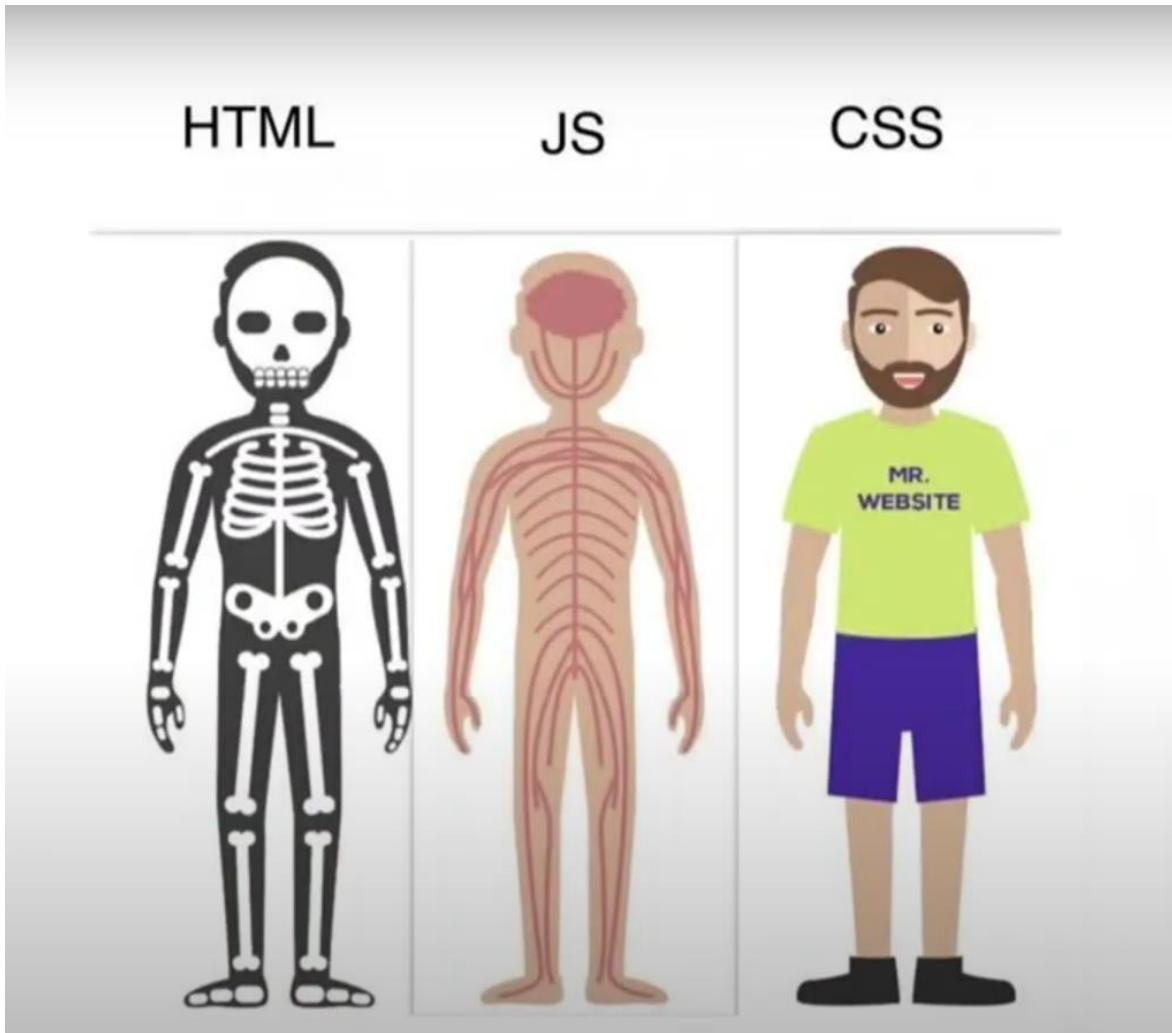


Java Script

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.



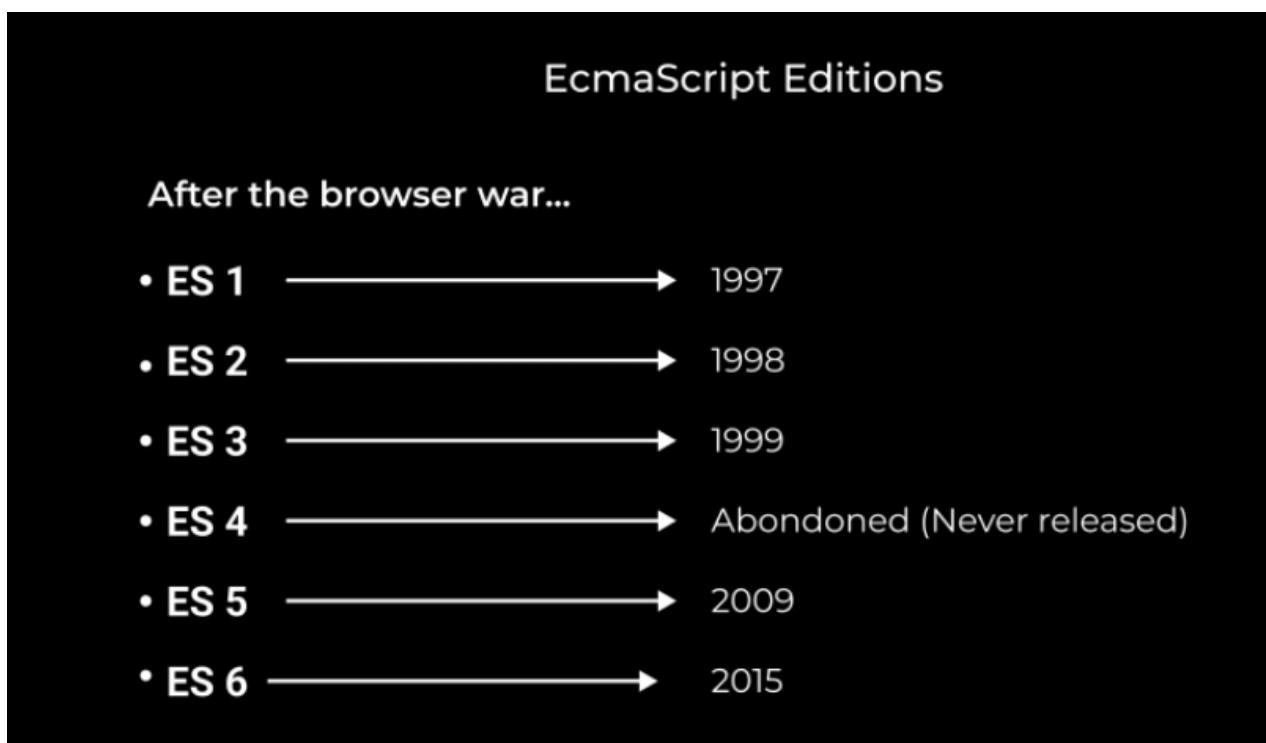
2. History of java script?

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.



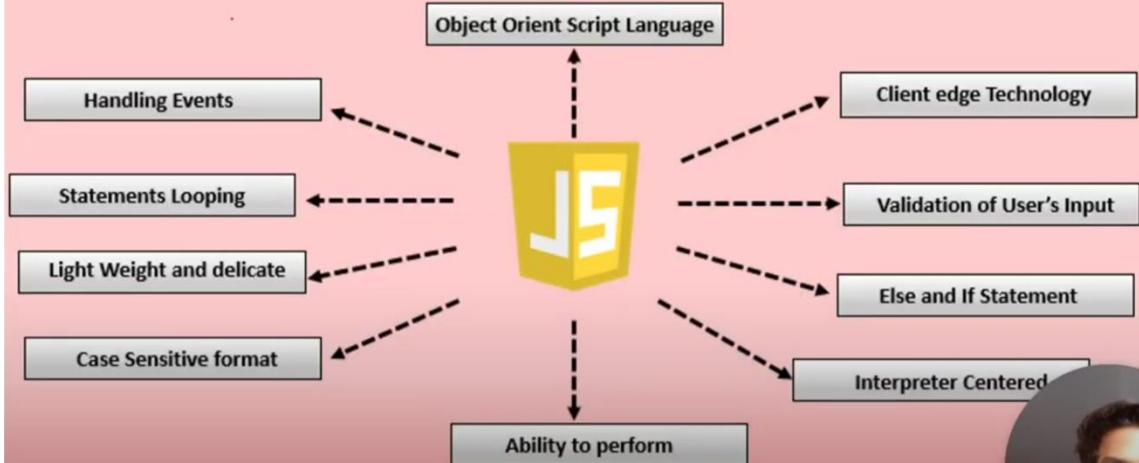
Originally, the name ECMAScript was just the formalization of JavaScript, but now languages like JScript and ActionScript are also based on the ECMAScript standard.

3. What is java script?

Java Script is a High level Programming Language that is primarily 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



➤ 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-threaded

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

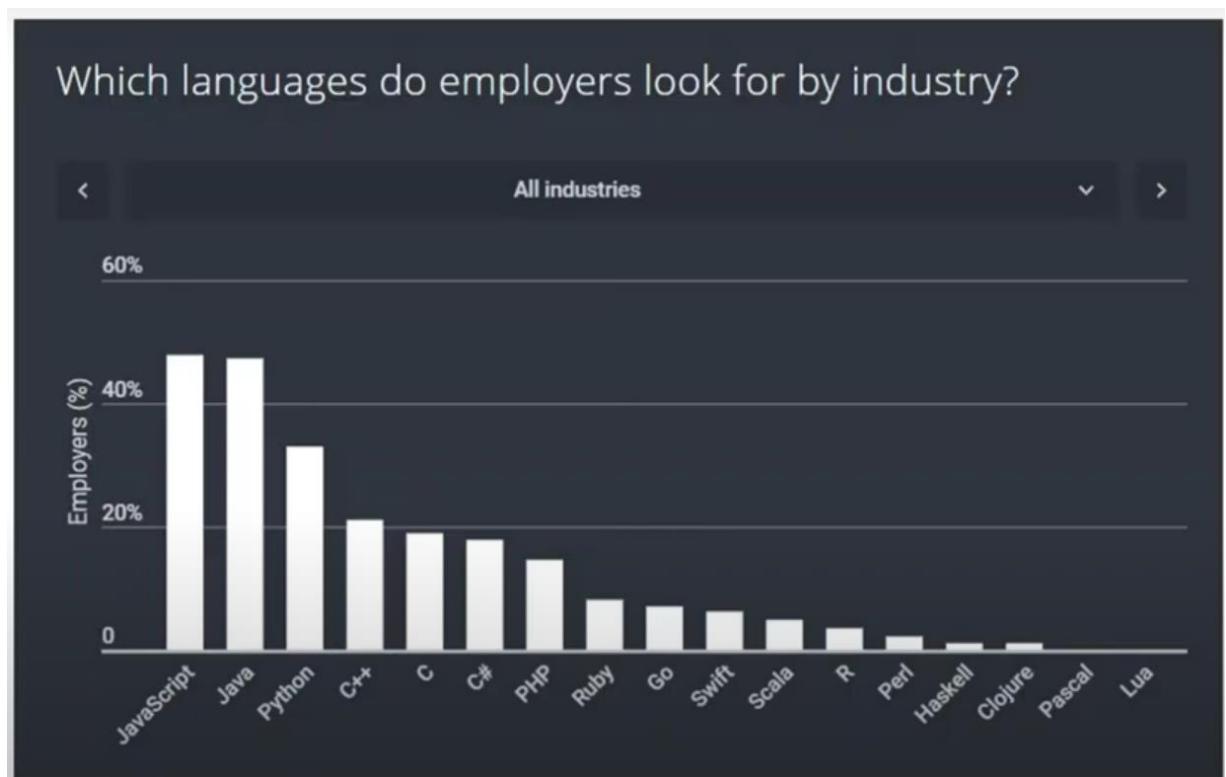
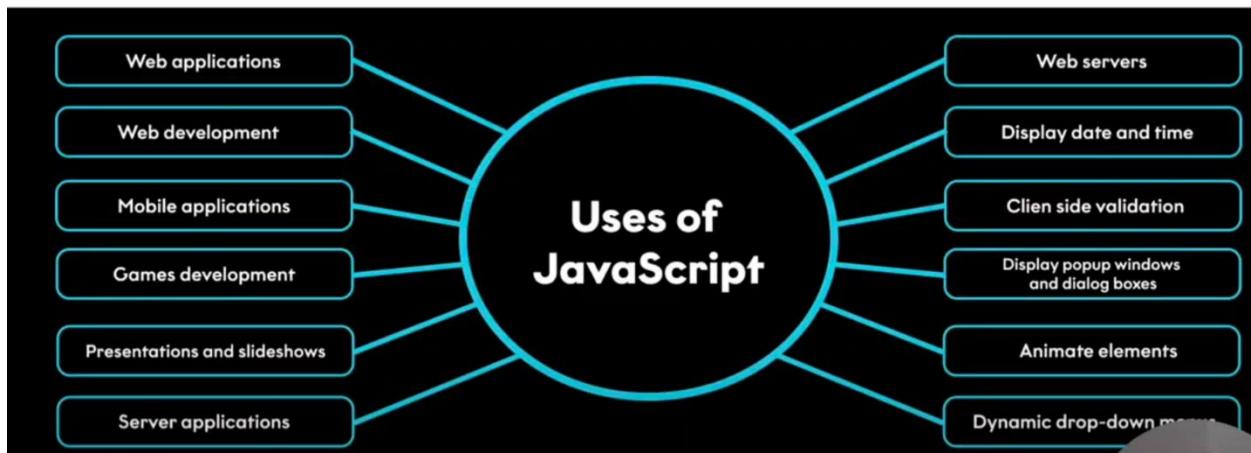
Synchronous means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

➤ Object 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.

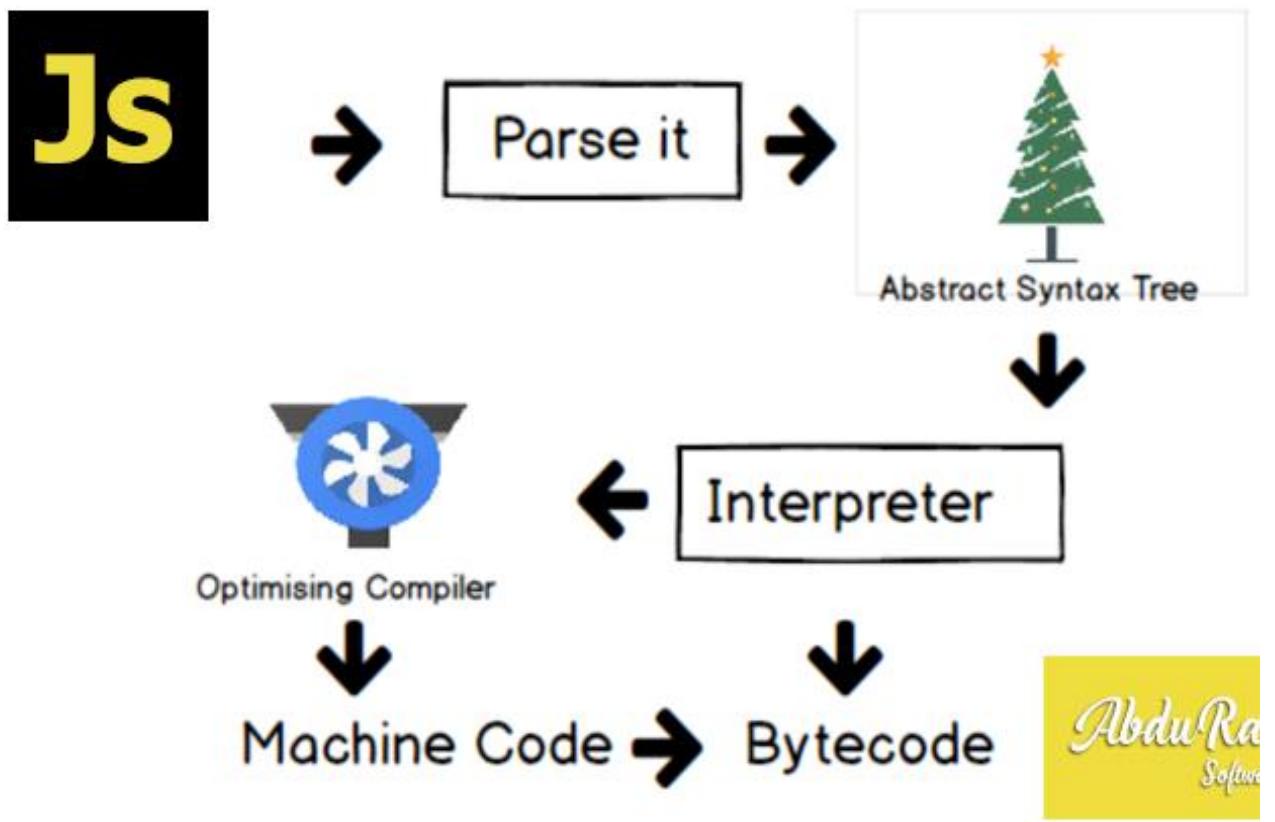
The team of developers that created the vanilla JavaScript is continuously working on it to improve it and make it more useful for the web-developers.

