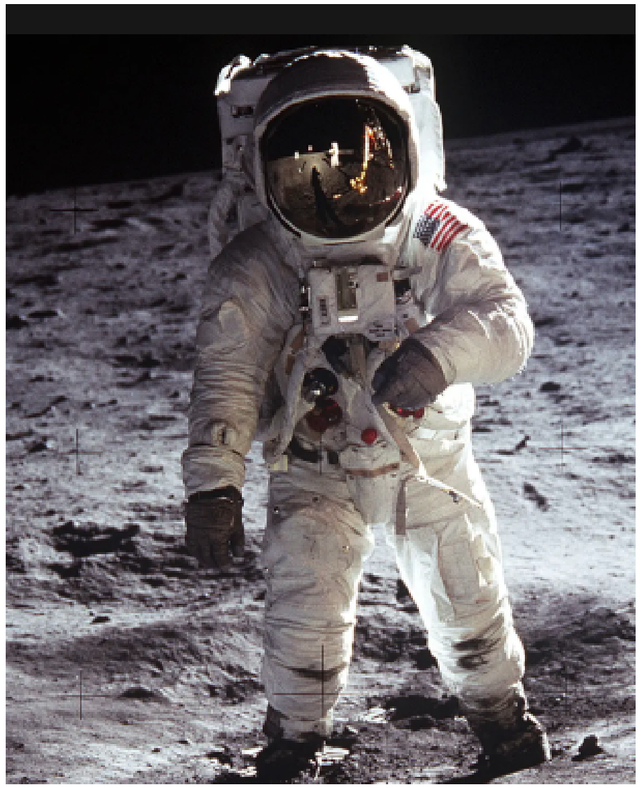

JS

What is Programming...?

Programming, in simple words, refers to the process of creating sets of instructions that tell a computer what to do.



Inventory

Price : low to high

Delivery Zip Code

90201

Models

- (●) Model S
- ( ) Model 3
- ( ) Model X

Inventory Type

- ( ) New
- (●) Used

Trim

- [ ] 70
- [ ] 70D
- [ ] 75
- [ ] 75D
- [ ] 85D
- [ ] P85
- [ ] 90D
- [ ] P90D
- [ ] P90DL
- [ ] Long Range

Order online for touchless delivery. Learn more

2016 Model S

75

79,047 mile odometer

Marina Del Rey, CA

$38,600

$526 /mo

No Est. Transport Fee

BUY NOW

VIEW DETAILS

2016 Model S

75

41,444 mile odometer

Costa Mesa, CA

$40,900

$561 /mo

No Est. Transport Fee

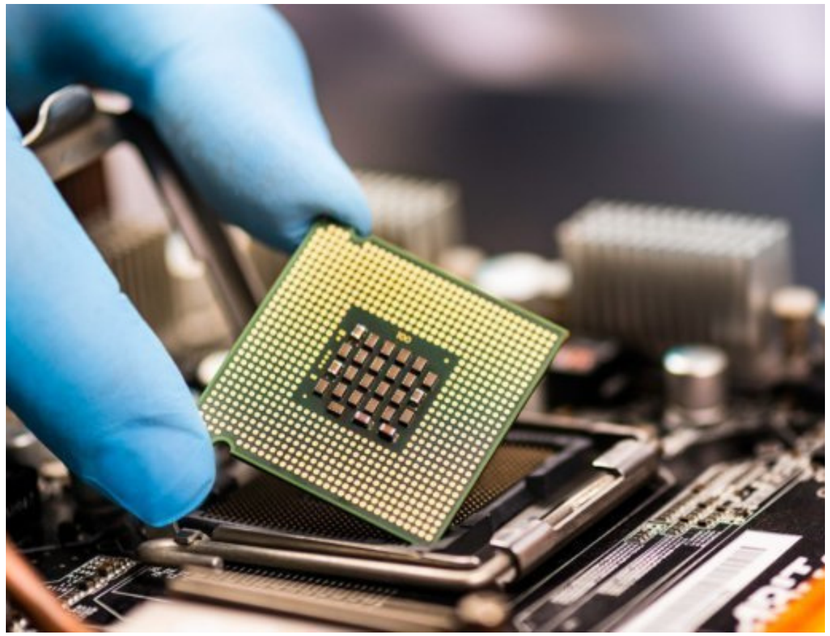
5.5s
0-60 mph

140mph
Top Speed

249mi
range (EPA)

What is Programming Language...?

