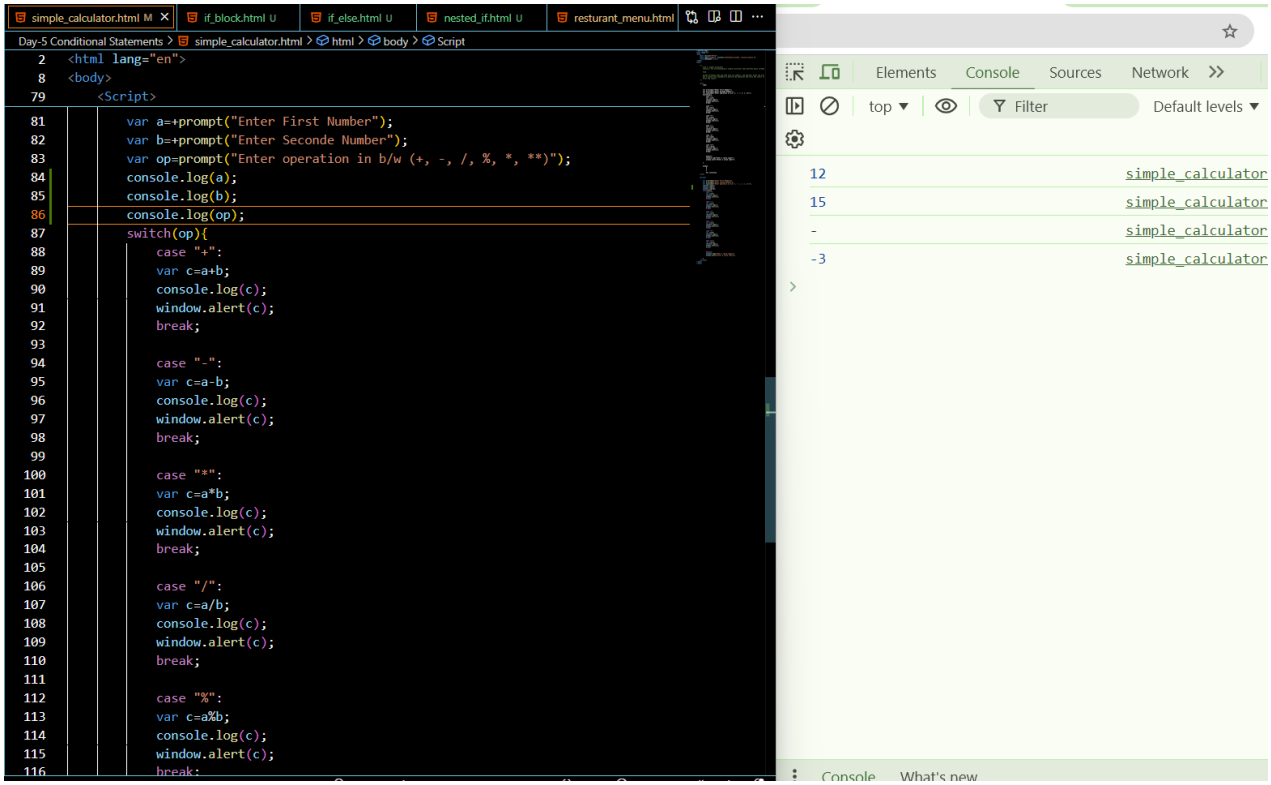
Scenario: You are developing a simple calculator that performs basic arithmetic operations.

Task:

Assume variables num1 and num2 hold two numbers, and operator holds the arithmetic operator as a string (e.g., “+”).

Use a switch case statement to perform the operation and store the result in a variable result.

Print the result.



LOOPS

Loops:

In JavaScript, the for loop is used for iterating over a block of code a certain number of times, or to iterate over the elements of an array.

For loop:

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

Syntax:

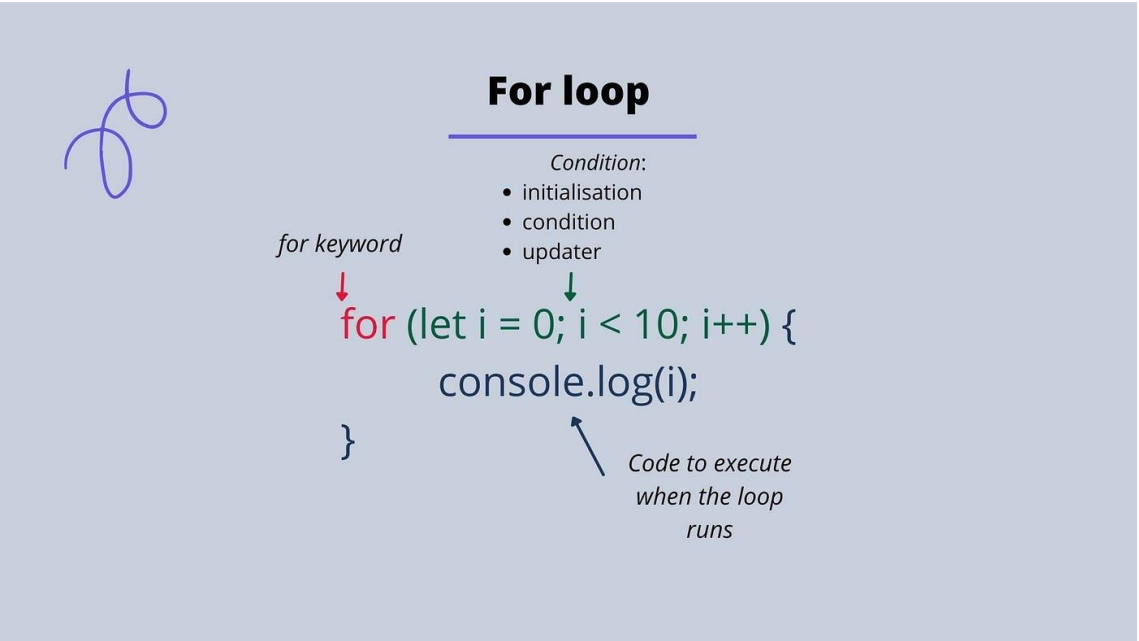
```
for (initialization; condition; increment)
{
   code to be executed
}
```