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

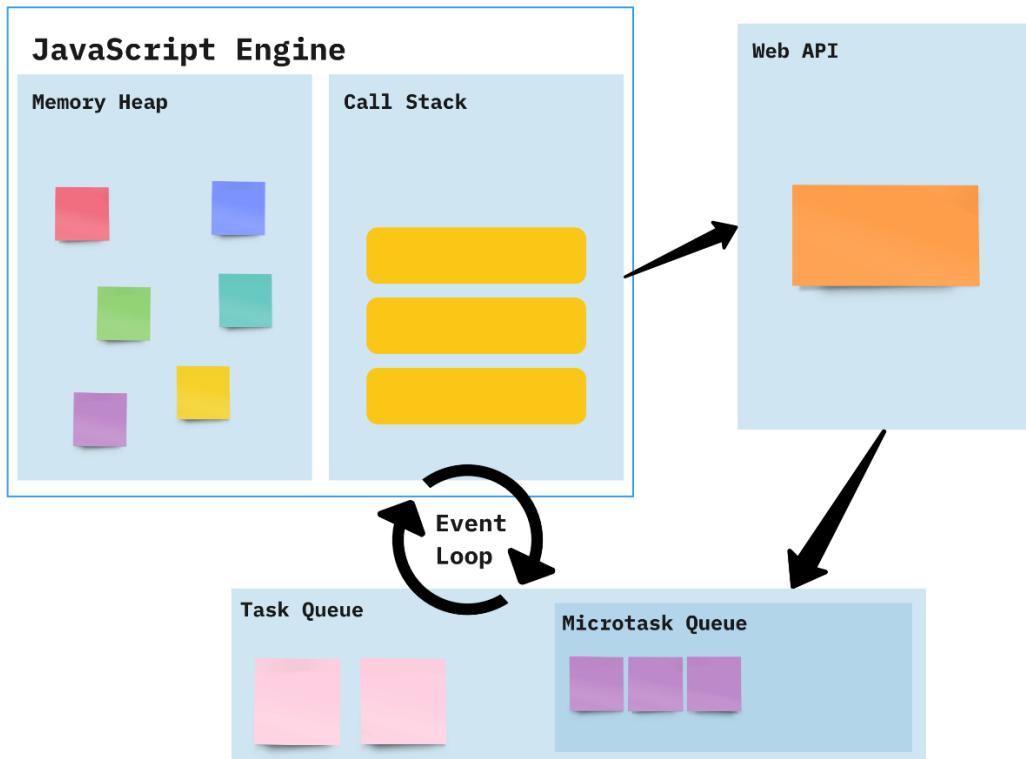
1. Synchronous:

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.

JavaScript Runtime Environment



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: <ul style="list-style-type: none">• Program Creation.• Analysis of language by the compiler and throws	Steps of Programming: <ul style="list-style-type: none">• Program Creation.• Linking of files or generation of Machine Code is not required by Interpreter.

Compiler

- 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.

The compiler saves the Machine Language in form of Machine Code on disks.

Compiled codes run faster than Interpreter.

The compiler generates an output in the form of (.exe).

Errors are displayed in Compiler after Compiling together at the current time.

Interpreter

- Execution of source statements one by one.

The Interpreter does not save the Machine Language.

Interpreted codes run slower than Compiler.

The interpreter does not generate any output.

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(“=”).

Variable is used to Store Data



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 :- (Ans)

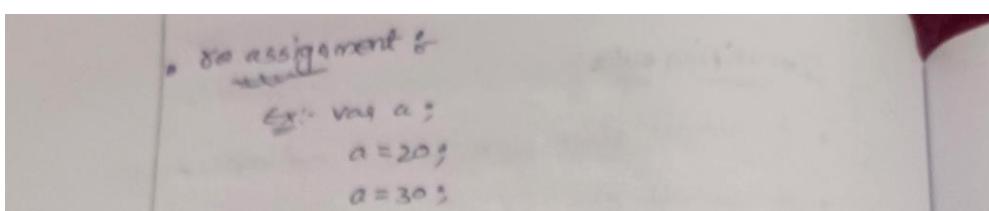
Var

- Variable declaration -- Var a;
- Variable initialization -- Initialising the value-declara
and Value var a=10;
- Variable assignment -- assigning a value after
declaration.

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

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

Ex- var a=20;
var a=30;



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

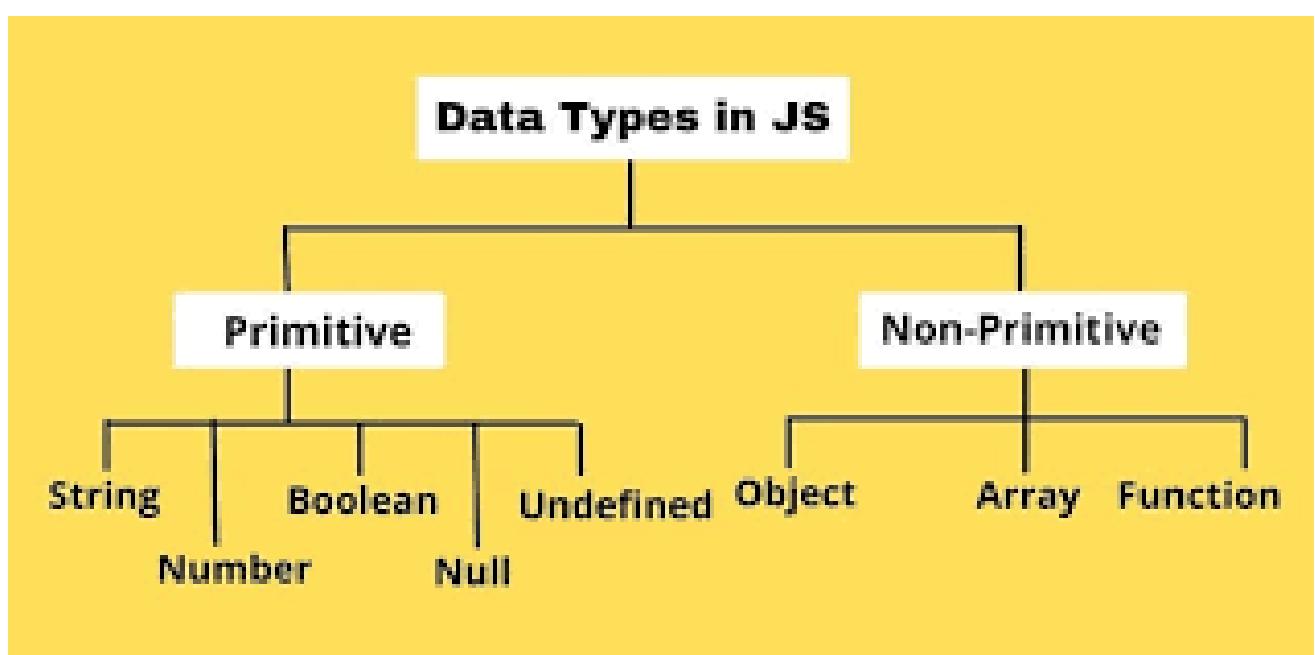
- It is a behaviour where the declaration of the variable and functions are moved to the top even before the execution.
- Only Declaration is hoisted not the Initialization
- Only works in `var` remaining `let` and `const` goes to temporary deadzone

Example:

```
a=20;  
Console.log(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 reference)
- Which are immutable. We can access the data but cannot change.

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

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

Primitive

- Number :- represents numeric values

Eg:- var a = 100;

- String :- represents Group of characters

Eg:- var b = "Hello"

- Boolean :- represents boolean value either

true - 1 or false - 0

- NULL :- represents null, ie. no value at all
(Intentionally Empty Value)

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

Non primitive

- array :- represents group of Siminal Elements
- Objects :- represents instance through which we can access members.
- functions :- it is a block of code to perform Particular task and 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: 20.
```

```
console.log(b.age); // 20
```

date :-

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

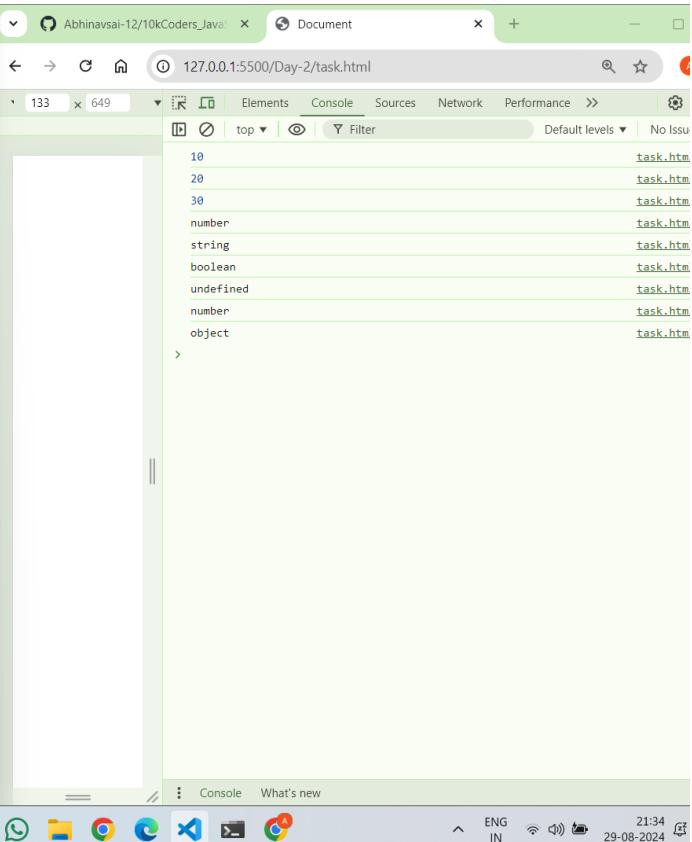
Function :-

```
var d = function() {
    console.log('Hello World');
}
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".



The screenshot shows a browser window with developer tools open. On the left, a code editor displays a script with various declarations and log statements. On the right, the browser's developer tools console shows the results of these logs. The console output is as follows:

```
10          task.htm
20          task.htm
30          task.htm
number      task.htm
string      task.htm
boolean     task.htm
undefined   task.htm
number      task.htm
object      task.htm
>
```

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 global scope means can be accessed from anywhere.

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 within 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;
const c=30; } block scope
```

Debugger

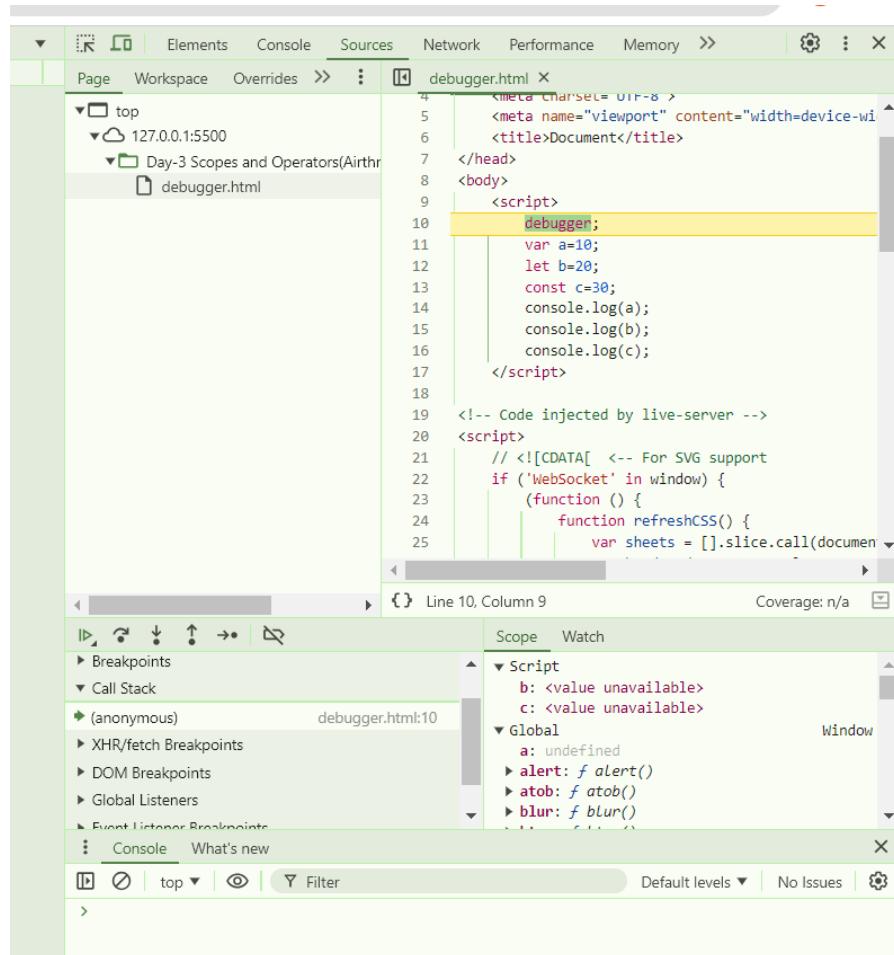
We can check Execution line by line by using keyword debugger followed by semi colon (:)

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:

