Core JavaScript

---

1. Introduction

JavaScript is a **client-side scripting language** used to make web pages interactive. It can manipulate the content, respond to user actions, and communicate with servers. Runs in **browser** and optionally on **server (Node.js)**.

Syntax Example:

```
<script>
  console.log("Hello JavaScript"); // Prints message in browser console
</script>
```

---

2. Variables & Constants

Explanation:

- `var`: Function-scoped variable; can be re-declared.
- `let`: Block-scoped; safer to use.
- `const`: Block-scoped constant; cannot be reassigned.

Syntax:

```
var x = 10;
let y = 20;
const PI = 3.14;
```

---

3. Data Types

Explanation:
JavaScript has **primitive** and **reference types**:

- Primitive: store single values (`string`, `number`, `boolean`, `null`, `undefined`, `symbol`, `bigint`)
- Reference: store collections/objects (`object`, `array`, `function`)

Syntax:

```
let name = "Ali";      // string
let age = 25;          // number
let isStudent = true;  // boolean
let empty = null;      // null
let notDefined;        // undefined
let big = 123456789012345678901n; // bigint
```

---

4. Operators

Explanation:

- **Arithmetic:** perform math (`+`, `-`, `*`, `/`, `%`, `**`)
- **Assignment:** store values (`=`, `+=`, `-=`)
- **Comparison:** check equality (`==`, `===`, `!=`, `!==`, `>`, `<`)
- **Logical:** combine conditions (`&&`, `||`, `!`)
- **Ternary:** shorthand `if` (`condition ? value1 : value2`)

Syntax:

```
let a = 10, b = 5;
console.log(a + b);      // 15
console.log(a > b);      // true
console.log(a === "10"); // false
```

---

5. Control Flow

Explanation:

- `if-else`: Executes code if a condition is true.
- `switch`: Handles multiple possible values for a variable.

Syntax:

```
let num = 2;
if (num === 1) console.log("One");
else if (num === 2) console.log("Two");
else console.log("Other");

switch(num) {
  case 1: console.log("One"); break;
  case 2: console.log("Two"); break;
  default: console.log("Other");
}
```

---

6. Loops

Explanation:
Loops repeat code until a condition is met. Useful for arrays, objects, or repeated tasks.

- `for`: Counted loop
- `while`: Checks before executing
- `do...while`: Executes at least once
- `for...of`: Iterate array values
- `for...in`: Iterate object keys

Syntax:

```
for (let i = 0; i < 5; i++) console.log(i);
```

```
let j = 0;
while (j < 5) { console.log(j); j++; }

let arr = [10, 20, 30];
for (let val of arr) console.log(val);

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

---

7. Functions

Explanation:
Functions group code for **reuse**. Can take parameters and return values.

- Function declaration: `function name() {}`
- Function expression: `let name = function() {}`
- Arrow function: `(params) => {}` (short syntax)

Syntax:

```
function add(a, b) { return a + b; }
let sub = function(a, b) { return a - b; }
let mul = (a, b) => a * b;

console.log(add(2,3)); // 5
```

---

8. Scope & Hoisting

Explanation:

- **Scope** determines **where variables are accessible**.
    - Global: anywhere in script

    - Local: inside function/block

- **Hoisting**: `var` variables and function declarations are **moved to top** of scope during compilation.

Syntax:

```
var a = 10; // global
function test() {
  let b = 20; // local
  console.log(a, b);
}
```

---

9. Arrays

Explanation:
Arrays store multiple values. Methods allow adding/removing items, transforming data, etc.

Syntax:

```
let arr = [1, 2, 3];
arr.push(4); // add at end
arr.pop();   // remove last

let doubled = arr.map(x => x*2);  // transform
let evens = arr.filter(x => x%2==0); // filter
let sum = arr.reduce((a,b)=>a+b,0); // sum all
```

---

10. Objects

Explanation:
Objects store key-value pairs; keys are strings and values can be anything. Can include **methods** (functions inside objects).

Syntax:

```
let person = {
  name: "Sara",
  age: 22,
```

```
  greet: function() { console.log("Hello " + this.name); }
};
person.greet();
```

---

11. Strings

Explanation:
Strings are sequences of characters. JavaScript provides **methods** to manipulate them.

Syntax:

```
let str = " JavaScript ";
console.log(str.length);       // length
console.log(str.trim());       // remove spaces
console.log(str.toUpperCase()); // uppercase
console.log(str.slice(0,4));    // "Java"
```

---

12. Math & Date

Explanation:

- **Math:** performs calculations (rounding, random numbers, min/max)
- **Date:** get current date/time

Syntax:

```
console.log(Math.random());   // random 0-1
console.log(Math.floor(3.9)); // 3

let now = new Date();
console.log(now.getFullYear());
```

---

13. DOM Manipulation

Explanation:
The **DOM** (Document Object Model) is the browser's representation of HTML. JS can **read and modify it**.

Syntax:

```
<p id="demo">Hello</p>
<script>
  document.getElementById("demo").innerHTML = "Hi!";
  document.getElementById("demo").style.color = "red";
</script>
```

---

14. Events

Explanation:
Events detect **user actions** like clicks, typing, mouseover. JS can respond with functions.

Syntax:

```
<button id="btn">Click Me</button>
<script>
  document.getElementById("btn").addEventListener("click", function(){
    alert("Button Clicked!");
  });
</script>
```

---

15. Timing Functions

Explanation:

- `setTimeout`: Run code after a delay
- `setInterval`: Repeat code at intervals

Syntax:

```
setTimeout(()=>console.log("Hello after 2s"),2000);
setInterval(()=>console.log("Repeating..."),1000);
```

---

16. Error Handling

Explanation:

- `try...catch`: Handle runtime errors gracefully
- `finally`: Runs regardless of error
- `throw`: Manually trigger error

Syntax:

```
try {
  let x = y + 1; // error
} catch(e) {
  console.log("Error: " + e.message);
} finally {
  console.log("Done");
}
```

---

17. JSON

Explanation:
JSON (JavaScript Object Notation) is a **data format** for exchanging info. Use `stringify` to convert object → string, `parse` for string → object.

Syntax:

```
let obj = {name:"Ali", age:25};
let json = JSON.stringify(obj);
let newObj = JSON.parse(json);
console.log(newObj.name);
```

---

18. Fetch API

Explanation:
`fetch()` is used to get **data from APIs**. Returns a **promise**.

Syntax:

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.log(err));
```

---

19. Local Storage

Explanation:
Store small data in browser (**persistent even after refresh**).

- `setItem`, `getItem`, `removeItem`, `clear`

Syntax:

```
localStorage.setItem("username", "Ali");
console.log(localStorage.getItem("username"));
localStorage.removeItem("username");
```

---

20. ES6+ Features

Explanation:

- **Destructuring**: extract array/object values
- **Spread/rest**: copy/merge data or collect parameters
- **Template literals**: `${variable}` inside strings

Syntax:

```
let [a,b] = [10,20];
let obj = {x:1, y:2};
let {x,y} = obj;
```

```
let arr1 = [1,2];
let arr2 = [...arr1, 3,4];

console.log(`Sum: ${a+b}`);
```