Example:

```
<script>
for (i=1; i<=5; i++)
{
document.write(i + "<br/>")
}
</script>
```

Output:

1
2
3
4
5

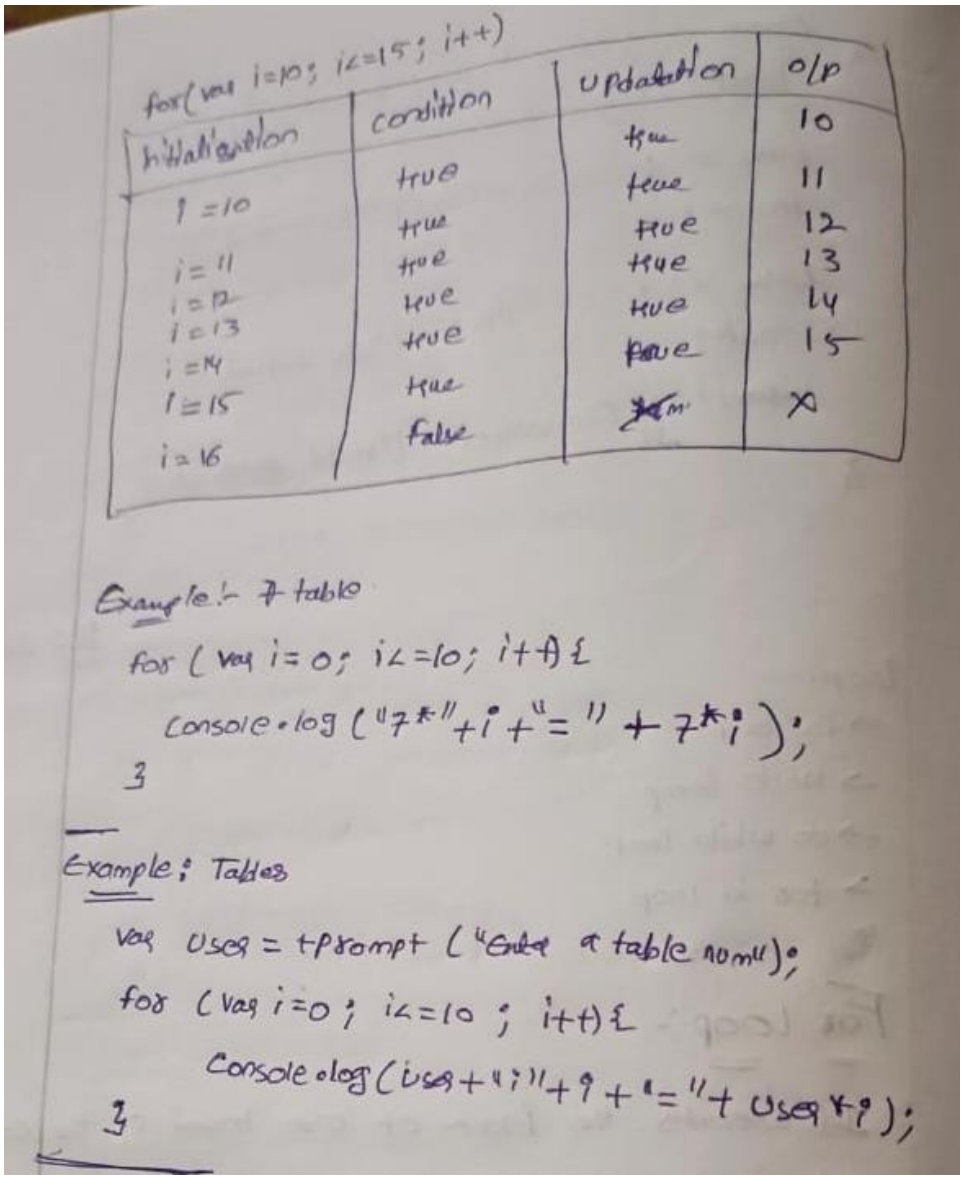


Nested For loop:

If we have loop inside loop, this is called a nested loop.