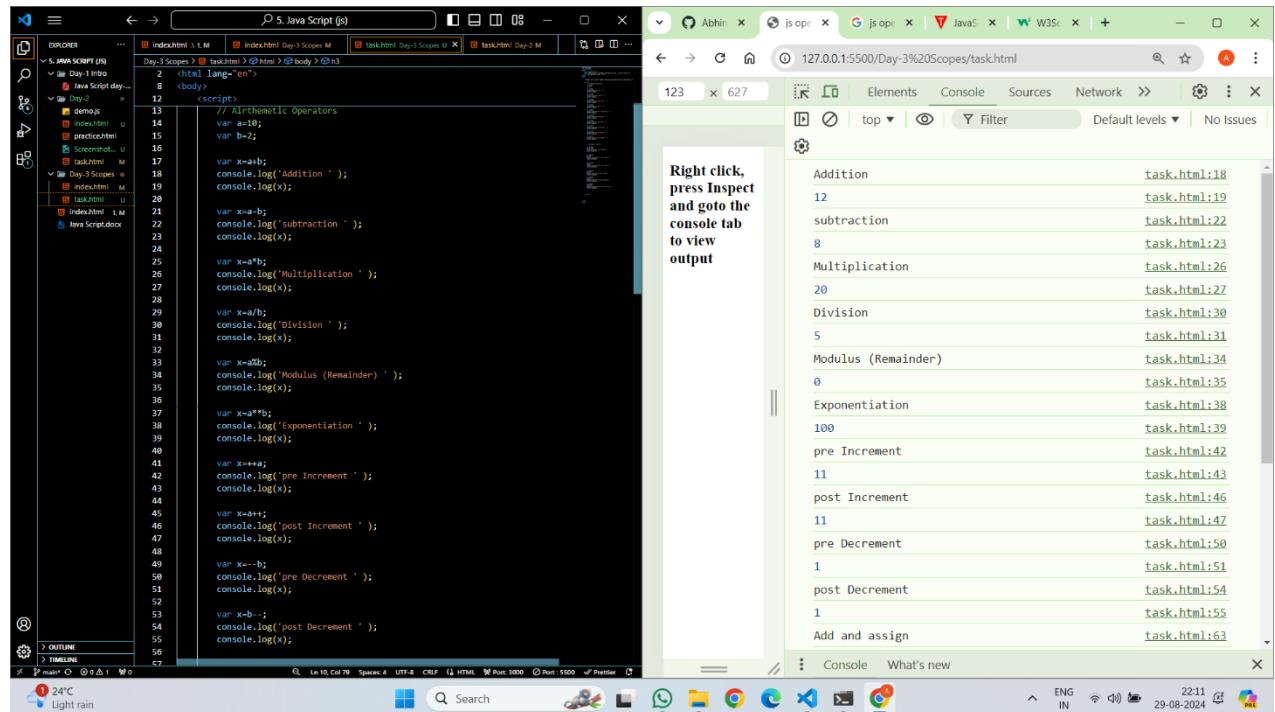
1. Airthmetic Operators
2. Assignment Operator
3. Comparision Operator
4. Logical Operator
5. Ternary Operator
6. Bitwise Oberator
7. String Operator
8. Typeof Operator

Arithmetic operators

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

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

Example:



The screenshot shows a browser window with developer tools open. The left pane is the code editor with a script file containing arithmetic operations. The right pane is the developer tools console tab, which displays the results of these operations. A tooltip in the center says: "Right click, press Inspect and go to the console tab to view output".

```
// Arithmetic 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);
```

Output in the console tab:

- Addition task.html:18
- 12 task.html:19
- Subtraction task.html:22
- 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 = a2

Example:

The screenshot shows a browser window with developer tools open. The left pane is an IDE-like interface with code snippets for various assignment operators. The right pane is a browser console showing the results of running those snippets. A tooltip on the right side of the console says: "Right click, press Inspect and goto the console tab to view output".

Code in the IDE:

```
// Assignment Operators
var a1=10;
a1 += 10;
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 /= 2;
console.log('Divide and assign');
console.log(a4);

var a5=10;
a5 %= 2;
console.log('Modulus and assign');
console.log(a5);

var a6=10;
a6 **= 2;
console.log('Exponential and assign');
console.log(a6);
```

Console Output:

Operation	Value
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
<code>==</code>	Equal to	<code>3 == 5 // false</code>
<code>!=</code>	Not equal to	<code>3 != 4 // true</code>
<code>===</code>	Strictly equal to	<code>3 === "3" // false</code>
<code>!==</code>	Strictly not equal to	<code>3 !== "3" // true</code>
<code>></code>	Greater than	<code>4 > 4 // false</code>
<code><</code>	Less than	<code>3 < 3 // false</code>
<code>>=</code>	Greater than or equal to	<code>4 >= 4 // true</code>
<code><=</code>	Less than or equal to	<code>3 <= 3 // true</code>

```
<html lang="en">
<body>
<script>
    // Comparision Operators
    var a=10;
    var b=12;
    a!=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);
</script>

```

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
<code>&&</code> (Logical AND)	<code>expression1 && expression2</code>	<code>true</code> only if both <code>expression1</code> and <code>expression2</code> are <code>true</code>
<code> </code> (Logical OR)	<code>expression1 expression2</code>	<code>true</code> if either <code>expression1</code> or <code>expression2</code> is <code>true</code>
<code>!</code> (Logical NOT)	<code>!expression</code>	<code>false</code> if <code>expression</code> is <code>true</code> and vice versa

The screenshot shows a browser window with developer tools open. The left pane displays a portion of a JavaScript file named 'logicalop.html' containing code for demonstrating logical operators. The right pane shows the browser's developer tools console tab, which has the following output:

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

The code in the file includes examples for Logical AND, Logical OR, and Logical NOT operators, along with comments explaining their usage.

```
2 <html lang="en">
8   <body>
10  <html lang="en">
16    <body>
17      <h3>Right click, press Inspect and goto the console tab to view output</h3>
18      <script>
20        // Comparison Operators
21
22        //Logical And
23        var age=+prompt('Enter Your age:')
24        var ac=(age>=0 && age<18) ? "Child":"Adult";
25        window.alert("your are a "+ac); // output in alert box
26        console.log("your are a "+ac); // output in console tab
27
28        //Logical OR
29        var x=5;
30        var or=( (x<4) || (4>x) );
31        window.alert(or); // output in alert box
32        console.log(or); // output in console tab
33
34        //Logical Not
35        console.log(!(2 < 3)); //false
36
37
```

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:

The screenshot shows a browser's developer tools console tab. The code being run is:

```
2 l lang="en">
8 y>
10 l lang="en">
16 y>
19 <script>
20 // Ternary Operator
21
22 var age=+prompt('Enter Your age:')
23 var ac=(age<=0 || age>18) ? "Child": "Adult";
24 window.alert("your are a "+ac); // output in alert box
25 console.log("your are a "+ac); // output in console tab
26
27
28
```

The console output shows the result of the ternary operator: "your are a Adult". A tooltip on the right side of the console area provides instructions: "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>
    //Nullish 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>
```

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The console output is as follows:

```
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:

The screenshot shows a browser window with developer tools open. The left panel displays a portion of an HTML file with embedded JavaScript code. The right panel is a developer tools console showing the output of `console.log` statements. The log entries are:

Output	Source
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:

The screenshot shows a browser window with the URL `127.0.0.1:5500/Day-4%20Operators/truthy_falsy.html`. On the left, the code for `truthy_falsy.html` is visible, containing several examples of the ternary operator (`a ? true : false`) with different inputs (empty string, null, undefined, 0, Abhinav, array). On the right, the browser's developer tools console tab is open, showing the output of the script. The console log entries are:

Output	File
false	truthy_falsy.htm
true	truthy_falsy.htm

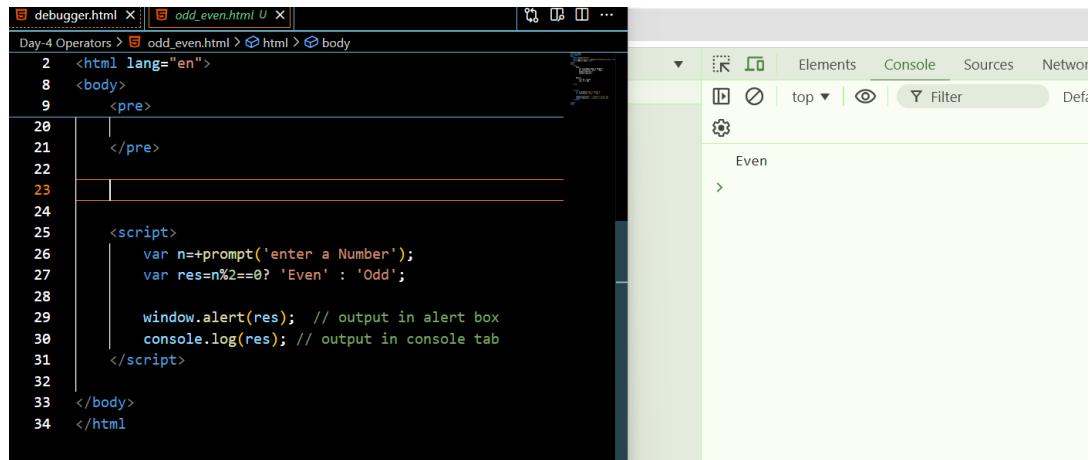
Tasks:

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

The screenshot shows a browser window with the URL `127.0.0.1:5500/Day-4%20Operators/comparison.html`. On the left, the code for `comparison.html` is visible, demonstrating various comparison operations between variables `a` and `b` (e.g., `a < b`, `a > b`, `a == b`, `a != b`, `a === b`, `a !== b`). On the right, the browser's developer tools console tab is open, showing the output of the script. The console log entries are:

Output	File
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.

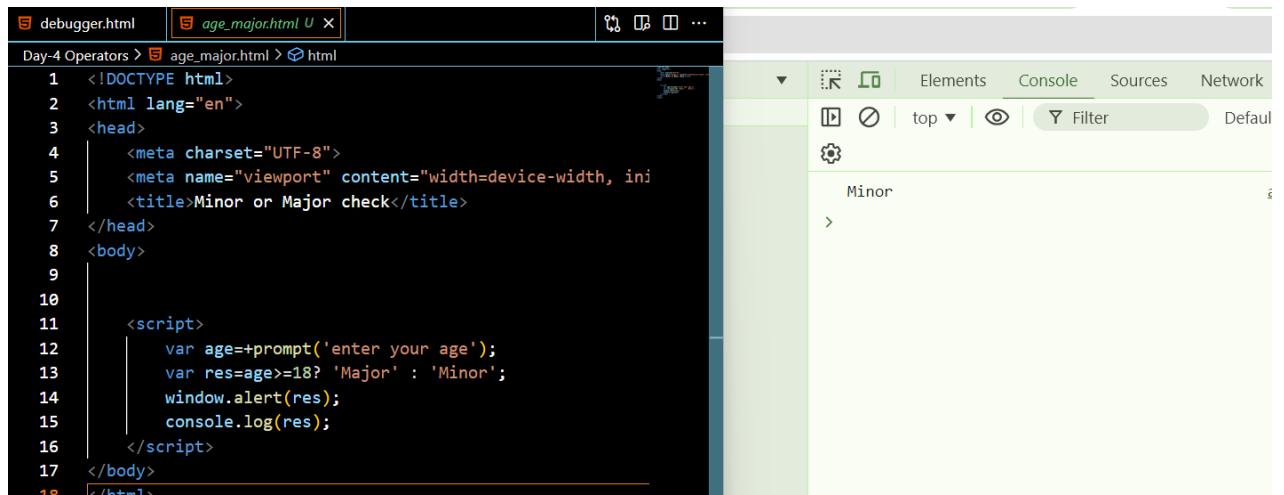


```

1 <!DOCTYPE html>
2 <html lang="en">
3 <body>
4 <pre>
5
6 <script>
7     var n=+prompt('enter a Number');
8     var res=n%2==0? 'Even' : 'Odd';
9
10    window.alert(res); // output in alert box
11    console.log(res); // output in console tab
12 </script>
13 </body>
14 </html>

```

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

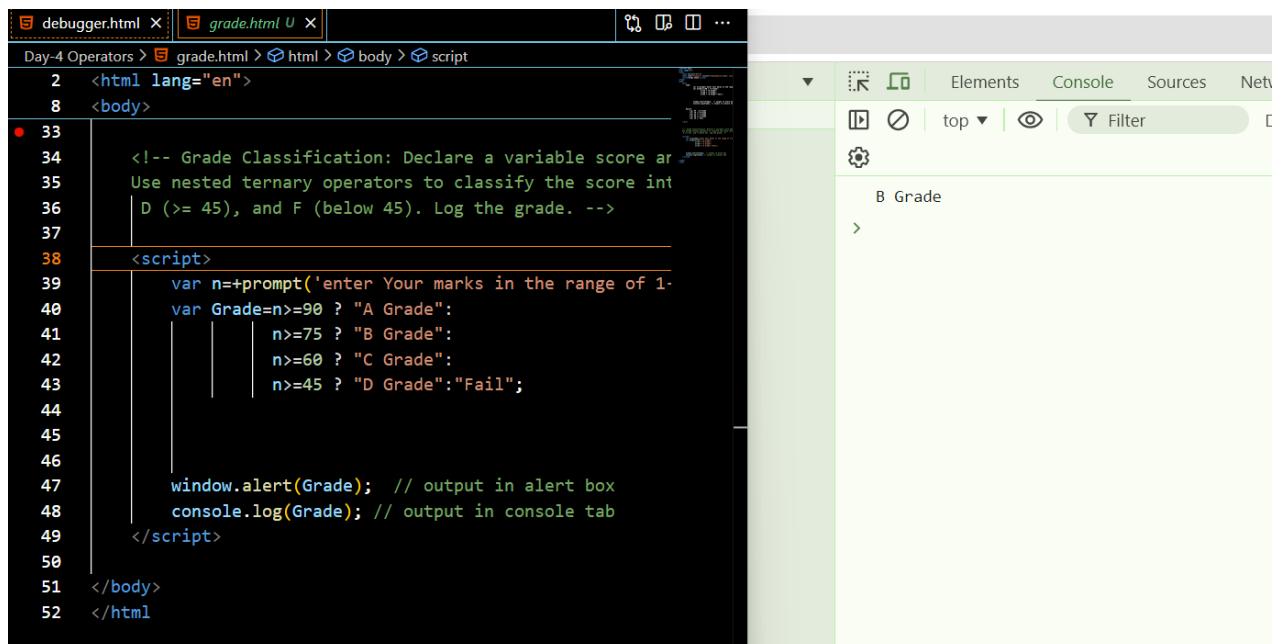


```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, ini
6     <title>Minor or Major check</title>
7 </head>
8 <body>
9
10
11 <script>
12     var age=+prompt('enter your age');
13     var res=age>=18? 'Major' : 'Minor';
14     window.alert(res);
15     console.log(res);
16 </script>
17 </body>
18 </html>

```

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.



```

1 <!DOCTYPE html>
2 <html lang="en">
3 <body>
4
5     <!-- Grade Classification: Declare a variable score ar
6     Use nested ternary operators to classify the score int
7     D ( $\geq 45$ ), and F (below 45). Log the grade. -->
8
9 <script>
10    var n=+prompt('enter Your marks in the range of 1-
11    var Grade=n>=90 ? "A Grade":
12        |   | n>=75 ? "B Grade":
13        |   | n>=60 ? "C Grade":
14        |   | n>=45 ? "D Grade":"Fail";
15
16    window.alert(Grade); // output in alert box
17    console.log(Grade); // output in console tab
18 </script>
19
20 </body>
21 </html>

```

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.

```

1 debugger.html
2 temperature_check.html U X
3 Day-4 Operators > temperature_check.html > html > body > script
4
5 2 <html lang="en">
6 8 <body>
7
8 <script>
9   var n=+prompt('enter Temperature for Temperature C');
10  var temp=n>30 ? "HOT":
11    |   |   n>=20 ? "Warm":
12    |   |   n>=10 ? "Cool":"cold";
13
14   window.alert(temp); // output in alert box
15   console.log(temp); // output in console tab
16
17 </script>
18
19 </body>
20
21 </html>

```

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.

