What is JavaScript?

- **JavaScript** is a **high-level, interpreted programming language** used to make web pages **interactive** and **dynamic**.
- It runs directly in the **browser** and can also run on the **server** using **Node.js**.
- It can:
    - Add interactivity (like buttons, animations, forms).
    - Manipulate HTML and CSS.
    - Handle user input and events.
    - Communicate with servers (using APIs).

Example:

```
alert("Hello, JavaScript!");
```

Variables in JavaScript

Variables are used to **store data values**.
There are **three ways** to declare a variable:

Keyword	Reassignment	Redeclaration	Example
`var`	Yes	Yes	`var x = 10;`
`let`	Yes	No	`let y = 20;`
`const`	No	No	`const z = 30;`

Naming Rules for Variables:

- ✓ Variable names are **case-sensitive** (`age` ≠ `Age`).
- ✓ Must **start with** a letter, _, or `$` (not a number).
- ✓ Can **contain letters, digits, underscores, and $**.
- ✓ Cannot use **JavaScript reserved keywords** (like `let`, `if`, `for`, etc.).
- ✓ Use **camelCase** naming convention for readability.

Example:

```
let firstName = "John";
const PI = 3.14;
var age = 25;
```

JavaScript Data Types

JavaScript has **7 primitive data types**:

Type	Description	Example
Number	Numeric values	`let x = 10;`
String	Text inside quotes	`let name = "Neeraj";`

Type	Description	Example
Boolean	True or false values	`let isLoggedIn = true;`
Null	Intentional empty value	`let car = null;`
Undefined	Declared but no value assigned	`let value;`
Symbol	Unique and immutable value	`let id = Symbol("id");`
BigInt	Very large integers	`let bigNum = 12345678901234567890n;`

Type Conversion

Type conversion means **changing data from one type to another**.

A. Implicit Conversion (Type Coercion)

JavaScript automatically converts data types.

Example:

```
let result = "5" + 2;  // "52" (number converted to string)
let sum = "5" - 2;     // 3   (string converted to number)
```

B. Explicit Conversion

Manually converting types using functions.

Conversion	Method	Example	Result
To Number	`Number()`	`Number("10")`	10
To String	`String()`	`String(100)`	"100"
To Boolean	`Boolean()`	`Boolean(0)`	false

JavaScript Operators

Operators are **symbols** used to perform **operations on values and variables**.

A. Arithmetic Operators

Used to perform **mathematical calculations**.

Operator	Description	Example	Output
`+`	Addition	`5 + 3`	`8`
`-`	Subtraction	`5 - 3`	`2`
`*`	Multiplication	`5 * 3`	`15`
`/`	Division	`10 / 2`	`5`
`%`	Modulus (Remainder)	`10 % 3`	`1`

Operator	Description	Example	Output
`**`	Exponentiation	`2 ** 3`	`8`
`++`	Increment	`let x = 5; x++;`	`6`
`--`	Decrement	`let x = 5; x--;`	`4`

Example Code:

```
let a = 10;
let b = 3;

console.log(a + b);  // 13
console.log(a - b);  // 7
console.log(a * b);  // 30
console.log(a / b);  // 3.333...
console.log(a % b);  // 1
console.log(a ** b); // 1000
```

B. Assignment Operators

Used to **assign values** to variables.

Operator	Description	Example	Same as
`=`	Assign	`x = 5`	`x = 5`
`+=`	Add and assign	`x += 3`	`x = x + 3`
`-=`	Subtract and assign	`x -= 2`	`x = x - 2`
`*=`	Multiply and assign	`x *= 4`	`x = x * 4`
`/=`	Divide and assign	`x /= 2`	`x = x / 2`
`%=`	Modulus and assign	`x %= 3`	`x = x % 3`

Example Code:

```
let x = 10;
x += 5;   // x = 15
x -= 3;   // x = 12
x *= 2;   // x = 24
x /= 4;   // x = 6
console.log(x);
```

C. Comparison Operators

Used to **compare two values** and return a **Boolean** (`true` or `false`).

