Chirag Bhangale

JavaScript Synopsys

What is JavaScript ?

JavaScript is a high-level programming language used to create dynamic and interactive webcontent. It works alongside HTML and CSS.

Why Use JavaScript and Features?

1.Full Stack Development: Enables development across the entire web application stack.

2. Client-Side Processing: Handles tasks on the user's device, reducing server load.

3. Versatility: Used in web development, server-side (Node.js), mobile apps (React Native), etc.

4. Ecosystem: Vast libraries and frameworks (e.g., React, Angular).

5. Cross-Browser Compatibility: Supported by all major browsers.

6. Server-side Scripting:With environments like Node.js, JavaScript runs on the server.

Data Types

Primitive Data Types :

1. String :
```
var name = “Chirag”;
var firstName = ‘Chirag’;
```

2. Number :
```
var id = 10;
var id = 12.43;
```

3. Boolean:
```
var result = true;
var result = false;
```

4. Undefined :
Undefined means a variable has been declared but has yet not been assigned a value.
```
var x;          //undefined
```

5. Null :
variable intentionally set to null.
```
let num = null;
```

6. BigInt :
```
let num = BigInt("12344578765467673989");
```

7. Symbol :
Symbols are immutable and are unique.
```
let value = Symbol('hello');
```

Non-Primitive Data Types :

1. Arrays :
An array is a special variable, which can hold more than one value.
```
let names = ["Chirag","Ankit","Pratik"];
```

2. Objects :
An object literal is a list of name:value pairs inside curly braces {}.
```
let obj1 = {
        id : 1,
        name : "Chirag",
        role : "Software Developer",
        exp : 5
}
console.log(obj1);
```

3. Functions :

   Regular Function :

```
function functionName(parameters) {
    // code to be executed
}
```

Eg.

```
function sum(a,b){
    return a+b;
}
console.log(sum(2,3));
```

Function Expression :

A JavaScript function can also be defined using an expression.

A function expression can be stored in a variable

Eg.

```
const x = function (a, b) {
return a * b
};
```

Array

1. Array Functions:

const fruits = ["Banana", "Orange", "Apple", "Mango", ”Grapes”, ”Papaya”];

Array length :

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

```
let size = fruits.length;
console.log(size);          // 6
```

Array at() :

The at() method returns an indexed element from an array.

```
let fruit = fruits[2];

console.log(fruit);            // Apple
```

pop() :

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

```
fruits.pop();            // Papaya removed

console.log(fruits);

 ["Banana", "Orange", "Apple", "Mango", ”Grapes”]
```

push() :

push() method adds element at last of an array.

```
fruits.push("Kiwi");          // Kiwi Added
console.log(fruits);
[“Banana", "Orange", "Apple", "Mango", ”Grapes”, ”Kiwi”]
```

shift() :

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

```
fruits.shift();          // Banana removed
console.log(fruits);
["Orange", "Apple", "Mango", ”Grapes”, ”Kiwi”]
```

unshift() :

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

```
fruits.unshift("Cranberry");        // Cranberry added
console.log(fruits);
["Cranberry", "Orange", "Apple", "Mango", ”Grapes”, ”Kiwi”]
```

splice() :

The splice() method can be used to add new items to an array.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
//  first parameter (2) defines the position where new elements should
be added
```

```
//   The second parameter (0) defines how many elements should be
removed.
//   The rest of the parameters ("Lemon" , "Kiwi") define the new
elements to be added.

console.log(fruits);// ["Banana", "Orange","Lemon", "Kiwi", "Apple",
"Mango"]

const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);   //first argument start index and second argument
count to delete element
console.log(fruits);// ["Orange", "Apple", "Mango"];
```

slice() :

The slice() method slices out a piece of an array into a new array.

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const fruit = fruits.slice(1);
console.log(fruit);  // ["Orange", "Lemon", "Apple", "Mango"]


const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const fruit = fruits.slice(1,3);      //from starting argument index to but
not including second argument index
console.log(fruit);  // ["Orange", "Lemon"]
```

Arrow Function :

Arrow functions allow us to write shorter function syntax.

Eg.   `let sum = (a, b) => a + b;`

map Function :

Creates a new array from calling a function for every array element.