What language computer really understands ?



Binary digits

0's and 1's

01001001 00100000 01101000 01101111 01110000 01100101 00100000
01111001 01101111 01110101 00100000 01101100 01101001 01101011 01100101
01100100 00100000 01101101 01111001 00100000 01001001 01101110
01110011 01110100 01110010 01110101 01100011 01110100 01100001
01100010 01101100 01100101 00101110 00100000 01101000 01100001
01110110 01100101 00100000 01100110 01110101 01101110 00100000
01110100 01110010 01100001 01101110 01110011 01101100 01100001
01110100 01101001 01101110 01100111 00100000 01110100 01101000
01101001 01110011 00100000 01100010 01101001 01101110 01100001
01110010 01111001 00100000 01100001 01101110 01100100 00100000
01110000 01110010 01100001 01100011 01110100 01101001 01100011
01100101 00100000 01110000 01110010 01100001 01100011 01110100
01101001 01100011 01100101 00100000 01110000 01110010 01100001
01100011 01110100 01101001 01100011 01100101 00101110.

Why cant we use English instead ?

1. Ambiguity
2. Lot to write

```
console.log("The sum is:", 5 + 3);
```


Java


JS

JavaScript, often referred to as JS, was initially created to fulfill a need for interactivity in web browsers. In the mid-1990s, web pages were primarily static and lacked the ability to respond to user actions or dynamically update content. This limitation led to the development of JavaScript.

In 1995, Brendan Eich, a programmer at Netscape Communications Corporation (now Mozilla), was tasked with developing a scripting language for the Netscape Navigator web browser. The goal was to create a language that could be embedded in HTML pages and executed by web browsers.

Eich's initial implementation of JavaScript, originally named "Mocha" and later renamed to "LiveScript" before settling on "JavaScript," introduced the ability to add interactivity to web pages. JavaScript provided a way to validate user inputs, manipulate HTML elements, and perform other client-side operations. This brought a new level of dynamism to the web and enhanced the user experience.

Over time, JavaScript evolved and gained popularity beyond its original purpose. It expanded to support server-side development (with **Node.js**), mobile app development (with frameworks like React Native), and even desktop applications (with frameworks like Electron). Today, JavaScript is one of the most widely used programming languages, powering not only web development but also a broad range of applications across different platforms.


node
JS

Node.js, in simple terms, is a runtime environment that allows you to run JavaScript code on the server-side, outside of a web browser. It provides a platform for building scalable and efficient server-side applications using JavaScript.

INSTALLATION

FIRST JS PROGRAM

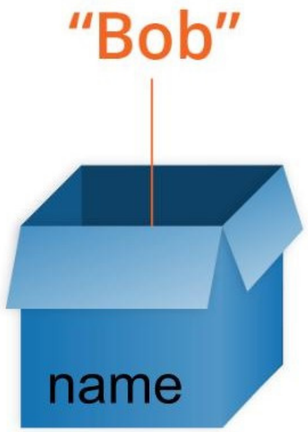
```
Console.log("Hello world")
```

node filename

- Write a program that prints "Today is Monday" to the terminal?
- Write a program that prints a number to the terminal ?

Variables

In JavaScript, a variable is a named container that can store different types of information. It's like a labeled box where you can put things and change them whenever you want.



- We declare a variable called "message" using the **var** keyword. This variable will store a message, and we assign it the initial value of "Hello, world!".

- We use the **console.log()** function to print the value of the "message" variable to the console. This will display "Hello, world!" as output.

Rules for Declaring a variable

- Variable names can contain letters (A-Z, a-z), digits (0-9), underscores (_), and dollar signs ($). They must start with a letter, underscore, or dollar sign. However, it is a convention to start variable names with a lowercase letter.

- Variable names are case-sensitive, so "myVariable" and "myvariable" would be considered two different variables.

- JavaScript has reserved words that cannot be used as variable names because they have special meanings in the language. For example, you cannot use "var" or "function" as a variable name.

