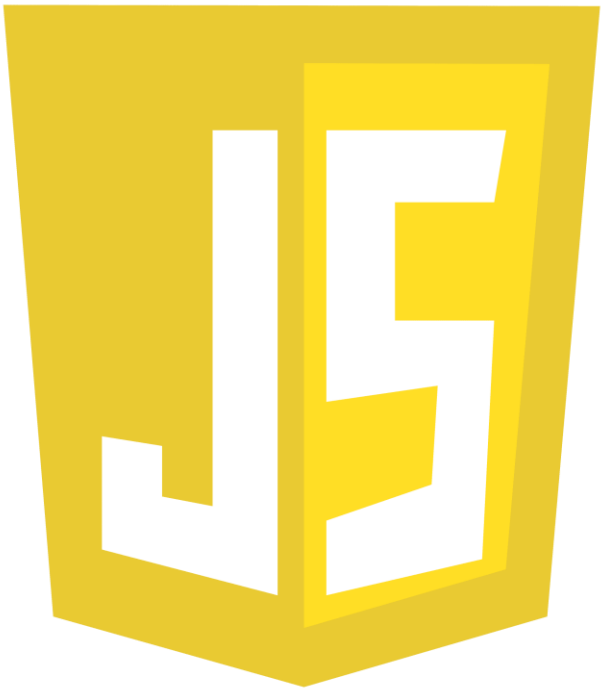
JAVA SCRIPT (JS)

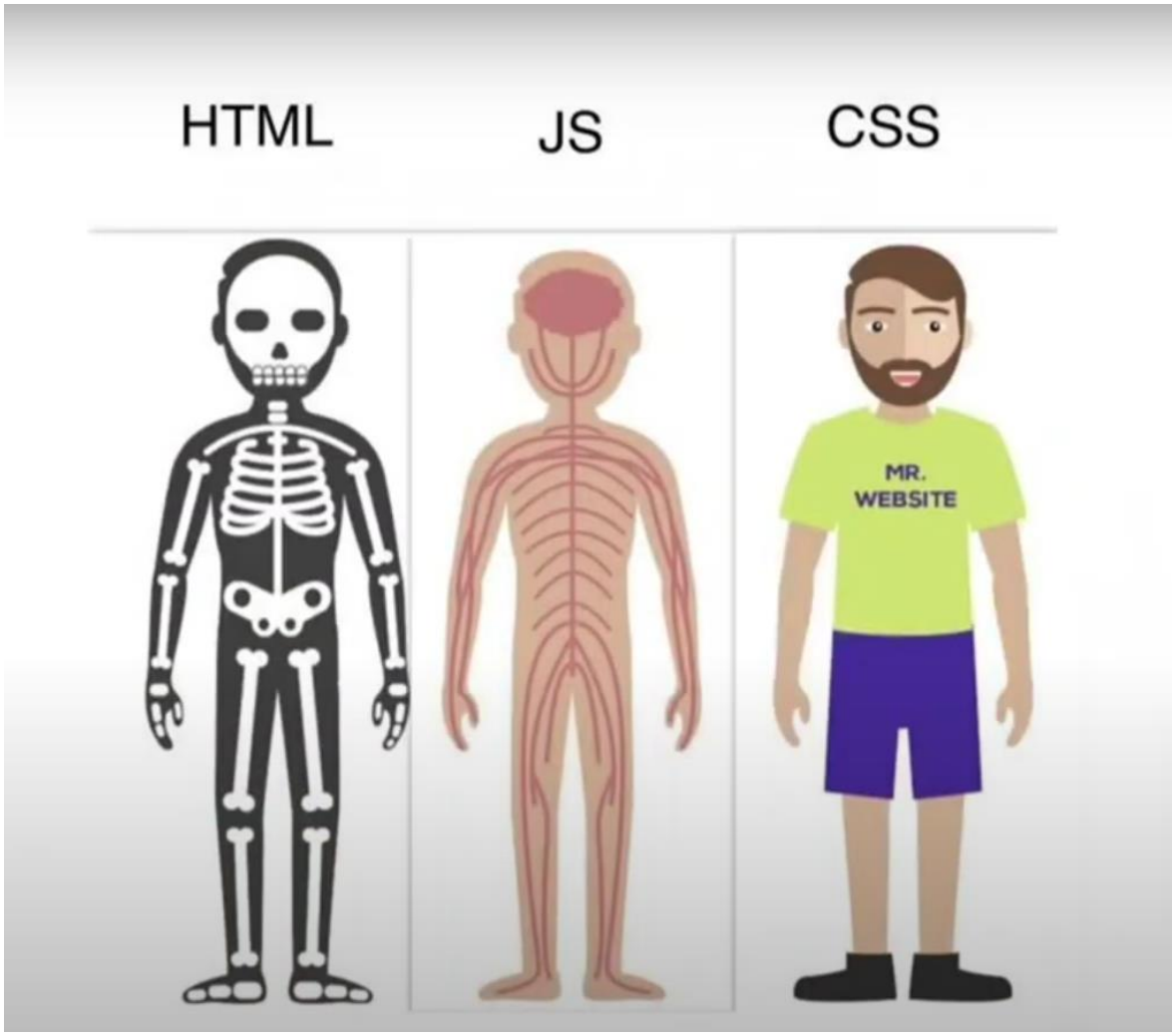


- **Abhinav**

Java Script

1. why we need java script?

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



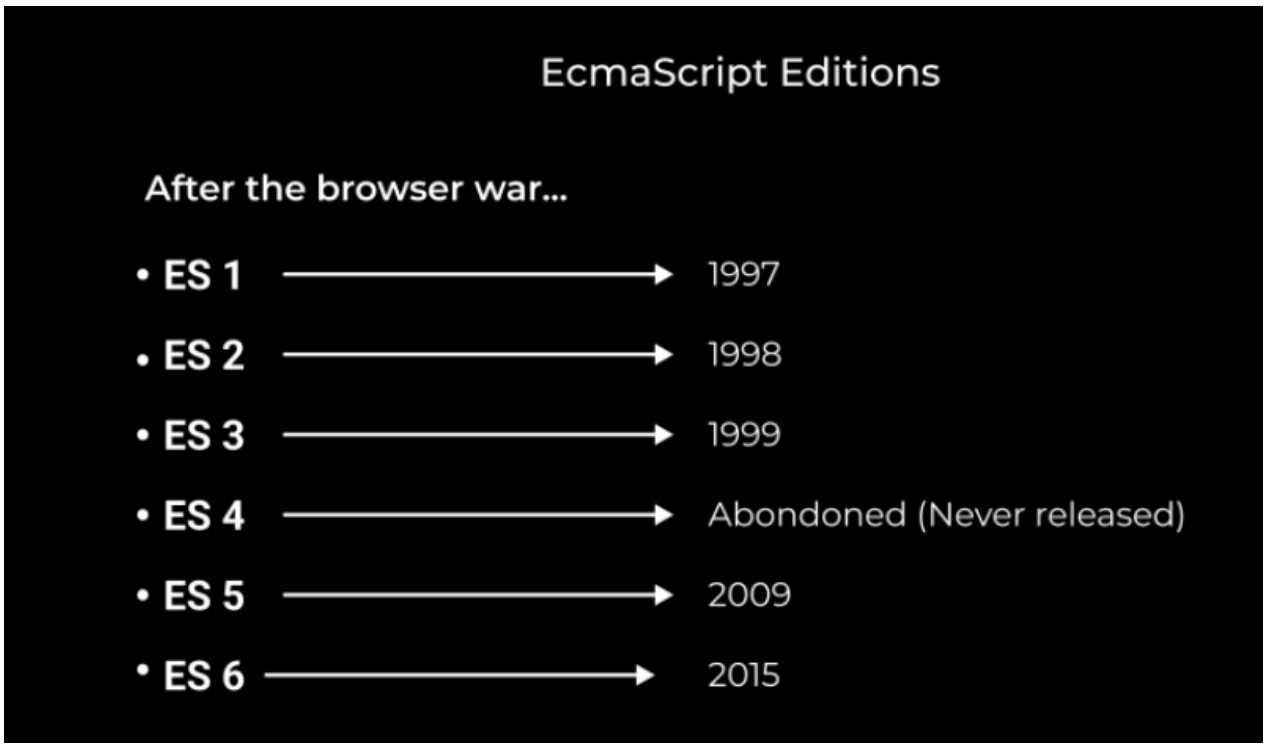
2. History of java script?

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

What Is ECMAScript?

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

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

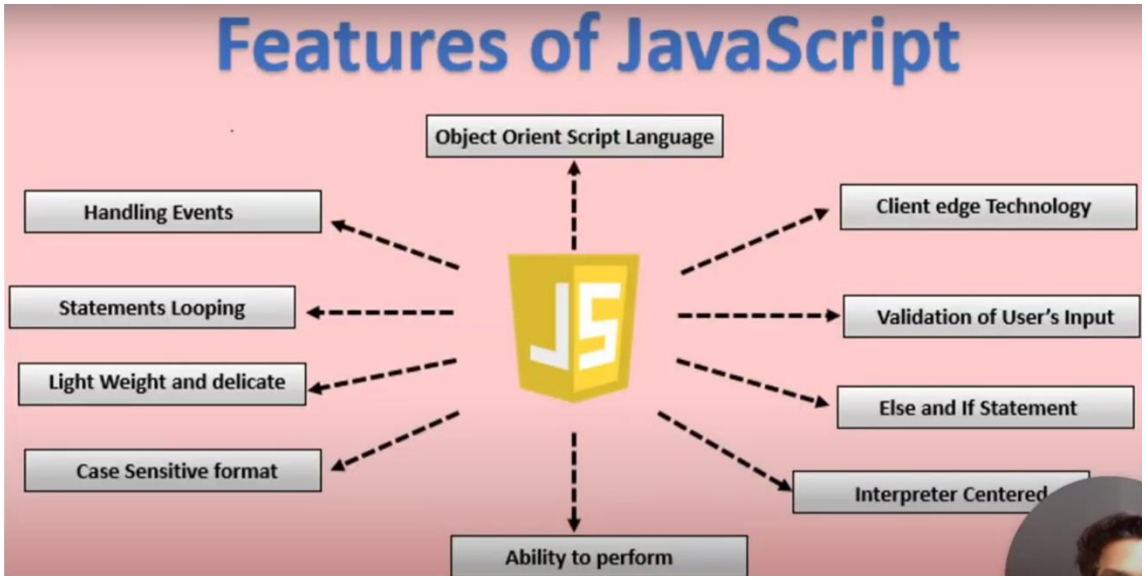


Originally, the name ECMAScript was just the formalization of JavaScript, but now languages like JScript and ActionScript are also based on the ECMAScript standard.

3. What is java script?

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

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



- **High-Level Language**

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

- **Cross-platform**

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

- **Single-theaded**

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

- **Asynchronous**

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

- **Synchronous**

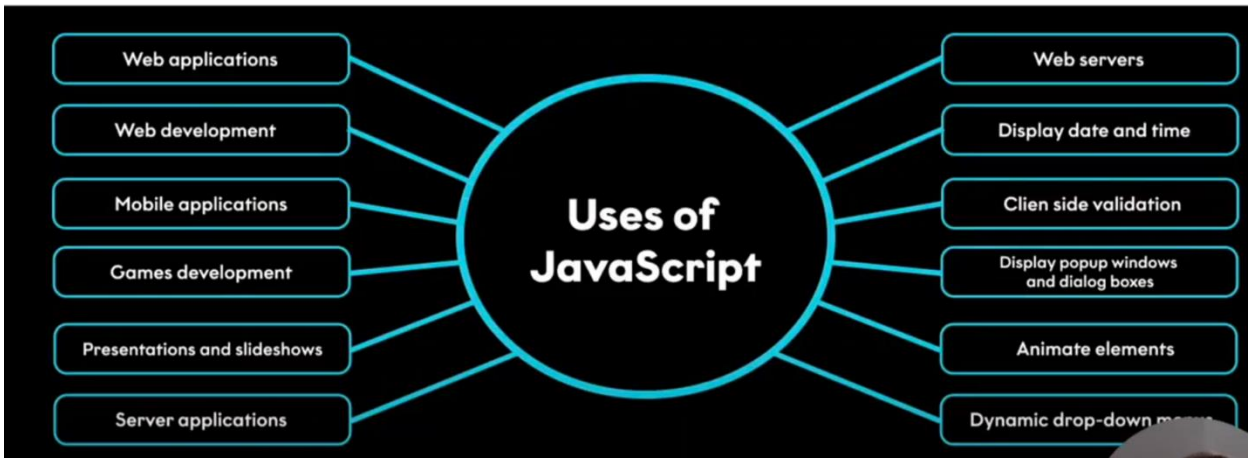
  Synchronous means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

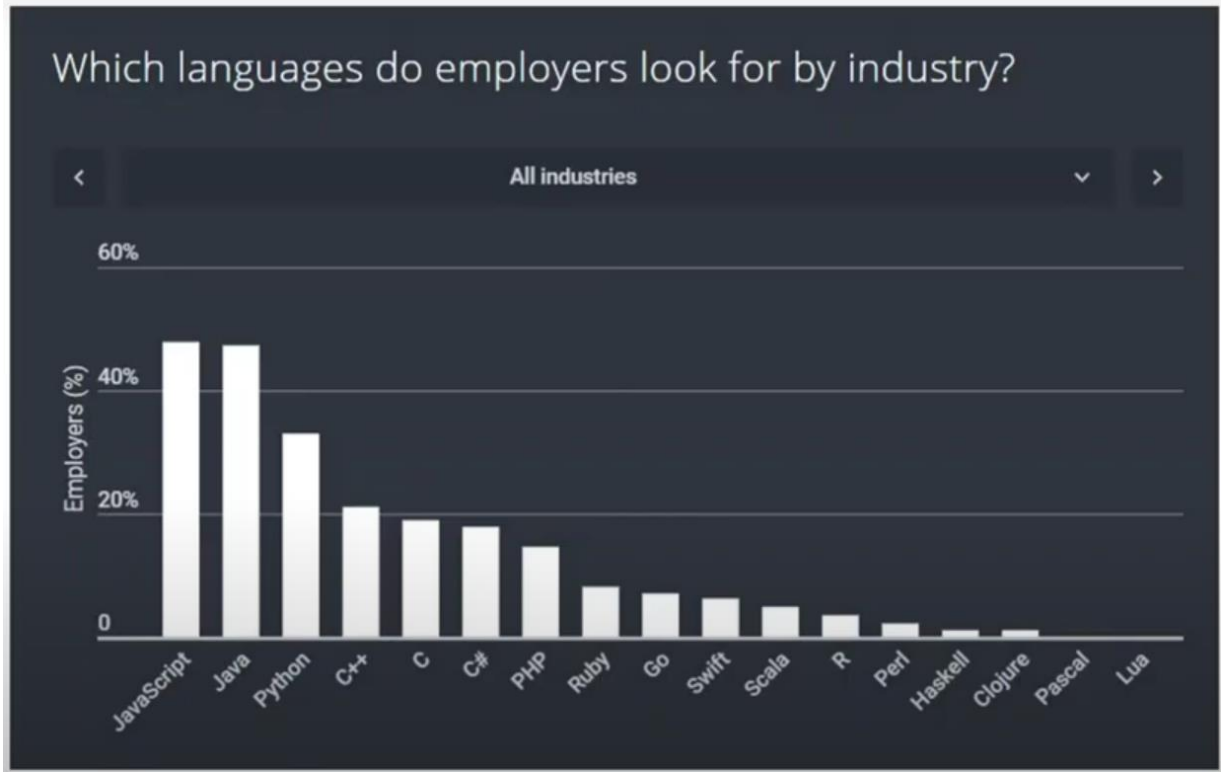
- **Obeject Oriented**

It provides an overview of the basic concepts of OOP.

- **Scripting**

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





4. What is Vanilla JavaScript

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

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

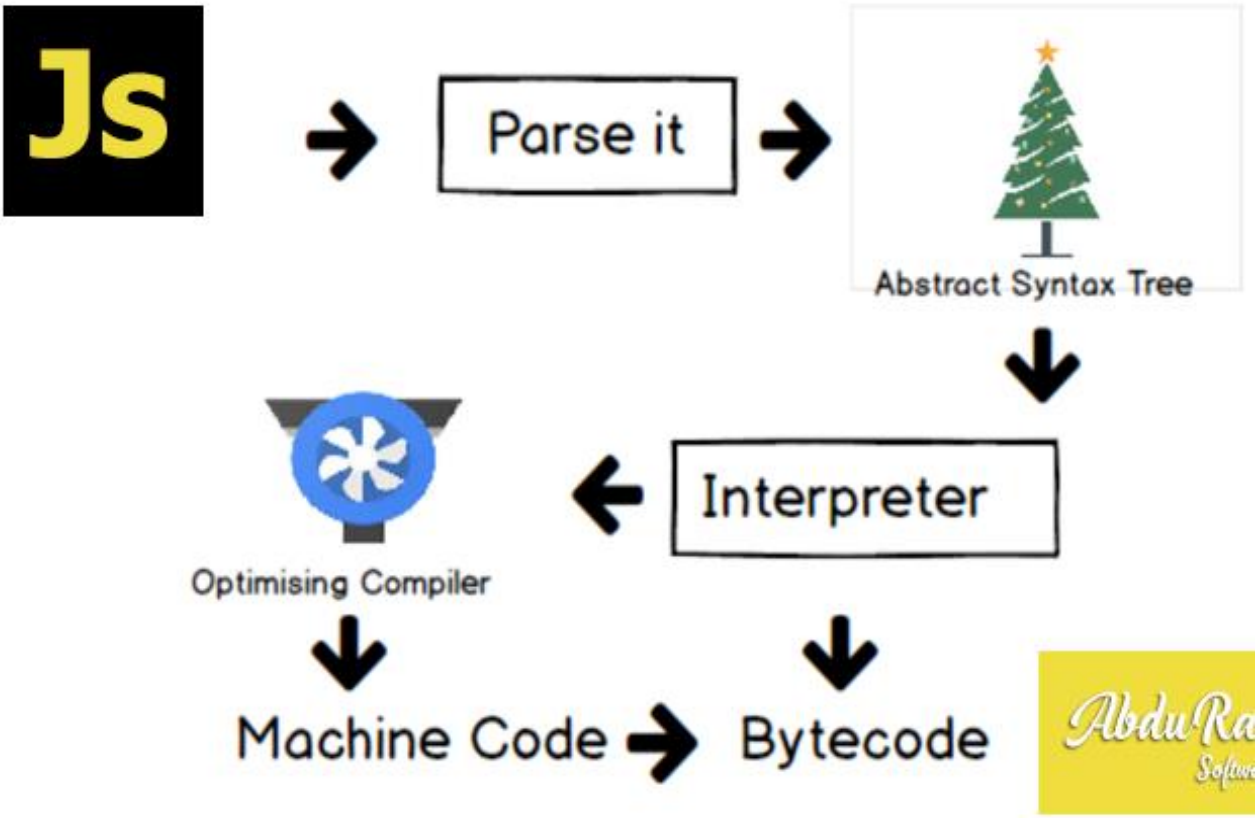
The team of developers that created the vanilla JavaScript is continuously working on it to improve it and make it more useful for the web-developers.

5. why JavaScript programming language?

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

6. What is a JavaScript Engine?

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



Parsing

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

Compilation

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

Execution

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

7. Java script Run time environment?

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

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

call stack:

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

Heap:

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

Web API:

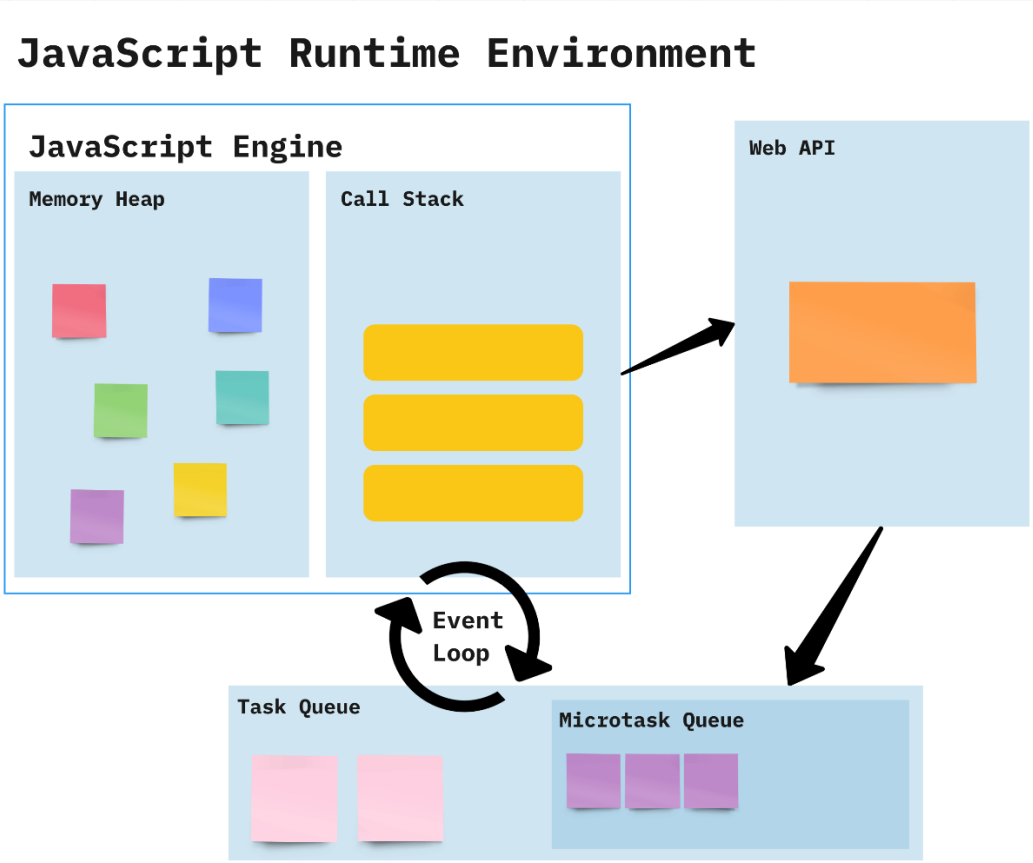
A Web API (Application Programming Interface) is a set of rules and tools that allows different software applications to communicate with each other over the web. It acts as an intermediary, enabling one application to interact with another application's data or functionality using standard web protocols,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

Event loop:

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

Synchronous vs. Asynchronous

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



How it Works:

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

Difference Between Compiler and Interpreter

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

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

How Many Ways To Insert JS

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

1. **internal JS:**

   By using script tag at the bottom of the document

   Syntax:

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

2. **inline JS:**

   we can apply inline js within the element.

   Syntax:

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

3. **external JS:**

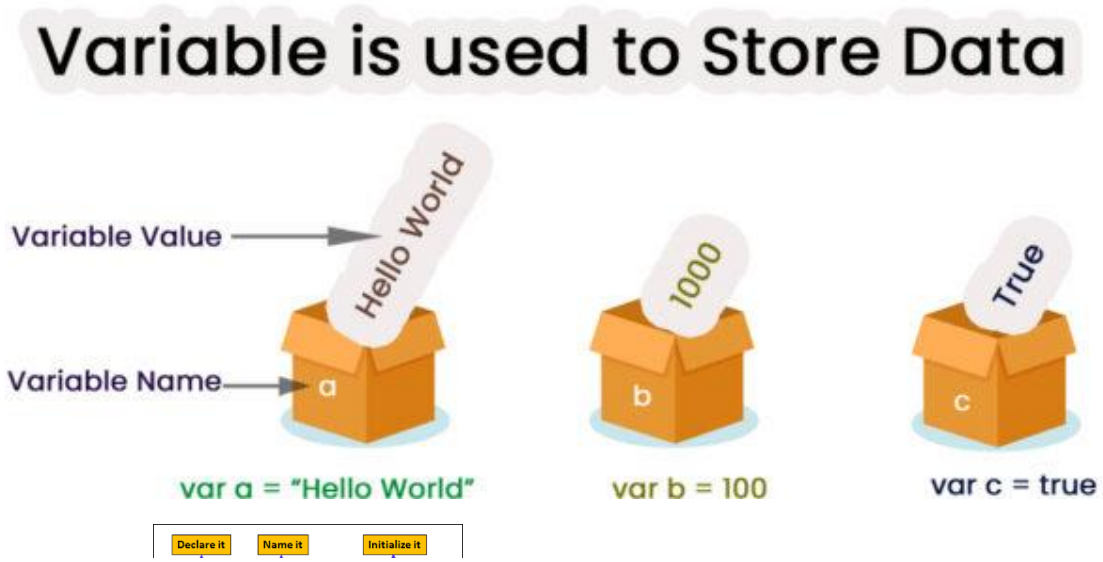
   By creating a js file with .js extention we can insert js file into the html document. That js file is linked in the script tag.

   Syntax:

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

Variables:

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



JavaScript Variables can be declared in 4 ways:

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

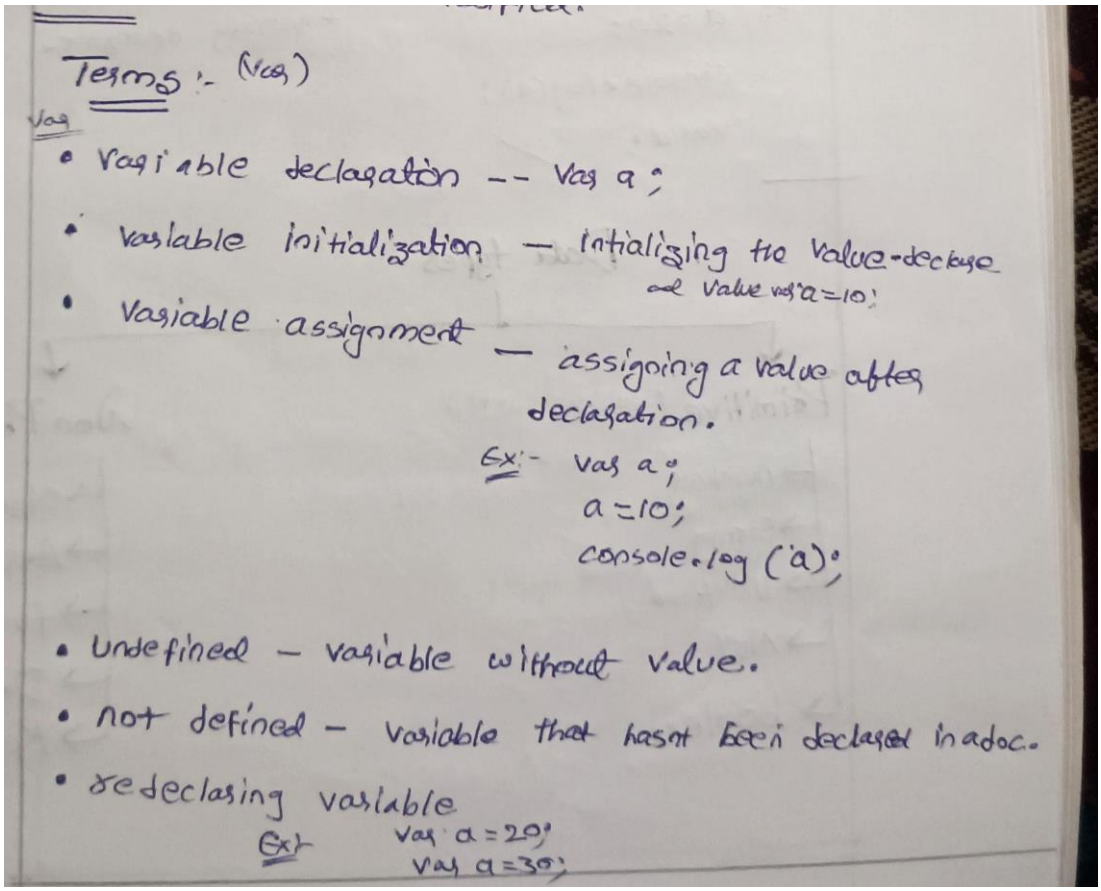
Example:

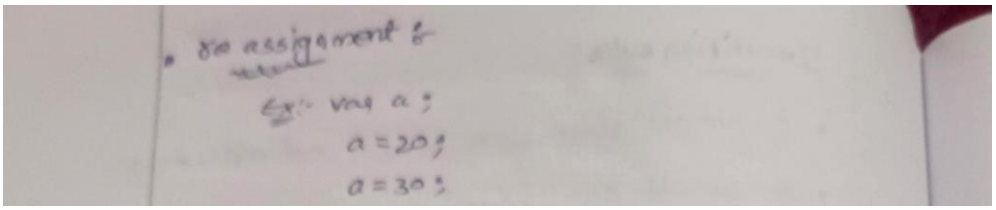
```
a=10;
Var b=20;
let c=5;
Const d=15;
console.log(a);     //10
console.log(a);    //20
console.log(a);   //5
console.log(a);   //15
```

Rules for Identifiers:

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

Terms:





Dynamic typing:

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

Hoisting

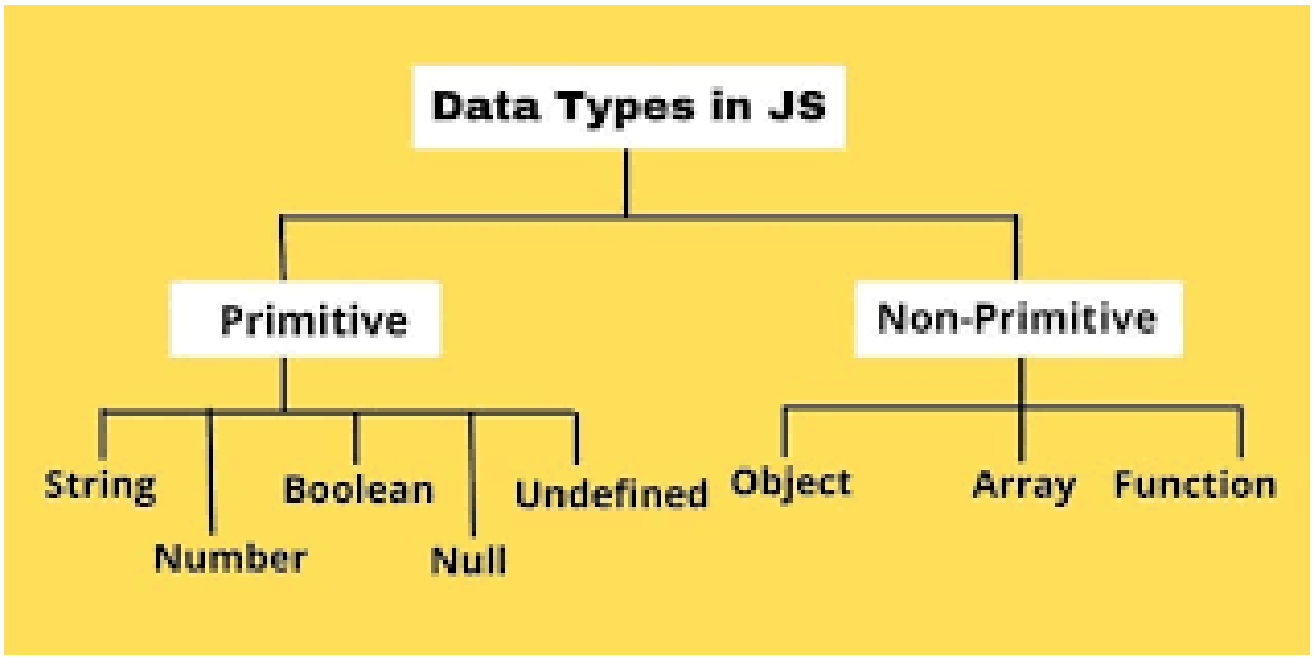
- It is a behaviour where the declaration of the variable and functions are moved to the top even before the execution.
- Only Declaration is hoisted not the Initialization
- Only works in **var** remaining **let** and **const** goes to temporary deadzone

Example:

```
a=20;
Console.lob(a);     //20
var a;
```

Data types

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



Primitive Data Types :-

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

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

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

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

Primitive

- Numbers :- represents numeric values

  Eg :- var a = 100;

- String :- represents Group of characters

  eg :- var b = "Hello"

- Boolean :- represents boolean value either true - 1 or false - 0

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

Non primitive

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

Examples :-

primitive

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

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

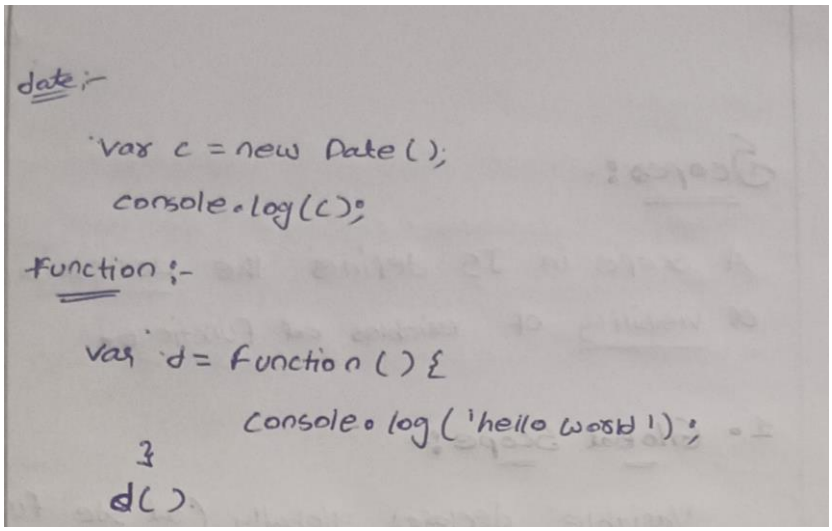
non primitive

arrays (number index)

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

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

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

objects (named index)

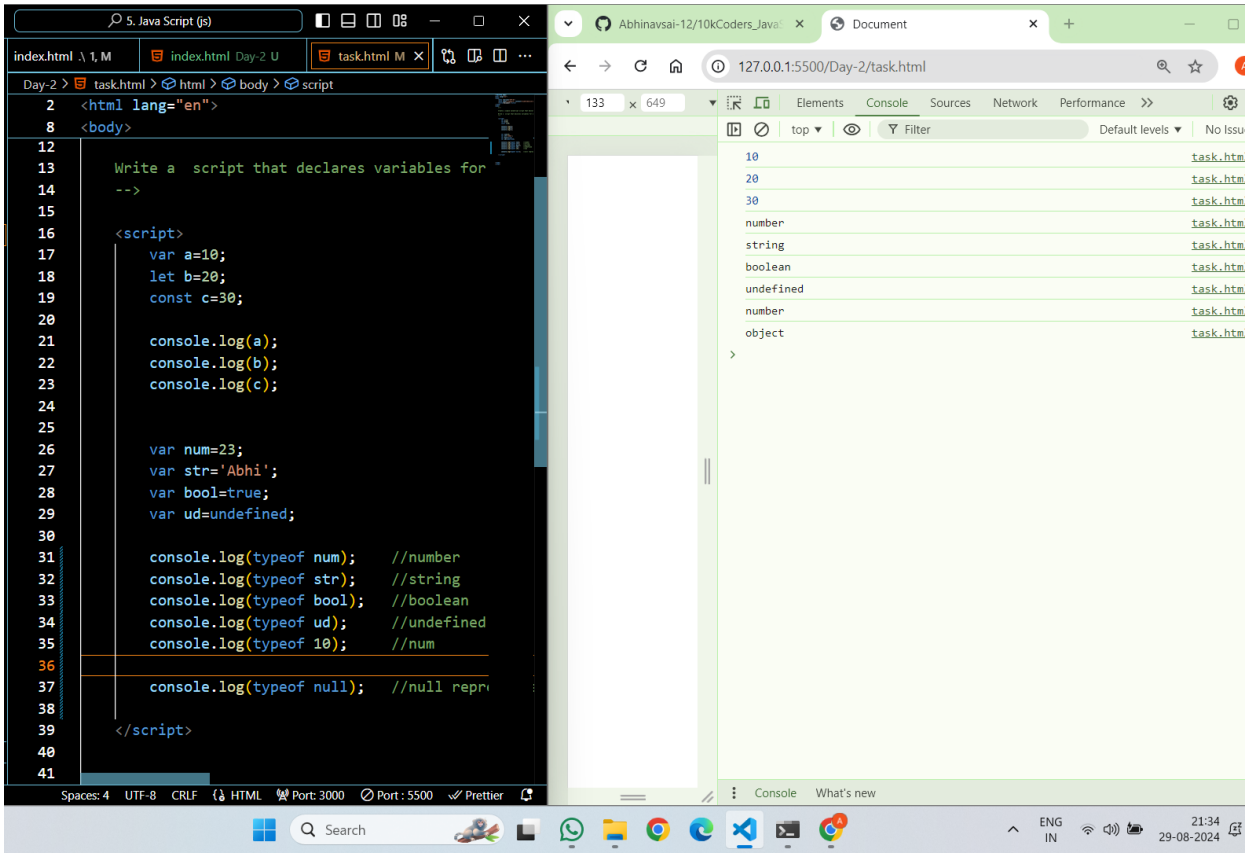
```
var b={
        id:1201,
        name: 'abhi',
        age: 22
    }
Consde.log(b.age);  //22
```



Typeof:

The typeof operator returns the data type of a variable.

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



Scopes

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

1. Global Scope:

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

   Var have global Scope and Function Scope.

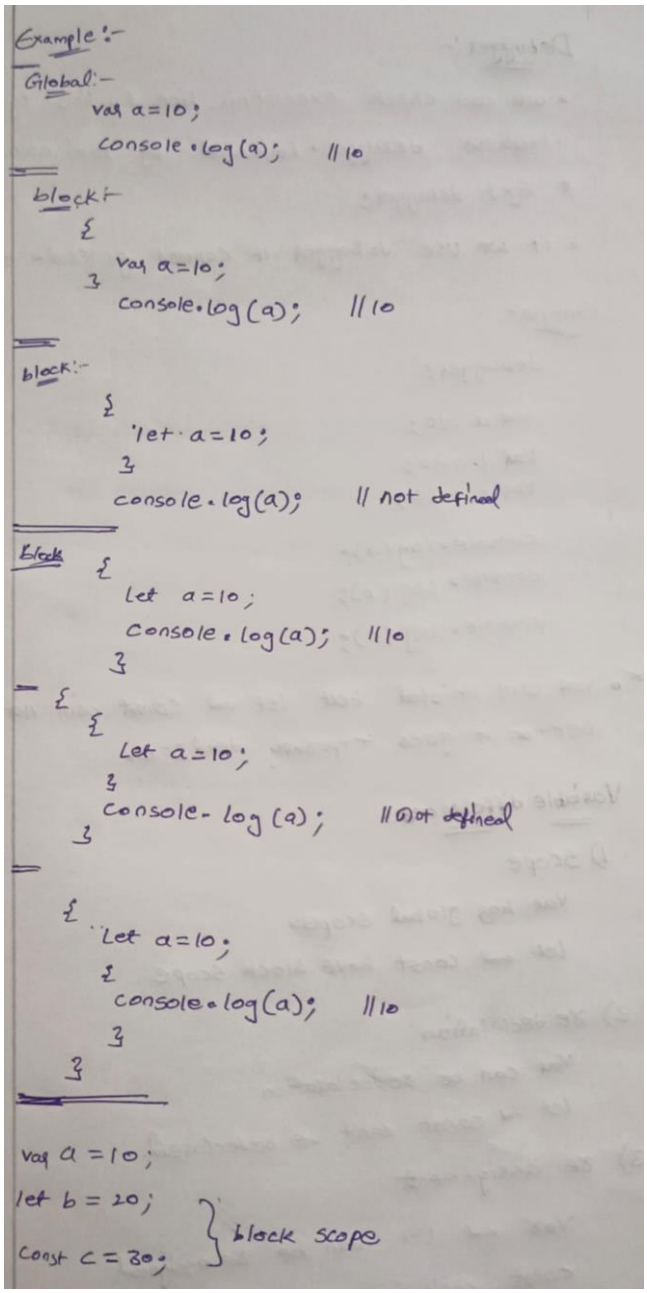
2. Block Scope:

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

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

3. Local Scope:

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



Debugger

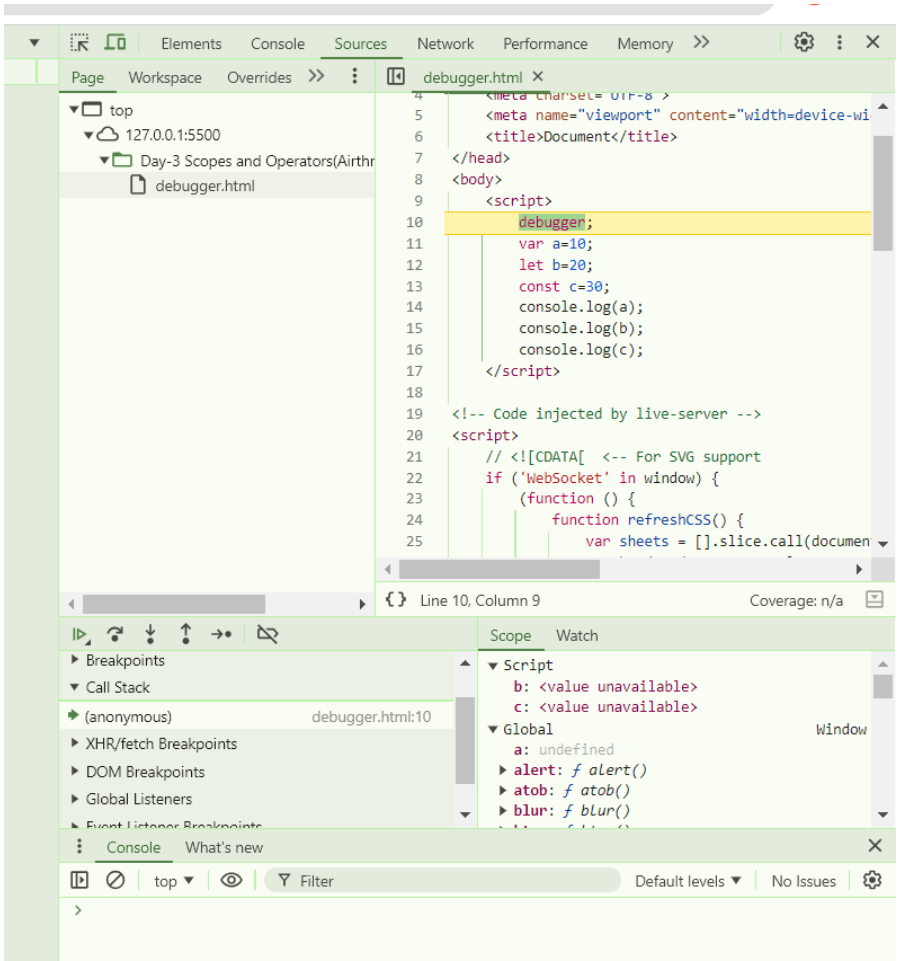
We can check Execution line by line by using keyword debugger followed by semi colon (:)

syntax:

```
debugger;
```

Example:

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



Variable Difference:

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

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

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

Operators

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

(or)

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

Types:

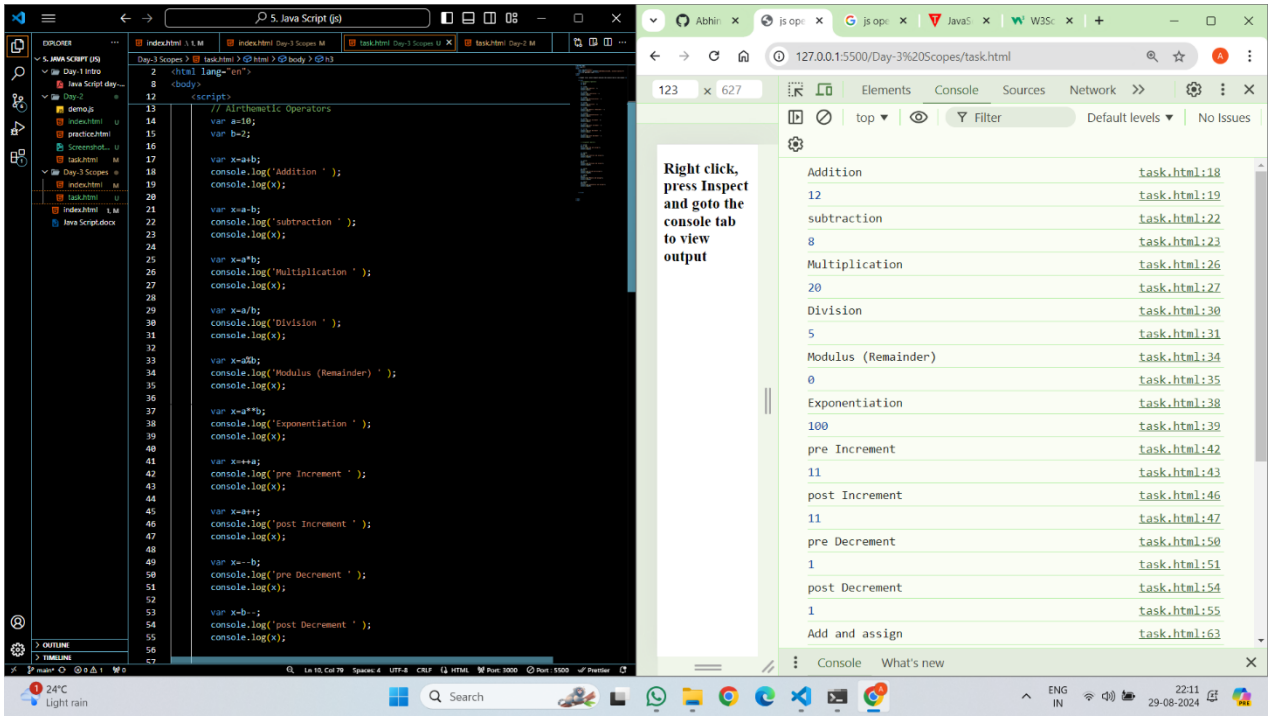
1. Airthmetic Operators
2. Assignment Operator
3. Comparision Operator
4. Logical Operator
5. Ternary Operator
6. Bitwise Oberator
7. String Operator
8. Typeof Operator

Arithmetic operators

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

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

Example:

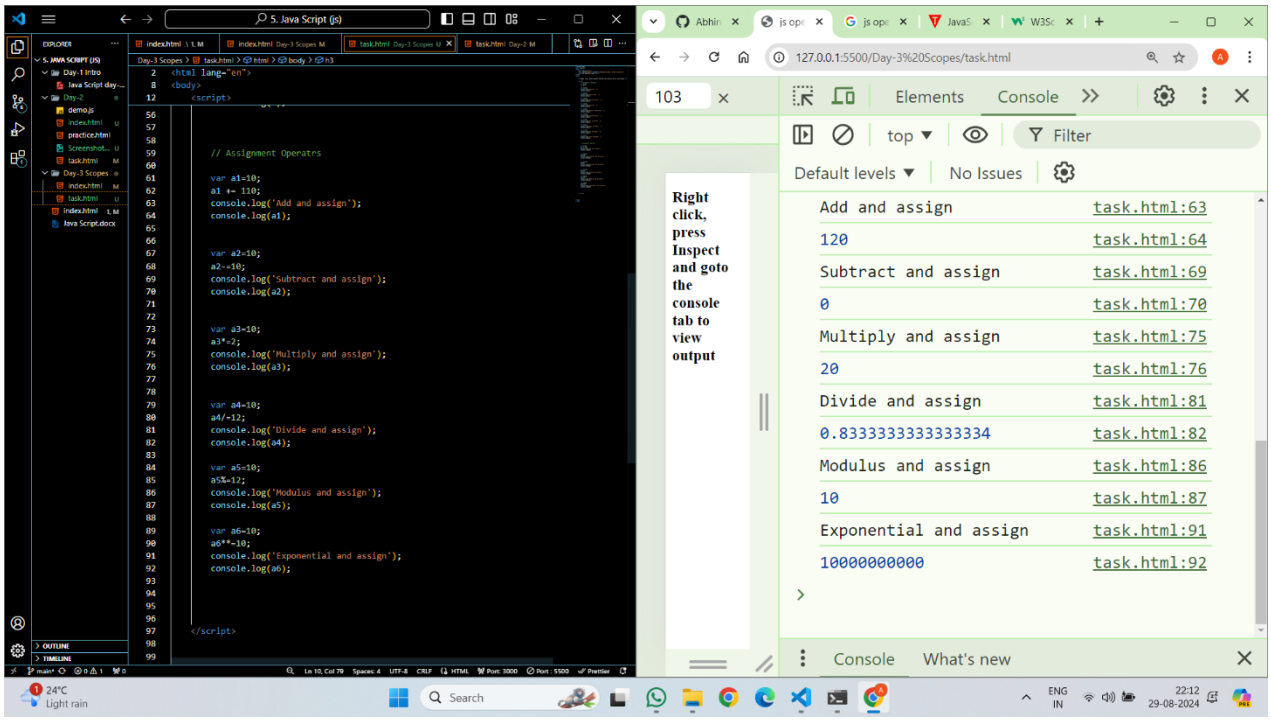


Assignment Operators:

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

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

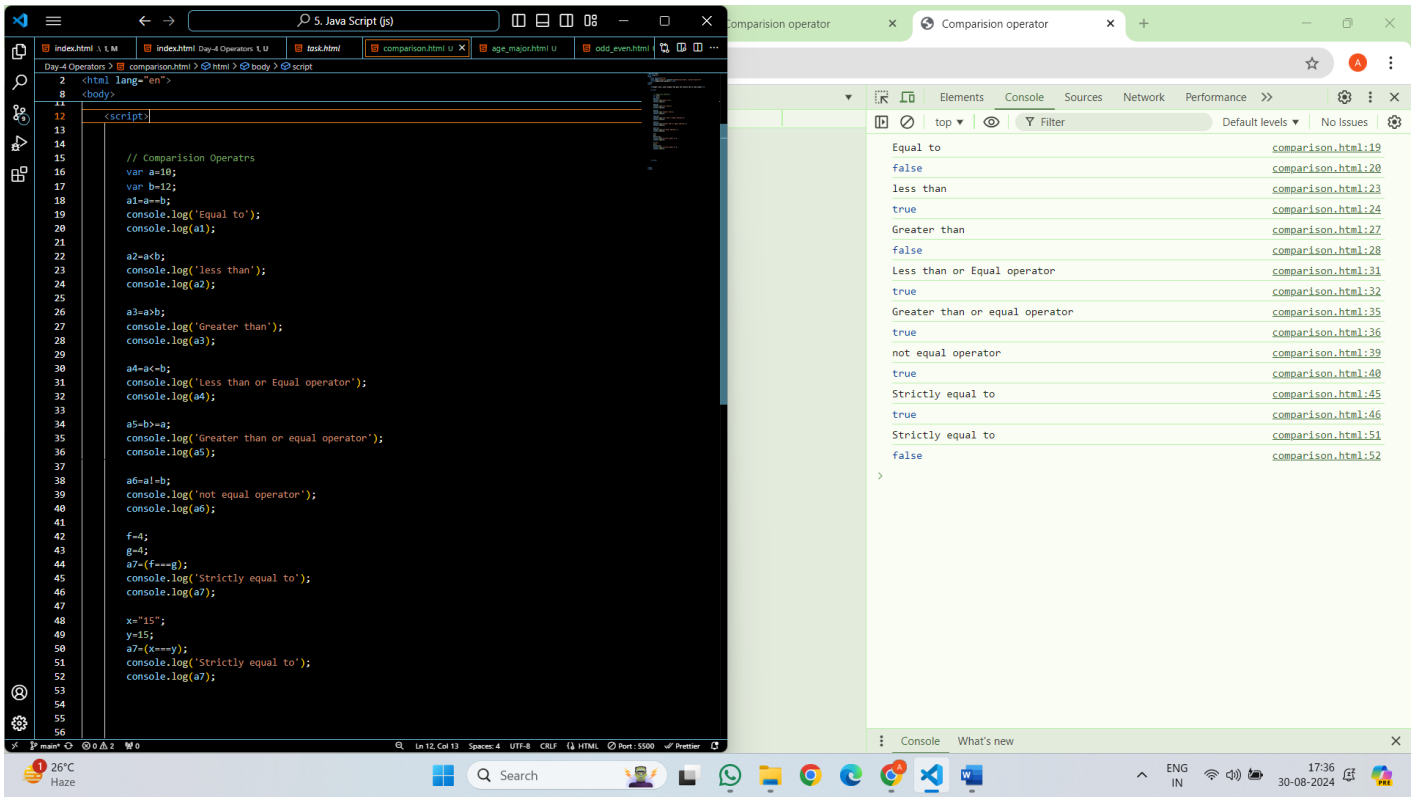
Example:



Comparision Operators

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

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

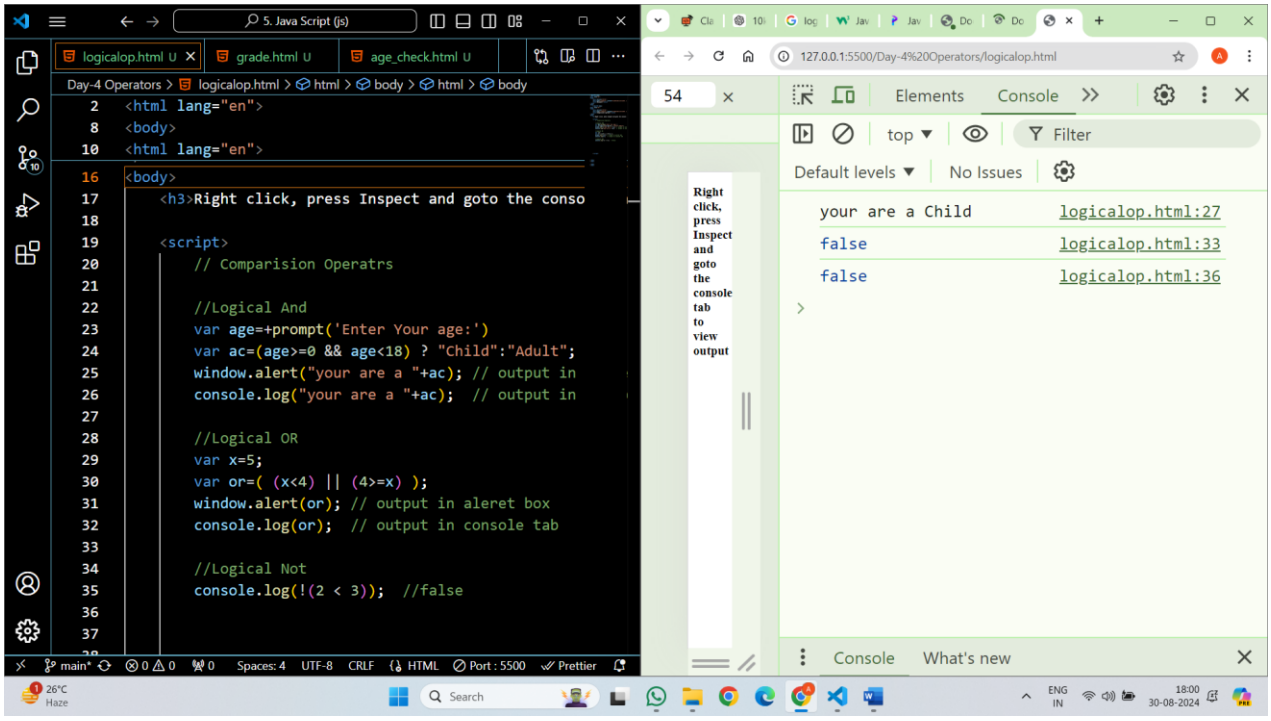


Logical Operator:

Logical operators return a boolean value by evaluating boolean expressions.

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

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



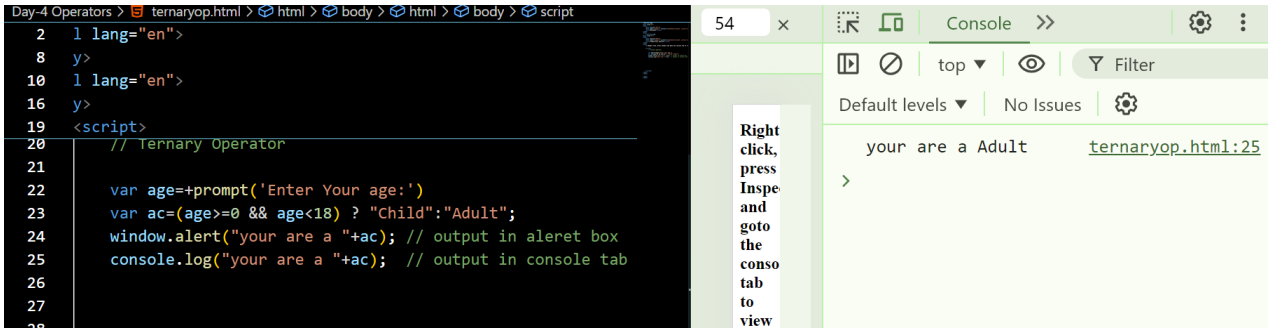
Ternary operator:

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

Syntax:

condition ? trueExpression : falseExpression

Example:



Nullish coalescing operator (??)

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

Example:

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

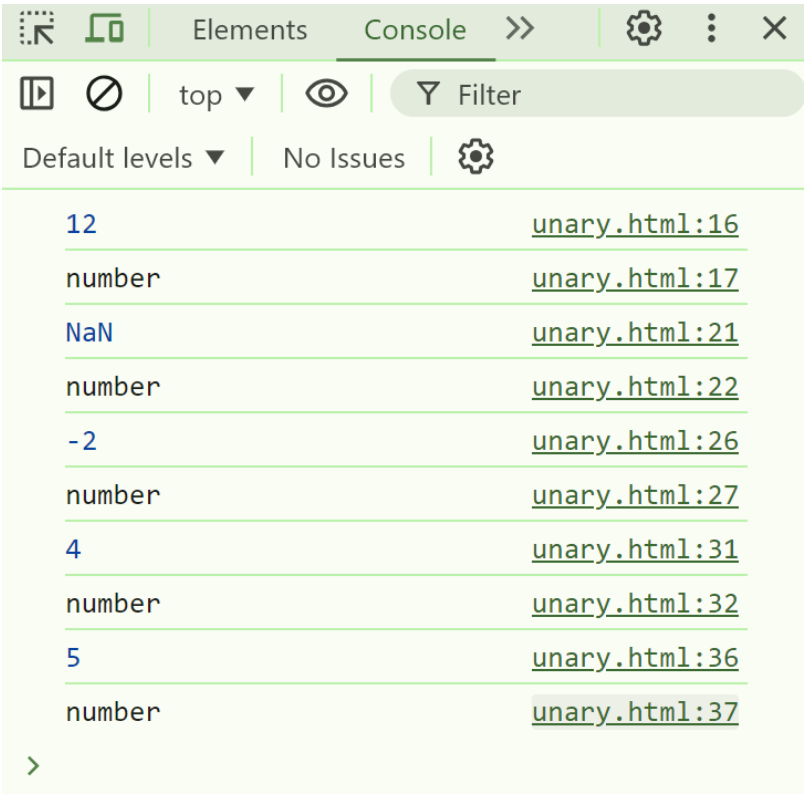
Unary Operator:

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

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

Example:

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

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

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

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

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



Type Coercion

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

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

There are two types of type conversion in JavaScript:

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

Explicit Type Conversion

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

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

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

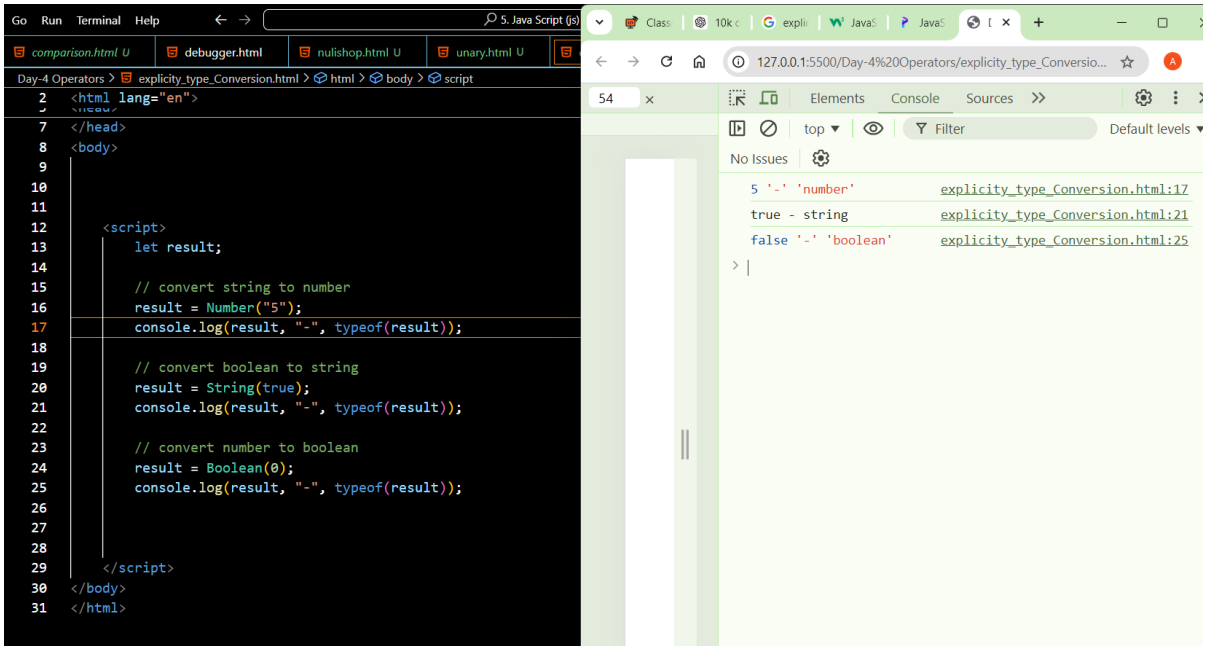
2. **Number()**: Converts a value to a number.