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

const newNumbers = numbers.map(n => n * 2);

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

forEach :

The forEach() method calls a function for each element in an array.
The forEach() method is not executed for empty elements.

Eg.
```
const numbers = [1,2,3];
numbers.forEach(num => console.log(num));
// 1    2    3
```

filter :

The filter() method creates a new array filled with elements that pass a test provided by a function.
The filter() method does not change the original array

Eg.
```
let numbers = [10,20,30,40,50,60];
numbers.filter(num => num >=30).forEach(num => console.log(num));
// 30  40    50    60
```

2. Looping Through an Array:

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

for (let i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);
}
// Banana
// Orange
// Apple
// Mango


for(let fruit of fruits){
    console.log(fruit);
}
// Banana
// Orange
// Apple
// Mango
```

Variables

Variable is Containers for storing data values.

Declaration: var, let, const.

We can use the var or let keywords to declare variables

```
let name = "Chirag";
var name = "Chirag";
```

A constant is a type of variable whose value cannot be changed.

```
const name = "Chirag";
name = "Bhangale";
console.log(name)           // Error! constant cannot be changed
```

Scope: Global vs. Local.

Global Variable: Outside of all the functions.

Local Variable: Within a function block.

Operators

Operators is Symbols that perform operations on variables and values.

Types: Arithmetic, Assignment, Comparison, Logical, Bitwise.

1. Comparison Operators:

```
let x = 5;
let y = "5";
```

```
console.log(x == y);          // true checks only value
console.log(x === y);         // false check both datatype and value
console.log(x != y);          // false
console.log(x !== y);         // true
console.log(x >= 5);          // true
console.log(x <= 5);          // true
```

2. Logical Operators:

```
let isTrue = true;
let isFalse = false;
console.log(isTrue && isFalse);      // false
console.log(isTrue || isFalse);      //  true
console.log(!isTrue);                // false
```

3. Assignment Operators:

```
let num = 10;
a. num += 5;          // num = num + 5
console.log(num);     // 15

b. num -= 3;          // num = num - 3
console.log(num);     // 12

c. num *= 2;          // num = num * 2
console.log(num);     // 24

d. num /= 4;          // num = num / 4
```

```
console.log(num);   // 6


e. num %= 2;          // num = num % 2

console.log(num);   // 0
```

4. Arithmetic Operators:

It is Used to perform arithmetic on numbers.

List: +, -, *, /, %, ++, --.

Example:

```
let a = 10;

let b = 5;

a. console.log(a + b);        // 15

b. console.log(a - b);       // 5

c. console.log(a * b);       // 50

d. console.log(a / b);       // 2

e. console.log(a % b);       // 0
```

Promises

Promises are objects that represent the eventual result of an asynchronous operation,allowing for cleaner and more manageable asynchronous code.

A JavaScript Promise object can be:

Pending

Fulfilled

Rejected

The Promise object supports two properties: state and result.

While a Promise object is "pending" (working), the result is undefined.

When a Promise object is "fulfilled", the result is a value.

When a Promise object is "rejected", the result is an error object.

Eg:

```
let promise = new Promise((resolve, reject) => {
    let success = true;
    if (success) {
        resolve("Correct!");
    } else {
        reject("Incorrect");
    }
    });
promise.then((message) => console.log(message)) // "Correct!"
    .catch((error) => console.error(error)); // "Incorrect"
```

setTimeout() :

This method executes a function, after waiting a specified number of milliseconds.

Syntax:

```
    setTimeout(function, delay in milliseconds);
```

Eg.

```
function hello(){
    console.log("Hello");
    }
    setTimeout(hello,2000);
```

Output:

```
     Hello        //will print after 2 seconds
```

setInterval() :

The setInterval() method calls a function at specified intervals (in milliseconds). It continues calling the function until clearInterval() is called or the window is closed. This method is useful for tasks that need periodic execution, like updating animations or refreshing data.

Syntax :

```
setInterval(function, milliseconds);
```

Eg.

```
function hello() {
   console.log('Hello);
}
setInterval(hello, 1000);
Output:     Hello
            Hello   …. // will keep printing in every 1 sec
```