Session 1: JavaScript Basics

Goal: Understand core JavaScript syntax and concepts to build a foundation.

- **Variables: let, const, var**
- **JavaScript Data Types: Strings, Numbers, Booleans, Arrays, Objects**
- **Operators: Arithmetic, Comparison, Logical**
- **Conditional Statements : if, else, switch.**
- **Functions: Declaration, Expression**

Variables: let, const, var

In JavaScript, variables store data like numbers, text, or objects. We use let, const, and var to declare variables. Each has different rules and scope

1. var

- **What**: Old way to declare variables. Can be reassigned and redeclared.
- **Scope**: Function scope (available inside the function it's declared in) or global scope (if declared outside).
- **Issue**: Can cause bugs due to redeclaration and hoisting (variable is "moved" to the top of scope).
- **Use**: Rarely used in modern JavaScript.

```
var name = "Rahul";
var name = "Priya"; // Redeclaring is okay
name = "RAM"; // Reassigning is okay
console.log(name); // Output: RAM
```

2. let

- **What:** Modern way to declare variables. Can be reassigned but not redeclared in the same scope.
- **Scope:** Block scope (available only inside {} where it's declared, like in loops or if blocks).
- **Use:** Good for variables that change value.

```
let age = 25;
age = 30; // Reassigning is okay
// let age = 40; // Error: Cannot redeclare
console.log(age); // Output: 30

if (true) {
  let city = "Delhi";
  console.log(city); // Output: Delhi
}
// console.log(city); // Error: city is not defined (outside
block)
```

3. const

- **What**: Declares variables that cannot be reassigned or redeclared.
- **Scope**: Block scope (like let).
- **Note**: For objects/arrays, the content can change, but the variable cannot point to a new object/array.
- **Use**: Use for values that won't change, like constants or fixed references.

```
const pi = 3.14;
// pi = 3.14159; // Error: Cannot reassign
console.log(pi); // Output: 3.14

const person = { name: "Sita" };
person.name = "Gita"; // Okay: Changing object property
// person = { name: "Rita" }; // Error: Cannot reassign
console.log(person.name); // Output: Gita
```

Feature	`var`	`let`	`const`
Scope	Function or global	Block ({})	Block ({})
Reassign	Yes	Yes	No
Redeclare	Yes	No	No
Hoisting	Yes (undefined initially)	Yes (but not initialized)	Yes (but not initialized)
Use Case	Avoid in modern code	Changing values	Fixed values/objects

Tips:

- Use const by default for safety.
- Use let when you need to change the value.
- Avoid var to prevent bugs.

JavaScript Data Types: Strings, Numbers, Booleans, Arrays, Objects

In JavaScript, data types define the kind of data a variable can hold.

1. Strings

- **What**: Text or characters (e.g., names, words) enclosed in quotes (', ", or `).
- **Use**: Store names, messages, or any text.

```
let name = 'Rahul'; // Single quotes
let city = "Delhi"; // Double quotes
let greeting = `Hello, ${name}!`; // Template literal
console.log(greeting); // Output: Hello, Rahul!
```

2. Numbers

- **What**: Numeric values, including integers and decimals (no separate type for integers/floats).
- **Use**: For calculations, counts, or measurements.

```
let roll = 25; // Integer
let price = 99.99; // Decimal
let sum = roll+ 5;
console.log(sum); // Output: 30
console.log(price * 2); // Output: 199.98
```

3. Booleans

- **What:** Represents true or false values.
- **Use:** For conditions, comparisons, or toggles.

```
let isStudent = true;
let isAdult = age >= 18;
console.log(isAdult); // Output: true
```

4. Arrays

- **What:** Ordered list of values (can hold any data type) inside square brackets [].
- **Use:** Store multiple values, like a list of names or numbers

```
let fruits = ["Apple", "Mango", "Banana"];
console.log(fruits[0]); // Output: Apple (first item)
console.log(fruits.length); // Output: 3
console.log(fruits); // Output: ["Apple", "Mango", "Banana", "Orange"]
```