Example:

```
<script>
    for(var i=1; i<=5; i++){
        console.log("Table "+ i);
        for(var j=1; j<=10; j++){

            var x=i*j;
            console.log(i+ " * "+j+'='+x);


        }
    }
</script>
```



for(var i=10; i<=15; i++)

Initialization	condition	updation	o/p
i=10	true	true	10
i=11	true	true	11
i=12	true	true	12
i=13	true	true	13
i=14	true	true	14
i=15	true	true	15
i=16	false	~~true~~	X

Example:- 7 table

```
for (var i=0; i<=10; i++){
  console.log("7*"+i+"="+7*i);
}
```

Example: Tables

```
var useq = +prompt("Enter a table num");
for (var i=0; i<=10; i++){
    console.log(useq+"*"+i+"="+useq*i);
}
```

Example:- break the loop

```
for(i=1; i<=20; i++){
    if (i%2==0){
        console.log(i+" is even");
        break;
    }
    else{
        console.log(i);
    }
}
```

Example:- continue for skip the current iteration.

```
for (i=1; i<=2; i++){
    if (i%2==0){
        continue;
        console.log(i);
    }
    else{
        console.log(i)
    }
}
```

Example:- sum of even numbers

```
var count=0;
for (i=1; i<=10; i++){
    if (i%2==0){
        count = count+1;
    }
}
console.log(count);    //30
```

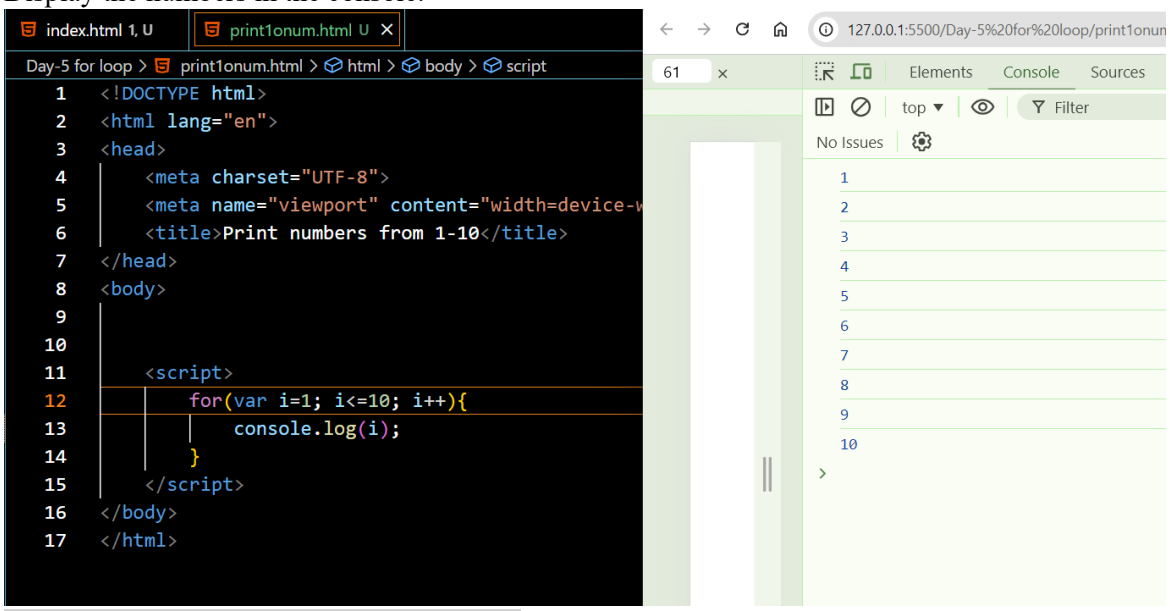
Tasks:

Task 1: Print Numbers from 1 to 10

Instructions:

Use a for loop to print numbers from 1 to 10.

Display the numbers in the console.