- Avoid using special characters or symbols in variable names, except for underscores (_) or dollar signs ($). JavaScript does not allow spaces, hyphens, or other punctuation marks.

- Create a variable called **myNumber** and assign it the value 10. Output the value of myNumber to the console.

- Create a variable called **myString** and assign it a text value of your choice. Output the value of **myString** to the console.

- Create a variable called **username** and assign it an initial value of "John". Change the value of **username** to "Jane" and output the updated value to the console.

DATATYPES

In JavaScript, data types are like different categories for information. It's a way for the computer to know what kind of data we are using. Just like you have different types of things in your room, JavaScript has different types of data.

Number

Numbers are data types in JavaScript. You can use them for counting, doing math, or measuring things. Think of it like having a box specifically for numbers.

String

Then there are words or sentences, which are called strings in JavaScript. Strings are like a special box for storing text. You can put letters, words, or even whole stories in these boxes.

Boolean

There's also a special type called boolean, which is like a yes or no box. It's used when we want to say "yes" or "no" to something. It helps us make decisions in our code.

Number

This data type represents numeric values. It includes whole numbers (integers) and numbers with decimal points (floating-point numbers).

```
var age = 25;
var temperature = -10
var pi = 3.14159;
```

Arithmetic Operations

We can perform various arithmetic operations on numbers, such as addition (+), subtraction (-), multiplication (*), division (/), and modulus (%).

```
var sum = 5 + 3
var product = 4 * 2
var quotient = 10 / 3
var exponent = 2**3
var remainder = 10 % 3
```

- Write a program that takes two numbers as input and calculates their sum.

- Write a program that calculates the area of a circle given its radius.

- Write a program that calculates the area of rectangle.

Comparison Operations

In JavaScript, comparison operators are used to compare values and return a Boolean result (true or false). They are often used to determine the relationship between two values or variables.

- **Equality (==):**
  - Compares two values for equality, after performing type coercion if necessary.
  - Returns true if the values are equal, false otherwise.
- **Inequality (!=):**
  - Compares two values for inequality, after performing type coercion if necessary.
  - Returns true if the values are not equal, false otherwise.
- **Strict Equality (===):**
  - Compares two values for equality without performing type coercion.
  - Returns true if the values are equal and of the same type, false otherwise.
- **Strict Inequality (!==):**
  - Compares two values for inequality without performing type coercion.
  - Returns true if the values are not equal or of different types, false otherwise.

- **Greater Than (>):**
  - Checks if the left operand is greater than the right operand.
  - Returns true if the condition is true, false otherwise.
- **Less Than (<):**
  - Checks if the left operand is less than the right operand.
  - Returns true if the condition is true, false otherwise.
- **Greater Than or Equal To (>=):**
  - Checks if the left operand is greater than or equal to the right operand.
  - Returns true if the condition is true, false otherwise.
- **Less Than or Equal To (<=):**
  - Checks if the left operand is less than or equal to the right operand.
  - Returns true if the condition is true, false otherwise.

NaN - Not a Number

```
var a;
var y = a+5    // NaN
var y = 100/0    // Infinity
```

if condition

In JavaScript, the if condition is like a decision-making tool. It helps the computer make choices based on certain conditions. Imagine you have a box of chocolates, and you want to eat one only if it's your favorite flavor. You can use an if condition to check if the chocolate is your favorite before you eat it.

```
if (condition) {
  // Code to be executed if the condition is true
} else {
  // Code to be executed if the condition is false
}
```

1. Check if a number is positive:
2. Check if a number is even.
3. Check if a number is divisible by 3

logical operators

In JavaScript, the logical operators && (and), || (or), and ! (not) are used to combine or negate boolean expressions.

&& (and): It returns true if both the expressions on its left and right sides are true.

```
let a = 5;
let b = 10;
if (a > 0 && b < 20) {
  console.log("Both conditions are true.");
}
```

|| (or): It returns true if at least one of the expressions on its left or right side is true.

```
let a = 5;
let b = 10;
if (a > 10 || b < 20) {
  console.log("At least one condition is true.");
}
```

! (not): It negates the boolean value of an expression. If the expression is true, it returns false, and if the expression is false, it returns true.

```
let a = 5;
if (!(a > 10)) {
  console.log("The condition is false.");
}
```

1. Program to check if a number is positive and even
2. Program to check if a number is either negative or divisible by 3
3. Program to determine if a number is greater than 20 and less than 30

String

In JavaScript, a string is a data type used to represent a sequence of characters. It is a fundamental data type in the language and is commonly used to store and manipulate text.

