Introduction to Programming

Programming is a way to talk to the computer
A Langauge like kannada, English, Hindi can be used to talk to human same-way to talk to computer we need straight forward instructions that is we called programming

JAVASCRIPT

JavaScript is a programming language commonly used to add dynamic and interactions to web pages, applications

3 WAYS To import JS

1. External
2. Internal
3. Inline

JS ENGINE

the browsers have inbuilt JavaScript engine which help to convert our JavaScript program into computer-understandable language.

Browser	**Name of Javascript Engine**
Google Chrome	V8
Edge (Internet Explorer)	Chakra
Mozilla Firefox	Spider Monkey
Safari	Javascript Core Webkit

JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

Single Line Comments

Single line comments start with `//`

Multi-line Comments

Multi-line comments start with `/*` and end with `*/`

VARIABLES

A variable is a container that stores a value
Ex: var a; //declaring variable
a= 5; //assigning a value to variables
a // identifier
= //assignment operator

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter ,$ and _ .
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

```
Ex: let person = "John Doe", carName = "Volvo", price = 200;
```

JavaScript Let

The `let` keyword was introduced in (2015)

Variables defined with `let` cannot be **Redeclared**

Variables defined with `let` must be **Declared** before use

Variables defined with `let` have **Block Scope**

JavaScript Const

the `const` keyword was introduced in ES6 (2015)

Variables defined with `const` cannot be **Redeclared**

Variables defined with `const` cannot be **Reassigned**

Variables defined with `const` have **Block Scope**

Always declare a variable with `const` when you know that the value should not be changed.

Console log

The console.log () is a used for debugging purpose in JavaScript & which helps to print any kind of variables defined before that needs to be displayed to the user.

JavaScript Data Types

JavaScript has 7 Datatypes

1. String
2. Number
3. Boolean
4. Undefined
5. Null
6. Bigint
7. Object --> object, Array, Date

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

```
Ex: let carName1 = "Volvo XC60";
```

JavaScript Numbers

All JavaScript numbers are stored as decimal numbers (floating point).

Numbers can be written with, or without decimals:

Ex:
```
// With decimals:
let x1 = 34.00;

// Without decimals:
let x2 = 34;
```

JavaScript Booleans

Booleans can only have two values: `true` or `false`.

Ex: const a = true;

JAVASCRIPT undefined

The undefined property indicates that a variable has not been assigned a value, or not declared at all.

JavaScript Null

null expresses a lack of identification, indicating that a variable points to no object

Ex: const a = null;

JavaScript BigInt

JavaScript integers are only accurate up to 15 digits:

All JavaScript numbers are stored in a a 64-bit floating-point format.

Ex: let x = 99999999999999999;

let y = BigInt("9999999999999999999")

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

Ex: `const cars = ["Saab", "Volvo", "BMW"];`

JavaScript Objects

JavaScript objects are written with curly braces `{}`.

Object properties are written as name:value pairs, separated by commas.

```
Ex: const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators

Arithmetic Operators

Arithmetic Operators are used to perform arithmetic on numbers:

Ex: `let x = (100 + 50)`

Arithmetic operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Assignment Operators

Assignment operators assign values to JavaScript variables.

Ex: `let x = 10;`

`= //assignment operator`

The **Addition Assignment Operator** (`+=`) adds a value to a variable.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

String operator

```
let text1 = "What a very ";
text1 += "nice day";
let text3 = text1 + " " + text2;
```

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string: this we called as type Coercion

Type Coercion

conversion is the process of converting data of one type to another.

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

Ternary Operator

```
Ex: let result = (10 > 0) ? true : false;
```

Type Operator

You can use the JavaScript `typeof` operator to find the type of a JavaScript variable.

```
console.log(typeof 42);
console.log(typeof 'blubber');
```

Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1

```
// bitwise AND operator example
let a = 12; let  b = 25;

result = a & b; console.log(result); // 8
```

```
12 --> 1100,  25 --> 11001
```

String Built-in Methods

Length

Ex:
```
let text = "Hello World!";
let length = text.length;
```

charAt

```
let text = "HELLO WORLD";
let letter = text.charAt(0);
let letter = text.charAt(text.length-1);
```

concat()

```
let text1 = "sea";
let text2 = "food";
let result = text1.concat(text2);
let result = text1.concat(" ", text2);
```

includes()

The `includes()` method returns `true` if a string contains a specified string.

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

Repeat()

The `repeat()` method returns a string with a number of copies of a string.

