🔹 1. JavaScript Fundamentals

✅ Variables (var, let, const)

Variables are **containers for storing data values**. You can think of a variable like a label attached to some data so you can reuse it in your program.

• var is function-scoped, while let and const are block-scoped.

• Use const by default, let when reassigning.

var name = "John"; // function scoped

let age = 25; // block scoped

const country = "India"; // block scoped and constant

| **Keyword** | **Scope** | **Reassignable** | **Hoisted** | **Use Case** |
| --- | --- | --- | --- | --- |
| var | Function | Yes | Yes | Legacy, avoid in modern JS |
| let | Block | Yes | No | Use when value changes |
| const | Block | No | No | Use when value stays constant |

Data Types

JavaScript is **dynamically typed**, meaning you don’t need to declare the type of variable. The interpreter automatically determines it.

• Primitive: string, number, boolean, null, undefined, symbol, bigint

• Reference: object, array, function

**🔹 Two Categories of Data Types:**

| **Category** | **Examples** |
| --- | --- |
| **Primitive** | string, number, boolean, null, undefined, symbol, bigint |
| **Non-Primitive** | object, array, function |

**🟦 A. Primitive Data Types**

**1. String**

Represents textual data.

let city = "Delhi";

let greeting = 'Hello';

let message = `Welcome to ${city}`; // Template literal

Strings can be declared with ", ', or `.

**2. Number**

Represents integers or floating-point values.

let age = 30;

let price = 199.99;

No difference between integers and floats.

**3. Boolean**

Represents true or false values.

let isLoggedIn = true;

let isAdmin = false;

Used in conditions and logic.

**4. Undefined**

A variable that has been declared but not assigned a value.

let score;

console.log(score); // undefined

**5. Null**

Represents an empty or "no value" intentionally.

let data = null;

Commonly used to reset or clear a value.

**6. Symbol (ES6)**

A unique and immutable value often used as object keys.

let sym = Symbol("id");

**7. BigInt (ES11)**

Used for integers larger than Number.MAX\_SAFE\_INTEGER.

let big = 1234567890123456789012345678901234567890n;

**🟦 B. Non-Primitive (Reference) Data Types**

**1. Object**

Used to store collections of data and more complex entities.

let user = {

name: "Alice",

age: 28

};

React heavily uses objects in state and props.

**2. Array**

A special type of object for ordered collections.

Used in React to render lists with .map().

**3. Function**

Also a type of object.

function greet(name) {

return `Hello, ${name}`;

}

Functions are first-class citizens and are crucial in React for defining components and handling events.

**JavaScript is Loosely Typed**

JavaScript allows you to reassign a variable with a value of a different type (unless using const).

let data = 42; // number

data = "forty two"; // string

**🟦 Type Checking: typeof Operator**

You can check the data type of a variable using typeof.

console.log(typeof 42); // "number"

console.log(typeof "hello"); // "string"

console.log(typeof true); // "boolean"

console.log(typeof {}); // "object"

console.log(typeof []); // "object"

console.log(typeof null); // "object" (quirk in JS)

console.log(typeof undefined); // "undefined"

**Best Practices with Variables in Modern JS**

* Use const by default.
* Use let only if you need to reassign.
* Avoid var.

Operators

• Arithmetic: +, -, \*, /

• Comparison: ==, ===, !=, !==, <, >

• Logical: &&, ||, !

if (isLoggedIn && isAdmin) { ... }

🔹 2. Control Structures

Conditionals

if (age > 18) {

  console.log("Adult");

} else {

  console.log("Minor");

}

Switch Statement

switch(day) {

  case "Monday":

    break;

  default:

    break;

}

Loops

• for, while, do...while, for...of, for...in

for (let i = 0; i < 5; i++) {

  console.log(i);

}

🔹 3. Functions (Essential for Components)

Function Declaration

function greet(name) {

  return `Hello, ${name}`;

}

Function Expression

const greet = function(name) {

  return `Hello, ${name}`;

};

Arrow Functions (Very common in React)

const greet = (name) => `Hello, ${name}`;

🔹 4. Arrays and Array Methods (Very useful for JSX rendering)

Common Methods:

• map() → for rendering lists

• filter()

• forEach()

• reduce()

• find()

const numbers = [1, 2, 3];

const doubled = numbers.map(n => n \* 2);  // [2, 4, 6]

🔹 5. Objects

Object creation & access

const user = {

  name: "Alice",

  age: 25

};

console.log(user.name);

Destructuring

const { name, age } = user;

Spread and Rest Operators

const newUser = { ...user, location: "India" };

🔹 6. DOM Manipulation (less used in React, but useful for understanding)

• document.querySelector

• addEventListener

• innerHTML, textContent

React handles the DOM via virtual DOM, so direct DOM manipulation is rare.

🔹 7. Events

Event Handling

function handleClick() {

  console.log("Clicked");

}

<button onClick={handleClick}>Click Me</button>

🔹 8. ES6+ Features (Must Know for React)

Template Literals

const name = "React";

console.log(`Welcome to ${name}`);

Default Parameters

function greet(name = "User") {

  return `Hello ${name}`;

}

Destructuring (Array/Object)

const [first, second] = [1, 2];

const { title } = { title: "React Guide" };

Spread / Rest

const arr1 = [1, 2];

const arr2 = [...arr1, 3]; // [1,2,3]

function sum(...args) {

  return args.reduce((a, b) => a + b);

}

Ternary Operator

const message = isLoggedIn ? "Welcome" : "Login";

🔹 9. Asynchronous JavaScript

setTimeout, setInterval

setTimeout(() => console.log("Hello"), 1000);

Promises

fetch("/api")

  .then(response => response.json())

  .then(data => console.log(data));

async / await

async function getData() {

  const response = await fetch("/api");

  const data = await response.json();

  console.log(data);

}

🔹 10. Modules and Import/Export

Exporting and Importing

// math.js

export const add = (a, b) => a + b;

// app.js

import { add } from './math.js';

React uses modules heavily (every component is a module).

