JavaScript Basics

---

4. Functions

- A reusable block of code.

Function Declaration:

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

Arrow Function:

```
const greet = (name) => {
  console.log("Hi " + name);
};
```

---

5. Conditionals

Make decisions using if, else if, and else.

Example:

```
let score = 75;

if (score >= 90) {
  console.log("A");
} else if (score >= 80) {
  console.log("B");
} else {
  console.log("C");
}
```

---

6. Loops

Repeat tasks using loops.

For loop:

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

While loop:

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

---

7. Arrays

Arrays store multiple values.

```
let fruits = ["apple", "banana", "mango"];
console.log(fruits[0]); // "apple"
```

Common JavaScript Array Methods

---

1. push()

Adds an item to the end of the array.

```
fruits.push("orange");
// ["apple", "banana", "mango", "orange"]
```

---

2. pop()

Removes the last item from the array.

```
fruits.pop();
// ["apple", "banana", "mango"]
```

---

3. shift()

Removes the first item from the array.

```
fruits.shift();
// ["banana", "mango"]
```

---

4. unshift()

Adds an item to the start of the array.

```
fruits.unshift("kiwi");
// ["kiwi", "banana", "mango"]
```

---

5. includes()

Checks if an item exists in the array.

```
fruits.includes("banana");
// true
```

---

6. indexOf()

Finds the index of a given item.

```
fruits.indexOf("mango");
// 2
```

---

7. slice(start, end)

Returns a part of the array (without changing the original).

```
let nums = [1, 2, 3, 4];
let part = nums.slice(1, 3);
```

```
// [2, 3]
```

---

8. splice(start, deleteCount, ...items)

Adds/removes items from the array.

```
let nums = [1, 2, 3, 4];
nums.splice(1, 2, 99);
// [1, 99, 4]
```

---

9. forEach()

Loops through each item in the array.

```
fruits.forEach(fruit => {
  console.log(fruit);
});
```

---

10. map()

Creates a new array by transforming each element.

```
let numbers = [1, 2, 3];
let doubled = numbers.map(n => n * 2);
// [2, 4, 6]
```

🔸 Spread Operator (`...`)

Note: Used to expand or copy arrays and objects.

```
const a = [1, 2];
const b = [...a, 3]; // [1, 2, 3]
```

- ✅ Copy arrays/objects without changing the original

- ✅ Merge arrays or objects easily

🔸 Destructuring

Note: Quickly extract values from arrays or objects.

🔹 Array Destructuring

```
const arr = [1, 2];
const [a, b] = arr;
```

- ✅ Cleaner way to assign multiple values
- ✅ Good for function returns

🔹 Object Destructuring

```
const user = { name: "Alex", age: 25 };
const { name, age } = user;
```

- ✅ Great for pulling specific properties
- ✅ Can rename: `const { name: userName } = user;`

🔸 Template Literals (Backticks ``)

Note: Use variables or expressions inside strings easily.

```
const name = "Alex";
console.log(`Hello, ${name}!`);
```

- ✅ Supports multi-line strings
- ✅ Easy string + variable combination

🔸 Ternary Operator (`condition ? true : false`)

Note: Short way to write simple `if...else`.

```
const age = 18;
const msg = age >= 18 ? "Adult" : "Minor";
```

- ✅ One-liner decision making
- ⚠️ Avoid for complex logic (hard to read)

🔸 Short-Circuiting (`&&` and `||`)

Note: Use logic operators to return fallback values.

```
const username = "" || "Guest"; // Guest
```

- `||` returns **first truthy**
- `&&` returns **first falsy or last truthy**

🟨 **Summary**

Expression	Result	Why?
`true && "Alex"`	"Alex"	All truthy → return last
`false && "Alex"`	false	First falsy → return it
`"Name" && 0 && "Done"`	0	0 is falsy → return it
`1 && 2 && 3`	3	All truthy → return last

🔸 Nullish Coalescing (`??`)

Note: Returns the right side **only if the left is null or undefined**.

```
const input = null;
console.log(input ?? "Default"); // "Default"
```

- ✅ More accurate than `||` if you want to keep `""` or `0`

🔸 Optional Chaining (`?.`)

Note: Safely access nested properties.

```
const user = { profile: { name: "Alex" } };
console.log(user.profile?.name); // Alex
console.log(user.address?.city); // undefined (safe!)
```

- ✅ Prevents runtime errors
- ⚠️ Only checks if a path exists, not if it's valid