Declaring a string

You can declare a string by enclosing the text within single quotes (''), double quotes ("") or backticks (``).

```
var str1 = 'Hello, world!';
var str2 = "JavaScript is awesome!";
var str3 = `The value of pi is approximately ${pi}`;
```

Template literals

Template literals, also known as template strings, are a feature in JavaScript that allow you to embed expressions within string literals. They are enclosed by backticks ( ) instead of single quotes or double quotes. Template literals offer a more convenient and expressive way to manipulate strings, especially when you need to concatenate multiple values or include dynamic content.

```
var name = 'John';
let age = 25;
const greeting = `My name is ${name} and I am ${age} years old.`
```

Concatenation

Strings can be concatenated using the + operator or the concat() method. This allows you to combine multiple strings into one.

```
var firstName = 'John';
var lastName = 'Doe';
var fullName = firstName + ' ' + lastName; // "John Doe"
var greeting = 'Hello '.concat(firstName, ' ', lastName); // "Hello John Doe"
```

String length

You can determine the length of a string using the length property.
It returns the number of characters in the string.

```
let message = 'Hello, world!';
let length = message.length; // 13
```

Accessing characters

You can access individual characters within a string using square brackets [] and the zero-based index of the character.

```
let message = 'Hello, world!';
let firstCharacter = message[0]; // "H"
let sixthCharacter = message[5]; // ","
```

Common string methods

You can access individual characters within a string using square brackets [] and the zero-based index of the character.

```
let message = 'Hello, world!';
let upperCaseMessage = message.toUpperCase(); // "HELLO, WORLD!"
let thirdCharacter = message.charAt(2); // "l"
let extractedSubstring = message.substring(0, 5); // "Hello"
let indexOfWorld = message.indexOf('world'); // 7
let replacedMessage = message.replace('world', 'JavaScript'); // "Hello, JavaScript!"

let splitArray = message.split(', '); // ["Hello", "world!"]
```

- Write a JavaScript program that takes two strings and concatenates them.
- Write a JavaScript program that takes a string and prints the length of the string.
- Write a Js program that converts the string to uppercase.

1. Write a JavaScript program to check if a number is positive or negative.
2. Write a JavaScript program to check if a number is even or odd.
3. Write a JavaScript program to check if a number is a multiple of 5.
4. Write a JavaScript program to check if a number is between 50 and 100.
5. Write a JavaScript program to check if a string is empty.
6. Write a JavaScript program to check if a string has more than 10 characters.
7. Write a JavaScript program to check if two strings are equal.
8. Write a JavaScript program to concatenate two strings.
9. Write a JavaScript program to convert a string to uppercase.
10. Write a JavaScript program to calculate the sum of two numbers.
11. Write a JavaScript program to calculate the difference between two numbers.
12. Write a JavaScript program to calculate the product of two numbers.
13. Write a JavaScript program to calculate the average of three numbers.
14. Write a JavaScript program to find the smallest of two numbers.
15. Write a JavaScript program to find the largest of two numbers.

LOOPING

Looping is a fundamental concept in JavaScript that allows you to execute a block of code repeatedly

While Loop

The while loop repeatedly executes a block of code as long as a specified condition is true. The loop continues until the condition becomes false

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

- Write a program that prints numbers from 10 to 1 in descending order.
- Write a program that prints all even numbers between 1 and 20.
- Write a program to find the sum of n numbers.
- Write a program that calculates the factorial of a given number.

Do...While Loop

Similar to the while loop, the do...while loop executes a block of code at least once, and then repeatedly executes it as long as the specified condition is true

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

For Loop

The for loop is widely used when you know the number of iterations in advance. It consists of three parts: initialization, condition, and increment/decrement

```
for (initialization; condition; increment/decrement) {
        // code to be executed in each iteration
}
```

- Write a program that prints numbers from 1 to 10
- Write a program that calculates the sum of the even numbers from 1 to 10.
- Write a program that prints numbers from 10 to 1 in descending order.
- Write a program that calculates the factorial of a given number.
- Write a program that prints a pattern of asterisks (*) in the shape of a right triangle.