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

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

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

Example:



How to take or get input from Users:

Var a= +prompt('Enter Your Data');

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

Falsy Values:

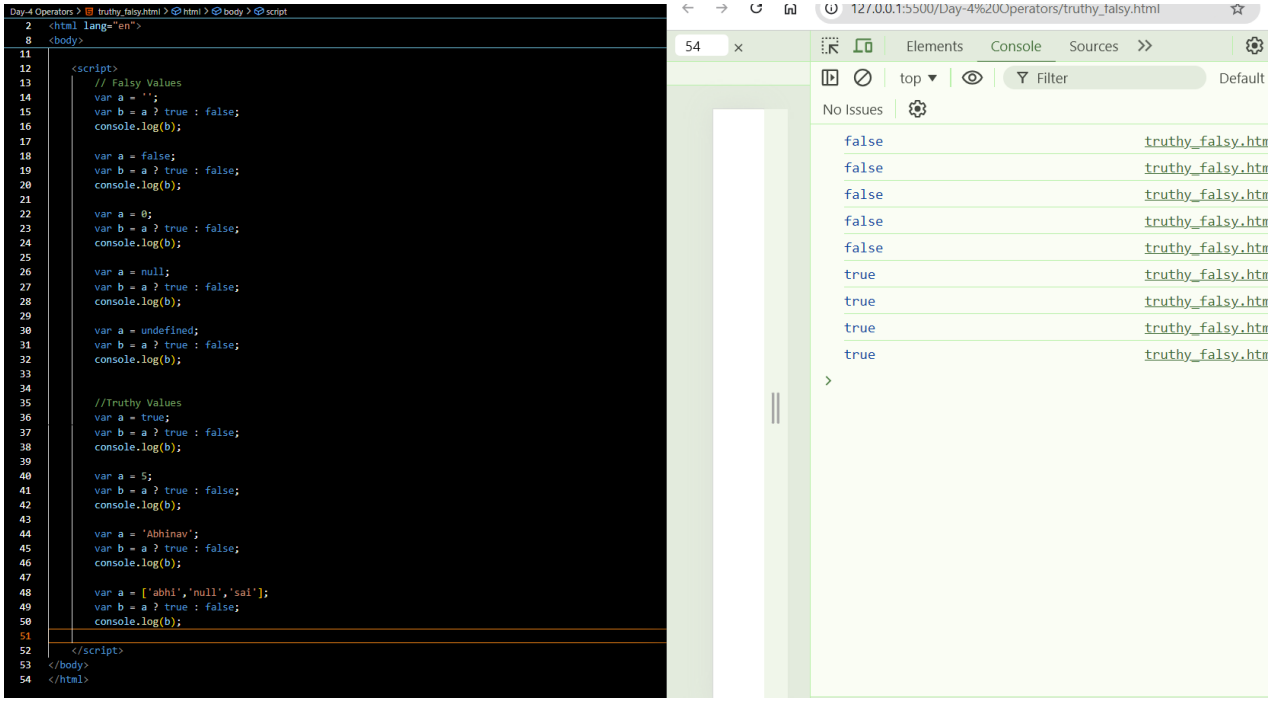
1. **false**: The boolean value false itself.
2. **0**: The number zero.
3. **""**: Empty string.
4. **null**: The absence of any value.
5. **undefined**: A variable that has not been assigned a value or a property that does not exist.
6. **NaN**: Not-a-Number.

Truthy Values:

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

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

Check Truthy, Falsy values using ternary operator:

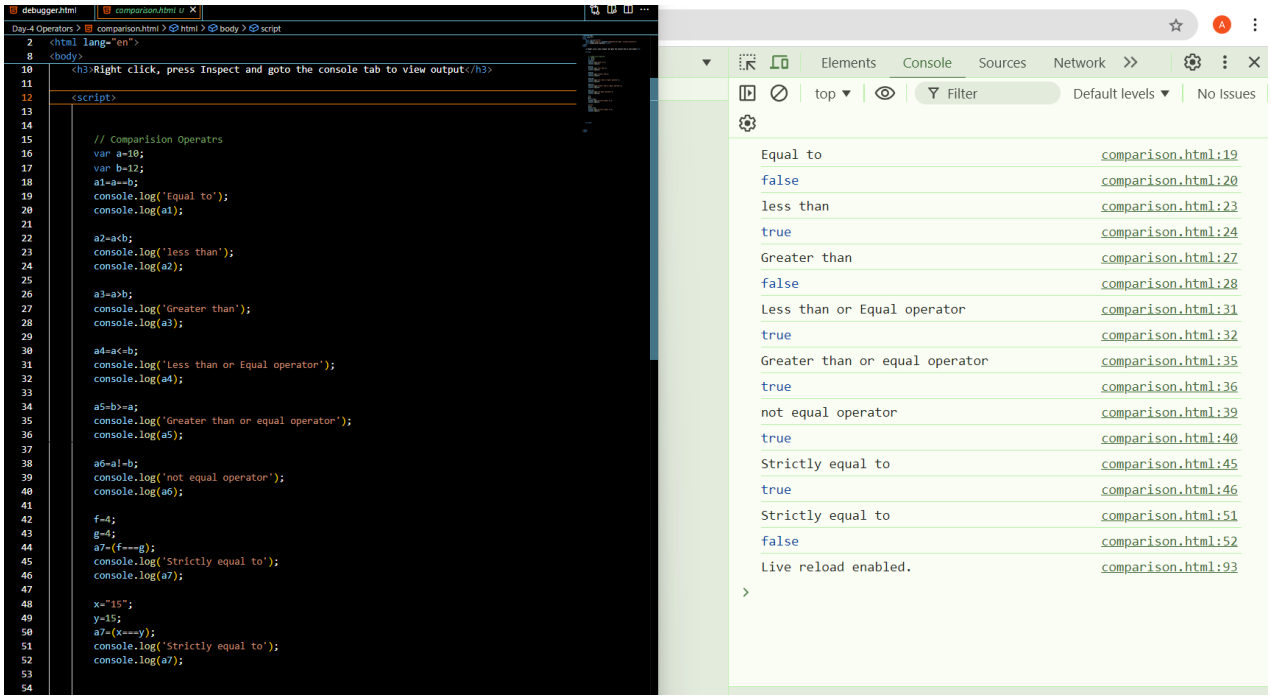


```
Day-4 Operators > truthy_falsy.html > html > body > script
<html lang="en">
<body>

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

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

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

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

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


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

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

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

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

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

```
127.0.0.1:5500/Day-4%20Operators/truthy_falsy.html
54
Elements  Console  Sources
top  Filter  Default
No Issues
false  truthy_falsy.htm
false  truthy_falsy.htm
false  truthy_falsy.htm
false  truthy_falsy.htm
false  truthy_falsy.htm
true  truthy_falsy.htm
true  truthy_falsy.htm
true  truthy_falsy.htm
true  truthy_falsy.htm
```

Tasks:

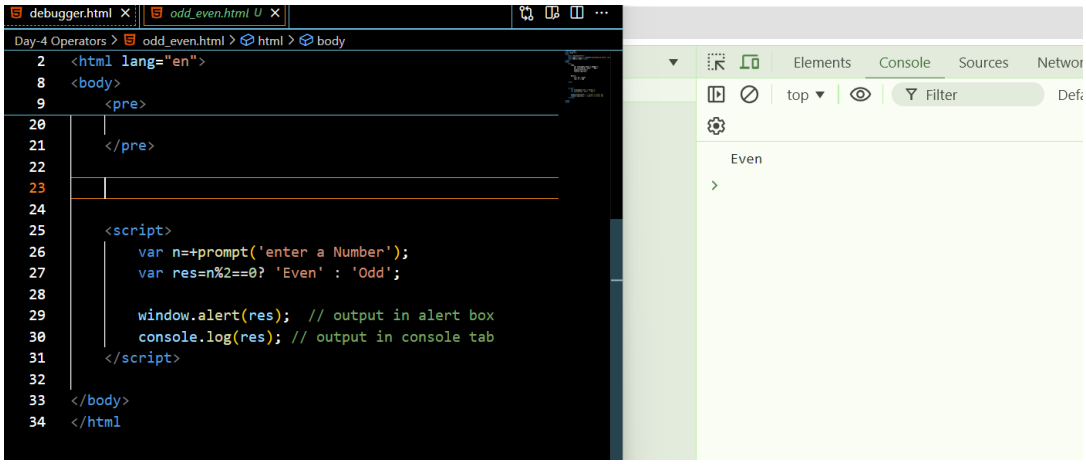
1. Write a JavaScript script that compares two variables using different comparison operators (==, ===, !=, !==, >, <, >=, <=) and prints the results.



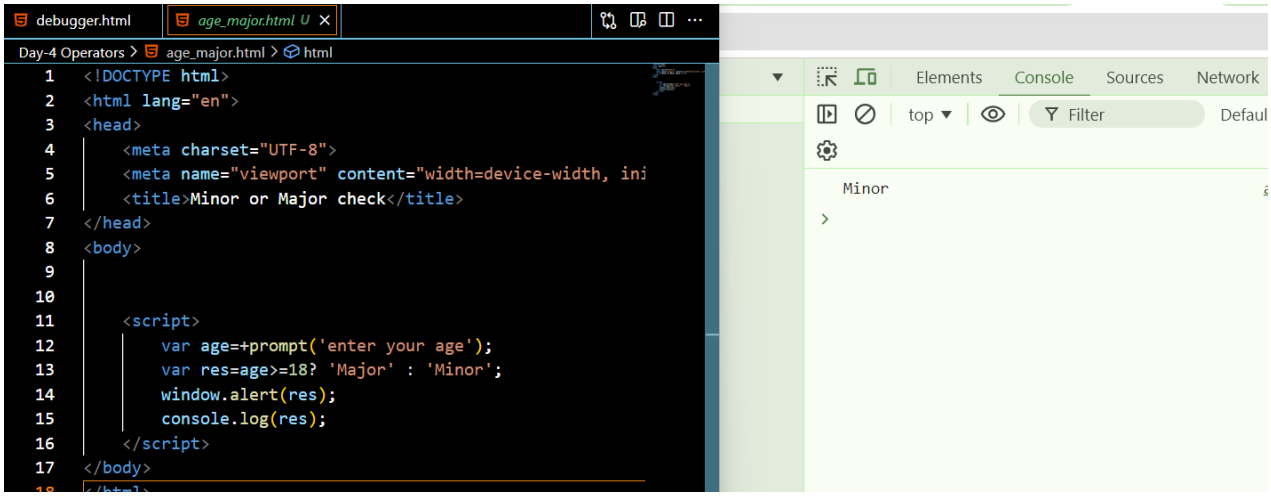
```
debugger.html  comparison.html U
Day-4 Operators > comparison.html > html > body > script
<html lang="en">
<body>
    <h3>Right click, press Inspect and goto the console tab to view output</h3>
    <script>


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

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

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

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

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

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

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

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

```
Elements  Console  Sources  Network
top  Filter  Default levels  No Issues
Equal to  comparison.html:19
false  comparison.html:20
less than  comparison.html:23
true  comparison.html:24
Greater than  comparison.html:27
false  comparison.html:28
Less than or Equal operator  comparison.html:31
true  comparison.html:32
Greater than or equal operator  comparison.html:35
true  comparison.html:36
not equal operator  comparison.html:39
true  comparison.html:40
Strictly equal to  comparison.html:45
true  comparison.html:46
Strictly equal to  comparison.html:51
false  comparison.html:52
Live reload enabled.  comparison.html:93
```

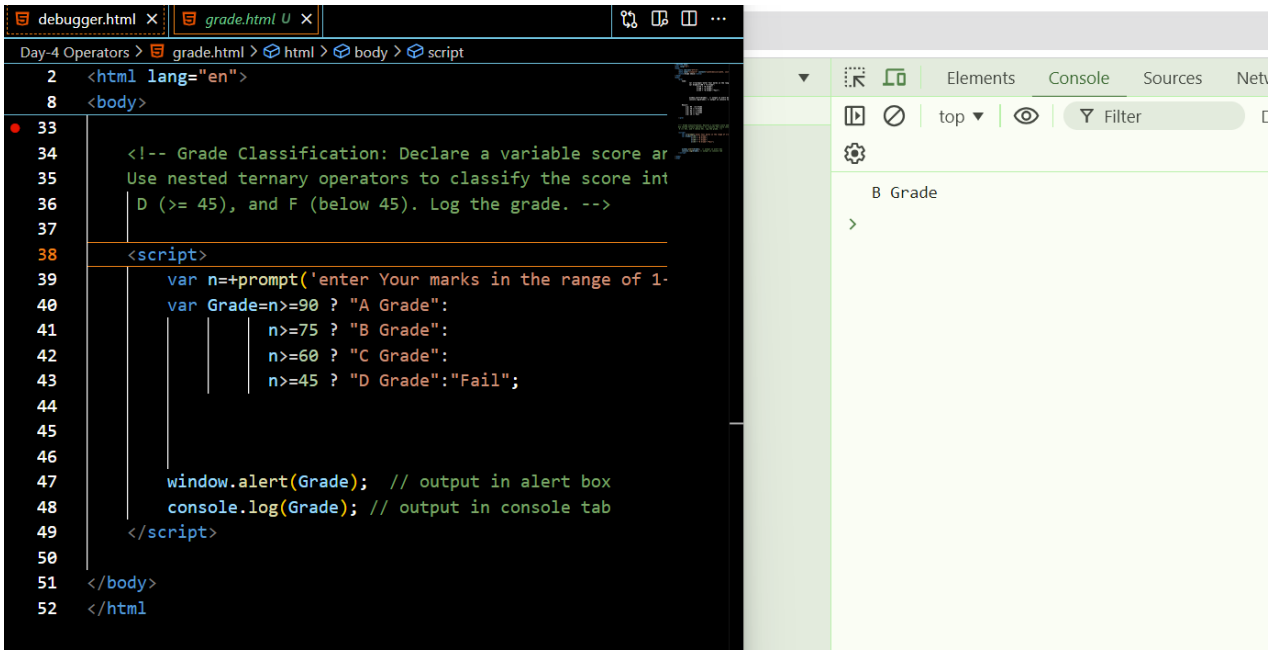
2. Write a JavaScript script that uses the ternary operator to determine if a number is even or odd.



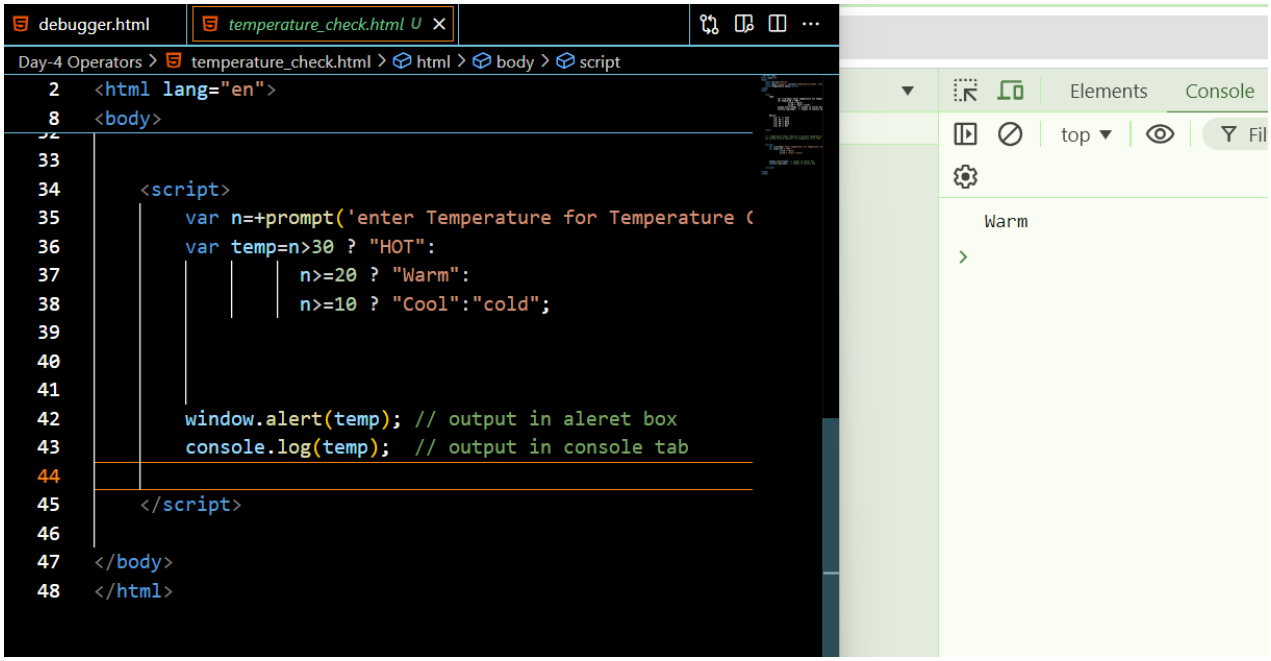
3. Expand the script to include a ternary operation that checks if a user is an adult (18+) or a minor.



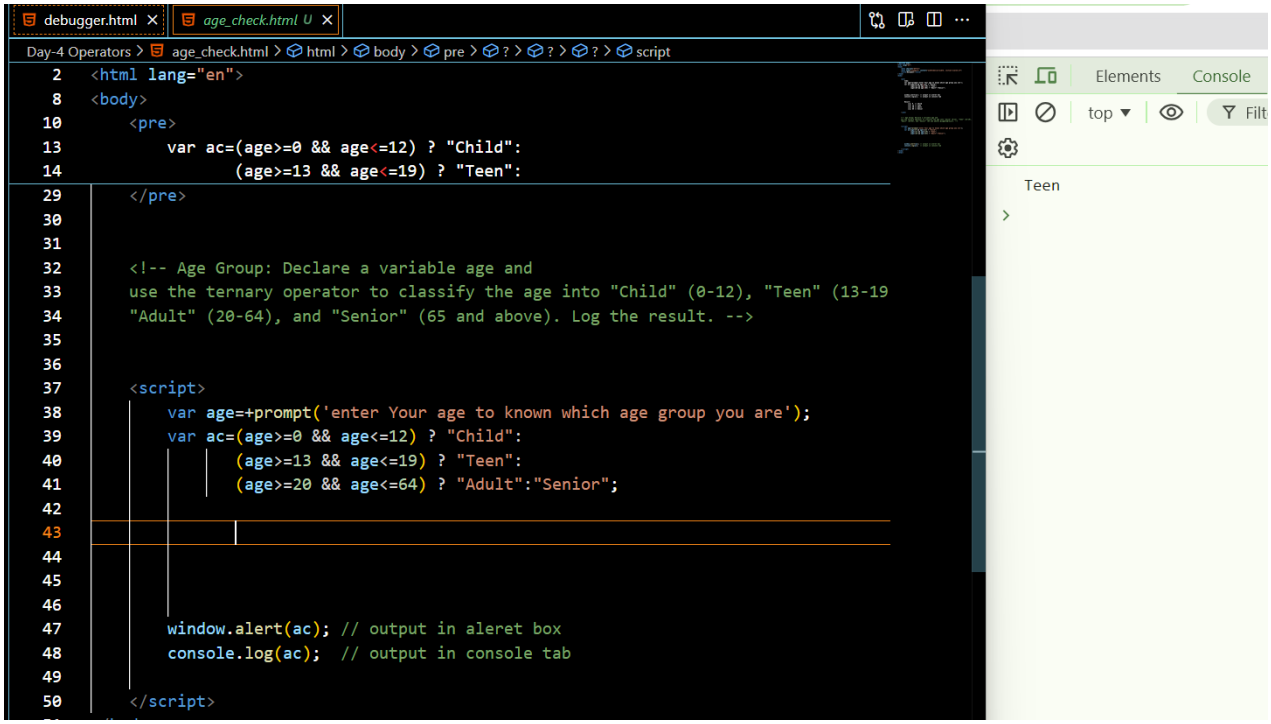
4. Grade Classification: Declare a variable score and set it to a value between 0 and 100. Use nested ternary operators to classify the score into grades: A (>= 90), B (>= 75), C (>= 60), D (>= 45), and F (below 45). Log the grade.



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

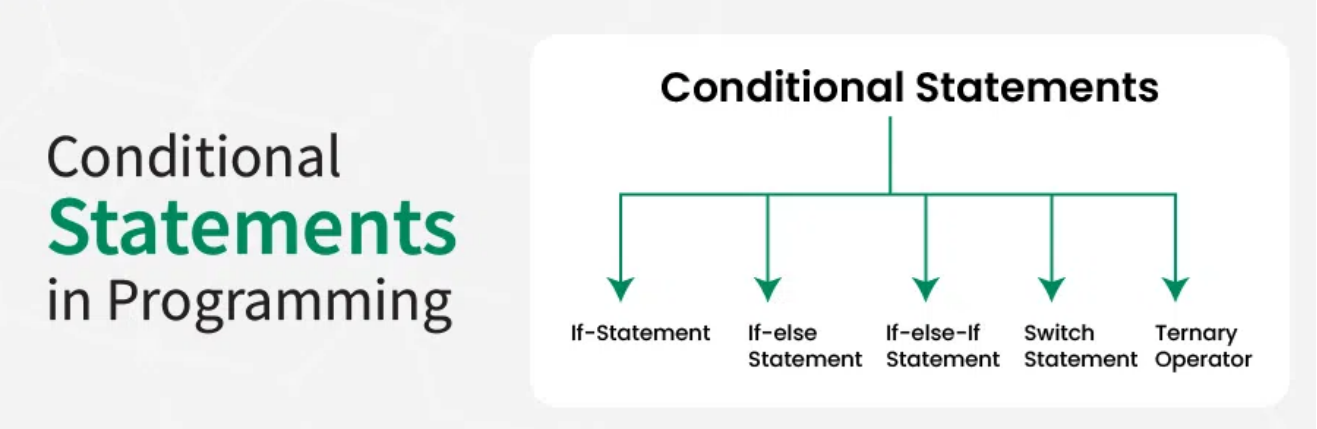


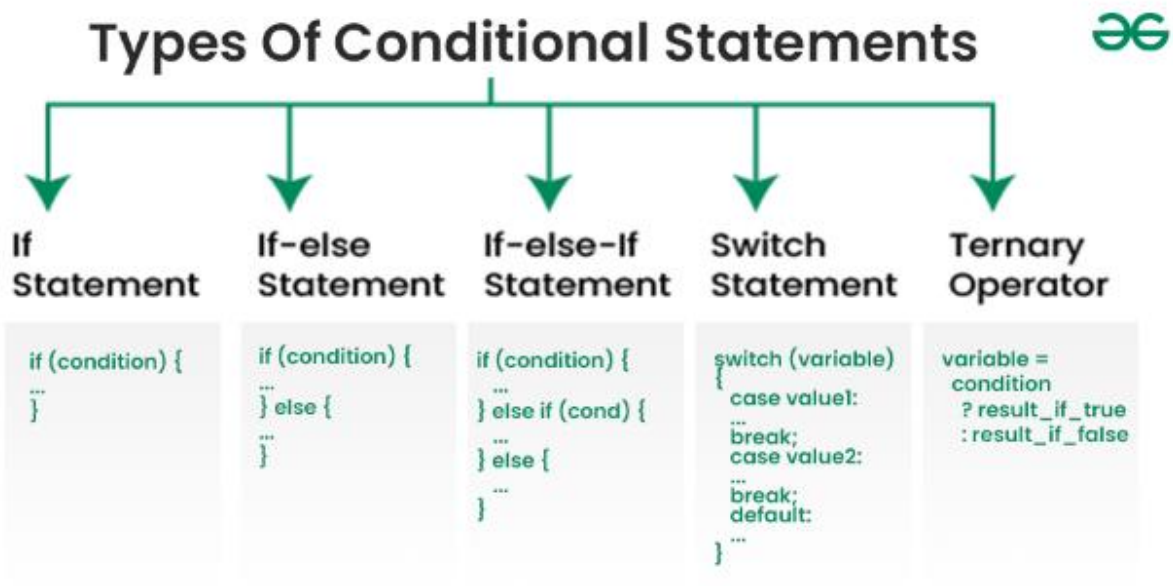
6. Age Group: Declare a variable age and use the ternary operator to classify the age into "Child" (0-12), "Teen" (13-19), "Adult" (20-64), and "Senior" (65 and above). Log the result.



Conditional Statements

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





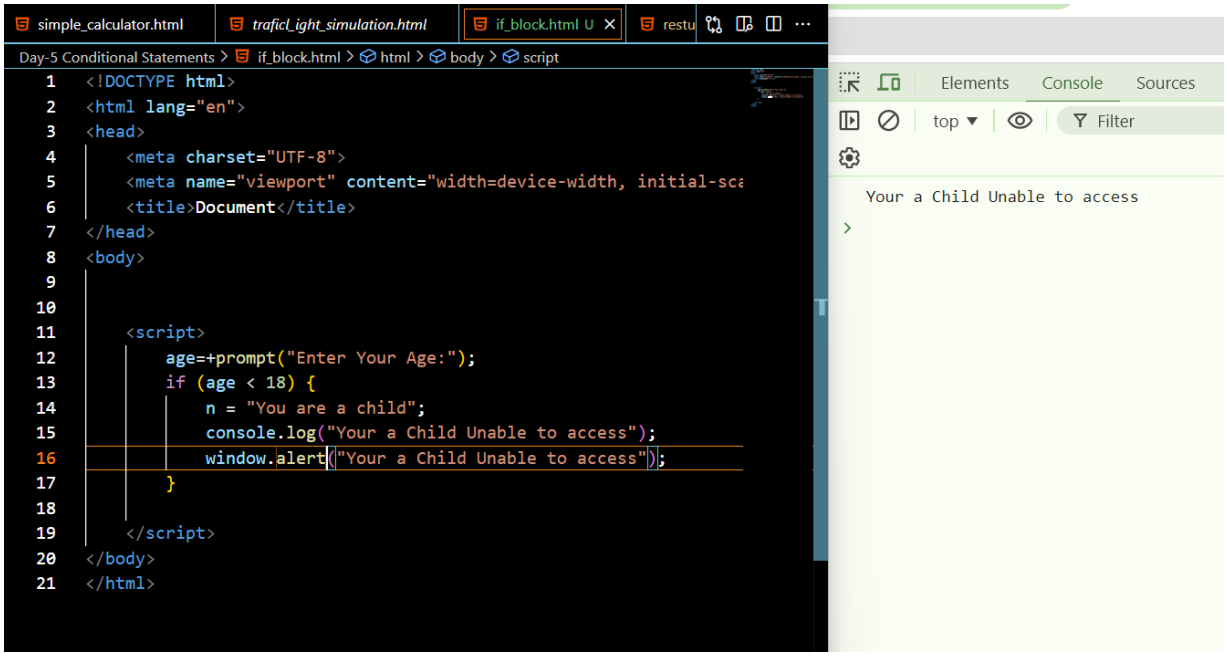
1. if Statement:

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

Syntax:

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

Example:



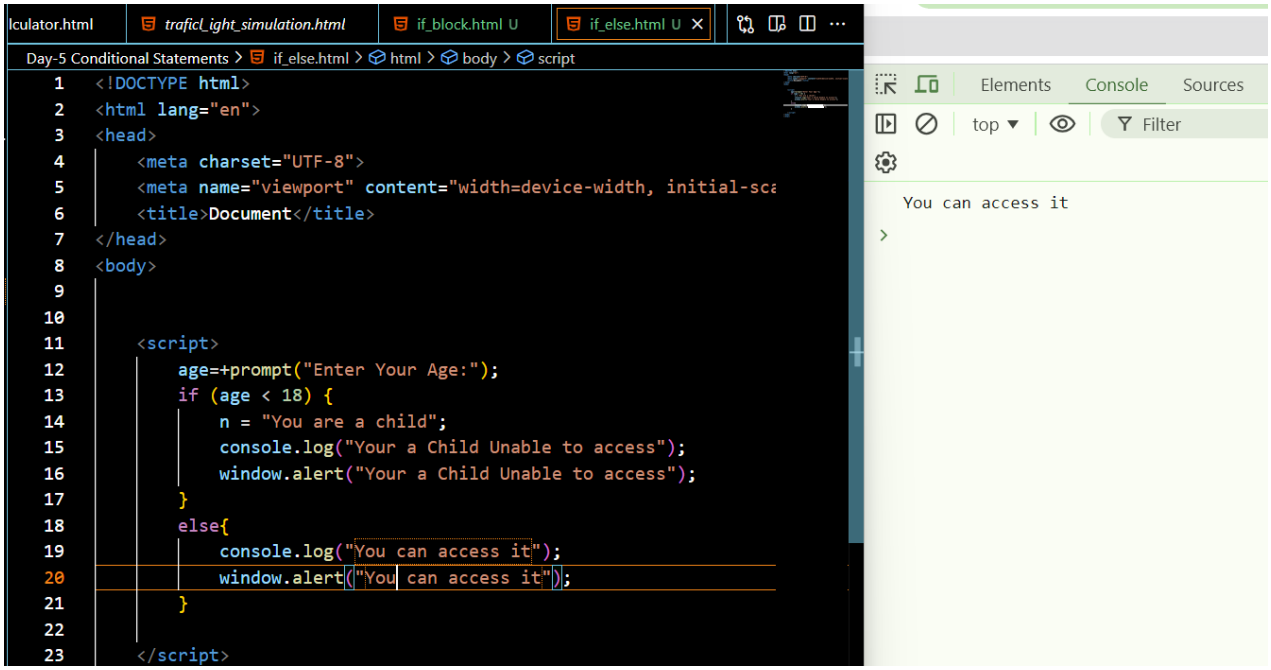
2. if...else Statement:

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

Syntax:

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

Example:



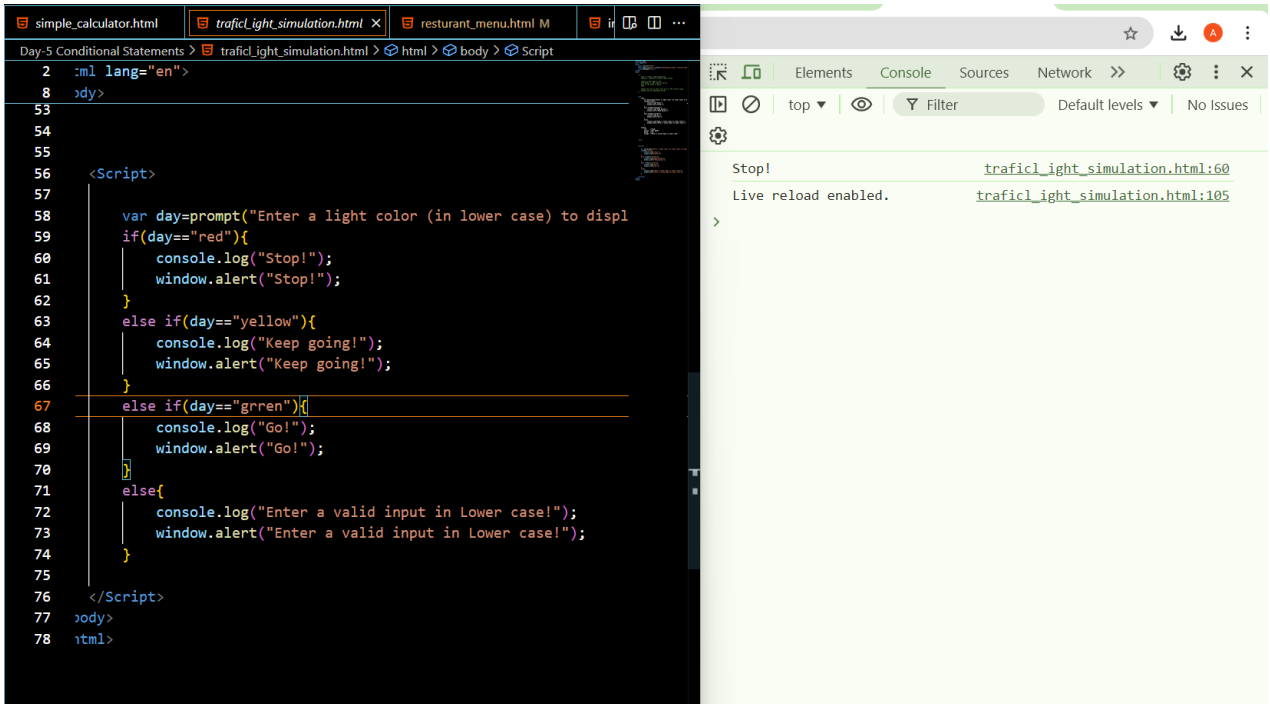
3)if...else if...else Statement:

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

Syntax:

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

Example:



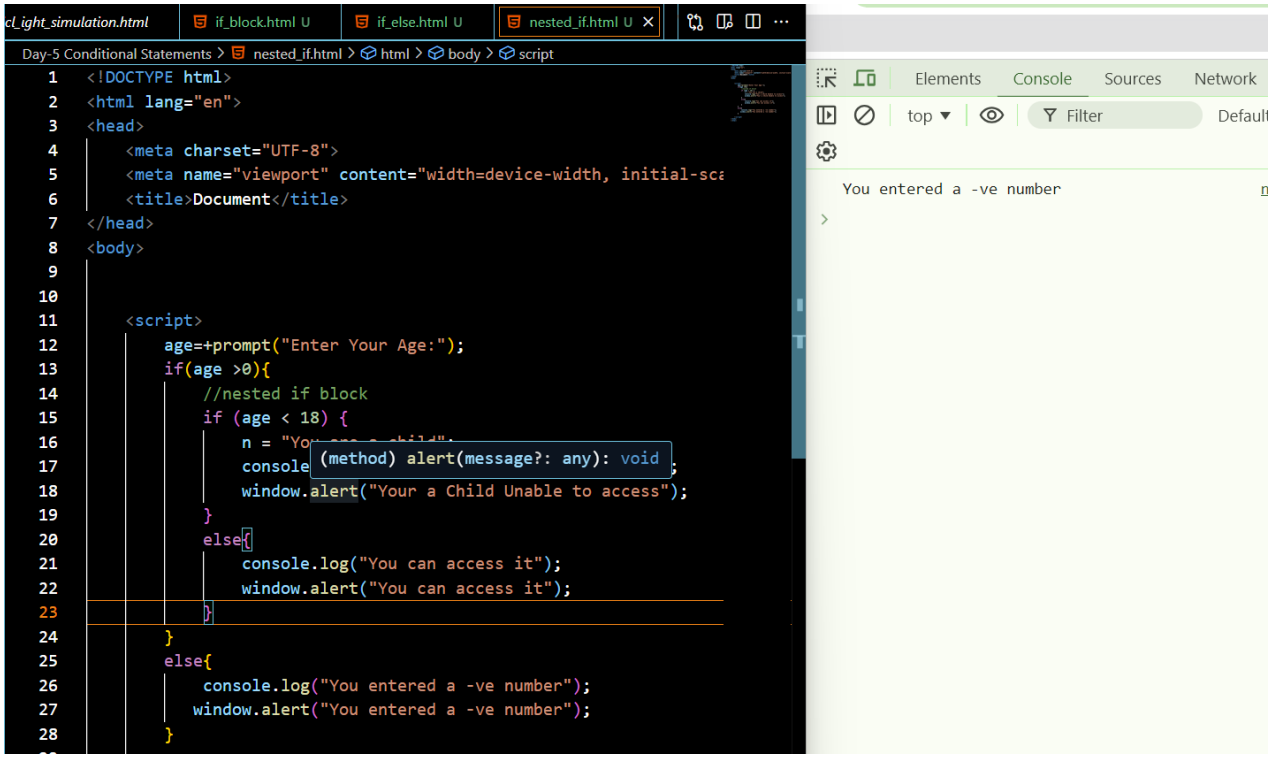
4)Nested if:

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

Syntax

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

Example:



Switch statements

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

Key Points

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

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

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

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

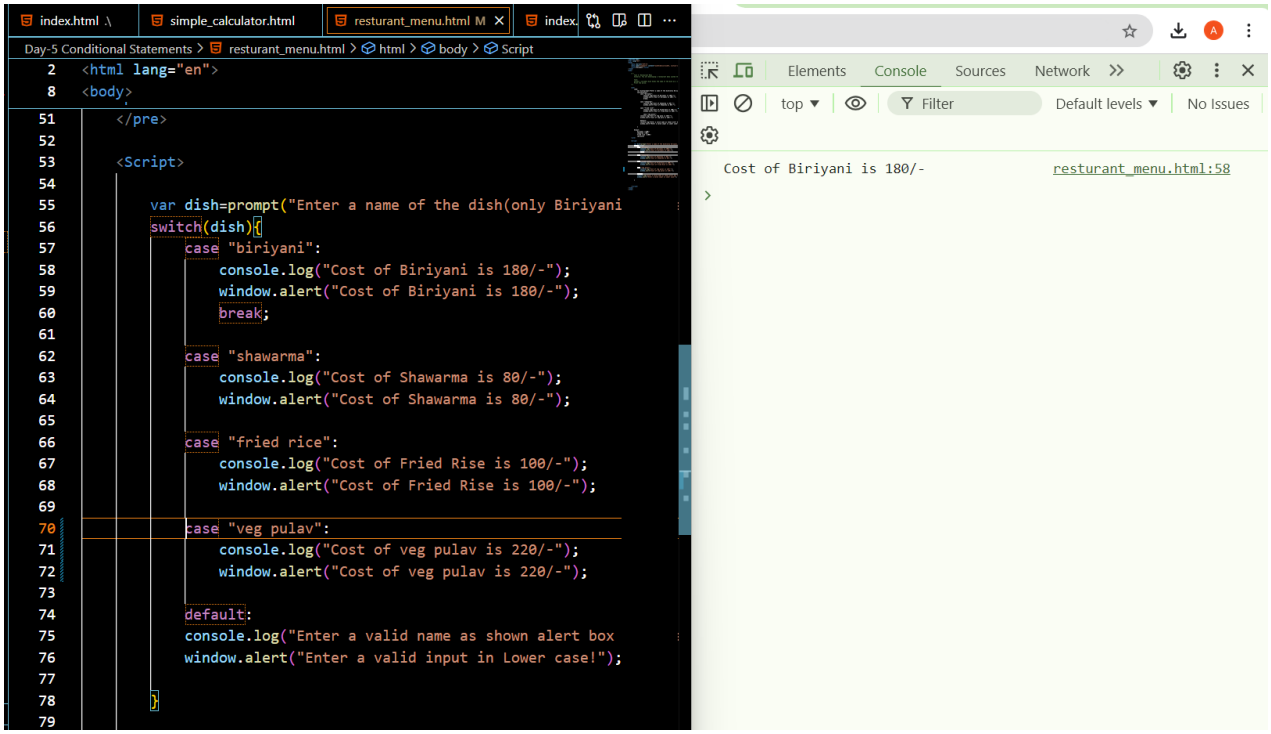
5. **Default Case**: The **default** clause is optional and executes if no matching **case** is found. It acts like the **else** in an if-else structure.

Syntax:

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

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

Example:



TASKS

Task 1: Day of the Week Message

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

“Start your week strong!” for Monday.

“Keep going!” for Tuesday.

“Halfway there!” for Wednesday.

“Almost the weekend!” for Thursday.

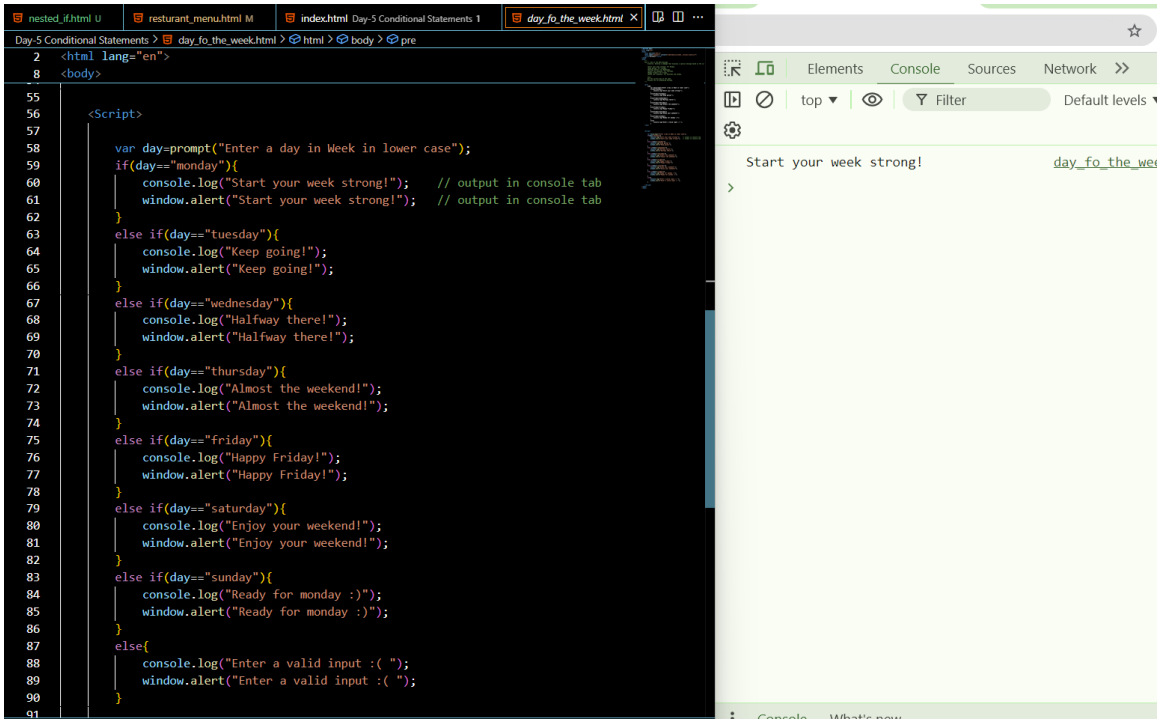
“Happy Friday!” for Friday.

“Enjoy your weekend!” for Saturday and Sunday.

Task:

Get the current day of the week.

Display the corresponding message.



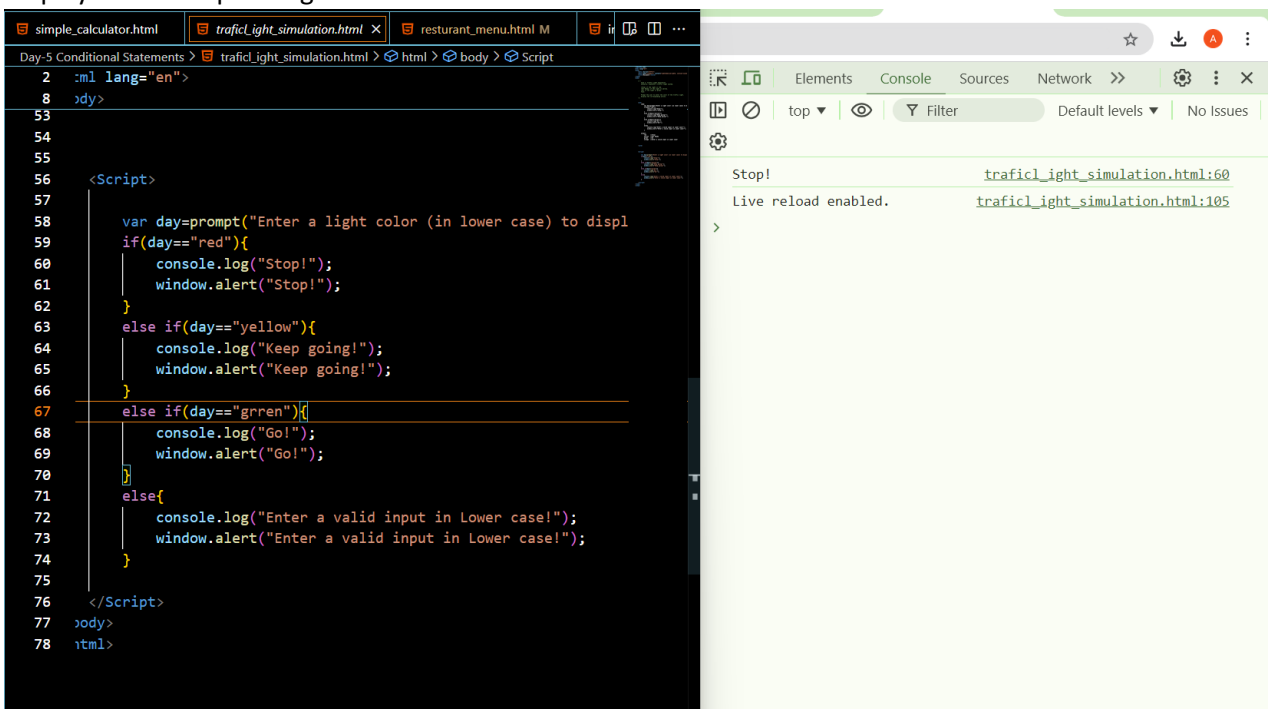
Task 2: Traffic Light Simulation

Scenario: Simulate a traffic light system.

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

Task:

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

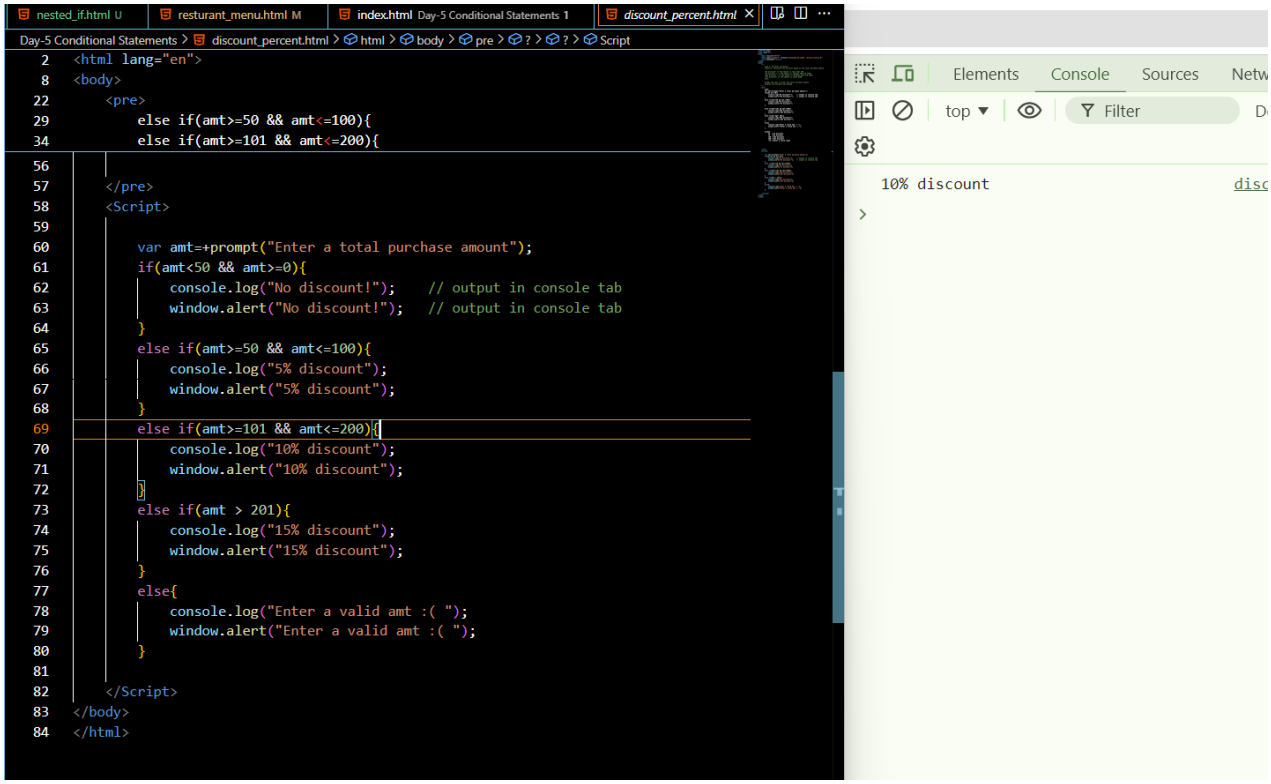


Task 3: Discount Calculator

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

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

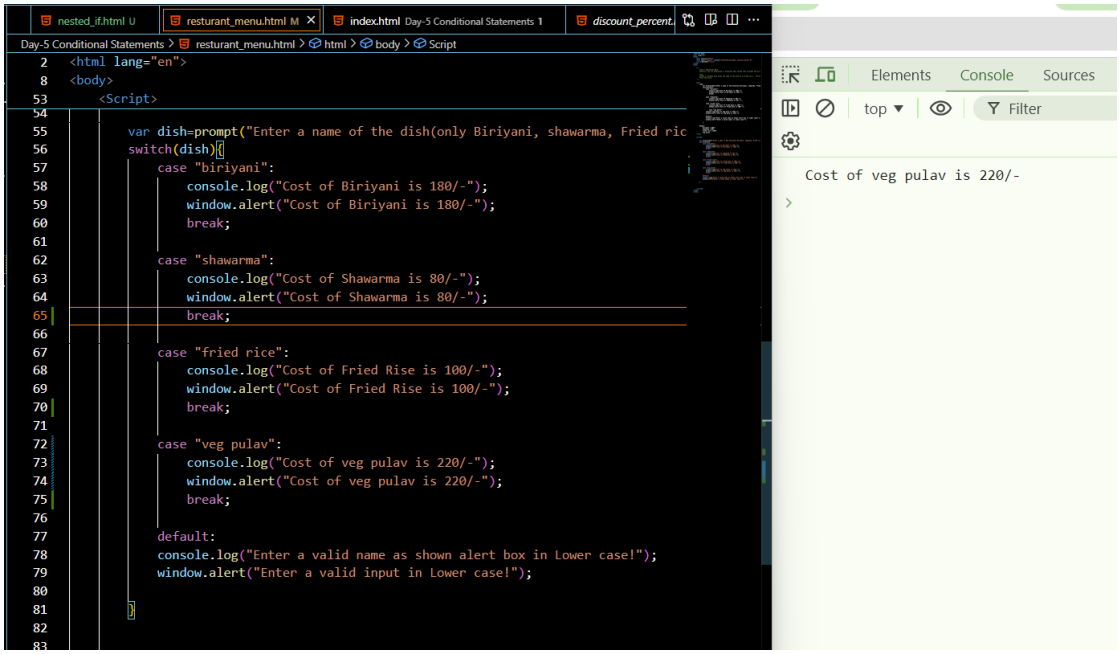
Task:
Prompt the user to enter the total purchase amount.
Display the discount percentage.



Task 4: Restaurant Menu

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

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



Task 5: Simple Calculator

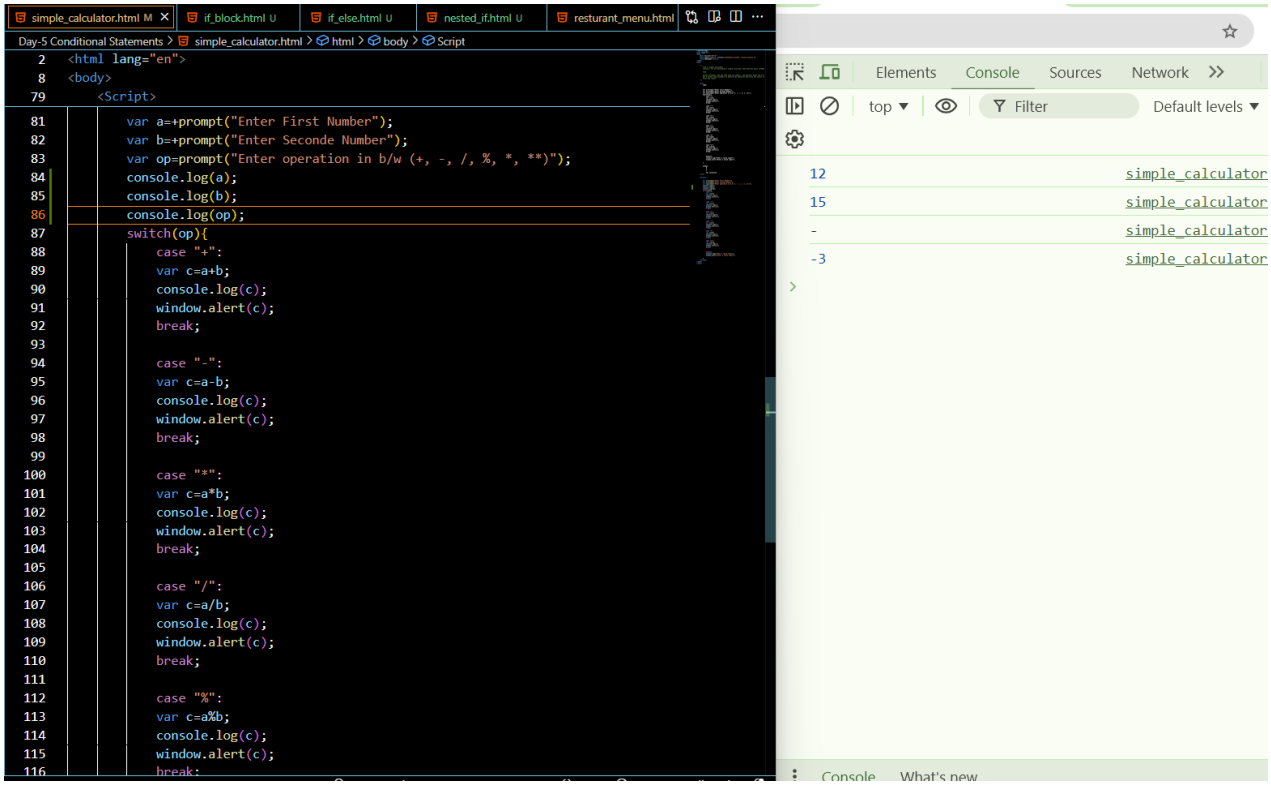
Scenario: You are developing a simple calculator that performs basic arithmetic operations.

Task:

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

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

Print the result.