```
let text = "Hello world!";
let result = text.repeat(2);
```

Replace()

```
let text = "Visit Microsoft!";
let result = text.replace("Microsoft", "W3Schools");
```

indexOf()

--> returns -1 if the value is not found.

```
let text = "Hello world, welcome to the universe.";
let result = text.indexOf("welcome");
```

lastIndexOf()

--> returns -1 if the value is not found.

```
let text = "Hello planet earth, you are a great planet.";
let result = text.lastIndexOf("planet");
```

Trim()

Remove spaces:

```
let text = "       Hello World!        ";
let result = text.trim();
```

LowerCase() & toUpperCase()

```
let text = "Hello World!";
let result = text.toUpperCase();
let result = text.toLowerCase();
```

JavaScript Arrays

An array is a special variable, which can hold more than one value:

Collection of some items

```
const cars = ["Saab", "Volvo", "BMW"];
const cars = new Array("Saab", "Volvo", "BMW");
```

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";
```

Array Methods

Array length
Array totring()
Array join()
Array concat()

Array pop()
Array push()
Array shift()
Array unshift()
Array delete()

Array slice()
Array splice()

Array length

The `length` property returns the length (size) of an array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let size = fruits.length;
```

Array toString()

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const res = fruits.toString();
```

Array `join`()

The `join()` method also joins all array elements into a string.

It behaves just like `toString()`, but in addition you can specify the separator:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const res = fruits.join("*");
```

Concatenating Arrays

The `concat()` method creates a new array by merging (concatenating) existing arrays:

```
const firstArr= ["Cecilie", "Lone"];
const secArr= ["Emil", "Tobias", "Linus"];


const myChildren = firstArr.concat(secArr);
```

Array pop()

The `pop()` method removes the last element from an array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

Array push()

The `push()` method adds a new element to an array (at the end):

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

Array shift()

The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

Array unshift()

The `unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Array splice()

The `splice()` method can be used to add new items / remove items to an array:

```
splice(start, delete count, optional items to add)
```

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

Array slice()

The `slice()` method can be used to remove items to an array:

```
slice(optional start parameter, optional end parameter)
```

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
```

```
const citrus = fruits.slice(1);
```

```
const citrus = fruits.slice(1, 3);
```

Array delete()

Array elements can be deleted using the JavaScript operator `delete`. But leaves `undefined` holes in the array

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];
```

Object

Objects are variables too. objects can contain many values along with property name and value.

This code assigns **many values** (Fiat, 500, white) to a **variable** (named car)

```
const car = {type:"Fiat", model:"500", color:"white"};
```

```
objectName["propertyName"] = 'value';
```

```
//Ex:
let user = {
```

```
    firstName: 'Mickel',
    lastName: "John",
    gender: "Male",

}
console.log(user)
console.log(user.firstName)
console.log(user["firstName"])
user.firstName = 'carlos';
user.proffesinal = 'cricketer';
console.log(user)
delete user.gender;
console.log(user)
```

New Date()

Const todaysDate = new Date()

IF Else

if (condition1) {

  statement1

}else if (condition2){

  statement2

}else if (condition3)

  statement3

Else{

  statementN

}

Ex:

```
const number = prompt("Enter a number: ");


if (number > 0) {
    console.log("The number is positive");
}
else if (number == 0) {
  console.log("The number is 0");
}
else {
    console.log("The number is negative");
}
```

Ex: find largest 2 numbers
find largest 3 numbers

Switch

```
switch (expression) {
     case value1:
          statement1;
          break;
     case value2:
          statement2;
          break;
     case valueN:
          statementN;
          break;
     default:
          statementDefault;
}
```

```
Ex: const number = prompt("Enter a number: ");

switch (parseInt(number)) {
  case 1:
    console.log("The number is 1");
break;
  case 0:
    console.log("The number is 0");
  break;
  case -1:
    console.log("The number is -1");
  break;
  default:
    console.log("neither 1 or 0 or -1");
}
```

For Loop

```
for (initialization; checkcondition; update)

{

    // statements inside the body of loop

}
```

```
Ex: for (let i = 0; i < 5; i++) {
   console.log(i)
  }
const givenArr = [1,2,3,4,5]
for(let i=0; i<givenArr; i++){
    console.log(i, givenArr[i])
}
```

Ex: find odd , even numbers in array

while Loop

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

```
Ex: let i = 0;
while (i < 5) {
    console.log(i);
  i++;
}
```

do while Loop

The `do...while` statements combo defines a code block to be executed once, and repeated as long as a condition is `true`.

The `do...while` is used when you want to run a code block **at least one time**.