Operator	Description	Example	Result
`==`	Equal to (compares value only)	`5 == "5"`	`true`
`===`	Strict equal (compares value + type)	`5 === "5"`	`false`
`!=`	Not equal	`5 != 8`	`true`
`!==`	Strict not equal	`5 !== "5"`	`true`

Operator	Description	Example	Result
>	Greater than	`10 > 5`	`true`
<	Less than	`10 < 5`	`false`
>=	Greater than or equal to	`10 >= 10`	`true`
<=	Less than or equal to	`5 <= 8`	`true`

Example Code:

```
let a = 10, b = "10";

console.log(a == b);   // true
console.log(a === b);  // false
console.log(a != b);   // false
console.log(a !== b);  // true
console.log(a > 5);    // true
console.log(a <= 10);  // true
```

D. Logical Operators

Used to **combine multiple conditions**.

Operator	Description	Example	Result
`&&`	AND	`(5 > 2 && 3 < 5)`	`true`
`||`	OR	`(5 > 2 || 3 < 5)`	`true`
`!`	NOT	`!(5 > 3)`	`false`

Example Code:

```
let age = 20;
let hasID = true;

if (age >= 18 && hasID) {
  console.log("Allowed to vote");
}

let isStudent = false;
if (isStudent || age < 18) {
  console.log("Discount available");
}

console.log(!true); // false
```

E. Ternary Operator

A **short form of if-else** statement.

Syntax:

```
condition ? valueIfTrue : valueIfFalse
```

Example Code:

```
let age = 18;
let message = (age >= 18) ? "Adult" : "Minor";
console.log(message); // Output: Adult
```

Conditional Statements (If-Else)

The `if...else` statement is used to **make decisions** in a program based on **conditions**.
It checks a condition and executes code **only if** that condition is true.

A. Basic `if` Statement

Syntax:

```
if (condition) {
  // code to run if condition is true
}
```

Example:

```
let age = 18;

if (age >= 18) {
  console.log("You are eligible to vote.");
}
```

Output:

```
You are eligible to vote.
```

B. if...else Statement

Used when we want to run **one block of code if the condition is true**, and another block if it's false.

Syntax:

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

Example:

```
let marks = 45;

if (marks >= 50) {
  console.log("You passed the exam!");
} else {
  console.log("You failed the exam.");
}
```

Output:

```
You failed the exam.
```

C. if...else if...else Statement

Used when there are **multiple conditions** to check.

Syntax:

```
if (condition1) {
  // code if condition1 is true
} else if (condition2) {
  // code if condition2 is true
} else {
  // code if both are false
}
```

Example:

```
let score = 85;

if (score >= 90) {
  console.log("Grade: A");
} else if (score >= 75) {
  console.log("Grade: B");
} else if (score >= 50) {
  console.log("Grade: C");
} else {
  console.log("Grade: Fail");
}
```

Output:

```
Grade: B
```

D. Nested if Statement

An `if` statement **inside another if** is called **nested if**.

Example:

```
let username = "admin";
let password = "1234";

if (username === "admin") {
  if (password === "1234") {
    console.log("Login successful!");
  } else {
    console.log("Incorrect password!");
  }
} else {
  console.log("Invalid username!");
}
```

Output:

```
Login successful!
```

switch-case Statement

The `switch` statement is used to **compare one expression with multiple values** (called *cases*).
It is an alternative to writing many `if...else if` statements.