🔹 11. Classes (Used in Class-based Components)

Class Declaration

class Person {

  constructor(name) {

    this.name = name;

  }

  greet() {

    return `Hello ${this.name}`;

  }

}

🔹 12. Error Handling

try / catch

try {

  const res = await fetch("/api");

} catch (error) {

  console.error(error);

}

🔹 13. Closures & Scope (Advanced but useful in React Hooks)

Closure

function outer() {

  let count = 0;

  return function inner() {

    count++;

    return count;

  };

}

const counter = outer();

counter(); // 1

🔹 14. Truthy/Falsy and Short-Circuiting

const name = "";

console.log(name || "Guest"); // Guest

**1. Array Creation**

js

Copy code

let arr = [1, 2, 3, 4, 5];

let arr2 = new Array(3, 6, 9); // [3, 6, 9]

**🔹 2. Adding/Removing Elements**

**push() – Adds to the end**

js

Copy code

arr.push(6); // [1, 2, 3, 4, 5, 6]

**pop() – Removes from the end**

arr.pop(); // [1, 2, 3, 4, 5]

**unshift() – Adds to the beginning**

js

Copy code

arr.unshift(0); // [0, 1, 2, 3, 4, 5]

**shift() – Removes from the beginning**

arr.shift(); // [1, 2, 3, 4, 5]

**🔹 3. Searching and Testing**

**includes() – Checks if value exists**

arr.includes(3); // true

**indexOf() – Returns index of first match**

arr.indexOf(3); // 2

**lastIndexOf() – Returns last match index**

[1, 2, 3, 2].lastIndexOf(2); // 3

**find() – Returns first element matching condition**

js

Copy code

arr.find(x => x > 3); // 4

**findIndex() – Index of first match by condition**

arr.findIndex(x => x > 3); // 3

**some() – Is any element true for condition?**

arr.some(x => x > 4); // true

**every() – Are all elements true for condition?**

arr.every(x => x > 0); // true

**🔹 4. Looping/Iteration**

**forEach() – Executes function for each element**

arr.forEach(el => console.log(el)); // prints 1 2 3 4 5

**🔹 5. Transforming Arrays**

**map() – Returns new array from function**

let doubled = arr.map(x => x \* 2); // [2, 4, 6, 8, 10]

**filter() – New array with elements matching condition**

js

Copy code

let evens = arr.filter(x => x % 2 === 0); // [2, 4]

**reduce() – Reduces array to a single value**

let sum = arr.reduce((acc, curr) => acc + curr, 0); // 15

**flat() – Flattens nested arrays**

[1, [2, 3], [4, [5]]].flat(); // [1, 2, 3, 4, [5]]

[1, [2, 3], [4, [5]]].flat(2); // [1, 2, 3, 4, 5]

**flatMap() – Map + Flat (1 level deep)**

[1, 2, 3].flatMap(x => [x, x \* 2]); // [1, 2, 2, 4, 3, 6]

**🔹 6. Sorting and Reversing**

**sort() – Sorts in place**

arr.sort((a, b) => b - a); // [5, 4, 3, 2, 1]

**reverse() – Reverses array**

arr.reverse(); // [1, 2, 3, 4, 5]

**🔹 7. Slicing and Splicing**

**slice(start, end) – Returns portion (non-destructive)**

arr.slice(1, 4); // [2, 3, 4]

**splice(start, deleteCount, ...items) – Changes original array**

arr.splice(2, 1, 99); // removes index 2, inserts 99

console.log(arr); // [1, 2, 99, 4, 5]

**🔹 8. Joining and Splitting**

**join() – Converts to string**

arr.join('-'); // "1-2-99-4-5"

**toString() – Similar to join()**

arr.toString(); // "1,2,99,4,5"

**🔹 9. Length Property**

console.log(arr.length); // 5

**🔹 10. Array.isArray() – Checks if variable is array**

Array.isArray(arr); // true

**🔹 11. fill() – Fill array with static values**

[1, 2, 3].fill(0); // [0, 0, 0]

new Array(5).fill(7); // [7, 7, 7, 7, 7]

**🔹 12. copyWithin() – Copies array part within same array**

[1, 2, 3, 4, 5].copyWithin(0, 3); // [4, 5, 3, 4, 5]

**🔹 13. at() – Returns item at specific index (can use negative)**

[10, 20, 30].at(-1); // 30

**🔹 14. from() – Creates array from iterable**

Array.from('abc'); // ['a', 'b', 'c']

**🔹 15. Array.of() – Creates array from arguments**

Array.of(1, 2, 3); // [1, 2, 3]

**✅ Summary Table**

| **Method** | **Description** |
| --- | --- |
| push/pop | Add/remove from end |
| unshift/shift | Add/remove from start |
| map/filter | Transform/filter elements |
| reduce | Combine into single value |
| find/findIndex | Find first match |
| some/every | Boolean test on elements |
| sort/reverse | Sort/reverse array |
| slice/splice | Extract/modify elements |
| flat/flatMap | Flatten nested arrays |
| join/toString | Convert to string |
| from/of | Create arrays |

Top of Form

Bottom of Form