```
Ex: let i = 0;
do {
```

```
   console.log(i)
    i++;
  }
  while (i < 5);+
```

For of Loop

Iterate (loop) over the values of an array:

```
const cars = ['BMW', 'Volvo', 'Mini'];
for (let x of cars) {
  console.log(x)
} //array
```

For in Loop

Iterate (loop) over the properties of an object:

```
const person = {fname:"John", lname:"Doe", age:25};
let text = "";
for (let x in person) {
  text += person[x] + " ";
} //object
```

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can call that function.

With functions you can reuse code

Syntax:
function functionName(Parameter1, Parametern)

```
{
    // code to be executed
}
```

- Every function should begin with the keyword *function* followed by,
- A user-defined function name that should be unique,
- A list of statements composing the body of the function enclosed within curly braces {}.

Ex:

```
function addNum(number1, number2) {
   return number1 + number2;
}
```

To create a function in JavaScript, we have to first use the keyword *function and* The function declaration must have a function name.

In the above example, we have created a function named addNum and number1, number2 is a **parameter**

addNum(2, 3) //function call

here 2 and 3 are the arguments
the arguments 2 and 3 are assigned to x and y, respectively.

argument
A value provided as input to a function.
parameter
A variable identifier provided as input to a function.

//Create a function to calculate the sum of an array of numbers.

Events

An HTML event can be something the browser does, or something a user does.

Mouse events:

Event Performed	Event Handler	Description
click	**onclick**	**When mouse click on an element**
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	**onchange**	**When the user modifies or changes the value of a form element**

The HTML DOM (Document Object Model)

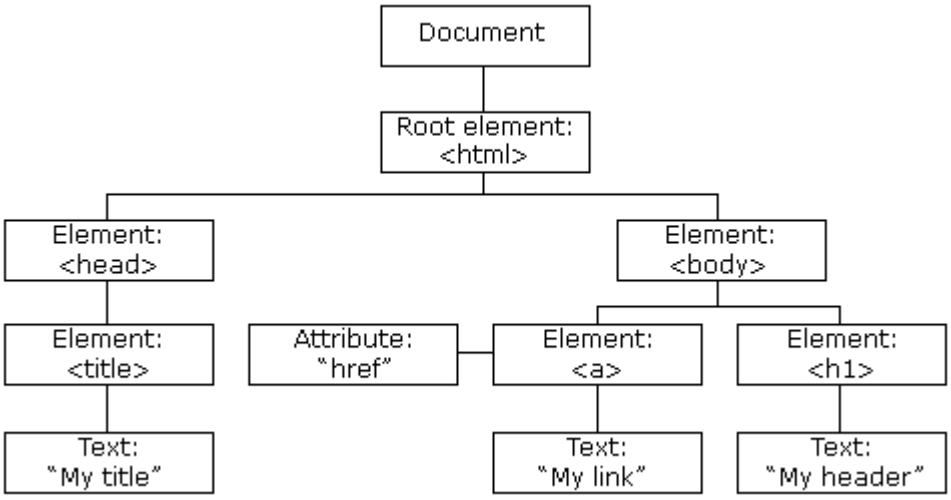
When an HTML file is loaded into the browser, the javascript can not understand the HTML document directly. So, a browser creates corresponding Document Object Model of the page(DOM).

DOM is basically the representation of the same HTML document but in a different format with the use of objects.

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The HTML DOM Tree of Objects



Methods of Document Object:

- write: Writes the given string on the document.
- getElementById(): returns the element having the given id value.
- getElementsByTagName(): returns all the elements having the given tag name.
- getElementsByClassName(): returns all the elements having the given class name.

getElementsByName(): returns all the elements having the given name value.

Changing HTML Elements

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element
element.setAttribute(attributeName, value)	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement(element)	Create an HTML element

document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

addEventListener