LOOPS

Loops:

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

For loop:

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

Syntax:

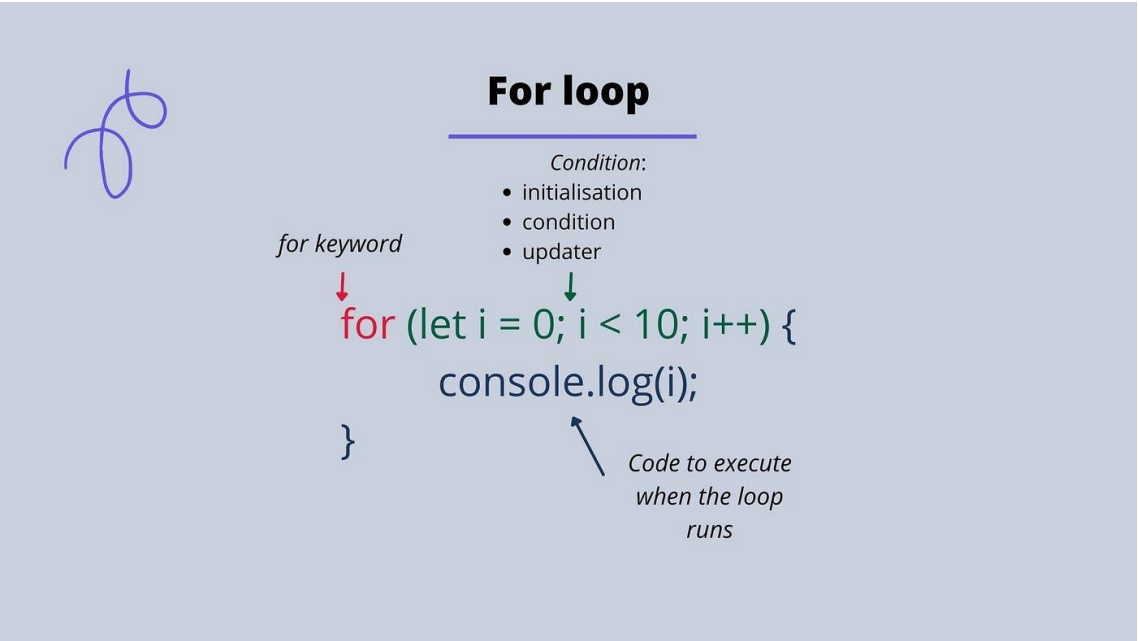
```
for (initialization; condition; increment)
{
   code to be executed
}
```

Example:

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

Output:

1
2
3
4
5

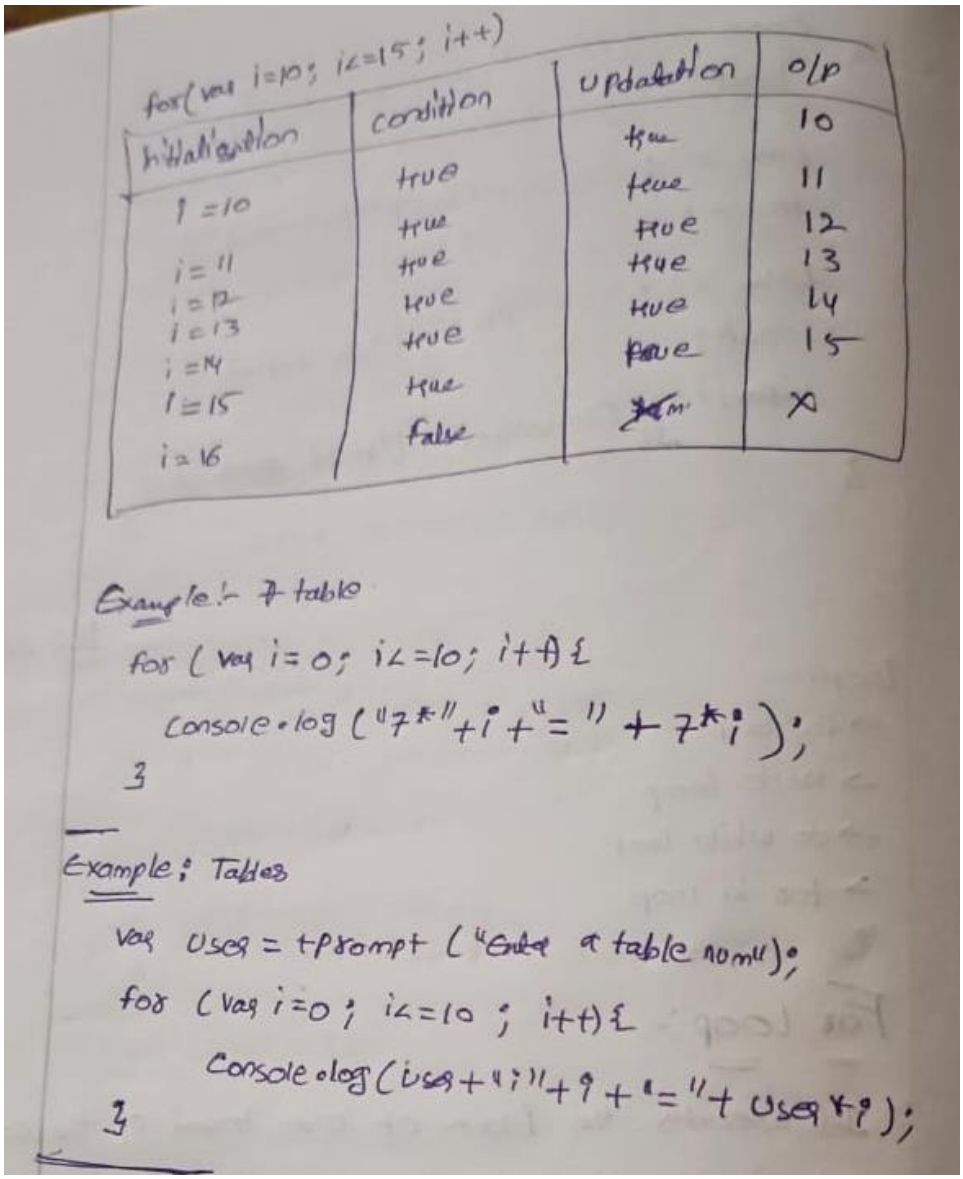


Nested For loop:

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

Example:

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

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


        }
    }
</script>
```



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

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

Example:- 7 table

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

Example: Tables

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

Example:- break the loop

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

Example:- continue for skip the current iteration.

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

Example:- sum of even numbers

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

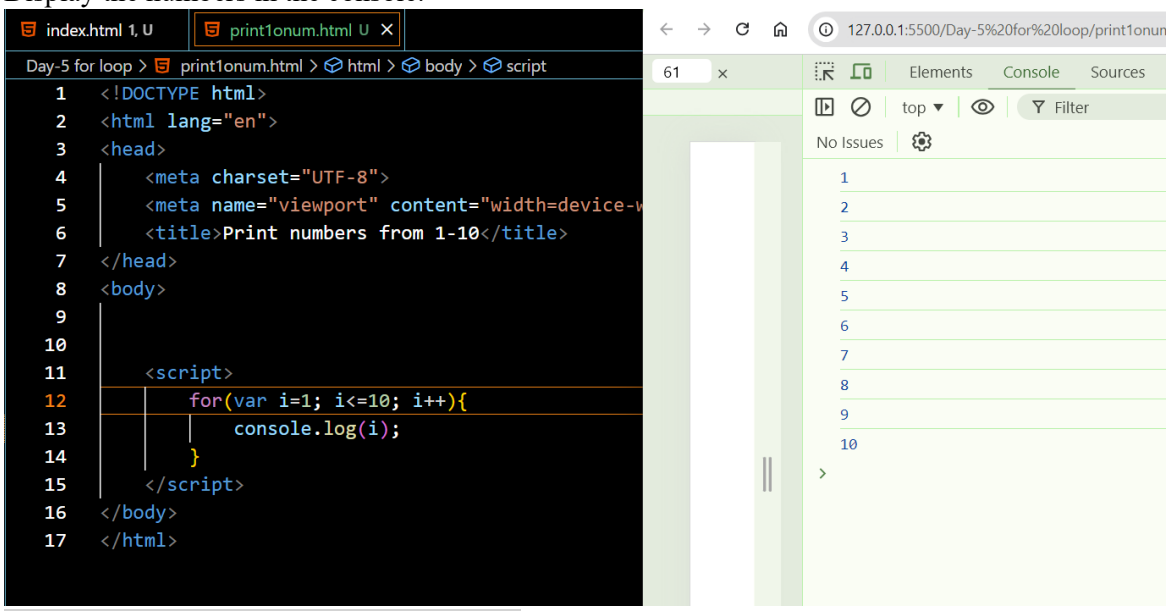
Tasks:

Task 1: Print Numbers from 1 to 10

Instructions:

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

Display the numbers in the console.



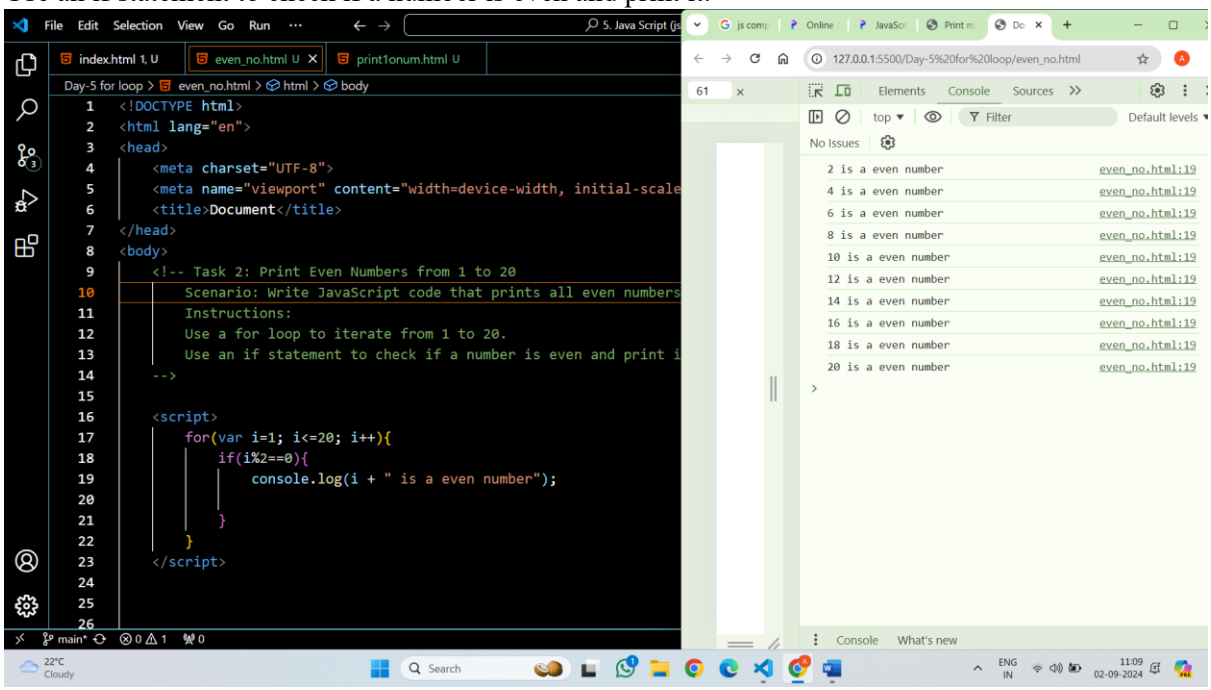
Task 2: Print Even Numbers from 1 to 20

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

Instructions:

Use a for loop to iterate from 1 to 20.

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



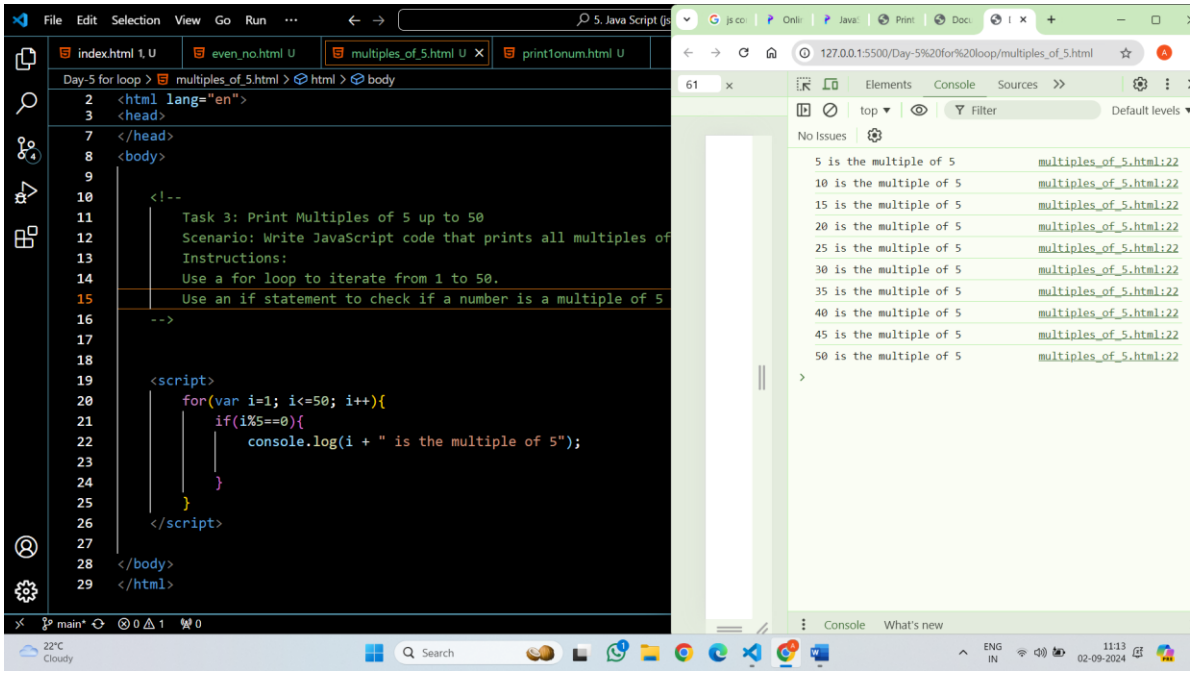
Task 3: Print Multiples of 5 up to 50

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

Instructions:

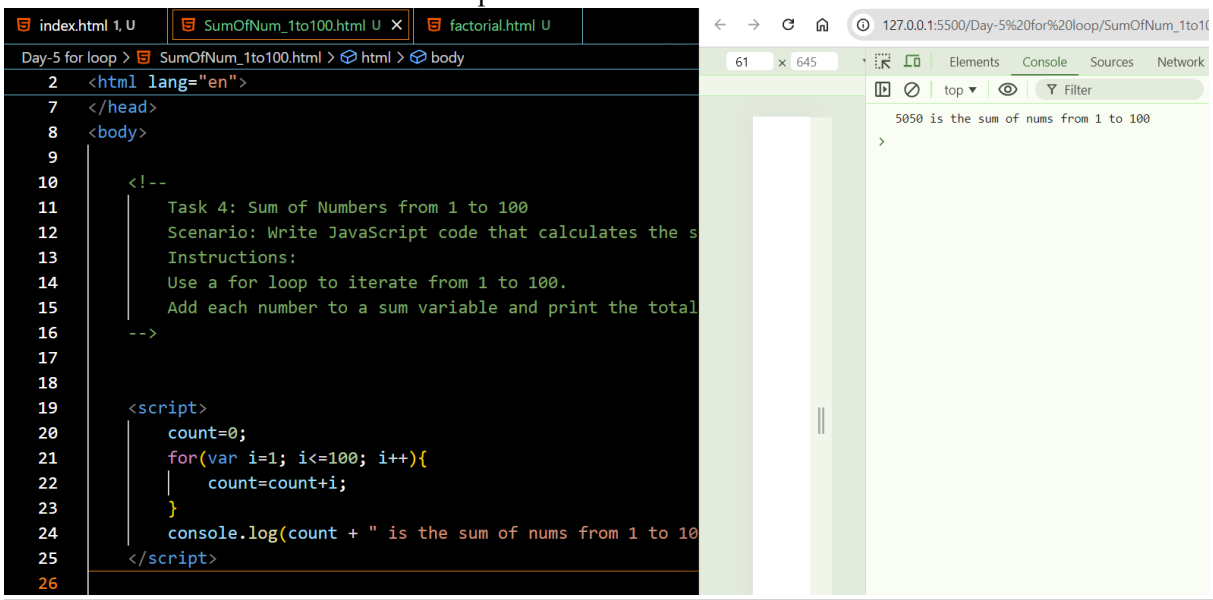
Use a for loop to iterate from 1 to 50.

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



Task 4: Sum of Numbers from 1 to 100

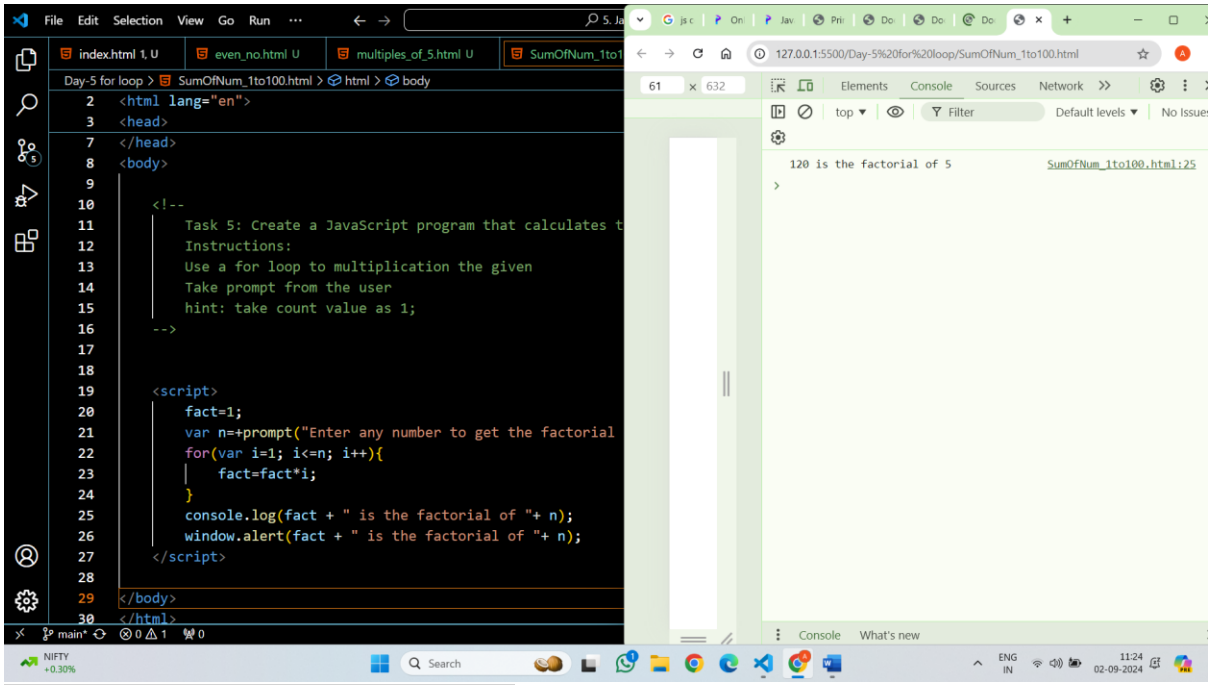
Scenario: Write JavaScript code that calculates the sum of numbers from 1 to 100.
Instructions:
Use a for loop to iterate from 1 to 100.
Add each number to a sum variable and print the total sum.



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

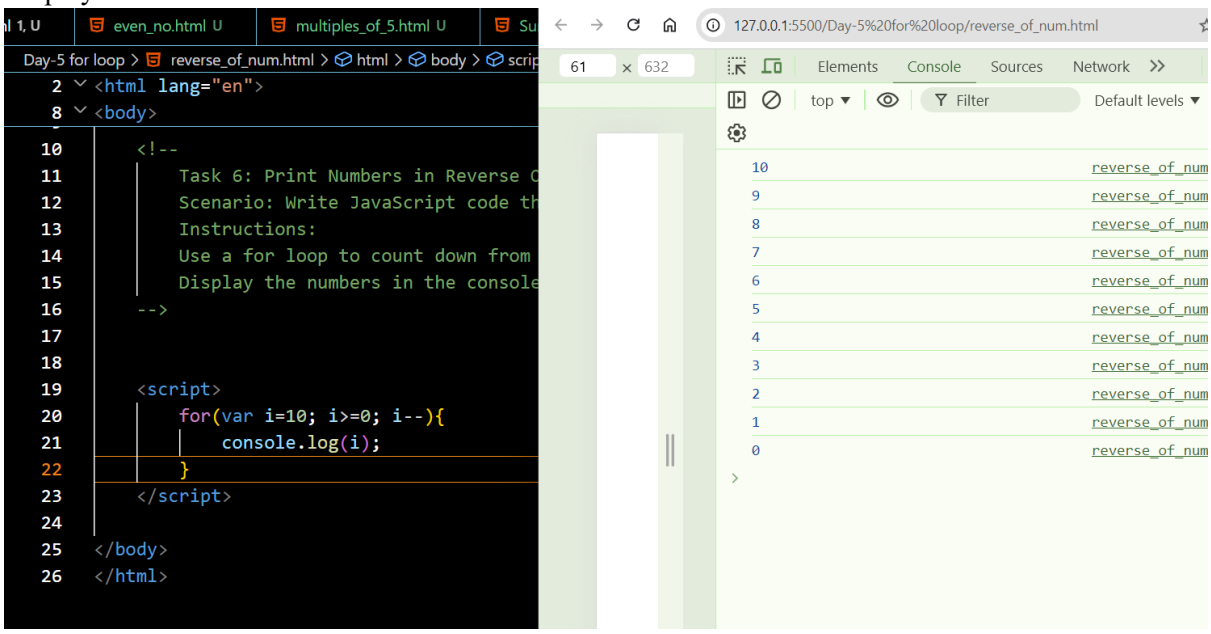
Instructions:

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



Task 6: Print Numbers in Reverse Order

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



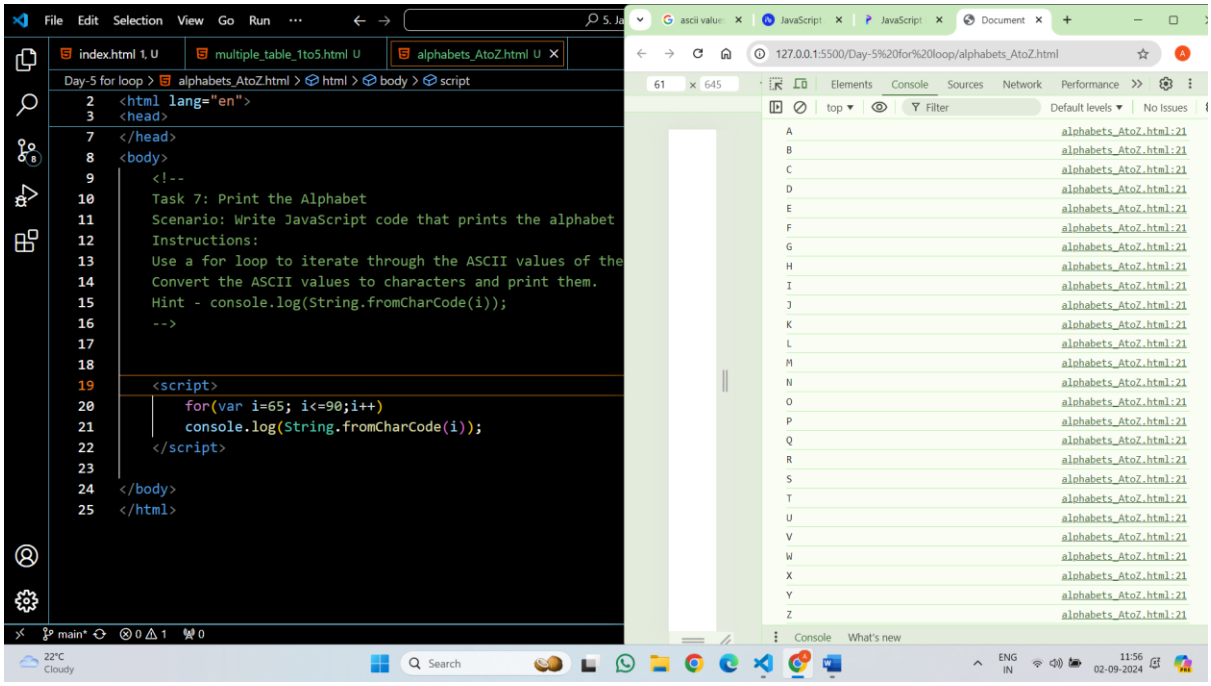
(optional)

Task 7: Print the Alphabet

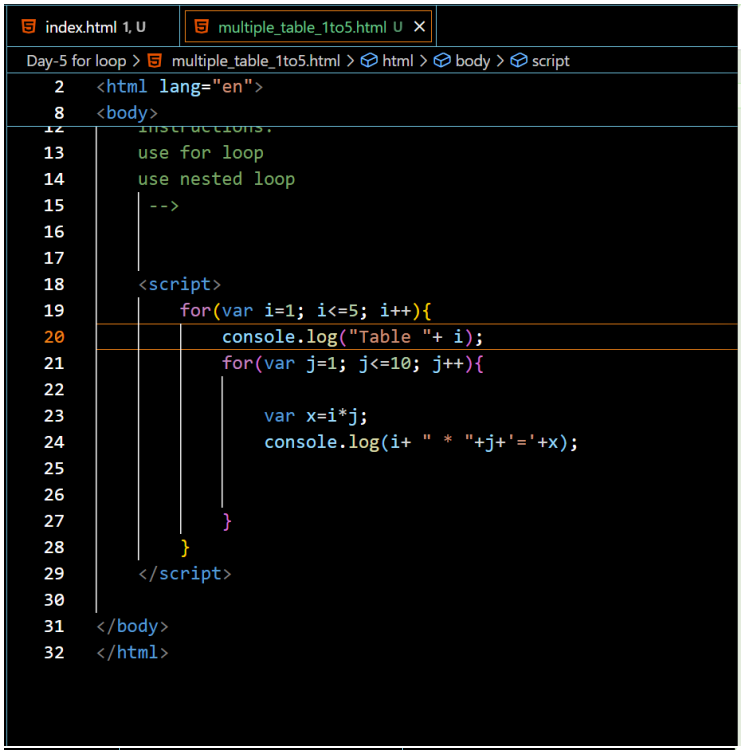
Scenario: Write JavaScript code that prints the alphabet from A to Z.
Instructions:
Use a for loop to iterate through the ASCII values of the letters A to Z.
Convert the ASCII values to characters and print them.
Hint - console.log(String.fromCharCode(i));

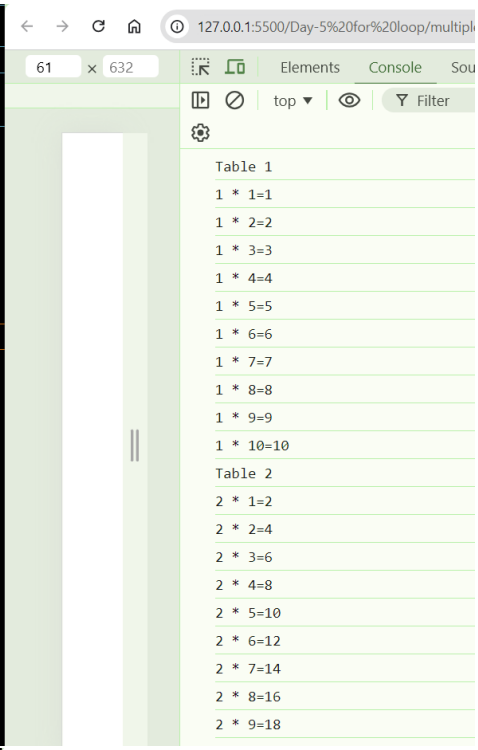
ASCII TABLE

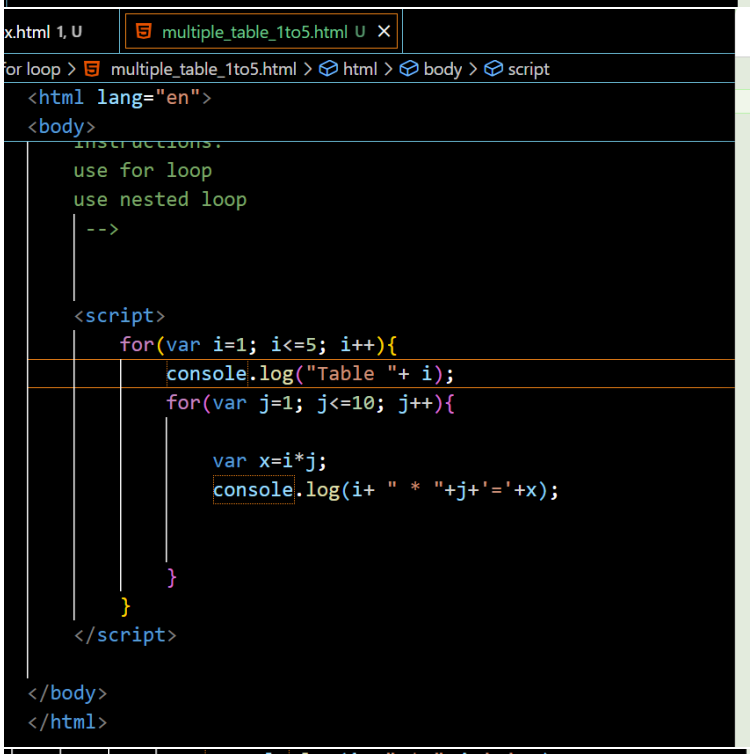
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]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

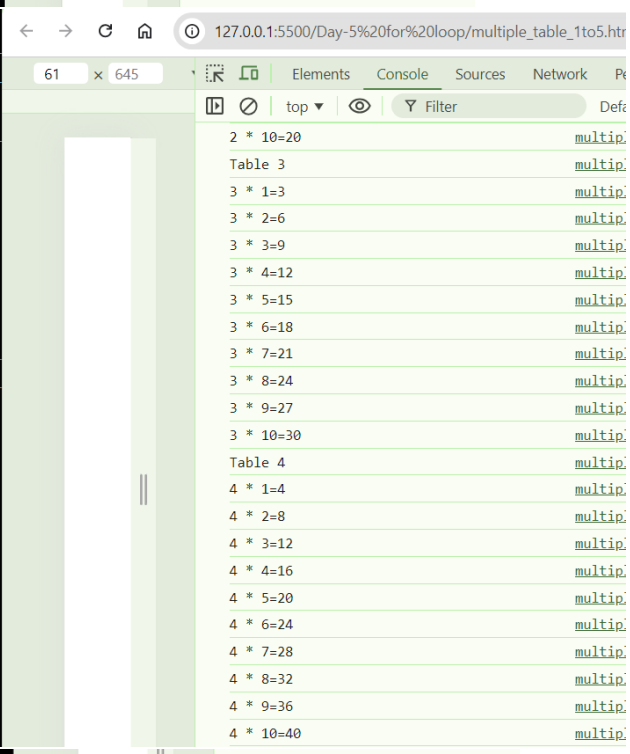


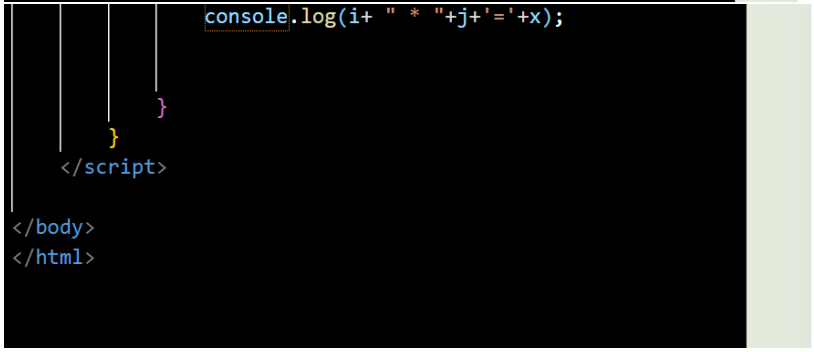
Task 8:Write a JavaScript script that uses nested loops to print a multiplication table for numbers 1 through 5.
Instructions:
use for loop
use nested loop


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


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


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


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


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

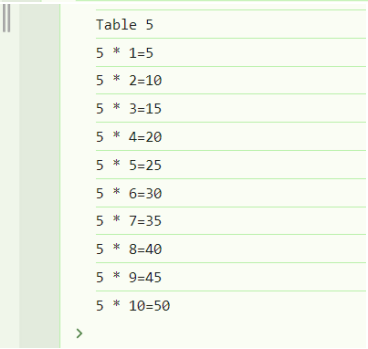


Table 5
5 * 1=5
5 * 2=10
5 * 3=15
5 * 4=20
5 * 5=25
5 * 6=30
5 * 7=35
5 * 8=40
5 * 9=45
5 * 10=50

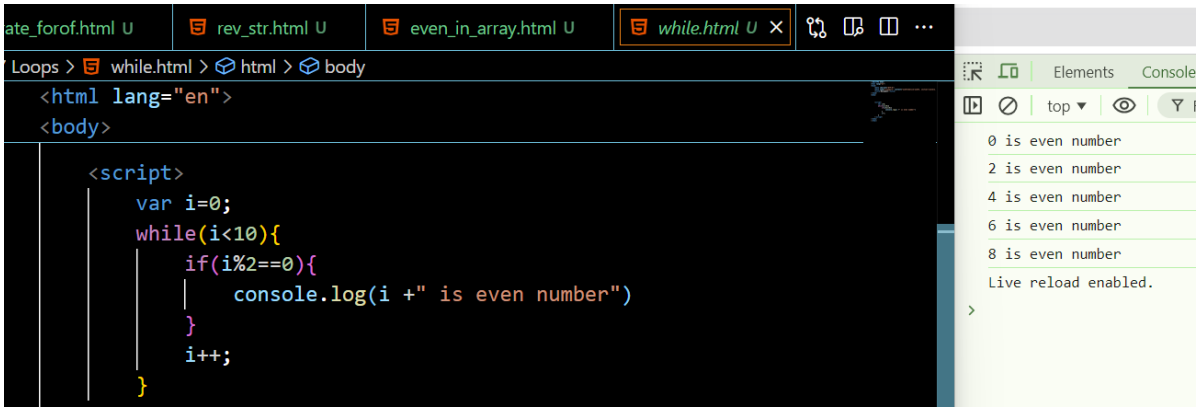
While Loop:

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:



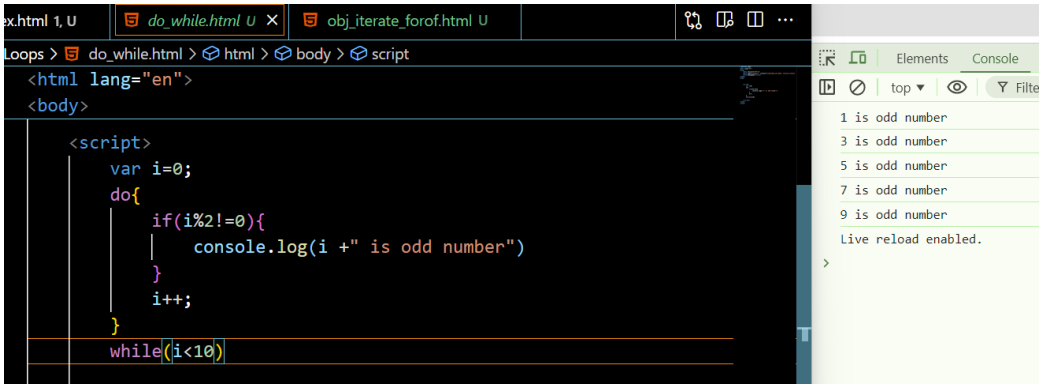
Do While:

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:



For in loop:

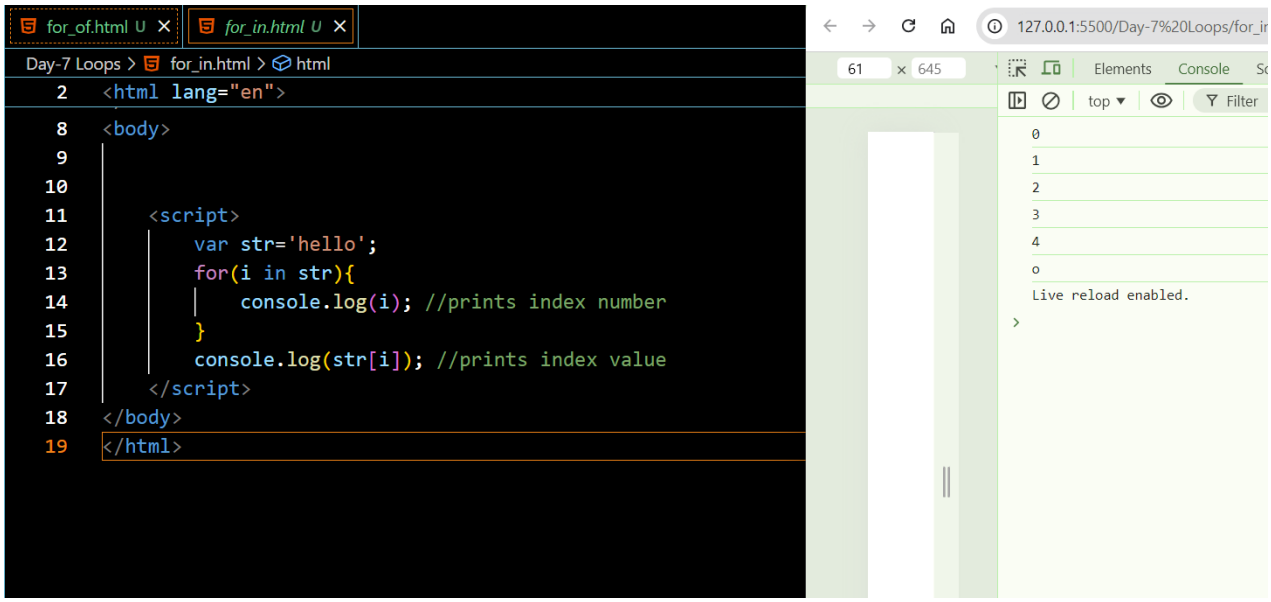
Used to iterate over the properties of an object,array, string. It iterates over enumerable properties of an object, in an arbitrary order.

We can access index and values.

Syntax:

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

Example:



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.

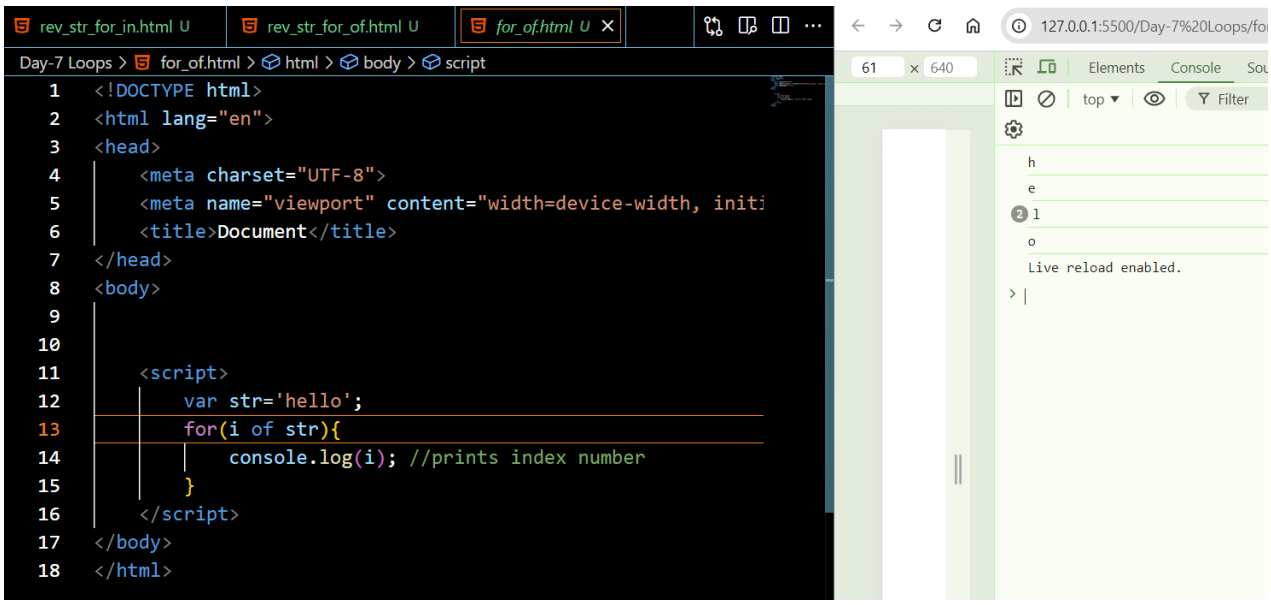
1) **Iterates over Values**:

- 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**:

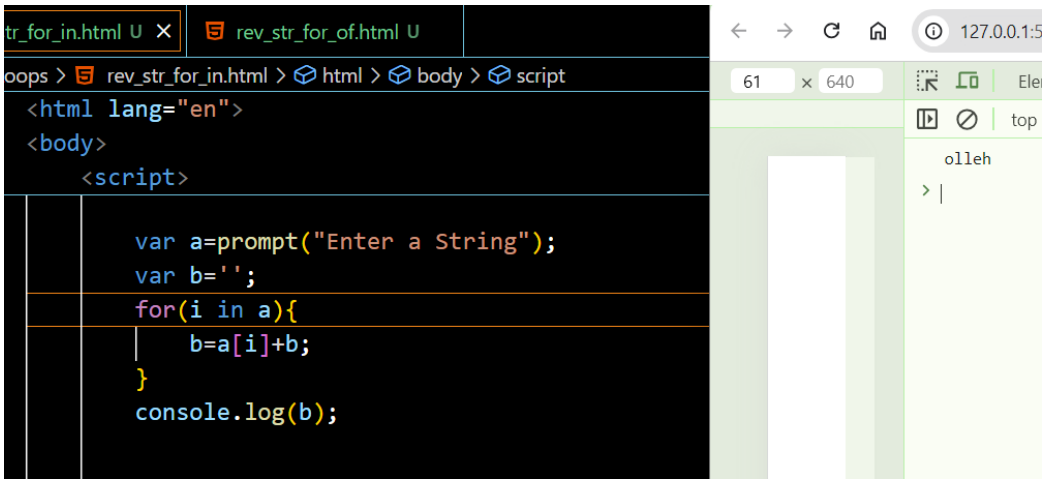
- Commonly used with arrays, strings, maps, sets, and other iterable objects.

Example:



TASK :

1. Reverse a string Input : hello output : olleh



```
<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);
```

```
olleh
```

2. Find the even numbers in the array - [23,54,67,64.45,95,98].

```
<!-- 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>
```

```
54
64
98
Live reload enabled.
```

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

```
<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>
```

```
name : Alice
age : 30
profession : Engineer
```

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: {}

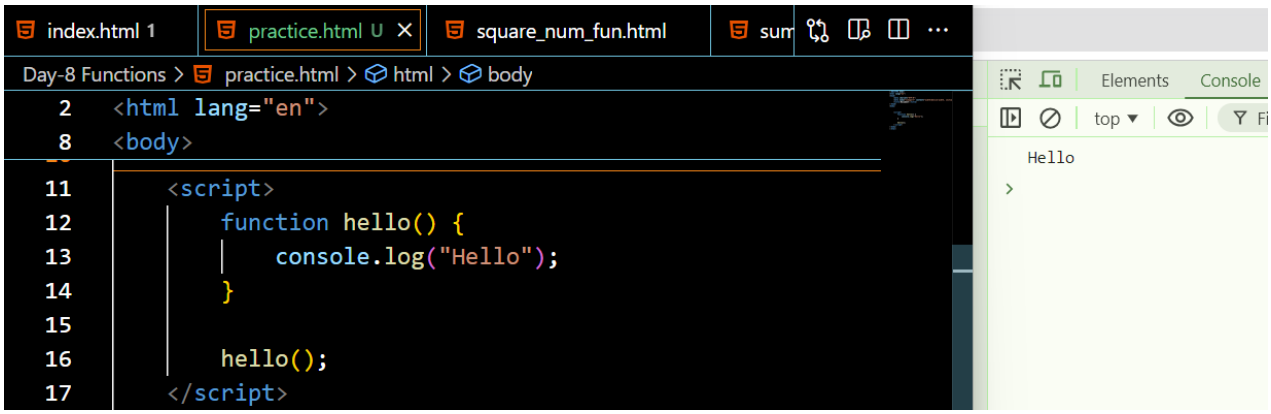
Function parameters are listed inside the parentheses () in the function definition. Function arguments are the values received by the function when it is invoked. Inside the function, the arguments (the parameters) behave as local variables.

Named functions can be hoisted

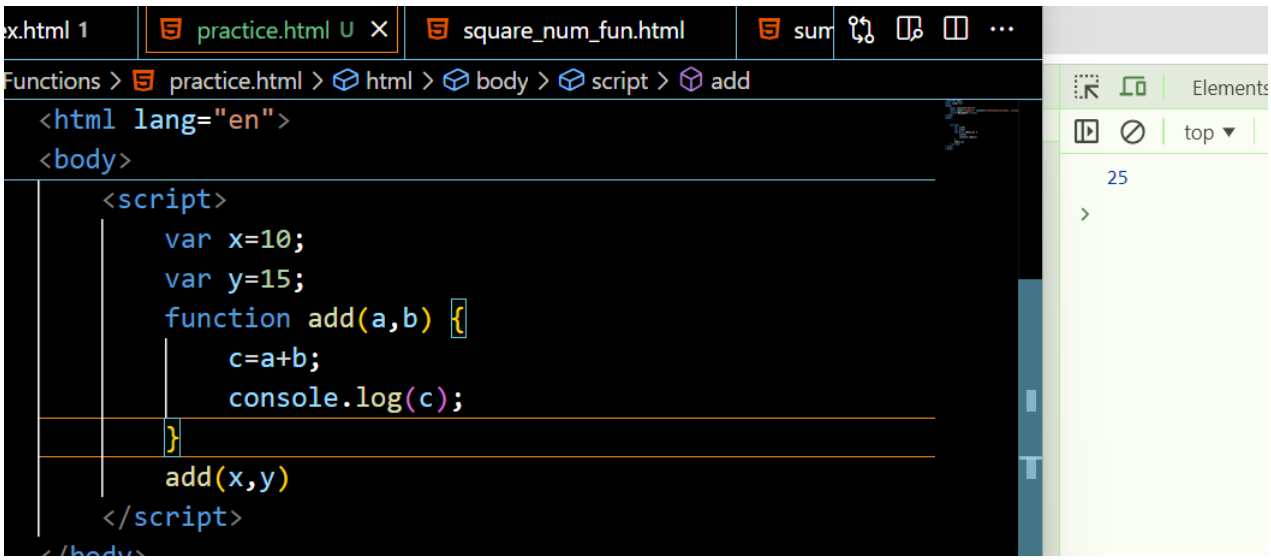
Syntax:

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

Example:



Example: adding two numbers



Example: even numbers in range

practice.html U | square_num_fun.html

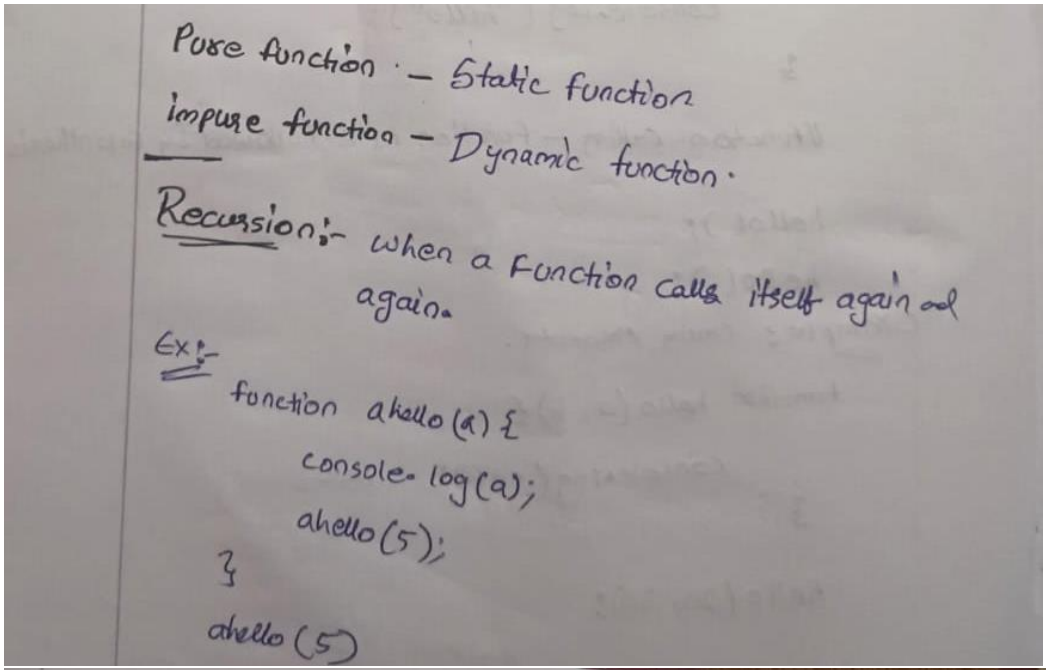
nctions > practice.html > html > body > script

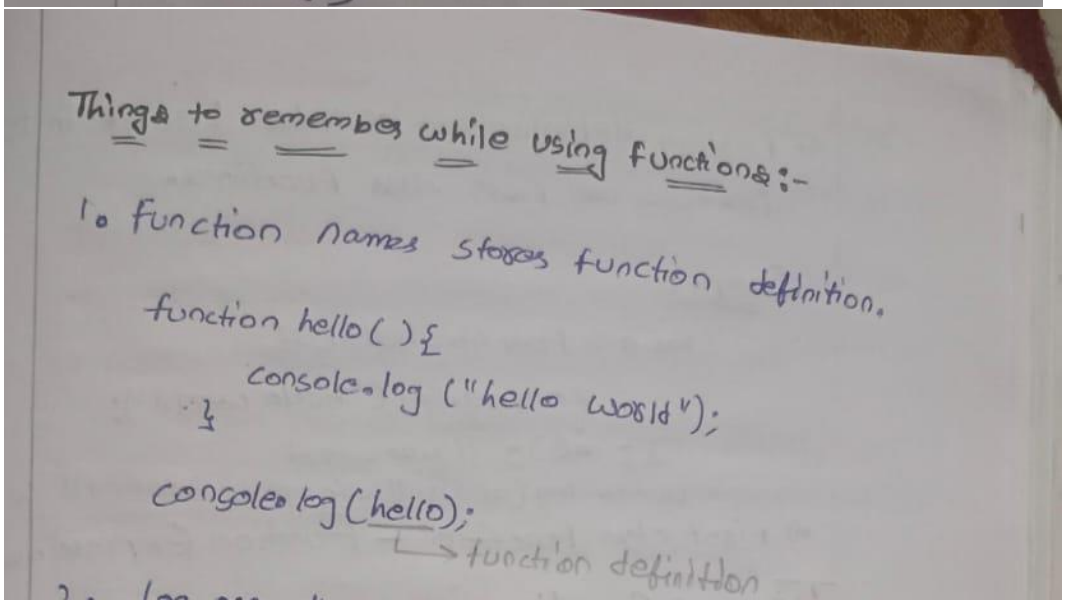
```
<html lang="en">
<body>
    <script>
        function even(a) {
            for(i=1;i<=a;i++){
                if(i%2==0){
                    console.log(i+"is even number");
                }
            }
        }
        even(10);
    </script>
```

Elements | Console

top

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





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());
```

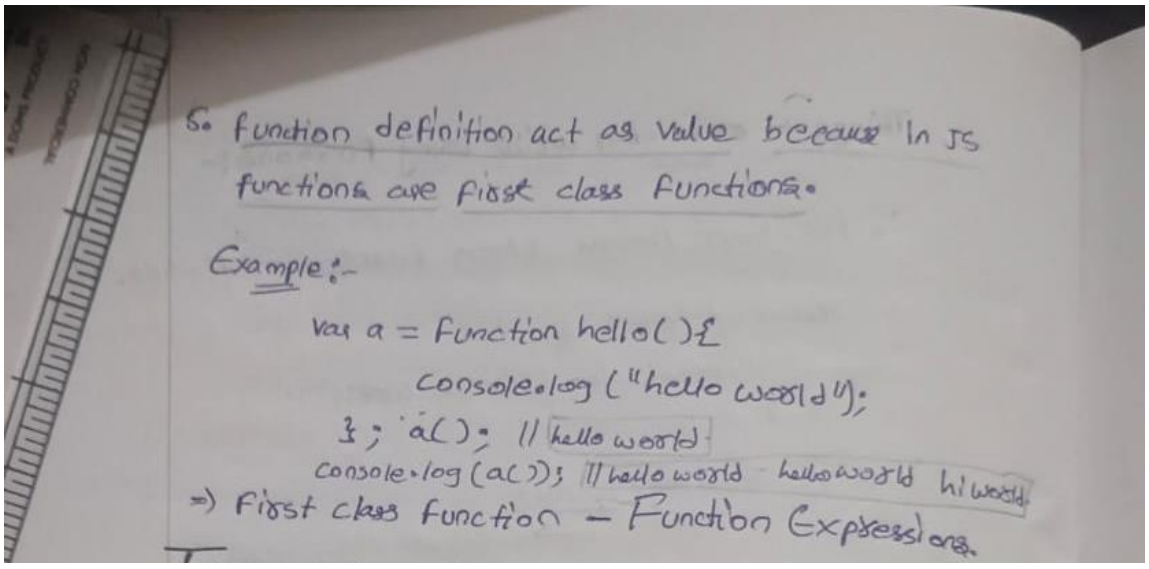
stores return value after execution

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

Annonymous function :-

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

Default parameters (ES-6) :- 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.

Example:-

```
function greet(a = "dear"){
    console.log("hello " + a);
}
greet("John");  // hello John
great ("");  // hello Dear
```

Example:

re assign a value

```
function greet(a, a, b){
    console.log(a, a, b);  // 3 3 5
}
greet(2, 3, 5)
```

Example:- to access nested function

```
function greet(){
    console.log("first function");
    return function greet2(){
        console.log("seconde function");
    }
}
greet()();
```

Research

1. Currying

Call back Function:-

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

Example:-

```
function hi(){
    console.log("hi");
}

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

Example:-

```
function payment(){
    Console.log("Payment Recived");
}
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.

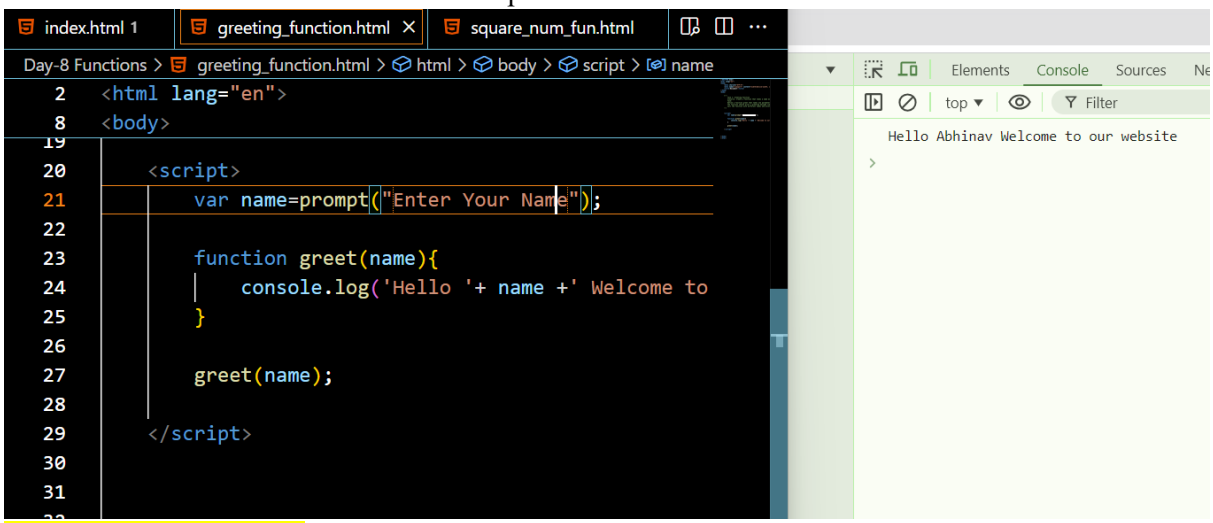
order(Payment);

↓ hof ↓ call back

Tasks

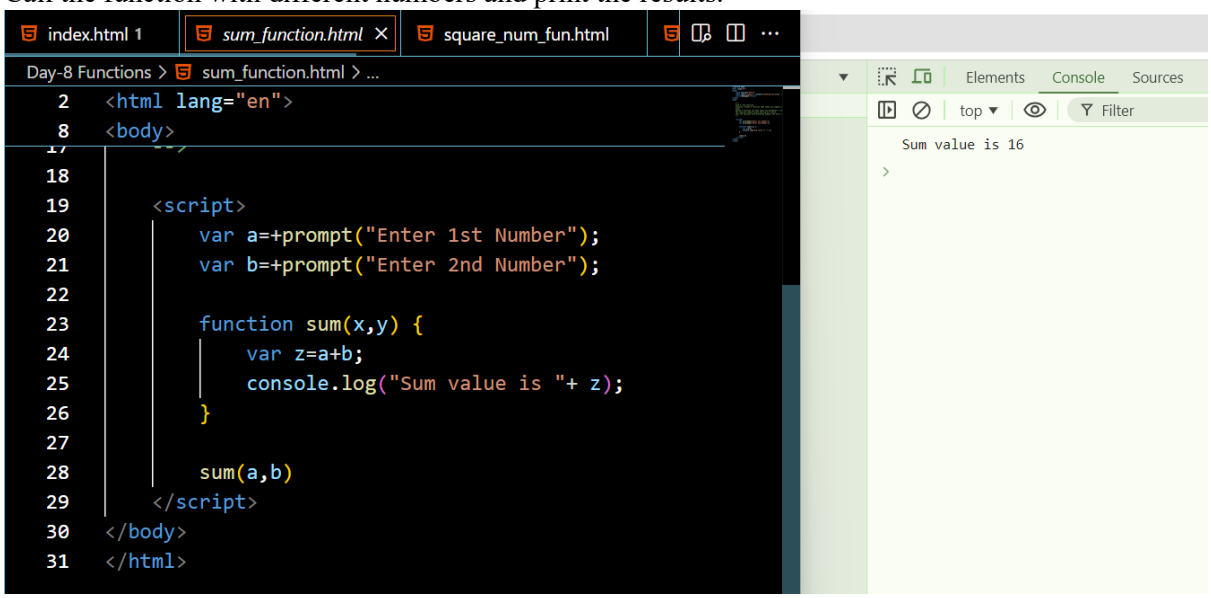
Task 1: Greeting Function

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.



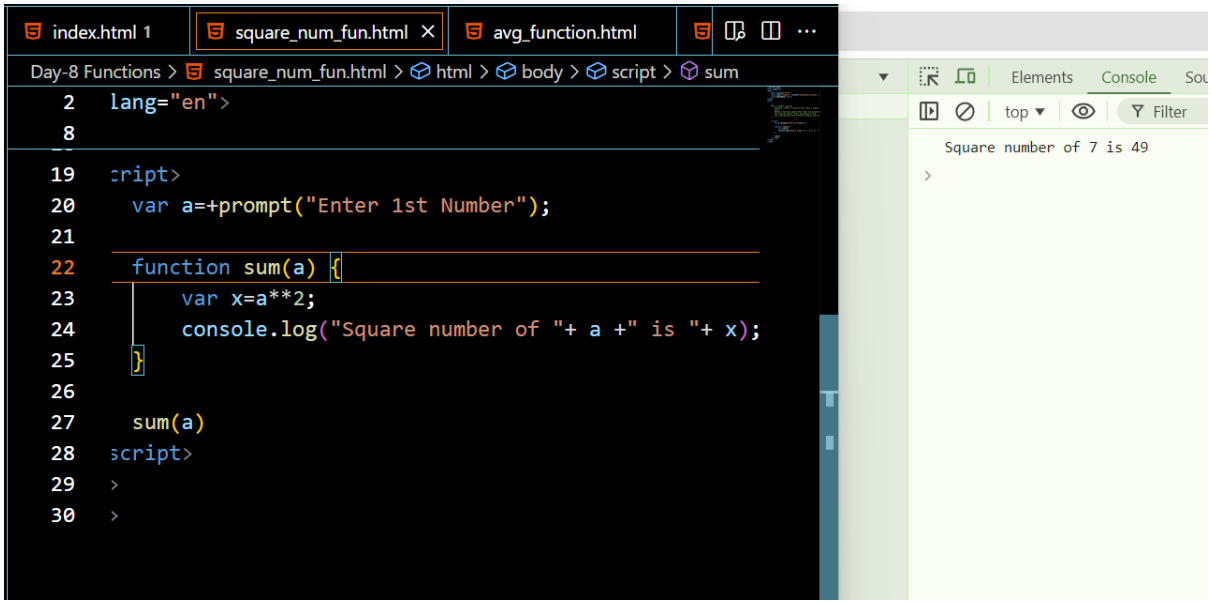
Task 2: Sum Function

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.



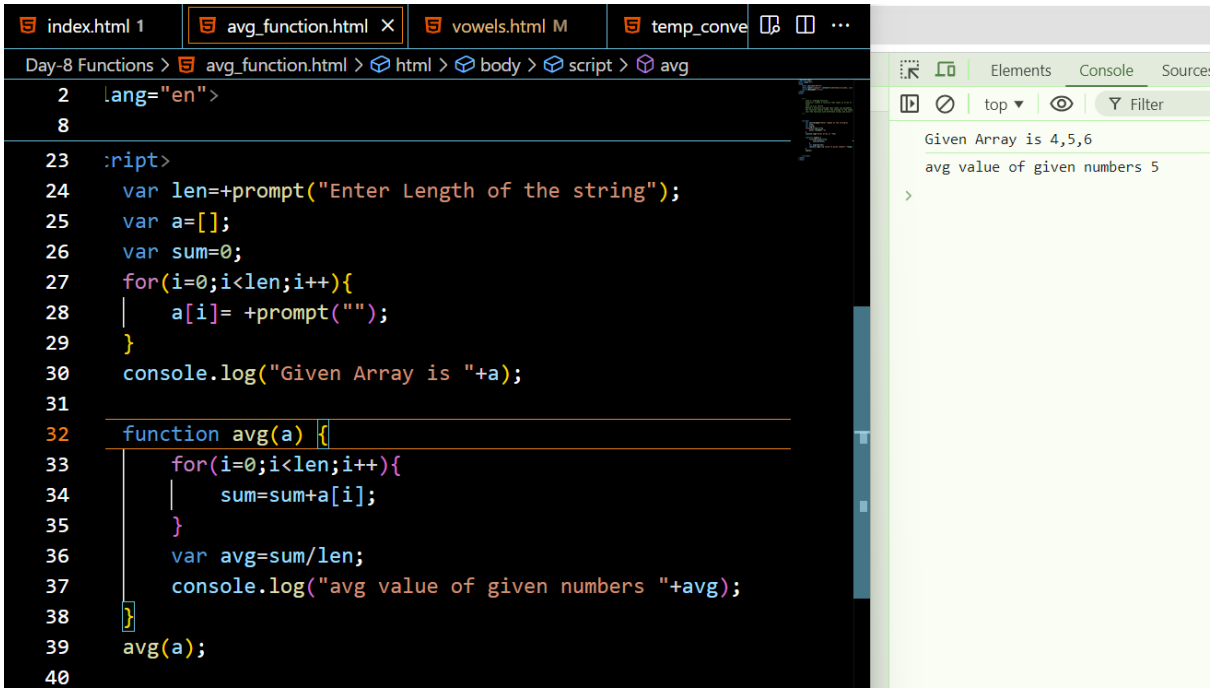
Task 3: Square Function

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.



