JavaScript

Introduction

JavaScript was initially created to “make web pages alive”. The programs in this language are called scripts. They can be written right in a web page’s HTML and run automatically as the page loads. Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.

Variables

Variables in JavaScript are used to store data values. JavaScript provides three ways to declare variables:

- **var:** Function-scoped meaning its scope is confined to the function where it is declared and can be redeclared.
- **let:** Block-scoped meaning its scope is confined to the block, statement, or expression in which it is used and cannot be redeclared within the same scope.
- **const:** Block-scoped but immutable; it cannot be reassigned after being declared.

Example:

```
function testVar() {
  var a = 30;
  if (true) {
    var a = 40;
    console.log(a); // 40
  }
  console.log(a); // 40
}

function testLet() {
  let a = 30;
  if (true) {
    let a = 40;
    console.log(a); // 40
  }
  console.log(a); // 30
}
```

Operators

JavaScript supports a variety of operators for performing operations on variables and values. These include arithmetic operators, comparison operators, logical operators, and assignment operators.

Arithmetic Operators:

Used to perform arithmetic calculations.

- **+:** Addition
- **-:** Subtraction
- ***:** Multiplication
- **/:** Division
- **%:** Modulus (remainder)
- **++:** Increment
- **--:** Decrement

Comparison Operators:

Used to compare two values.

- **==:** Equal to
- **===:** Strict equal (both value and type)
- **!=:** Not equal
- **!==:** Strict not equal
- **>:** Greater than
- **<:** Less than
- **>=:** Greater than or equal to
- **<=:** Less than or equal to

Logical Operators:

Used to combine multiple conditions.

- **&&:** Logical AND
- **||:** Logical OR
- **!:** Logical NOT

Example:

```
let a = 10;
let b = 20;

console.log(a + b); // 30
```

```
console.log(a > b); // false
console.log(a < b && a != b); // true
```

Data Types

JavaScript supports dynamic typing, meaning a variable can hold different types of values. There are two types of data types:

Primitive Data Types:

- **Number**: Represents numeric values, such as 5, 10.5, 10e5.
- **String**: Represents text, such as "Hello".
- **Boolean**: Represents true or false values.

Non-Primitive Data Types:

- **Object**: Represents collections of key-value pairs.
- **Array**: Special type of object used for storing ordered data.

Example:

```
let num = 42; // Number
let text = "Hello"; // String
let isActive = true; // Boolean
let y={firstName:"John", lastName:"Doe", age:50}
let fruits = ["Apple", "Banana", "Mango"]
```

Functions

Functions are blocks of reusable code. A function is defined with the function keyword, followed by a name and a set of parentheses ().

Simple Function Declaration:

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

Example:

```
function greet(name) {
  return 'Hello ' +name+ '!';
}
console.log(greet("John")); // "Hello, John!"
```

Default Parameters:

You can set default values for function parameters.

Example:

```
function greet(name = "Guest") {
  return 'Hello ' +name+ '!'; }
console.log(greet()); // "Hello, Guest!"
```

Rest Parameters

The rest parameter syntax allows us to represent an indefinite number of arguments as an array.

Example:

```
function sumNums(...nums) {
let sum = 0;
for (let i = 0; i < nums.length; i++)
sum += nums[i];
return sum;
}
console.log(sumNums(1, 1, 1, 1, 1));
```

Pass by Value

JavaScript is Pass by Value for Primitive Data (Number, String, Boolean)

Example:

```
function sum(n1, n2) {
n1 = n1 + n2;
return n1;
}
var a = 10, b = 10
console.log(a, "+", b, "=", sum(a, b))
```

Pass by Reference

Objects are of Reference Type therefore, JavaScript is Pass by Reference for Objects.

Example:

```
function strFun(p1) {
p1.value = "Eleven"
}
var a = { value: "Ten" }
strFun(a)
console.log(a.value)
```

Function Expression:

Functions can also be assigned to variables.

Example:

```
const sum = function(a, b) {
  return a + b;
};
console.log(sum(5, 10)); // 15
```

Arrow Functions

Arrow functions are a shorter syntax for writing functions introduced in ES6. They do not have their own this value, which makes them useful in some contexts like event handlers.

Example:

```
const greet = (name) => {
  return 'Hello ' +name+ '!';
};
console.log(greet("John")); // "Hello, John!"
```

For simple functions with one expression, the curly braces and return statement can be omitted:

```
const greet = name => 'Hello ' +name+ '!';
console.log(greet("Jane")); // "Hello, Jane!"
```

Arrow functions with multiple parameters:

```
const sum = (a, b) => a + b;
console.log(sum(5, 7)); // 12
```

Arrays

An array is a special variable, which can hold more than one value at a time. Arrays are zero-indexed, meaning the first element is at index 0.

Array Declaration:

Example:

```
let fruits = ["Apple", "Banana", "Mango"];
```

Accessing Array Elements:

Example:

```
console.log(fruits[0]); // "Apple"
```

Array Methods:

- push(): Adds a new element to the end.
- pop(): Removes the last element.
- shift(): Removes the first element.
- unshift(): Adds a new element to the beginning.
- forEach(): Loops through array elements.
- map(): Creates a new array by applying a function to each element.
- concat(): method concatenates (joins) two or more arrays.
- copyWithin(): method copies array elements to another position in an array, overwriting the existing values.
- find(): method returns the value of the array element that passes a test.
- sort(): method sorts an array alphabetically.
- reverse(): method reverses the elements in an array.

Example:

```
let numbers = [1, 2, 3, 4, 5];
let numbers1 = [6, 7, 8, 9, 0];
numbers.push(6); // Adds 6 at the end
numbers.pop();   // Removes 6
numbers.map(num => num * 2); // [2, 4, 6, 8, 10]
console.log(numbers.concat(numbers1))
```

```
const numbers = [1, 2, 3, 4, 5];
let sum = 0;
numbers.forEach(num =>
{ sum += num; }
);
console.log(sum); // 15
```