```
element.addEventListener("click", myFunction);
```

Create Elements using JS

1. createElement
2. createTextNode
3. appendChild

Ex:

```
const node = document.createElement("li");

const textnode = document.createTextNode("Water");

node.appendChild(textnode);

document.getElementById("myList").appendChild(node);
```

setAttribute

1. createAttribute
2. setAttributeNode

Ex: // Create a class attribute:

```
let att = document.createAttribute("class");

// Set a value of the class attribute
```

```
att.value = "democlass";

// Add the class attribute to the first h1;

document.getElementsByTagName("h1")[0].setAttributeNode(att);
```

querySelector

The querySelector() method returns the **first** element that matches a CSS selector.

```
document.querySelector(".example").style.backgroundColor = "red";
```

querySelectorAll

The querySelectorAll() method returns all elements that matches a CSS selector(s).

```
const nodeList = document.querySelectorAll(".example");
for (let i = 0; i < nodeList.length; i++) {
  nodeList[i].style.backgroundColor = "red";
}
```

JavaScript Math

The JavaScript Math object allows you to perform mathematical tasks on numbers.

1. Math.PI
2. Math.round(x) - returns the value of x rounded to its nearest integer
3. Math.ceil(x) - rounds a number up to its nearest integer
4. Math.floor(x) - rounds a number down to its nearest integer
5. Math.pow(8, 2);
6. Math.sqrt(64);
7. Math.random() //0.1 to 1
8. Math.min(0, 150, 30, 20, -8, -200);
9. Math.max(0, 150, 30, 20, -8, -200);

Number Methods

In the chapter Number Methods, you will find more methods that can be used to convert strings to numbers:

Method	Description
Number()	Returns a number, converted from its argument
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer

Ex: parseInt(10.33), Number(10.33), parseFloat(10.33)

toString() -- convert to string
Ex: x.toString()

Converting Booleans to Strings

The global method String() can convert booleans to strings.

```
String(false)      // returns "false"
String(true)       // returns "true"
```

toFixed() - returns a string, with the number written with a specified number of decimals

```
let x = 9.656;
x.toFixed(0);
x.toFixed(2);
x.toFixed(4);
x.toFixed(6);
```

toPrecision() - returns a string, with a number written with a specified length:

```
let x = 9.656;
x.toPrecision();
x.toPrecision(2);
x.toPrecision(4);
x.toPrecision(6);
```

JavaScript BOM

The Browser Object Model (BOM)deals with the browser itself and provides objects and methods for interacting with the browser window.

Window – understand the window object.

Alert – display an alert dialog.

Confirm – display a modal dialog with a question.

Prompt – prompt the user to input some text.

setTimeout – set a timer and execute a callback function once the timer expires.

Ex: setTimeout(myGreeting, 5000);

```
function myGreeting() {

  document.getElementById("demo").innerHTML = "Happy Birthday!"

}
```

setInterval – method calls a function repeatedly at specified intervals (in milliseconds).

Ex: setInterval (hello, 5000);

```
function hello() {

  alert(“hello”)
}
```

Location – manipulate the location of a document via the location object.

Console.log(location)

history – manage the web browser's history stack with the history object.

Console.log(history.back(), history.forward())

screen - object provides the attributes of the screen on which the current window is being rendered.

Console.log(screen)

Hoisting

Hoisting is a concept that enables us to extract values of variables even before initializing/assigning value without getting errors and this happens during the 1st phase (memory creation phase) of the Execution Context.

Features of Hoisting:

- In JavaScript, Hoisting is the default behavior of moving all the declarations at the top of the scope before code execution. Basically, it gives us an advantage that no matter where functions and variables are declared, they are moved to the top of their scope regardless of whether their scope is global or local.

What is JSON?

JSON or JavaScript Object Notation is a format for structuring data.

JSON is the most commonly used format for transmitting data (data interchange) from a server to a client and vice-versa. JSON data are very easy to parse and use. It is fast to access and manipulate JSON data as they only contain texts.

JSON Object

The JSON object is written inside curly braces `{ }`. JSON objects can contain multiple **key/value** pairs. For example,

```
// JSON object

{ "name": "John", "age": 22 }
```

JSON Array

JSON array is written inside square brackets `[ ]`. For example,

```
// JSON array

[ "apple", "mango", "banana"]



// JSON array containing objects

[

    { "name": "John", "age": 22 },

    { "name": "Peter", "age": 20 }.