Task 4: Average Function

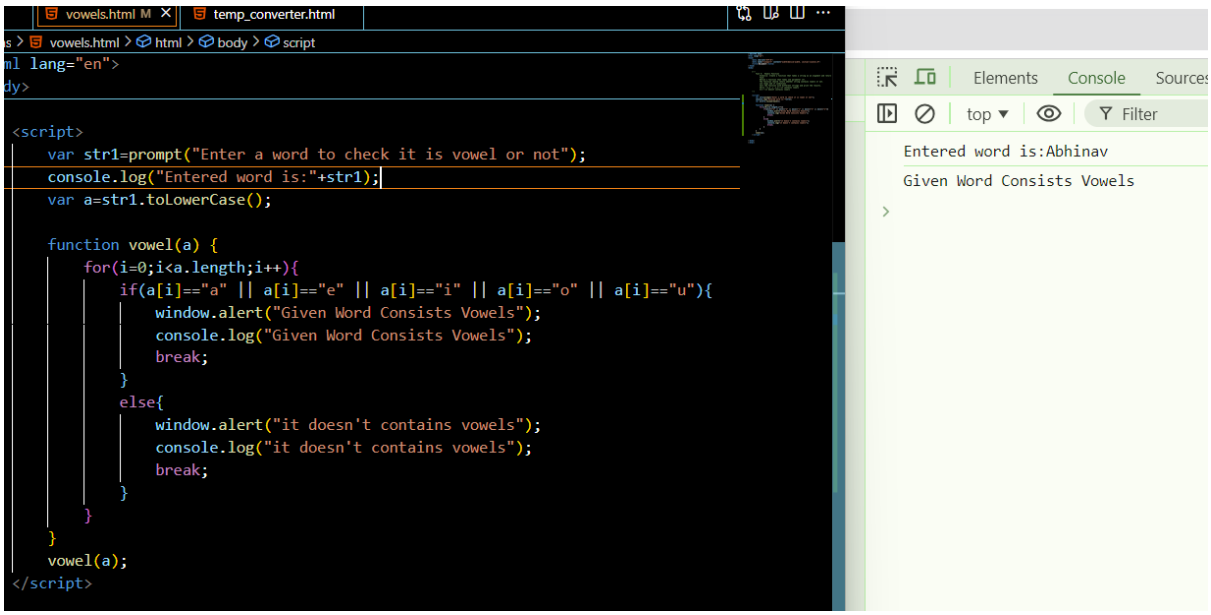
Scenario: Create a function that takes an array of numbers as an argument and returns the average.
Task:
hint:[2,5,2] =9/3=3
Define a function average that takes one parameter arr.
The function should return the average of the numbers in arr.
Call the function with different arrays and print the results.

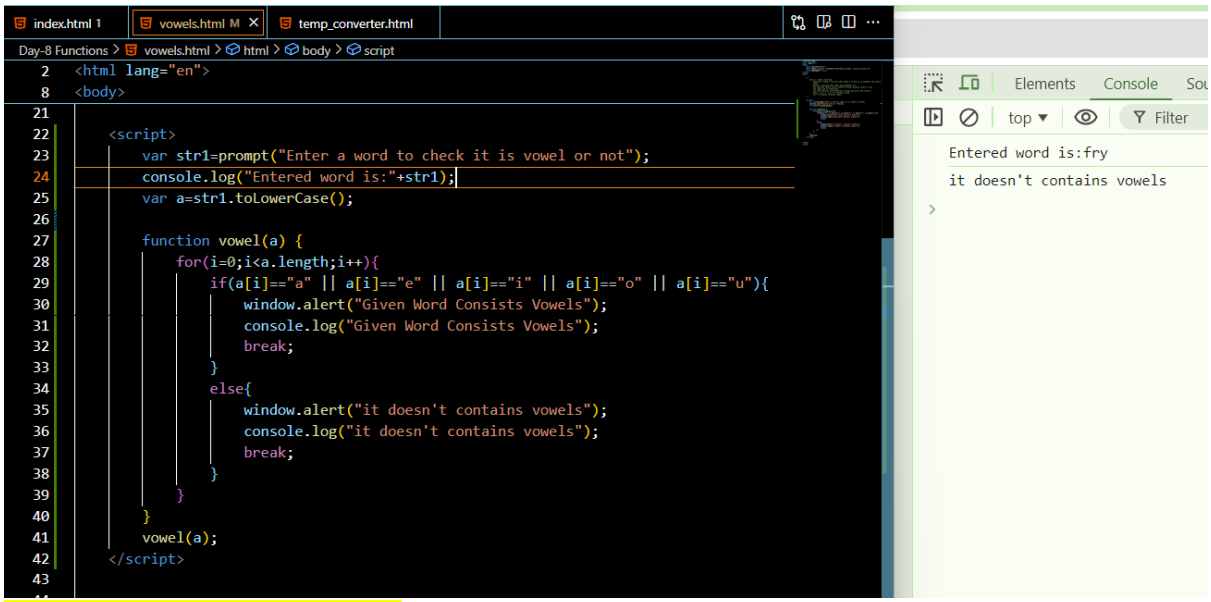


Task 5: Vowels Function

Scenario: Create a function that takes a string as an argument and returns weather the strings contains vowels or not .
Task:
Define a function that takes one parameter str.
The function should return whether string contains vowels or not.

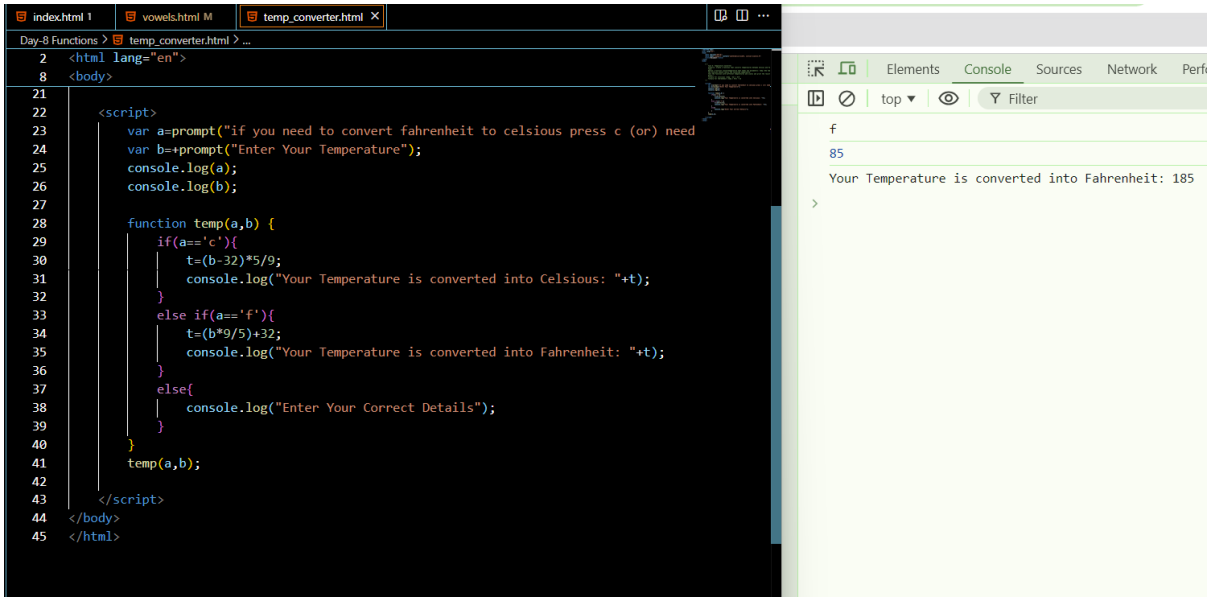
use loops and if conditions
Call the function with different strings and print the results.
hello --a,e ,i ,o,u---it contains vowels
hll---it doesnt contains vowels

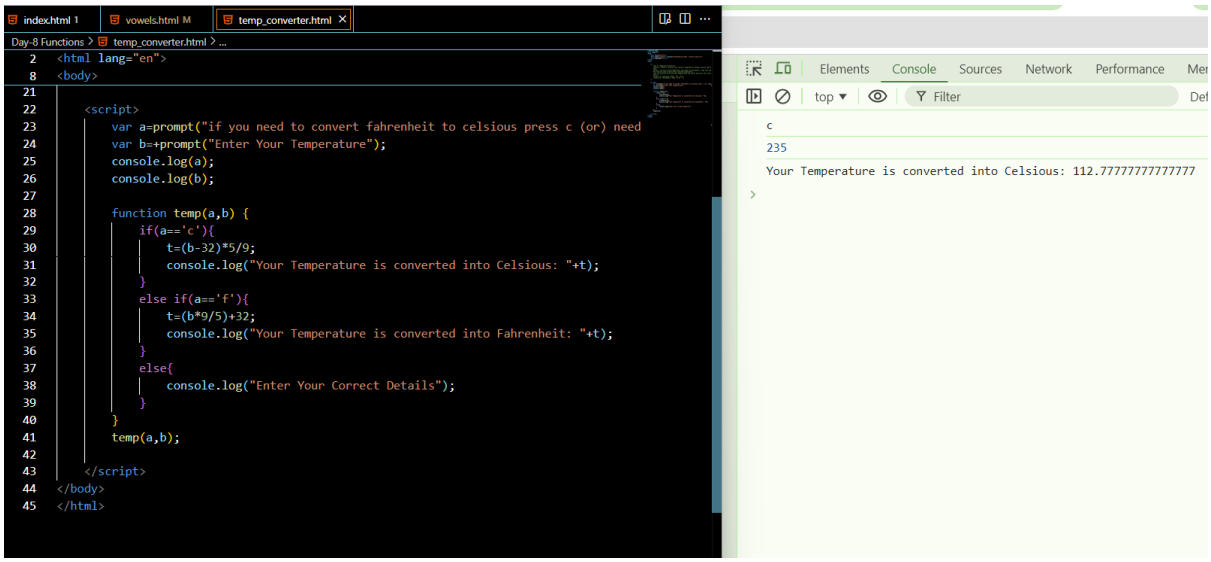




Task 6: Temperature Converter

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 fahrehnheit (temp * 9/5) + 32


index.html 1
vowels.html M
temp_converter.html
Day-8 Functions > temp_converter.html > ...
2 <html lang="en">
8 <body>
21
22 <script>
23 var a=prompt("if you need to convert fahrenheit to celsious press c (or) need
24 var b=+prompt("Enter Your Temperature");
25 console.log(a);
26 console.log(b);
27
28 function temp(a,b) {
29 if(a=='c'){
30 t=(b-32)*5/9;
31 console.log("Your Temperature is converted into Celsious: "+t);
32 }
33 else if(a=='f'){
34 t=(b*9/5)+32;
35 console.log("Your Temperature is converted into Fahrenheit: "+t);
36 }
37 else{
38 console.log("Enter Your Correct Details");
39 }
40 }
41 temp(a,b);
42
43 </script>
44 </body>
45 </html>
Elements
Console
Sources
Network
top
Filter
f
85
Your Temperature is converted into Fahrenheit: 185


index.html 1
vowels.html M
temp_converter.html
Day-8 Functions > temp_converter.html > ...
2 <html lang="en">
8 <body>
21
22 <script>
23 var a=prompt("if you need to convert fahrenheit to celsious press c (or) need
24 var b=+prompt("Enter Your Temperature");
25 console.log(a);
26 console.log(b);
27
28 function temp(a,b) {
29 if(a=='c'){
30 t=(b-32)*5/9;
31 console.log("Your Temperature is converted into Celsious: "+t);
32 }
33 else if(a=='f'){
34 t=(b*9/5)+32;
35 console.log("Your Temperature is converted into Fahrenheit: "+t);
36 }
37 else{
38 console.log("Enter Your Correct Details");
39 }
40 }
41 temp(a,b);
42
43 </script>
44 </body>
45 </html>
Elements
Console
Sources
Network
Performance
top
Filter
c
235
Your Temperature is converted into Celsious: 112.77777777777777

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.

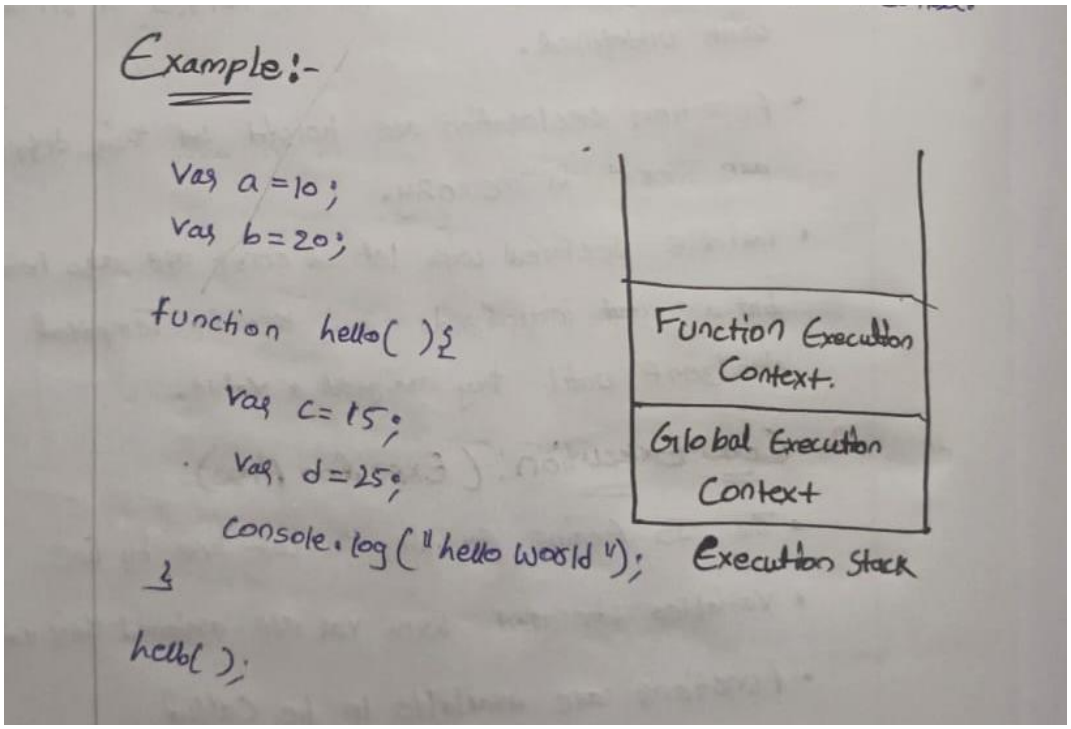
· Functions are available to be called.

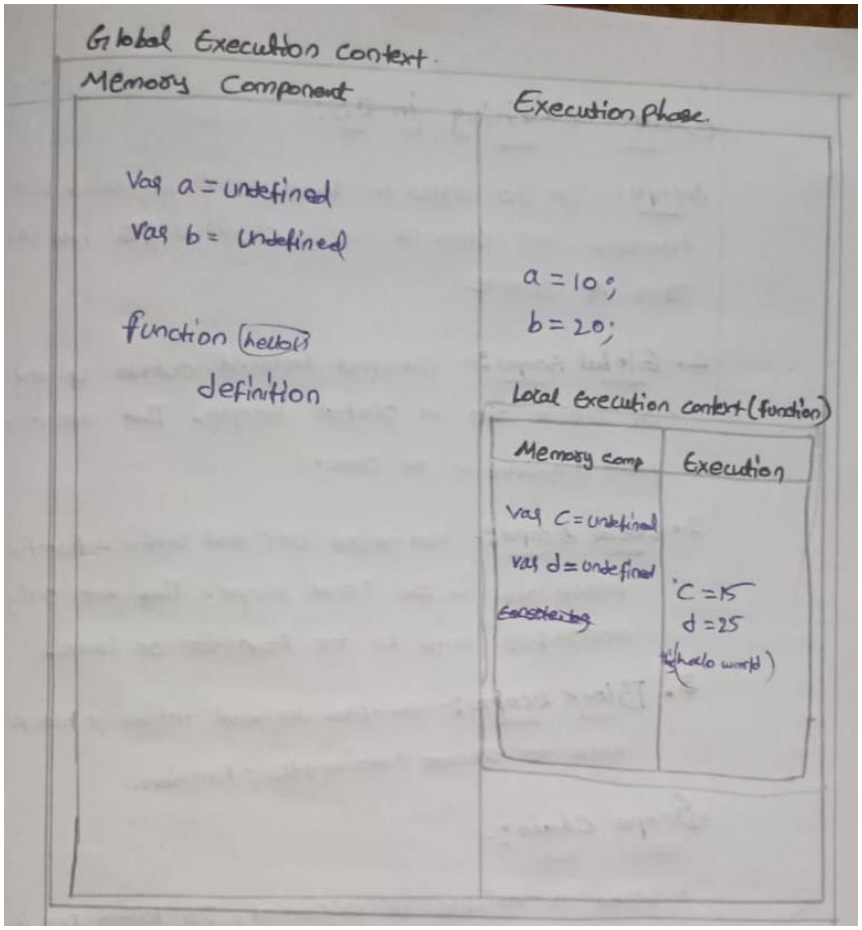
· Variables declared with **let** and **const** are assigned values when their declaration is encountered in the code.

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 a inner function have a 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();
```

```
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.

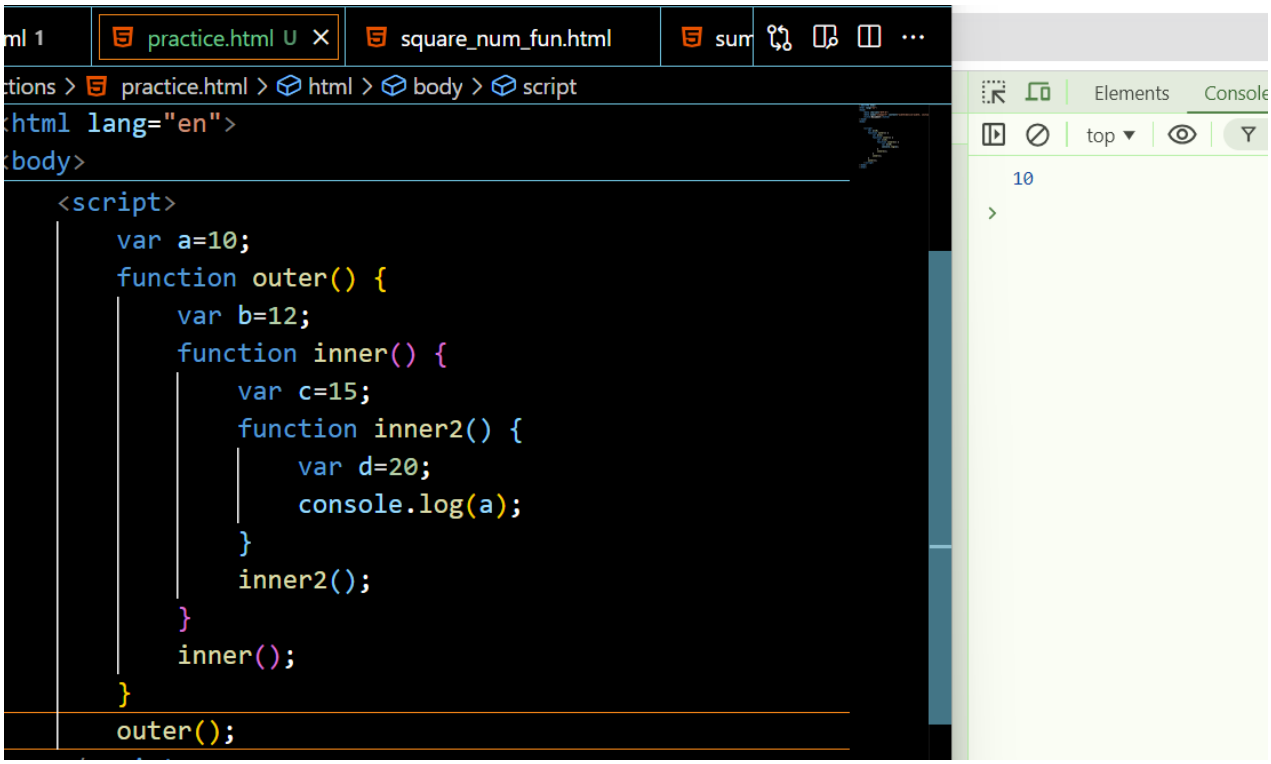
2. **Local Scope:** Variables declared within a function or block are in the local scope. They are only accessible within that function or block.

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:



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).

DOM- Document Object Model

DOM is a standard object model that allows programs and scripts to dynamically access and update the content, structure, and style of a document

Document Object Model (DOM) connects web pages to scripts languages by representing the structure of a document

The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree. With them, you can change the document's structure, style, or content.

Main Object:
Here's a breakdown of some key concepts related to the JavaScript DOM:

• Document: The top-level object in the DOM hierarchy, representing the entire HTML document. It serves as an entry point to access and manipulate the document's content.

```
console.log(document);
```

Logging document to the console in JavaScript will display the entire Document Object Model (DOM) of the current HTML page.

Methods for Accessing Elements:(Get methods using dom)

1)document.getElementById(): Retrieves an element by its unique ID

Example:

```
<div id="demo">Content1</div>
var a=document.getElementById("demo");
console.log(a);
```

2)document.getElementsByClassName(): Retrieves elements by their class name.

Example:

```
  <p class="myClass">Paragraph 1</p>
  <p class="myClass">Paragraph 2</p>
  var a=document.getElementsByClassName("myClass");
  console.log(a);

 //here the point to note is classnames are always in collections
 You can get the element by their index numbers
 Var a= document.getElementsByClassName(“myClass”)[0]
```

3) **document.getElementsByTagName():** Retrieves elements by their tag name.

```
  <h1>Heading</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  var a= document.getElementsByTagName("p");
  console.log(a);

//here the point to note is tagnames are always in collections
You can get the element by their index numbers
Var elementsByTagName= document.getElementByTagName(“p”)[0]
```

4)**Accessing Elements by CSS Selector:**

Example:

```
<div class="container">
  <p class="para">Paragraph 1</p>
  <p class="para">Paragraph 2</p>
</div>
```

querySelector() method allows you to select the first element in the document

Example:

```
var elementBySelector = document.querySelector(".para");//selects by classname
var myDiv = document.querySelector("#myDiv");//select by id
var elselector = document.querySelector("div");//select by element name
```

querySelectorAll -iIt operates similarly to querySelector(), but instead of returning only the first matching element, it returns a list of all matching elements.

Example:

```
var paragraphs = document.querySelectorAll(".para");//select all elements by class names
var divs = document.querySelectorAll("div");//select all div elements in a collections
```

Get content of the html

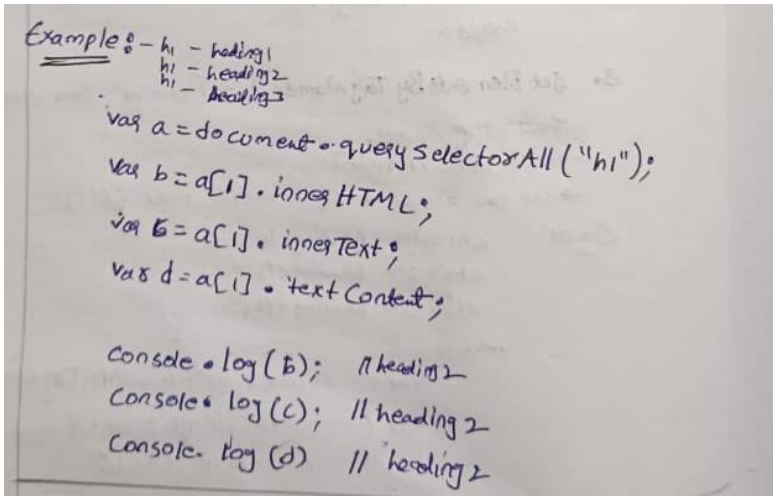
innerText and innerHTML are properties of DOM elements in JavaScript that deal with the content of HTML elements

innerText:

- innerText is a property that represents the visible text content of an element.
- It retrieves the text content of the element, excluding any HTML tags.

innerHTML:

- innerHTML is a property that represents the HTML content of an element.
- It retrieves or sets the HTML markup within the element, including any nested elements and tags.
- It can be used to dynamically change the structure and content of an element.



How to modify existing content

```
// Select the element by its ID
var paragraph = document.getElementById("myParagraph");

// Update the text content using innerText
paragraph.innerText = "Updated text!";
```

Modifing the Content :-

document.getElementById("id").innerHTML = "modified content";

1. innerText :-

document.getElementById("id").innerText = "Modified content";

Ex:-

```
<h1> heading 1 </h1>
<h1> heading 2 </h1>
<h1> heading 3 </h1>
<style>
    var a = document.querySelectorAll("h1");
    a[0].innerText = "hello world";
    console.log(a);
</style>
```

output:- :hello world
heading 2
heading 3

2. ~~Exam~~ innerHTML :-

Example:-

```
var a = document.querySelectorAll("h1");
a[0].innerText = "<u>hello world</u>";
console.log(a);        // <u> Hello world </u>

a[0].innerHTML = "<u> Hello world </u>";
a[1].innerHTML = "<del> Hello world </u>";

console.log(a);
```

Output:-

Hello world ;

~~Hello world~~;

heading 3

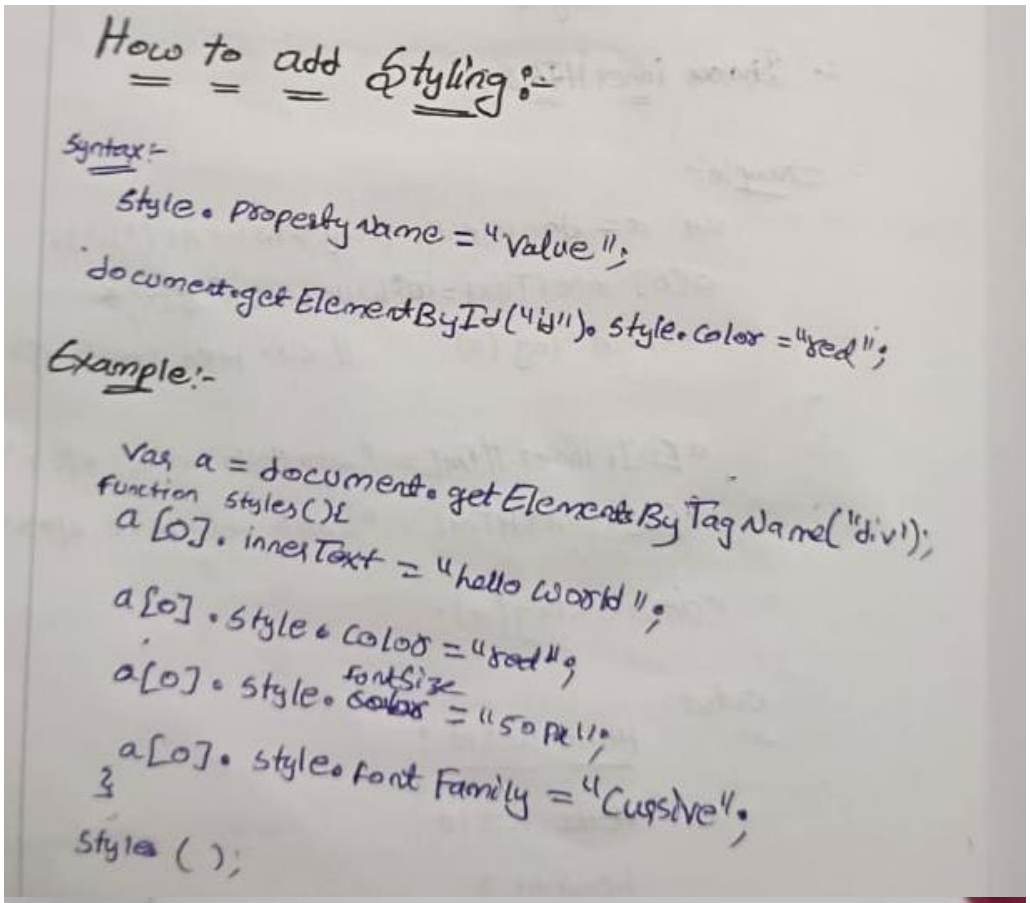
3. txt content:-

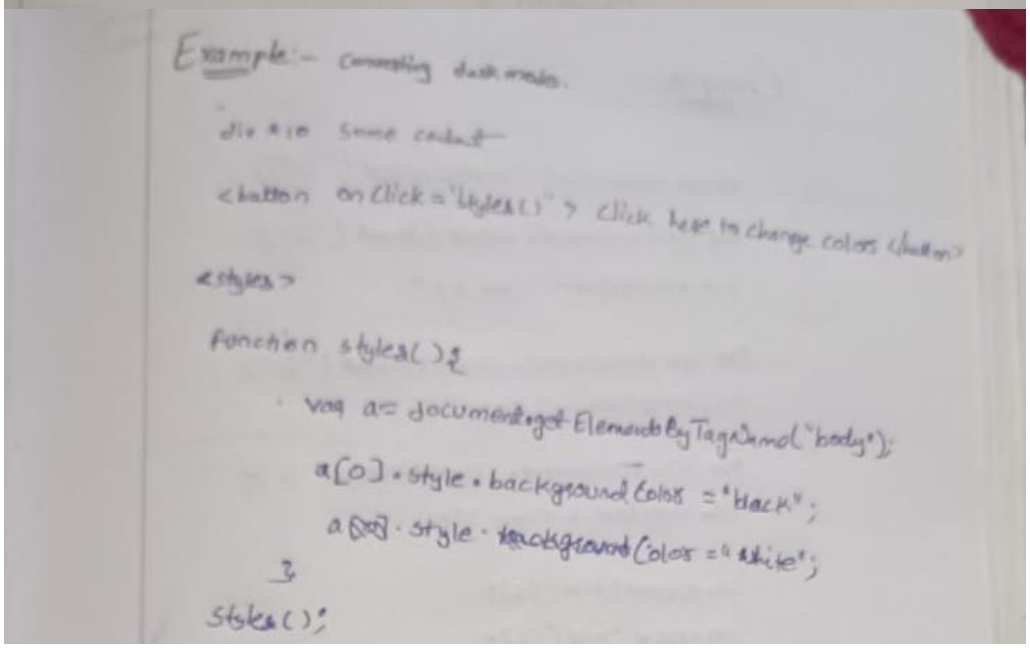
a[0].textContent = "<u> Hello world </u>";

// <u> Hello world </u>

How to apply styles using dom

Step 1: Access the element where you want to append the text node
Step 2: Apply styles





How to change attribute values by using setAttribute

This is some text and have id myDiv but it will changed to demo
var a=document.getElementById("myDiv").setAttribute("id","demo");
console.log(document)//can inspect and check weather it was changed or not
we can change the attribute by using .setAttribute.(“attribute name”,”attribute value”) //output

How to create element and how to append element in dom

Creating element

- A new paragraph element is created using document.createElement("p").
- The innerText property of the newly created paragraph element is set to "This is a dynamically created paragraph."
- The paragraph element is appended to the document body using **document.body.appendChild(newParagraph).**

Appendchild and Append

Append and appendChild methods are used in JavaScript to add nodes to the DOM, but they have some differences in terms of usage, accepted parameters, and behavior:

appendChild syntax:

parentNode.appendChild(newChild);

Parameters:

newChild: A single node (an element, text node, or any other node) that will be appended as the last child of parentNode

Behavior:

If the newChild is already in the DOM, it will be removed from its current position and moved to the new position.

Only accepts a single node.

Append syntax:

parentNode.append(node1, node2, node3);

Parameters:

nodes: One or more nodes or strings that will be appended as the last children of parentNode.

Behavior:

Can append multiple nodes and/or strings at once.
If a string is provided, it will be added as a text node.
Allows appending a combination of nodes and text.

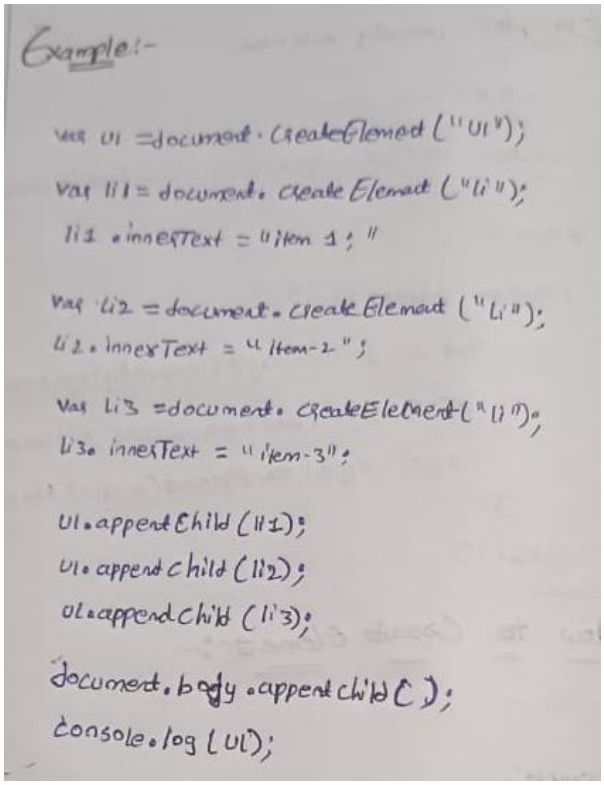
How to create textNode

```
// Step 1: Access the element where you want to append the text node
var myDiv = document.getElementById("myDiv");

// Step 2: Create a text node
var textNode = document.createTextNode("This is a dynamically created text node.");

// Step 3: Append the text node to the element
myDiv.appendChild(textNode);
```

- Access the element where you want to append the text node.
- Create a text node using document.createTextNode().
- Append the text node to the desired element.

Example:

```
// Create a new paragraph element
var newParagraph= document.createElement("p");

// Set innertext or other properties if needed
newParagraph.innerText = "This is a dynamically created paragraph.";

// Append the paragraph to the document body
document.body.appendChild(newParagraph);
```

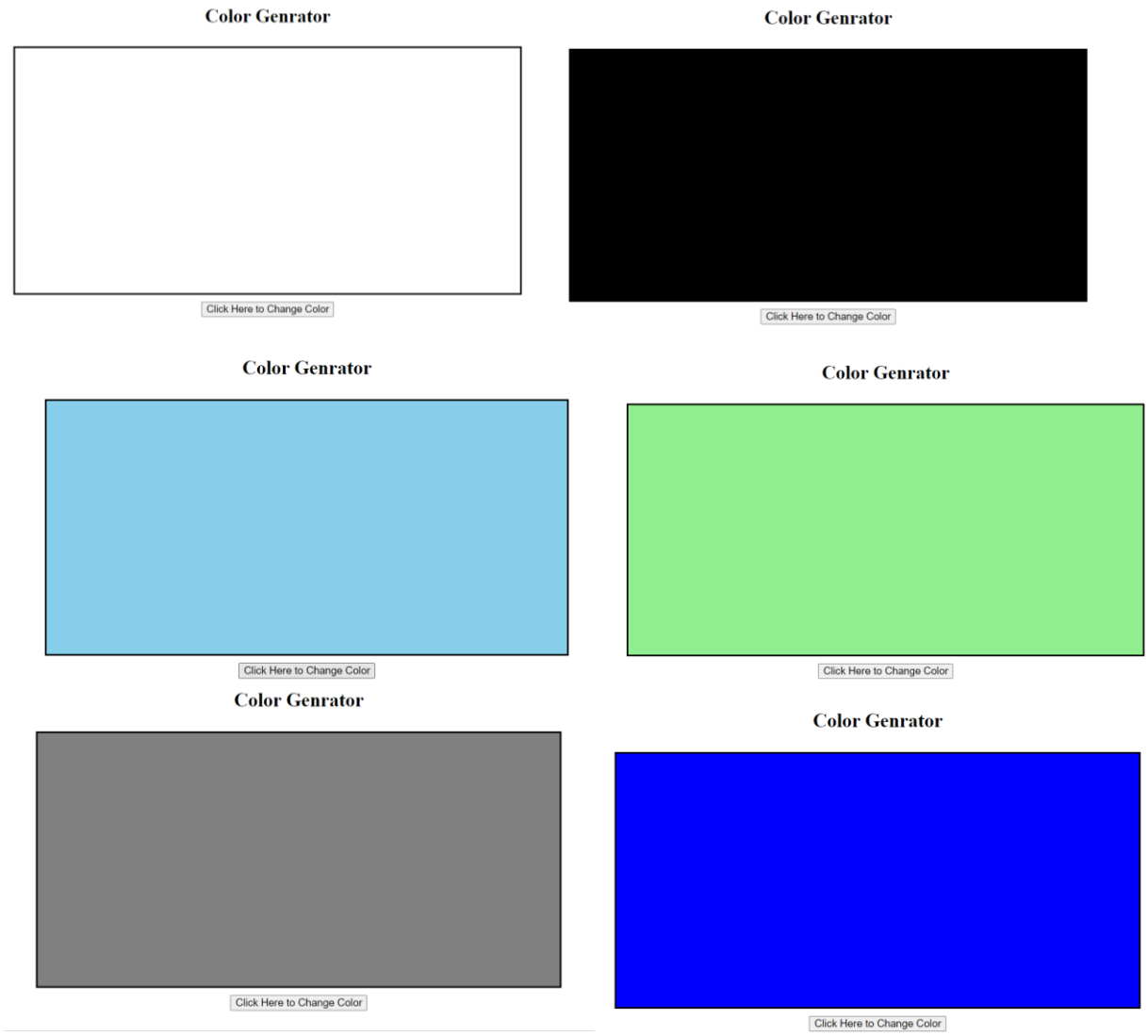


Tasks:

1. **Color Genrator:** By clicking button change multiple colors (use count and arrays)

Code:

```
<script>
    var body=document.getElementsByTagName("body")[0];
    body.style.display="flex";
    body.style.flexDirection="column";
    body.style.width="98vw";
    body.style.height="98vh";
    body.style.justifyContent="center";
    body.style.alignItems="center";
    body.style.gap="10px";

    var head1=document.createElement("h1");
    head1.innerText="Color Genrator";
    head1.style.fontSize="28px";
    document.body.appendChild(head1);

    var area=document.createElement("div");
    area.style.width="50%";
    area.style.height="50vh";
    area.style.border="3px solid black";
    area.style.color="white";
    document.body.appendChild(area);

    var btn=document.createElement("button");
    btn.innerText="Click Here to Change Color";
    btn.style.fontSize="15px";
    btn.addEventListener("click", cg);
    document.body.appendChild(btn);

    var count=0;
    var arr=['black','skyblue','lightgreen','grey','blue'];
    function cg(){
        area.style.backgroundColor=arr[count];
        count++;
        if(count>=arr.length){
            count=0;
        }
    }
</script>
```



2. **Resume:** create a Resume without using HTML

```
var body=document.getElementsByTagName("body")[0];
body.style.width="48vw";
body.style.height="auto";
body.style.border="3px solid black";

// Name
var head1=document.createElement("h1");
head1.innerText="ABHINAV SAI ";
head1.style.textAlign="center";
// head1.style.paddingLeft="25px";
head1.style.color="green";
head1.style.fontSize="40px";
document.body.appendChild(head1);

// CAREER OBJECTIVE
var sh1=document.createElement("h3");
sh1.innerText="CAREER OBJECTIVE";
sh1.style.paddingLeft="25px";
sh1.style.color="red";
sh1.style.fontSize="26px";
document.body.appendChild(sh1);

// CAREER OBJECTIVE statement
var co=document.createElement("p");
```

```
co.innerText="Responsible and motivated student ready to apply education in the workplace. Offers excellent technical abilities with software and applications, ability to handle challenging work, and excellent time management skills.";
co.style.paddingLeft="25px";
document.body.appendChild(co);

// Technical Skills
var sh2=document.createElement("h3");
sh2.innerText="Technical Skills";
sh2.style.paddingLeft="25px";
sh2.style.color="red";
sh2.style.textAlign="left";
sh2.style.fontSize="26px";
document.body.appendChild(sh2);
var ts=document.createElement("ul");
var li1=document.createElement("li");
var li2=document.createElement("li");
var li3=document.createElement("li");
var li4=document.createElement("li");
var li5=document.createElement("li");
ts.style.listStyleType="none";
li1.innerText="Java";
li2.innerText="Python";
li3.innerText="HTML";
li4.innerText="CSS";
li5.innerText="Java Script";
ts.appendChild(li1);
ts.appendChild(li2);
ts.appendChild(li3);
ts.appendChild(li4);
ts.appendChild(li5);
document.body.appendChild(ts);


// Soft Skills
var sh3=document.createElement("h3");
sh3.innerText="Soft Skills";
sh3.style.paddingLeft="25px";
sh3.style.color="red";
sh3.style.textAlign="left";
sh3.style.fontSize="26px";
document.body.appendChild(sh3);
var ts=document.createElement("ul");
var li1=document.createElement("li");
var li2=document.createElement("li");
var li3=document.createElement("li");
var li4=document.createElement("li");
ts.style.listStyleType="none";
li1.innerText="Adaptapility";
li2.innerText="Self Motivation";
li3.innerText="Self Confidence";
li4.innerText="Team Work";
ts.appendChild(li1);
ts.appendChild(li2);
ts.appendChild(li3);
ts.appendChild(li4);
document.body.appendChild(ts);

//Hobbies.
var sh4=document.createElement("h3");
sh4.innerText="Soft Skills";
sh4.style.paddingLeft="25px";
sh4.style.color="red";
sh4.style.textAlign="left";
sh4.style.fontSize="26px";
document.body.appendChild(sh4);

var hob=document.createElement("ul");
hob.style.listStyleType="none";
hobbies=["playing Cricket","Listening Music","watching Movies"];
for(var i=0;i<hobbies.length;i++){
   var li=document.createElement("li");
   li.innerText=hobbies[i];
   li.style.color="green";
   li.style.fontSize="16px";
   hob.appendChild(li);
}
```

```
document.body.appendChild(hob);

// Declaration
var sh5=document.createElement("h3");
sh5.innerText="Declaration";
sh5.style.paddingLeft="25px";
sh5.style.color="red";
sh5.style.textAlign="left";
sh5.style.fontSize="26px";
document.body.appendChild(sh5);

// Declaration Statement
var ds=document.createElement("p");
ds.innerText="I hereby declare that the information furnished above is genuine to the best of my belief and I hold the responsibility of their authenticity and correctness.";
ds.style.paddingLeft="25px";
document.body.appendChild(ds);

// sign
var sign=document.createElement("p");
sign.innerText="-Abhinav Sai";
sign.style.paddingLeft="600px";
document.body.appendChild(sign);
```

ABHINAV SAI

CAREER OBJECTIVE

Responsible and motivated student ready to apply education in the workplace. Offers excellent technical abilities with software and applications, ability to handle challenging work, and excellent time management skills.

Technical Skills

- Java
- Python
- HTML
- CSS
- Java Script

Soft Skills

- Adaptapility
- Self Motivation
- Self Confidence
- Team Work

Soft Skills

- playing Cricket
- Listening Music
- watching Movies

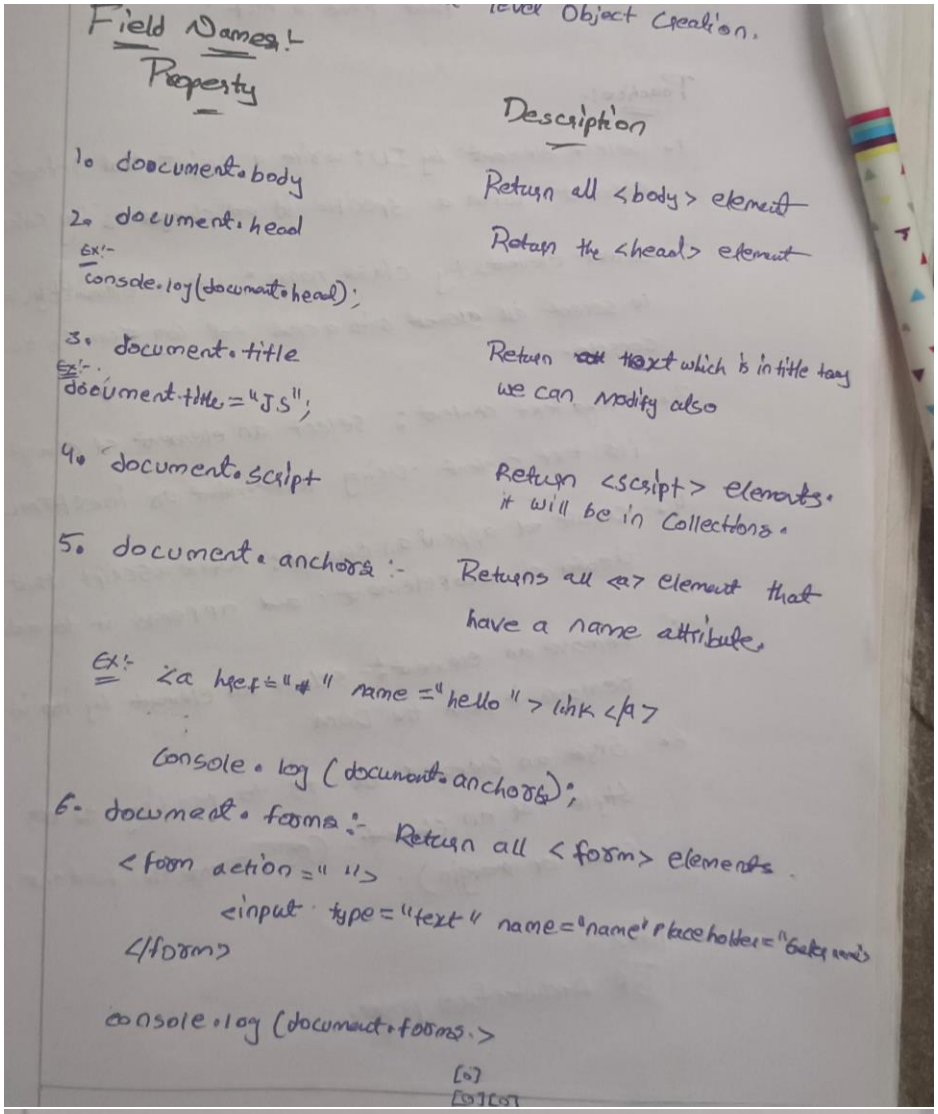
Declaration

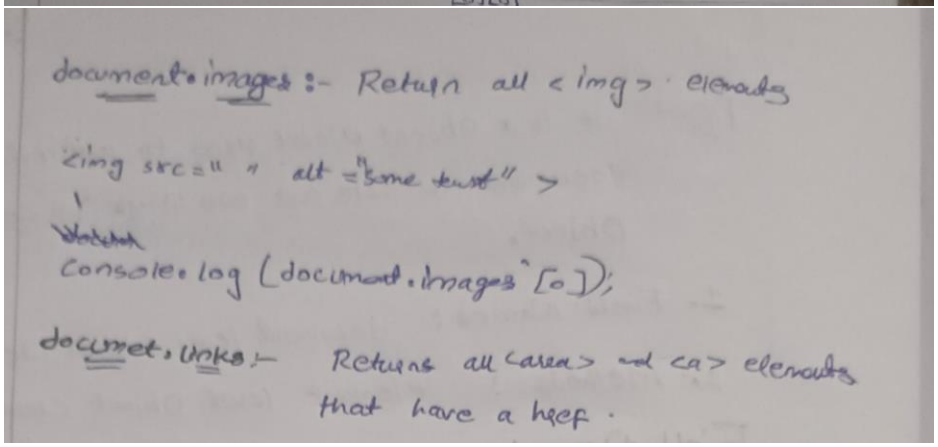
I hereby declare that the information furnished above is genuine to the best of my belief and I hold the responsibility of their authenticity and correctness.

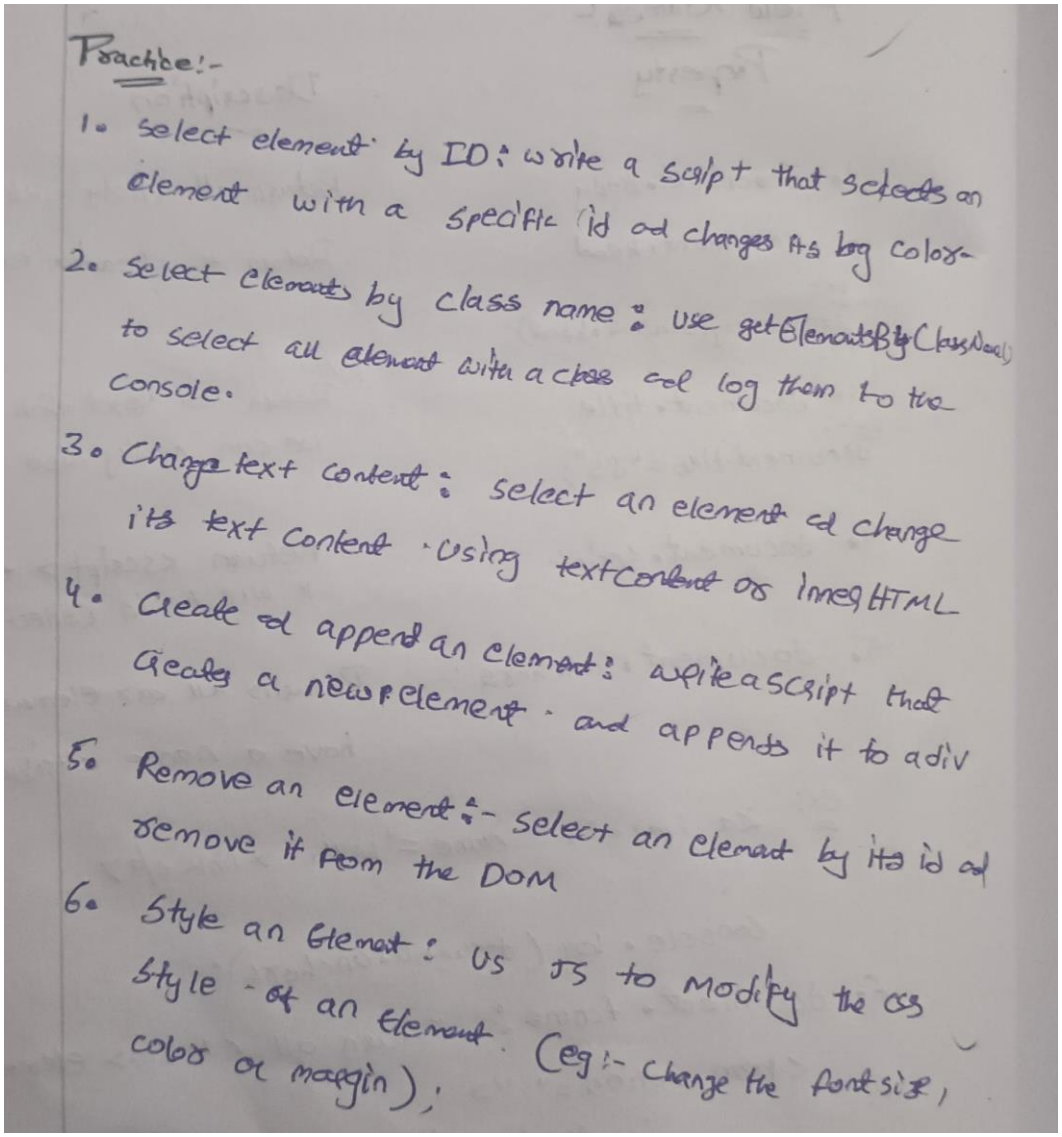
-Abhinav Sai

DOM

It is a object model used to manipulate the document and there are two ways to create document object
1) Field Names – document level object creation
2) Methods – element level object creation



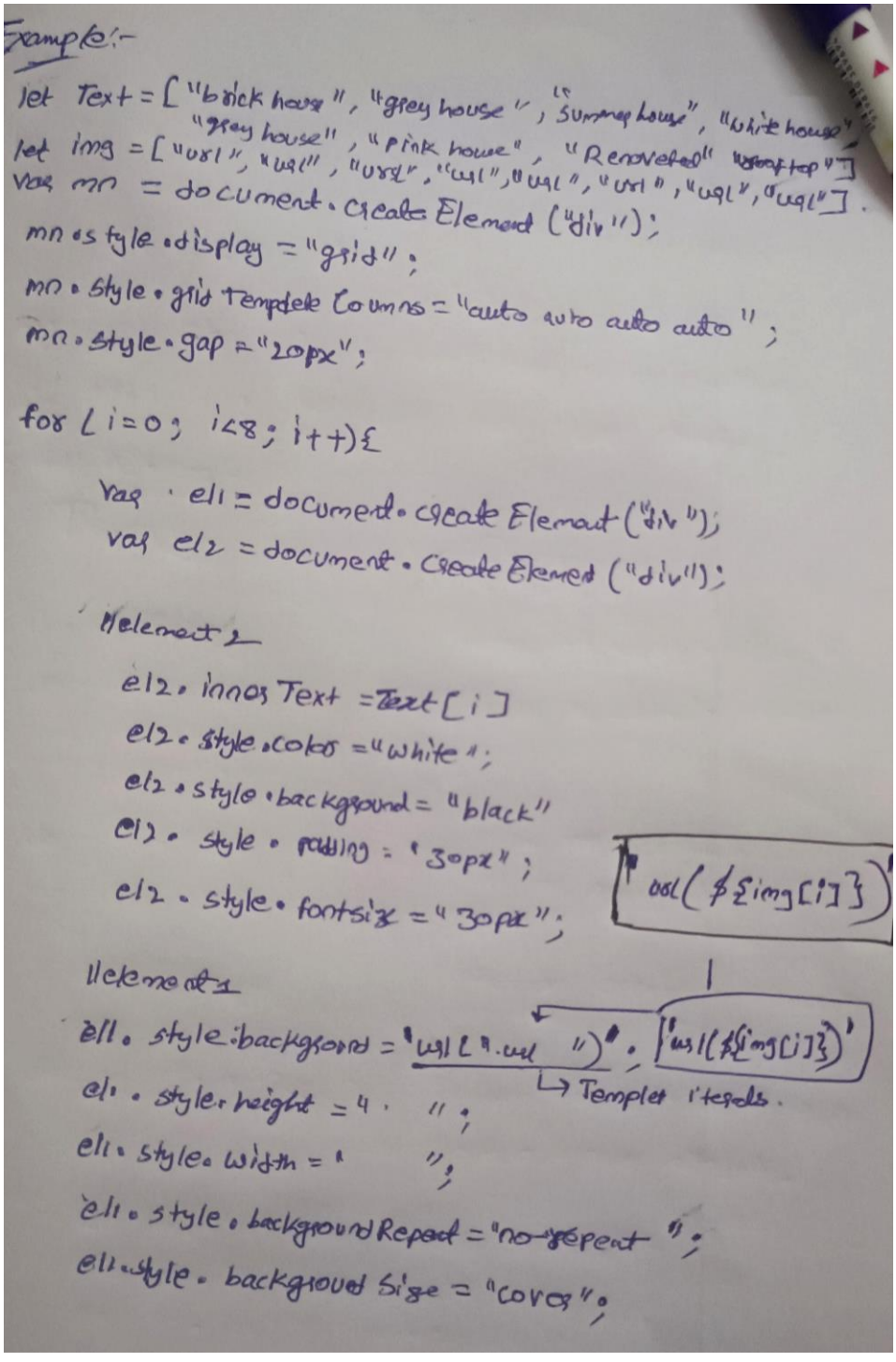





```
//1. Changing bg color
    var a=document.getElementById("p1");
    a.style.backgroundColor="navy";
    a.style.color="White";
    console.log(a);

    //2. select elements by class name
    var b=document.getElementsByClassName("p2");
    console.log(b);

    //3. changing Text
    var c=document.querySelector("h3");
    c.innerHTML="Changed text";
    console.log(c);

    //4. create and append
    var d1=document.querySelector(".app");
    var d=document.createElement("h2");
    d.innerText="h2 is created";
    d1.appendChild(d);

    //5. Remove an Element
    var e=document.getElementById("id1");
    id1.remove();
```

```
// 6. Style an element
var f=document.getElementsByTagName("h2");
f[0].style.backgroundColor="navy";
f[0].style.color="white";
```