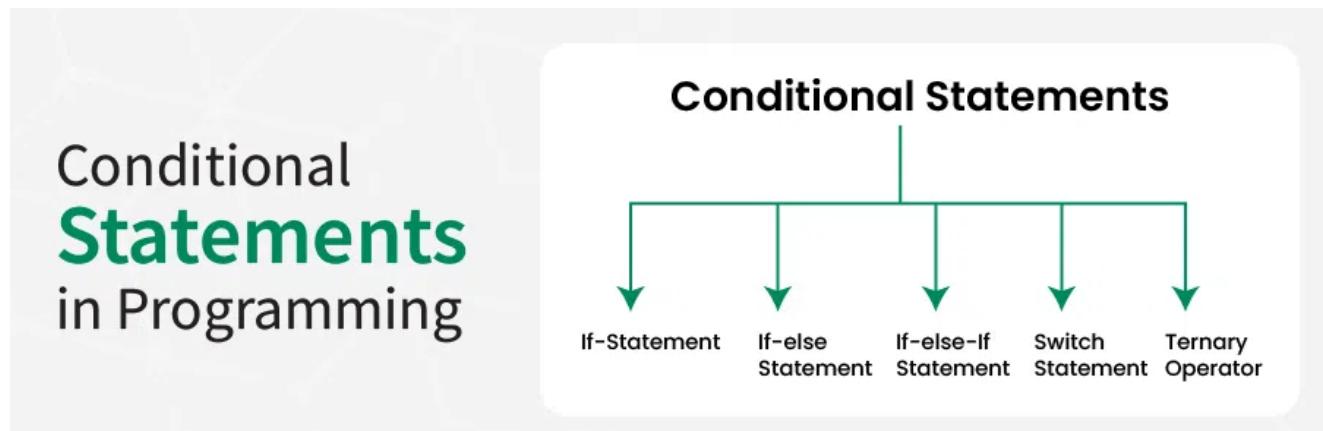
```

1 debugger.html X
2 age_check.html U X
3 Day-4 Operators > age_check.html > html > body > pre > ? > ? > ? > script
4
5 2 <html lang="en">
6 8 <body>
7 10 <pre>
8 12   var ac=(age>=0 && age<=12) ? "Child":
9 13     |   (age>=13 && age<=19) ? "Teen":
10
11   </pre>
12
13   <!-- Age Group: Declare a variable age and
14   use the ternary operator to classify the age into "Child" (0-12), "Teen" (13-19)
15   "Adult" (20-64), and "Senior" (65 and above). Log the result. -->
16
17
18 <script>
19   var age=+prompt('enter Your age to known which age group you are');
20   var ac=(age>=0 && age<=12) ? "Child":
21     |   (age>=13 && age<=19) ? "Teen":
22     |   (age>=20 && age<=64) ? "Adult":"Senior";
23
24
25
26
27   window.alert(ac); // output in alert box
28   console.log(ac); // output in console tab
29
30 </script>
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50

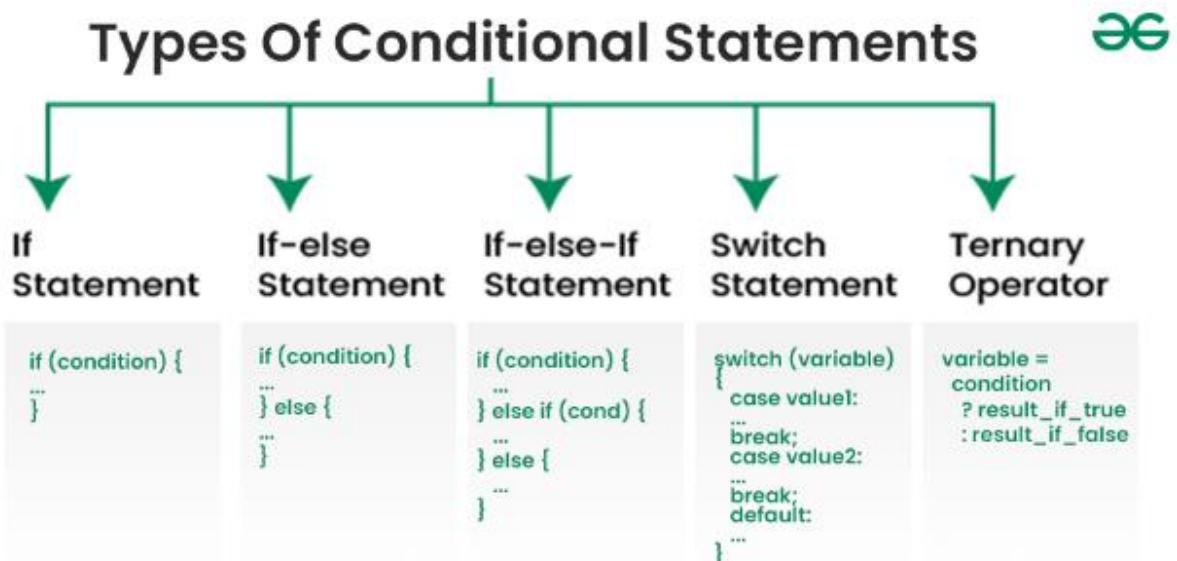
```

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.



Conditional Statements in Programming



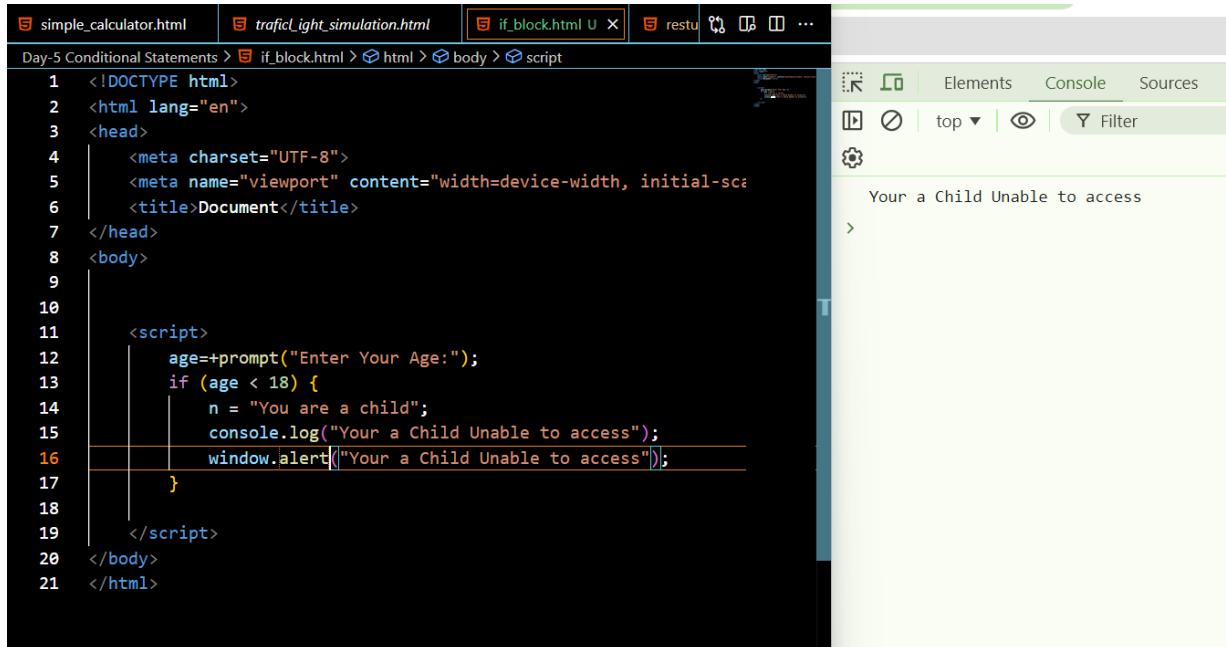
1. if Statement:

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:



The screenshot shows a browser window with several tabs open. The active tab is 'if_block.html'. The code in the editor is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10
11   <script>
12     age+=prompt("Enter Your Age:");
13     if (age < 18) {
14       n = "You are a child";
15       console.log("Your a Child Unable to access");
16       window.alert("Your a Child Unable to access");
17     }
18
19   </script>
20 </body>
21 </html>
```

In the developer tools' console tab, the output is:

```
Your a Child Unable to access
```

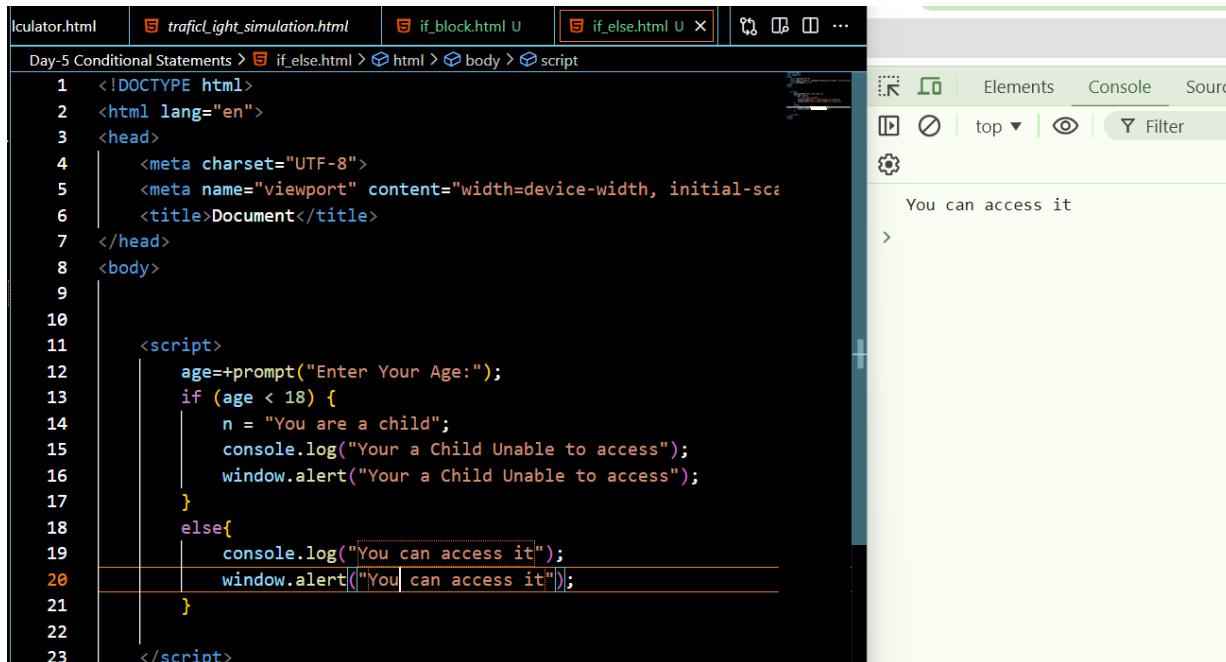
2. if...else Statement:

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:



The screenshot shows a browser window with several tabs open. The active tab is 'if_else.html'. The code in the editor is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10
11   <script>
12     age+=prompt("Enter Your Age:");
13     if (age < 18) {
14       n = "You are a child";
15       console.log("Your a Child Unable to access");
16       window.alert("Your a Child Unable to access");
17     }
18     else{
19       console.log("You can access it");
20       window.alert("You can access it");
21     }
22
23   </script>
```

In the developer tools' console tab, the output is:

```
You can access it
```

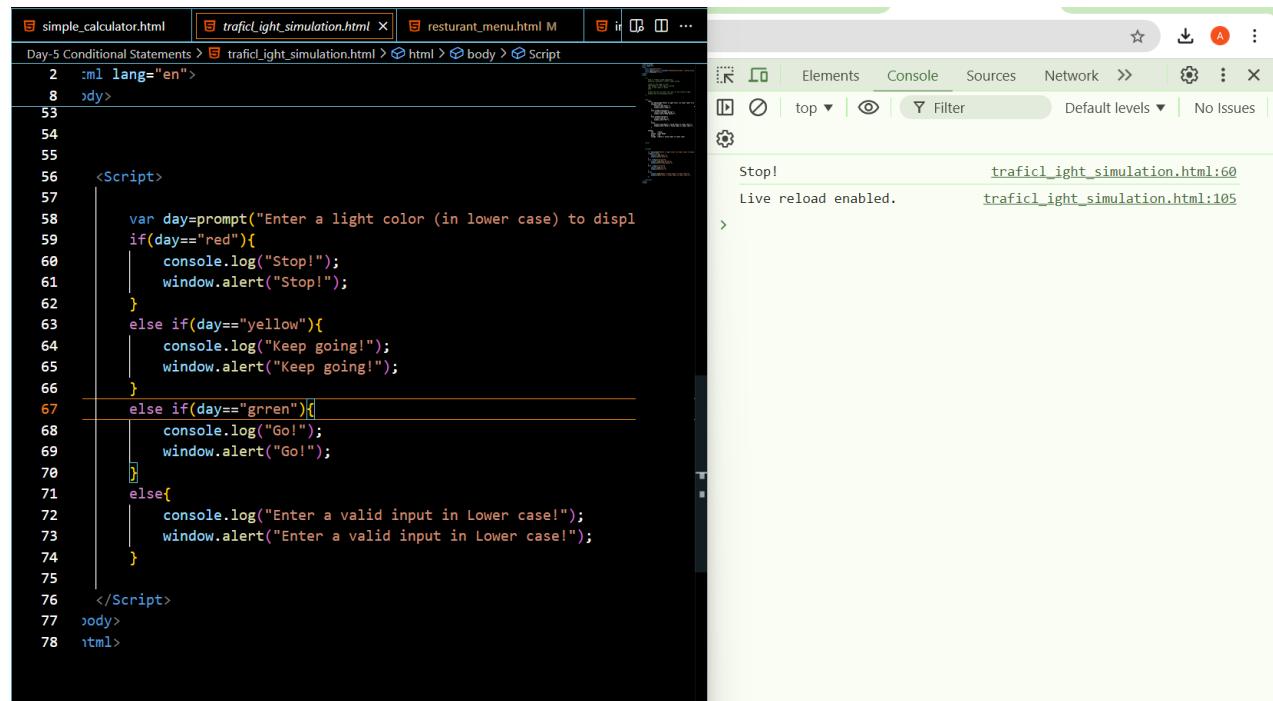
3)if...else if...else Statement:

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:



The screenshot shows a browser developer tools window with the "Console" tab selected. It displays the following output:

```
Stop!          traficLight_simulation.html:60  
Live reload enabled.  traficLight_simulation.html:105
```

The console output indicates that the script has stopped executing at line 60, and live reload is enabled. The code being run is part of a file named "traficLight_simulation.html".

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:

The screenshot shows a browser developer tools interface with the 'Console' tab selected. The code being run is a script that prompts for an age, then uses a nested if block to check if the age is less than 18. If true, it logs a message and alerts the user they are a child. If false, it logs a message and alerts the user they can access it. An else block at the bottom handles negative numbers by logging and alerting them.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10
11   <script>
12     age=+prompt("Enter Your Age:");
13     if(age >0){
14       //nested if block
15       if (age < 18) {
16         n = "You are a Child";
17         console[methodName](message?: any): void;
18         window.alert("Your a Child Unable to access");
19       }
20       else{
21         console.log("You can access it");
22         window.alert("You can access it");
23       }
24     }
25     else{
26       console.log("You entered a -ve number");
27     }
28   </script>
```

You entered a -ve number

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

- Expression Evaluation:** The **expression** inside the switch statement is evaluated once.
- Case Matching:** The result of the expression is compared with the values specified in each **case** clause using strict equality (`==`).
- Code Execution:** If a match is found, the code block associated with that **case** is executed.
- 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).
- 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:

```

index.html | simple_calculator.html | restaurant_menu.html M | index.html ...
Day-5 Conditional Statements > restaurant_menu.html > html > body > Script
2   <html lang="en">
8     <body>
51       </pre>
52
53     <Script>
54
55       var dish=prompt("Enter a name of the dish(only Biriyani")
56       switch(dish){[{"highlighted": true, "start": 56, "end": 60}]
57         case "biriyani":
58           console.log("Cost of Biriyani is 180/-");
59           window.alert("Cost of Biriyani is 180/-");
60           break;
61
62         case "shawarma":
63           console.log("Cost of Shawarma is 80/-");
64           window.alert("Cost of Shawarma is 80/-");
65
66         case "fried rice":
67           console.log("Cost of Fried Rise is 100/-");
68           window.alert("Cost of Fried Rise is 100/-");
69
70         case "veg pulav":
71           console.log("Cost of veg pulav is 220/-");
72           window.alert("Cost of veg pulav is 220/-");
73
74       default:
75         console.log("Enter a valid name as shown alert box");
76         window.alert("Enter a valid input in Lower case!");
77
78     }
79

```

Cost of Biriyani is 180/-

restaurant_menu.html:58

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.