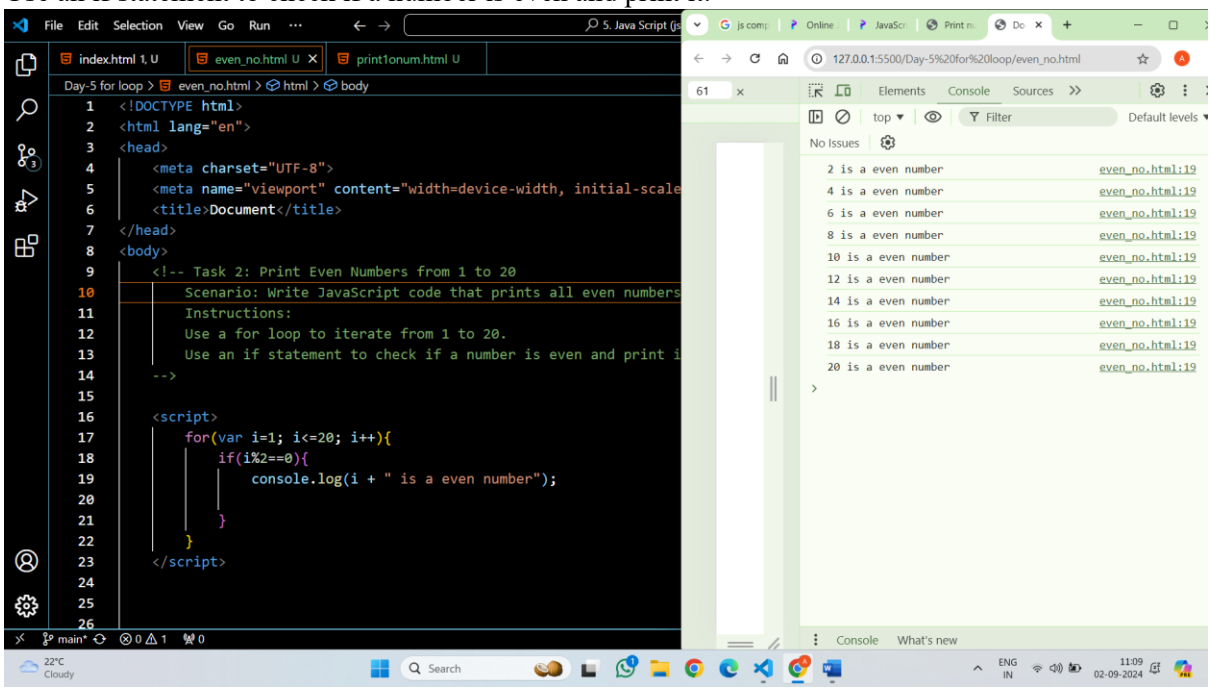
Task 2: Print Even Numbers from 1 to 20

Scenario: Write JavaScript code that prints all even numbers from 1 to 20.

Instructions:

Use a for loop to iterate from 1 to 20.

Use an if statement to check if a number is even and print it.



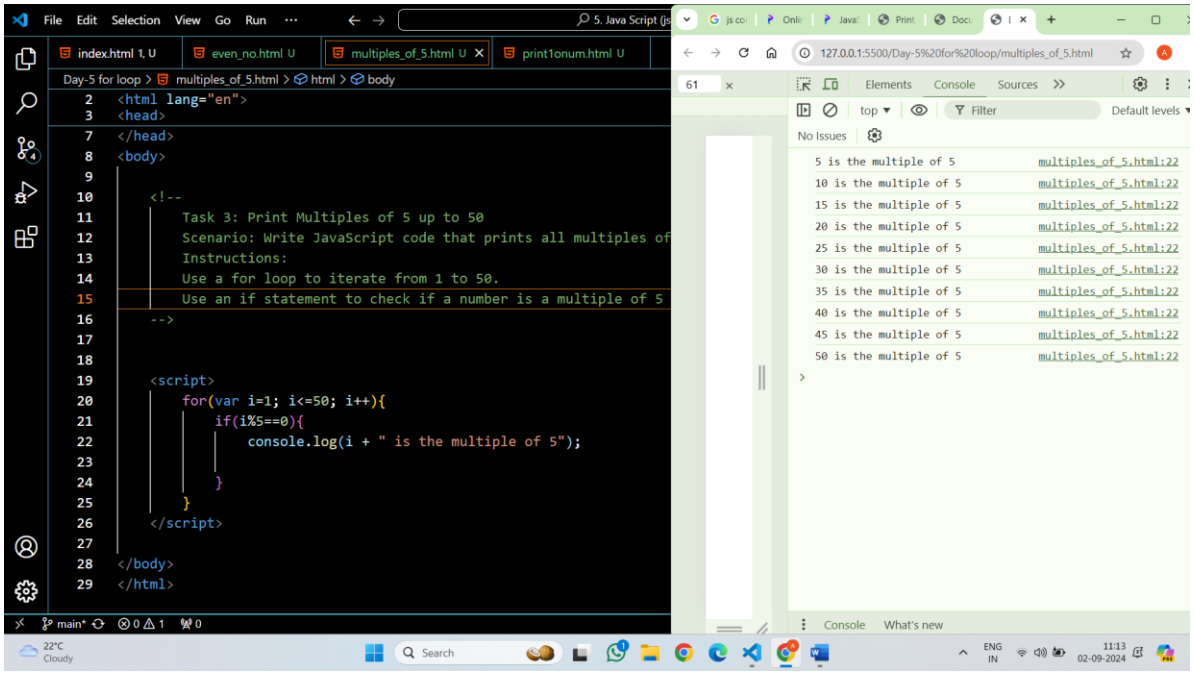
Task 3: Print Multiples of 5 up to 50

Scenario: Write JavaScript code that prints all multiples of 5 up to 50.

Instructions:

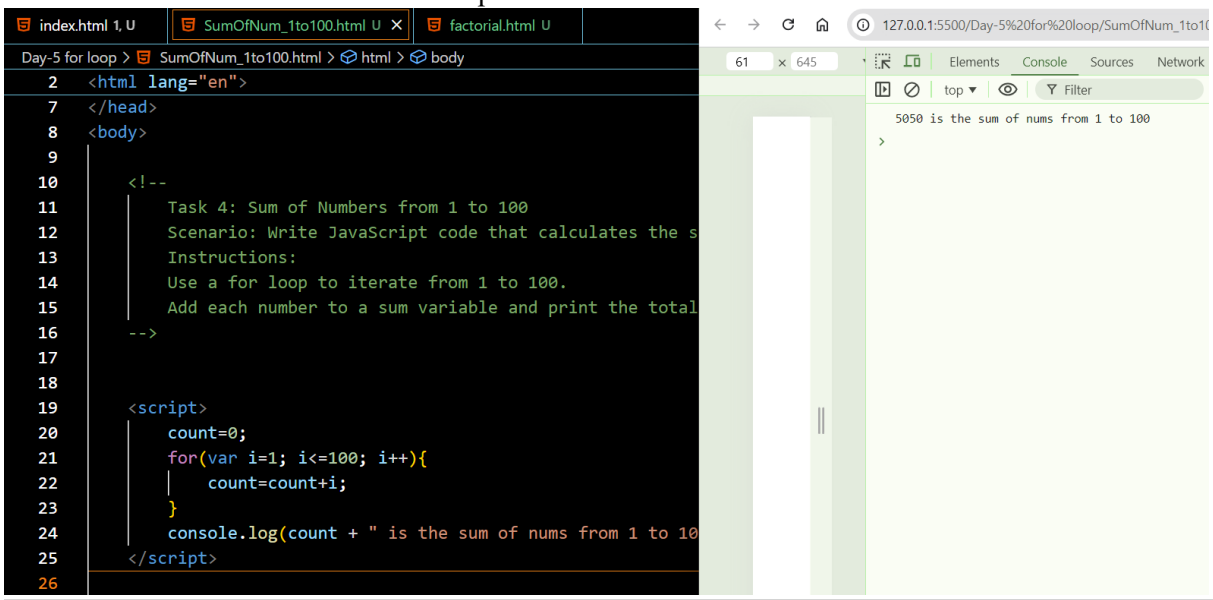
Use a for loop to iterate from 1 to 50.

Use an if statement to check if a number is a multiple of 5 and print it.



Task 4: Sum of Numbers from 1 to 100

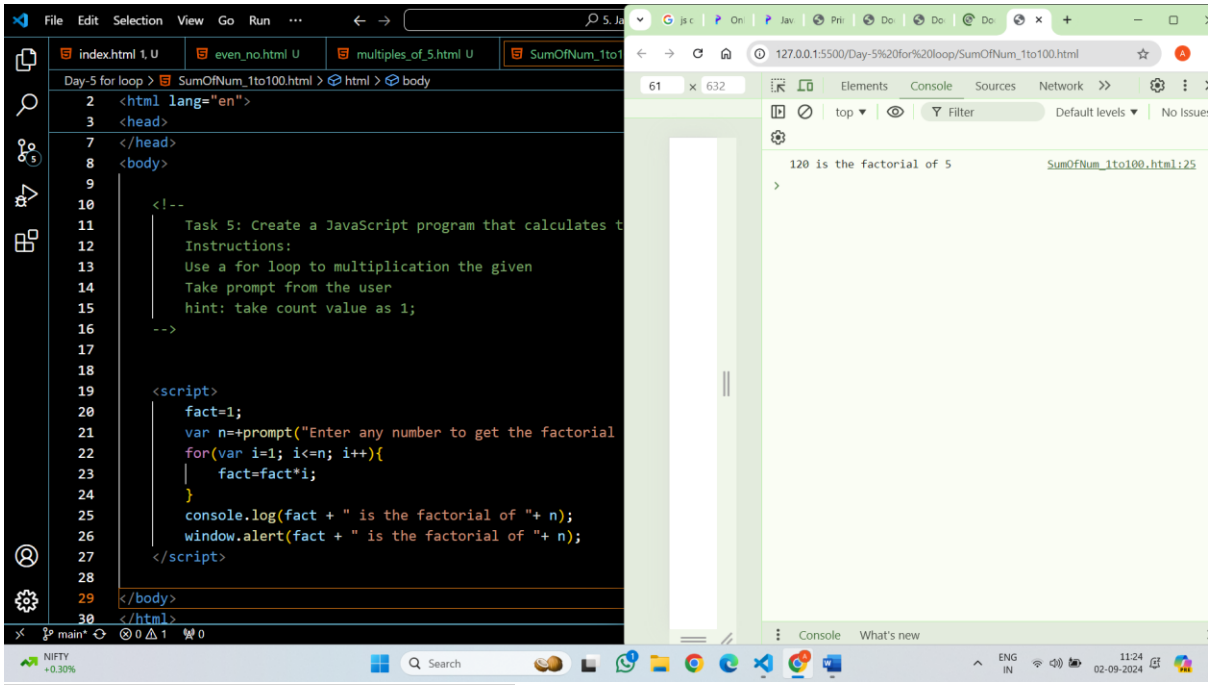
Scenario: Write JavaScript code that calculates the sum of numbers from 1 to 100.
Instructions:
Use a for loop to iterate from 1 to 100.
Add each number to a sum variable and print the total sum.



Task 5: Create a JavaScript program that calculates the factorial of a given number using a for loop.

