JavaScript

- ## What is javaScript?

⇨ JavaScript is a high-level, Object-oriented, multi-paradigm programming language**.**

- **Programming Language:** instruct computer to do things
- **High-level:** we don't have to worry about complex stuff like memory management.
- **Object Oriented:** based on objects for storing most kind of data.
- **Multi-paradigm**: refers to the ability to use multiple programming styles within a single programming language.

- ## What are the key features of javaScript?

⇨ The kye features of javaScript as below:

- **Client-Side Scripting:** Client-side scripting refers to the execution of scripts or code on the client's (user's) web browser rather than on the web server.
- **Object Oriented:**
- **Event Driven Programming:** where actions or events (like button clicks, mouse movements, etc.) trigger specific functions or behaviors.
- **Cross-platform Compatibility:** javaScript is supported by all major web browsers, making it a cross-platform language. This allows developers to write code that works consistently across different browsers and operating systems.
- **Asynchronous Programming:** JavaScript supports asynchronous programming through mechanisms like callbacks, promises, and async/await. This is essential for handling tasks such as fetching data from servers without blocking the execution of other code.

- ## What are JavaScript Variables?

⇨ JavaScript, variables are used to store and manage data. Variables are containers that hold values and can be referenced by a name. The var, let, and const keywords are used to declare variables.

- **var (functional scoped variable):**
  - **'var'** is the old way of declaring variables in JavaScript.

- Variables declared with '**var'** are function-scoped, meaning they are only accessible within the function where they are defined.
- Eg. var a = 10;

➢ **let (Block-Scoped Variable):**

- Introduced in ECMAScript 6 (ES6), '**let'** is the preferred way to declare variables in modern JavaScript.
- Variables declared with '**let'** are block-scoped, meaning they are only accessible within the block (statements inside curly braces) where they are defined.
- Eg. let message = "Hello, World!";

➢ **const (Block-Scoped Constant):**

- Also introduced in ES6, const is used to declare constants.
- Variables declared with const are block-scoped and cannot be reassigned after initialization.
- However, the content of an object declared with const can be modified.
- Eg. const PI = 3.14;

o **Variable naming rules**

- Variable names are case-sensitive ("myVar" is different from "myvar").
- Variable names can include letters, digits, underscores, and dollar signs.
- The first character must be a letter, an underscore (_), or a dollar sign ($).

- **What are the different between 'var', 'let' and 'const'?**
- **What are JavaScript Data Types?**

o **Primitive Datatypes:**

➢ **Number:**

- Representing numeric values (integer or floating point)
- Eg. let a=10;(integer). let a=1.5(float/decimal).

➢ **String:**

- Represents textual data (text)
- Eg. let first_name= "Saphal";

- **Boolean:**
  - Represent logical type data that can be true or false. (used for taking decision)
  - Eg. let isPaid: true;
- **Undefined:**
  - Value taken by a variable that is not yet defined (empty value)
  - Eg. let a;
- **Null:**
  - Represents the intentional absence of any object value.
  - Eg. let number=null;
- **Symbol:**
  - Introduced in ECMAScript 6 (ES6), represents a unique identifier**.**
  - Eg. let sym = Symbol("description");

- **Non-Primitive**
  - **Object:**
    - Represents a collection of key-value pairs (properties and methods)
    - Eg. let person = {

```
name: "John",
age: 30,
greet: function() {
console.log("Hello!");
    }

};
```

  - **Array:**
    - Represents an ordered collection of values, typically of the same data type
    - Eg. let numbers=[1,3,5,3,5];
  - **Function:**
    - Represent a reusable block of code.

- Eg. function add(a, b) {

  return a + b;

  }

- **Which symbol is used for comments in JavaScript?**
  - **Single line Comment:**
    - Double slash: //single line comment
  - **Multi Line Comment:**
    - /*…. multi line comment…*/
- **Operators In JavaScript:**

⇨ JavaScript uses various operators to perform different operations on variables and values.

- **Arithmetic Operators**
  - **Addition (+):**
    - Eg. let sum = a + b;
  - **Subtraction (-):**
    - Eg. let sub = a - b;
  - **Multiplication (*):**
    - Eg. let mul = a * b;
  - **Division (/):**
    - Eg. let div = a / b;
  - **Reminder (%):**
    - Eg. let rem = a % b;
  - **Increment by 1 (++):**
    - Eg. let x; ++x;
  - **Decrement by 1 (--):**
    - Eg. let y; --y;

- **Assignment Operator:** Assignment operators are used to assign values to variables.