```

nested_if.html U  restaurant_menu.html M  index.html Day-5 Conditional Statements 1  day_to_the_week.html X  ⌂ ⌂ ...
Day-5 Conditional Statements > day_to_the_week.html > html > body > pre
2 <html lang="en">
8 <body>
55
56
57
58   var day=prompt("Enter a day in Week in lower case");
59   if(day=="monday"){
60     console.log("Start your week strong!"); // output in console tab
61     window.alert("Start your week strong!"); // output in console tab
62   }
63   else if(day=="tuesday"){
64     console.log("Keep going!");
65     window.alert("Keep going!");
66   }
67   else if(day=="wednesday"){
68     console.log("Halfway there!");
69     window.alert("Halfway there!");
70   }
71   else if(day=="thursday"){
72     console.log("Almost the weekend!");
73     window.alert("Almost the weekend!");
74   }
75   else if(day=="friday"){
76     console.log("Happy Friday!");
77     window.alert("Happy Friday!");
78   }
79   else if(day=="saturday"){
80     console.log("Enjoy your weekend!");
81     window.alert("Enjoy your weekend!");
82   }
83   else if(day=="sunday"){
84     console.log("Ready for monday :)");
85     window.alert("Ready for monday :)");
86   }
87   else{
88     console.log("Enter a valid input :( ");
89     window.alert("Enter a valid input :( ");
90   }
91

```

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.

```

simple_calculator.html  traficLight_simulation.html X  restaurant_menu.html M  index.html Day-5 Conditional Statements 1  ⌂ ⌂ ...
Day-5 Conditional Statements > traficLight_simulation.html > html > body > Script
2 <!ml lang="en">
8 <body>
53
54
55
56 <Script>
57
58   var day=prompt("Enter a light color (in lower case) to displ
59   if(day=="red"){
60     console.log("Stop!");
61     window.alert("Stop!");
62   }
63   else if(day=="yellow"){
64     console.log("Keep going!");
65     window.alert("Keep going!");
66   }
67   else if(day=="grren"){
68     console.log("Go!");
69     window.alert("Go!");
70   }
71   else{
72     console.log("Enter a valid input in Lower case!");
73     window.alert("Enter a valid input in Lower case!");
74   }
75
76 </Script>
77 </body>
78 </!ml>

```

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.

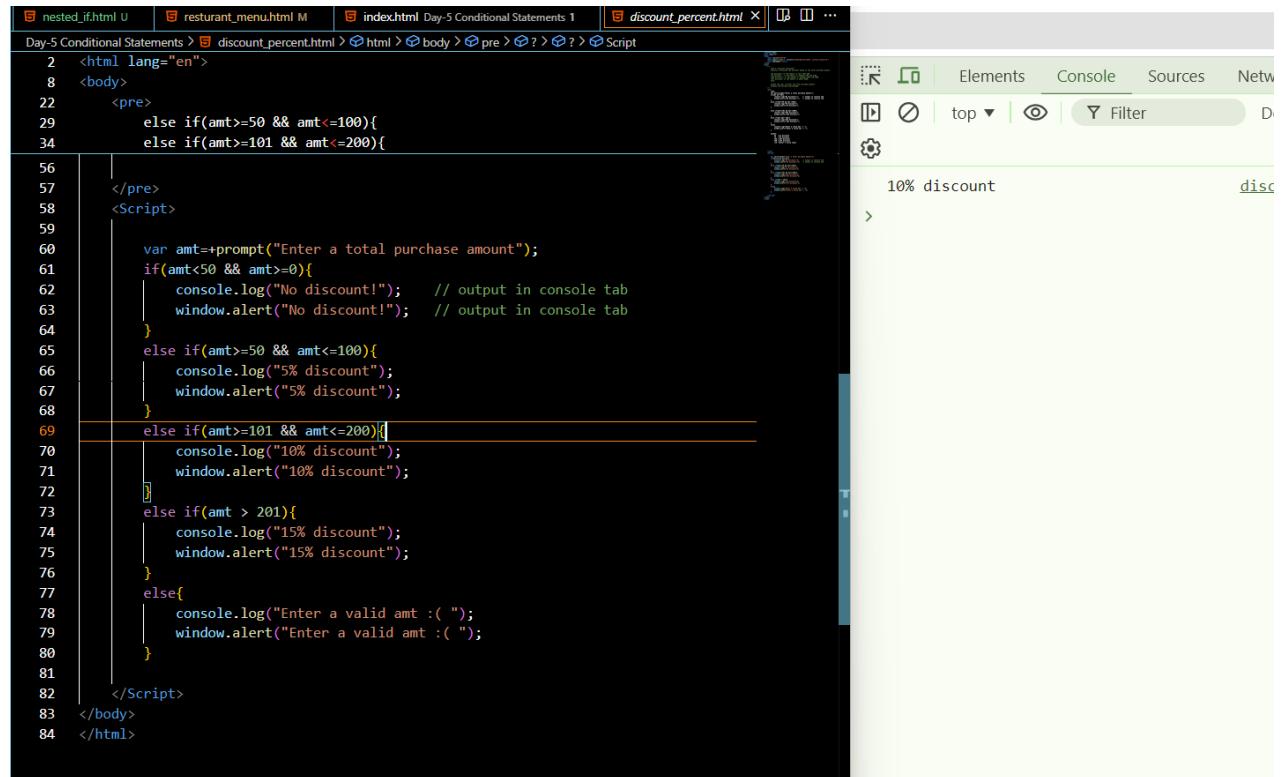
“10% discount” if the amount is between \$101 and \$200.

“15% discount” if the amount is above \$200.

Task:

Prompt the user to enter the total purchase amount.

Display the discount percentage.



The screenshot shows a browser window with several tabs at the top: nested_if.html, restaurant_menu.html, index.html Day-5 Conditional Statements 1, discount_percent.html, and another tab that is partially visible. The main area is a code editor with the following JavaScript code:

```
2 <html lang="en">
8   <body>
22     <pre>
29       else if(amt>=50 && amt<=100){
34         else if(amt>=101 && amt<=200){
56
57           </pre>
58           <Script>
59
60             var amt=+prompt("Enter a total purchase amount");
61             if(amt<50 && amt>=0){
62               console.log("No discount!"); // output in console tab
63               window.alert("No discount!"); // output in console tab
64             }
65             else if(amt>=50 && amt<=100){
66               console.log("5% discount");
67               window.alert("5% discount");
68             }
69             else if(amt>=101 && amt<=200){
70               console.log("10% discount");
71               window.alert("10% discount");
72             }
73             else if(amt > 201){
74               console.log("15% discount");
75               window.alert("15% discount");
76             }
77             else{
78               console.log("Enter a valid amt :( ");
79               window.alert("Enter a valid amt :( ");
80             }
81           </Script>
83   </body>
84 </html>
```

To the right of the code editor is a browser's developer tools interface. The "Console" tab is selected, showing the output "10% discount". The "Sources" tab is also visible. The URL in the address bar is "discount_percent.html".

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., "Biryani", "shawarma", "Fried rice", "veg pulao").
Print the price.

```

nested_if.html U restaurant_menu.html M index.html Day-5 Conditional Statements 1 discount_percent.js ...
Day-5 Conditional Statements > restaurant_menu.html > html > body > Script
2 <html lang="en">
8 <body>
53 <Script>
54     var dish=prompt("Enter a name of the dish(only Biriyani, shawarma, Fried rice");
55     switch(dish){
56         case "biriyani":
57             console.log("Cost of Biriyani is 180/-");
58             window.alert("Cost of Biriyani is 180/-");
59             break;
60
61         case "shawarma":
62             console.log("Cost of Shawarma is 80/-");
63             window.alert("Cost of Shawarma is 80/-");
64             break;
65
66         case "fried rice":
67             console.log("Cost of Fried Rise is 100/-");
68             window.alert("Cost of Fried Rise is 100/-");
69             break;
70
71         case "veg pulav":
72             console.log("Cost of veg pulav is 220/-");
73             window.alert("Cost of veg pulav is 220/-");
74             break;
75
76     default:
77         console.log("Enter a valid name as shown alert box in Lower case!");
78         window.alert("Enter a valid input in Lower case!");
79
80
81
82
83

```

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.

```

simple_calculator.html M if_block.html U if_else.html U nested_if.html U restaurant_menu.html ...
Day-5 Conditional Statements > simple_calculator.html > html > body > Script
2 <html lang="en">
8 <body>
79 <Script>
80     var a=+prompt("Enter First Number");
81     var b=+prompt("Enter Second Number");
82     var op=prompt("Enter operation in b/w (+, -, /, *, **)");
83     console.log(a);
84     console.log(b);
85     console.log(op);
86     switch(op){
87         case "+":
88             var c=a+b;
89             console.log(c);
90             window.alert(c);
91             break;
92
93         case "-":
94             var c=a-b;
95             console.log(c);
96             window.alert(c);
97             break;
98
99         case "*":
100            var c=a*b;
101            console.log(c);
102            window.alert(c);
103            break;
104
105         case "/":
106            var c=a/b;
107            console.log(c);
108            window.alert(c);
109            break;
110
111         case "%":
112            var c=a%b;
113            console.log(c);
114            window.alert(c);
115            break;
116

```

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:

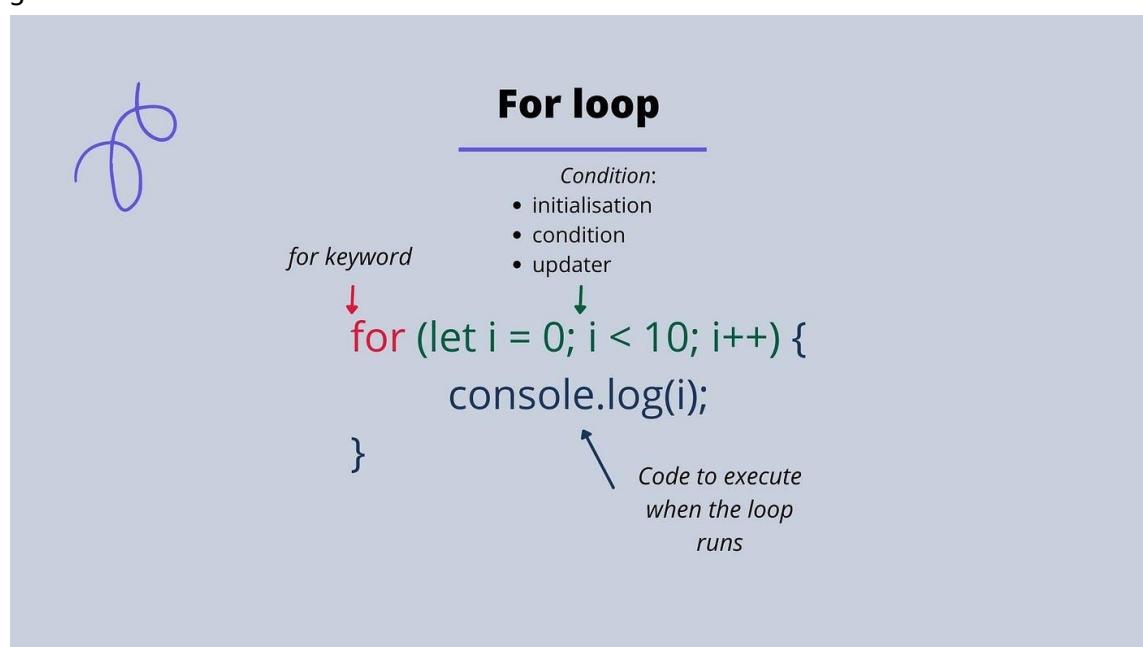
```
for (initialization; condition; increment)
{
    code to be executed
}
```

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>
```

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

Example:- 7 table

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

Example:- Tables

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

Example :- sum of even numbers

```
var count=0;  
for(i=1; i<=10; i++) {  
    if (i%12==0) {  
        count = count + 1;  
    }  
    console.log(count); //30
```

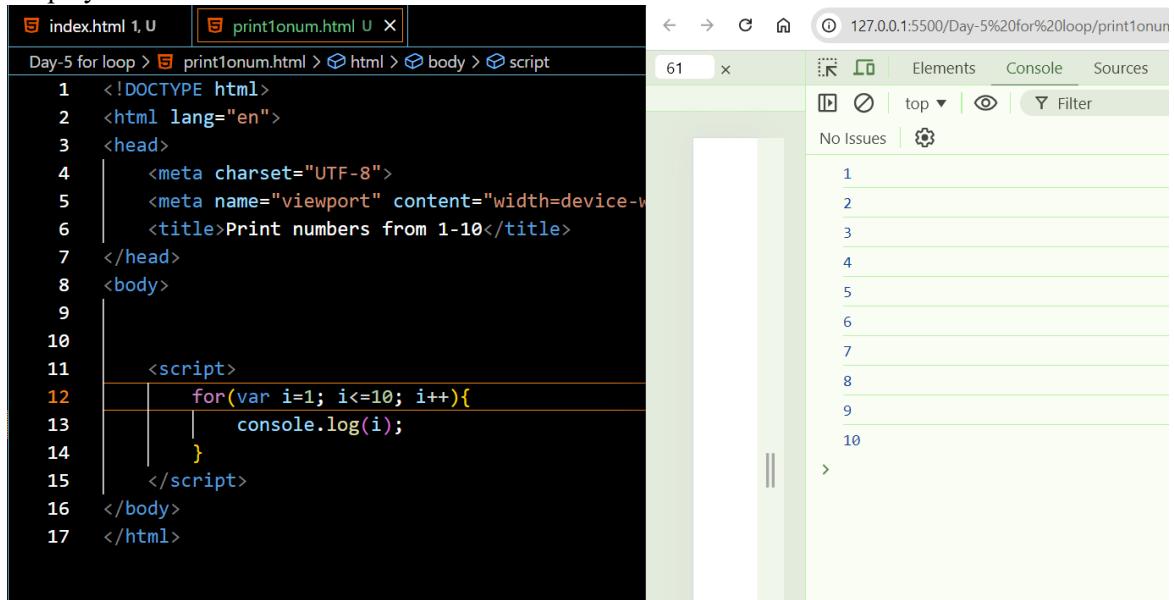
Tasks:

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.



