**🌐 JavaScript Fundamentals (Simplified Overview)**

**🧠 1. Keywords and Reserved Words**

Special words already used by JavaScript — you **can’t** use them as variable names.

🪑 **Analogy:** Reserved seats in a theatre — you can’t sit there.

🔹 Examples:  
var, let, const, function, if, else, switch, break, try, catch, return, new, this, class, import, export

🚫 Example:

let if = 10; // ❌ Error: 'if' is reserved

**🔢 2. Data Types**

JavaScript is **dynamically typed** — type decided at runtime.

**🧩 Primitive (Single value)**

| **Type** | **Example** | **Description** |
| --- | --- | --- |
| Number | 42 | Integers, decimals |
| String | "Hello" | Text |
| Boolean | true | True/false |
| Undefined | let x; | Declared but no value |
| Null | null | Empty value |
| Symbol | Symbol("id") | Unique value |
| BigInt | 1234567890n | Very large numbers |

🧠 **Analogy:** Each is like a sticky note — holds one small value.

**🧱 Non-Primitive (Reference types)**

| **Type** | **Example** | **Description** |
| --- | --- | --- |
| Object | {name:"John"} | Key-value pairs |
| Array | [1,2,3] | Ordered list |
| Function | function() {} | Reusable code |

🧠 **Analogy:** Objects = containers holding many values.

**📦 3. Variables (var, let, const)**

Used to store data.

| **Keyword** | **Scope** | **Reassign?** | **Redeclare?** | **Hoisted?** |
| --- | --- | --- | --- | --- |
| var | Function | ✅ | ✅ | ✅ |
| let | Block | ✅ | ❌ | ❌ |
| const | Block | ❌ | ❌ | ❌ |

🧠 **Analogy:**  
var → old leaky bag  
let → secure container  
const → sealed box

Example:

let name = "Alice";

const PI = 3.14;

**➕ 4. Operators**

Used to perform actions.

| **Type** | **Example** | **Meaning** |
| --- | --- | --- |
| Arithmetic | + - \* / % | Math |
| Comparison | == === != > < | Compare values |
| Logical | `&& |  |
| Assignment | = += -= | Assign values |
| Ternary | ? : | Short if-else |

Example:

let a = 10, b = 5;

console.log(a > b ? "A bigger" : "B bigger");

**🔁 5. Flow Control**

**if / else**

let score = 85;

if (score > 90) console.log("A");

else if (score > 75) console.log("B");

else console.log("C");

**switch**

let day = "Mon";

switch(day) {

case "Mon": console.log("Start"); break;

case "Fri": console.log("Weekend!"); break;

default: console.log("Midweek");

}

**Loops**

for (let i = 1; i <= 3; i++) console.log(i); // for loop

let j = 1;

while (j <= 3) console.log(j++); // while loop

["apple","banana"].forEach(f => console.log(f)); // forEach

🧠 **Analogy:** Loops are like repeating machines — “Do this again and again.”

**⚙️ 6. Functions**

Reusable blocks of code.

**Normal function**

function greet(name) {

return `Hello ${name}`;

}

**Arrow function**

const greet = name => `Hello ${name}`;

🧠 **Analogy:** Like a coffee machine — give input (water), get output (coffee).

**🌟 7. Best Practices**

✅ Use let & const, avoid var  
✅ Use === (strict compare)  
✅ Use meaningful names  
✅ Keep functions short  
✅ Avoid global variables  
✅ Use template literals — Hello ${name}

**🧩 8. Objects**

Group related data & behavior.

const car = {

brand: "Tesla",

color: "red",

start() {

console.log("Car started");

}

};

car.start(); // Car started

🧠 **Analogy:** A car has properties (color) and actions (start).

**🧮 9. Arrays and Methods**

Store lists of data.

let fruits = ["apple", "banana"];

fruits.push("cherry"); // Add

console.log(fruits[0]); // apple

🔧 **Common Methods:**  
push, pop, shift, map, filter, reduce, forEach

Example:

let nums = [1,2,3];

console.log(nums.map(n => n\*2)); // [2,4,6]

🧠 **Analogy:** Array = train, each element = coach.