Eg. let x =2; // Here, the = operator is used to assign value 2 to variable x.

Name	Operator	Example
Assignment	=	x=8; //7
Addition Assignment	+=	a += 10; // a = a +10
Subtraction Assignment	-=	a -= 5; // a= a-5
Multiplication Assignment	*=	a *= 2; // a = a *2
Division Assignment	/=	a /=3; // a = a/3
Remainder Assignment	%=	a %=4; // a = a%4
Exponential Assignment	**=	a **=5; // a = a**5

- **Comparison Operator:** Comparison operators compare two values and return a boolean value, either true or false.

  E.g: let a = 10; let b = 5; console.log(a>b);//true

Name	Operator	Meaning	Example
Equal to	= =	Returns true if the operands are equal	x = = y
Not equal to	!=	Returns true if the operands are not equal	x != y
Strict equal to	= = =	Returns true if the operands are equal and same type	x = = = y
Strict not equal to	!= =	Returns true if the operands are not equal and same type	x != = y
Greater than	>	Return true if left operand is greater than right operand	x > y
Greater or equal to	>=	Return true if left operand is greater than or equal to right operand	x >= y
Less than	<	Return true if the left operand is less than right operand	x < y
Less than or equal	<=	Return true if the left operand is less than or equal to the right operand	x <= y

- **Logical Operator:** Logical operators are used to check whether one or more expressions result in either true or false.

Name	Operator	Meaning	Example
Logical AND	&&	Returns true if all operands are true	x && y
Logical OR	||	Returns true if one of the operands is true	x || y
Logical Not	!	Reverse the result: returns true if false and vice versa	!x

• Conditional Statement in JavaScript:

⇨ Conditional statements are used to perform different actions based on different conditions.

- **if Statement:** The 'if' statement is used to execute a block of code if a specified condition evaluates to true.

Eg.

```
let x = 6;
if (x > 2) {
        console.log('x is greater than 2');
    }
```

- **if-else Statement:** The 'if-else' statement allows you to execute one block of code if the condition is true and another block if the condition is false.

Eg.

```
let y = 2;
if (y > 3) {
        console.log('y is greater than 3');
} else {
        console.log('y is not greater than 3');
}
```

- **else if statement:** We can use multiple 'else if' statements to check additional conditions if the preceding ones are false.

Eg.

```
let z = 5;
if (z > 10) {
        console.log('z is greater than 10');
} else if (z > 2) {
        console.log('z is greater than 2 but not greater than 10');
```

```
} else {
        console.log('z is 5 or less');
}
```

- **Switch Statement:** The ‘switch’ statement is useful when you want to compare a single expression against multiple possible values.

  Eg. let day = 'Monday';

```
switch (day) {
  case 'Monday':
    console.log(‘The day is Monday’);
    break;
  case 'Friday':
    console.log('Weekend is almost here');
    break;
  default:
    console.log('It\'s a regular day');
}
```

• Functions in JavaScript:

⇨ Function is a block of code designed to perform a specific task. Functions can take parameters, return values, and be assigned to variables.

The ways to writing function as below:

- **Function Declaration:** We can declare a function using the **‘function’** keyword, followed by the function name, parameters in parentheses, and the function body in curly braces.

  E.g: function greet(name) {

```
  console.log("Hello, " + name + "!");
}
```

- **Function Invocation/Calling:** We can call (invoke) a function by using its name followed by parentheses. If the function has parameters, we pass values inside the parentheses.

  E.g: greet(“saphal”);

- **Function Expression:** Functions can also be assigned to variables. These are called function expressions.

  E.g: let greet = function(name) {

  console.log("Hello, " + name + "!");

  };

  greet("Saphal"); // Output: Hello, Saphal!

- **Return Statement:** Functions can return values using the return keyword. A function stops executing when it encounters a return statement.

  E.g: function add(a, b) {

  return a + b;

  }

  console.log(add(5, 3)); // Output: 8

- **Arrow Function**: Arrow functions are a concise way to write function expressions in JavaScript. They were introduced in ECMAScript 6 (ES6) and provide a shorter syntax compared to traditional function expressions. Here's the basic syntax of an arrow function:

  // Basic syntax

  const functionName = (param1, param2, ...) => {

  // Function body

  return // result;

  };

  1. Single-line function:

  const sum = (a, b) => a + b;

  2. Multi - line function:

  const double = (x) => {

  return x * 2;

  };

- **Templates literals in JavaScript:**

⇨ Template literals, introduced in ECMAScript 6 (ES6), are a more powerful way to work with strings. They are enclosed by backticks (``) instead of single or double quotes.