EXAMPLE: Br Architecture Houses

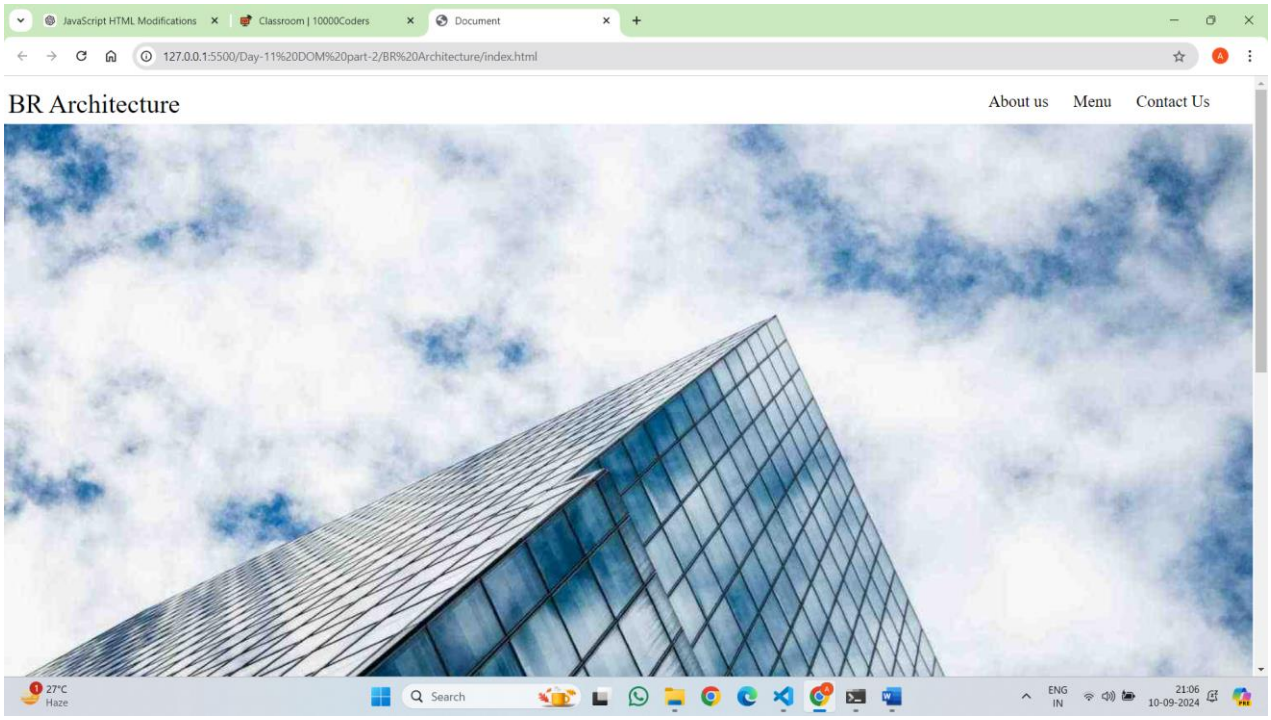


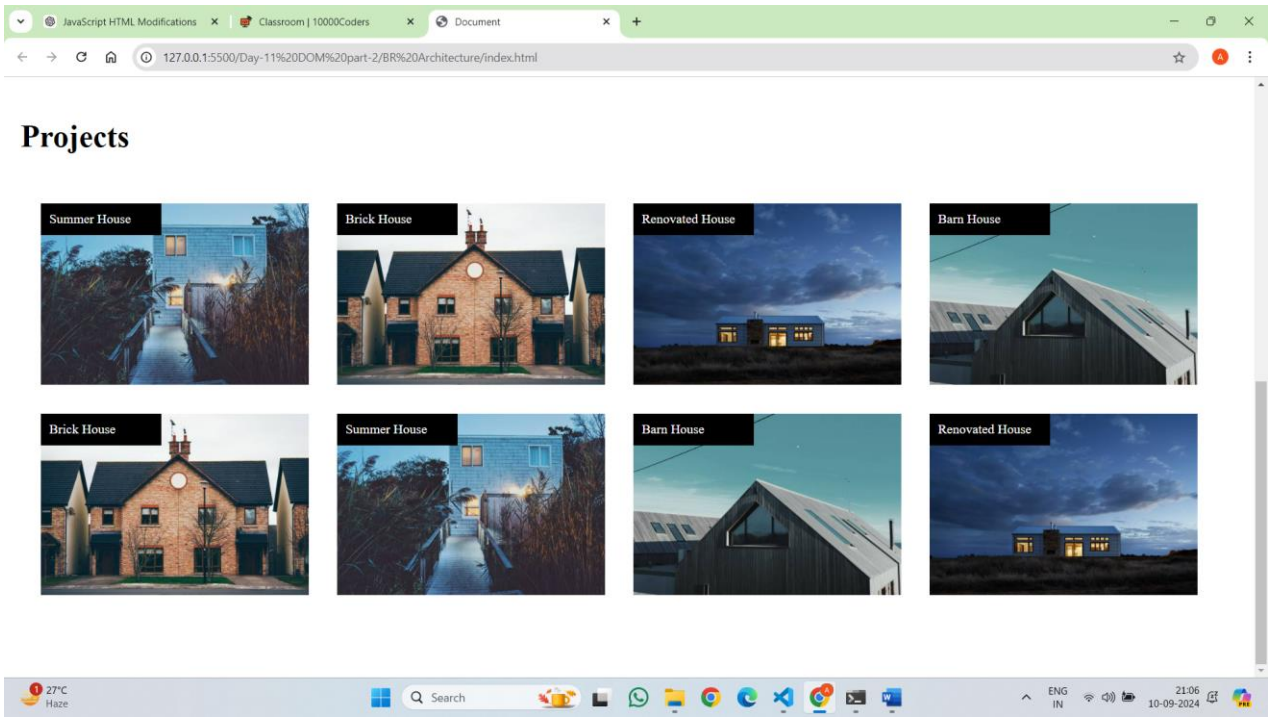
Task:

```
var nav=document.createElement("div");
        nav.style.display="flex";
        nav.style.justifyContent="space-between";
        nav.style.padding="5px";
        nav.style.height="8vh";
        nav.style.width="98vw";
        nav.style.position="relative";

        var nav1=document.createElement("div");
        nav1.style.paddingTop="10px";
        var br=document.createElement("a");
        br.innerText="BR Architecture";
        br.style.fontSize="32px";
        br.style.color="black";
        br.setAttribute("href","#");
        br.style.textDecoration="none";
        nav1.appendChild(br);
        nav.appendChild(nav1);


        var nav2=document.createElement("div");
        nav2.style.paddingRight="20px";
        nav2.style.paddingTop="15px";


        var about1=document.createElement("a");
        about1.innerText="About us";
        about1.style.padding="15px";
        about1.setAttribute("href","#");
        about1.style.color="black";
        about1.style.textDecoration="none";
        about1.style.fontSize="20px";
        nav2.appendChild(about1);

        var menu1=document.createElement("a");
        menu1.innerText="Menu";
        menu1.style.padding="15px";
        menu1.setAttribute("href","#");
        menu1.style.textDecoration="none";
        menu1.style.color="black";
        menu1.style.fontSize="20px";
        nav2.appendChild(menu1);

        var contact1=document.createElement("a");
        contact1.innerText="Contact Us";
        contact1.style.padding="15px";
        contact1.setAttribute("href","#");
        contact1.style.textDecoration="none";
        contact1.style.color="black";
        contact1.style.fontSize="20px";
        nav2.appendChild(contact1);

        nav.appendChild(nav2);
        document.body.appendChild(nav);
        console.log(nav);

        //home section
        var homee=document.createElement("div");
```

```
var home=document.createElement("div");
home.style.background=`url("architect.jpg")`;
home.style.backgroundRepeat="no-repeat";
home.style.backgroundSize="cover";
home.style.height="95vh";
home.style.width="98.8vw";
homee.appendChild(home);
document.body.appendChild(homee);

// Houses
let text=["Summer House","Brick House" ,"Renovated House","Barn House","Brick House","Summer House","Barn House","Renovated House"];
let img=["https://www.w3schools.com/w3images/house5.jpg","https://www.w3schools.com/w3images/house2.jpg","https://www.w3schools.com/w3images/house3.jpg","https://www.w3schools.com/w3images/house4.jpg","https://www.w3schools.com/w3images/house2.jpg","https://www.w3schools.com/w3images/house5.jpg","https://www.w3schools.com/w3images/house4.jpg","https://www.w3schools.com/w3images/house3.jpg"];

var hs=document.createElement("div");
hs.style.height="100vh";
hs.style.width="98px";

var h1=document.createElement("h1");
h1.innerText="Projects";
h1.style.color="black";
h1.style.fontSize="38px";
h1.style.padding="50px 10px 60px 20px";

hs.appendChild(h1);

var houses=document.createElement("div");
houses.style.display="grid";
houses.style.gridTemplateColumns="auto auto auto auto";
houses.style.justifyContent= "space-around" ;
houses.style.paddingLeft="45px";
houses.style.gap="35px";

for(var i=0;i<8; i++){
   var hs1=document.createElement("div");
   var hs2=document.createElement("div");

   // element-2
   hs1.innerHTML=text[i];
   hs1.style.color="white";
   hs1.style.background="black";
   hs1.style.width="45%";

   hs1.style.padding="10px";
   hs1.style.fontSize="16px";

   //element-1
   hs2.style.background=`url(${img[i]})`;
   hs2.style.height="220px";
   hs2.style.width="325px";
   hs2.style.backgroundRepeat="no-repeat";
   hs2.style.backgroundSize="cover";

   hs2.appendChild(hs1);
   houses.appendChild(hs2);
   hs.appendChild(houses);
   document.body.appendChild(hs);
}
```

OUTPUT:


JavaScript HTML Modifications
Classroom | 10000Coders
Document
127.0.0.1:5500/Day-11%20DOM%20part-2/BR%20Architecture/index.html
BR Architecture
About us
Menu
Contact Us


JavaScript HTML Modifications
Classroom | 10000Coders
Document
127.0.0.1:5500/Day-11%20DOM%20part-2/BR%20Architecture/index.html
Projects
Summer House
Brick House
Renovated House
Barn House
Brick House
Summer House
Barn House
Renovated House

Topic: Dom – setAttribute, getAttribute, classlist, event listeners and event handlers

How to change attribute values by using setAttribute

Syntax:

This is some text and have id myDiv but it will changed to demo

var a=document.getElementById("myDiv").**setAttribute**("id","demo");

console.log(document)//can inspect and check weather it was changed or not

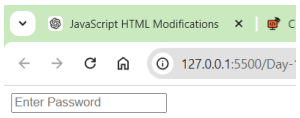
we can change the attribute by using .setAttribute.(“attribute name”,”attribute value”)

//output

Example-1:

```
var a=document.createElement("form");
      var b=document.createElement("input");
      b.setAttribute("type","password");
      b.setAttribute("placeholder","Enter Password");
      a.appendChild(b);
      document.body.appendChild(a);
```

Output:

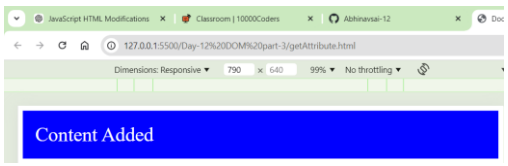


Example-2: img

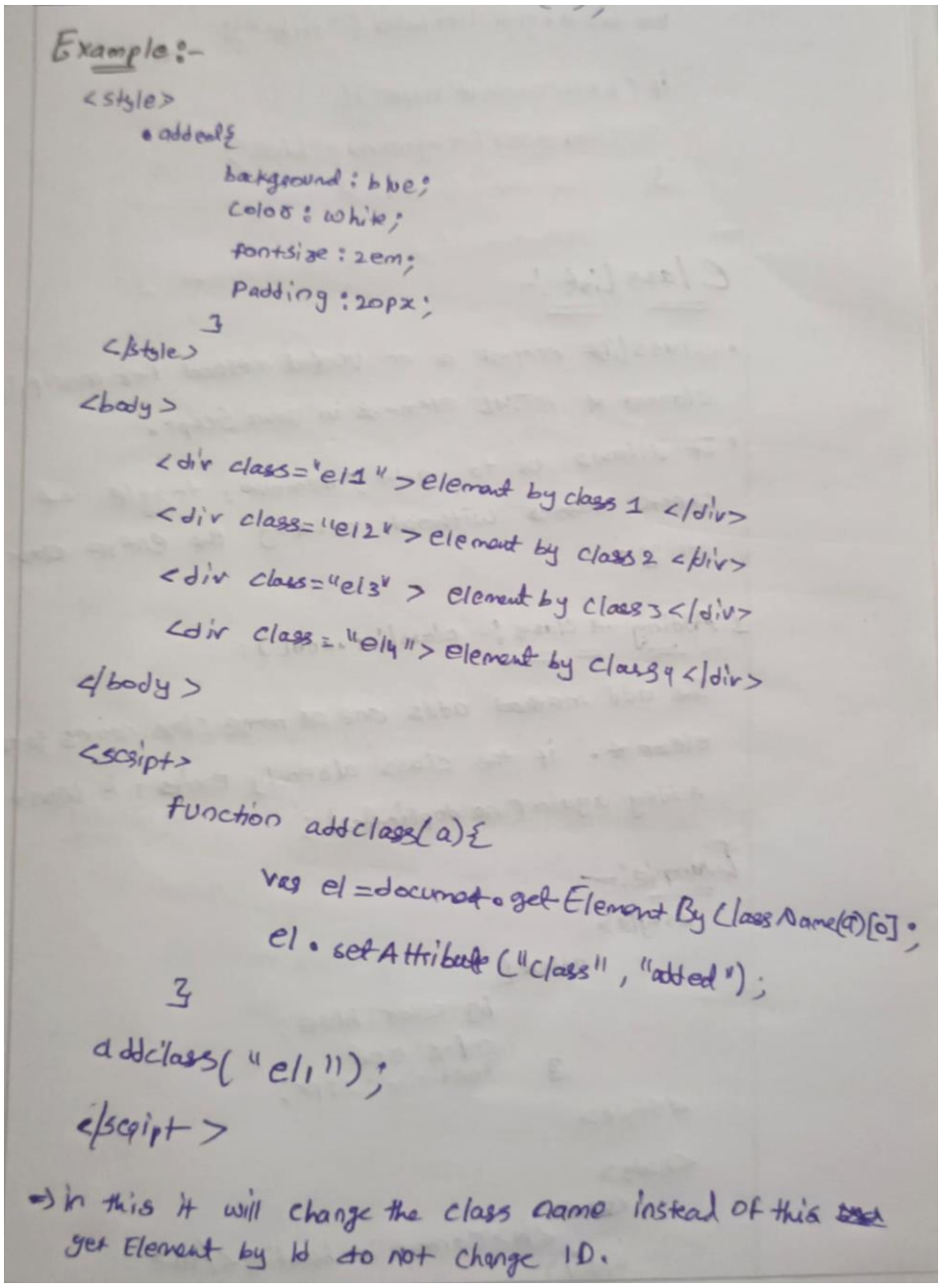
```
var a=document.createElement("img");
       img.setAttribute("src","a1.jpge");
       document.body.appendChild(a);
```

Example-3: How to set class name by using setAtribute and use css of class name

```
<style>
      .adder{
         background-color:blue;
         color: white;
         font-size:2em;
         padding:20px;
      }
   </style>
</head>
<body>
   <script>
      var a=document.createElement("div");
      a.innerHTML="Content Added";
      a.setAttribute("class","adder");
      document.body.appendChild(a);
   </script>
```



Example-4:



How to get attribute

We can get the element attribute by using get attribute method in dom

Example:

```
<div class="el1" title="ct">Content Added</div>
<script>
   var a=document.getElementsByClassName("el1")[0];
   var b=a.getAttribute("title");
   if(b=="ct"){
      a.style.background="blue";
   }
</script>
```

Classlist add and remove

- The classList property is an incredibly useful method for manipulating the classes of HTML elements in JavaScript.
- It allows you to add, remove, toggle, and check classes without altering the entire className string. Here's an in-depth explanation of how classList.add() and classList.remove() work.

1. **What is classList?**

• classList is a property that returns a live DOMTokenList collection of the classes of an element.

• You can think of classList as a way to interact with the classes applied to an HTML element, allowing you to add, remove, or toggle CSS classes dynamically.

2. Syntax: The syntax for accessing classList is as follows:

element.classList
classList methods include:
.add(), .remove(), .toggle()

3. **Adding a Class: classList.add()**

The add() method adds one or more class names to the element. If the class already exists, it won't be added again (no duplicates).

Syntax: element.classList.add(className1, className2, ..., classNameN);

Example:

```
<style>
      .added{
         background-color:blue;
         color: white;
         font-size:2em;
         padding:20px;
      }
   </style>
</head>
<body>
   <div>
      <h1>Content in heading</h1>
   </div>
   <button onclick="fun()">click here</button>
   <script>
      function fun(){
         var div=document.querySelector("div");
         div.classList.add("added");
      }
   </script>
</body>
```

4. **Removing a Class: classList.remove()**

The remove() method removes one or more class names from the element. If the class does not exist, nothing happens.

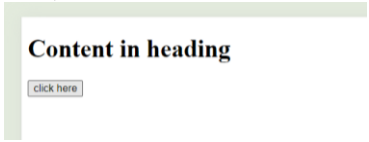
Syntax: element.classList.remove(className1, className2, ..., classNameN);

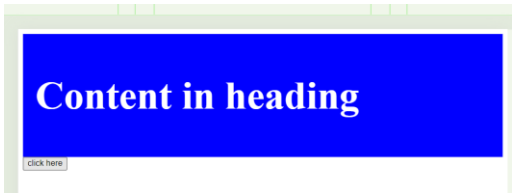
Example-1:

```
function fun(){
        var div=document.querySelector("div");
        div.classList.remove("added");
    }
```

Example-2:

```
var count=true;
    function fun(){
      var div=document.querySelector("div");
      if(count){
        div.classList.add("added");
        count=false;
      }
      else{
        div.classList.remove("added");
        count=true;
      }
    }
```





5. **Common Use Cases**

A. Toggling Classes (With classList.toggle())

• Sometimes, you may want to add a class if it's not present or remove it if it is. This can be done with the toggle() method.

Syntax: element.classList.toggle('className');

Example:

```
<div>
    <h1>Content in heading</h1>
  </div>
  <button onclick="fun()">click here</button>
  <script>
    var count=true;
    function fun(){
      var div=document.querySelector("div");
      div.classList.toggle("added");
    }
```

B. Checking If an Element Has a Class (classList.contains())

• To check if an element has a certain class, use the contains() method.

Syntax: if (element.classList.contains('className')) { // do something }

Types of Events

Mouse Events:

click: Occurs when a mouse button is clicked.

dblclick: Occurs when a mouse button is double-clicked.

mouseover: Occurs when the mouse pointer enters the area of an element.

mouseout: Occurs when the mouse pointer leaves the area of an element.

mousemove: Occurs when the mouse pointer is moved over an element.

Keyboard Events:

keydown: Occurs when a keyboard key is pressed down.

Keyup: Occurs when a keyboard key is released.

keypress: Occurs when a keyboard key is pressed and released.

Form Events:

submit: Occurs when a form is submitted.

change: Occurs when the value of an input element changes.

focus: Occurs when an element receives focus.

blur: Occurs when an element loses focus.

Window Events:

load: Occurs when a resource and its dependent resources have finished loading.

resize: Occurs when the browser window is resized.

scroll: Occurs when the user scrolls through a webpage.

Event handlers:

Event handlers are functions in JavaScript that are responsible for handling specific types of events. They define what should happen when a particular event occurs. Event handlers are associated with HTML elements and are triggered when the corresponding event takes place.

1.Inline Event Handlers: Inline event handlers are defined directly within the HTML markup using the on attribute followed by the event name.

<button onclick="myFunction()">Click me</button>

Example:

```
<div>
      <h1>Content in heading</h1>
   </div>
   <button onclick="fun()">click here</button>
   <script>
      var count=true;
      function fun(){
         var div=document.querySelector("div");
         div.classList.toggle("added")
      }
   </script>
```

2.DOM Event Handlers: DOM event handlers are assigned to HTML elements using JavaScript code.

You can attach event handlers using methods like addEventListener()

const button = document.getElementById('myButton');
button.addEventListener('click', myFunction);

Event listeners:

Event listeners in JavaScript are functions that wait for a specific event to occur and then execute code in response to that event.

Using addEventListener() Method: The addEventListener() method attaches an event listener to an HTML element.

It takes three parameters: the event name, the function to be executed when the event occurs, and an optional boolean value indicating whether to use capturing or bubbling (default is false, indicating bubbling).

```
const button = document.getElementById('myButton');
button.addEventListener('click', function() {
    console.log('Button clicked!');
});
```

Removing Event Listeners: You can remove event listeners using the removeEventListener() method. It requires the same parameters as addEventListener().

```
function handleClick() {
    console.log('Button clicked!');
}
const button = document.getElementById('myButton');
button.addEventListener('click', handleClick);
// Later, if you want to remove the event listener
button.removeEventListener('click', handleClick);
```

==========

Mouse Events Examples:

1. click: Occurs when a mouse button is clicked.

```
<div>
      <h1>Content in heading</h1>
   </div>
   <button>click here</button>
   <script>
      var btn=document.querySelector("button");
      var div=document.querySelector("div");
      btn.addEventListener("click",fun);
            function fun(){
```

```
        var div=document.querySelector("div");
        div.style.color="red";
        div.style.background="blue";
    }
</script>
```

2. **dblclick:** Occurs when a mouse button is double-clicked.

```
<div>
    <h1>Content in heading</h1>
</div>
<button>click here</button>
<script>
    var btn=document.querySelector("button");
    var div=document.querySelector("div");
    btn.addEventListener("dblclick",fun2);

// dblclick
    function fun2(){
        var div=document.querySelector("div");
        div.style.color="white";
        div.style.background="blue";
    }
</script>
```

3. **mouseover:** Occurs when the mouse pointer enters the area of an element.

```
<div>
        <h1>Content in heading</h1>
</div>
<button>click here</button>
<script>
    var btn=document.querySelector("button");
    var div=document.querySelector("div");
    div.addEventListener("mouseover",fun3);

    // mouseover
    function fun3(){
        var div=document.querySelector("div");
        div.style.color="white";
        div.style.background="red";
    }
<script>
```

4. mousemove: Occurs when the mouse pointer is moved over an element.

```
<div>
    <h1>Content in heading</h1>
</div>
<button>click here</button>
<script>
    var btn=document.querySelector("button");
    var div=document.querySelector("div");
    div.addEventListener("mousemove",fun4);

    //mousemove
    function fun4(){
        var div=document.querySelector("div");
        div.style.color="red";
        div.style.background="green";
    }
</script>
```

5. mouseout: Occurs when the mouse pointer leaves the area of an element.

```
<div>
    <h1>Content in heading</h1>
</div>
<button>click here</button>
<script>
```

```
    var btn=document.querySelector("button");
    var div=document.querySelector("div");
    div.addEventListener("mouseout",fun5);
    // mouseout
    function fun5(){
        var div=document.querySelector("div");
        div.style.color="red";
        div.style.background="grey";
    }
</script>
```

Keyboard Events:

1. keydown: Occurs when a keyboard key is pressed down.

```
<script>
    document.addEventListener("keydown",function(event){
        console.log(event.key);
    })
</script>
```

2. Keyup: Occurs when a keyboard key is released.

```
<script>
    document.addEventListener("keyup",function(event){
        console.log(event.key);
    })
</script>
```

3. keypress: Occurs when a keyboard key is pressed and released.

```
<script>
    document.addEventListener("keypress",function(event){
        console.log(event.key);
    })
</script>
```

Tasks:

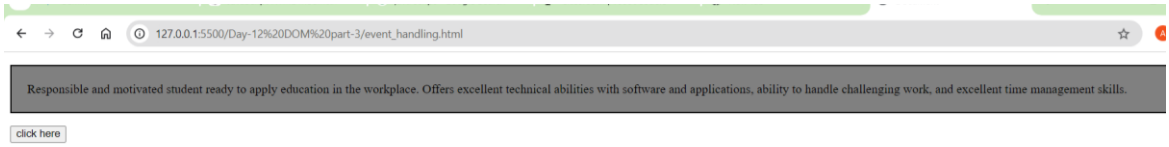
1. **Modify element attributes: Select an image element and change its src and alt attributes.**
   Code:

```
<img id="js" src="https://training.epam.com/static/news/496/JavaScriptforatestautomationengineerfeaturesadvantagestools_014111.png"
alt="myimg" height="350px" width="350px">
    <script>
        var img=document.getElementById("js");
        img.src="https://miro.medium.com/v2/resize:fit:800/1*PEzOBf4AkvDoE4VME4kq4Q.jpeg";
        img.alt="new js img";
        console.log(img);
    </script>
```

2. **Event handling:** Write a script that adds a click event listener to a button that changes the text of a paragraph when clicked.
   Code:

```
    <p>Event handling: Write a script that adds a click event listener
        to a button that changes the text of a paragraph when clicked.
    </p>
    <button>click here</button>

    <script>
        var btn=document.querySelector("button");
        btn.addEventListener("click",fun);
    function fun(){
            var p=document.querySelector("p");
            p.innerText="Responsible and motivated student ready to apply education in the workplace. Offers excellent technical abilities with
software and applications, ability to handle challenging work, and excellent time management skills.";
            p.style.background="grey";
            p.style.padding="20px";
            p.style.border="2px solid black";
```

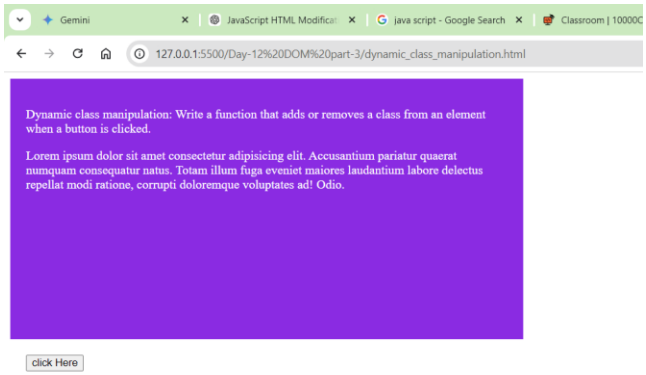
```
    }
  </script>
```

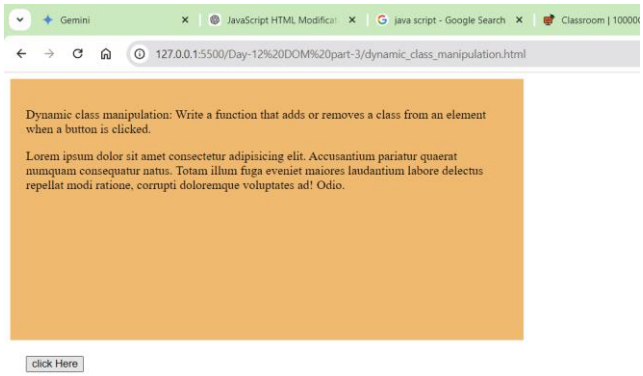


3. Dynamic class manipulation: Write a function that adds or removes a class from an element when a button is clicked.

Code:

```
<style>
    .div11{
      background-color:blueviolet;
      color: white;
      padding:20px;
      height:40vh;
      width: 40vw;
    }
    .adder{
      background-color: rgb(240, 185, 112);
      color: black;
      padding:20px;
      height:40vh;
      width: 40vw;
    }
    button{
      margin: 20px;
    }
  </style>
</head>
<body>
  <!--
    Dynamic class manipulation: Write a function that adds or removes
    a class from an element when a button is clicked.
  -->
  <div id="div11" class="div11">
    <p>Dynamic class manipulation: Write a function that adds or removes
      a class from an element when a button is clicked.
    </p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium
      pariatur quaerat numquam consequatur natus. Totam illum fuga eveniet
      maiores laudantium labore delectus repellat modi ratione, corrupti doloremque voluptates ad! Odio.
    </p>
  </div>
  <button onclick="fun()">click Here</button>
  <script>
    function fun(){
      var div=document.getElementById("div11");
      div.classList.toggle('adder');
    }
  </script>
</body>
```

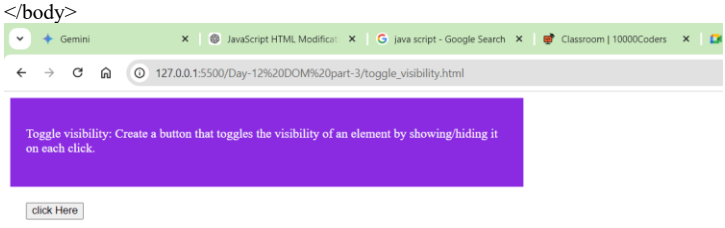


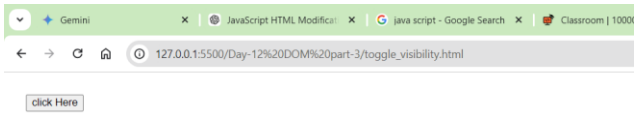


4. Toggle visibility: Create a button that toggles the visibility of an element by showing/hiding it on each click.

Code:

```
<style>
        .div1{
            background-color:blueviolet;
            color: white;
            padding:20px;
            height:10vh;
            width: 40vw;
        }
        .adder{
        display: none;
        }
        button{
            margin: 20px;
        }
    </style>
</head>
<body>
    <div id="div1" class="div1">
        <p>Toggle visibility: Create a button that toggles the visibility of
            an element by showing/hiding it on each click.
        </p>
    </div>
    <button onclick="fun()">click Here</button>
    <script>
        function fun(){
            var div=document.getElementById("div1");
            div.classList.toggle('adder');
        }
    </script>
</body>
```





5. Task:5 => by clicking the button display an image

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .div1{
            background-color:blueviolet;
            color: white;
            padding:20px;
            height:40vh;
            width: 40vw;
        }
        .html{
            background-image: url("https://media.licdn.com/dms/image/D4D12AQEaTK5_cv5Fmg/article-cover_image-
shrink_720_1280/0/1677082453584?e=2147483647&v=beta&t=hy4wcIl2vOpp5lr5k74obYVN63IT-U4lydNOIo3hHGo");
            background-repeat: no-repeat;
            background-size: cover;
            color: white;
            padding:20px;
            height:40vh;
            width: 40vw;
        }
        .css{
            background-image: url("https://ksra.eu/wp-content/uploads/2021/05/Vp9WvV7YKdH4k8sKRePcE8-1200-80.jpeg");
            background-repeat: no-repeat;
            background-size: cover;
            color: white;
            padding:20px;
            height:40vh;
            width: 40vw;
        }
        .boot{
            background-image: url("https://www.drupal.org/files/project-images/b5-new-logo.png");
            background-repeat: no-repeat;
            background-size: cover;
            color: white;
            padding:20px;
            height:40vh;
            width: 40vw;
        }
        .js{
            background-image: url("https://cyberhoot.com/wp-content/uploads/2020/07/Free-Courses-to-learn-JavaScript-1024x576.jpg");
            background-repeat: no-repeat;
            background-size: cover;
            color: white;
            padding:20px;
            height:40vh;
            width: 40vw;
        }
        button{
            margin: 20px;
            margin-left:60px;
        }
    </style>
</head>
<body>
    <div id="div1" class="div1">

    </div>
    <button onclick="html()">HTML</button>
    <button onclick="css()">CSS</button>
    <button onclick="boot()">Bootstrap</button>
    <button onclick="js()">Java Script</button>

    <script>
```

```
        function html(){
            var div=document.getElementById("div1");
            div.classList.toggle('html');
        }

        function css(){
            var div=document.getElementById("div1");
            div.classList.toggle('css');
        }

        function boot(){
            var div=document.getElementById("div1");
            div.classList.toggle('boot');
        }

        function js(){
            var div=document.getElementById("div1");
            div.classList.toggle('js');
        }
    </script>

</body>
</html>
```

Topic: Dom forms (Form Handling)

1. Accessing Form Elements:

```
<form id="myForm">
    <input type="text" id="username">
    <input type="email" id="email">
    <button type="submit">Submit</button>
</form>

<script>
    const form = document.getElementById('myForm');
    const usernameInput = document.getElementById('username');
    const emailInput = document.getElementById('email');
</script>
```

2. Setting Input Values:

You can set input values using the value property:

```
 usernameInput.value = "John Doe";
emailInput.value = "john@example.com";
```

3. Getting Form Data:

You can get form data when the form is submitted:

```
 form.addEventListener('submit', function(event) {
event.preventDefault(); // Prevent form submission
const formData = new FormData(form);
// Access form data
const username = formData.get('username');
const email = formData.get('email');
// Do something with the data
});
```

4. Changing Input Field Values

You can change the value of input fields dynamically based on user actions or other events:

```
<input type="text" id="myInput">
<button onclick="changeInputValue()">Change Value</button>

<script>
    function changeInputValue() {
        const input = document.getElementById('myInput');
        input.value = 'New Value';
    }
</script>
```

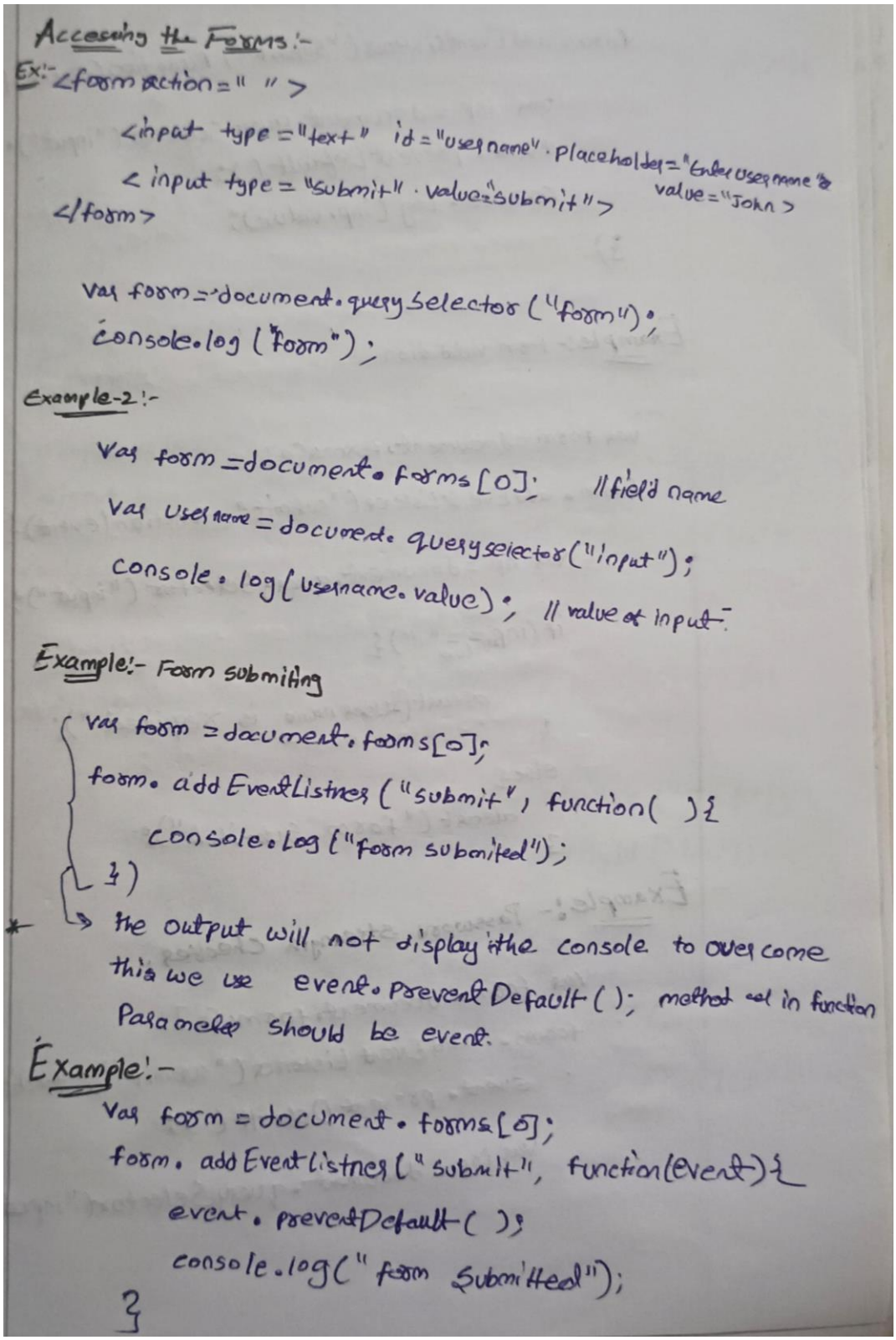
5.Accessing through values

```
document.forms[0].addEventListener('submit', function(event){
event.preventDefault(); // Prevent form submission
console.log(event);//stores the lot of form info including child tags of form
console.log(event.target[0].value);//prints the value of first child in the first form
})
```

In above program document.forms[0] get the first form,
e.preventdefault prevents the form default submission

console.log(event) prints a lot of form data and event.target gets the childrens of the form in collection format.
Examples:

https://github.com/Abhinavsai-12/10kCoders_JavaScript/tree/main/Day-13%20DOM%20part-4/practice



Example :- Save i/p which is given by User

```
<input type="text" placeholder="Enter Username" id="usn">
var form = document.forms[0];

form.addEventListener("submit", function(event){

    var inp = document.querySelector("input");
    event.preventDefault();
    console.log(inp.value);
})
```

this will not submit the value.

Example :- Form validation.

```
var form = document.forms[0];
Form.addEventListener("submit", function(event){

    var inp = document.querySelector("input");

    if(inp.value == ""){

        alert("User name is required");
    }
    else{
        alert("Form Submited");
    }
```

Example :- Password strength checker.

```
var form = document.forms[0];
forms.addEventListener("submit", function(event){
    event.preventDefault();

    var inp = document.querySelector("input");
```

```
if ( inp.value.length >10){
        Console.log("Password is strong");
}
else if ( inp.value.length <=10 && inp.value.length >5){
    console.log("Password is Medium");
}
else{
    console.log(" Password is week");
}
})
```

Example:- Password Matches

```
<input type="password" placce holder="Enter Pass" id="ip1">
<input type = "password" Place holder = "Confirm pass" id ="ip2">
<script>
  var form = document.forms[0];
  forms.addEventListener("submit", function(event){
      event.preventDefault();
      var ip1 = document.querySelector("input");
      var ip2 = document.getElementById("ip2");
      if ( ip1.value == ip2.value){
          console.log ("both are Matched");
      }
      else{
          console.log ("both are not Matched");
      }
})
```

Example :- display msg if passwords not matched

```
<input type = password >

<input type = password >
<span> password not match </span>
<submit > click here </submit >

<script>
    var form = document.forms[0];
    form.addEventListener("submit", function (event){
        event.preventDefault ();
    var inp1= document.querySelector ("input");
    var inp2 = document.querySelectorAll ("input")[1];
    if [ inp1.value == inp2.value){
    }     console.log ("both are not matched");
    else {
        console.log ("both are not matched");
        var span = document.querySelector ("span");
        span.style.display = "block";
    }
```

css:-

```
span{
    color: red;
    display: none;
}
```

Example :- if text field is empty display It is empty text

```
span{
    display: none;
    color : red;
}
```

```
<form action=" ">
  <input type='text', placeholder="Enter your Name ">
  <submit> Submit </Submit>
  <span> Enter your name </span>
</form>

<script>
    var form = document.form[0];
    form.addEventListener("submit", function(event){
      event.preventDefault();

    var ip = document.querySelector("input");
    (or)
    var ip = form[0].value;
    if (ip.value == "") {
          console.log("Enter your Name");
          var span = document.querySelector("span");
          span.style.display = "block";
    }
    else {var span = document.querySelector("span");
      console.log("Name is Submited");
      span.style.display = "none";
    }
```

event.target :-

```
    var form = document.forms[0];
    form.addEventListener("submit", function(event){
        event.preventDefault();
        console.log(event.target[1]); // consoles the 1st element in forms
    });
```

=) for value :-

```
    event.prev
    console.log(event.target[0].value);
    console.log(event); // consoles the forms and forms elements.
```

Example:-

```
<input type="text" id="inp1" placeholder="enter a value">
<input type="text" id="inp2" placeholder="enter a value">
<div class="output"> = value </div>
<button id="add"> add </button>

<script>
    var btn = document.getElementById("add");
    btn.addEventListener("click", function(){
        var inp1=document.getElementById("inp1");
        var inp2 = document.getElementById("inp2");
        var value= Number(inp1.value) + Number(inp1.value);
        var div = document.getElementsByClassName("output")[0];
        div.innerHTML= " = " + value;
        inp1.value = "";
        inp2.value = "";
});
```

3)

TASKS:

1) Password Strength Checker:

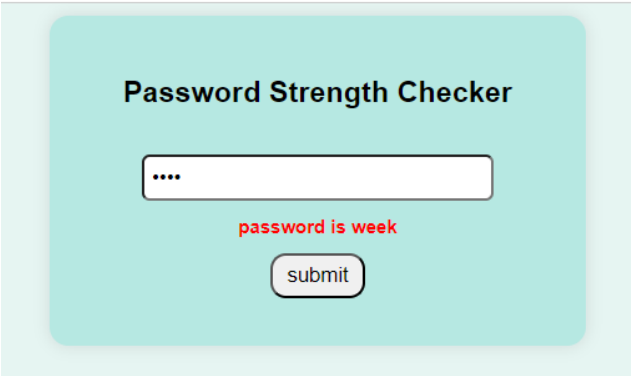
Create an input field where a user types a password, and the script checks if the password is "Strong", "Medium", or "Weak" based on its length.

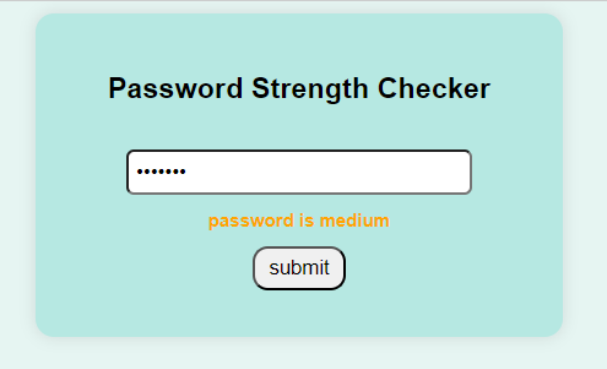
Code:

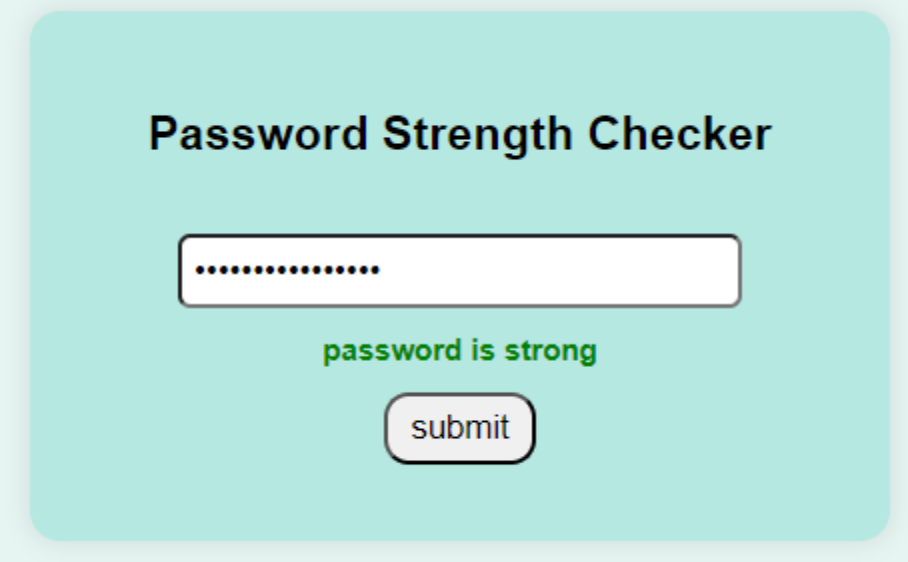
```
  <style>
    body{
      font-family: Arial, sans-serif;
      display: flex;
      justify-content: center;
      align-items: center;
      background-color: #c6e9e36f;
    }
    .container {
      background-color: #6cd5c965;
      padding: 20px;
      border-radius: 12px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      width: 20%;
    }
    #fm{
      display: flex;
      flex-direction: column;
      justify-items:center;
      align-items: center;
    }
    input{
      width: 70%;
      padding: 5px;
      margin: 10px;
      border-radius:5px;
    }
    #sub{
      width: 20%;
      padding: 5px;
      margin: 10px;
      border-radius:10px;
    }
    span{
      font-size: 12px;
      font-weight:bold;
      display: none;
    }
    </style>

</head>
<body>
  <div class="container">
    <form action="" id="fm">
      <h3>Password Strength Checker</h3>
      <input type="password" id="pass" placeholder="Enter password">
      <span></span>
      <input type="submit" id="sub" value="submit">
    </form>
  </div>

  <script>
    var form=document.forms[0];
```

```
    form.addEventListener("submit",function(event){
        event.preventDefault();
        var inp=document.getElementById("pass");
        var span=document.querySelector("span");
        var span=document.getElementsByTagName("span")[0];

        if(inp.value.length>10){
            span.innerHTML="password is strong";
            span.style.display="block";
            span.style.color="green";
        }
        else if(inp.value.length<=10 && inp.value.length>5){
            span.innerHTML="password is medium";
            span.style.display="block";
            span.style.color="orange";
        }
        else{
            span.innerHTML="password is week";
            span.style.display="block";
            span.style.color="red";
        }
    })
</script>
```

Output







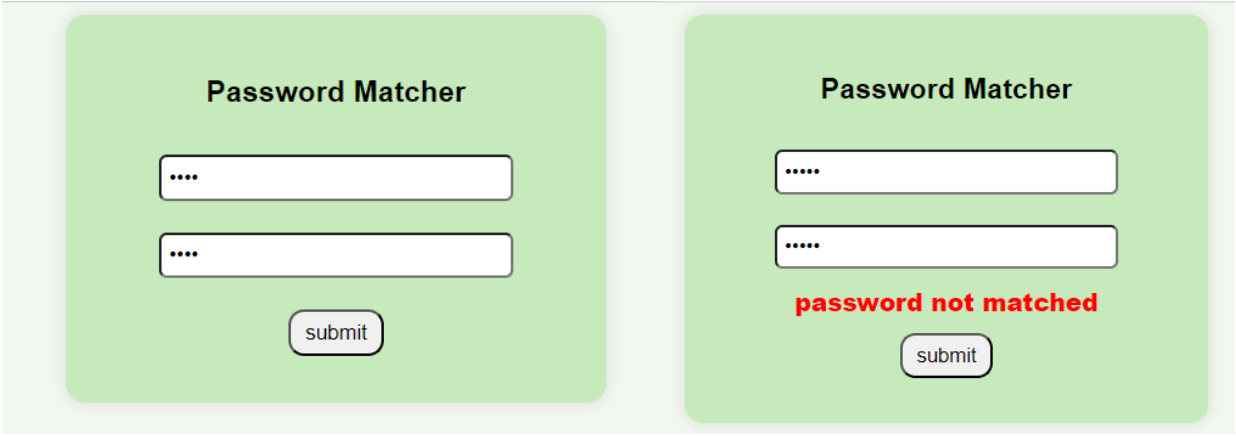
2) **password matcher**

Code:

```
<style>
      body{
         font-family: Arial, sans-serif;
         display: flex;
         justify-content: center;
         align-items: center;
         background-color: #e3f1dd6f;
      }
      .container {
         background-color: #83d56c65;
         padding: 20px;
         border-radius: 12px;
         box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
         width: 20%;
      }
      #fm{
         display: flex;
         flex-direction: column;
         justify-items:center;
         align-items: center;
      }
      input{
         width: 70%;
         padding: 5px;
         margin: 10px;
         border-radius:5px;
      }
      #sub{
         width: 20%;
         padding: 5px;
         margin: 10px;
         border-radius:10px;
      }
      span{
         color: red;
         font-weight:900;
         display: none;
      }
      </style>
</head>
<body>
   <div class="container">
      <form action="" id="fm">
         <h3>Password Matcher</h3>
         <input type="password" id="pass1" placeholder="Enter password">
         <input type="password" id="pass2" placeholder="Confirm password">
         <span>password not matched</span>
         <input type="submit" id="sub" value="submit">
      </form>
   </div>

   <script>
      var form=document.forms[0];
      form.addEventListener("submit",function(event){
         event.preventDefault();
         var pass1=document.getElementById("pass1");
         var pass2=document.getElementById("pass2");

         if(pass1.value==pass2.value){
            console.log("password is matched");
         }
         else{
            var span=document.getElementsByTagName("span")[0];
            span.style.display="block";
            console.log("password not matched");
         }
```

```
    })
  </script>
</body>
```

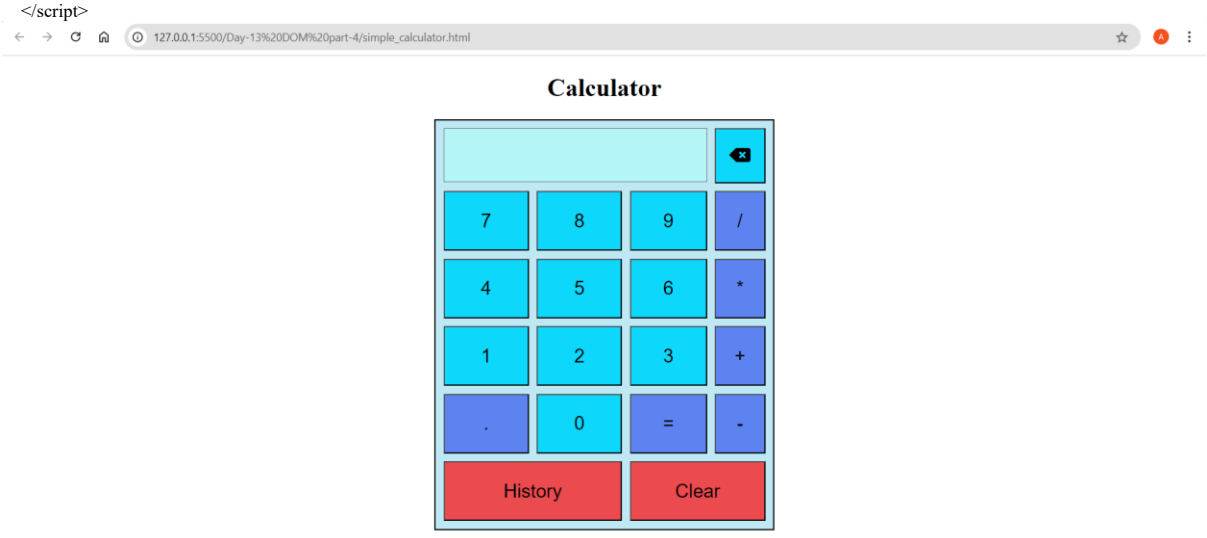
Output



3) simple calculator

```
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
  <style>
    h1{
      text-align: center;
    }
    .container{
      width: 27%;
      height: 500px;
      margin: auto;
      border: 2px solid black;
      display: grid;
      grid-template-columns: auto auto auto auto;
      grid-template-rows: 70px;
      gap: 10px;
      padding: 10px;
      background-color: rgba(144, 215, 238, 0.571);
    }
    #result{
      grid-column: 1/4;
      height: 55px;
      text-align: right;
      padding-top: 10px;
      font-size: 24px;
      font-weight: 800;
      resize: none;
      scrollbar-width: none;
      background-color: rgb(180, 245, 248);
    }
    .button{
      font-size: 24px;
      font-weight: 500;
      background-color:rgba(4, 214, 251, 0.951) ;
    }
    .op{
      background-color: rgb(92, 131, 240);
    }
    .history{
      grid-column: 1/3;
      background-color: rgba(255, 0, 0, 0.682);
    }
    .clear{
      grid-column: 3/5;
      background-color: rgba(255, 0, 0, 0.682);
    }

  </style>
</head>
<body>
  <h1>Calculator</h1>
  <div class="container">
    <textarea id="result"></textarea>
    <button onclick="back()" class="button backspace" ><i class="fa-solid fa-delete-left"></i></button>

    <input type="button" onclick="dis('7')" value="7" class="button">
    <input type="button" onclick="dis('8')" value="8" class="button">
    <input type="button" onclick="dis('9')" value="9" class="button">
```

```
    <input type="button" onclick="dis('/')" value="/" class="button op">

    <input type="button" onclick="dis('4')" value="4" class="button">
    <input type="button" onclick="dis('5')" value="5" class="button">
    <input type="button" onclick="dis('6')" value="6" class="button">
    <input type="button" onclick="dis('*')" value="*" class="button op">

    <input type="button" onclick="dis('1')" value="1" class="button">
    <input type="button" onclick="dis('2')" value="2" class="button">
    <input type="button" onclick="dis('3')" value="3" class="button">
    <input type="button" onclick="dis('+')" value="+" class="button op">

    <input type="button" onclick="dis('.')" value="." class="button op">
    <input type="button" onclick="dis('0')" value="0" class="button">
    <input type="button" onclick="solve('')" value="=" class="button op">
    <input type="button" onclick="dis('-')" value="-" class="button op">

    <input type="button" onclick="hist()" value="History" class="button history">
    <input type="button" onclick="clr()" value="Clear" class="button clear">
</div>

<script>
    var his="";
    function dis(a){
        document.getElementById("result").value +=a;
        return a;
    }
    function back(){
        let x=document.getElementById("result").value;
        let y='';
        for(i=0;i<x.length-1;i++){
            y+=x[i];
        }
        document.getElementById("result").value=y;
    }
    function clr(){
        document.getElementById("result").value="";
    }
    function solve(){
        let x=document.getElementById("result").value;
        his +="->"+x+"\n";
        let y=eval(x);
        his +="="+y+"\n";
        document.getElementById("result").value=y;
        return y;
    }
    function hist(){
        document.getElementById("result").value=his;
    }
</script>
```



Topic: String methods

Strings

collection of words or characters enclosed in a single or double quotes
Strings are immutable and can't be changed directly. We can get the values of string but we cant change.

```
//iterate the string using for loop
 var a ="hello";
 for(i=0;i<5;i++){
   console.log(a[i]);
 }
```

String.length
It is a method used to find the length of a string
The .length property returns the number of characters in the string, including spaces, punctuation marks, and special characters.

```
let str = "Hello, world!";
console.log(str.length); // This will output 13
```

charAt()
The charAt() method is used to return the character at a specified index (position) within a string.

```
let str = "Hello";
console.log(str.charAt(0)); // Output: "H"
console.log(str.charAt(1)); // Output: "e"
console.log(str.charAt(4)); // Output: "o"
```

at()
The at() method allows you to directly access a character at a specific position within a string, similar to charAt() but also takes negative values.

```
let str = "Hello";
console.log(str.at(0)); // Output: "H"
console.log(str.at(1)); // Output: "e"
console.log(str.at(-1)); // Output: "o"
```

charCodeAt()
the charCodeAt() method returns the Unicode value (integer between 0 and 65535) of the character at a specified index in a string.

```
let str = "Hello";
console.log(str.charCodeAt(0)); // Output: 72
console.log(str.charCodeAt(1)); // Output: 101
console.log(str.charCodeAt(4)); // Output: 111
```

slice(start,end)
The slice() method is used to extract a section of a string and return it as a new string. It doesn't modify the original string. This method takes two parameters: the start index and the end index (optional).

```
let str = "Hello, world!";
console.log(str.slice(0, 5));    // Output: "Hello"
console.log(str.slice(7));     // Output: "world!"
console.log(str.slice(-6));    // Output: "world!"
console.log(str.slice(7, -1)); // Output: "world"
console.log(str.slice(0)); // Output: "Hello, world!"
console.log(str.slice(-1)); // Output: "!"
```

substring(start, end)
The substring() method is used to extract a portion of a string and return it as a new string. It is similar to the slice() method, but there are differences in how negative indices are handled because negative values are considered as zero

```
let str = "Hello, world!";
console.log(str.substring(0, 5)); // Output: "Hello"
console.log(str.substring(7)); // Output: "world!"
console.log(str.substring(7, 12)); // Output: "world"
console.log(str.substring(-6)); // Output: "Hello, world!"
console.log(str.substring(7, -1)); // Output: "Hello, world"
```

substr()
In JavaScript, the substr() method is used to extract a portion of a string, starting from a specified index and extending for a specified length of characters. This method is different from substring() in that the second parameter specifies the length of the extracted substring rather than the end index.

```
let str = "Hello, world!";
console.log(str.substr(0, 5)); // Output: "Hello"
console.log(str.substr(7)); // Output: "world!"
console.log(str.substr(7, 5)); // Output: "world"
console.log(str.substr(-6)); // Output: "world!"
console.log(str.substr(7, -1)); // Output: ""
```

toUpperCase()
The toUpperCase() method is used to convert all characters in a string to uppercase letters.

```
let str = "Hello, world!";
let a= str.toUpperCase();
console.log(a); // Output: "HELLO, WORLD!"
```

toLowerCase()
The toLowerCase() method is used to convert all characters in a string to lowercase letters.

```
let str = "Hello, WORLD!";
let a = str.toLowerCase();
console.log(a); // Output: "hello, world!"
```

concat()
The concat() method is used to concatenate two or more strings.

```
let str1 = "Hello";
```

```
let str2 = "world";
let str3 = "!";
let result = str1.concat(", ", str2, str3);
console.log(result); // Output: "Hello, world!"
```

trim()
The trim() method is used to remove whitespace from both ends of a string.

```
let str = "   Hello, world!   ";
let trimmedStr = str.trim();
console.log(trimmedStr); // Output: "Hello, world!"
```

repeat()
The repeat() method is used to construct and return a new string by concatenating the string on which it is called a certain number of times.

```
let str = "Hello";
let repeatedStr = str.repeat(3);
console.log(repeatedStr); // Output: "HelloHelloHello"
```

Split()
The split() method is used to split a string into an array.

```
let str = "Hello, world!";
let parts = str.split(", ");
console.log(parts); // Output: ["Hello", "world!"]

let characters = str.split("");
console.log(characters); // Output: ["H", "e", "l", "l", "o", ",", " ", "w", "o", "r", "l", "d", "!"]
```

replace()
replace() method used to replace the current occurrences of substring within a string with another string
syntax: string.replace(searchValue, replaceValue)

```
let originalString = "Hello, world!,world";
let newString = originalString.replace("world", "universe");
console.log(newString); // Output: Hello, universe!,world
```

Above program replaces only the first match to replace all matches use a regular expression with /g flagset
let newString = originalString.replace(/world/g, "red fox");

```
let originalString = "Hello, world!,world";
let newString = originalString.replace(/world/g, "universe");
console.log(newString); // Output: Hello, universe!,universe
```

Replace method is case sensitive writing World will not consider
To replace case insensitive, use a regular expression with an /i

```
let originalString = "Hello, world!, World";
let newString = originalString.replace(/world/ig, "universe");
console.log(newString); // Output: Hello, universe!,universe
```

replaceAll()
it is a method to replace a substring with another sting but it doesn't compatible in all browsers

string search methods

indexOf() and lastIndexOf

```
let str= "Hellow, world!";
let newString = str.indexOf("w")//output 5 because it checks from the starting
let newString1= str.lastIndexOf("w")//output 8 because it checks from the ending
```

Both indexOf(), and lastIndexOf() return -1 if the text is not found

```
let newString1= str.lastIndexOf("w",5)//output 8 because it checks from the ending
```

if the second parameter is 5, the search starts at position 5, and searches

search()
the search() method is used to search for a specified substring within a string. It returns the index of the first occurrence of the specified substring, or -1 if the substring is not found. It can take regular expressions also

```
let str = "Hello, world!";
let index = str.search("world");
console.log(index); // Output: 7
```

two methods, indexOf() and search(), are not equal because
search() method cannot take a second start position argument.

match()
The match() method in JavaScript is used to search a string for a specified pattern (regular expression), and returns an array containing the matches, or null if no matches are found.
It can take regular expression and it can print the values in an array

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/gi);    //4 [ain,AIN,ain,ain]
```

includes()
the includes() method returns true if a string contains a specified value.

```
let text = "Hello world, welcome to the universe.";
text.includes("world");//true
```

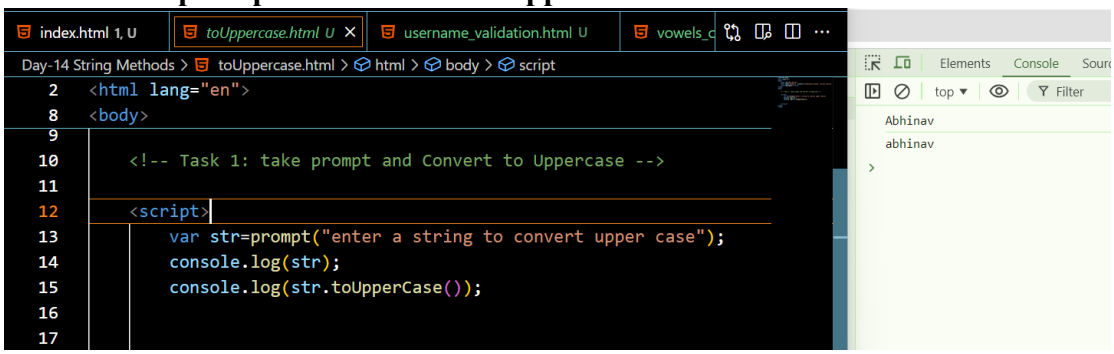
Template literals
Template literals allow you to embed expressions and variables directly within the string using ${}. This makes string interpolation more intuitive and readable.

```
let a = 5;
let b = 10;
```

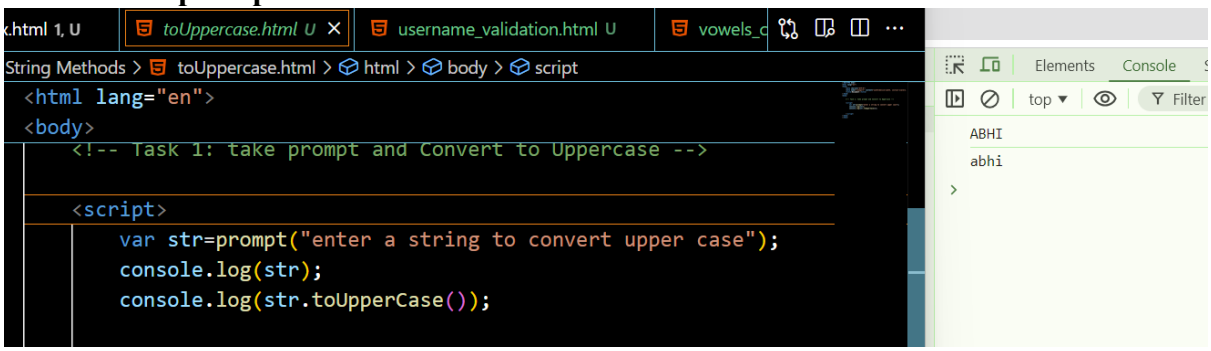
let result = `The sum of ${a} and ${b} is ${a + b}.`;
// result is "The sum of 5 and 10 is 15.

