Table Of Content

1. Comments in JavaScript

Comments are used to explain JavaScript code, and to make it more readable. They are ignored by the JavaScript engine during execution.

- **Single-line comment:**
  Syntax: // comment here Example:

  ```
  // This is a single-line comment
  ```

- **Multi-line comment:**
  Syntax: /* comment here */ Example:

  ```
  /* This is a multi-line comment
  It can span across multiple lines */
  ```

2. Identifiers in JavaScript

An identifier is a name used to identify variables, functions, objects, or other entities.

- **Rules for naming identifiers:**
  - Must start with a letter (a-z, A-Z), underscore (_), or dollar sign ($).
  - Identifiers can include numbers (0-9).
  - Cannot use JavaScript reserved keywords (e.g., var, let, function).
- **Examples:**

```
let myVariable;  // valid
let $amount; // valid
let _name;  // valid
let 1name;  // invalid
```

3. Variables in JavaScript

Variables are containers used to store data values.

- **Types of variable declarations:**
  - **var**: Function-scoped, can be redeclared.
  - **let**: Block-scoped, cannot be redeclared in the same scope.
  - **const**: Block-scoped, must be initialized and cannot be reassigned.
- **Examples:**

```
var age = 25;      // using var
```

```
let name = "John"; // using let

const PI = 3.14;   // using const
```

Difference Between var, let, and const

Feature	var	let	const
Scope	Function-scoped	Block-scoped	Block-scoped
Hoisting	Hoisted and initialized as undefined	Hoisted but not initialized	Hoisted but not initialized
Re-declaration	Allowed in the same scope	Not allowed in the same scope	Not allowed in the same scope
Re-assignment	Allowed	Allowed	Not allowed
Use Case	Used in older JavaScript code	Recommended for variables that change	Recommended for constants

4. Operators in JavaScript

Operators are used to perform operations on variables and values.

- **Types of operators:**
  - **Arithmetic operators:** +, -, *, /, %
  - **Assignment operators:** =, +=, -=, *=, /=
  - **Comparison operators:** ==, ===, !=, !==, >, <, >=, <=
  - **Logical operators:** &&, ||, !
  - **Unary operators:** ++, --, typeof, void
  - **Ternary operator:** condition ? expr1 : expr2
- **Examples:**

```
let x = 5, y = 10;

let sum = x + y;     // Addition operator

let result = x > y;  // Comparison operator

let isValid = true && false; // Logical operator
```

5. JavaScript Data Types

1. Primitive Data Types

Primitive types store single, simple values and are immutable (they don’t change once created).

Data Type	Description	Example
String	Used to represent text. Strings are written in quotes (single or double).	let name = "Shruthi";
Number	Represents both whole numbers and decimal values. No separate type for int or float.	let age = 25;
Boolean	Represents true or false. Used in conditions or logic checks.	let isStudent = true;
Undefined	A variable that is declared but not assigned a value will be undefined.	let x; // x is undefined
Null	Used to represent "no value" or "empty value". It’s intentionally set by the programmer.	let data = null;
BigInt	Used for very large integers that can't be represented by the Number type.	let bigNum = 123456789012345678901n;
Symbol	Represents a unique value, often used as a unique object key. Each symbol is different.	let id = Symbol("userId");

---

2. Non-Primitive (Reference) Data Types

These types are used to store **multiple values or complex structures**. They are **mutable**, meaning their values can be changed.

Data Type	Description	Example
Object	A collection of key-value pairs. Used to group related data. Keys are strings, values can be any type.	let user = { name: "Shruthi", age: 25 };
Array	An ordered list of values (can be of different types). Values are accessed by index.	let colors = ["red", "blue", "green"];
Function	A block of reusable code that performs a task. Functions can be stored in variables or passed as arguments.	function greet() { console.log("Hi"); }

---

6. Array Methods

Method	Description	Example
push()	Adds element at end	arr.push(4);
pop()	Removes last element	arr.pop();
shift()	Removes first element	arr.shift();
unshift()	Adds element at start	arr.unshift(0);
length	Returns number of elements	arr.length
indexOf()	Finds index of a value	arr.indexOf(2);
includes()	Checks if value exists	arr.includes(3);
slice()	Returns a portion of array	arr.slice(1, 3);
splice()	Adds/removes elements	arr.splice(1, 1, 99);
forEach()	Executes function for each item	arr.forEach(el => console.log(el));

Example:

```
let arr = [1, 2, 3];
arr.push(4);        // [1, 2, 3, 4]
arr.pop();          // [1, 2, 3]
console.log(arr.length); // 3
```

7. String Methods

Method	Description	Example
length	Returns number of characters	str.length
toUpperCase()	Converts to uppercase	str.toUpperCase()
toLowerCase()	Converts to lowercase	str.toLowerCase()
charAt()	Returns character at index	str.charAt(1)
indexOf()	Finds index of first match	str.indexOf("a")
includes()	Checks if substring exists	str.includes("hello")

Method	Description	Example
slice()	Extracts a part of the string	str.slice(0, 4)
replace()	Replaces a substring with newstring	str.replace("old", "new")
trim()	Removes whitespace	str.trim()

Examples:

```
let str = "  Hello World! ";
console.log(str.trim());          // "Hello World!"
console.log(str.toUpperCase());    // "  HELLO WORLD! "
console.log(str.includes("Hello")) // true
```

8. Conditional Statements in JavaScript

Conditional statements allow us to execute code based on certain conditions.

- **if statement:**

```
if (condition) {
  // Executes if condition is true
}
```

- **if-else statement:**

```
if (condition) {
  // Executes if condition is true
} else {
  // Executes if condition is false
}
```

- **else if statement:**

```
if (condition1) {
  // Executes if condition1 is true
} else if (condition2) {
  // Executes if condition2 is true
} else {
```

```
  // Executes if none of the above conditions are true
}
```

- **switch statement:** The switch statement is used to perform different actions based on different conditions.

```
switch (expression) {
  case value1:
    // Executes if expression === value1
    break;
  case value2:
    // Executes if expression === value2
    break;
  default:
    // Executes if no case matches
}
```

9. Looping Statements in JavaScript

Loops are used to repeat a block of code multiple times.

- **for loop:**
  Used when the number of iterations is known.

```
for (let i = 0; i < 5; i++) {
  console.log(i); // Prints 0 to 4
}
```

- **while loop:**
  Repeats as long as the condition is true.

```
let i = 0;
while (i < 5) {
  console.log(i); // Prints 0 to 4
  i++;
}
```

- **do while loop:**
  Executes at least once, then repeats as long as the condition is true.

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

```
  console.log(i); // Prints 0 to 4
  i++;
} while (i < 5);
```

- **for in loop:**
  Iterates over the properties of an object.

```
let person = { name: "John", age: 30 };
for (let key in person) {
  console.log(key + ": " + person[key]); // Prints name and age
}
```

- **for of loop:**
  Iterates over the values in an iterable object (like an array).

```
let arr = [1, 2, 3, 4];
for (let num of arr) {
console.log(num); // Prints 1 to 4
 }
```