Syntax:

```
switch (expression) {
  case value1:
    // code block
    break;

  case value2:
    // code block
    break;

  default:
    // code block if no match found
}
```

Example 1: Day of the Week

```
let day = 3;
let dayName;

switch (day) {
  case 1:
    dayName = "Monday";
    break;
  case 2:
    dayName = "Tuesday";
    break;
  case 3:
    dayName = "Wednesday";
    break;
  case 4:
    dayName = "Thursday";
    break;
  case 5:
    dayName = "Friday";
    break;
  case 6:
    dayName = "Saturday";
    break;
  case 7:
    dayName = "Sunday";
    break;
  default:
    dayName = "Invalid day";
}

console.log(dayName);
```

Output:

```
Wednesday
```

Example 2: Simple Calculator

```
let a = 10;
let b = 5;
let operator = "*";
let result;

switch (operator) {
  case "+":
    result = a + b;
    break;
  case "-":
    result = a - b;
    break;
  case "*":
    result = a * b;
    break;
  case "/":
    result = a / b;
    break;
  default:
    result = "Invalid operator";
}

console.log("Result:", result);
```

Output:

```
Result: 50
```

8. Loops in JavaScript

Definition:

A **loop** is used to **repeat a block of code multiple times** until a condition becomes false.

There are different types of loops in JavaScript:

1. `for` loop
2. `while` loop
3. `do...while` loop
4. `for...of` loop
5. `for...in` loop

A. for Loop

Used when the number of repetitions is **known** in advance.

Syntax:

```
for (initialization; condition; update) {
  // code to execute
}
```

Example:

```
for (let i = 1; i <= 5; i++) {
  console.log("Number:", i);
}
```

Output:

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

B. while Loop

Used when the number of iterations is **not known** in advance.
The loop runs **while the condition is true**.

Syntax:

```
while (condition) {
  // code to execute
}
```

Example:

```
let i = 1;
while (i <= 5) {
  console.log("Count:", i);
  i++;
}
```

Output:

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

C. do...while Loop

The **do...while** loop runs the code **at least once**, even if the condition is false.

Syntax:

```
do {
  // code to execute
} while (condition);
```

Example:

```
let i = 1;
```

```
do {
  console.log("Value:", i);
  i++;
} while (i <= 3);
```

Output:

```
Value: 1
Value: 2
Value: 3
```

Example (Condition False Initially):

```
let x = 10;
do {
  console.log("Runs at least once!");
} while (x < 5);
```

Output:

```
Runs at least once!
```

D. for...of Loop

Used to **iterate through values** of **arrays, strings, or other iterable objects**.

Syntax:

```
for (let variable of iterable) {
  // code to execute
}
```

Example 1: Array

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

for (let fruit of fruits) {
  console.log(fruit);
}
```

Output:

```
Apple
Banana
Cherry
```

Example 2: String

```
let name = "Neeraj";

for (let char of name) {
  console.log(char);
}
```

Output:

```
N
e
e
r
a
j
```

E. for...in Loop

Used to **iterate over the keys (property names)** of an **object**.

Syntax:

```
for (let key in object) {
  // code to execute
}
```

Example:

```
let person = {
  name: "Rahul",
  age: 23,
  city: "Lucknow"
};

for (let key in person) {
  console.log(key + ":", person[key]);
}
```

Output:

```
name: Rahul
age: 23
city: Lucknow
```

9. break and continue Statements

Both are **loop control statements** used to change how loops work.

A. break Statement

- The `break` statement **stops** the loop **immediately** — even if the condition is still true.
- Control jumps out of the loop.

Syntax:

```
break;
```

Example:

```
for (let i = 1; i <= 10; i++) {
  if (i === 5) {
    break;  // Stop loop when i = 5
  }
```

```
  console.log(i);
}
```

Output:

```
1
2
3
4
```

Explanation: The loop stops completely when `i` becomes 5.

B. continue Statement

- The `continue` statement **skips the current iteration** and moves to the **next iteration** of the loop.

Syntax:

```
continue;
```

Example:

```
for (let i = 1; i <= 5; i++) {
  if (i === 3) {
    continue;  // Skip 3
  }
  console.log(i);
}
```

Output:

```
1
2
4
5
```

Explanation: When `i = 3`, the loop skips printing and continues with the next value.

Example with while loop

```
let i = 0;

while (i < 5) {
  i++;
  if (i === 2) continue;
  if (i === 4) break;
  console.log(i);
}
```

Output:

```
1
3
```