The screenshot shows a browser developer tools window with the URL `127.0.0.1:5500/Day-5%20for%20loop/print1onum.html`. The left pane displays the HTML code for `print1onum.html`, which contains a script block with a for loop that logs numbers 1 through 10 to the console. The right pane shows the browser's developer tools console tab, which lists the output: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Print numbers from 1-10</title>
</head>
<body>
<script>
    for(var i=1; i<=10; i++){
        console.log(i);
    }
</script>
</body>
</html>
```

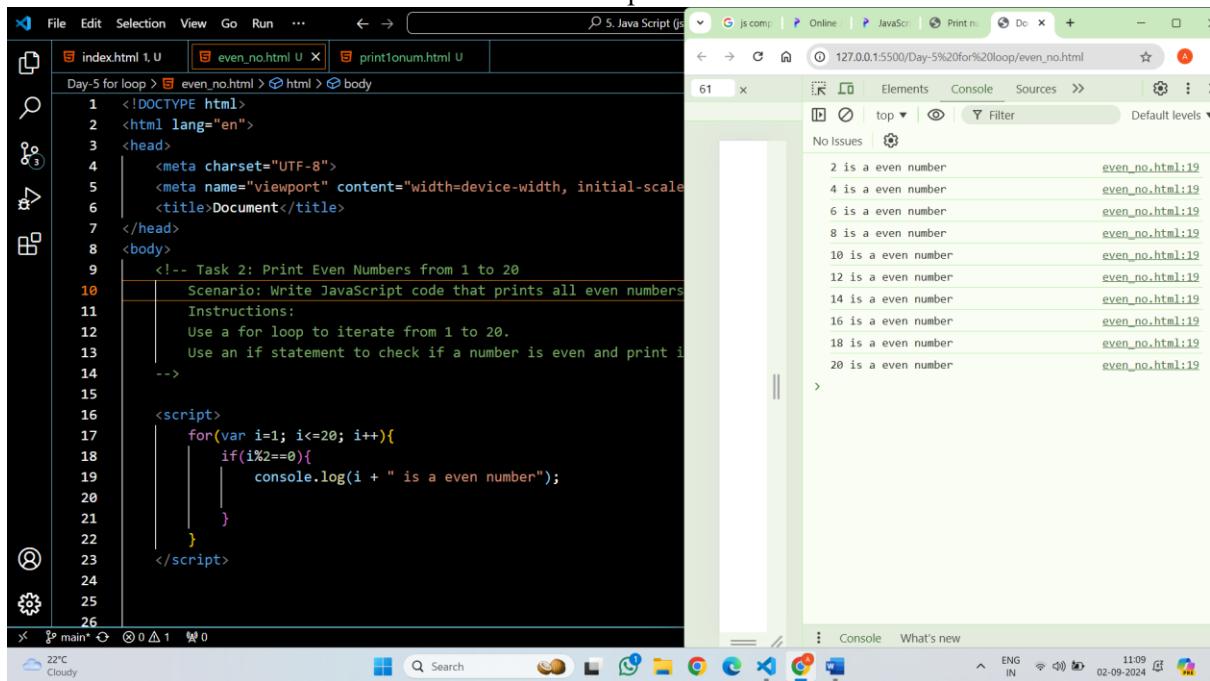
Task 2: Print Even Numbers from 1 to 20

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.



The screenshot shows a browser developer tools window with the URL `127.0.0.1:5500/Day-5%20for%20loop/even_no.html`. The left pane displays the HTML code for `even_no.html`, which includes comments for Task 2, scenario instructions, and a script block that logs even numbers from 2 to 20. The right pane shows the browser's developer tools console tab, which lists the output: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20. Each output line is preceded by the message "is a even number".

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    
    Scenario: Write JavaScript code that prints all even numbers
    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.
    -->

    <script>
        for(var i=1; i<=20; i++){
            if(i%2==0){
                console.log(i + " is a even number");
            }
        }
    </script>
</body>
</html>
```

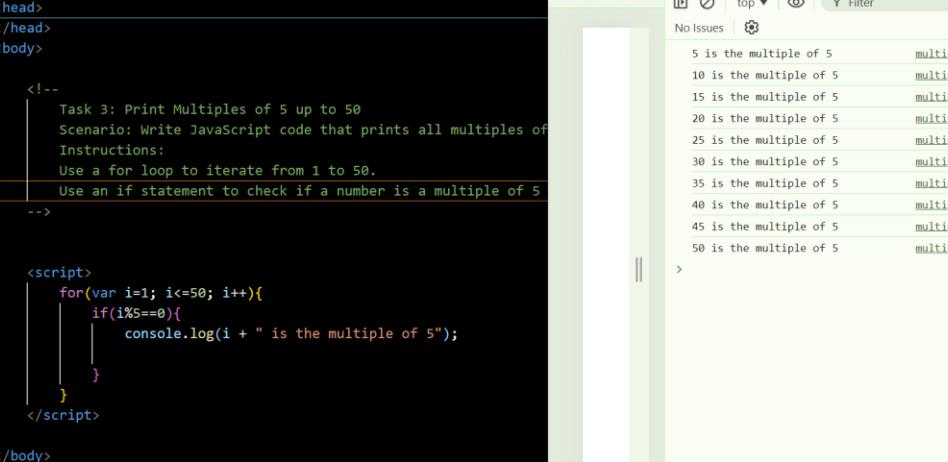
Task 3: Print Multiples of 5 up to 50

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.



The screenshot shows a browser window displaying the output of a JavaScript program. The URL is `127.0.0.1:5500/Day-5%20for%20loop/multiples_of_5.html`. The page content lists all multiples of 5 from 5 to 50, each preceded by the message "is the multiple of 5". The browser interface includes tabs for index.html, even_no.html, multiples_of_5.html (active), and print1num.html. The developer tools are open, showing the Elements, Console, and Sources tabs. The console tab displays the following log entries:

```
5 is the multiple of 5
10 is the multiple of 5
15 is the multiple of 5
20 is the multiple of 5
25 is the multiple of 5
30 is the multiple of 5
35 is the multiple of 5
40 is the multiple of 5
45 is the multiple of 5
50 is the multiple of 5
```

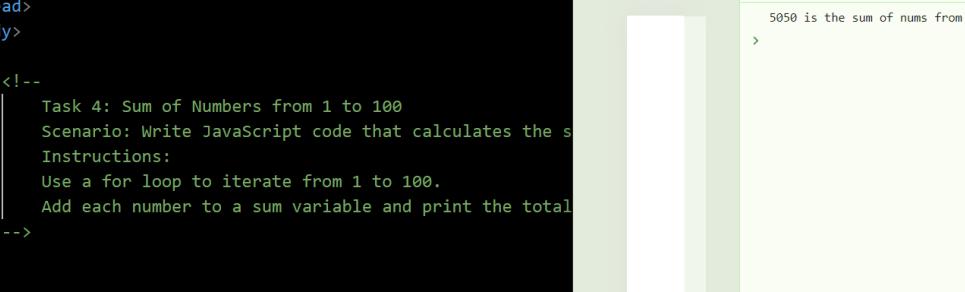
Task 4: Sum of Numbers from 1 to 100

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.



The screenshot shows a browser window with three tabs at the top: "index.html, U", "SumOfNum_1to100.html U X", and "factorial.html U". The main content area displays a code editor with a file named "Day-5 for loop > SumOfNum_1to100.html > html > body". The code contains a comment block with instructions and a script block that uses a for loop to calculate the sum of numbers from 1 to 100, printing the result to the console. The browser's developer tools are open, showing the console tab with the output "5050 is the sum of nums from 1 to 100".

```
2 <html lang="en">
3 </head>
4 <body>
5
6     <!--
7         Task 4: Sum of Numbers from 1 to 100
8         Scenario: Write JavaScript code that calculates the sum of all numbers from 1 to 100.
9         Instructions:
10            Use a for loop to iterate from 1 to 100.
11            Add each number to a sum variable and print the total
12        -->
13
14
15     <script>
16         count=0;
17         for(var i=1; i<=100; i++){
18             |     count=count+i;
19         }
20         console.log(count + " is the sum of nums from 1 to 100")
21     </script>
22
23
24
25
```

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

Task 3: Create

Use a for loop to multiplication the given

Take prompt from the user

Take prompt from the user
hint: take count value as 1.

```

<html lang="en">
<head>
</head>
<body>
<!--
Task 5: Create a JavaScript program that calculates the factorial of a given number.
Instructions:
Use a for loop to multiplication the given number.
Take prompt from the user
hint: take count value as 1;
-->

<script>
fact=1;
var n=prompt("Enter any number to get the factorial");
for(var i=1; i<=n; i++){
    fact=fact*i;
}
console.log(fact + " is the factorial of "+ n);
window.alert(fact + " is the factorial of "+ n);
</script>
</body>
</html>

```

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.

```

<html lang="en">
<head>
</head>
<body>
<!--
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.
-->

<script>
for(var i=10; i>=0; i--){
    console.log(i);
}
</script>
</body>
</html>

```

Number	Console Output
10	reverse_of_num
9	reverse_of_num
8	reverse_of_num
7	reverse_of_num
6	reverse_of_num
5	reverse_of_num
4	reverse_of_num
3	reverse_of_num
2	reverse_of_num
1	reverse_of_num
0	reverse_of_num

(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]

The screenshot shows a browser window with two tabs. The left tab contains a code editor with a script that uses a for loop to print the alphabet from A to Z. The right tab shows the resulting output in the browser's developer tools, where each letter from A to Z is listed with its corresponding ASCII value and file path.

```

<html lang="en">
<head>
<!--
Task 7: Print the Alphabet
Scenario: Write JavaScript code that prints the alphabet
Instructions:
Use a for loop to iterate through the ASCII values of the letters A-Z.
Convert the ASCII values to characters and print them.
Hint - console.log(String.fromCharCode(i));
-->

<script>
for(var i=65; i<=90;i++)
    console.log(String.fromCharCode(i));
</script>
</body>
</html>

```

Letter	ASCII Value	File Path
A	65	alphabets_AtoZ.html:21
B	66	alphabets_AtoZ.html:21
C	67	alphabets_AtoZ.html:21
D	68	alphabets_AtoZ.html:21
E	69	alphabets_AtoZ.html:21
F	70	alphabets_AtoZ.html:21
G	71	alphabets_AtoZ.html:21
H	72	alphabets_AtoZ.html:21
I	73	alphabets_AtoZ.html:21
J	74	alphabets_AtoZ.html:21
K	75	alphabets_AtoZ.html:21
L	76	alphabets_AtoZ.html:21
M	77	alphabets_AtoZ.html:21
N	78	alphabets_AtoZ.html:21
O	79	alphabets_AtoZ.html:21
P	80	alphabets_AtoZ.html:21
Q	81	alphabets_AtoZ.html:21
R	82	alphabets_AtoZ.html:21
S	83	alphabets_AtoZ.html:21
T	84	alphabets_AtoZ.html:21
U	85	alphabets_AtoZ.html:21
V	86	alphabets_AtoZ.html:21
W	87	alphabets_AtoZ.html:21
X	88	alphabets_AtoZ.html:21
Y	89	alphabets_AtoZ.html:21
Z	90	alphabets_AtoZ.html:21

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 127.0.0.1:5500/Day-5%20for%20loop/multip

Day-5 for loop > multiple_table_1to5.html > html > body > script

```

2   <html lang="en">
8     <body>
12       instructions.
13         use for loop
14         use nested loop
15         -->
16
17
18     <script>
19       for(var i=1; i<=5; i++){
20         console.log("Table "+ i);
21         for(var j=1; j<=10; j++){
22
23           var x=i*j;
24           console.log(i+ " * " +j+'=' +x);
25
26
27         }
28     </script>
29
30   </body>
31 </html>

```

multiple_table_1to5.html > 127.0.0.1:5500/Day-5%20for%20loop/multiple_table_1to5.htm

for loop > multiple_table_1to5.html > html > body > script

```

<html lang="en">
<body>
  instructions.
    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>

```

multiple_table_1to5.html > 127.0.0.1:5500/Day-5%20for%20loop/multip

61 632 Elements Console Sources Network Performance Settings 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 > 127.0.0.1:5500/Day-5%20for%20loop/multip

61 645 Elements Console Sources Network Performance Settings top Filter Definitions

2 * 10=20

multiple_table_1to5.html > 127.0.0.1:5500/Day-5%20for%20loop/multip

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

multiple_table_1to5.html > 127.0.0.1:5500/Day-5%20for%20loop/multip

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

multiple_table_1to5.html > 127.0.0.1:5500/Day-5%20for%20loop/multip

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

>

While Loop:

A **while** loop repeats a block of code while a specified condition is true.

The condition is evaluated before each iteration. If it returns true, the loop continues; otherwise, it stops.

Syntax:

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

Example:

The screenshot shows a browser window with several tabs at the top: 'ate_forof.html U', 'rev_str.html U', 'even_in_array.html U', 'while.html U' (which is the active tab), and others. Below the tabs, the page title is 'Loops > while.html > html > body'. The code in the body is:

```
<html lang="en">  
<body>  
    <script>  
        var i=0;  
        while(i<10){  
            if(i%2==0){  
                console.log(i +" is even number")  
            }  
            i++;  
        }  
    </script>  
</body>
```

To the right of the code, the browser's developer tools show the 'Console' tab with the following output:

0 is even number
2 is even number
4 is even number
6 is even number
8 is even number
Live reload enabled.

Do While:

Similar to the **while** loop, but it always executes its block of code at least once, even if the condition evaluates to false.

The block of code is executed first, then the condition is evaluated. If true, the loop continues; if false, it stops.

Syntax:

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

Example:

The screenshot shows a browser window with several tabs at the top: 'ex.html 1, U', 'do_while.html U' (active tab), 'obj_iterate_forof.html U', and others. Below the tabs, the page title is 'Loops > do_while.html > html > body > script'. The code in the script tag is:

```
<html lang="en">  
<body>  
    <script>  
        var i=0;  
        do{  
            if(i%2!=0){  
                console.log(i +" is odd number")  
            }  
            i++;  
        }  
        while(i<10)  
    </script>  
</body>
```

To the right of the code, the browser's developer tools show the 'Console' tab with the following output:

1 is odd number
3 is odd number
5 is odd number
7 is odd number
9 is odd number
Live reload enabled.

For in loop:

Used to iterate over the properties of an object, array, string. It iterates over enumerable properties of an object, in an arbitrary order.

We can access index and values.

Syntax:

```
for (ref in strname){  
    console.log(ref); //indexes  
}
```

Example:

The screenshot shows a browser window with two tabs: "for_of.html" and "for_in.html". The "for_in.html" tab is active, displaying an HTML file with a script block. The script declares a variable `str` with the value 'hello', then uses a `for` loop to iterate over its properties. Inside the loop, it logs the index `i` and the value `str[i]` to the console. The browser's developer tools console on the right shows the output: the index `i` (0, 1, 2, 3, 4) and the corresponding character of the string ('h', 'e', 'l', 'l', 'o').

```
for (ref in strname){  
    console.log(ref); //indexes  
}  
  
for (i in str){  
    console.log(i); //prints index number  
    console.log(str[i]); //prints index value  
}
```

1) Iterates over Properties:

- The for...in loop iterates over all enumerable properties of an object.

2) Order Not Guaranteed:

- The order of iteration is not guaranteed. It's generally the order in which properties were defined, but this can vary.

3) Use with Objects:

- Typically used for objects, not arrays, because it iterates over property names (keys) rather than values.

For of Loop:

Introduced in ES6, it iterates over iterable objects such as arrays, strings, maps, sets, etc.

Syntax:

```
for (ref of strname){  
    console.log(ref); //values  
}
```

```
=====  
for (variable of iterable) {  
    // code to be executed  
}
```

- It provides a more concise syntax compared to the traditional **for** loop for iterating over arrays and other iterable objects.

1) Iterates over Values:

- The for...of loop iterates over the values of an iterable object.
 - This loop does not work with objects unless they implement the iterable protocol.

2) Use with Arrays and Other Iterables:

- Commonly used with arrays, strings, maps, sets, and other iterable objects.

Example:

The screenshot shows a browser window with three tabs at the top: "rev_str_for_in.html U", "rev_str_for_of.html U", and "for_of.html X". The current tab is "for_of.html". The main content area displays an HTML file with a script that uses a for-of loop to log index numbers to the console. The browser's developer tools are open, showing the DOM tree on the left and the JavaScript console on the right. In the console, the output of the script is visible: 'h', 'e', and 'o' (with a count of 2) followed by a message "Live reload enabled.".

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10
11   <script>
12     var str='hello';
13     for(i of str){
14       |   console.log(i); //prints index number
15     }
16   </script>
17 </body>
18 </html>
```

TASK :

1. Reverse a string Input : hello output : olleh

The screenshot shows a browser window with two tabs open: "tr_for_in.html" and "rev_str_for_of.html". The "rev_str_for_of.html" tab is active, displaying the following HTML code:

```
<html lang="en">
<body>
    <script>
        var a=prompt("Enter a String");
        var b='';
        for(i in a){
            b=a[i]+b;
        }
        console.log(b);
    </script>

```

The browser's developer tools are open, specifically the Element tab under the Tools menu. The element inspector shows the DOM tree for the current page. At the top of the tree is the `html` element, which contains a `body` element. Inside the `body` element is a `script` element. The `script` element contains the JavaScript code shown above. The right side of the developer tools interface shows the current URL as "127.0.0.1:5161" and the status bar displays "61 x 640".

The screenshot shows a browser window with the URL `rev_str_for_of.html`. The page content is an HTML file with a script that prompts for a string and logs its reverse. The developer tools' element inspector shows the reversed string "olleh" in the DOM.

