JavaScript Introduction

JavaScript is a versatile, high-level programming language primarily used for creating interactive and dynamic web pages. It enables developers to implement features like form validation, animations, event handling, and API interactions. JavaScript can run on both the client-side (browser) and server-side (e.g., Node.js).

Features of JavaScript:

1. Interpreted language (runs directly in the browser).
2. Supports object-oriented, functional, and imperative paradigms.
3. Cross-platform compatibility.
4. Provides asynchronous programming capabilities with Promises and `async/await`.

Syntax

JavaScript syntax is the set of rules defining how the language is written.

Basic Syntax Rules:

- Statements end with a semicolon (`;`) (optional but recommended for clarity).
- Case-sensitive (e.g., `let` is different from `Let`).
- Code blocks are enclosed in `{}`.
- Functions are defined using `function` or arrow (`=>`) syntax.

Example:

```
let name = "John"; // Variable declaration
if (name) {
    console.log("Hello, " + name); // Prints Hello, John
}
```

Comments

Comments are used to make the code more readable and explain its purpose. They are ignored by the JavaScript engine.

Types of Comments:

1. **Single-line comment:** Starts with `//`.
2. **Multi-line comment:** Enclosed in `/* */`.

Examples:

```
// This is a single-line comment
let x = 5; /* Multi-line comment explaining
the purpose of the variable */
```

Variables

Variables store data for use in a program. JavaScript uses three keywords for declaring variables: `var`, `let`, and `const`.

Declaring Variables:

```
var a = 10; // Function-scoped
let b = 20; // Block-scoped
const c = 30; // Block-scoped and read-only
```

`let`

The `let` keyword declares block-scoped variables.

Characteristics:

- Can be updated but not re-declared in the same scope.
- Supports block scope (e.g., inside `if` or `for`).

Example:

```
let x = 5;
if (true) {
    let x = 10; // New variable in block scope
    console.log(x); // 10
```

```
}
console.log(x); // 5
```

`const`

The `const` keyword is used to declare variables with values that cannot be reassigned.

Characteristics:

- Block-scoped like `let`.
- Must be initialized during declaration.
- Does not make objects immutable but prevents re-assignment.

Example:

```
const pi = 3.14;
console.log(pi); // 3.14

// pi = 3.15; // Error: Assignment to constant variable
```

Data Types

JavaScript supports primitive and non-primitive data types.

Primitive Data Types:

1. **String:** Sequence of characters (`"Hello"`, `'World'`).
2. **Number:** Numeric values (`10`, `3.14`).
3. **Boolean:** Logical values (`true`, `false`).
4. **Undefined:** Variable declared but not assigned.
5. **Null:** Intentional absence of value.

Non-Primitive Data Types:

1. **Object:** Key-value pairs.
2. **Array:** Ordered list of values.
3. **Function:** Reusable blocks of code.

Examples:

```
let age = 25;           // Number
let isStudent = true;   // Boolean
let name = "Alice";     // String
let value = null;       // Null
let person = {name: "John", age: 30}; // Object
```

Operators

1. Arithmetic Operators

These operators are used to perform mathematical calculations.

Operator	Description	Example	Result
`+`	Addition	`10 + 5`	`15`
`-`	Subtraction	`10 - 5`	`5`
`*`	Multiplication	`10 * 5`	`50`
`/`	Division	`10 / 5`	`2`
`%`	Modulus (Remainder)	`10 % 3`	`1`
`++`	Increment (by 1)	`let x = 5; x++;`	`6`
`--`	Decrement (by 1)	`let x = 5; x--;`	`4`

Example Code:

```
let a = 10, b = 3;
console.log(a + b);  // 13
console.log(a % b);  // 1
let c = 5;

// 5 (post-increment, prints first, then increments)
console.log(c++);
```

```
// 7 (pre-increment, increments first, then prints)
console.log(++c);
```

---

2. Assignment Operators

These assign values to variables and can combine with arithmetic operations.

Operator	Description	Example	Result
`=`	Assign	`a = 10`	`a = 10`
`+=`	Add and assign	`a += 5`	`a = 15`
`-=`	Subtract and assign	`a -= 5`	`a = 5`
`*=`	Multiply and assign	`a *= 2`	`a = 20`
`/=`	Divide and assign	`a /= 2`	`a = 5`
`%=`	Modulus and assign	`a %= 3`	`a = 1`

Example Code:

```
let x = 10;
x += 5;  // x = 15
x *= 2;  // x = 30
console.log(x);
```

---

3. Comparison Operators

These compare two values and return a Boolean (`true` or `false`).