**🧬 10. Prototype & Inheritance**

Objects can inherit properties from other objects.

class Animal {

speak() { console.log("Sound"); }

}

class Dog extends Animal {

bark() { console.log("Woof"); }

}

new Dog().bark(); // Woof

🧠 **Analogy:** Children inherit traits from parents.

**🔹 11. Scope**

Defines where variables are accessible.

| **Type** | **Example** |
| --- | --- |
| Global | declared outside any function |
| Local | inside a function |
| Block | inside {} |

🧠 **Analogy:**  
Global = world 🌍  
Local = country 🇮🇳  
Block = city 🏙️

**👤 13. this Keyword**

Refers to the current object.

const person = {

name: "Alice",

greet() { console.log(this.name); }

};

person.greet(); // Alice

🧠 **Analogy:** “this” = “myself” — depends on who’s speaking.

**⚡ 14. Asynchronous JS**

Lets JS handle time-consuming tasks without blocking.

**Callback**

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

**Promise**

new Promise(res => setTimeout(() => res("OK"), 1000))

.then(console.log);

**Async/Await**

async function load() {

const data = await Promise.resolve("Loaded");

console.log(data);

}

load();

🧠 **Analogy:** Chef cooks (main thread), waiter handles async tasks.

**🚨 15. Error Handling**

try {

throw new Error("Oops!");

} catch (e) {

console.error(e.message);

} finally {

console.log("Done!");

}

🧠 **Analogy:**  
Try → Do it  
Catch → Handle if fails  
Finally → Clean up

✅ Best Practices

* Use try/catch
* Handle async errors with .catch()
* Create custom error types if needed
* Always log meaningful messages

**🎯 Everything in Action**

const students = ["Alice", "Bob", "Charlie"];

for (let s of students) {

let score = Math.floor(Math.random() \* 100);

let grade = score > 90 ? "A" : score > 75 ? "B" : "C";

console.log(`${s} scored ${score} → Grade: ${grade}`);

}

🧠 **Analogy:** Teacher (function) grading students (data) — looping, checking, printing results.

Got it 👍 — here’s a **clear and concise overview** (not in depth) of the topics:

**🧱 JavaScript Classes and Inheritance — Overview**

**🔹 Introduction to Classes**

* Classes are **blueprints** for creating objects.
* Introduced in **ES6** for easier OOP syntax (uses prototypes behind the scenes).

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

console.log(`Hello, I’m ${this.name}`);

}

}

**🔹 Constructors and Methods**

* constructor() → special method for initializing object properties.
* Methods → define object behaviors.

class Car {

constructor(brand) {

this.brand = brand;

}

drive() {

console.log(`${this.brand} is driving`);

}

}

**🔹 Inheritance and extends Keyword**

* Allows a class to **inherit** from another.

class Animal {

speak() {

console.log("Animal sound");

}

}

class Dog extends Animal {

bark() {

console.log("Woof!");

}

}

**🔹 Superclass and Subclass**

* **Superclass** → parent class.
* **Subclass** → child class that inherits from parent.
* Use super() to call parent constructor or methods.

class Animal {

constructor(name) {

this.name = name;

}

}

class Dog extends Animal {

constructor(name, breed) {

super(name); // call parent constructor

this.breed = breed;

}

}

**🔹 Getters and Setters**

* Used to **get or set** property values safely.

class User {

constructor(name) {

this.\_name = name;

}

get name() {

return this.\_name.toUpperCase();

}

set name(value) {

this.\_name = value.trim();

}

}

**🔹 Best Practices**

✅ Use meaningful class and method names.  
✅ Keep methods small and focused.  
✅ Use inheritance only when needed.  
✅ Prefer composition (has-a) over inheritance (is-a).  
✅ Use getters/setters for encapsulation.

**✳️ Regular Expressions in JavaScript — Overview**

**🔹 Introduction to Regular Expressions**

* **RegEx** are patterns used to **match, search, and manipulate strings**.
* Declared using /pattern/ or new RegExp("pattern").