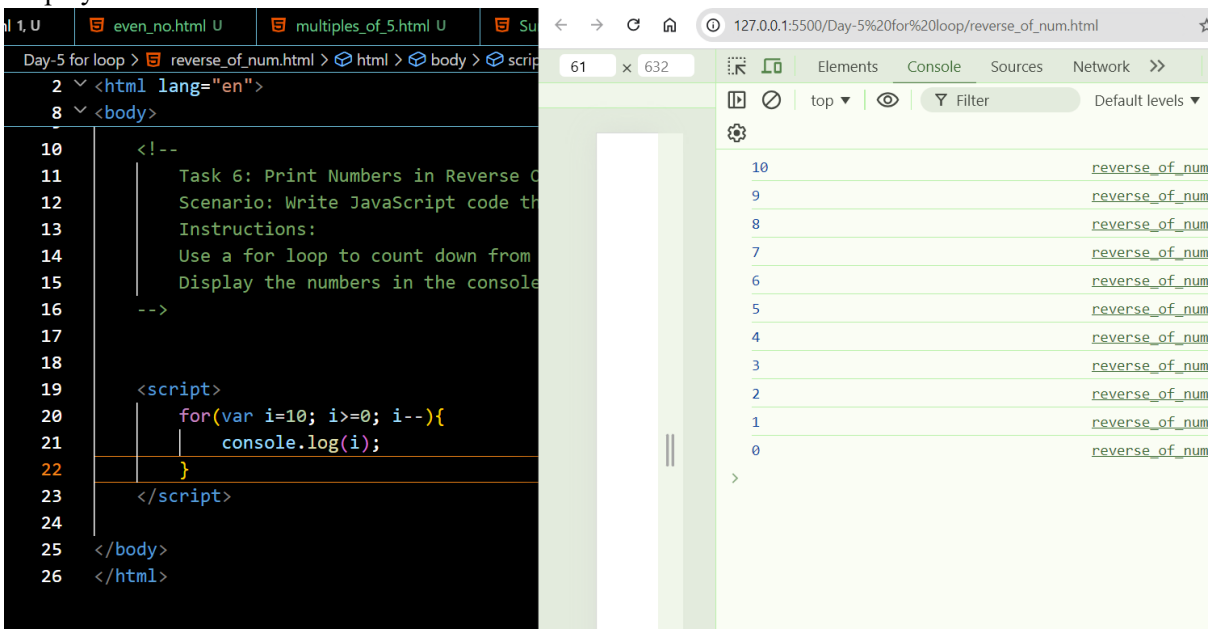
Instructions:

Use a for loop to multiplication the given
Take prompt from the user
hint: take count value as 1;



Task 6: Print Numbers in Reverse Order

Scenario: Write JavaScript code that prints numbers from 10 to 1 in reverse order.
Instructions:
Use a for loop to count down from 10 to 1.
Display the numbers in the console.



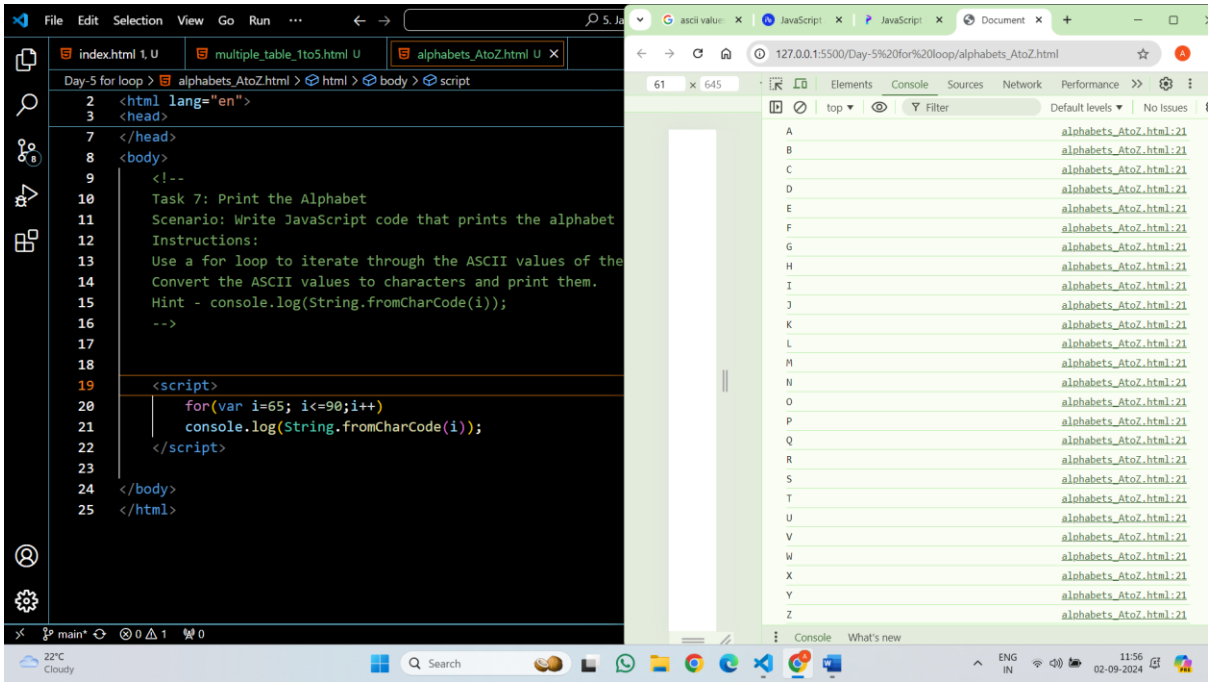
(optional)