```

<html lang="en">
<body>
    <!-- TASK-1: Reverse a string Input : hello ou
    <script>
        var a=prompt("Enter a String");
        var b='';
        for(i of a){
            b=i+b;
        }
        console.log(b);
    </script>
</body>
</html>

```

2. Find the even numbers in the array - [23,54,67,64,45,95,98].

The screenshot shows a browser window with the URL `even_in_array.html`. The page content is an HTML file with a script that iterates over an array and logs even numbers. The developer tools' console tab shows the output: 54, 64, and 98.

```

<html lang="en">
<body>
    <!-- Find the even numbers in the array -
    | [ 23,54,67,64,45,95,98].
    -->
    <script>
        var a=[ 23,54,67,64,45,95,98];
        for(i of a){
            if(i%2==0){
                console.log(i);
            }
        }
    </script>
</body>
</html>

```

3. Iterate an object values & keys using a for of loop.

The screenshot shows a browser window with the URL `obj_iterate_forof.html`. The page content is an HTML file with a script that iterates over an object using `Object.entries()` and logs key-value pairs. The developer tools' console tab shows the output: name : Alice, age : 30, and profession : Engineer.

```

<html lang="en">
<body>
    <script>
        const obj = {
            name: "Alice",
            age: 30,
            profession: "Engineer"
        };
        // Using Object.entries() to get key-value pairs
        for (let [key, value] of Object.entries(obj)) {
            console.log(key,':',value);
        }
    </script>
</body>
</html>

```

Functions

A function is a reusable block of code that is used to perform a specific task when something invokes it.

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses (). The code to be executed, by the function, is placed inside curly brackets: {}

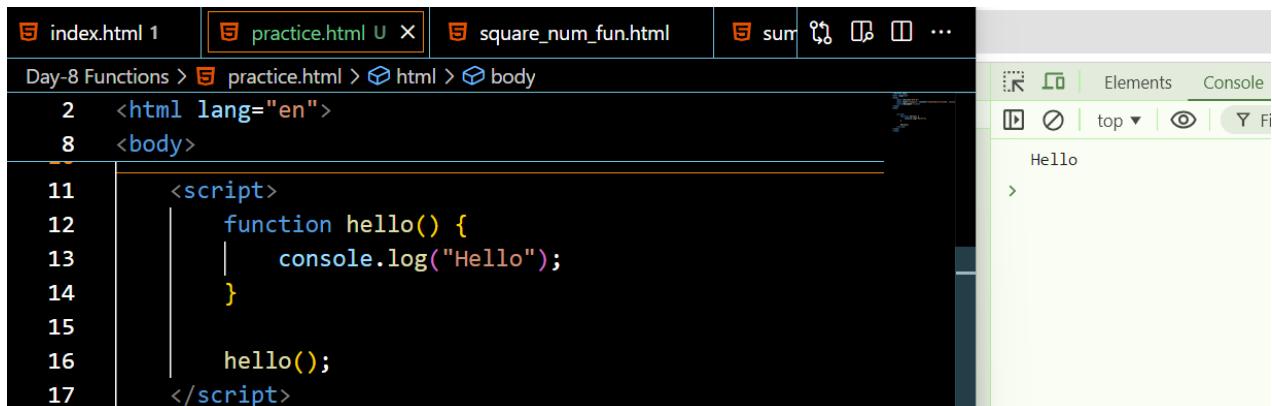
Function parameters are listed inside the parentheses () in the function definition. Function arguments are the values received by the function when it is invoked. Inside the function, the arguments (the parameters) behave as local variables.

Named functions can be hoisted

Syntax:

```
function name(params) {  
    //code to be executed  
}
```

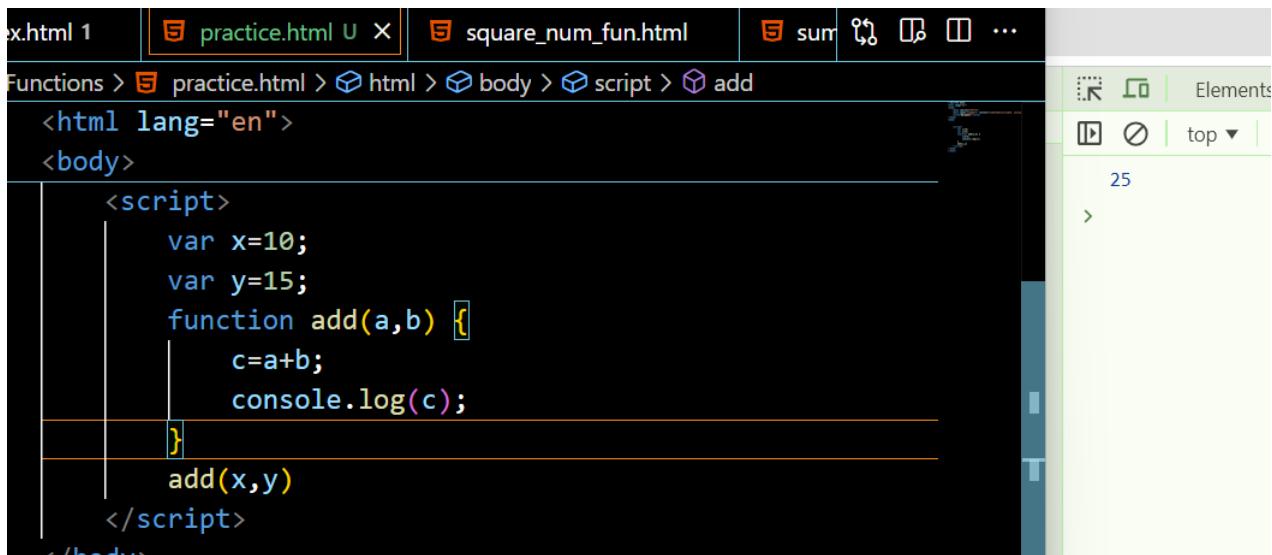
Example:



```
index.html 1 | practice.html U X | square_num_fun.html | sum ↻ ⏪ ⏴ ...  
Day-8 Functions > practice.html > html > body  
2 <html lang="en">  
8 <body>  
11 <script>  
12     function hello() {  
13         |   console.log("Hello");  
14     }  
15  
16     hello();  
17 </script>
```

The screenshot shows a browser window with developer tools open. The left pane displays the HTML code with line numbers. Lines 11-17 define a function named 'hello' that logs 'Hello' to the console. Line 16 calls this function. The right pane shows the 'Console' tab with the output 'Hello'.

Example: adding two numbers



```
index.html 1 | practice.html U X | square_num_fun.html | sum ↻ ⏪ ⏴ ...  
Functions > practice.html > html > body > script > add  
<html lang="en">  
<body>  
    <script>  
        var x=10;  
        var y=15;  
        function add(a,b) {  
            |   c=a+b;  
            |   console.log(c);  
        }  
        add(x,y)  
    </script>  
</body>
```

The screenshot shows a browser window with developer tools open. The left pane displays the HTML code with line numbers. Lines 11-17 define a function named 'add' that takes two parameters 'a' and 'b', calculates their sum, and logs the result to the console. Lines 18-20 call this function with 'x' and 'y' as arguments. The right pane shows the 'Console' tab with the output '25'.

Example: even numbers in range

The screenshot shows a browser window with several tabs at the top: "html 1", "practice.html", "square_num_fun.html", "sum", and "...". Below the tabs, the page structure is shown: "functions > practice.html > html > body > script". The script content is:

```
<html lang="en">
<body>
<script>
    function even(a) {
        for(i=1;i<=a;i++){
            if(i%2==0){
                console.log(i+"is even number");
            }
        }
    even(10);
</script>
```

To the right, the "Console" tab is selected, showing the output of the script:

```
2is even number
4is even number
6is even number
8is even number
10is even number
```

Pure function - Static function
impure function - Dynamic function.

Recursion: - When a Function calls itself again and again.

Ex:-

```
function ahello(a) {
    console.log(a);
    ahello(5);
}
ahello(5)
```

Things to remember while using functions:-

1. function names stores function definition.

```
function hello() {
    console.log ("hello world");
}

console.log(hello);
```

↳ function definition

2. Log off the function calling stores return value, or, return value stores in function calling.

Example:-

function hello() {
 console.log("hello world");
 return "hi world";
}
console.log(hello());

3. Statement after return will not execute because it was in void.

Example:-

function hello() {
 return "hello"; // it terminates from here
 console.log("This line will not print");
}

4. Named functions can only hoisted.

5. Function definition act as value because in JS functions are first class functions.

Example:-

```
var a = function hello() {  
    console.log("hello world");  
}; a(); // hello world  
console.log(a()); // hello world - hello world - hi world  
⇒ First class function - Function Expressions.
```

Types of Functions

Anonymous function:-

Anonymous function is a function defined without a name.

Example:-

```
Var a = function() {  
    console.log("hello");  
}  
a(); //hello
```

Arrow function (ES-6) Simple

Arrow function is a concise way of writing function is shorter way.

Example:-

```
Var a = () => {  
    console.log("hello");  
}  
a();  
(or)  
var a = () => console.log("hello");  
a();
```

Named function:- Normal function.

Immediately Invoked Function Expression (IIFE) {}()

An IIFE is a JS function that runs as soon as it is defined.

Example :-

Function () {

```
}); ()
```

Default Parameters :- When calling function with out arguments values are set undefined. Some times it is acceptable, but sometimes it is better to assign a default value to the parameters.

```
function greet(a = "dear") {
```

```
console.log("Hello " + a);
```

```
greet("John"); // hello John
```

great ("") ; // hello Doug

Example

function greet(a, a, b) {
 ~~ele!~~ re assign a value

console.log(a, a, b); // 3 3 5

3 \rightarrow (10:1)

Great (2, 3, 5)

Example :- to access nested Function

```
function greet() {
```

```
console.log("first function");
```

return function greater() {

Console.log ("second function");

110

Research c
-curing

Call back Function:-

call back function is a function definition passed into another function as a argument which is invoked inside the outer function to complete some kind of task.

Example:-

```
function hi () {  
    console.log ("hi");  
}  
  
function hello(a, b, c) {  
    console.log (a, b, c); // John 23 function definition  
    c(); // hi  
}  
  
hello ("John", 23, hi);
```

Example:-

```
function Payment() {  
    console.log ("Payment Received");  
}  
  
function order(a) {  
    console.log ("Order Placed");  
    a();  
}  
  
order (Payment);
```

Higher Order function (hof):-

hof is function which takes one or more functions as arguments is called hof.

order (Payment);
↓ ↓
hof. call back

Tasks

Task 1: Greeting Function

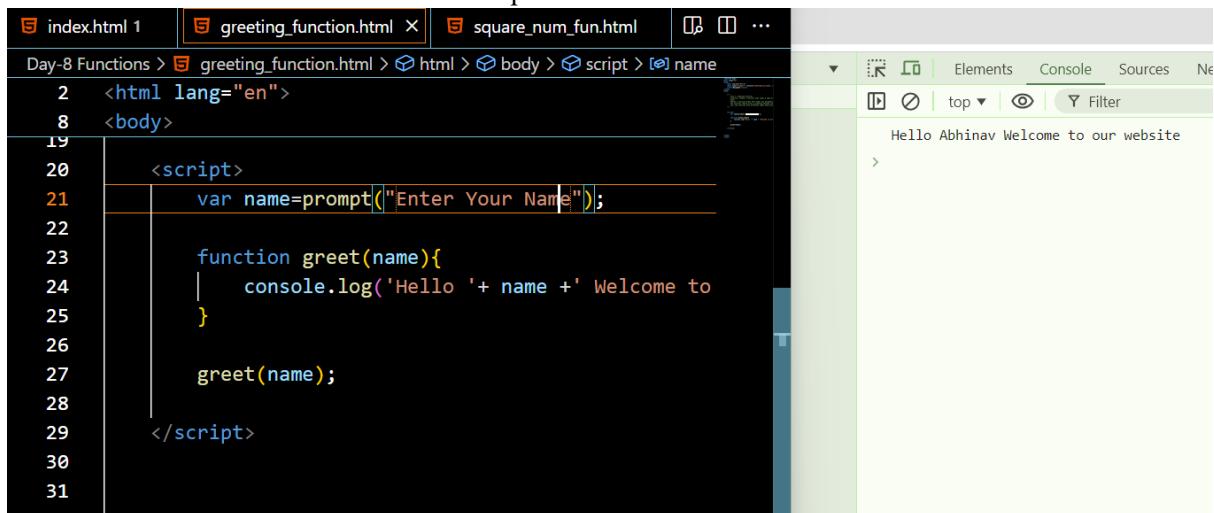
Scenario: Create a function that takes a name as an argument and returns a greeting message.

Task:

Define a function greet that takes one parameter name.

The function should print a greeting message like “Hello, [name]!”.

Call the function with different names and print the results.



A screenshot of a browser developer tools console. The left pane shows the code for `greeting_function.html`. The right pane shows the output of the `console.log` statement, which reads "Hello Abhinav Welcome to our website".

```
index.html 1 | greeting_function.html X | square_num_fun.html | ...  
Day-8 Functions > greeting_function.html > html > body > script > name  
2 <html lang="en">  
8 <body>  
19 <script>  
21     var name=prompt("Enter Your Name");  
22  
23     function greet(name){  
24         console.log('Hello ' + name + ' Welcome to')  
25     }  
26  
27     greet(name);  
28  
29 </script>  
30  
31  
32
```

Task 2: Sum Function

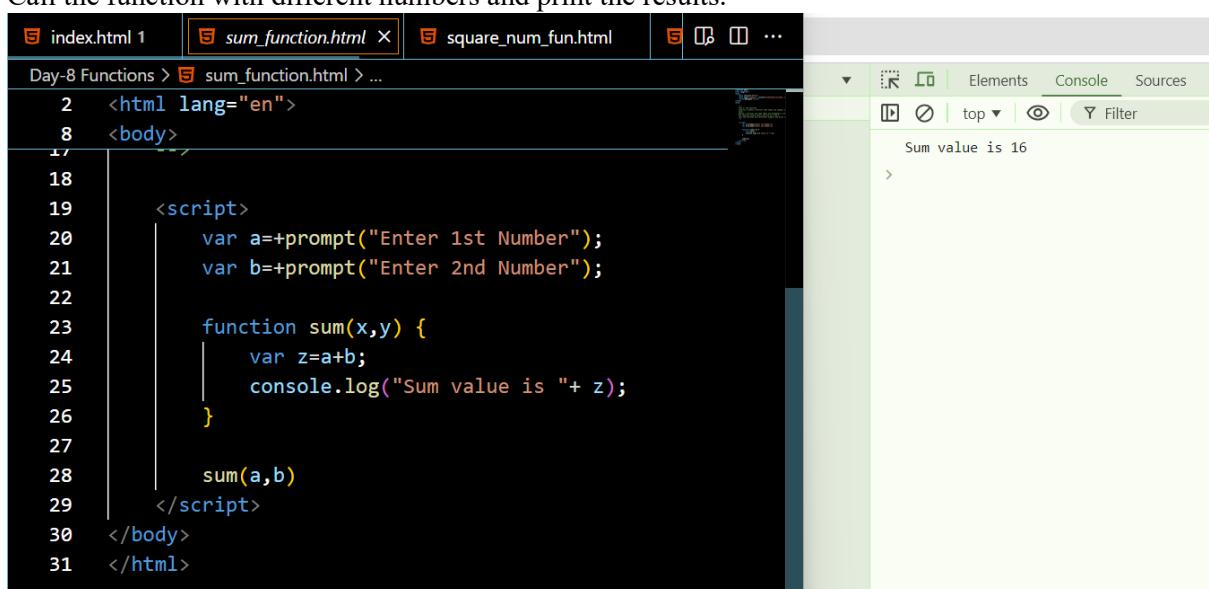
Scenario: Create a function that takes two numbers as arguments and returns their sum.

Task:

Define a function sum that takes two parameters a and b.

The function should return the sum of a and b.

Call the function with different numbers and print the results.



A screenshot of a browser developer tools console. The left pane shows the code for `sum_function.html`. The right pane shows the output of the `console.log` statement, which reads "Sum value is 16".