    { "name": "Mark", "age": 23 }

]
```

JSON Data

You can access JSON data using the dot notation. For example,

```
// JSON object
const data = {


    "name": "John",


    "age": 22,


    "hobby": {
```

```
            "reading" : true,

            "gaming" : false,

            "sport" : {

            "reading" : true,

            "gaming" : false,

            "sport" : {

            "reading" : true,

            "gaming" : false,

            "sport" : "cricket"

    }

    }

    },

    "class" : ["JavaScript", "HTML", "CSS"]

}

// accessing JSON object
console.log(data.name); // John
console.log(data.hobby); // { gaming: false, reading: true, sport: "football"}

console.log(data.hobby.sport); // football
console.log(data.class[1]); // HTML
```

Converting JSON to JavaScript Object

You can convert JSON data to a JavaScript object using the built-in `JSON.parse()` function. For example,

```
// json object
const jsonData = '{ "name": "John", "age": 22 }';


// converting to JavaScript object
const obj = JSON.parse(jsonData);


// accessing the data
console.log(obj.name); // John
```

Converting JavaScript Object to JSON

You can also convert JavaScript objects to JSON format using the JavaScript built-in `JSON.stringify()` function. For example,

```
// JavaScript object
const jsonData = { "name": "John", "age": 22 };


// converting to JSON
const obj = JSON.stringify(jsonData);


// accessing the data
console.log(obj); // "{"name":"John","age":22}"
```

REGEX

A regular expression (regex) is a sequence of characters that define a search pattern

A **regular expression** (*regex* for short) allow developers to match strings against a pattern, extract submatch information, or simply test if the string conforms to that pattern.

Syntax

/pattern/modifiers;

Ex:

```
const text = 'john is very very good boy';
Console.log(text.replace(very, Very))

const text = 'john is very VErY very good boy';
const regex = /very/gi;

console.log(text.replace(regex, "VERY"))

 // const regValue = /e00/i;
```

```
const regValue = /[a-e]/;
regValue.test(textInfo)
```

[abc] -> a,b,c
[^abc] -> except a,b,c
[a-x] -> a to x
[A-Z] -> A to Z
[a-zA-Z] -> a to z , A to Z
[0-9] -> 0 to 9
[a-z]? -> occurs 0 or 1 times
[a-d]+ -> occurs 1 or more times
[ ]* -> occurs 0 or more times
[ ]{n} -> occur n times
[ ]{n, } -> occur n or more times
[ ]{y,z} -> occur atleast y times but less than z times
[ ]{n}$ -> doller denotes end of string
^[7-9] -> ^ denotes always should start between the range given
\d -> [0-9]
\w -> [a-zA-Z_0-9]
\ -> add for speacial character . , -

Ex: Mobile Number validation

```
const mobVal= /^[7-9][0-9]{9}$/;
```

Password Validation

```
const passVal= /[A-Z][a-z]+[0-9]+[@#$]/;
```

Email Validation

```
const emailValidation = /[a-z0-9_\-\.]+@[a-z][\.][a-z]{2, 10}/;
```

Try catch

The try statement defines the code block to run (to try).

The **catch block** catches the error and executes the code to handle it

The code in the try block is executed first, and if it throws an exception, the code in the catch block will be executed

The `finally` statement defines a code block to run regardless of the result.

```
try {
    Try Block to check for errors.
}
catch(err) {
     Catch Block to display errors.
```

}

finally {

Finally Block executes regardless of the try / catch result.

}

Ex:

```
try {
    console.log('try');
} catch (e) {
    console.log('catch');
} finally {
    console.log('finally');
}
```

```
try {
    dadalert("Welcome Fellow Geek!");
}
catch (err) {
    console.log(err);
}
```

JavaScript this Keyword

In JavaScript, the `this` keyword refers to an **object**.

Which object depends on how `this` is being invoked (used or called).

Ex:

```
const person = {
    name: "ram",
    age: 22,
    greet: function(){
        return `Hello ${this.name}, you are ${this.age} years old`
    }
}
console.log(person.greet());
```

JavaScript Callbacks

JavaScript is an asynchronous language, which means that it can handle multiple tasks simultaneously. Callbacks are a fundamental aspect of JavaScript, as they allow you to run code after an

asynchronous operation has been completed.

Why use Callbacks?
Sometimes the program would freeze and wait for the operation to complete before continuing, in order to avoid this
Callbacks allow you to continue executing code while the operation is being executed in the background. Once the operation has completed, the callback function is called with the result of the operation. This way, you can ensure that the program remains responsive and the user experience is not impacted.