E.g: let name = 'John';

```
let greeting = `Hello, ${name}!`;
console.log(greeting);  // Output: Hello, John!
```

- ## Loops in JavaScript:

Loops are used in JavaScript to perform repeated tasks based on a condition.

- **For Loop:**

The for loop in JavaScript is a control flow statement that allows you to repeatedly execute a block of code as long as a specified condition is true. It consists of three parts: initialization, condition, and iteration expression. The basic syntax is as follows:

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

Initialization: It is executed once at the beginning of the loop. It is typically used to initialize a counter variable.

Condition: It is evaluated before each iteration. If the condition is true, the loop continues; otherwise, it exits.

Iteration: It is executed at the end of each iteration. It is typically used to update the counter variable.

E.g:
```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

- ## While loop:

The while loop executes a block of code while a specified condition is true. The condition is checked before the execution of the block.

E.g:
```
let i = 0;
```

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

In this example, the loop will iterate as long as i is less than 5, and it will print the value of i in each iteration. The output will be: 0 1 2 3 4

- **Do While loop:**

The do...while loop is similar to the while loop, but the condition is checked after the execution of the block. This ensures that the block is executed at least once, even if the condition is initially false.

E.g:

```
let i = 0;

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

In this example, the loop will also print the value of i for each iteration, and the output will be the same as with the while loop: 0 1 2 3 4

- **Difference between 'while' and 'do while loop.**
- **Array:**

An array is a special type of object that is used to store and organize multiple values. Arrays can hold values of any data type, and the elements in an array are indexed numerically, starting from zero**.**

Creating Array:

You can create an array in JavaScript using the array literal syntax [] or the Array constructor**.**

```
// Using array literal
const vehicles = ['car', 'bus', 'ambulance'];
```

// Using Array constructor

const numbers = new Array(1, 2, 3, 4, 5);

Accessing Elements:

Elements in an array are accessed using square brackets [] and the index of the element. Remember, JavaScript arrays are zero-indexed, meaning the index starts at 0.

```
console.log(fruits[0]);  // Output: 'car'

console.log(numbers[2]);  // Output: 3
```

Modify Elements:

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

```
vehicles[1] = 'taxi';

console.log(vehicles);  // Output: ['car', 'taxi', 'ambulance']
```

Array Methods:

JavaScript provides a variety of methods to manipulate arrays. Here are a few examples:

- **Push and Pop:**

```
vehicles.push('tractor');  // Adds 'tractor' to the end
console.log(vehicles);   // Output: ['car', 'taxi', 'ambulance', 'tractor']

vehicles.pop();         // Removes the last element ('tractor')
console.log(vehicles);   // Output: ['car', 'taxi', 'ambulance']
```

- **Shift and Unshift:**

```
vehicles.shift();       // Removes the first element ('car')
console.log(vehicles);   // Output: ['taxi', 'ambulance']

vehicles.unshift('tractor');  // Adds 'tractor' to the beginning
```

```
console.log(vehicles);     // Output: ['tractor','taxi', 'ambulance']
```

- **Concat():**Joins two or more arrays and returns a new array.

```
const arr1 = [1, 2];
const arr2 = [3, 4];
const result = arr1.concat(arr2);
console.log(result); // Output: [1, 2, 3, 4]
```

- **indexOf():** Returns the first index at which a given element can be found in the array.

```
const fruits = ['apple', 'banana', 'orange'];
const index = fruits.indexOf('banana');
console.log(index); // Output: 1
```

- **filter():** Creates a new array with elements that pass a test.

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // Output: [2, 4]
```

- **find():** Returns the first element in an array that satisfies a provided testing function.

```
const numbers = [1, 2, 3, 4, 5];
const found = numbers.find(num => num > 2);
console.log(found); // Output: 3
```

- **map():** Creates a new array by applying a function to each element in the original array.

```
const numbers = [1, 2, 3];
const squaredNumbers = numbers.map(num => num * num);
console.log(squaredNumbers); // Output: [1, 4, 9]
```

- **join():** join the elements of the array with the given separator.