TASKS: Basic level

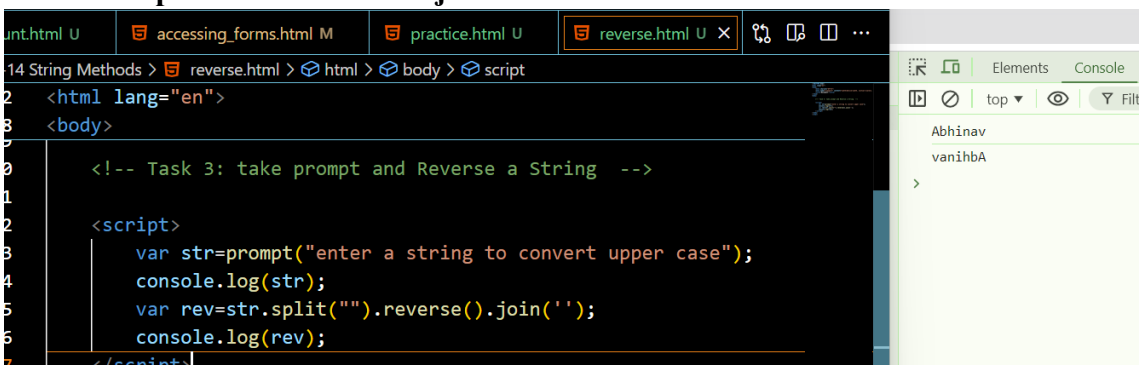
Task 1: take prompt and Convert to Uppercase



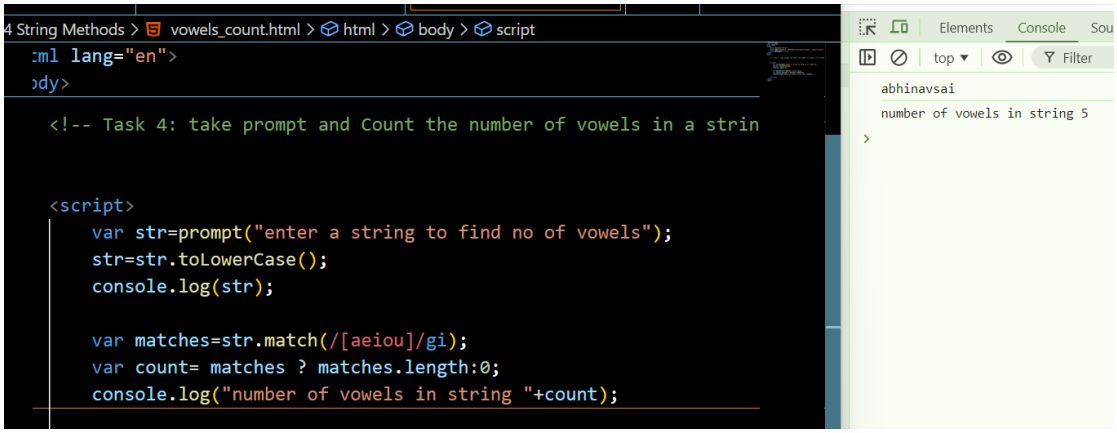
Task 2: take prompt and Convert to Lowercase



Task 3: take prompt and Reverse a String
hints: use split and reverse and join method



Task 4: take prompt and Count the number of vowels in a string--
hints use match method

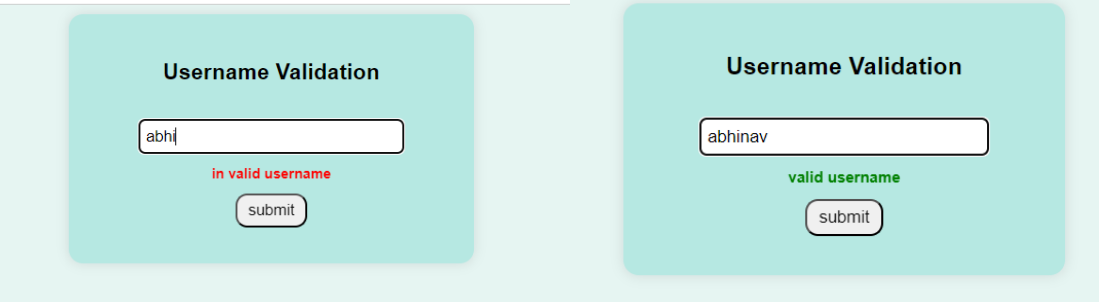


Advanced

Task 1: Username Validation

Scenario: Validate a username based on specific criteria. Task:
Prompt the user to enter a username.
Check if the username contains characters in between 5 to 15 long.
Display a message indicating whether the username is valid or not.

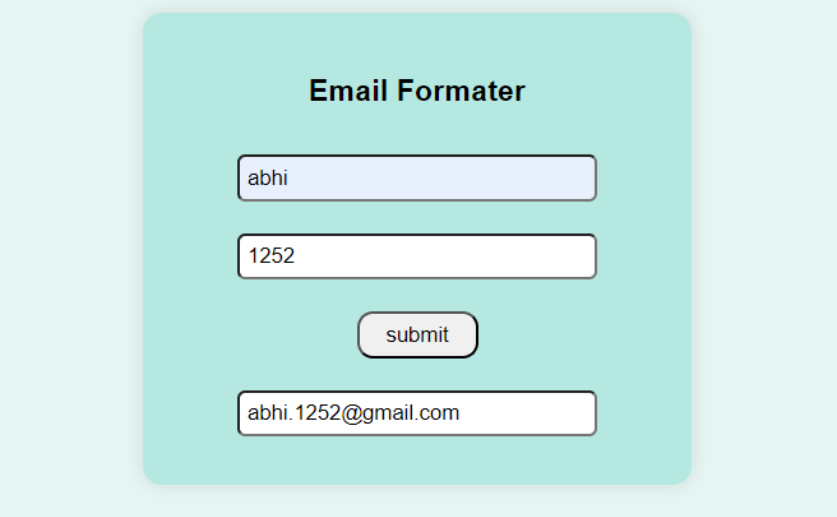
```
66
67      <script>
68          var form=document.forms[0];
69          form.addEventListener("submit",function(event){
70              event.preventDefault();
71
72              var ip=document.querySelector("input");
73              var span=document.querySelector("span");
74              if(ip.value.length>5 && ip.value.length<15){
75                  span.innerHTML="valid username";
76                  span.style.display="block";
77                  span.style.color="green";
78              }
79              else{
80                  span.innerHTML="in valid username";
81                  span.style.display="block";
82                  span.style.color="red";
83              }
84          })
85      </script>
```



Task 2: Email Formatter

Scenario: Format an email address. Task:
Prompt the user to enter their first and last name.
Convert the names to lowercase and concatenate them with a dot in between.
concat “@gmail.com” to the result and display the formatted email address.
inp: jenny and joy
out: jenny.joy@gmail.com

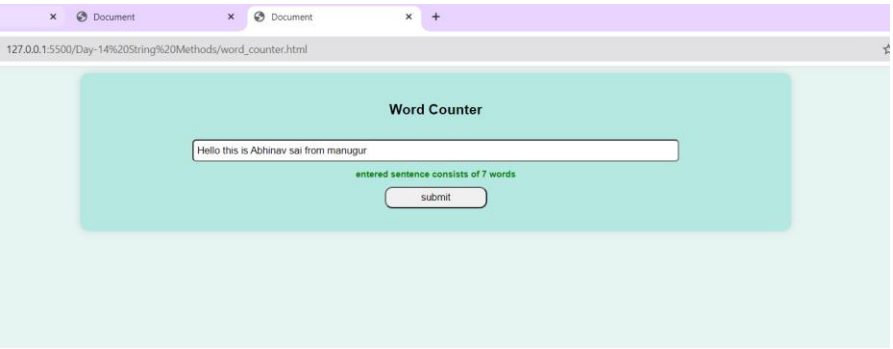
```
59      <div class="container">
60          <form action="" id="fm">
61              <h3>Email Formater</h3>
62              <input type="text" class="text" id="user" placeholder="Enter First name" >
63              <input type="text" class="text" id="user" placeholder="Enter Last name" >
64              <span></span>
65              <input type="submit" value="submit" id="sub">
66              <input type="text" class="text" id="user" placeholder="your genrated email will display he
67          </form>
68      </div>
69      <script>
70          var form=document.forms[0];
71          form.addEventListener("submit",function(event){
72              event.preventDefault();
73
74              var ip1=document.querySelectorAll("input")[0];
75              var ip2=document.querySelectorAll("input")[1];
76
77              res=`${ip1.value}.${ip2.value}@gmail.com`;
78              var ip3=document.querySelectorAll("input")[3];
79              ip3.setAttribute("value",res);
80          })
81      </script>
```



Task 3: Word Counter

Scenario: Count the number of words in a sentence. Task:
Prompt the user to enter a sentence.
Split the sentence into words and count them.
Display the word count.
Hint: use split and length method

```
<body>

    <div class="container">
        <form action="" id="fm">
            <h3>Word Counter</h3>
            <input type="text" class="text" id="user" placeholder="Enter a sentance to find no of wo
            <span></span>
            <input type="submit" value="submit" id="sub">
        </form>
    </div>
    <script>
        var form=document.forms[0];
        form.addEventListener("submit",function(){
         event.preventDefault();

         var ip=document.querySelector("input");
         var count=ip.value.split(' ').length;
         var span=document.querySelector("span");
         span.innerHTML=`entered sentence consists of ${count} words`;
         span.style.color='green';
         span.style.display='block';

        })
```



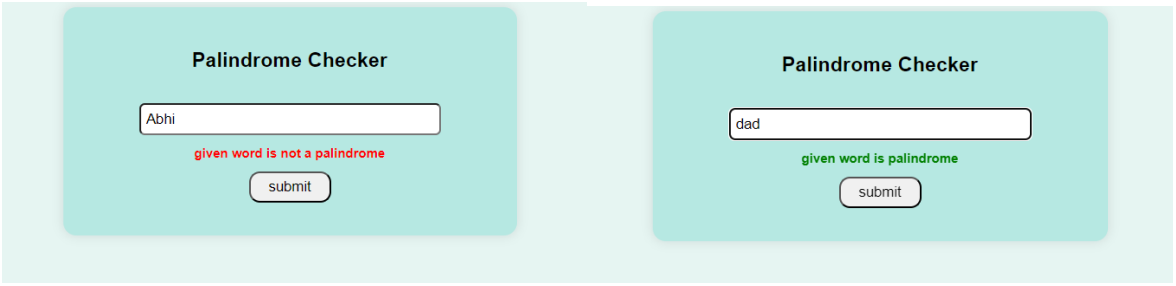
Task 4: Palindrome Checker

Scenario: Check if a given string is a palindrome. Task:
Prompt the user to enter a string.
Check if the string reads the same forwards and backwards.
Display a message indicating whether the string is a palindrome
Hints : use split,join, reverse method in arrays

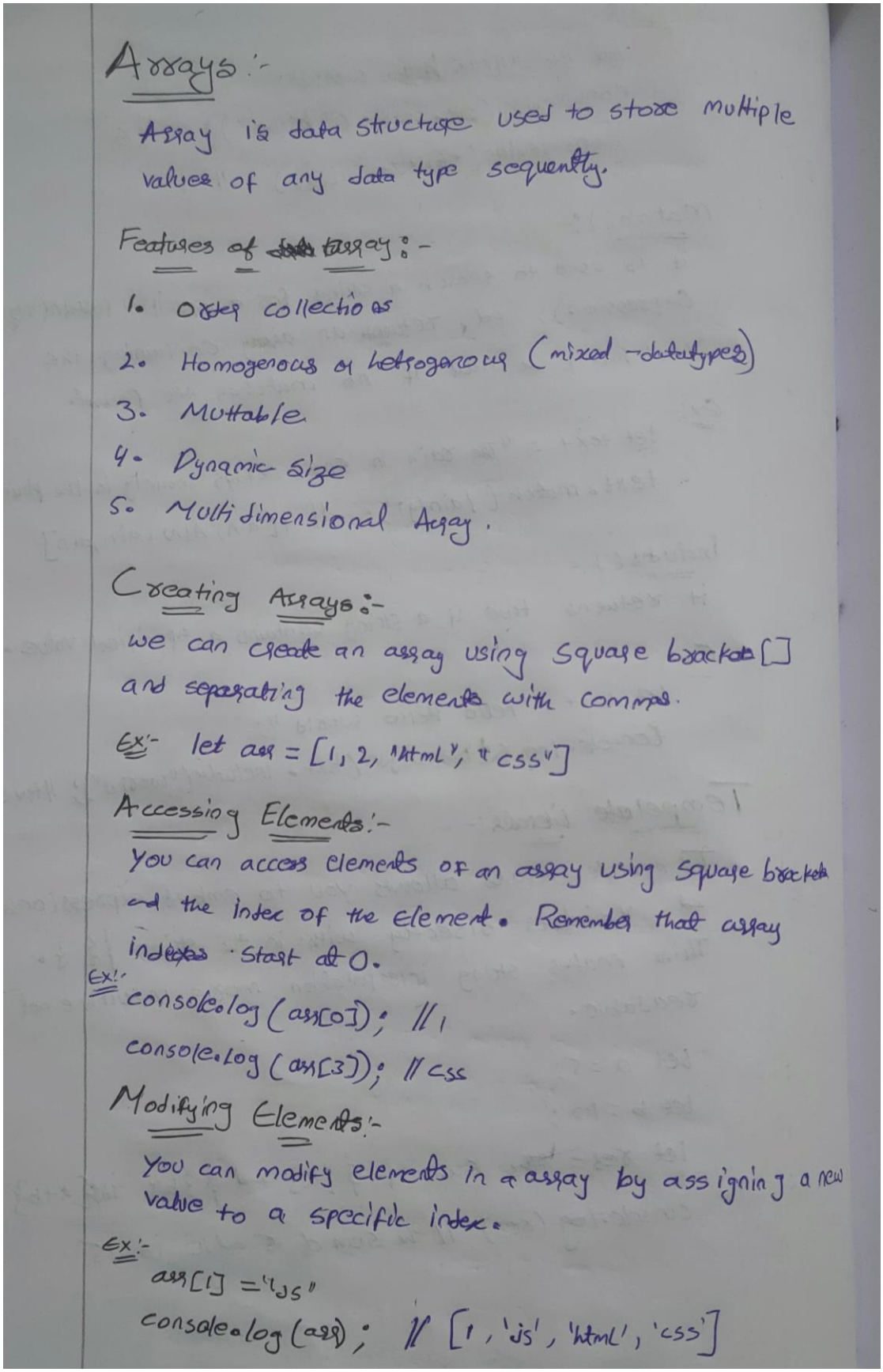
inp:john
out: not a palandrome

inp:dad
out:it is a palandrome

```
<script>
    var form=document.forms[0];
    form.addEventListener("submit",function(){
    event.preventDefault();

    var ip=document.querySelector("input");
    var rev=ip.value.split('').reverse().join('');
    var span=document.querySelector("span");

    if(ip.value == rev){
        span.innerHTML="given word is palindrome";
        span.style.color='green';
        span.style.display='block';
    }
    else{
        span.innerHTML="given word is not a palindrome";
        span.style.color='red';
        span.style.display='block';
    }
   })
</script>
```



Array Methods:



Arrays :-

Array is data structure used to store multiple values of any data type sequently.

Features of array :-

1. Ordered collections
2. Homogenous or Heterogenous (mixed - datatypes)
3. Muttable
4. Dynamic size
5. Multi dimensional Array.

Creating Arrays :-

we can create an array using square brackets [] and separating the elements with commas.

Ex:- let arr = [1, 2, "Html", "css"]

Accessing Elements :-

You can access elements of an array using square brackets and the index of the element. Remember that array indexes start at 0.

Ex:-

```
console.log(arr[0]); // 1
console.log(arr[3]); // css
```

Modifying Elements :-

You can modify elements in a array by assigning a new value to a specific index.

Ex:-

```
arr[1] = "js"
console.log(arr); // [1, 'js', 'html', 'css']
```

dynamic size :-

we can increase assay size dynamically

Ex:- var ass = ["hello", "world", 2];
ass[3] = 3;
console.log(ass); //["hello", "world", 2, 3]

Multidimensional Assay :-

Ex:- var ass = [1, 2, 3, [4, 5, [6, 7, 8]]];
console.log(ass[3][2][2]);

Assay Methods :-

Java script provides many built in methods to work with assays such as push pop, shift, unshift, slice, splice, concat, indexOf, includes, etc many more

1. Assay length :-

length property returns the number of elements in an Assay

Ex:-
let ass = [1, 2, "html", "css"]
console.log(ass.length); // 4

2. Assay at() :-

The at() method of Assay instance takes an integer value and returns the item at the index, allowing for positive and negative integers. Negative integers count back from the last item of an assay

* Print last index value of assay (length-1)

Print : console.log(ass[ass.length-1]);

Example :-

let ass = [1, 2, 3, 4];
console.log(ass.at(0)); // 1
console.log(ass.at(-1)); // 4

3. Concat():- Method is used to merge two ~~arrays~~ or more arrays. It does not modify the existing arrays but instead returns a new array containing the elements of orginal arrays concatenated together.

Ex:-

```
var arr1 = [1, 2, 3]
var arr2 = ['a', 'b', 'b']
console.log(arr1.concat(arr2)); //[1,2,3,'a','b','b']
```

• Loops in arrays:-

```
var a = [1, 2, 3, 4, 5, 6, 7, 8]
for (i=0; i<arr.length; i++){
    console.log(arr[i]);
}
```

```
for (i in arr) {
    console.log[i];
}
```

4. Array Splice():-

Splice() method is used to change the contents of an array by removing or replacing existing elements and/or adding new elements in place.

• It modifies the orginal array & return an array containing the removed elements.

Syntax:-

```
array.splice(startIndex, endIndex, add item1, add item2);
```

Example:- // adding at specific index.

```
var arr = ["html", "css", "JS", "react"]
arr.splice(1, 0, "bootstrap");
console.log(arr); //[html, bootstrap, css, JS, react]
```

```
// replacing at a specific index
ass.splice(1, 1, "bootstap");
console.log(ass); // [html, bootstap, JS, react]
// replacing all values at a time
ass.splice(0, 4, "Java", "Python", "NodeJs", "mySql");
console.log(ass); [Java, Python, NodeJs, mySql];
```

5. Assay Sliced:-

slice() method is used to extact section of an element assay and returns a new assay containing the extracted elements.

Syntax:- assay.slice(startIndex, endIndex);

Example:-

```
var ass = ["html", "css", "js", "react"]
var ass2 = ass.slice(0, 2);
console.log(ass2); // [html, css]

var ass2 = ass.slice(-1);
console.log(ass2); // [react]
```

6. Assay Pop():-

Pop method removes the last elemenat from an assay and returns the element.

7. Assay Push():-

Push method adds one ore more elements to the end of an assay and returns new length of assay.

Example:-

```
var ass = ['html', 'css', 'JS', 'react'];
ass.pop(); // ['html, css, JS];
ass.push("git");
console.log(ass); // [html, css, JS, git]
```

8. Array shift();

shift method() is used to remove the first element of an array and returns that element.

9. Array unshift();

unshift method adds one or more elements to the beginning of an array and return new length of an array.

Example:-

```
var arr = ["html", "css", "JS", "React"];
arr.shift(); // [ css, JS, React ]
arr.unshift("git"); // ["git", "html", "css", "JS", "React"]
```

10. Array Sort and reverse Methods:-

```
arr.sort();
arr.reverse();
```

Examples:-

```
var arr = [9, 3, 4, 67];
console.log(arr.sort());
```

Output:- 3, 4, 67, 9 [Lexographical order it takes only first letter of an element.]

reverse Example:-

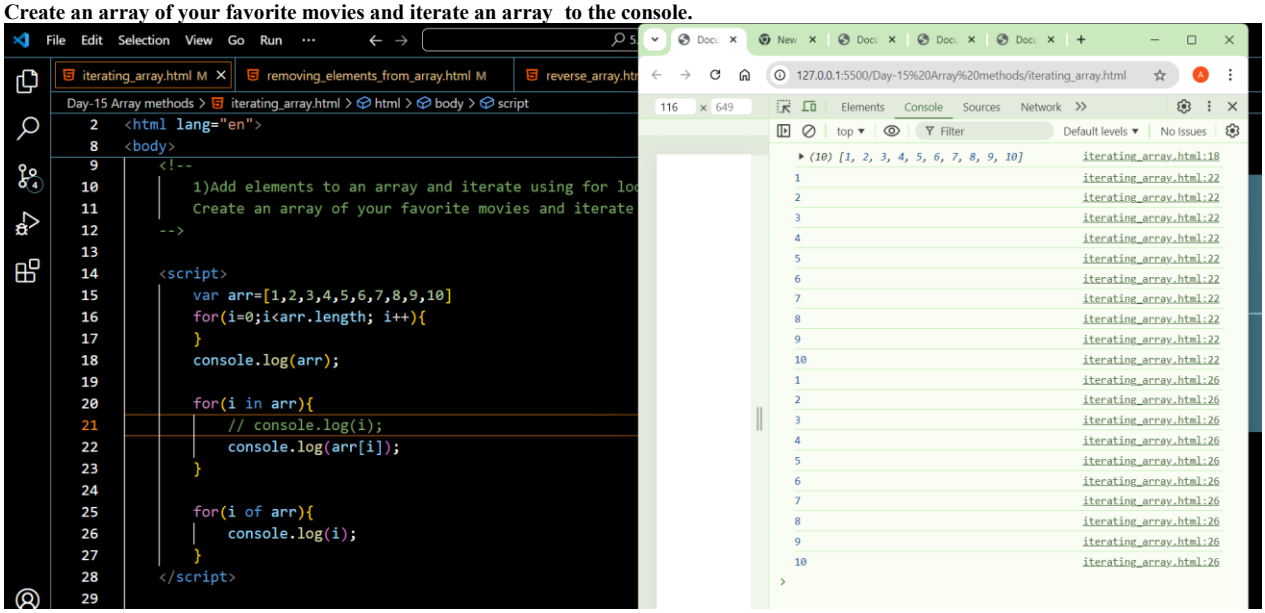
```
var arr = ["html", "css", "JS"];
console.log(arr.reverse()); // [JS, css, html]
```

Problem:- reverse an array and push into the new array

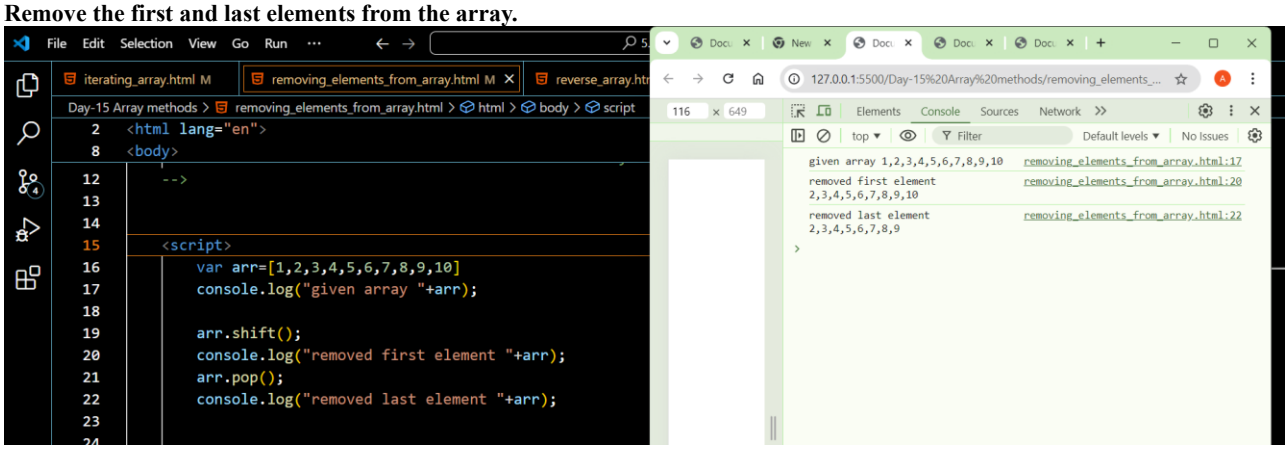
```
var a = ["html", "css", "JS", "React"]
var new = []
for( i=a.length-1; i>=0; i--){
        console.log(a[i]);
        new.push(a[i]);
}
console.log(new); // [React, JS, css, html]
```

TASKS:

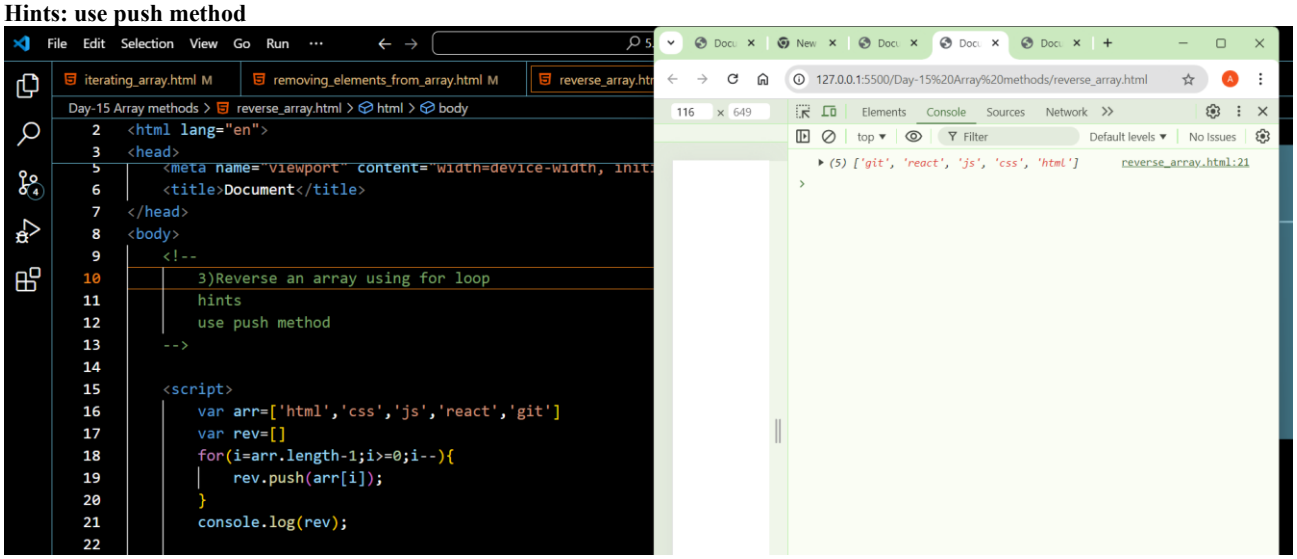
1)Add elements to an array and iterate using for loop and for in loop and for of loop
Create an array of your favorite movies and iterate an array to the console.



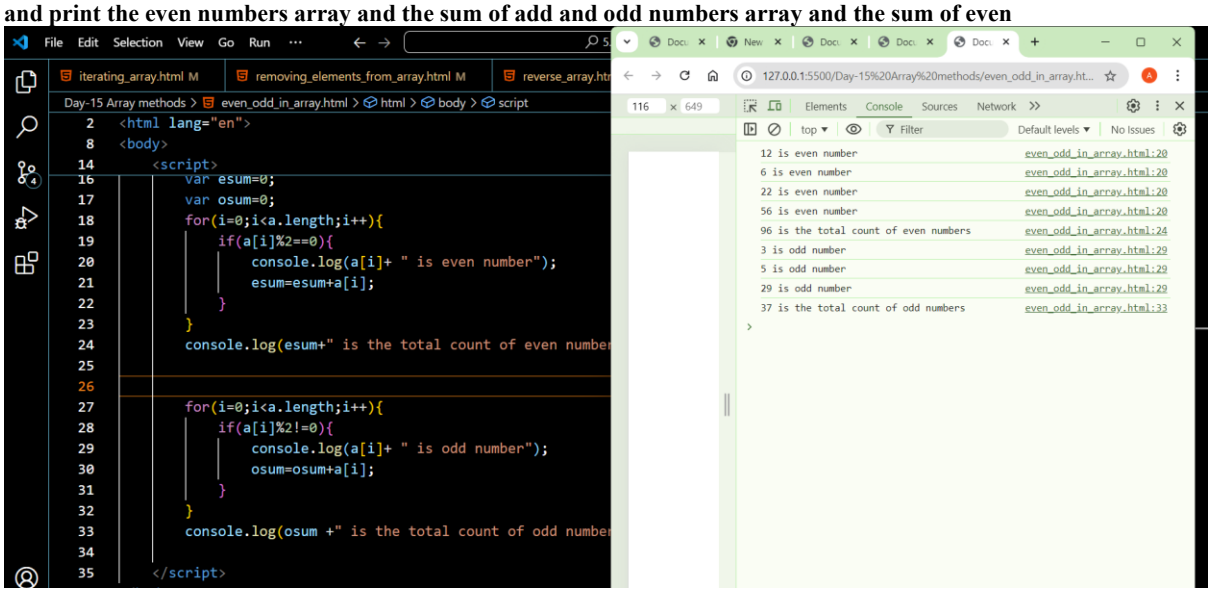
2)**Remove elements from an array**
Remove the first and last elements from the array.



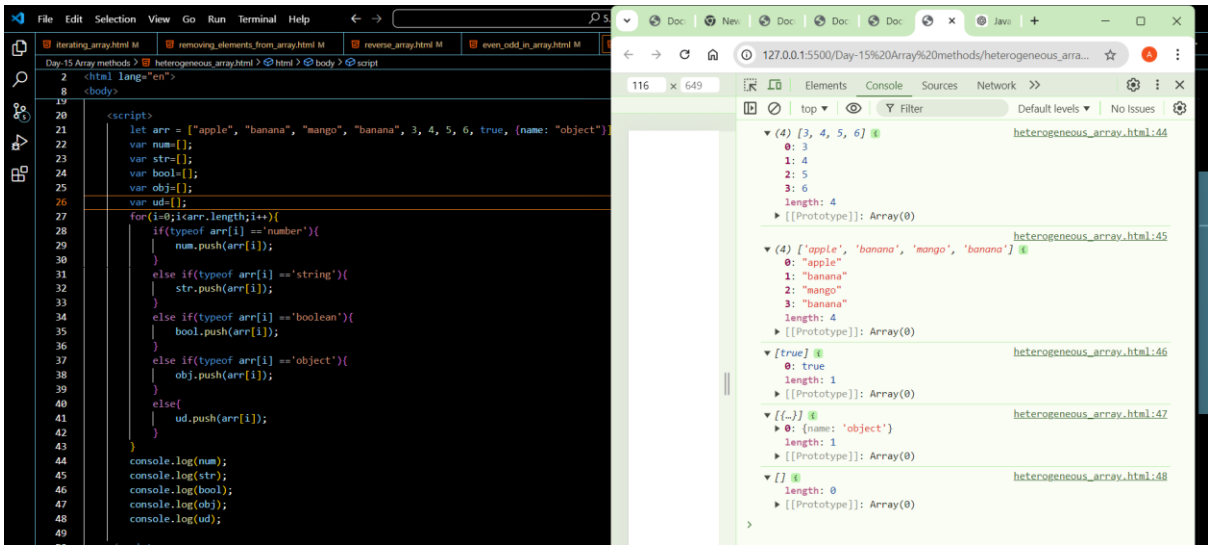
3)Reverse an array using for loop
Hints: use push method



4) find the even and odd numbers in an array [12,3,5,6,22,56,29]
and print the even numbers array and the sum of add and odd numbers array and the sum of even



5) Take a heterogeneous array and separate each data type into new array
hints : use loop, typeof and push method
inp: let arr = ["apple", "banana", "mango", "banana", 3, 4, 5, 6, true, {name: "object"}];
out :
num=[3,4,5,6]
str=["apple","banana","mango","banana"]
bool=[true]
obj=[{name: "object"}]



Dt:- 18-09-24

11. Array toString():-

toString() method is used to convert an array to string. this method converts each elements of an array to string then concatenates them together, separating each element with a comma.

Example:-

```
Var arr = [1,2,3,"hello"];
console.log(arr.toString());
```

O/P:- 1,2,3,hello

12. Array join:-

join() method is used to join the element of an array with a different seperator, you can use the join() method, passing the desired seperator as an argument.

Example:-
```
Var arr = [1,2,3,"hello"];
console.log(arr.join("-"));
```

O/P:- 1-2-3-hello

13. Array copyWithin():-

copyWithin() method copies a sequence of elements within the array to the position starting at the target index.

Example:-
```
let arr = [10,20,30,40,50];
console.log(arr.copyWithin(0,3));
// O/P:- 40 50 30 40 50
console.log(arr.copyWithin(1,3));
// O/P:- 10 40 50 40 50
```

```
console.log ( arr.copyWithin (0,3,4));
// o/p:- 40 20 30 40 50

console. log (arr. copy Within (1, 2,4));
//o/p:- 10 30 40 40 50

var arr =["a", "b", "c", "d", "e", "f"];
console.log (arr. copy Within (0,4,5));
//o/p:- [e, b, c, d, e, f]
```

14. Array flat() :-

flat() method creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.

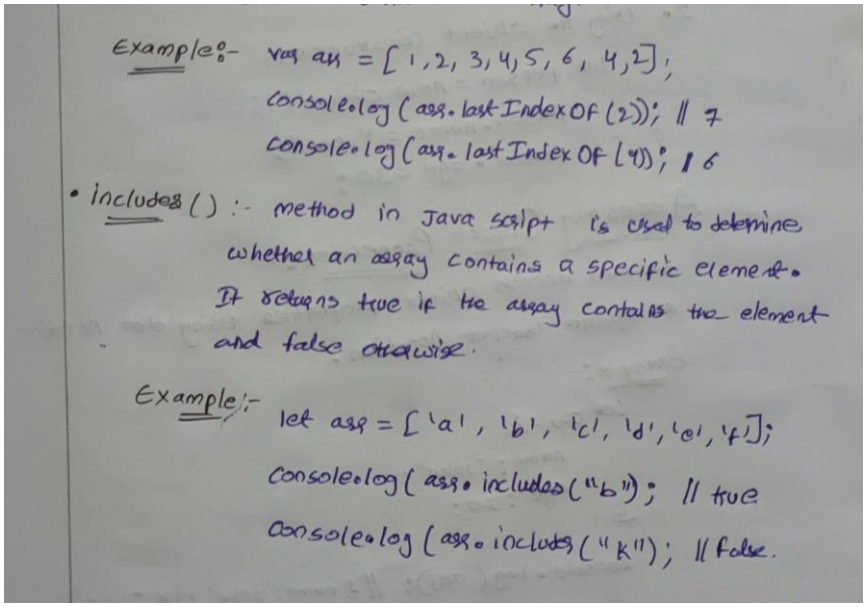
```
var arr =[1, 2, 3, [4,5, [6,7, [8,9]]]];
var b = arr.flat ()
console.log( b);
//o/p:-[1,2, 3,4, 5, Array [3]]
```

Search Methods :-

- indexOf(): method in JS is used to search an element within an array. it returns the index of the first occurance of the specified element or -1 if the element is not found.

  Example:-
  ```
  var arr = [1,2, 3, [4,5]];
  console.log (arr. indexOf ([4,5])); //-1
  console.log (arr. indexOf (3)); // 2
  ```

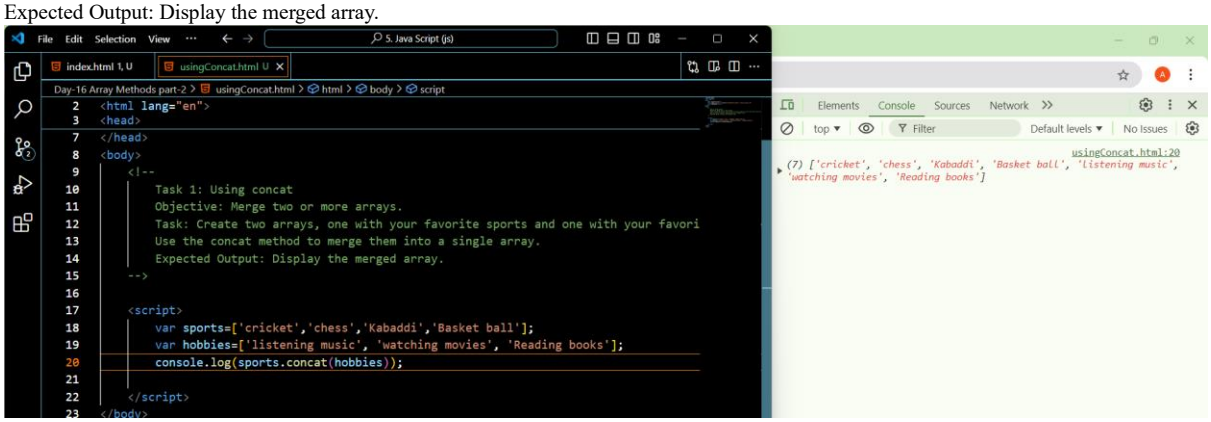
- lastIndexOf ():- method in JS is similar to the indexOf() method, but it searches for the last occurence of a specified element with in an array.



TASKS:

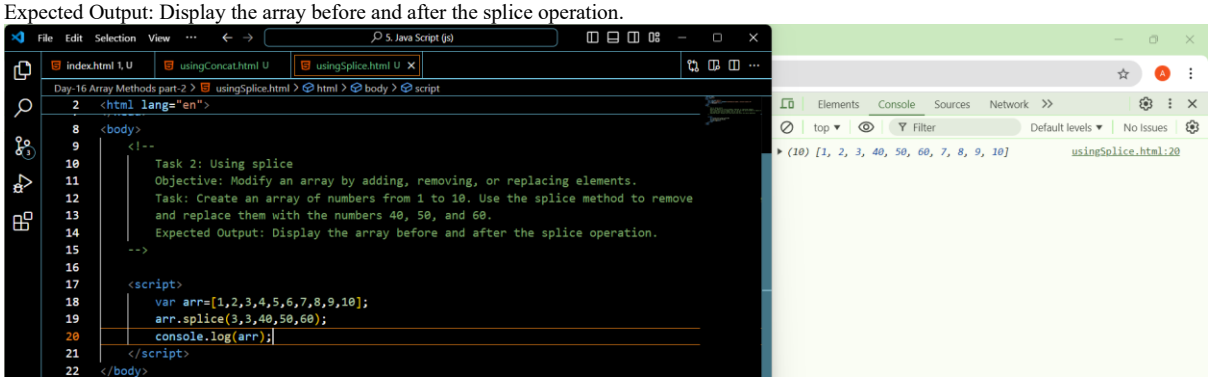
Task 1: Using concat

Objective: Merge two or more arrays.
Task: Create two arrays, one with your favorite sports and one with your favorite hobbies. Use the concat method to merge them into a single array.
Expected Output: Display the merged array.



Task 2: Using splice

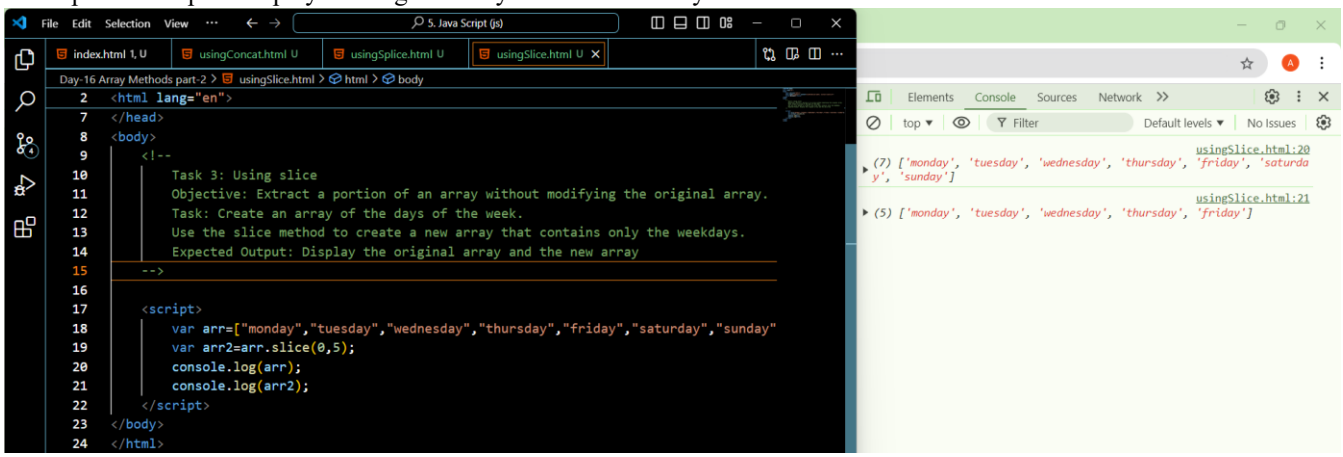
Objective: Modify an array by adding, removing, or replacing elements.
Task: Create an array of numbers from 1 to 10. Use the splice method to remove the numbers 4, 5, and 6, and replace them with the numbers 40, 50, and 60.
Expected Output: Display the array before and after the splice operation.



Task 3: Using slice

Objective: Extract a portion of an array without modifying the original array.
Task: Create an array of the days of the week. Use the slice method to create a new array that contains only the weekdays.

Expected Output: Display the original array and the new array.

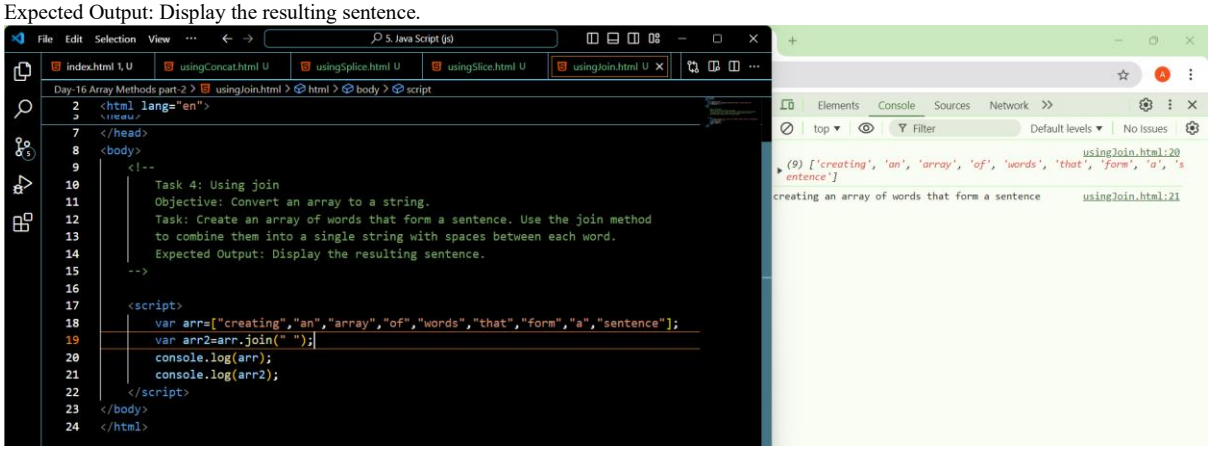


```
<html lang="en">
</head>
<body>
    <!--
        Task 3: Using slice
        Objective: Extract a portion of an array without modifying the original array.
        Task: Create an array of the days of the week.
        Use the slice method to create a new array that contains only the weekdays.
        Expected Output: Display the original array and the new array
    -->

    <script>
        var arr=["monday","tuesday","wednesday","thursday","friday","saturday","sunday"
        var arr2=arr.slice(0,5);
        console.log(arr);
        console.log(arr2);
    </script>
</body>
</html>
```

```
(7) ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday']
(5) ['monday', 'tuesday', 'wednesday', 'thursday', 'friday']
```

Task 4: Using join

Objective: Convert an array to a string.
Task: Create an array of words that form a sentence. Use the join method to combine them into a single string with spaces between each word.
Expected Output: Display the resulting sentence.

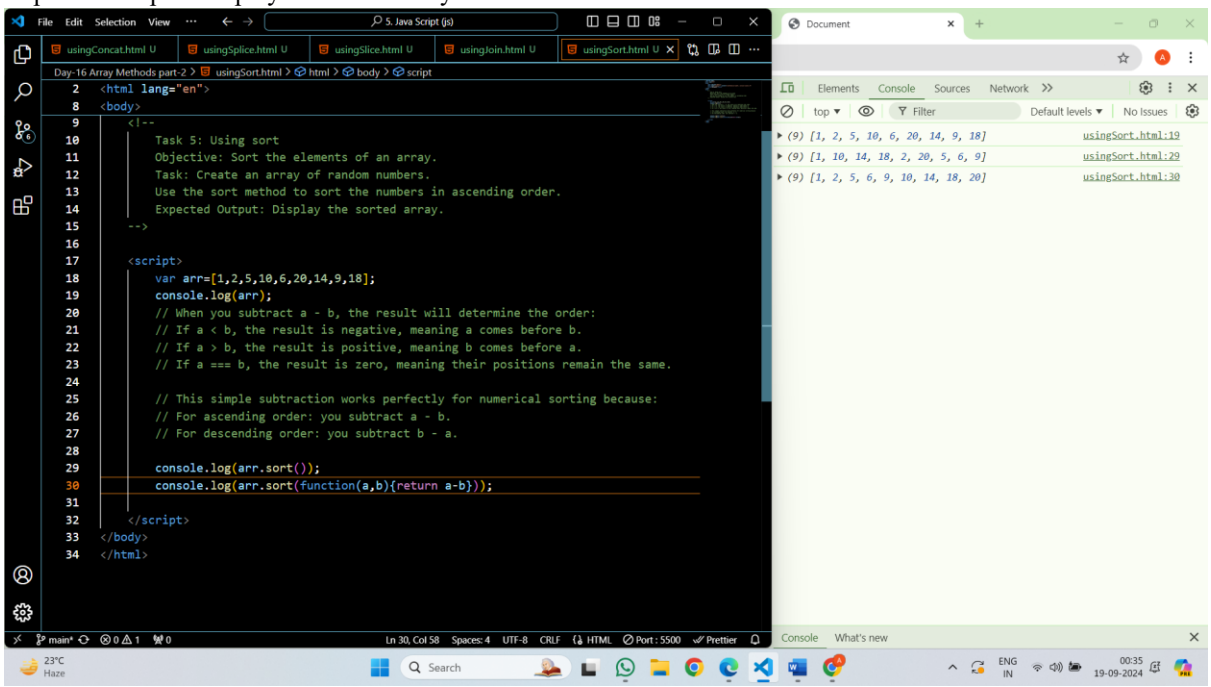


```
<html lang="en">
</head>
<body>
    <!--
        Task 4: Using join
        Objective: Convert an array to a string.
        Task: Create an array of words that form a sentence. Use the join method
        to combine them into a single string with spaces between each word.
        Expected Output: Display the resulting sentence.
    -->

    <script>
        var arr=["creating","an","array","of","words","that","form","a","sentence"];
        var arr2=arr.join(" ");
        console.log(arr);
        console.log(arr2);
    </script>
</body>
</html>
```

```
(9) ['creating', 'an', 'array', 'of', 'words', 'that', 'form', 'a', 'sentence']
creating an array of words that form a sentence
```

Task 5: Using sort

Objective: Sort the elements of an array.
Task: Create an array of random numbers. Use the sort method to sort the numbers in ascending order.
Expected Output: Display the sorted array.



```
<html lang="en">
<body>
    <!--
        Task 5: Using sort
        Objective: Sort the elements of an array.
        Task: Create an array of random numbers.
        Use the sort method to sort the numbers in ascending order.
        Expected Output: Display the sorted array.
    -->

    <script>
        var arr=[1,2,5,10,6,20,14,9,18];
        console.log(arr);
        // When you subtract a - b, the result will determine the order:
        // If a < b, the result is negative, meaning a comes before b.
        // If a > b, the result is positive, meaning b comes before a.
        // If a === b, the result is zero, meaning their positions remain the same.

        // This simple subtraction works perfectly for numerical sorting because:
        // For ascending order: you subtract a - b.
        // For descending order: you subtract b - a.

        console.log(arr.sort());
        console.log(arr.sort(function(a,b){return a-b}));

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

```
(9) [1, 2, 5, 10, 6, 20, 14, 9, 18]
(9) [1, 10, 14, 18, 2, 20, 5, 6, 9]
(9) [1, 2, 5, 6, 9, 10, 14, 18, 20]
```

Object Methods:

Objects
An object in JavaScript is a collection of data in key-value pairs where each key is a string (or a Symbol) and each value can be of any data type, including other objects, functions, arrays, and primitive data types like strings, numbers, and booleans. Objects are created using curly braces {}.

Creating Objects:

Literal notation:

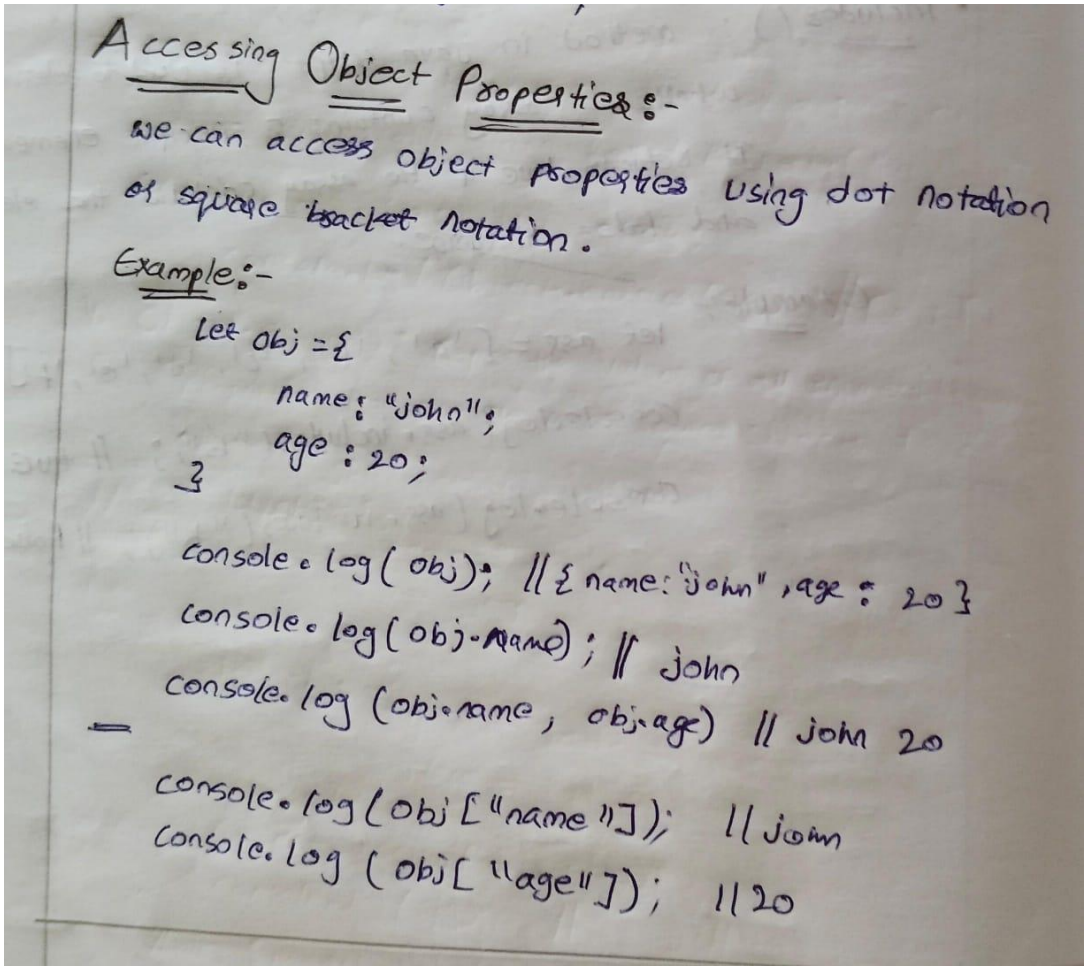
let person = { name: "John", age: 30 };

Using the Object constructor:

let person = new Object();
person.name = "John";
person.age = 30;

Accessing Object Properties:

You can access object properties using dot notation or square bracket notation:
console.log(person.name); // Dot notation
console.log(person['age']); // Square bracket notation



Adding & Modifing Properties:-

```
obj.gender = "Male";  // Adding New property
obj.age = 31;         // Modifing an existing property

console.log (obj.gender);  // male
console.log (obj.age);     // 31
```

Examples:-

```
let obj = {
        val1: {
            name: "John";
            age : 30;
        },
        val2 : 300
}
console.log (obj["val1"]["age"]);   // 30
```

Ex:-

```
Let obj ={
        val1 : function (){
            alert("hello world");
        },
        val2 : 300
}
console.log (obj["val1"]());
```

Object Methods:-

Methods are functions stored as object properties.

Example:-

```
let obj ={
        val1 : [1,2,{
                name : "john",
                val2 : [40,50,60, {
                    name : "john2",
                    val3 : [1,2,3,4,5]
                }]
            }]
}
console.log(obj["val1"][2]["val2"][3]["val3"][4]);
```

↳ access :

Deleating an object:-

Ex: delete obj.name;

Methods in JS:-

Object.keys()
object.values()
object.entries()
object.assign()
object.create()
Object.freeze()
Object.seal()
Object.hasown property()

1. Object.keys():- Returns an array of a given objects property name.

Example:- var obj = {
a:1,
b:2,
c:3 };
console.log(Object.key(obj)); // ["a", "b", "c"]

2. Object.values():- Returns an array of a given object's own enumerable property values.

Ex:- var obj = { a:1, b:2, c:3};
console.log (Object.values (obj)); // [1,2,3]

3. Object.entries():- Returns an array of a given object's own enumerable string-keyed property [key, value] pairs.

Ex:- var obj = {a:1, b:2, c:3};
console.log (Object.entries (obj)); // ["a",1] ["b",2] ["c",3]

4. Object.assign():- Copies the values of all enumerable own properties from one or more source objects to a target object.

Syntax:- Object.assign(target, new);

Ex:- let obj = {
name = "john";
age = 25;
}

let obj1 = {
gender = "Male";
place = "hyd";
}

Object.assign(obj, obj1);
console.log(obj);

// o/p:- {name: 'john', age = 25, gender: 'male', place: 'hyd'}

5. Object.create():- creates a new object with the specified prototype object and properties.

Ex:- let obj = Object.Create(null);
↳ object or null is mandatory.

console.log(obj) // { }

obj.name = "jenny";
obj.age = 22;
console.log(obj); // { name: "jenny", age: 22 }

let obj1 = object.create({city: "vizag"});
obj1.name = "jenny"
console.log(obj1) // { name: "jenny" }
console.log(obj1.city) // vizag.

6. Object.freeze():- Freezes an Object, preventing new properties from being added to it, existing properties from being removed, and values from being changed.

Ex:- let obj = {
name: "john",
age: 25;
}
Object.freeze(obj);
obj.name = "jenny"
obj.gender = "female";
delete obj.age;
console.log(obj);

// o/p:- { name: "john", age: 25 }

7. Object.seal():-

Seal an object, preventing new properties from being added to it and marking all existing properties as non-configurable.

Ex:- var obj = { name: john, age: 25 };

Object.seal(obj)

obj.name = "jenny";

add:- obj.gender = "female";

del:- delete obj.age;

console.log(obj);

// o/p:- { name = "jenny", age: 25 };

- it will just modify an object unable to add (or) delete.

8. Object.hasOwnProperty():-

Returns a boolean indicating whether the object has the specified property as its own property.

Example:-

```
let obj = {
    name: "john",
    age: 25
}
console.log(Objects.hasOwnProperty("name"));
// true.
```

How to iterate Objects:-

for in loop:-

```
Var obj = {
    name: "john", age: 25
};
for(i in obj){
    console.log(i); // name age (keys)
    console.log(obj[i]); // John 25 (values)
```

Iteration using for of method:-

Object.values(obj):-

```
for (i of Objects.values(obj)){
    console.log(i);        // john  25
}
```

Objects.keys(obj):-

```
for (i of Objects.keys(obj)) {
    console.log(i);        // name  age
}
```

Objects.entries(obj):-

```
for(i of Object.entries(obj)) {
    console.log(i);        // ['name', 'john']
}                          // ['age', 25]
```

```
for ([i,j] of Objects.entries(obj)){
    console.log(i,j);  // name  john
}                      // age   25
```

Api object Iteration:-

```
var obj = [
    // fake store API
]

var id = [];

for (let i in obj){
    id.push(obj[i]["id"]);
}

console.log(id);
```

Topic: Number & Math methods

Number methods

toFixed() method formats a number using fixed-point notation, which means it returns a string representation of the number with a specified number of decimal places. This is useful for rounding numbers to a certain number of decimal places.

```
let num = 123.45678;
console.log(num.toFixed(2)); // "123.46" - rounded to two decimal places
console.log(num.toFixed(3)); // "123.457" - rounded to three decimal places
console.log(num.toFixed(0)); // "123" - no decimal places, rounds to nearest integer

let num1 = 150;
console.log(num1.toFixed(2)); // "150.00" - Adds two decimal places with zeros
console.log(num1.toFixed(0)); // "150" - No change
```

parseInt(): Parses a string argument and returns an integer.

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

parseFloat(): Parses a string argument and returns a floating point number.

```
// Example 1: Basic Parsing
console.log(parseInt("42")); // 42 (simple integer parsing)
console.log(parseFloat("42")); // 42 (simple float parsing)
console.log(parseFloat("42.5")); // 42.5 (floating-point parsing)

// Example 2: Handling Leading and Trailing Whitespaces
console.log(parseInt("   123   ")); // 123 (leading/trailing spaces ignored)
console.log(parseFloat("   123.45   ")); // 123.45 (leading/trailing spaces ignored)

// Example 3: Parsing with Non-Numeric Characters
console.log(parseInt("123abc")); // 123 (parsing stops at "abc")
console.log(parseFloat("123.45abc")); // 123.45 (parsing stops at "abc")

// Example 4: Strings Starting with Non-Numeric Characters
console.log(parseInt("abc123")); // NaN (no leading numeric characters)
console.log(parseFloat("abc123.45")); // NaN (no leading numeric characters)

// Example 5: Handling Float Strings in parseInt()
```

```
console.log(parseInt("3.14")); // 3 (truncates the decimal part)
console.log(parseFloat("3.14")); // 3.14 (returns the full float)

// Example 6: Parsing Strings with Exponential Notation
console.log(parseInt("1e4")); // 1 (stops at "e")
console.log(parseFloat("1e4")); // 10000 (interpreted as 1 * 10^4)

// Example 8: Large and Small Numbers
console.log(parseInt("999999999999999999")); // 1e+21 (very large integer)
console.log(parseFloat("999999999999999999")); // 1e+21 (very large float)
console.log(parseInt("0.0001")); // 0 (fractional part ignored)
console.log(parseFloat("0.0001")); // 0.0001 (floating-point number)
```

isNaN(): Checks if a value is NaN (Not-a-Number). If it is number it returns false if it is not a number it returns true.

NUMBER – False

Not a Number - True

```
console.log(isNaN("hello")); // true
```

Output: true console.log(isNaN(123)); // Output: false

5. Number method: The Number constructor converts a value to a number.

```
console.log(Number("123")); // 123
console.log(Number("123abc")); // NaN
console.log(Number(true)); // 1
console.log(Number(false)); // 0
console.log(Number(null)); // 0
console.log(Number(undefined)); // NaN
```

Type Coercion:

isNaN() first tries to convert the parameter to a number, and then tests if the resulting value is NaN.