ARRAYS

Data Type

- Data type refers to the classification or categorization of data based on its nature and the operations that can be performed on it.
- It defines the kind of values that can be stored in a variable or used in expressions.
- Data types are predefined or built-in in programming languages.
- Examples of data types include integers, floating-point numbers, characters, booleans, and strings.

Data Structure

- A data structure refers to the way data is organized and stored in memory.
- Data structures are used to efficiently manage and access data.

In JavaScript, an array is a data structure that allows you to store and manipulate multiple values in a single variable. It is a fundamental concept in programming and is widely used to manage collections of data.

Arrays in JavaScript can hold any type of value, including numbers, strings, objects, and even other arrays. The values within an array are called elements, and each element has an index associated with it. The index represents the position of the element within the array, starting from zero for the first element.

```
var fruits = ['apple', 'banana', 'orange'];
```

INDEX

To access individual elements in an array, you can use square brackets along with the **index** of the element. Index value starts from 0.

```
console.log(fruits[0]); // Output: 'apple'
console.log(fruits[1]); // Output: 'banana'
console.log(fruits[2]); // Output: 'orange'
```

Array methods

Array length

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

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

Push

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

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

pop

The pop() method removes the last element of an array

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

splice

The splice() method in JavaScript is used to modify an array by adding, removing, or replacing elements at specific positions. It allows you to change the contents of an array in place.

array.splice(start, deleteCount, item1, item2, ...);

- **start**: This is the index at which the modification should begin. It specifies the position where elements will be added or removed.
- **deleteCount**: The number of elements to remove from the array starting at the start index. If set to 0, no elements are removed.
- **item1, item2**, ...: Optional. The elements to be inserted at the start index.

```
let numbers = [1, 2, 3, 4, 5];
numbers.splice(2, 2);
console.log(numbers); // Output: [1, 2, 5]
```

In JavaScript, the **includes()** method is used to check if an array or a string includes a specific element or substring.

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

console.log(numbers.includes(3)); // Output: true
console.log(numbers.includes(6)); // Output: false
```

```
const str = "Hello, world!";

console.log(str.includes("world")); // Output: true
console.log(str.includes("foo")); // Output: false
```

for loop: The traditional for loop allows you to iterate over an array using an index.

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


for (let i = 0; i < array.length; i++) {
  console.log(array[i]);
}
```

for...of loop: The for...of loop iterates over the values of an iterable object, such as an array.

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


for (const element of array) {
  console.log(element);
}
```

- Write a for loop to calculate the sum of all elements in an array.
- Write a for loop to find the index of a specific element in an array.
- Print all odd numbers from an array using a for loop.
- Create a new array that contains the square of each element from the original array using a for loop.
- Count the number of occurrences of a given element in an array using a for loop.
- write a function to remove all occurrences of a specific element from an array
- Find the largest number in an array using a for loop.
- Write a for loop to reverse an array.
- write a for loop to remove duplicates from an array.
- write a program to find intersection of two arrays.
- write a for loop to sort a number array .

In JavaScript, arrays are reference data types. This means that when you assign an array to a variable , you are actually working with a reference to the original array in memory.

```
const originalArray = [1, 2, 3];
const newArray = originalArray; // Assigning the reference to newArray

newArray.push(4); // Modifying newArray

console.log(originalArray); // Output: [1, 2, 3, 4]
console.log(newArray); // Output: [1, 2, 3, 4]
```

OBJECTS

In JavaScript, objects are one of the fundamental data types and provide a way to store and manipulate structured data. An object is a collection of key-value pairs, where each key represents a property or method name, and the associated value represents the data or function associated with that property or method.

```
const person = { name: 'John', age: 25};
```

Accessing Properties

Object properties can be accessed using dot notation (objectName.propertyName) or bracket notation (objectName['propertyName']).

```
console.log(person.name);        // Output: John
console.log(person['age']);      // Output: 25
person.sayHello();               // Output: Hello!
```

Adding and Modifying Properties

Properties can be added or modified by assigning a value to a new or existing property.