```
const fruits = ['apple', 'banana', 'orange'];
const joinedString = fruits.join(', ');
console.log(joinedString);
// Output: "apple, banana, orange"
```

- **includes():** checks if the array contains an element

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

```
// Check if the array includes the number 3
const includesThree = numbers.includes(3);
console.log(includesThree); // Output: true
// Check if the array includes the number 6
const includesSix = numbers.includes(6);
console.log(includesSix); // Output: false
```

- **slice():** The slice() method in JavaScript is used to extract a portion of an array and returns a new array containing the selected elements. It doesn't modify the original array; instead, it creates a shallow copy of the specified portion.

```
const numbers = [1, 2, 3, 4, 5];
// Extract elements from index 1 to index 3 (excluding index 3)
const slicedArray = numbers.slice(1, 3);
console.log(slicedArray);
// Output: [2, 3]
```

- **splice():** The splice() method in JavaScript is used to change the contents of an array by removing or replacing existing elements and/or adding new elements. It modifies the original array and returns an array containing the removed elements.

Syntax: array.splice(startIndex, deleteCount, item1, item2, ...);

```
const numbers = [1, 2, 3, 4, 5];
// Remove 2 elements starting from index 1 and insert 6, 7
const removedElements = numbers.splice(1, 2, 6, 7);
console.log(numbers);
// Output: [1, 6, 7, 4, 5]
console.log(removedElements);
// Output: [2, 3]
```

- **What is object in JavaScript?**

  An object is a composite data type that allows you to store and organize data using key-value pairs. Objects are used to represent and model real-world entities, and they can have properties and methods associated with them.

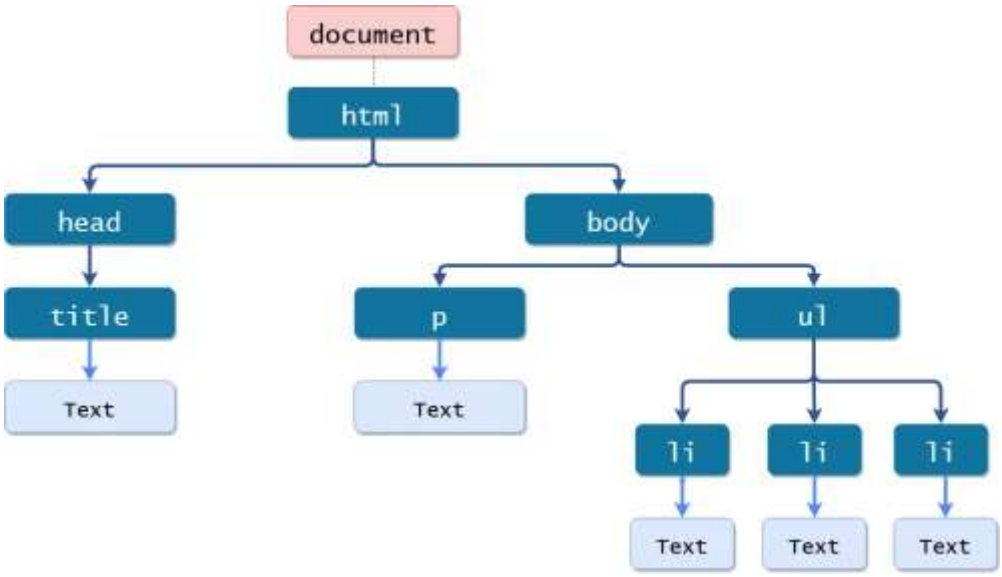
  e.g:

```
// Creating an object using object literal syntax
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 25,
  isStudent: false,
  sayHello: function() {
    console.log(`Hello, my name is   ${this.firstName} ${this.lastName}`);
  }
};

// Accessing properties of the object
console.log("First Name:", person.firstName); // Output: John
console.log("Age:", person.age); // Output: 25

// Modifying a property
person.age = 26;

// Adding a new property
person.gender = "Male";

// Accessing the updated properties
console.log("Updated Age:", person.age); // Output: 26
console.log("Gender:", person.gender); // Output: Male

// Calling a method of the object
```

person.sayHello(); // Output: Hello, my name is John Doe

- **What is DOM in JavaScript?**

  DOM stands for Document Object Model. It is a programming interface that allows us to create, change, or remove elements from the document.



- **What is an eventListener in JavaScript?**

An event listener in JavaScript is a mechanism that allows you to detect and respond to specific actions (events) that occur on a web page. The events such as:

- Clicking on Button
- Hovering Over an image
- Pressing key
- Scrolling etc.

Eg.

```
const button = document.querySelector(“button”);

button.addEventListener(“click”,function(){

alert(“hello”);
```

```
})
```