```
isNaN(NaN); // true
isNaN(undefined); // true
isNaN({}); // true

isNaN(true); // false
isNaN(false);//false
isNaN(null); // false
```

isNaN(37); // false

// Strings

isNaN("37"); // false: "37" is converted to the number 37 which is not NaN

isNaN("37.37"); // false: "37.37" is converted to the number 37.37 which is not NaN

isNaN("37,5"); // true

isNaN("123ABC","jhkhk"); // true: Number("123ABC") is NaN

isNaN(""); // false: the empty string is converted to 0 which is not NaN

isNaN(" "); // false: a string with spaces is converted to 0 which is not NaN

// Dates

isNaN(new Date()); // false; Date objects can be converted to a number (timestamp)

isNaN(new Date().toString()); // true; the string representation of a Date object cannot be parsed as a number

// Arrays

isNaN([]); // false; the primitive representation is "", which coverts to the number 0

isNaN([1]); // false; the primitive representation is "1"

isNaN([1, 2]); // true; the primitive representation is "1,2", which cannot be parsed as number

Math methods

1. Math.abs(): Returns the absolute value of a number.

```
console.log(Math.abs(10)); // 10
console.log(Math.abs(-10)); // 10
console.log(Math.abs(0)); // 0
console.log(Math.abs(-0)); // 0
console.log(Math.abs("-42")); // 42 (string converted to number)
console.log(Math.abs(null)); // 0 (null converted to 0)
console.log(Math.abs("Hello")); // NaN (string that can't be converted to a number)
```

2. Math.ceil(): Rounds a number up to the next largest integer.

```
console.log(Math.ceil(4.2)); // 5
console.log(Math.ceil(-4.2)); // -4
console.log(Math.ceil(0)); // 0
console.log(Math.ceil(7.004)); // 8
console.log(Math.ceil(-7.004)); // -7
```

3. Math.floor(): Rounds a number down to the previous largest integer.

```
console.log(Math.floor(4.7)); // 4
console.log(Math.floor(-4.7)); // -5
console.log(Math.floor(0)); // 0
console.log(Math.floor(7.999)); // 7
console.log(Math.floor(-7.999)); // -8
```

4. Math.round()

Rounds a number to the nearest integer. If the fractional part is 0.5 or greater, the argument is rounded to the next higher integer.

```
console.log(Math.round(4.5)); // 5
console.log(Math.round(4.4)); // 4
console.log(Math.round(-4.5)); // -4
console.log(Math.round(-4.6)); // -5
console.log(Math.round(7.999)); // 8
console.log(Math.round(-7.999)); // -8
```

5. Math.trunc()

Returns the integer part of a number by removing any fractional digits.

```
console.log(Math.trunc(4.9)); // 4
console.log(Math.trunc(-4.9)); // -4
console.log(Math.trunc(0)); // 0
console.log(Math.trunc(7.004)); // 7
console.log(Math.trunc(-7.004)); // -7
```

6. Math.max() Returns the largest of zero or more numbers

```
console.log(Math.max(1, 2, 3)); // 3
console.log(Math.max(-1, -2, -3)); // -1
console.log(Math.max(1, 2, 3, 10, 20)); // 20
```

7. Math.min(): Returns the smallest of zero or more numbers.

```
console.log(Math.min(1, 2, 3)); // 1
console.log(Math.min(-1, -2, -3)); // -3
console.log(Math.min(1, 2, 3, 10, 20)); // 1
```

8. Math.pow(): Returns the base raised to the power of the exponent.

```
console.log(Math.pow(2, 3)); // 8 (2^3)
console.log(Math.pow(5, 2)); // 25 (5^2)
console.log(Math.pow(4, 0.5)); // 2 (square root of 4)
console.log(Math.pow(-7, 2)); // 49 (negative base, even exponent)
```

9. Math.sqrt(): Returns the square root of a number.

```
console.log(Math.sqrt(16)); // 4
console.log(Math.sqrt(9)); // 3
console.log(Math.sqrt(0)); // 0
```

10. Math.random(): Returns a pseudo-random number between 0 (inclusive) and 1 (exclusive).

```
console.log(Math.random()); // Random number between 0 and 1
console.log(Math.random() * 10); // Random number between 0 and 10
console.log(Math.floor(Math.random() * 10)); // Random integer between 0 and 9
console.log(Math.floor(Math.random() * 100) + 1); // Random integer between 1 and 100
```

```
<script>
    var div=document.getElementById("div1");
    function n10(){
        var a=Math.round(Math.random()*10+1);
        div.innerHTML=a;
        console.log(a);

    }

    function n100(){
        var a=Math.round(Math.random()*100+1);
        div.innerHTML=a;
        console.log(a);
    }

    function n1000(){
        var a=Math.round(Math.random()*1000+1);
        div.innerHTML=a;
        console.log(a);
    }

</script>
```

Topic: Date methods , callback, for each and map

Date Methods

Creating Dates

Var now=new Date();

Specific Date and Time

let specificDate = new Date('2024-06-12T10:20:30Z');

Getting Date Components

let year = now.getFullYear();// year

let month = now.getMonth(); //0 -11

let day = now.getDate(); // 1-31

Day of the Week

let dayOfWeek = now.getDay(); // 0-6 (0 = Sunday, 6 = Saturday)

Hours, Minutes, Seconds, Milliseconds

let hours = now.getHours(); // 0-23

let minutes = now.getMinutes(); // 0-59

let seconds = now.getSeconds(); // 0-59

let milliseconds = now.getMilliseconds(); // 0-999

Setting Date Components

now.setFullYear(2025);

now.setMonth(6); // July

now.setDate(15);

Set Hours, Minutes, Seconds, Milliseconds

now.setHours(15);

now.setMinutes(30);

now.setSeconds(45);

now.setMilliseconds(500);

Formatting Date and Time

JavaScript provides methods to format dates as strings in different formats:

toDateString()

let dateStr = now.toDateString(); // e.g., "Wed Sep 22 2024"

toTimeString()

let timeStr = now.toTimeString(); // e.g., "15:30:45 GMT+0530 (India Standard Time)"

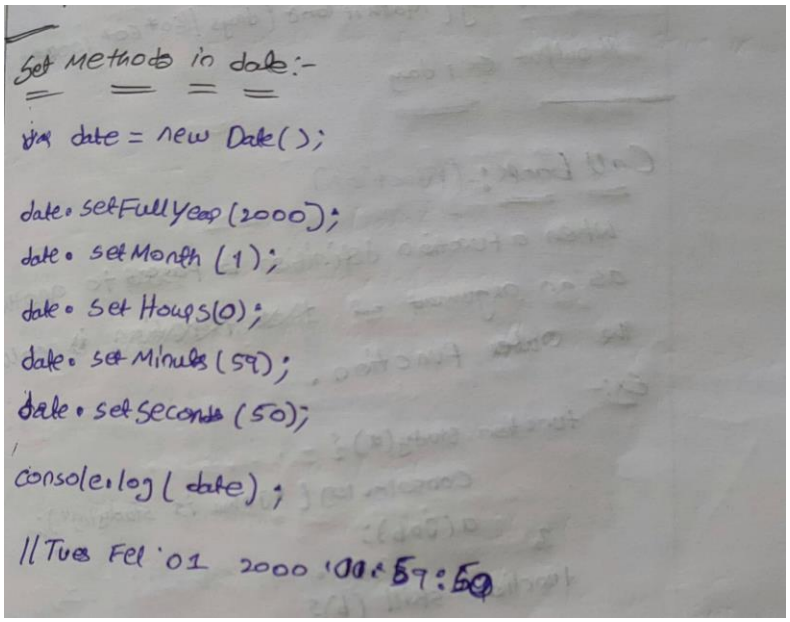
toLocaleDateString()

let localDateStr = now.toLocaleDateString(); // e.g., "9/22/2024" in US format

toLocaleTimeString()

let localTimeStr = now.toLocaleTimeString(); // e.g., "3:30:45 PM" in US format

These methods allow you to present dates in a human-readable or standardized format.



Age calculator

```
<input type="date" id="el1" />
<button onclick="fun()">clicke me</button>

<script>
 function fun() {
   var d = document.getElementById("el1").value;

   var olddate = new Date(d);
   var newdate = new Date();

   var year = newdate.getFullYear() - olddate.getFullYear();

   var milliseconds = newdate - olddate;
   var days = Math.floor(milliseconds / (60 * 60 * 24 * 1000));
```

```
    console.log(days);
    console.log(year);
  }
</script>
```

Callbacks

callback is a function defintion is passed as an argument to another function and is executed after some operation has been completed. This is a powerful feature that allows for asynchronous programming, enabling tasks to run concurrently without blocking the main execution thread.

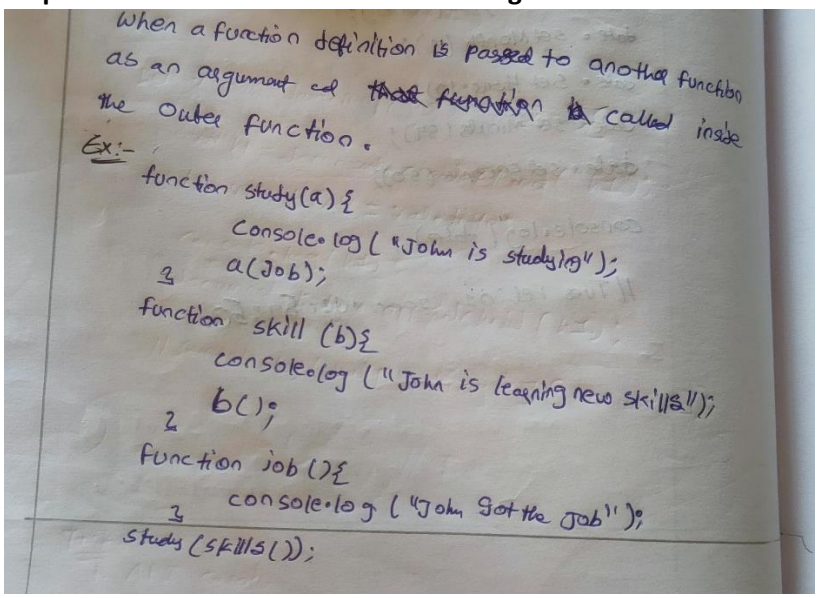
Defining the Callback:

```
// Step 1: Define the callback function
function myCallback() {
  console.log("Callback function executed!");
}
```

The callback function can be defined separately or inline.

```
// Defining separately
function myCallback() {
  console.log("Callback executed!");
}
doSomething(myCallback);

// Defining inline
doSomething(function () {
  console.log("Callback executed!");
});
```

Step 1: Define a Function that Takes Another Function as a Parameter(HOF)
Step 2: Define a function that takes another function as a parameter
Step 3: Execute the callback function
Step 4: Pass the callback function as an argument



Ex-1

```
function myFirst() {
  console.log("Hello");
}

function mySecond(a) {
  console.log("Goodbye");
}

var a=myFirst();
mySecond(a);
//Hello
//Goodbye
```

Ex-2

```
function myFirst() {
  console.log("Hello");
}

function mySecond() {
    myFirst()
  console.log("Goodbye");
}

mySecond();
```

The problem with the first example above, is that you have to call two functions to display the result.
The problem with the second example, is that you cannot prevent the function from displaying the result when it invokes every time.

```
function myFirst() {
  console.log("Hello");
}
function mySecond(a) {
  console.log("Goodbye");
}

mySecond(myFirst());//can call two functions at a time
mySecond()//one function
```

Usage of Callbacks

Callbacks are commonly used in situations where you want to perform tasks asynchronously,
such as:
Event Handling
Array Methods
Higher-Order Functions
Asynchronous Operations

Array Iteration and Transformation Methods:

Array for each

Array.forEach() is a method in JavaScript used to iterate over elements in an array. It executes a provided function once for each array element

Syntax:

```
array.forEach(function(value, index, array) {
   // Your code here
});
```

value: The current item in a array.
index (optional): The index of the item in the array.
array (optional): The array that forEach()

```
const numbers = [1, 2, 3, 4, 5];

numbers.forEach(function(val, index) {
   console.log(`Element at index ${index} is ${val }`);
});
//output
// Element at index 0 is 1
// Element at index 1 is 2
// Element at index 2 is 3
// Element at index 3 is 4
// Element at index 4 is 5
```

You can also use arrow functions for a more concise syntax:

```
numbers.forEach((number, index) => {
   console.log(`Element at index ${index} is ${number}`);
});
```

One important thing to note about forEach() is that it doesn't return anything. It simply iterates over the array. If you need to transform the elements of the array and create a new array based on those transformations, you might want to use methods like map() instead.

```
array.forEach(function(element) {
   return element * 2; // This return statement has no effect
});
```

Map method:

The map() method in JavaScript is used to create a new array by calling a provided function on every element in the calling array. It doesn't change the original array; instead, it returns a new array with the results of applying the provided function to each element.

Syntax:

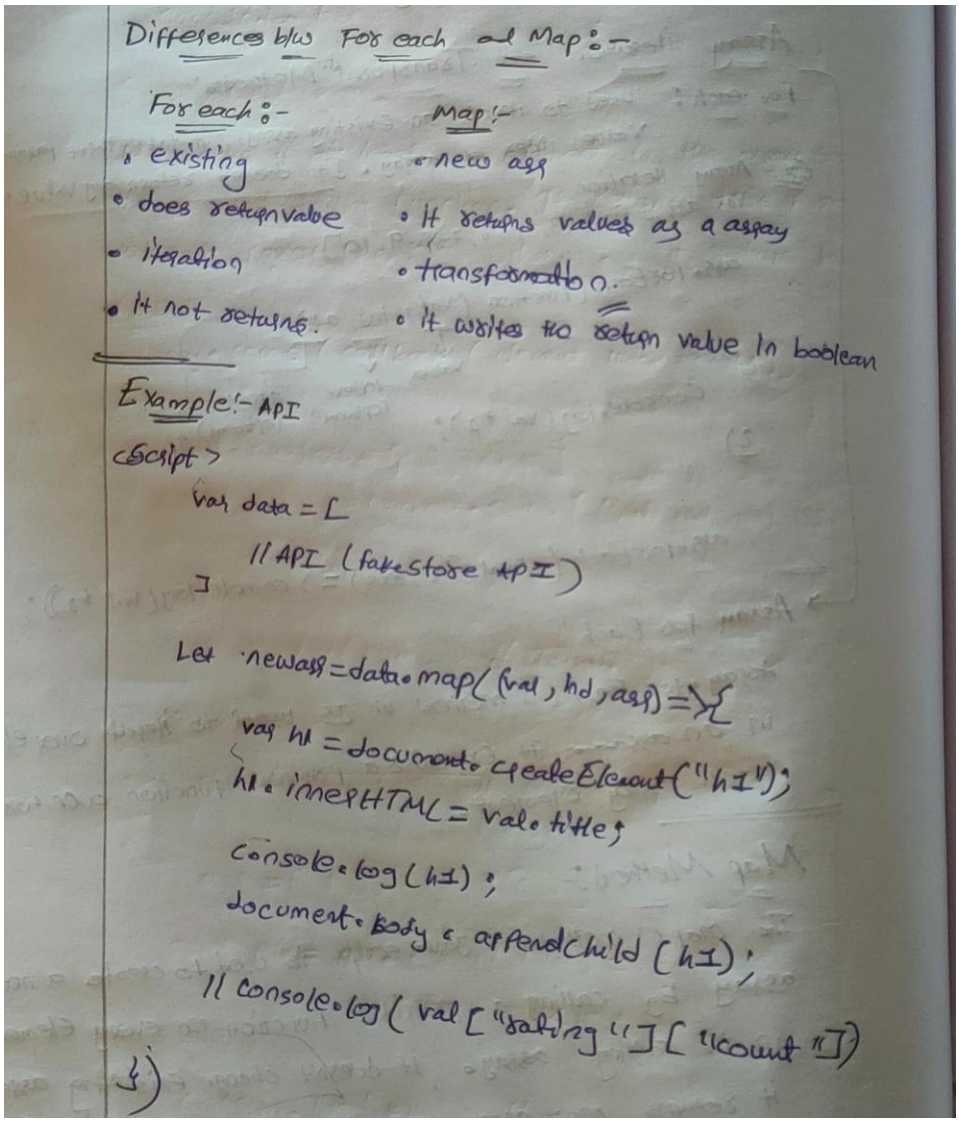
```
const newArray = array.map(function callback(currentValue, index, array) {
  // Return element for newArray
});
```

value: The current item in a array.
index (optional): The index of the item in the array.
array (optional): The array that forEach()

```
const numbers = [1, 2, 3, 4, 5];

const doubledNumbers = numbers.map(function(number) {
  return number * 2; // Return value determines the value
});
console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

You can also use arrow functions for more concise syntax:

```
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = numbers.map(number => number * 2);
console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```



Filter :-

- iteration & transformation
- it returns values
- it returns values which are satisfying the condition.

Ex: arr = [1,2,3,4,5,6,7];

```
new_arr = arr.filter ( (val, ind, arr) => {
        return val%2 == 0
});
```

(or)

```
new_arr = arr.filter (val => val%2 == 0)
;
console.log (new_arr);
```

Reduce(); used to reduce the elements of an array to single value.

Accumulator :- The accumulated value computed from previous iteration.

Ex:- sum of an Array.

```
arr = [1,2,3,4,5]

new_arr = arr.reduce ( (count, val, ind, arr) => {
    return val + count;
}, 0)
console.log (new_arr); // 15
```

count → accumulator
accumulator → accumulator value

Reduce Right Method :-

used to reduce the elements of an array to a single value but it process the array from right to left.

Sort method

- To remove lexographical

```
arr = [12, 1, 12, 10021, 63, 3456, 322];
arr.sort((a,b) => {
    return a-b;
})
console.log(arr);
```

Some Method :-

It checks if at least one element in the array satisfies the provided testing function. It returns true if any element passes the test, Otherwise, it returns false.

Syntax:

```
array.some(callback(element, index, array));
```

Ex:-

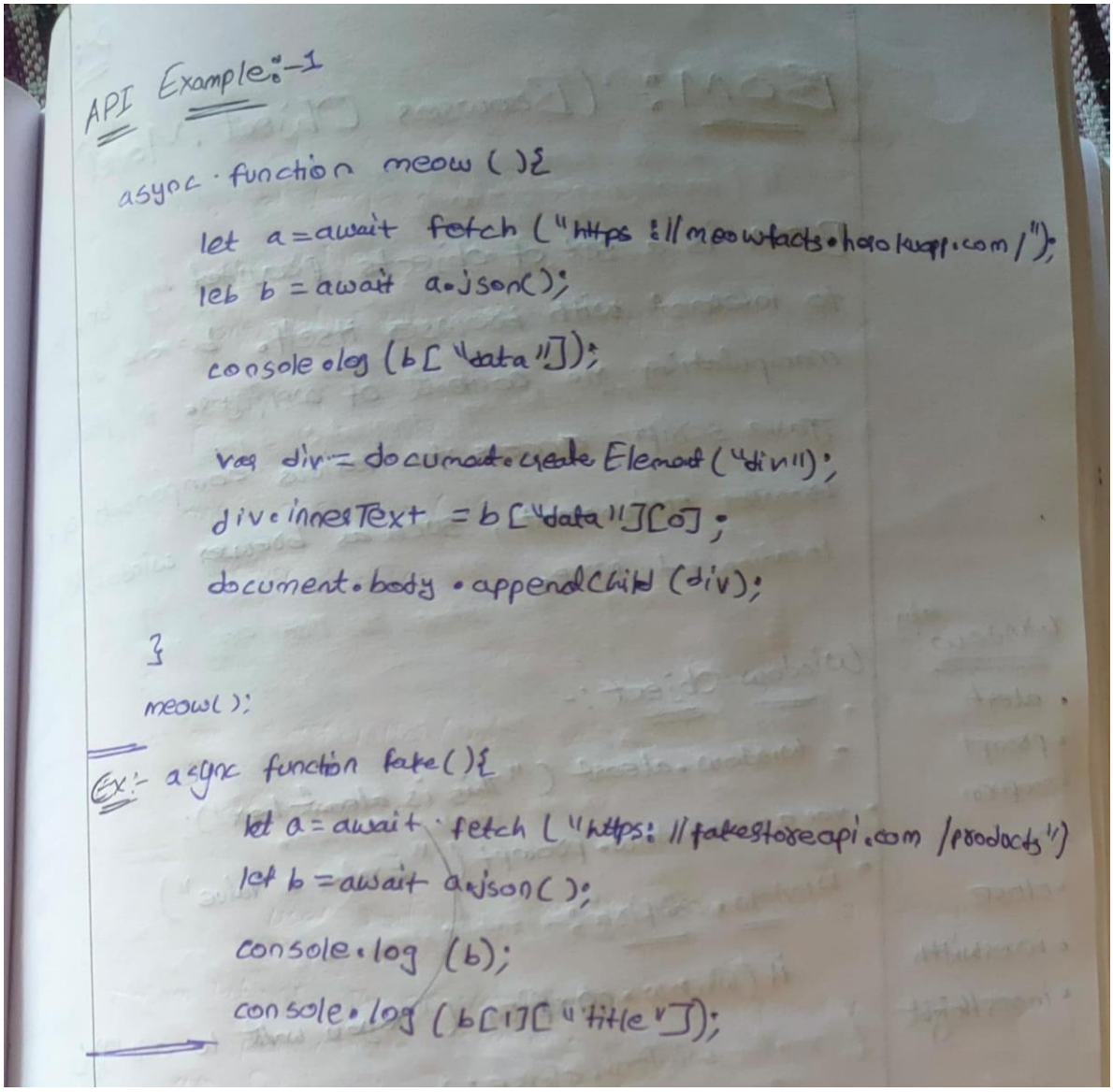
```
const numbers = [1,2,3,4,5]
const num2 = numbers.some((val ind arr) => {
    return val = 3;
})
console.log(num2) // True.
```

Every Method :-

- It should stabisfies all elements present in array.

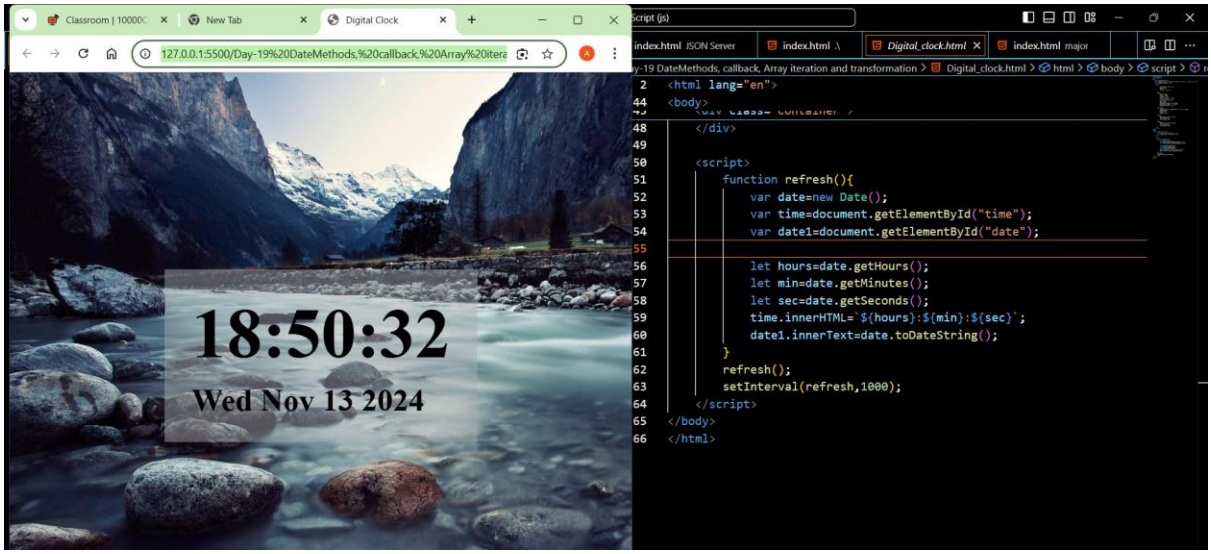
Ex:-

```
arr = [1,2,3,4,5,6,7]
const num = arr.every((val) => {
    return arr<10;
})
console.log(num); // True
```



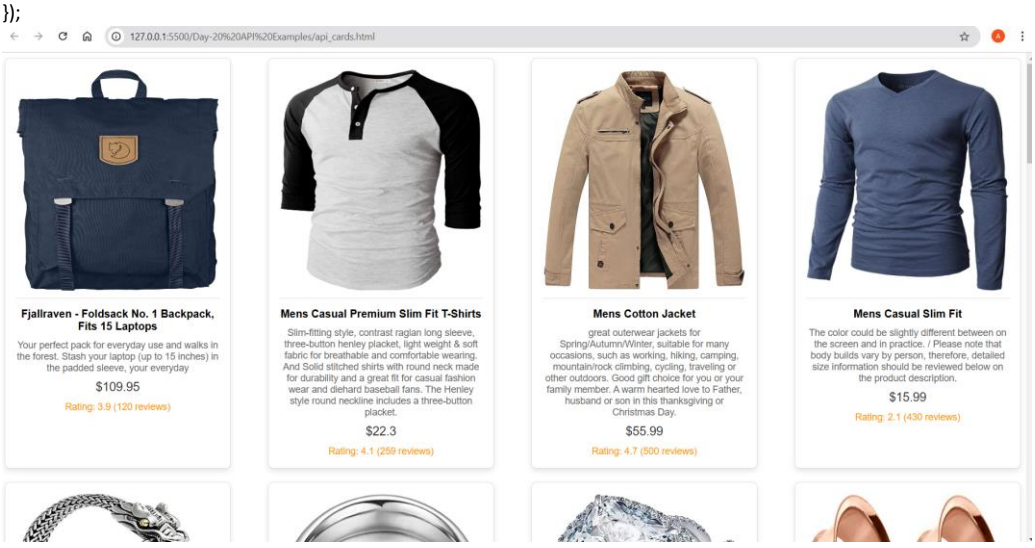
Task:

1. **Digital Clock**



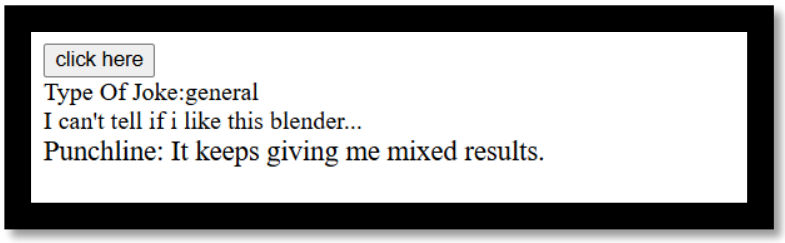
2. API Cards

```
const products = [
    //Fake Store API Data
 ]
const container = document.getElementById('card-container');

// Function to create a card for each product
function createProductCard(product) {
   const card = document.createElement('div');
   card.className = 'card';

   const image = document.createElement('img');
   image.src = product.image;

   const title = document.createElement('div');
   title.className = 'title';
   title.textContent = product.title;

   const description = document.createElement('div');
   description.className = 'description';
   description.textContent = product.description;

   const price = document.createElement('div');
   price.className = 'price';
   price.textContent = `$${product.price}`;

   const rating = document.createElement('div');
   rating.className = 'rating';
   rating.textContent = `Rating: ${product.rating.rate} (${product.rating.count} reviews)`;

   card.appendChild(image);
   card.appendChild(title);
   card.appendChild(description);
   card.appendChild(price);
   card.appendChild(rating);

   return card;
}

// Loop through the products and create cards
products.forEach(product => {
   const productCard = createProductCard(product);
   container.appendChild(productCard);
});
```



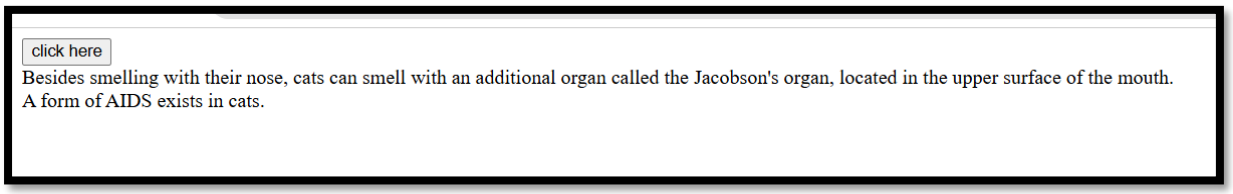
3. Genrate Jokes Using API:

```
<button onclick="jokes()">click here</button>
  <div class="container"></div>

  <script>
    async function jokes(){
      let a=await fetch("https://official-joke-api.appspot.com/random_joke");
      let b=await a.json();

      // console.log(b);
      // console.log(b["setup"]);

      var span=document.createElement("span");
      span.innerHTML="Type Of Joke:";
      var span1=document.createElement("span");
      span1.innerHTML=b["type"];
      span1.style.paddingBottom="15px";
      var div1=document.createElement("div");
      div1.innerHTML=b["setup"];

      var div2=document.createElement("div");
      div2.innerHTML="Punchline: "+b["punchline"];
      div2.style.fontSize="18px"

      var div=document.getElementsByClassName("container")[0];
      span.appendChild(span1);
      div.appendChild(span);
      div.appendChild(div1);
      div.appendChild(div2);

      // document.body.appendChild(div);
    }
    // jokes();
  </script>
```



4. Genrate Meow facts using API

```
  <button onclick="me()">click here</button>
  <script>
    async function me(){
      let a=await fetch("https://meowfacts.herokuapp.com/");
      let b=await a.json();

      console.log(b["data"][0]);

      var div=document.createElement("div");
      div.innerHTML=b["data"][0];
      document.body.appendChild(div);
    }
    // me();
  </script>
```



Topic: Browser Object Model (BOM)

The **Browser Object Model (BOM)** is a set of objects provided by web browsers to interact with the browser itself, beyond just manipulating the content of a web page. It provides JavaScript access to various components of the browser environment, such as the browser window, history, location, and more.

1. Window Object

The window object represents the browser's window. All global JavaScript objects, functions, and variables automatically become members of the window object.

2. **Window Object**:

Get the width and height of the browser window

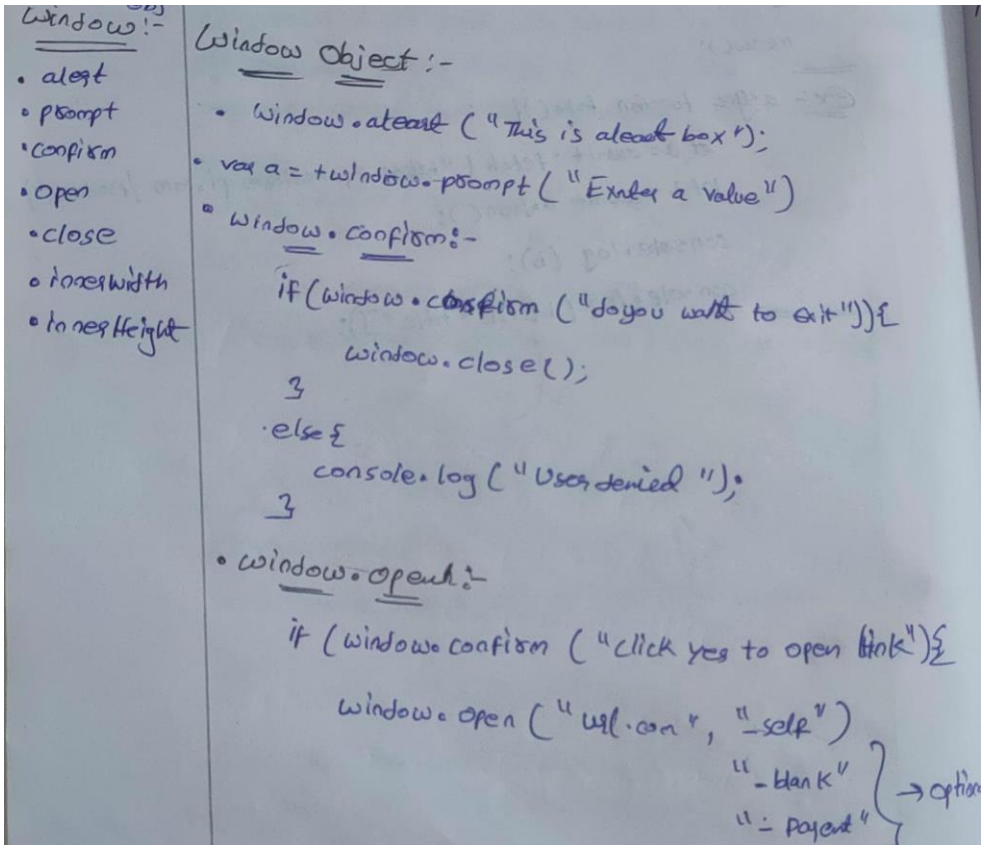
```
console.log(window.innerWidth);
console.log(window.innerHeight);
window.open("https://example.com", "_blank", "width=600,height=400");
```

3. Navigator objects

The Navigator object in JavaScript provides information about the browser's name, version, platform, and capabilities.

4. Location objects

It provides properties that allow you to access and manipulate different parts of the URL



- innerHeight & innerWidth :-

```
console.log (window.innerHeight);
console.log (window.innerWidth);
```

Ex:-
```
if (window.innerWidth < 600){
    document.body.style.backgroundColor = "red";
}
```

Navigator objects :-

It provides the information about the browser's name, version, platform and capabilities.

Navigator obj:-
- geolocation
- appName
- appCodeName
- appVersion
- userAgent
- platform
- online
- cookieEnabled

- navigator.online (say online or offline)

Ex:-
```
console.log (navigator.online);
if (navigator.online){
    console.log ("browser is online");
}
else{
    console.log ("browser is offline");
}
```

- appCodeName, appName, AppVersion, UserAgent

```
console.log(navigator.appCodeName); // Mozilla
console.log (navigator.appName); // Netscape
console.log (navigator.appVersion); // 5.0 (Linux- - -)
console.log (navigator.userAgent); // Mozilla /5.0version
console.log (navigator.platform); // Win32
```

- geolocation :-

```
if (navigator.geolocation){
    navigator.geolocation.getCurrentPosition(function(position){
        console.log("latitude:" + position.coords.latitude);
        console.log("longitude:" + position.coords.longitude);
    });
}
else{
    console.log("geolocation is not supported in this browser");
}
```

- cookieEnabled :-

checks cookies are Enabled or not.

Ex:-

```
console.log(navigator.cookieEnabled);
// true.
```

Location obj's:-
- href
- pathname
- hostname
- protocol

Location Objects :-

```
console.log(location.href); // https://127.0.0.1:5500/day22/main.html
console.log(location.pathname); // 127.0.0.1
console.log(location.hostname); // day22/main.html
console.log(location.protocol); // http
```

```
if (window.confirm("redirect")){
    location.href = "day22.html";
}
```

→ this will redirects to another page.

Screen Objects:-

screen objects in javascript provides information about the users screen or display, such as width, height, colors, depth and pixel density.

screen objs:-
- height
- width
- pixelDepth
- colorDepth

Ex:-

```
console.log (screen.height);      // 1080
console.log (screen.width);       // 1920
console.log (screen.pixelDepth);  // 24
console.log (screen.colorDepth);  // 24
```

History Objects:-

History objs:-
- history.back();
- history.forward();
- history.go();

History object in Javascript represents the user's navigation history for the current browser window it allows you to navigate back and forward through the history stack.

Ex:-

```
<button onclick="his()"> click me </button>

<script>
function his(){
    window.open("1.html","_self");
}
</script>
```

1.html:-

```
<button onclick="gb()"> click me </button>

<script>
function gb(){
    window.history.back();
}
</script>
```

```
history.back();      // moves the browser back one page
history.forward();   // moves the browser forward one page
history.go(-2);      // moves the browser back two pages.
```

Cookie Objects:-

Cookies are small piece of data stored in the users web browser. They are Typically used by websites to remember users preferences, authentication status and other information related to their browsing session.

Timing functions:-

Timing fonctions:-

setTimeout()
setInterval()
clearInterval()

Timing functions are crucial for managing when and how often certain blocks of code execute.

- setTimeout() // takes time to execute
- setInterval() // Executes for every time period

Ex:- `<button onclick = "hello"> click me </button>`

```
function hello(){
    console.log("hello world");
}
setTimeout(hello, 5000);
```

Ex:-

```
function hello(){
    console.log("hello world");
}
setInterval(hello, 1000);
```

- clearInterval:-

```
function hello(){
    console.log("Hello world");
}
let a = setInterval(hello, 1000)
function hi(){
    clearInterval(a);
}
```

**Example:-

```
for (var i=0; i<10; i++) {
    setTimeOut ( ()=>{
        console.log(i);
    }, 1000)
}
```

o/p:- 10 (10 times)

→ difference is scoping.
var has global scope
let has block scope

**Example :-

```
for(let i=0; i<10; i++){
    setTimeOut (()=>{
        console.log(i)
    },0)
}
```

o/p:- 0 to 9

Session storage & Local storage:-

Session storage:- is a part of web storage API in web browsers that provides a way to key value pairs locally on the client side.

- Data stored in session storage is cleared when the page session ends (until browser closes).

Ex:- 1.html:-

```
let a = "john";
window.localStorage.setItem ("store", a);
```

2.html:-

```
let a = localStorage.getItem ("store");
console.log (a); //john
```

2N (A) setItem ("store", "something")
// something

Ex:- 1.html:-

```
let a={
    name: "john";
    age: 25
};
localStorage.setItem("store", JSON.stringify(a));
```

↳ to store object in local storage

2.html:-

```
let ans = localStorage.getItem('store');
console.log(JSON.parse(ans));
```

↳ to convert JSON to object.

Example:-

a.html:-

```
<input type="text" id="inp1"><br>
<input type="text" id="inp2"><br>
<button onclick="fun()"> Submit Information </button>
<script>
    function fun(){
        var inp1 = document.getElementById("inp1").value;
        var inp2 = document.getElementById("inp2").value;
        localStorage.setItem("inp1", inp1)
        localStorage.setItem("inp2", inp2)
        window.open("day22.html", "_self")
    }
```

b.html:-

```
let ans1 = localstorage.getItem('inp1');
let ans2 = localStorage.getItem("inp2");
var h1 = document.createElement("h1");
h1.innerText = ans1 + ans2;
document.body.appendChild(h1)
console.log(ans1, ans2);
```

Topic: Bubbling, capturing, binding, local storage and session storage

Event Bubbling:

- Event bubbling is a mechanism where when an event is triggered on a nested element inside another element, the event 'bubbles up' through its ancestors.

- By default, most events bubble.

- You can stop the bubbling phase using `**event.stopPropagation()**`.

```
<div id="parent">
    <button id="child">
        click me
    </button>
</div>

<script>
let parent=document.getElementById("parent");
let child=document.getElementById("child");

child.addEventListener("click", function(event){
    event.stopPropagation()// this will prevent the bubbling to the parent
    console.log("child is clicked");
})
parent.addEventListener("click", function(){
    console.log("parent is clicked")
})
</script>
```

Event Capturing:

- Event capturing is the opposite of event bubbling.

- During the capturing phase, the event is first captured by the outermost element and then propagated to the innermost element.

- You can listen to events during the capturing phase by passing `true` as the third parameter to **`addEventListener()`.**

```
<div id="parent">
    <button id="child">
        click me
    </button>
</div>

<script>
let parent=document.getElementById("parent");
let child=document.getElementById("child");

child.addEventListener("click", function(event){
    console.log("child is clicked");
})
parent.addEventListener("click", function(){
    console.log("parent is clicked")
},true)
```

Event Binding:

- Event binding refers to the **process of attaching event listeners to DOM elements**.

- This is typically done using `**addEventListener()**` or by assigning event handler properties like `onclick`.

Session storage and local storage

Session storage is a part of the Web Storage API in web browsers that provides a way to store key-value pairs locally on the client-side.

- sessionStorage maintains a separate storage area for each given origin that's available for the duration of the page session (as long as the browser is open, including page reloads and restores).
- Data stored in sessionStorage is cleared when the page session ends.
- Data is only accessible within the window/tab that set it.

// Storing data in sessionStorage

```
sessionStorage.setItem('username', 'John');
```

// Retrieving data from sessionStorage

```
let username = sessionStorage.getItem('username');
console.log(username); // Output: John
```

// Removing data from sessionStorage

```
sessionStorage.removeItem('username');
```

localStorage:

localStorage is a feature of web browsers that allows web applications to store key-value pairs locally on the client-side. It provides a persistent storage mechanism, meaning that the data stored in localStorage remains available even after the browser is closed and reopened, and across browser sessions.

- localStorage does almost the same thing as sessionStorage, but it persists even when the browser is closed and reopened.
- Data stored in localStorage has no expiration time.
- Data is accessible across windows and tabs within the same origin.

// Storing data in localStorage

```
localStorage.setItem('email', 'example@example.com');
```

// Retrieving data from localStorage

```
let email = localStorage.getItem('email');
console.log(email); // Output: example@example.com
```

// Removing data from localStorage
localStorage.removeItem('email');

how to display some data from one page to another page using local storage

local storage limited to handle only string key/value pairs you can do like below using **JSON.stringify** and while getting value **JSON.parse**
var testObject ={name:"test", time:"Date 2017-02-03T08:38:04.449Z"};
Put the object into storage:
localStorage.setItem('testObject', JSON.stringify(testObject));

Retrieve the object from storage:
var retrievedObject = localStorage.getItem('testObject');

console.log('retrievedObject: ', JSON.parse(retrievedObject));

Example: Add to cart functionality

//first file

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      .container {
        display: grid;
        grid-template-columns: auto auto auto;
        gap: 20px;
      }
      .container > div {
        padding: 20px;
        border: 1px solid red;
      }
      .container > div > div,
      h1 {
        padding: 10px;
        border: 2px solid blue;
      }
    </style>
  </head>
  <body>
    <h1>
      <button onclick="cart()">cart</button>
    </h1>
    <div id="row" class="container"></div>

    <script>
      async function apicall() {
        var newarr = [];
```

```
      var result = await fetch("https://fakestoreapi.com/products");
      var apidata = await result.json();
      console.log(apidata);

      var iterated = apidata.map((val) => {
        // console.log(val);
        var row = document.getElementById("row");
        var main = document.createElement("div");
        var child1 = document.createElement("h1");
        var child2 = document.createElement("div");
        var child3 = document.createElement("div");
        var child4 = document.createElement("div");
        child1.innerHTML = val.id + " <br>";
        child2.innerHTML = val.title + " <br>";
        child3.innerHTML = val.description + " <br>";
        child4.innerHTML = val.price + " <br>";

        var btn = document.createElement("button");
        btn.innerHTML = "click";
        btn.addEventListener("click", function () {
          newarr.push(val);
          sessionStorage.setItem("arr", JSON.stringify(newarr));
        });

        main.append(child1, child2, child3, child4, btn);
        row.appendChild(main);
      });
    }
    apicall();

    function cart() {
      window.open("sub.html", "_self");
    }
  </script>
 </body>
</html>
```

//second file

```
<!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>
  <div id="row"></div>
  <script>
    var newarrdata = JSON.parse(sessionStorage.getItem("arr"));
    console.log(newarrdata);

    var iterated = newarrdata.map((val) => {
      // console.log(val);
      var row = document.getElementById("row");
      var main = document.createElement("div");
      var child1 = document.createElement("h1");
```

```
        var child2 = document.createElement("div");
        var child3 = document.createElement("div");
        var child4 = document.createElement("div");

        child1.innerHTML = val.id + " <br>";
        child2.innerHTML = val.title + " <br>";
        child3.innerHTML = val.description + " <br>";
        child4.innerHTML = val.price + " <br>";

        var btn = document.createElement("button");
        btn.innerHTML = "click";
        btn.addEventListener("click", function () {
          main.style.display = "none";
        });
        main.append(child1, child2, child3, child4, btn);
        row.appendChild(main);
      });
    </script>
  </body>
</html>
```

Topic: regex, synchronous and asynchronous

Regular Expressions (Regex)

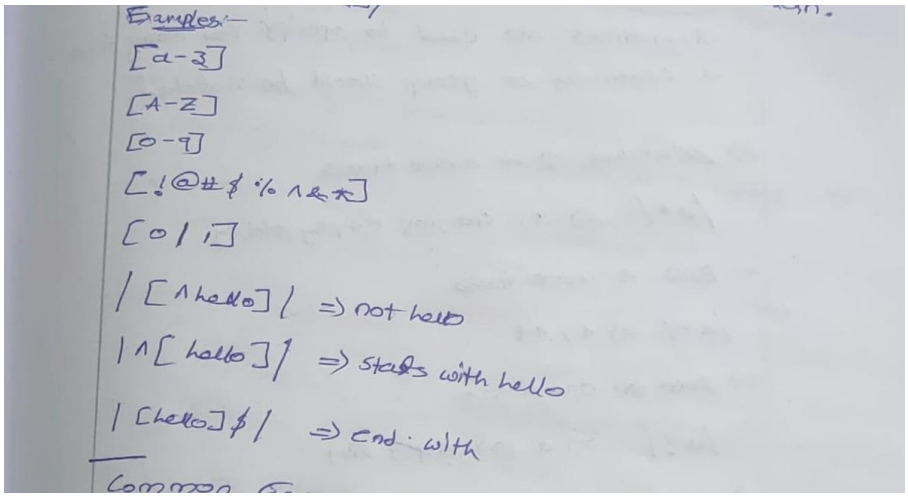
A regular expression is a sequence of characters that helps to create a search pattern, often used for string matching and manipulation.

Ways to Create Regular Expressions

Literal Notation: Uses slashes to define the pattern.

Syntax: /pattern/flags

Example:



/hello/ matches the string "hello".

/^[0-9]{10}$/ matches a 10-digit number (starts with a digit, ends with a digit).

/[0|1]/ matches either "0" or "1".

Bracket Expressions:

[ ] Matches any character inside the brackets.

Example: [a-z] matches any lowercase letter between 'a' and 'z'.

[^] Matches any character not in the brackets.

Example: [^a-z] matches any character except lowercase letters.

[0-9] Matches any digit between '0' and '9'.

[A-Z] Matches any uppercase letter between 'A' and 'Z'.

Common Escape Sequences

\d Matches any **digit** (equivalent to [0-9]).

\D Matches any **non-digit character** (equivalent to [^0-9]).

\w Matches **alphanumeric characters and underscores** (equivalent to [A-Za-z0-9_]).

\W Matches any **non-alphanumeric character**.

\. Matches a **literal period** (dot).

Methods for Using Regular Expressions

test(): Checks if the pattern exists in a string and returns true or false.

```
const regex = /\d+/;

console.log(regex.test("123"));  // true

console.log(regex.test("abc"));  // false
```

Quantifiers

Quantifiers are used to **specify how many times a character or group should be matched**:

- Matches **zero or more times**.
- **Example:** /a*/ matches "a", "aa", or an empty string.

- \+ Matches **one or more times**.
- **Example:** /a+/ matches "a", "aa", but not an empty string.

- ? Matches **zero or one time**.
- **Example:** /a?/ matches "a" or an empty string.

- {n}: Matches exactly **n occurrences** of the preceding element.
- **Example:** /a{3}/ matches exactly three "a" characters in a row ("aaa").
- "aaa" matches, "aa" does not match.

- {n,}: **Matches n or more occurrences** of the preceding element.
- **Example:** /a{3,}/ matches three or more "a" characters.
- aaa", "aaaa", and "aaaaa" match, but "aa" does not.

- {n,m}: **Matches between n and m occurrences** of the preceding element.
- **Example:** /a{3,5}/ matches between 3 to 5 "a" characters.
- "aaa", "aaaa", and "aaaaa" match, but "aa" and "aaaaaa" do not.

Examples of Regex

Phone Number Validation:
Regex: /^[0-9]{10}$/
Explanation: Matches exactly 10 digits from 0-9, used for validating phone numbers.

Binary Values (0 or 1):
Regex: /[0|1]/
Explanation: Matches either 0 or 1.

Simple Email Validation:
Regex: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/
Explanation: Matches a basic email format.

Examples

1. Validate an Email Address
Pattern: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/
Explanation:
^[a-zA-Z0-9._%+-]+ - Matches the username (alphanumeric characters and special symbols like ._%+-).
@ - Requires the "@" symbol.
[a-zA-Z0-9.-]+ - Matches the domain name.
\.[a-zA-Z]{2,} - Ensures a valid top-level domain (TLD), like .com or .net.

2. Validate a Phone Number (10 digits)
Pattern: /^[0-9]{10}$/
Explanation: ^[0-9]{10}$ - Ensures exactly 10 digits between 0 and 9.

3. Validate a URL
Pattern: /^(https?:\/\/)?(www\.)?[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}(\/\S*)?$/
Explanation:
https?:\/\/ - Matches "http" or "https".
(www\.)? - Optional "www.".
[a-zA-Z0-9.-]+ - Matches the domain name.
\.[a-zA-Z]{2,} - Ensures a valid TLD.
(\/\S*)? - Optionally allows a path after the domain.

4. Validate a Credit Card Number
Pattern: /^(?:\d{4}-?){3}\d{4}$/
Explanation:
(?:\d{4}-?){3} - Matches three groups of 4 digits, optionally separated by hyphens.
\d{4} - Matches the final group of 4 digits.

5. Validate a ZIP Code (US, 5 digits)
Pattern: /^\d{5}(-\d{4})?$/
Explanation:
^\d{5} - Matches exactly 5 digits.
(-\d{4})? - Optionally matches a hyphen followed by 4 digits for ZIP+4 codes.

6. Match a Date (MM/DD/YYYY)
• **Pattern:** /^(0[1-9]|1[0-2])\/(0[1-9]|[12][0-9]|3[01])\/\d{4}$/
• **Explanation:**
• (0[1-9]|1[0-2]) - Matches months from 01 to 12.
• (0[1-9]|[12][0-9]|3[01]) - Matches days from 01 to 31.
• \d{4} - Matches exactly 4 digits for the year.

7. Match a Time (24-hour format)
• **Pattern**: /^([01][0-9]|2[0-3]):[0-5][0-9]$/
• **Explanation**:
• ([01][0-9]|2[0-3]) - Matches hours from 00 to 23.
• :[0-5][0-9] - Matches minutes from 00 to 59.

8. Match Only Numbers
Pattern: /^\d+$/
Explanation:
^\d+$ - Matches any sequence of one or more digits (whole numbers only).

9. Match Only Alphanumeric Characters
Pattern: /^[a-zA-Z0-9]+$/
Explanation: ^[a-zA-Z0-9]+$ - Matches any sequence of alphanumeric characters.

10. Match Hexadecimal Colors
Pattern: /^#?([a-fA-F0-9]{6}|[a-fA-F0-9]{3})$/
Explanation:
#? - The # symbol is optional.
([a-fA-F0-9]{6}|[a-fA-F0-9]{3}) - Matches either a 6-character or 3-character hex color.

11. Strip Whitespace from the Beginning and End of a String
Pattern: /^\s+|\s+$/g
Explanation:
^\s+ - Matches one or more whitespace characters at the start.
\s+$ - Matches one or more whitespace characters at the end.
g flag - Global search for all matches.

12. Match a Word Boundary
Pattern: /\bword\b/
Explanation:
\b - Ensures the pattern matches at word boundaries.
word - The specific word to match.

13. Match a Floating-Point Number
Pattern: /^[+-]?([0-9]*[.])?[0-9]+$/
Explanation:
[+-]? - Allows an optional "+" or "-" sign.
([0-9]*[.])? - Optionally matches digits before and after a decimal point.
[0-9]+ - Requires at least one digit.

14. Validate a Strong Password
Pattern: /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/
Explanation:
(?=.*[a-z]) - Requires at least one lowercase letter.
(?=.*[A-Z]) - Requires at least one uppercase letter.
(?=.*\d) - Requires at least one digit.
(?=.*[@$!%*?&]) - Requires at least one special character.
{8,} - Must be at least 8 characters long.

15. Match a Repeated Character
Pattern: /(\w)\1+/
Explanation:
(\w) - Captures any alphanumeric character.

\1+ - Matches one or more occurrences of the same captured character.

16. Match an IP Address (IPv4)
Pattern: /^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/
Explanation:
Each part of the IP is matched using (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?) which ensures a valid number between 0 and 255.

17. Match HTML Tags
Pattern: /^<\/?[\w\s"'-=]+>$/
Explanation:
<\/? - Matches an opening or closing tag.
[\w\s"'-=]+ - Matches tag content (attributes, values).
> - Matches the closing bracket.

18. Match Leading Zeroes
Pattern: /^0+(?=\d)/
Explanation:
^0+ - Matches one or more leading zeroes.
(?=\d) - Ensures a digit follows.

19. Validate a Username
Pattern: /^[a-zA-Z0-9_]{3,16}$/
Explanation:
^[a-zA-Z0-9_]{3,16}$ - Matches a username that is 3-16 characters long, only allowing alphanumeric characters and underscores.

20. Match a Word with Only Letters
Pattern: /^[A-Za-z]+$/
Explanation:
^[A-Za-z]+$ - Matches a word containing only letters (upper or lowercase).

Synchronous and asynchronous

Synchronous: In synchronous operations, code is executed sequentially, one line at a time. Each line must wait for the previous one to finish executing before it can start. This can sometimes lead to blocking behavior, where one task prevents another from executing until it's complete.

```
console.log("Start");
console.log("Middle");
console.log("End");
// start
// middle
// end
```

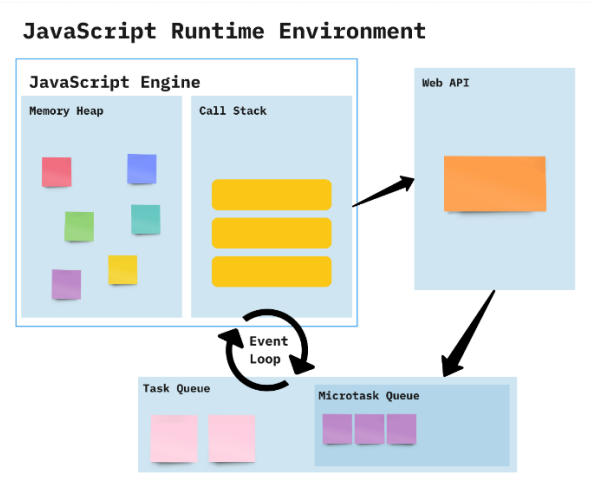
In this synchronous code, "Start" will be logged first, followed by "Middle", and then "End".

Asynchronous: Asynchronous operations allow code to execute independently from the main program flow. This means that while one operation is being processed, the program can continue to execute other tasks. Asynchronous operations are typically used for tasks that may take some time to complete, such as fetching data from a server or reading a file. In JavaScript, common asynchronous operations are handled using callbacks, promises, or async/await syntax.

```
console.log("Start");

setTimeout(()=>{
  console.log("middle")
}),2000;
console.log("End");

// start
// end
// middle
```

In this example, "Start" is logged first, then after a delay of 2000 milliseconds, "End" is logged, followed by " Middle ".

How js works



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

Synchronous:
Synchronous operations are executed sequentially, one after the other. Each operation must complete before the next one begins.

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.

CallStack Overflow: If the call stack grows too large, typically due to infinite recursion or excessive nested function calls, it can exceed the available memory allocated for the stack. This results in a "stack overflow" error and crashes the program.

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

1. Basic Synchronous Code

```
console.log("Task 1: Start");
console.log("Task 2: Middle");
console.log("Task 3: End");

// start
// middle
// end
```

2 Asynchronous with setTimeout (Event Loop in Action)

```
console.log("Task 1: Start");

setTimeout(() => {
  console.log("Task 2: Asynchronous Task (After 2 seconds)");
}, 2000);

console.log("Task 3: End");

start
```

```
// end
// asynchronous task
```

3. Asynchronous Code with a Promise

```
console.log("Task 1: Start");

const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Task 2: Promise Resolved (After 1 second)");
  }, 1000);
});

promise.then((message) => {
  console.log(message);
});

console.log("Task 3: End");

// Task 1: Start
// Task 3: End
// Task 2: Promise Resolved (After 1 second)
```

4. Multiple Asynchronous Tasks

```
 console.log("Task 1: Start");

  setTimeout(() => {
    console.log("Task 2: Asynchronous Task 1 (After 2 seconds)");
  }, 2000);

  setTimeout(() => {
    console.log("Task 3: Asynchronous Task 2 (After 0 seconds)");
  }, 0);

  console.log("Task 4: End");

  // Task 1: Start
  // Task 4: End
  // Task 3: Asynchronous Task 2 (After 0 seconds)
  // Task 2: Asynchronous Task 1 (After 2 seconds)
```

5. setTimeout and Promise Together

```
console.log("Task 1: Start");

setTimeout(() => {
  console.log("Task 2: Asynchronous Task (setTimeout)");
}, 0);

Promise.resolve().then(() => {
  console.log("Task 3: Promise Resolved");
});
```

```
console.log("Task 4: End");

// Task 1: Start
// Task 4: End
// Task 3: Promise Resolved
// Task 2: Asynchronous Task (setTimeout)
```

6. Using async/await

async/await is a syntactic sugar for working with promises. It behaves synchronously within an async function until an await keyword is encountered, at which point it returns to the event loop to handle other tasks.

```
console.log(“Task 1: Start");

   async function asyncTask() {
      console.log("Task 2: Inside asyncTask");
      await new Promise(resolve => setTimeout(resolve, 1000)); // Wait for 1 second
      console.log("Task 3: After 1 second wait in asyncTask");
   }

   asyncTask();

   console.log("Task 4: End");


//    Task 1: Start
//     Task 2: Inside asyncTask
//     Task 4: End
//     Task 3: After 1 second wait in asyncTask
```

7) setTimeout with Different Delays

```
  console.log("Task 1: Start");

   setTimeout(() => {
      console.log("Task 2: 2 seconds delay");
   }, 2000);

   setTimeout(() => {
      console.log("Task 3: 1 second delay");
   }, 1000);

   setTimeout(() => {
      console.log("Task 4: 0 seconds delay");
   }, 0);

   console.log("Task 5: End");

   // Task 1: Start
   // Task 5: End
   // Task 4: 0 seconds delay
   // Task 3: 1 second delay
   // Task 2: 2 seconds delay
```

Topic: callback hell and promises

Callback hell

Callback hell, also known as "**Pyramid of Doom,**" is a term used in JavaScript programming to describe a situation where multiple nested callbacks make the code difficult to read, understand, and maintain. This usually happens when dealing with asynchronous operations, such as making API requests or reading files.