```
function mainFunction(callback) {
  console.log("Performing operation...");
 callback("Operation complete");
}
function callbackFunction(result) {
  console.log("Result: " + result);
}

mainFunction(callbackFunction);
```

Closure in JavaScript

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). I
n other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

A closure is a feature of JavaScript that allows inner functions to access the outer scope of a function. Closure helps in binding a function to its outer boundary and is created automatically whenever a function is created.

Ex:

```
function x() {
  let a = 10;
  console.log(a);
  function y() {
    console.log(a);
  }
  a = 40;
  return y;
}

z = x();
```

```
z();
```

Arrow Function

Arrow functions were introduced in ES6.
Arrow functions allow us to write shorter function syntax:
Arrow functions reduce the size of the code.
it increases the readability of the code.

//Ex:

```
const arrowFunc = () => {
    console.log( "Hi Arrow Function!" );
}
arrowFunc();
```

Synchronous JavaScript:

As the name suggests synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed.

```
document.write("Hi"); // First
 document.write("Mayukh") ;// Second
 document.write("How are you");
```

Asynchronous JavaScript:

Asynchronous programming is a technique that enables your program to start a potentially long-running task and still be able to be responsive to other events while that task runs, rather than having to wait until that task has finished. Once that task has finished, your program is presented with the result.

Functions running in **parallel** with other functions are called **asynchronous**

A good example is JavaScript setTimeout()

```
document.write("Hi");
setTimeout(() => {
    document.write("Let us see what happens");
}, 2000);
document.write("End");
```

Promises

Promises are used to handle asynchronous operations in javascript.

The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

Before promises, callbacks were used to handle asynchronous operations. But due to the limited functionality of callbacks, using multiple callbacks to handle asynchronous code can lead to unmanageable code.

Promise object has three states -

- Pending - Initial state of promise. This state represents that the promise has neither been fulfilled nor been rejected, it is in the pending state.
- Fulfilled - This state represents that the promise has been fulfilled, meaning the async operation is completed.
- Rejected - This state represents that the promise has been rejected for some reason, meaning the async operation has failed.

A promise is created using the **Promise** constructor which takes in a callback function with two parameters, **resolve** and **reject** respectively.

resolve is a function that will be called when the async operation has been successfully completed.

reject is a function that will be called, when the async operation fails or if some error occurs.

We can consume any promise by attaching then() and catch() methods to the consumer.

then() method is used to access the result when the promise is fulfilled.

catch() method is used to access the result/error when the promise is rejected. In the code below, we are consuming the promise.

```
Ex: let promise = new Promise(function (resolve, reject) {
    const x = "text";
    const y = "text"
    if (x === y) {
        resolve("text");
    } else {
        reject("error");
    }
});

promise.
    then(function (item) {
        console.log(item);
    }).
    catch(function (err) {
        console.log(err);
    });
```

```
fetch('https://dummyjson.com/products/1')
.then(res => res.json())
.then(json => console.log(json))
```

Async function

Async simply allows us to write **promises-based** code as if it was synchronous and it checks that we are not breaking the execution thread. It operates asynchronously via the event loop. Async functions will always return a value. It makes sure that a promise is returned and if it is not returned then JavaScript automatically wraps it in a promise which is resolved with its value.
//Ex:

```
const getData = async () => {
    let data = "Hello World";
    return data;
}
Console.log(getData)

getData().then(data => console.log(data));
```

Await Function

Await function is used to wait for the promise. It could be used within the async block only. It makes the code wait until the promise returns a result. It only makes the async block wait.