```
person.name = 'Alice';          // Modifying existing property
person['age'] = 30;             // Modifying existing property
person.city = 'New York';       // Adding new property
```

Removing Properties

Properties can be removed from an object using the delete operator.

```
delete person.city;
```

Object Methods

JavaScript provides several built-in methods for objects, such as Object.keys(), Object.values(), and Object.entries(), which allow you to retrieve keys, values,

```
const keys = Object.keys(person);       // Retrieves all keys of the object
const values = Object.values(person);   // Retrieves all values of the object
const entries = Object.entries(person); // Retrieves key-value pairs of the object
```

Object.hasOwnProperty()

Returns a boolean indicating whether an object has a specific property as its own property.

for...in loop

The for...in loop iterates over the enumerable properties of an object, including inherited properties from the prototype chain.

```
const person = { name: 'John', age: 30 };


for (let key in person) {
  console.log(key, person[key]);
}
// Output: name John
//         age 30
```

Questions

- Check if a number is prime or not.
- Write a program that prints the Fibonacci series.
- Check if a given string is palindrome.

Array and object destructuring

Array and object destructuring are powerful features in JavaScript that allow you to extract values from arrays and objects and assign them to variables in a more concise and convenient way.

Array destructuring allows you to unpack values from an array into separate variables.

```
const numbers = [1, 2, 3];
const [a, b, c] = numbers;
console.log(a); // Output: 1
console.log(b); // Output: 2
console.log(c); // Output: 3
```

Object destructuring allows you to extract values from an object and assign them to variables with the same property names.

```
const person = { name: 'John', age: 30 };
const { name, age } = person;
console.log(name); // Output: John
console.log(age); // Output: 30
```

You can also specify default values for variables in case the corresponding value is undefined or missing in the array or object.

```
const numbers = [1, 2];
const [a, b, c = 3] = numbers;
console.log(a); // Output: 1
console.log(b); // Output: 2
console.log(c); // Output: 3

const person = { name: 'John' };
const { name, age = 30 } = person;
console.log(name); // Output: John
console.log(age); // Output: 30
```

The rest syntax (...) can be used in both array and object destructuring to collect the remaining elements into a new array or object.

```
const numbers = [1, 2, 3, 4, 5];
const [a, b, ...rest] = numbers;
console.log(a); // Output: 1
console.log(b); // Output: 2
console.log(rest); // Output: [3, 4, 5]

const person = { name: 'John', age: 30, city: 'New York' };
const { name, ...rest } = person;
console.log(name); // Output: John
console.log(rest); // Output: { age: 30, city: 'New York' }
```

Function

In JavaScript, a function is a block of reusable code that performs a specific task. Functions are one of the fundamental building blocks of JavaScript programming, allowing you to organize your code into modular and reusable units

Function Declaration

In JavaScript, you can declare a function using the function keyword.

```
function greet() {
  console.log('Hello, world!');
}

greet(); // Output: Hello, world!
```

- Write a function called sayHello that logs "Hello!" to the console when called.

Parameter

A parameter in a function is a variable that acts as a placeholder to receive values when the function is called. It allows you to pass values into the function so that it can perform specific actions or computations based on those values.

```
function functionName(parameter1, parameter2, ...) {
  // Function body
  // Code to be executed
  // Optionally, a return statement
}
```

1. Write a function called sayHello that takes a name as a parameter and logs "Hello, [name]!" to the console.
2. Write a function to find area of a circle.
3. Create a function named calculateArea that takes the width and height of a rectangle as parameters and prints the calculated area.
4. Define a function called getFullName that takes a first name and a last name as parameters and prints the full name as a string.
5. Write a function named checkEvenOdd that takes a number as a parameter and logs whether it's even or odd to the console.

Fat Arrow Function

A fat arrow function, also known as an arrow function or arrow function expression, is a concise way to write a function in JavaScript. It provides a more compact syntax compared to traditional function expressions

```
(parameters) => {
  // function body
  return expression;
}
```

Return statement

A parameter in a function is a variable that acts as a placeholder to receive values when the function is called. It allows you to pass values into the function so that it can perform specific actions or computations based on those values.