Task 7: Print the Alphabet

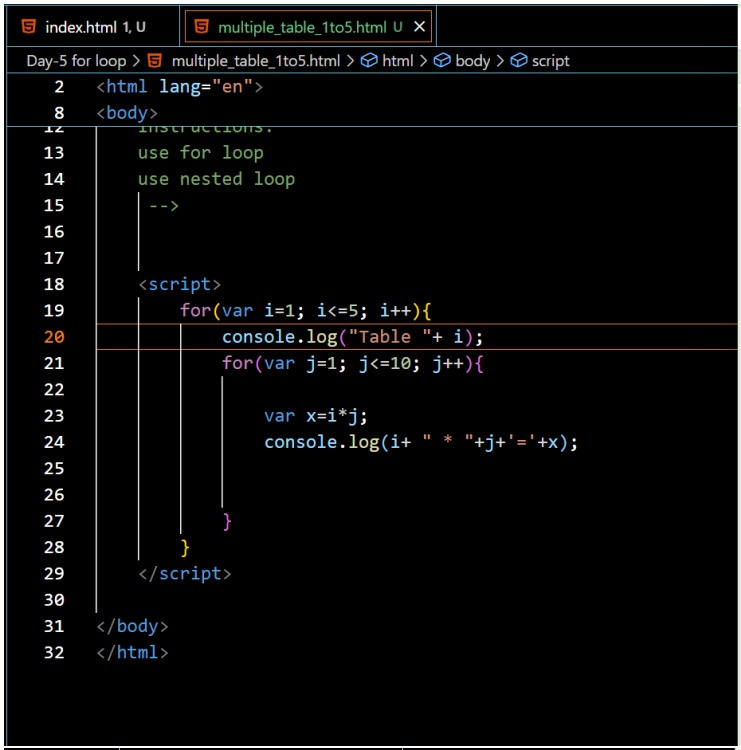
Scenario: Write JavaScript code that prints the alphabet from A to Z.
Instructions:
Use a for loop to iterate through the ASCII values of the letters A to Z.
Convert the ASCII values to characters and print them.
Hint - console.log(String.fromCharCode(i));

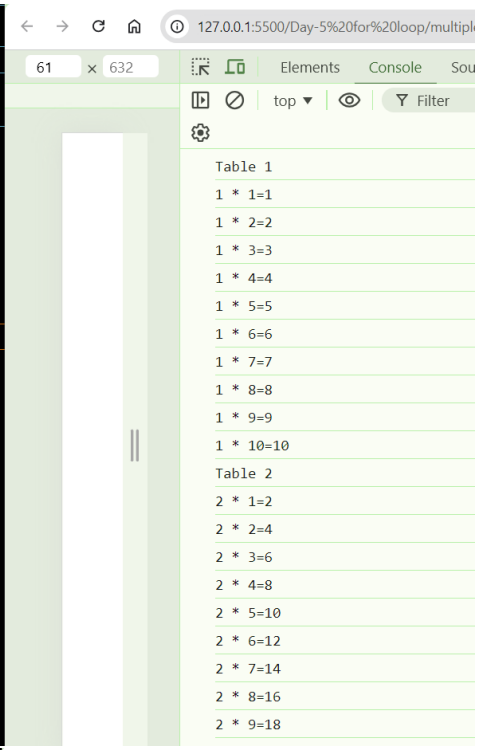
ASCII TABLE

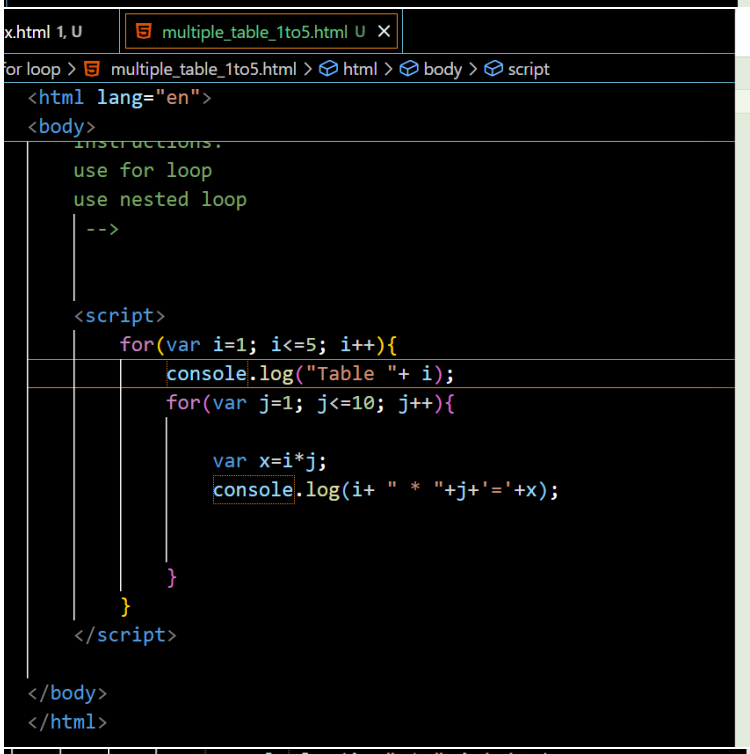
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