Syntax:

```
step1(function() {
   step2(function() {
      step3(function() {
         console.log("All steps completed");
      });
   });
});
```

Example:1

```
function first(callback) {
  console.log("first");
  callback();
}

function second(callback) {
  console.log("second");
  callback();
}

function third(callback) {
  console.log("third");
  callback();
}

function fourth(callback) {
  console.log("fourth");
}

first(() => {
  second(() => {
    third(() => {
      fourth();
    });
  });
});
```

Example:2

```
function add(val, callback) {
  callback(val + 10);
}

function sub(val, callback) {
  callback(val - 5);
}

function mul(val, callback) {
  callback(val * 2);
}

function div(val, callback) {
```

```
    callback(val / 5);
  }

  add(10, (address) => {
    sub(addres, (subres) => {
      mul(subres, (mulres) => {
        div(mulres, (finalres) => {
          console.log(finalres);
        });
      });
    });
  });
```

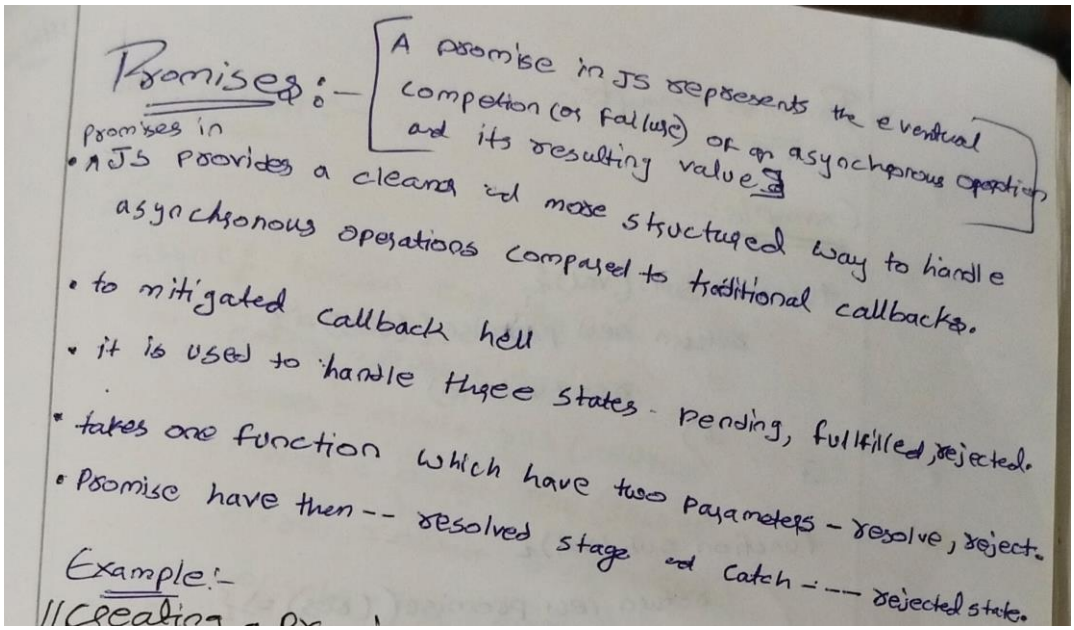
Example:3

```
function wakeUp(callback) {
  setTimeout(() => {
    console.log("1. Woke up");
    callback();
  }, 1000);
}

function eatBreakfast(callback) {
  setTimeout(() => {
    console.log("2. Ate breakfast");
    callback();
  }, 1000);
}

function study(callback) {
  setTimeout(() => {
    console.log("3. Studied");
    callback();
  }, 1000);
}

function goToSleep(callback) {
  setTimeout(() => {
    console.log("4. Went to sleep");
    callback();
  }, 1000);
}

// The callback hell part starts here
wakeUp(() => {
  eatBreakfast(() => {
    study(() => {
      goToSleep(() => {
        console.log("Finished all tasks!");
      });
    });
  });
});
```

To mitigate callback hell, several approaches have been developed:
Named functions
Promises
Async/await

Promises

Promises in JavaScript provide a cleaner and more structured way to handle asynchronous operations compared to traditional callbacks.It has three states: pending, fulfilled, or rejected.



Creating a Promise(producing) : You create a new Promise object using the Promise constructor. This constructor takes a function as an argument, which in turn takes two parameters: resolve and reject. Inside this function, you perform your asynchronous operation, and when it's completed, you call resolve with the result or reject with an error if it fails.

```
//promises creation
var promises=new Promise(function (resolve,reject){
var a=100;
if(a==10){
resolve("a is 10")
}
else{
reject("a is not 10")
}
});
```

Consuming a Promise: You consume a promise using the then method, which takes two optional parameters: a callback function to handle the resolved value, and a callback function to handle any errors.

```
//print the response
promises.then((val)=>{
console.log(val)
}).catch((err)=>{
console.log(err)
})
```

Promises also forming chain method which inturns make code readability difficult in order to avoid this

```
let add = (val) =>
  new Promise((resolve, reject) => {
    resolve(val + 10);
  });

let sub = (val) =>
  new Promise((resolve, reject) => {
```

```
    resolve(val - 10);
  });

let mul = (val) =>
  new Promise((resolve, reject) => {
    resolve(val * 5);
  });

let div = (val) =>
  new Promise((resolve, reject) => {
    resolve(val / 2);
  });

add(10)
  .then((addres) => sub(addres))
  .then((subres) => mul(subres))
  .then((mulres) => div(mulres))
  .then((divres) => console.log(divres))
  .catch((error) => console.error(error));
```

Promises aysnchronous

```
 let promise1= new Promise((resolve,reject)=>{
  console.log("promise 1");
  setTimeout(resolve, 2000, "promise 1 success")
 })

 let promise2= new Promise((resolve,reject)=>{
  console.log("promise 2");
  setTimeout(resolve, 1500, "promise 2 success")
 })

 let promise3= new Promise((resolve,reject)=>{
  console.log("promise 3");
  setTimeout(resolve, 1800, "promise 3 success")
 })

 let promise4= new Promise((resolve,reject)=>{
  console.log("promise 4");
  setTimeout(resolve, 500, "promise 4 success")
 })


 promise1.then((resolve)=>{console.log(resolve)})
 promise2.then((resolve)=>{console.log(resolve)})
 promise3.then((resolve)=>{console.log(resolve)})
 promise4.then((resolve)=>{console.log(resolve)})

//convert synchronous to aynchronous
 promise1
  .then((result) => {
    console.log(result);
    return promise2;
  })
  .then((result) => {
    console.log(result);
    return promise3;
  })
  .then((result) => {
    console.log(result);
    return promise4;
  })
  .then((result) => {
    console.log(result);
  });
```

Async/Await

Async/await is a modern feature in JavaScript that simplifies working with asynchronous code, especially when dealing with Promises. It allows you to write asynchronous code in a synchronous-like manner, making it easier to read, write, and maintain.

1. **Async Functions**: An async function is a function that operates asynchronously via the event loop. You declare an async function by prefixing the function declaration with the **async** keyword.

```
async function myAsyncFunction() {
  // Asynchronous code here
}
```

2. **Await Keyword:** The await keyword is used inside an async function to pause the execution of the function until a Promise is settled (resolved or rejected). It allows you to write code that looks synchronous but behaves asynchronously.

```
async function myAsyncFunction() {
  const result = await somePromise;
  // Code here executes after somePromise is resolved
}
```

Example

```
//promise is created
    function apromise() {
      return new Promise(function (res, rej) {
        var a = 20;
        if (a % 2 == 0) {
          res("num is even");
        } else {
          rej("num is odd");
        }
      });
    }

//resolving the promise value using async/await
    async function asyncfun() {
      var v = await apromise();
      console.log(v);
    }
    asyncfun();
```

//callback hell

```
    async function executor(){
      var addres=await add(10);
      var subres=await sub(addres);
      var mulres=await mul(subres);
      var divres=await div(mulres);
      console.log(divres);
    }
    executor()
```

//asynchronous

```
async function executor() {
  let result1 = await promise1;
  console.log(result1);

  let result2 = await promise2;
  console.log(result2);

  let result3 = await promise3;
  console.log(result3);

  let result4 = await promise4;
  console.log(result4);
}
executor();
```

// promise.all --> print all

//race – it declares the winner

Topic: Fetch, json, status codes and methods

Fetch

The **fetch()** API is a modern way to make network requests and handle responses. It is widely used to interact with web APIs, allowing to request and send data to servers. It returns a Promise that resolves to the Response object representing the response to the request.

fetch(url, options)

* **url**: The endpoint from which the resource is to be fetched.
* **options** (optional): An object containing additional settings such as HTTP method, headers, body, etc.

Example using fetch() API with Promises:

When Data fetched successfuly

```
fetch("https://fakestoreapi.com/products/1")
.then(res=>res.json())
.then(data=>console.log(data))
.catch(err=>console.log("there was a problem"))
```

When there is a error

```
fetch("https://fakestoreai.com/products/1")
.then(res=>res.json())
.then(data=>console.log(data))
.catch(err=>console.log("there was a problem"))
```

Explanation:

- fetch() returns a promise that resolves with the response object once the data is available.
- The .then() method is used to handle the fulfilled promise.
- If something goes wrong, we can use .catch() to handle the rejection.

. Async/Await

The async and await keywords simplify working with promises, allowing us to write asynchronous code that looks synchronous.

- async: Marks a function as asynchronous. This means it always returns a promise.
- await: Pauses the execution of the function until the promise is resolved or rejected.

Example using fetch() API with async/await:

```
async function executor(){
   var res= await fetch("https://fakestoreapi.com/products/1");
   var data = await res.json();
   console.log(data);
}
executor()
```

async with try and catch

```
async function executor() {
 try {
   var res = await fetch("https://fakestoreapi.com/products/1");
   var data = await res.json();
   console.log(data);
 } catch (err) {
   console.log("data not loading");
 }
}
executor();
```

Explanation:

- The executor function is marked as async, which means it can use await.
- Inside the function, await fetch() pauses the execution until the fetch() promise is resolved.

- Once resolved, the response is checked and the JSON data is awaited.
- If there's an error (e.g., network failure), it is caught by the try...catch block.

Differences Between Promises and Async/Await:
- Readability: Async/await makes asynchronous code look synchronous, which improves readability, especially in complex scenarios with many chained .then() blocks.
- Error Handling: Async/await uses try...catch, which is often more intuitive for handling errors compared to .catch() in promises.

JSON- javascript object notation

Ajax

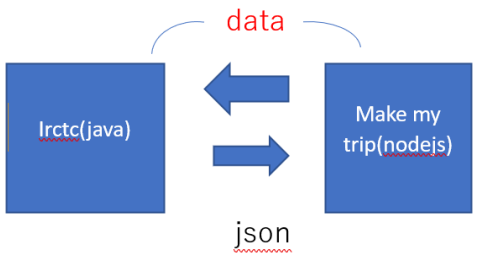
Ajax, which stands for "Asynchronous JavaScript and XML," is a web development technique used to create interactive web applications. It allows for the asynchronous exchange of data between the browser and the server

JSON

JSON, or **JavaScript Object Notation**, is a **lightweight data interchange format** widely used in web development and other software applications. Its **syntax is derived from JavaScript object notation, but it is language independent** making it easy to read and write for humans. JSON is commonly used for transmitting data between a server and a web application due to its simplicity, universality, and support for complex data structures like nested objects and arrays. It is supported by virtually all modern programming languages and is **commonly used in web APIs** for its lightweight nature and ease of parsing. Overall, JSON's simplicity, readability, and wide support make it a popular choice for **data interchange in various software applications**.

Why json

B2b



JSON.stringify():
JSON.stringify() is a built-in JavaScript method used to convert a JavaScript object into a JSON string.

JSON.parse():
JSON.parse() is a built-in JavaScript method used to parse a JSON-formatted string and convert it into a JavaScript object.

Http status codes

HTTP status codes are standard response codes returned by web servers to indicate the outcome of a client's request.

Important HTTP status codes along with their meanings:

1. **200 OK**: This status code indicates that the **request was successful**, and the server has returned the requested resource.
2. **201 Created**: Indicates that the **request was successful, and a new resource has been created as a result.**
3. **204 No Content**: The server successfully processed the request, but there is **no content to return**.
4. **400 Bad Request**: This status code is returned when the **server cannot process the request** due to a client error, such as malformed syntax or invalid parameters.
5. **401 Unauthorized**: Indicates that the **client needs to authenticate itself to access the requested resource.**
6. **403 Forbidden**: The server understood the request, but the client is not allowed to access the requested resource.

Http methods

HTTP methods, also known as HTTP request methods, are actions that indicate the desired operation to be performed on a resource identified by a URI (Uniform Resource Identifier).

1. **GET**: The GET method requests a representation of the specified resource. It is primarily used for retrieving data from the server. GET requests should only **retrieve data and should not have any other effect on the server**.
2. **POST**: The POST method submits data to be processed to a specified resource. It is commonly used for **creating new resources on the server or submitting form data**.
3. **PUT**: The PUT method replaces all current representations of the target resource with the request payload. It is typically **used to update or create a resource with a specific identifier**.
4. **PATCH**: The PATCH method is **used to apply partial modifications** to a resource. It is similar to the PUT method but only updates the parts of the resource specified in the request.
5. **DELETE**: The DELETE method **requests the removal of the specified resource**. It is used to delete resources identified by the URI from the server.

How to call api using fetch by then method

```
fetch("https://fakestoreapi.com/products")
  .then((val) => {
    return val.json();
  })
  .then((val) => {
    console.log(val);
  });
```

1. **fetch("https://fakestoreapi.com/products")**: This line initiates a request to the specified URL, which returns a Promise representing the response to that request.
2. **.then((response) => { return response.json(); })**: Once the request is complete, this line chains a **.then()** method to the Promise returned by **fetch()**. Inside this **.then()** method, it takes the response object, and the **json()** method is called on it. This method returns a Promise that resolves to the JSON representation of the response body.
3. **.then((data) => { console.log(data); })**: After parsing the JSON response, this line chains another **.then()** method to the Promise returned by **response.json()**. Inside this **.then()** method, it receives the parsed JSON data as a JavaScript object. In this example, it logs the retrieved data to the console using **console.log()**

Some apis

Food api: **https://api.edamam.com/search?q=biriyani&app_id=a52b4d43&app_key=e0e5c667605f5e91d8275c973531b80a**

Weather api - https://api.openweathermap.org/data/2.5/weather?q=**hyderabad**&units=metric&appid=466ddaa21a8de191e9f608bd11a56acb

Quotes api: https://api.quotable.io/random

Random joke: **https://v2.jokeapi.dev/joke/Programming?blacklistFlags=nsfw,religious,political,racist,explicit&type=single**

Movies info:- https://www.omdbapi.com/?t=**titanic**&apikey=76d079f0

Random joke

```
async function getRandomJoke() {
  const rawRes = await fetch('https://icanhazdadjoke.com/', {
    headers: {
      Accept: 'application/json'
    }
  });
  const res = await rawRes.json();
  console.log(res.joke);

}
getRandomJoke()
```

https://github.com/saiteja-yernagula/javascript-mini-projects

Topic: JSON Server

How to create a local json server

1) Download and Install Node js
2) Check whether it is installed or not by using below commands
   node –version
   npm –version

   Open powershell run as administrator and run the commands
   Get-ExecutionPolicy
   Set-ExecutionPolicy RemoteSigned
   Y (for yes)

   Get-ExecutionPolicy

3) Select the folder in your local , open in vs code terminal and use below command
   npm init –y
   You will find json packages then you can install any libraries
4) Install json server for api creation in local server
   npm install -g json-server@0
5) After installation watch the server using below command if it throws error follow the next step for script enabalation.

   json-server --watch db.json –port num

6) Adjust the version by reinstalling using below command
   npm install -g json-server@0

   json-server --watch db.json –port 6000

Stepwise All Methods

Step 1 – create db.json file
Step 2 – add the data

```
{
  "data":[{
  "name":"teja",
  "id":"4"
},
{
  "name":"sai",
  "id":"2"
},
{
  "name":"hemanth",
  "id":"3"
},
{
  "name":"chaitanya",
  "id":"1"
}
]}
```

Step 3 – json-server --watch db.json –port 6000

Step 4 - open js file paste the below program in it

```
// get method - used to get data from the server
    fetch("http://localhost:3000/data")
    .then(val=>val.json())
    .then(data=>console.log(data))
    .catch(err=>console.log("data not found"))
```

How to use post or patch in fetch

Index.html

```
<button onclick="getdata()">get data</button>
<button onclick="setdata()">set data</button>
```

Script.js:

```
<script>
  function getdata(){

    // get method - used to get data from the server
    fetch("http://localhost:3000/data")
    .then(val=>val.json())
    .then(data=>console.log(data))
    .catch(err=>console.log("data not found"))
  }


  let obj={
    name:"Raju",
    id:5
  };

  function setdata(){

    // set method - used to set  data to the server
    fetch("http://localhost:3000/data", {
      method: "POST",
      headers: {
        "Content-Type":"application/json"
      },
      body: JSON.stringify(obj)
    })
    .then(val=>val.json())
    .then(data=>console.log(data))
    .catch(err=>console.log("data not found"))
  }
</script>
```

Db.json:

```
{
 "data": [
   {
    "name": "Abhi",
    "id": "1"
   },
   {
    "name": "sai",
    "id": "2"
   },
   {
    "name": "hemanth",
    "id": "3"
   },
   {
    "name": "manoj",
    "id": "4"
   }
 ]
}
```

Json server

HTTP methods (GET, POST, PUT, PATCH, DELETE) using fetch() and interacting with a JSON Server. JSON Server is a simple API tool that allows you to simulate a REST API with basic HTTP methods

Assume your JSON server is running at http://localhost:3000 with the following db.json:

```
{
   "data":[{
   "name":"teja",
   "id":"4"
}
,
{
   "name":"sai",
   "id":"2"
},
{
   "name":"hemanth",
   "id":"3"
},
{
   "name":"chaitanya",
   "id":"1"
}
]}
```

1. GET Request: Fetching Data from the Server

```
fetch('http://localhost:3000/posts')
  .then(response => response.json())  // Parse the JSON response
  .then(data => console.log(data))    // Handle the received data
  .catch(error => console.error('Error:', error));  // Handle errors
```

If you want to add queries params
By name
```
 fetch("http://localhost:3000/data?name=teja")
```
By id
```
 fetch("http://localhost:3000/data?id =2")
```
By both
```
 fetch("http://localhost:3000/data?id =2&name=teja")
```
By limit- used for pagination
```
 fetch("http://localhost:3000/data?_limit=2 ")
```
By _sort
```
 fetch("http://localhost:3000/data?_sort=-id")
```

2. POST Request: Sending Data to the Server

```
fetch('http://localhost:3000/posts', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'  // Indicate we are sending JSON
  },
  body: JSON.stringify({                // Data to be added
    title: 'New post',
```

```
    author: 'Sam'
  })
})
  .then(response => response.json())    // Parse the JSON response
  .then(data => console.log('Post added:', data))  // Handle the response
  .catch(error => console.error('Error:', error));  // Handle errors
```

. Explanation: This adds a new post with the title "New post" and author "Sam" to the posts endpoint.

3. PUT Request: Replacing Existing Data on the Server

```
  fetch('http://localhost:3000/posts/1', {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    id: 1,            // `PUT` requires the `id`
    title: 'Updated first post',
    author: 'John Doe'
  })
})
  .then(response => response.json())
  .then(data => console.log('Post updated:', data))
  .catch(error => console.error('Error:', error));
```

Explanation: This updates the post with id: 1, changing its title and author.

4. PATCH Request: Partially Updating Data on the Server

```
  fetch('http://localhost:3000/posts/2', {
  method: 'PATCH',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    author: 'Jane Smith'  // Only update the author field
  })
})
  .then(response => response.json())
  .then(data => console.log('Post partially updated:', data))
  .catch(error => console.error('Error:', error));
```

Explanation: This updates only the author field of the post with id: 2.

5. DELETE Request: Removing Data from the Server

```
  fetch('http://localhost:3000/posts/1', {
  method: 'DELETE'
  })
  .then(() => console.log('Post deleted'))  // Handle deletion success
  .catch(error => console.error('Error:', error));
```

Topic: Apis, Exeception handling, this call apply bind and constructor function

API- application programming interface

API stands for Application Programming Interface

It is a collection of communication protocols and subroutines used by various programs to form the communication between them.

In simple words It allows communication between two different software systems.

How api works

Client initiates the request via the api uri (uniform resource identifier)

The api makes a call to the server after receiving the request

Then the server sends the response back to API with the information

Finally, the API transfers data to the client

Types of apis

- **Browser APIs**: Interact with browser and web page elements (dom api), storage api
- **Server Apis:** Provide data and services from a server (restfull api)
- **Third-party Apis:** Offered by external services (social media apis)
- **Rest apis:** REST APIs are designed to make server-side data readily available by representing it in simple formats such as JSON and XML.
- **Soap apis(Simple Object Access Protocol):** SOAP uses XML exclusively to format data, making it more verbose and complex compared to REST.
- **GraphQL:** Allows clients to request exactly the data they need, great for complex data relationships. The client controls what data to retrieve, reducing over-fetching or under-fetching of data.
- **gRPC (Google Remote Procedure Call):** gRPC is a modern, high-performance, open-source RPC (Remote Procedure Call) framework developed by Google suitable for microservices and performance-critical systems.
- **WebSockets:** WebSocket is a protocol that allows for real-time, two-way communication between the client and server over a single TCP connection. WebSocket can send data in JSON, XML, or any other format, but the communication is typically binary, Real-time, bi-directional communication for applications like chat and gaming.

Error handling methods is js

In JavaScript, try...catch statements are used to handle exceptions (errors) that occur in your code. By using these statements, you can ensure that your code continues to run even if an error occurs.

Error – It is an object that is created to represent a problem that occurs offten with userinput or establishing a connection.

Why we need try catch

// **case -1** ------> both statements will execute
console.log("hi hello");

```
console.log("you have reached the end");
```

// **case -2** ------>It interupts the program from 163 line to end because console.lag is not a method it

```
console.lag("hi hello");
console.log("you have reached the end");
```

To overcome this we need **try catch methods**

Syntax:

```
try {
  // Code that may throw an error
} catch (err) {
  // Code to handle the error
} finally {
  // Code that will always run, regardless of error
}
```

Explanation

try block: Contains the code that may throw an error. If no errors occur, the code inside the catch block is skipped.

catch block: Contains code that will execute if an error occurs in the try block. The err parameter contains the error object.

finally block (optional): Contains code that will run after the try and catch blocks, regardless of whether an error was thrown or not.

//try block contains error catch block takes one parameter err means error it will store the error caused by try

Ex-1:

```
try{
   console.lag("hi hello");
}
catch(err){
   console.log(err);
}
```

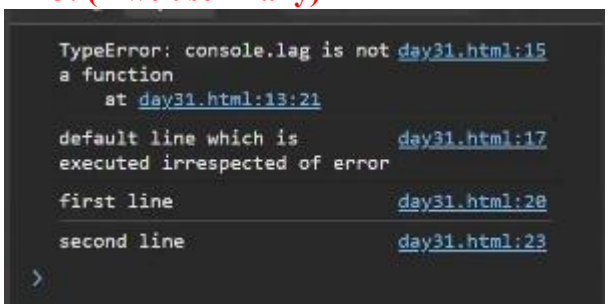
We can also use

console.error(err)

Ex-2

```
try{
   console.lag("hi hello");
}
catch(err){
  console.error(err);
}
finally{
   console.log("this will always run");
}
console.log("you have reached the end");
```

Ex-3: (if we use finally)



Throw statement in js

The throw statement is used to raise an exception in JavaScript. When an exception is thrown, the **normal flow of code execution is stopped**, and **control is passed to the nearest enclosing catch block.** If no catch block is found, the script will terminate.

Example:

```
num = prompt("entry");
try {
  if (num <= 0) {
    throw new Error("The number must be positive.");
  }
}
catch (err) {
  console.error(err);
}
console.log("you have reached the end");
```

This program prompts the user to enter a number, checks if the number is positive, and handles errors if the number is not positive. It uses a try...catch block to manage potential exceptions and ensures that a final message is logged to indicate the end of the program.

Try Block:
if (num <= 0) { throw new Error("The number must be positive."); }
Condition: Checks if the input num is less than or equal to 0.
Error Handling: If the condition is true, an error is thrown with the message "The number must be positive."

Catch Block:
catch (err) { console.error(err); }
Function: Catches the error thrown in the try block.
Error Logging: Uses console.error(err) to log the error object to the console, which includes the error message and stack trace.

Final Log Statement:
console.log("you have reached the end");
Purpose: To indicate that the program has reached its end, regardless of whether an error occurred.

Types of errors

1. **Syntax Errors**

Description: Occur when the code contains **invalid syntax**, which prevents the script from being parsed correctly.

```
if (true {
  console.log("This is a syntax error");
}
```

2. Reference Errors

Description: Occur when trying to reference a **variable that is not declared**.

```
console.log(nonExistentVariable);
```

3. Type Errors

Description: Occur when an operation is performed on a **value of an inappropriate type**

```
let num = 5;
num.toUpperCase(); // TypeError: num.toUpperCase is not a function
```

4. Range Errors

Description: Occur when a numeric variable or parameter is **outside its valid range**

```
  let arr = new Array(-1);
function createArray(size) {
  if (size > 100) {
    throw new RangeError("Array size is too large");
  }
  return new Array(size);
}
createArray(101);
```

5. Eval Errors

Description: Occur due to improper use of the **eval() function**. This error type is **rarely encountered** and is primarily included for **backward compatibility**.

```
eval("alert('Hello)");// This example won't necessarily throw an EvalError in modern browsers but shows misuse of eval.
```

6. URI Errors

Description: Occur when global **URI handling functions are used incorrectly**, such as encodeURI(), decodeURI(), encodeURIComponent(), and decodeURIComponent().

```
decodeURIComponent("%");
```

This keyword

`this` is a keyword that refers to the **current calling object** or context in which it is used. It is used to access the properties and methods of the current object.

1.Global Context: When this is used in the global scope (outside of any function), it refers to the global object. In a web browser environment, the global object is window.

```
  console.log(this === window); // true
```

Function Context: When this is used within a function that is not a method of an object, its value depends on how the function is called. If the function is called as a standalone function, this will typically refer to the global object (or undefined in strict mode).

```
 function sayHello() {
   console.log(this);
}
sayHello(); // In a browser, this would log the window object
```

3.Method Context: When this is used within a method of an object, it refers to the object that the method is called on.
//here this refers to the person means object name

```
 const person = {
   name: 'John',
   greet: function() {
      console.log('Hello, my name is ' + this.name);
   }
};
person.greet(); // Logs "Hello, my name is John"
```

This in arrow function

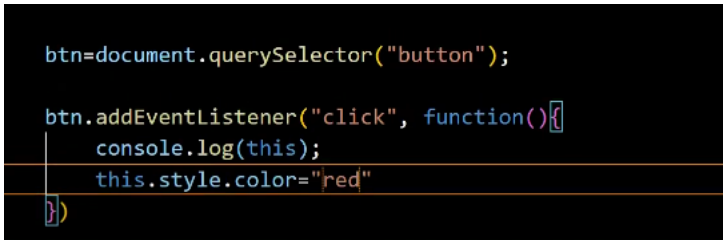
Ex-1:

```
let obj={
   name:"John",
   age:30,
   sayHello:()=>{
      console.log(this.age) //undefined
   }
}
obj.sayHello()
```

Ex-2:

```
let obj={
   name:"John",
   age:30,
   sayHello:()=>{
      console.log(this)// window
   }
}
obj.sayHello()
```

Understanding this in Arrow Functions
Arrow functions (() => { }) in JavaScript behave differently with respect to this compared to regular functions.



Here are the key points to note:

Lexical Scope: Arrow functions do not have their own this context. Instead, they inherit this from the surrounding (lexical) scope where they are defined.

Global Object: In most cases where obj is defined at the top level (not inside another function or block), this refers to the global object (window in browsers).

4.This refers to the particular event in evenhandlers

```
document.getElementById('myButton').onclick = function() {
console.log(this); // Refers to the button element with the ID 'myButton'
  };
```

Changing this with call, apply, and bind

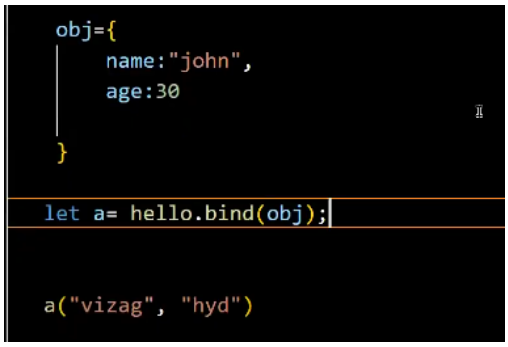
call and apply immediately invoke the function with a specified this value and arguments.

```
function showThis() {
  console.log(this);
}
const obj = { name: "John" };

showThis()// window
showThis.call(obj); // obj
showThis.apply(obj); // obj
```

bind

bind returns a new function with a specified this value, without immediately invoking the function.

```
const boundFunction = showThis.bind(obj);
boundFunction(); // obj
```



call with parameters

```
function hello(city, profession) {
    console.log("hello my name is " + this.name + " iam from " + city + " my profesion is " + profession);
}

obj = {
  name: "john",
};
obj2 = {
  name: "peter",
};

hello.call(obj, " vizag", " senior trainer");
hello.call(obj2, "hyd", " developer ");
```

apply

```
function hello(city, profession) {
```

```
  console.log( "hello my name is " + this.name + " iam from " + city + " my profesion is " + profession);
}

obj = {
  name: "john",
};
obj2 = {
  name: "peter",
};

hello.apply(obj, [" vizag", " senior trainer"]);
hello.apply(obj2, ["hyd", " developer "]);
```

bind

```
function hello(city, profession) {
  console.log( "hello my name is " + this.name + " iam from " + city + " my profesion is " + profession );
}

obj = {
  name: "john",
};
var bind1 = hello.bind(obj);
bind1(" vizag", " senior trainer");
```

Key Differences and Use Cases
call vs apply: The primary difference between call and apply is how arguments are passed:
call passes arguments individually.
apply expects an array of arguments.
Use call when you know the exact arguments to pass, and use apply when you have arguments in an array or array-like object.
bind: Unlike call and apply, bind does not immediately invoke the function. Instead, it creates a new function with the bound this value and optionally prepended arguments. This is useful when you want to create a function with a fixed this value that you can later execute.

Constructor function

It is a old way to create objects
Single same objects

```
function Person(){
    this.name = "john";
    this.age = 30;
}

let person1=new Person();
console.log(person1);

let person2=new Person();
console.log(person2);-
```

Multi objects

```
// constructor function - it is a old way to create objects in js
function Person(name, age){
    this.name = name;
    this.age = age;
}

let person1=new Person("Hemanth", "26");
console.log(person1);
let person2=new Person("teja", "25");
```

```
console.log(person2);
```

Prototype

```
function Person(){
    this.name = "john";
    this.age = 30;
}

Person.prototype.course="trainers"

let person1=new Person();
console.log(person1.course);
let person2=new Person();
console.log(person2.course)
```


```
function Person(name, age) {
  this.name = name;
  this.age = age;
}

function Student(name, age, grade) {
  // Using call to inherit the properties from Person
  Person.call(this, name, age);
  this.grade = grade;
}

const student1 = new Student('John', 20, 'A');
console.log(student1); // Output: { name: 'John', age: 20, grade: 'A' }
```

Javascript classes

The old way to create objects was using constructor functions.

```
// constructor function
function Person () {
  this.name = "John",
  this.age = 23
}

// create an object
const person = new Person();
```

Topic: Oops (Object Oriented Programming Structure)

Oops in js

OOP in JavaScript provides a **way to structure and organize code** in a more modular and **reusable** manner. By utilizing objects, classes, inheritance, encapsulation, and polymorphism

1.class
2.object
3.encapsulatiion
4.inheritance
5.polymorphism
6.abstraction

Constructor class

Construtor class is a way to define a template for creating object
class was introduced recently in ES6
class is a collection of properties and methods(static/intance)
syntax :

```
class Classname{
}
```

obj syntax :
new Classname(arguments)
constructor should be one

Class Definition

- **Class**: A class is a template for creating objects. It encapsulates data and functions that operate on that data.
- **Properties (state)**: Variables that belong to the class.
- **Methods (behaviour)**: Functions that belong to the class.

Ex:- Marker Class

The **Marker** class has three properties: **color, lid**, and **type**. It also has a constructor to initialize these properties.

```
class Marker {
 color; // property to hold the color of the marker
 lid; // property to hold the type of lid of the marker
 type; // property to hold the type of the marker

 // Constructor to initialize the properties
 constructor(color,lid, type) {
  this.color = color; // initializes the color property
  this.lid = lid; // initializes the lid property
  this.type = type; // initializes the type property
 }
 // method initialisation - it is a behaviour of object
 write() {
    console.log("Writing with " + this.color + " marker");
 }
}
```

Creating Objects

Objects are instances of a class. The new keyword is used to create an object from a class.

```
var marker1 = new Marker("red", "square", "permanent");
var marker2 = new Marker("black", "circle", "temporary");
```

Logging the object

```
//logging the objects
console.log(marker1);
console.log(marker2);
```

Calling the method

```
//calling the methods
marker1.write();
```

Key Points

- **Class Definition**: Defines the state and behavior of objects.
- **Properties**: Variables within a class that hold data.
- **Methods** : function that holds the behaviour
- **Constructor**: Special method to initialize properties when an object is created.
- **this Keyword**: Refers to the current instance of the class.
- **Creating Objects**: Use new keyword followed by class name and arguments to create objects.

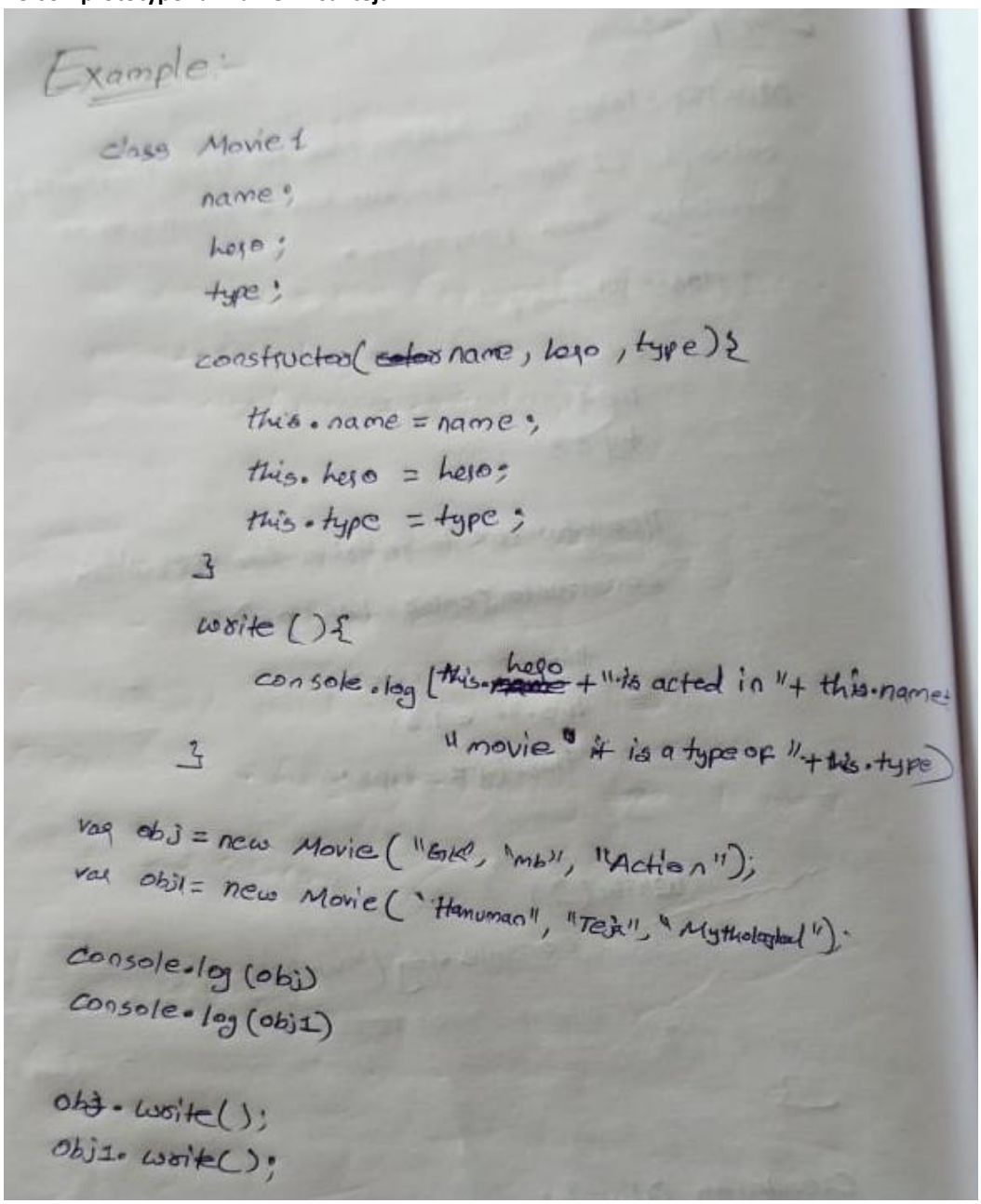
use the class keyword to create a JavaScript class.

```
// create a class
class Person{
  // body of class
};
```

To use the class constructor we must need to assign object to it

```
class Person{
  constructor (name,age){
    this.name=name;
    this.age =age;
  }

  hi(a,b){
    return "my name is "+ this.name+ " age is " + this.age;
  }
}
var b=new Person("teja",25);
console.log(b);//{name:teja; age:25}
console.log(b.age);//25

console.log(b.hi());//my name is teja age is 25
```

by using prototypes we can add properties directly
Person.prototype.fullName = “sai teja”.



Java script classes with private fields

Class Definition

- **Class**: Employee
  - **Private Fields**: #name, #age, #salary (indicated by # prefix)
  - Introduced to provide encapsulation and data protection within a class.
  - Enhances security and prevents accidental modification of sensitive data.

```
class Employee {
#name;
#age;
#salary;
```

```
  constructor(name, age, salary) {
    this.#name = "John";    // Private field initialization
    this.#age = 30;         // Private field initialization
    this.#salary = 50000;   // Private field initialization
  }

//private fields can only accessed inside the class
 hello(){
    console.log("Hello, my name is " + this.# )
    }
 }
```

Private Fields

- **Definition**: Private fields are declared using the # symbol before the field name (#name, #age, #salary).
- **Access Control**: Private fields can only be accessed and modified from inside the class where they are defined.

Accessing object will thrown an error because we cannot able to access them in outside

```
var person1=new Employee("john", 25, 50000);
console.log(person1.name)// error - private property cannot be accessed directly
```

method will print the output because properties can be accessed inside a class

```
person1.hello();
```

Encapsulation

Encapsulation in JavaScript refers to the **bundling of data (attributes) and methods (functions) that operate on the data into a single unit**, typically known as an object. This concept helps in hiding the internal state of an object and only exposing the necessary functionalities to interact with that state.

Access the values using getter

Syntax:

```
// we can giving access
get methodname(){
  return this.#name;
}
```

```
// we can giving access
get accessname(){
  return this.#name;
}
```

```
class Employee {
#name;
#age;
#salary;

constructor(name, age, salary) {
    this.#name = name;    // Private field initialization
    this.#age = age;        // Private field initialization
    this.#salary = salary;   // Private field initialization
}
hello(){
    console.log("Hello, my name is " + this.#name + " and I am a " )
}

// Getter for accessing private field #name
get accessname(){
  return this.#name;
}
}
```

```
var person1=new Employee("john", 25, 50000);
//we can able to access now because of encapsulation
console.log(person1.accessname)// john - will not thrown an error
```

Modify the values using setter

```
// Setter for modifying private field #salary
set setsalary(sal){
  this.#salary=sal;
}

// Getter for accessing private field #salary
get getsalary(){
  return this.#salary;
}
```

```
//creating a object
var person1=new Employee("john", 25, 50000);

//setting the values
person1.setsalary=99500;

//accessing the updated values
console.log(person1.getsalary);
```

//complete program

```
class Employee {
#name;
#age;
#salary;

constructor(name, age, salary) {
   this.#name = name;   // Private field initialization
   this.#age = age;       // Private field initialization
   this.#salary = salary;   // Private field initialization
}
hello(){
   console.log("Hello, my name is " + this.#name + " and I am a " )
}

// we can giving access
get accessname(){
  return this.#name;
}

//giving the access to change the values
set setsalary(sal){
  this.#salary=sal;
}

get getsalary(){
  return this.#salary;
}
}

//creating a object
var person1=new Employee("john", 25, 50000);

//setting the values
person1.setsalary=99500;

//accessing the updated values
console.log(person1.getsalary);
```

Certainly! Let's break down the concept of inheritance in JavaScript using the provided code as an example and create detailed notes.

Topic: OOPS (Inheritance, Polymorphism and abstraction)

JavaScript Inheritance

What is Inheritance?

- **Inheritance**: A feature in object-oriented programming that allows one class (subclass or derived class) to inherit properties and methods from another class (superclass or base class).
- **Purpose**: Promotes code reusability and establishes a natural hierarchy between classes.

Base Class: Animal

The Animal class serves as the base class with common properties and methods that can be inherited by other classes.

```
class Animal{
  weight;
  height;
  voice;

  constructor(weight, height, voice){
    this.weight = weight;
    this.height = height;
    this.voice = voice;
  }

  eating(){
    console.log("Eating...");
  }
}
```

Subclass: Dog

The Dog class extends the Animal class, inheriting its properties and methods while adding its own specific properties.

```
class Dog extends Animal {
  breed;
  food;

  constructor(weight, height, voice, breed, food) {
    super(weight, height, voice); // Calls the constructor of the Animal class
    this.breed = breed;
    this.food = food;
  }
}
```

Properties:

- Inherited: weight, height, voice
- Specific: breed, food

Constructor:

- Calls super(weight, height, voice) to initialize inherited properties.
- Initializes breed and food properties.

Creating Objects and Using Methods

```
var dog1 = new Dog("15kgs", "2feet", "bow bow", "indie", "nonveg");
console.log(dog1); // Outputs: Dog { weight: '15kgs', height: '2feet', voice: 'bow bow', breed: 'indie', food: 'nonveg' }
dog1.eating(); // Outputs: Eating...
```

Notes

1. **Inheritance in JavaScript**:
    - **Base Class**: Defines common properties and methods.
    - **Subclass**: Inherits from the base class and can add or override properties and methods.
2. **super Keyword**:
    - Used in the constructor of the subclass to call the constructor of the base class.
    - Ensures inherited properties are properly initialized before accessing subclass-specific properties.
3. **Constructor Rules**:
    - In a subclass constructor, super must be called before accessing this.
    - Ensures proper initialization of the inherited fields.
4. **Method Inheritance**:
    - Subclasses inherit all methods from the base class.
    - Methods can be overridden in the subclass if needed.
5. **Code Reusability**:
    - Inheritance promotes code reusability by allowing subclasses to use and extend the functionality of base classes.
    - Reduces redundancy and simplifies maintenance.
6. **Example Scenario**:
    - **Base Class**: Animal with common properties (weight, height, voice) and methods (eating).
    - **Subclasses**: Dog and Cat with additional specific properties (breed, food for Dog, and color for Cat).

What is Polymorphism?

- **Polymorphism**: A core concept in object-oriented programming that allows objects of different classes to be treated as objects of a common superclass. It enables methods to perform different tasks based on the object they are called on.
- **Purpose**: Enhances flexibility, reusability, and maintainability of code by allowing a single interface to represent different underlying forms (data types).

Example: Polymorphism in Action
Using the provided code as an example, we can see how polymorphism works in JavaScript

Base Class: Animal
The Animal class serves as the base class with common properties and methods that can be inherited by other classes.

```
class Animal {
  weight;
  height;
  voice;

  constructor(weight, height, voice) {
    this.weight = weight;
    this.height = height;
    this.voice = voice;
  }

  eating() {
    console.log("Eating...");
```

```
  }
}
```

Properties: weight, height, voice

Methods:

- eating(): Prints "Eating..." to the console.

Subclass: Dog

The Dog class extends the Animal class, inheriting its properties and methods but also overriding the eating method.

```
class Dog extends Animal {
  breed;
  food;
  constructor(weight, height, voice, breed, food) {
    // we cannot write anything above the super
    super(weight, height, voice);
    this.breed = breed;
    this.food = food;
  }
  eating() {
    console.log("Dog is eating " + this.food);
  }
}
```

Properties:

- Inherited: weight, height, voice
- Specific: breed, food

Methods:

- Overridden: eating(): Prints "Dog is eating " followed by the type of food the dog eats.

Object Creation and Method Overriding

```
var dog1 = new Dog("15kgs", "2feet", "bow bow", "indie", "nonveg");
console.log(dog1);
// Outputs: Dog { weight: '15kgs', height: '2feet', voice: 'bow bow', breed: 'indie', food: 'nonveg' }

dog1.eating();
// Outputs: Dog is eating nonveg
```

Object Creation: dog1 is created as an instance of the Dog class.

Method Overriding: The eating method in the Dog class overrides the eating method in the Animal class.

Polymorphism

Polymorphism, in the context of object-oriented programming, refers to the ability of **different objects to respond to the same message or method invocation** in different ways. It allows objects of different classes to be treated as objects of a common superclass, providing a unified interface for different classes.

Abstraction

Abstraction in JavaScript refers to the concept of hiding complex implementation details and showing only the necessary features of an object or function. It allows developers to work with high-level representations without needing to understand all the underlying complexities.

Topic: ES -6 concepts

ES -6 concepts

ES6, or ECMAScript 2015, brought significant enhancements to JavaScript, introducing many new features and syntax improvements.

let and const:

- **let** allows declaring block-scoped variables that can be reassigned.
- **const** declares constants whose values cannot be reassigned.

Arrow Functions:

Provides a concise syntax for writing functions, with an implicit return if the function body is a single expression.

```
const add = (a, b) => a + b;
```

Template Literals:

Allow embedding expressions inside strings using ${} syntax.

```
const name = "John";
console.log(`Hello, ${name}!`);
```

Destructuring Assignment:

Allows extracting values from arrays or objects into distinct variables.
Destructuring is a process assigining a value to a single varaible

Types

Array

Object

Spread Syntax:

- Allows expanding iterable elements in places where zero or more arguments or elements are expected.it creates the deep copy

```
const arr = [1, 2, 3];
const newArr = [...arr, 4, 5];
```

Rest Parameters:

* Rest operator is introduced in ES6

* Rest operator are used in function parameters

* Rest operator is denoted by ...

* Rest operator is implicitly an array

* Rest operator will accepts from 0 to n number of arguments

* Rest operator should be the last parameter of function

* In Functions parameters rest operators can be only one

* Rest Operator can be used in array/ object destructuring

* Rest Operator is used for better readibility

* **Syntax :**

* ...varName (passed as function parameter)

Allows representing an indefinite number of arguments as an array.

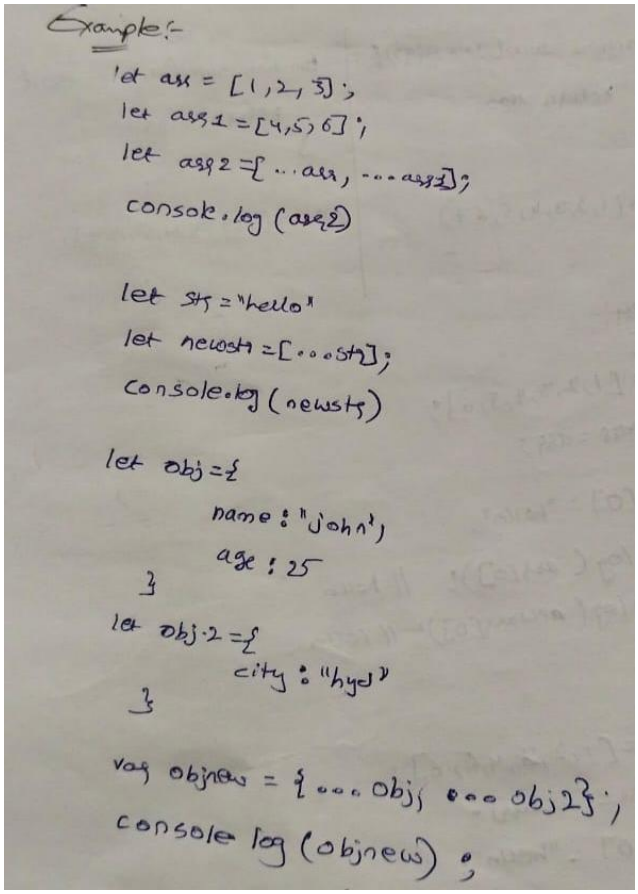
```
function sum(a,b,...args) {
   return …args
}
Sum(1,2,3,4,5,6,7)
```

Classes:
Modules:
Promises:
Async/Await:

Spread operator

spread operator (...) in JavaScript is a powerful tool used to expand elements of an iterable (like arrays, strings, or objects) into individual elements. It's often used for array manipulation, function arguments, and object spreading.

* Spread operator is used to spread the data
* which is present in array and string
* Spread operator is used as function argument
* Spread operator will create deep copy for array and object
* S+6++pread is denoted by ...
* for array [...arrRef] , for object {...objectRef}



Expanding an Array:

```
const arr1 = [1, 2, 3];
const arr2 = [...arr, 4, 5, 6];
console.log(arr2); // Output: [1, 2, 3, 4, 5, 6]
```

Passing Arrays as Function Arguments:

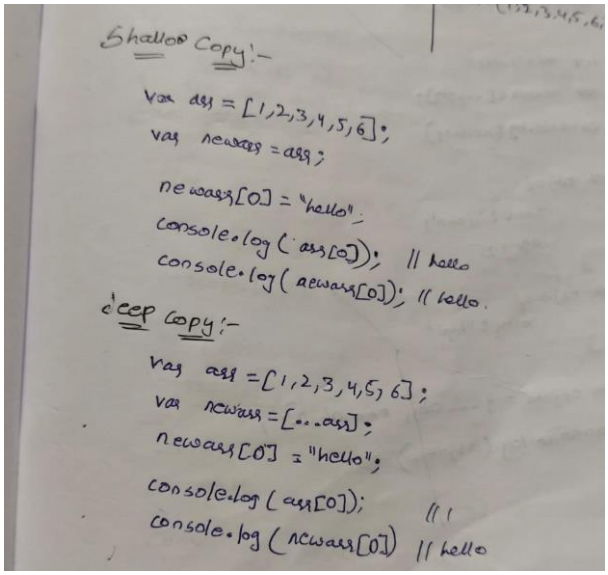
```
const numbers = [1, 2, 3, 4, 5];
const sum = (a, b, c, d, e) => a + b + c + d + e;
console.log(sum(...numbers)); // Output: 15
```

Merging Objects:

```
const obj1 = { name: 'john' };
const obj2 = { age: 23 };
const mergedObj = { ...obj1, ...obj2 };
console.log(mergedObj); // Output: { name: 'john', age: 23 }
```

shallow copy with the example

```
const originalArray = [1, 2, 3, 4, 5];
const shallowCopyArray = [...originalArray]; // Shallow copy using the rest operator
shallowCopyArray[0] = 10; // Modify the shallow copy

console.log(originalArray); // Output: [1, 2, 3, 4, 5]
console.log(shallowCopyArray); // Output: [10, 2, 3, 4, 5]
```



Why spread operator introduced

```
var a=[12,2,2,22,]  //create an array
var b=a;  //assigned a array to another variable
b[0]=500; //change the second variable value
console.log(a);//500,2,2,22
```

here value of array a is also changed because here value is not assigned . memory location is assigned in order to overcome this spread operator was introduced.

In JavaScript, when we assign an array or object to another variable, we do not create a new independent copy of the original array or object. Instead, we assign a reference to the original memory location. This means that any changes made to the new variable will also reflect in the original array or object.

Understanding Shallow Copy

The behavior observed in the above example is due to shallow copying, where only the reference to the original memory location is copied, not the actual data. To overcome this and create an independent copy of the array, we can use the spread operator or other methods

By using spread operator

```
var a=[12,2,2,22,]  //create an array
var b=[...a];  //create a deep copy of an array using spread operator
b[0]=500; //change the second variable value
console.log(a);//12,2,2,22    --here value of a is not changed
console.log(b);//500,2,2,22   --value of b is only changed because spread operator creates a copy of an array
```

Call by value and call by reference

understanding the difference between call by value and call by reference is crucial for managing how data is passed and manipulated within your code

Call by Value:

Call by Value means that when a variable is passed to a function, the value of the variable is passed. If the variable is a primitive type (like number, string, boolean, null, undefined, symbol, and bigint), its value is copied to the function parameter. Changes to the parameter do not affect the original variable.

```
function changePrimitive(val) {
   val = 100;
}
let num = 50;
changePrimitive(num);
console.log(num); // Output: 50
```

num remains 50 even after calling changePrimitive because the function works with a copy of num.

Call by Reference

Call by Reference means that when a variable is passed to a function, a reference to the variable is passed. If the variable is an object (like arrays, functions, objects, etc.), its reference is passed to the function. Changes to the parameter will affect the original object.

```
function changeObject(obj) {
   obj.name = "John";
}

let person = { name: "Doe" };
changeObject(person);
console.log(person.name); // Output: John
```

the name property of the person object changes to "John" because the function works with a reference to the person object.

Key Points

1. **Primitive Types (Call by Value)**:
    - number, string, boolean, null, undefined, symbol, bigint
    - Changes inside the function do not affect the original variable.
2. **Reference Types (Call by Reference)**:
    - object, array, function
    - Changes inside the function affect the original object or array.

Destructuring

Destructuring in JavaScript is a powerful feature that allows you to extract values from arrays or properties from objects and bind them to variables in a concise and expressive way. It provides a convenient syntax for extracting data from arrays and objects.

Array Destructuring:

1. **We can destructure values individually**
2. **We can can skip elements in the array by using commas.**
3. **If the value at the specified position is undefined, you can assign a default value.**
4. **We can use rest operator (...) to collect the remaining elements into a new array.**
5. **We can destructure nested arrays**

```
const numbers = [1, 2, 3, 4, 5];
const [first, second, ...rest] = numbers; // Extracting values from the array into variables
console.log(first); // Output: 1
console.log(second); // Output: 2
console.log(rest); // Output: [3, 4, 5]
```

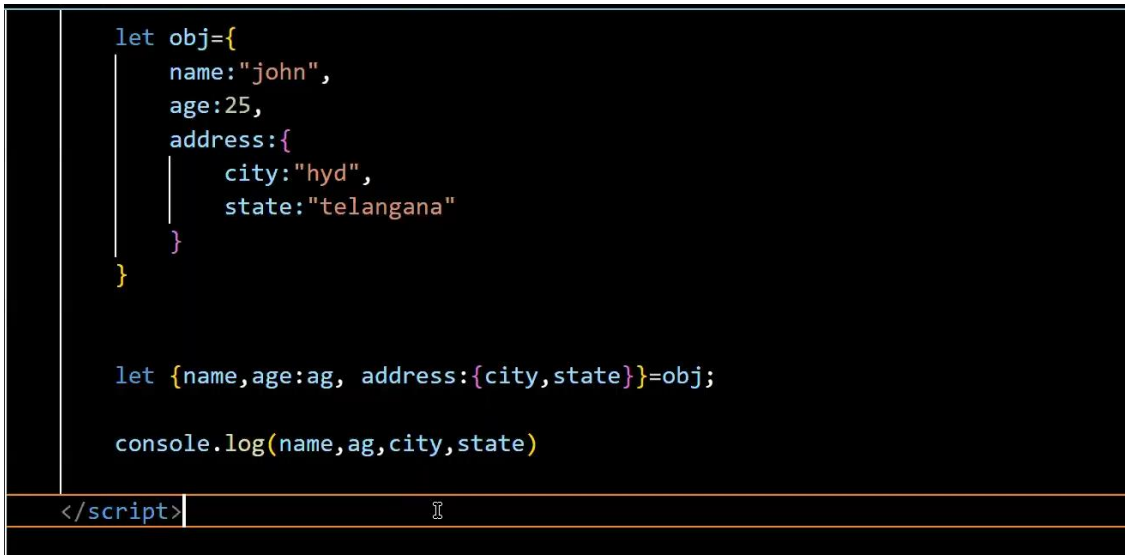
Object Destructuring:

1. **We can destructure values individually**
2. **We can rename variables while destructuring by using a colon (:).**
3. **We can set default values for properties that might be undefined.**
4. **Destructuring can be used to extract values from nested objects.**
5. **We can pass remaining values using rest operator**

const person = { name: 'John', age: 30, city: 'New York' };
const { name, age } = person; // Extracting properties from the object into variables
console.log(name); // Output: 'John'
console.log(age); // Output: 30

Nested objects destructuring

const person = { name: 'John', age: 30, address: { city: 'New York', country: 'USA' } };
const { name, address: { city, country } } = person; // Nested destructuring
console.log(name); // Output: 'John'
console.log(city); // Output: 'New York'
console.log(country); // Output: 'USA'



Modules :-

modules are reusable pieces of code that encapsulate related functionality and can be exported from one file and imported into another. This allows for better organization, maintainability, and reusability of code in large applications. ES6 (ECMAScript 2015) introduced native support for modules.

Note:- mention type=”module” in script tag to work with the modules

```
<script type="module">
import message from "./message.js";
</script>
```

There are two types of exports: **Named Exports** and **Default Exports**.

1)Named export

```
const name = "Jesse";
const age = 40;

export {name, age};
```

Named import

```
import { name, age } from "./person.js";
```

2)Default Export

```
export default  message = () => {
const name = "Jesse";
const age = 40;
return name + ' is ' + age + 'years old.';
};
```

Default import

```
import message from "./message.js";
```

sets and maps

Sets in JavaScript

A Set is a collection of unique values. This means that no value can occur more than once in a set. Sets allow you to store distinct values of any type, whether primitive or object references.

Key Characteristics of Sets:

- **Unique Values:** No duplicates are allowed. If you try to add the same value multiple times, it will only be added once.
- **Order of Values:** Insertion order is maintained. The elements are iterated in the order of their insertion.

- **No Key-Value Pairs:** Unlike objects or maps, a set is just a collection of values (no keys).

Common Methods for Sets:

1. **add(value):** Adds a new value to the set.
2. **delete(value):** Removes the specified value from the set.
3. **has(value):** Checks if the value exists in the set.
4. **clear():** Removes all values from the set.
5. **size:** Returns the number of values in the set.
6. **forEach():** Iterates over the values in the set.
7. **values():** Returns an iterator over the set's values.
8. **keys():** Identical to values(), as sets don't have keys.
9. **entries():** Returns an iterator of [value, value] pairs, mimicking the [key, value] behavior of maps.

Example:

```
let mySet = new Set();

  // Add values
  mySet.add(1);
  mySet.add(2);
  mySet.add(2);  // Won't be added, as 2 is already present
  mySet.add('hello');

  // Check if a value exists
  console.log(mySet.has(1));  // true
  console.log(mySet.size);    // 3

  // Remove a value
  mySet.delete(2);

  // Iterate over the set
  mySet.forEach(value => console.log(value));
```

Maps in JavaScript

A Map is a collection of key-value pairs, similar to an object. However, maps have some key differences that make them more useful in specific scenarios.

Key Characteristics of Maps:

- Key-Value Pairs: Each element in a map is stored as a pair of keys and values.
- Any Data Type as Keys: Unlike objects, where keys are strings or symbols, in maps, any type (primitive or object) can be used as a key.
- Map Size: You can easily determine the number of entries in a map using the size property.

Common Methods for Maps:

1. **set(key, value):** Adds a key-value pair to the map or updates an existing key.
2. **get(key):** Returns the value associated with the key.
3. **has(key):** Checks if the map contains the specified key.
4. **delete(key):** Removes the key and its associated value from the map.
5. **clear():** Removes all key-value pairs from the map.
6. **size:** Returns the number of key-value pairs in the map.
7. **forEach():** Iterates over the key-value pairs in the map.
8. **keys():** Returns an iterator over the keys of the Map.
9. **values():** Returns an iterator over the values of the Map.
10. **entries():** Returns an iterator over [key, value] pairs.

Example:

```
let myMap = new Map();

// Add key-value pairs
myMap.set('name', 'John');
myMap.set(1, 'one');
myMap.set(true, 'boolean value');

// Get values
console.log(myMap.get('name'));  // 'John'
console.log(myMap.get(1));       // 'one'

// Check for existence of a key
console.log(myMap.has(true));    // true

// Remove a key-value pair
myMap.delete('name');

// Iterate over the map
myMap.forEach((value, key) => {
console.log(key, value);
});
```

References:

1. https://www.w3schools.com/js/
2. https://drive.google.com/drive/u/0/folders/1571YaKXWXAxR7Q40k-jTQ_-4lUnBgFAB
3. https://abhinavsai-12.github.io/10kCoders_JavaScript/
4. https://github.com/Abhinavsai-12/10kCoders_JavaScript
5. https://github.com/saiteja-yernagula/javascript-mini-projects
6. https://docs.google.com/spreadsheets/d/1jxyvCelUXl22DhMAvKHIoK0vTIbgv9Ct9_j9-lVXQF0/edit?gid=0#gid=0