```
let response = await  fetch('https://dummyjson.com/products/1');
    console.log(response,'res')
    let user =  response.json();
```

```
function asynchronous_operational_method() {
    let first_promise =
        new Promise((resolve, reject) => resolve("Hello"));
    let second_promise =
        new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("world");
        }, 3000);
    });
    let combined_promise =
        Promise.all([first_promise, second_promise]);
    return combined_promise;
}

async function display() {
    let data = await asynchronous_operational_method();
    console.log(data);
}

display();
```

Promise All

it allows you to pass multiple promises, until all promises goes to complete status

forEach function

const array1 = ['a', 'b', 'c'];

array1.forEach((element) => console.log(element));

Map Function

The Javascript **map()** method in JavaScript creates an array by calling a specific function on each element present in the parent array.

const array1 = [1, 4, 9, 16];

const map1 = array1.map((item, index) => item * 2);

Filter Function

The JavaScript **Array filter()** Method is used to create a new array from a given array consisting of only those elements from the given array which satisfy a condition set by the argument method.

const array= [1,2,3,4,5,6,7,8];

const result = array.filter((item, index) => item%2 === 0)

Array reduce() Method

The Javascript **arr.reduce()** method in JavaScript is used to reduce the array to a single value and executes a provided function for each value of the array (from left to right) and the return value of the function is stored in an accumulator.

```
let arr = [1,2,3,4,5,6,7]
arr.reduce((total, currentValue)=> {
  console.log( total , 'a', currentValue)
  return total + currentValue;
})
```

Destructuring assignment

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from

objects, into distinct variables.
const x = [1, 2, 3, 4, 5];
const [y, z] = x;
console.log(y); // 1
console.log(z); // 2

```
const obj = { fullName: 'yogesh', professional: 'software
Engineer'  };
const { fullName, professional } = obj;
console.log(fullName, professional)
// is equivalent to
// const fullName = obj.fullName;
// const professional = obj.professional;
```

Spread Operator

The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

```
const numbers = [1, 2, 3, 4, 5, 6];
const [one, two, ...rest] = numbers;
console.log(rest)

const arr1 = [1,2,3]
const arr2 = [4,5,6]
const res = [...arr1, ...arr2]
console.log(res)

let obj1 = {
  name: 'yogesh',
  age: '10',
  email: 'yogesh@gmail.com'
}

const res = {
  ...obj1,
  email: 'test@gmail.com'
}
console.log(res)
```

rest parameter

The rest parameter is an improved way to handle function parameters, allowing us to more easily handle various inputs as

parameters in a function. The rest parameter syntax allows us to represent an indefinite number of arguments as an array. With the help of a rest parameter, a function can be called with any number of arguments, no matter how it was defined

```
function fun(a, ...input){
  let sum = 0;
  for(let i of input){
      sum+=i;
  }
  return a+sum;
}
console.log(fun(1,2)); //3
console.log(fun(1,2,3)); //6
console.log(fun(1,2,3,4,5)); //15
```

Array find()

find() method of Array instances returns the first element in the provided array that satisfies he provided testing function. If no values satisfy the testing function, undefined is returned.

```
const array1 = [5, 12, 8, 130, 44];
const found = array1.find((element) => element > 10);
console.log(found);

const inventory = [
  { id: 1, quantity: 2, name: "yogesh" },
  { name: 2, quantity: 0, name: "john" },
  { name: 3, quantity: 5, name: "mickel" },
];
let result = inventory.find((item)=> item.id == 2)
console.log(result)
```

POST API CALL

```
<!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>
    <p>Name</p>
    <input type="text" id="title">
  </div>
  <div>
    <p>salary</p>
    <input type="number" id="salary">
  </div>
  <div>
    <p>age</p>
    <input type="number" id="age">
  </div>
  <button type="button" onclick="handleFunc()">Submit</button>
  <p id="diplymsg"></p>
```

```
  <script>
    function handleFunc() {
      const title = document.getElementById("title").value;
      const salary = document.getElementById("salary").value;
      const age = document.getElementById("age").value;
      const object = { name: title, salary: salary, age: age }
      let result =
fetch(`https://dummy.restapiexample.com/api/v1/create`, {
        method: "POST",
         body: JSON.stringify(object)
      })
      result.then((item) => {
        return item.json()
      }).then((response) => {
        document.getElementById("diplymsg").innerHTML =
response.message
      })
    }
  </script>
```

```
</body>
```

```
</html>
```