Operator	Description	Example	Result
`==`	Equal to	`10 == "10"`	`true`
`===`	Strict equal (type + value)	`10 === "10"`	`false`
`!=`	Not equal	`10 != "5"`	`true`

Operator	Description	Example	Result
`!==`	Strict not equal	`10 !== "10"`	`true`
`>`	Greater than	`10 > 5`	`true`
`<`	Less than	`10 < 5`	`false`
`>=`	Greater than or equal to	`10 >= 10`	`true`
`<=`	Less than or equal to	`10 <= 5`	`false`

Example Code:

```
console.log(10 == "10");   // true
console.log(10 === "10");  // false (strict equality)
console.log(5 > 3);        // true
console.log(5 !== "5");    // true
```

---

4. Logical Operators

Used to combine or invert Boolean values.

Operator	Description	Example	Result
`&&`	Logical AND	`true && false`	`false`
`||`	Logical OR	`true || false`	`true`
`!`	Logical NOT	`!true`	`false`

Example Code:

```
let a = 10, b = 5, c = 15;
console.log(a > b && c > a);  // true (both conditions are true)
console.log(a > c || b < c);  // true (one condition is true)
console.log(!(a < b));        // true (not false)
```

---

5. Bitwise Operators

Operate directly on binary representations of numbers.

Operator	Description	Example	Binary Operation	Result
`&`	AND	`5 & 3`	`0101 & 0011 = 0001`	`1`
`|`	OR	`5 | 3`	`0101 | 0011 = 0111`	`7`
`^`	XOR	`5 ^ 3`	`0101 ^ 0011 = 0110`	`6`
`~`	NOT (One's Complement)	`~5`	`~0101 = 1010` (32-bit)	`-6`
`<<`	Left Shift	`5 << 1`	`0101 << 1 = 1010`	`10`
`>>`	Right Shift	`5 >> 1`	`0101 >> 1 = 0010`	`2`

Example Code:

```
console.log(5 & 3);  // 1 (binary AND)
console.log(5 | 3);  // 7 (binary OR)
console.log(5 << 1); // 10 (left shift)
```

---

6. Ternary (Conditional) Operator

The ternary operator provides a shorthand for `if-else`.

Syntax:

```
condition ? value_if_true : value_if_false;
```

Example:

```
let age = 18;
let access = age >= 18 ? "Allowed" : "Denied";
console.log(access); // "Allowed"
```

---

7. Type Operators

Used to identify or manipulate types of variables.

Operator	Description	Example	Result
`typeof`	Returns variable's type	`typeof 5`	`"number"`
`instanceof`	Checks object type	`[] instanceof Array`	`true`

Example Code:

```
console.log(typeof "hello"); // "string"
console.log([] instanceof Array); // true
```

Conditional Statements

Conditional statements execute different blocks of code based on conditions.

Types:

1. **if Statement**
2. **if-else Statement**
3. **if-else if-else Statement**
4. **Switch Statement**

Examples:

```
let age = 18;

// if-else
if (age >= 18) {
    console.log("Eligible to vote");
} else {
    console.log("Not eligible to vote");
}



// Switch
let grade = 'B';
switch (grade) {
    case 'A':
```

```
        console.log("Excellent");
        break;
    case 'B':
        console.log("Good");
        break;
    default:
        console.log("Average");
}
```

Loops

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

Types:

1. **for Loop**
2. **while Loop**
3. **do-while Loop**
4. **for...in Loop** (iterates over object properties)
5. **for...of Loop** (iterates over iterable objects)

Examples:

1. **for Loop:**

```
for (let i = 0; i < 5; i++) {
    console.log(i); // 0, 1, 2, 3, 4
}
```

2. **while Loop:**

```
let count = 0;
while (count < 5) {
    console.log(count); // 0, 1, 2, 3, 4
    count++;
}
```

3. **do-while Loop:**

```
let num = 0;
do {
```

```
    console.log(num); // Executes at least once
    num++;
} while (num < 5);
```

4. **for...in Loop:**

```
let obj = {a: 1, b: 2};
for (let key in obj) {
    console.log(key + ": " + obj[key]);
}
```

5. **for...of Loop:**

```
let arr = [10, 20, 30];
for (let value of arr) {
    console.log(value); // 10, 20, 30
}
```