**🔹 Basic Syntax**

| **Symbol** | **Meaning** | **Example** |
| --- | --- | --- |
| . | Any character | /a.b/ matches acb, arb |
| \* | 0 or more times | /go\*d/ → gd, good |
| + | 1 or more times | /go+d/ → good |
| ? | Optional | /colou?r/ → color, colour |
| ^ | Start of string | /^Hi/ → matches “Hi John” |
| $ | End of string | /end$/ |
| [] | Character set | /[abc]/ → a, b, or c |
| {n,m} | Range of repeats | /\d{2,4}/ → 2–4 digits |
| \d | Digit | /\d+/ |
| \w | Word char | /\w+/ |
| \s | Whitespace | /\s/ |

**🔹 Using RegExp Objects**

let pattern = /hello/i;

console.log(pattern.test("Hello World")); // true

console.log("Hello World".match(pattern)); // ["Hello"]

**🔹 Validating Data Example**

let emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

console.log(emailPattern.test("test@mail.com")); // true

console.log(emailPattern.test("invalid@com")); // false

**🌟 ES6 & TypeScript Fundamentals (Simplified)**

**📝 1. Var, Let, Const**

| **Keyword** | **Scope** | **Reassignable** | **Redeclarable** | **Hoisted?** |
| --- | --- | --- | --- | --- |
| var | Function | ✅ | ✅ | ✅ |
| let | Block | ✅ | ❌ | ❌ |
| const | Block | ❌ | ❌ | ❌ |

**Example:**

var x = 10; // old, function-scoped

let y = 20; // block-scoped

const PI = 3.14; // cannot change

🧠 **Analogy:**

* var → leaky bag
* let → secure container
* const → sealed jar

**⚡ 2. Arrow Functions & Default Arguments**

**Arrow Function**

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

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

**Default Arguments**

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

console.log(greet()); // Hello Guest

🧠 **Analogy:** Arrow = compact function, default arg = backup value.

**✨ 3. Template Strings & String Methods**

**Template Strings**

let name = "Alice";

console.log(`Hi ${name}, welcome!`); // Hi Alice, welcome!

**String Methods**

let str = "Hello World";

console.log(str.toUpperCase()); // HELLO WORLD

console.log(str.includes("World")); // true

console.log(str.split(" ")); // ["Hello", "World"]

**🏗️ 4. Object De-structuring**

const person = { name: "Alice", age: 25 };

const { name, age } = person;

console.log(name, age); // Alice 25

// Array destructuring

const nums = [1,2,3];

const [first, second] = nums;

console.log(first, second); // 1 2

🧠 **Analogy:** Take only the stuff you need from a box.

**🔄 5. Spread & Rest Operators**

**Spread: expand**

const arr1 = [1,2];

const arr2 = [...arr1,3,4];

console.log(arr2); // [1,2,3,4]

**Rest: collect**

function sum(...numbers) {

return numbers.reduce((a,b)=>a+b,0);

}

console.log(sum(1,2,3)); // 6

🧠 **Analogy:** Spread = scatter, Rest = gather.

**🟦 TypeScript Fundamentals**

**🔹 1. Types & Type Assertions**

let age: number = 25;

let name: string = "Alice";

let isActive: boolean = true;

// Type assertion

let someValue: any = "Hello";

let strLength: number = (someValue as string).length;

**🔹 2. Custom Object & Function Types**

// Object type

type Person = { name: string, age: number };

let p1: Person = { name: "Bob", age: 30 };

// Function type

let add: (x:number, y:number) => number;

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

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

**🏗️ 3. TypeScript OOP – Classes, Interfaces, Constructor**

**Class & Constructor**

class Animal {

name: string;

constructor(name: string) {

this.name = name;

}

speak() {

console.log(`${this.name} makes a sound`);

}

}

const dog = new Animal("Rex");

dog.speak(); // Rex makes a sound

**Interface**

interface Person {

name: string;

age: number;

greet(): void;

}

const user: Person = {

name: "Alice",

age: 25,

greet() { console.log("Hi!"); }

};

user.greet(); // Hi!

**Inheritance**

class Dog extends Animal {

bark() {

console.log(`${this.name} barks`);

}

}

const myDog = new Dog("Buddy");

myDog.speak(); // Buddy makes a sound

myDog.bark(); // Buddy barks

🧠 **Analogy:**

* Class = blueprint
* Constructor = builder
* Interface = contract

This **covers all ES6 + TypeScript essentials** in **short, clear lines** — perfect for quick reference or interviews.