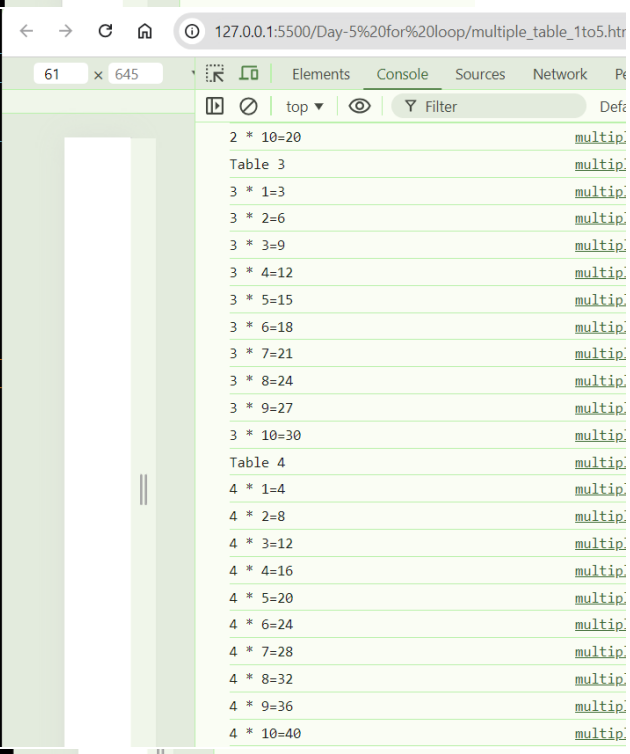


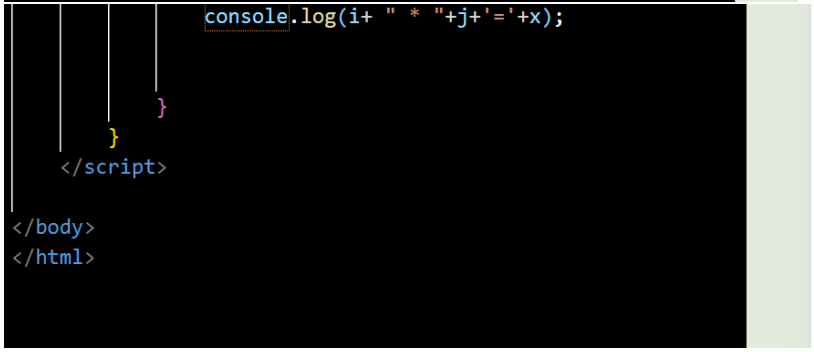
Task 8:Write a JavaScript script that uses nested loops to print a multiplication table for numbers 1 through 5.
Instructions:
use for loop
use nested loop


index.html 1, U
multiple_table_1to5.html U
Day-5 for loop > multiple_table_1to5.html > html > body > script
<html lang="en">
<body>
use for loop
use nested loop
-->
<script>
for(var i=1; i<=5; i++){
console.log("Table "+ i);
for(var j=1; j<=10; j++){
var x=i*j;
console.log(i+ " * "+j+'='+x);
}
}
</script>
</body>
</html>


127.0.0.1:5500/Day-5%20for%20loop/multipl
61 x 632
Elements
Console
top
Filter
Table 1
1 * 1=1
1 * 2=2
1 * 3=3
1 * 4=4
1 * 5=5
1 * 6=6
1 * 7=7
1 * 8=8
1 * 9=9
1 * 10=10
Table 2
2 * 1=2
2 * 2=4
2 * 3=6
2 * 4=8
2 * 5=10
2 * 6=12
2 * 7=14
2 * 8=16
2 * 9=18


multiple_table_1to5.html U
multiple_table_1to5.html > html > body > script
<html lang="en">
<body>
use for loop
use nested loop
-->
<script>
for(var i=1; i<=5; i++){
console.log("Table "+ i);
for(var j=1; j<=10; j++){
var x=i*j;
console.log(i+ " * "+j+'='+x);
}
}
</script>
</body>
</html>


127.0.0.1:5500/Day-5%20for%20loop/multiple_table_1to5.htr
61 x 645
Elements
Console
Sources
Network
top
Filter
2 * 10=20
Table 3
3 * 1=3
3 * 2=6
3 * 3=9
3 * 4=12
3 * 5=15
3 * 6=18
3 * 7=21
3 * 8=24
3 * 9=27
3 * 10=30
Table 4
4 * 1=4
4 * 2=8
4 * 3=12
4 * 4=16
4 * 5=20
4 * 6=24
4 * 7=28
4 * 8=32
4 * 9=36
4 * 10=40


console.log(i+ " * "+j+'='+x);
}
}
</script>
</body>
</html>

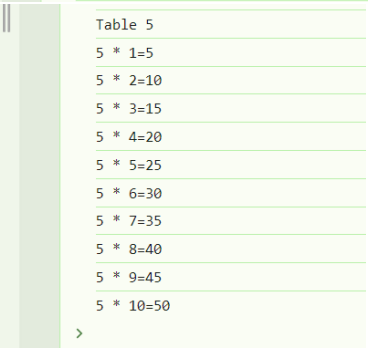


Table 5
5 * 1=5
5 * 2=10
5 * 3=15
5 * 4=20
5 * 5=25
5 * 6=30
5 * 7=35
5 * 8=40
5 * 9=45
5 * 10=50