```
function functionName(parameter1, parameter2, ...) {
  // Function body
  // Code to be executed
  // Optionally, a return statement
}
```

```
function myFunction(p1, p2) {
        return p1 * p2;
}
```

```
// Traditional function expression
function add(a, b) {
  return a + b;
}

// Equivalent fat arrow function
const add = (a, b) => a + b;
```

1. Write a function in JavaScript that takes two numbers as parameters and returns their sum.
2. Create a function that takes an array of numbers as an argument and returns the largest number in the array.
3. Create a function that takes a string as input and returns the number of vowels in the string.

- Create a function that takes a string as input and returns the number of vowels in the string.
- Write a function to remove duplicates from an array.
- Write a function to check if a given number is a prime number.
- Write a function to reverse a string without using the built-in reverse() method.
- Write a function that accepts a string as a parameter and returns the most frequent character in the string.
- Write a function that accepts a sentence as a parameter and returns the word with the highest number of vowels.
- Write a function to find the second smallest element in an array of numbers.

```
if (condition1) {
  // code to be executed if condition1 is true
} else if (condition2) {
  // code to be executed if condition1 is false
} else if (condition3) {
  // code to be executed if both condition1 and
} else {
  // code to be executed if all conditions are f
}
```

Callback function

In JavaScript, a callback function is a function that is passed as an argument to another function and is invoked or called within that function.

```
function greet(name, callback) {
  console.log("Hello, " + name + "!");

  // Invoke the callback function
  callback();
}

function sayGoodbye() {
  console.log("Goodbye!");
}

// Usage: Pass the sayGoodbye function as a callback to greet()
greet("John", sayGoodbye);
```

Create a function calculateSum that takes two numbers and a callback function as arguments. The function should pass the sum of the 2 numbers to the callback function and the callback should print the value

```
function calculateSum(num1, num2, callback) {
  const sum = num1 + num2;
  return callback(sum);
}

function printResult(result) {
  console.log("The result is:", result);
}

// Usage: Pass the printResult function as a callback to calculateSum()
calculateSum(5, 3, printResult); // Output: The result is: 8
```

map() function

The map() function applies a provided callback function to each element of an array and returns a new array containing the results. The resulting array will have the same length as the original array.

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

const doubledNumbers = numbers.map(num => num * 2);

console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

- Create a program that converts an array of strings to uppercase using map().
- Write a JavaScript program that extracts the lengths of strings in an array using map()
- Write a JavaScript program that adds a prefix "Hello, " to each element in an array using map()

filter() function

The filter() function creates a new array with all the elements that pass a certain condition. It tests each element in the array against the provided callback function and includes the element in the resulting array if the callback function returns true.

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

const evenNumbers = numbers.filter(num => num % 2 === 0);

console.log(evenNumbers); // Output: [2, 4]
```

- Write a JavaScript program that filters out even numbers from an array using the filter()
- Write a JavaScript program that filters out strings containing 'a' from an array using filter()

Primitives

Type	`typeof` return value	Object wrapper
Null	`"object"`	N/A
Undefined	`"undefined"`	N/A
Boolean	`"boolean"`	`Boolean`
Number	`"number"`	`Number`
BigInt	`"bigint"`	`BigInt`
String	`"string"`	`String`

Reference Datatypes

Variables that are assigned a non-primitive value are given a reference to that value. That reference points to the object’s location in memory. The variables don’t actually contain the value.

Objects are created at some location in your computer’s memory. When we write arr = [], we’ve created an array in memory. What the variable arr receives is the address, the location, of that array.

Let and Const

let and **const** are block-scoped variables

The difference between let and const lies in their mutability. Variables declared with let can be reassigned to a different value, while variables declared with const are read-only and cannot be reassigned once initialized.

Scope

Variables declared with var have function scope or global scope if declared outside any function. They are not limited to the block where they are defined.

- Variables declared with let and const have block scope. They are limited to the block (within curly braces) where they are defined.

- Block scope means that the variables are only accessible within the block in which they are declared.

When using let and const, variables are hoisted to the top of their block but are not accessible until the point of declaration. Accessing them before the declaration results in a ReferenceError

THANK YOU