```
index.html 1 | sum_function.html X | square_num_fun.html | ...  
Day-8 Functions > sum_function.html > ...  
2 <html lang="en">  
8 <body>  
17 <script>  
18     var a=+prompt("Enter 1st Number");  
19     var b=+prompt("Enter 2nd Number");  
20  
21     function sum(x,y) {  
22         var z=a+b;  
23         console.log("Sum value is "+ z);  
24     }  
25  
26     sum(a,b)  
27     </script>  
28 </body>  
29 </html>
```

Task 3: Square Function

Scenario: Create a function that takes a number as an argument and returns its square.

Task:

Define a function square that takes one parameter num.

The function should return the square of num.

Call the function with different numbers and print the results.

The screenshot shows a browser window with several tabs at the top: index.html 1, square_num_fun.html (active), avg_function.html, and others. The title bar indicates "Day-8 Functions > square_num_fun.html > html > body > script > sum". The code editor contains the following JavaScript:

```

2 lang="en">
8
19 <script>
20     var a=+prompt("Enter 1st Number");
21
22     function sum(a) {
23         var x=a**2;
24         console.log("Square number of "+ a +" is "+ x);
25     }
26
27     sum(a)
28 </script>
29
30

```

The browser's developer tools console on the right shows the output of the console.log statement: "Square number of 7 is 49".

Task 4: Average Function

Scenario: Create a function that takes an array of numbers as an argument and returns the average.

Task:

hint:[2,5,2] =9/3=3

Define a function average that takes one parameter arr.

The function should return the average of the numbers in arr.

Call the function with different arrays and print the results.

The screenshot shows a browser window with several tabs at the top: index.html 1, avg_function.html (active), vowels.html, and others. The title bar indicates "Day-8 Functions > avg_function.html > html > body > script > avg". The code editor contains the following JavaScript:

```

2 lang="en">
8
23 <script>
24     var len=+prompt("Enter Length of the string");
25     var a=[];
26     var sum=0;
27     for(i=0;i<len;i++){
28         a[i]= +prompt("");
29     }
30     console.log("Given Array is "+a);
31
32     function avg(a) {
33         for(i=0;i<len;i++){
34             sum=sum+a[i];
35         }
36         var avg=sum/len;
37         console.log("avg value of given numbers "+avg);
38     }
39     avg(a);
40

```

The browser's developer tools console on the right shows the output of the console.log statements: "Given Array is 4,5,6" and "avg value of given numbers 5".

Task 5: Vowels Function

Scenario: Create a function that takes a string as an argument and returns whether the string contains vowels or not .

Task:

Define a function that takes one parameter str.

The function should return whether string contains vowels or not.

use loops and if conditions

Call the function with different strings and print the results.

hello --a,e ,i ,o,u---it contains vowels

hll---it doesnt contains vowels

The screenshot shows a browser developer tools interface. On the left, the code for `vowels.html` is visible:

```
<html lang="en">
<body>
<script>
    var str1=prompt("Enter a word to check it is vowel or not");
    console.log("Entered word is:"+str1);
    var a=str1.toLowerCase();

    function vowel(a) {
        for(i=0;i<a.length;i++){
            if(a[i]=="a" || a[i]=="e" || a[i]=="i" || a[i]=="o" || a[i]=="u"){
                window.alert("Given Word Consists Vowels");
                console.log("Given Word Consists Vowels");
                break;
            }
            else{
                window.alert("it doesn't contain vowels");
                console.log("it doesn't contain vowels");
                break;
            }
        }
        vowel(a);
    }
</script>
```

On the right, the browser's console output is shown:

```
Entered word is:Abhinav
Given Word Consists Vowels
```

The screenshot shows a browser developer tools interface. On the left, the code for `vowels.html` is visible, identical to the one above:

```
<html lang="en">
<body>
<script>
    var str1=prompt("Enter a word to check it is vowel or not");
    console.log("Entered word is:"+str1);
    var a=str1.toLowerCase();

    function vowel(a) {
        for(i=0;i<a.length;i++){
            if(a[i]=="a" || a[i]=="e" || a[i]=="i" || a[i]=="o" || a[i]=="u"){
                window.alert("Given Word Consists Vowels");
                console.log("Given Word Consists Vowels");
                break;
            }
            else{
                window.alert("it doesn't contain vowels");
                console.log("it doesn't contain vowels");
                break;
            }
        }
        vowel(a);
    }
</script>
```

On the right, the browser's console output is shown:

```
Entered word is:fry
it doesn't contain vowels
```

Task 6: Temperature Converter

Scenario: Create a function that converts temperatures between Celsius and Fahrenheit.

Task:

Define a function `convertTemperature` that takes two parameters: `temp` (the temperature) and `scale` (the scale to convert to, either "C" or "F").

The function should return the converted temperature.

Test the function with different temperatures and scales and print the results.

Hints:-

formula for celscious $(temp - 32) * 5/9$

formula for fahrenheit $(temp * 9/5) + 32$

The screenshot shows a browser window with three tabs: index.html, vowels.html, and temp_converter.html. The temp_converter.html tab is active, displaying a script that converts temperature between Fahrenheit and Celsius. The script uses prompts to get user input for the temperature and unit, and then calls a function to perform the conversion. The function uses an if-else-if ladder to handle both units. The browser's developer tools console is open, showing the input 'f' and '85', and the output 'Your Temperature is converted into Fahrenheit: 185'. This indicates a bug where the script converts Fahrenheit to Fahrenheit instead of Celsius.

```
2 <html lang="en">
8 <body>
21 <script>
22     var a=prompt("if you need to convert fahrenheit to celsius press c (or) need");
23     var b=prompt("Enter Your Temperature");
24     console.log(a);
25     console.log(b);
26
27     function temp(a,b) {
28         if(a=='c'){
29             t=(b-32)*5/9;
30             console.log("Your Temperature is converted into Celsious: "+t);
31         }
32         else if(a=='f'){
33             t=(b*9/5)+32;
34             console.log("Your Temperature is converted into Fahrenheit: "+t);
35         }
36         else{
37             console.log("Enter Your Correct Details");
38         }
39     }
40     temp(a,b);
41
42 </script>
43 </body>
44 </html>
```

The screenshot shows a browser window with three tabs: index.html, vowels.html, and temp_converter.html. The temp_converter.html tab is active, displaying the same script as the previous screenshot. The browser's developer tools console is open, showing the input 'c' and '235', and the output 'Your Temperature is converted into Celsious: 112.77777777777777'. This indicates a bug where the script converts Celsius to Celsius instead of Fahrenheit.

```
2 <html lang="en">
8 <body>
21 <script>
22     var a=prompt("if you need to convert fahrenheit to celsius press c (or) need");
23     var b=prompt("Enter Your Temperature");
24     console.log(a);
25     console.log(b);
26
27     function temp(a,b) {
28         if(a=='c'){
29             t=(b-32)*5/9;
30             console.log("Your Temperature is converted into Celsious: "+t);
31         }
32         else if(a=='f'){
33             t=(b*9/5)+32;
34             console.log("Your Temperature is converted into Fahrenheit: "+t);
35         }
36         else{
37             console.log("Enter Your Correct Details");
38         }
39     }
40     temp(a,b);
41
42 </script>
43 </body>
44 </html>
```

Global Execution Context

Global execution context:

Global Execution Context is the first context that gets created when the JavaScript engine starts executing code.

Key Components

1. Memory Allocation (Creation Phase):

- During this phase, the engine sets up the memory for variables and functions.
- **Variables** declared with `var` are hoisted and initialized with `undefined`.
- **Function declarations** are hoisted and their definitions are stored in memory.
- Variables declared with `let` and `const` are also hoisted but are not initialized. They remain in a temporal dead zone until they are assigned a value

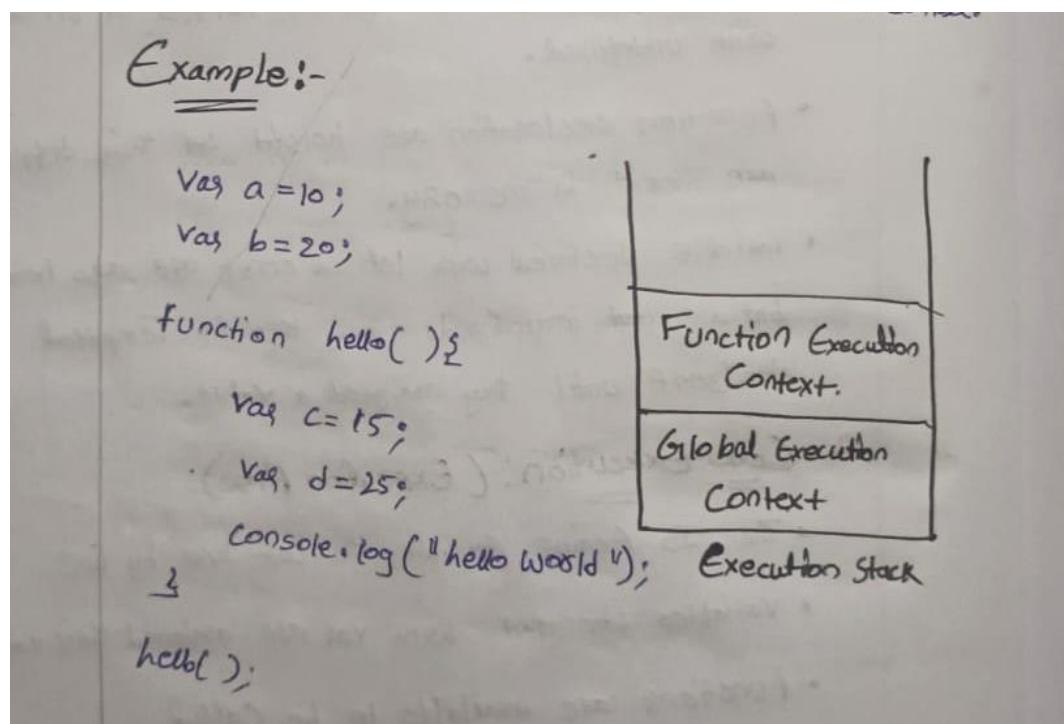
2. Code Execution (Execution Phase):

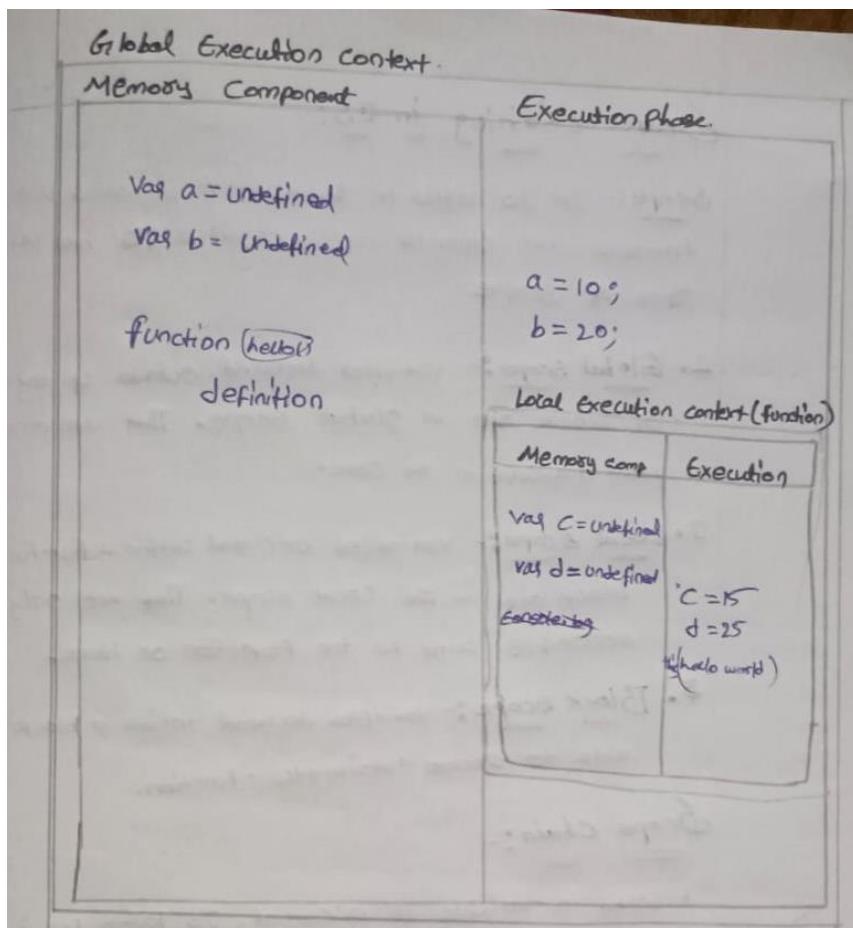
- The JavaScript engine executes the code line by line.
- Variables declared with `var` are assigned their values.
- **Functions** are available to be called.
- Variables declared with `let` and `const` are assigned values when their declaration is encountered in the code.

How it works on the functions

When a function is invoked, a new **Execution Context(function execution context)** is created specifically for that function. This context is separate from the Global Execution Context but follows similar principles.

- Each function invocation creates a new execution context.
- Variables declared inside a function are local to that function and are not accessible outside it.
- The scope chain allows inner functions to access variables from their parent functions and the global context.





Closures

A closure is a function that has access to its own scope, the scope of the outer function, and the global scope. This means a closure can remember and access variables from its outer function even after that function has finished executing.

(Or)

When an inner function has access to the variable of outer function even after the outer function has been executed.

```
body>
<script>
    function outer() {
        var a=10;
        console.log("Hello World");
        function inner(params) {
            var b=15;
            console.log(a);
        }
        inner();
    }
    outer();

```

Output in browser:

```
Hello World
10
```

Scope Chaining in JavaScript

Scope in JavaScript refers to the context in which variables, functions, and objects are accessible. JavaScript has three types of scope:

1. **Global Scope:** Variables declared outside of any function or block are in the global scope. They are accessible from anywhere in the code.
2. **Local Scope:** Variables declared within a function or block are in the local scope. They are only accessible within that function or block.
3. **Block Scope:** Variables declared inside a block can't able to access outside of the function

Scope Chain:

- When a variable is accessed, JavaScript looks for it in the current scope.
- If the variable is not found, it looks in the outer scope.
- This process continues until it reaches the global scope.
- If the variable is not found in any scope, it results in a `ReferenceError`

Example:

```
ml 1 practice.html ✘ square_num_fun.html sum ⌂ ⌂ ...  
options > practice.html > html > body > script  
html lang="en">  
body>  
<script>  
    var a=10;  
    function outer() {  
        var b=12;  
        function inner() {  
            var c=15;  
            function inner2() {  
                var d=20;  
                console.log(a);  
            }  
            inner2();  
        }  
        inner();  
    }  
    outer();
```

Console output: 10

Lexical Scoping:

- JavaScript uses lexical scoping, meaning that the scope of a variable is determined by its position in the source code.
- Inner functions have access to variables declared in their outer functions (but not vice versa).