5. Objects

- **What**: Collection of key-value pairs inside curly braces {}.
- **Use**: Store related data, like details of a person or product.

```
let student = {
  name: "Priya",
  age: 20,
  isPass: true
};
console.log(student.name); // Output: Priya
student.age = 21; // Update value
console.log(student); // Output: {name: "Priya", age: 21, isPass: true}
```

Operators: Arithmetic, Comparison, Logical

1. Arithmetic Operators

```
let a = 10, b = 5;
console.log(a + b); // Output: 15 (Addition)
console.log(a - b); // Output: 5 (Subtraction)
console.log(a * b); // Output: 50 (Multiplication)
console.log(a / b); // Output: 2 (Division)
console.log(a % b); // Output: 0 (Modulus)
```

2. Comparison Operators

```
let x = 10, y = 20;
console.log(x == y); // Output: false (Equal)
console.log(x != y); // Output: true (Not equal)
console.log(x > y); // Output: false (Greater than)
console.log(x <= y); // Output: true (Less than or equal)
console.log(x === "10"); // Output: false (Strict equal, checks type)
```

3. Logical Operators

```
let isMale = true, hasTicket = false;
console.log(isMale && hasTicket); // Output: false (AND)
console.log(isMale || hasTicket); // Output: true (OR)
console.log(!isMale); // Output: false (NOT)
```

Conditional Statements : if, else, switch.

Conditionals in JavaScript help make decisions based on conditions. They control the flow of your code.

1. if and else

- **What**: Runs code if a condition is true. Use else for alternative code if condition is false.
- **Use**: Check conditions like user input or values.

```
let studentAge = 20;
if (studentAge >= 18) {
  console.log("You can vote!"); // Output: You can vote!
} else {
  console.log("You cannot vote!");
}
```

With else if

```
let marks = 85;
if (marks >= 90) {
  console.log("Grade A");
} else if (marks >= 80) {
  console.log("Grade B"); // Output: Grade B
} else {
  console.log("Grade C");
}
```

2. switch

- **What**: Checks a value against multiple cases and runs matching code. Alternative to multiple if-else.
- **Use**: When you have fixed values to compare (e.g., days, grades).

```
let day = "Monday";
switch (day) {
  case "Monday":
    console.log("Start of week!"); // Output: Start of week!
    break;
  case "Friday":
    console.log("Weekend soon!");
    break;
  default:
    console.log("Some other day!");
}
```

Loops: for, while, forEach

Loops in JavaScript repeat code until a condition is met. They're useful for tasks like iterating over arrays or performing repetitive actions.

1. for Loop

- **What**: Repeats code a specific number of times using a counter.
- **Use**: When you know how many times to loop (e.g., iterating over an array).

```
for (let i = 1; i <= 5; i++) {
  console.log(i); // Output: 1, 2, 3, 4, 5
}
```

2. while Loop

- **What:** Repeat code as long as a condition is true.
- **Use:** When you don't know how many times the loop will run.

```
let count = 1;
while (count <= 3) {
console.log("Count: " + count); // Output: Count: 1, Count: 2, Count: 3
```

```
  count++;
}
```

3. forEach Loop

- **What**: Loops through each item in an array, running a function for each.
- **Use:** Simple way to iterate over arrays without managing a counter.

```
let fruits = ["Apple", "Mango", "Banana"];
fruits.forEach(function(fruit) {
  console.log(fruit); // Output: Apple, Mango, Banana
});
```

Functions: Declaration, Expression

Functions in JavaScript are reusable blocks of code that perform a task. They take inputs (parameters), process them, and can return outputs.

1. Function Declaration

- **What**: A named function defined using the function keyword.
- **Use**: For reusable tasks, callable anywhere in the code

```
function greet(name) {
  return "Hello, " + name + "!";
}
console.log(greet("Rahul")); // Output: Hello, Rahul!
```

2. Function Expression

- **What**: A function assigned to a variable, can be named or anonymous.
- **Use**: For functions used as values

```
const add = function(a, b) {
  return a + b;
};
console.log(add(5, 3)); // Output: 8
```

.