**JS Core Concepts**

**1. Execution Context & Scope**

* Every time JavaScript runs code, it creates an **Execution Context** — a kind of environment that decides how and where variables and functions are stored and accessed.
* **Global scope**: Variables available everywhere.
* **Function scope**: Variables available only inside that function.
* **Block scope**: Variables declared inside {} like (if, for) are available only there.

**2. Hoisting & Variable Declarations**

* **Hoisting** means JavaScript moves variable and function declarations to the top of their scope before executing code.
* **var** is hoisted and initialized as undefined.
* **let and const** are hoisted too but *not initialized* (so using them before declaration gives an error).

**3. Closures**

* A **closure** happens when an inner function “remembers” the variables from its outer function, even after the outer function has finished running.

**4. this Keyword & Binding**

* **This** keyword refers to the object that is currently executing the code.
* Its value depends on how the function is called:
  + In a regular function, this refers to the global object or undefined in strict mode.
  + In an object method, this refers to that object.
  + In an arrow function, this is inherited from where it was defined (not how it’s called).

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**5. Prototype & Inheritance**

* Every JavaScript object has a hidden **prototype**, which is another object it inherits properties and methods from.
* This is called **prototypal inheritance**.

**6. Promises & Async/Await**

* **Promises** represent a value that will be available *in the future* (like a pending task).
* **Async/Await** makes working with promises easier by letting code look synchronous (step-by-step).

**7. Event Loop & Task Queue**

* JavaScript is **single-threaded**, meaning it does one thing at a time.
* The **Event Loop** manages asynchronous tasks — when something (like a timeout or API call) finishes, it goes into the **task queue**, and the event loop checks when to execute it.

**8. Fetch API**

* Both are used to make **HTTP requests** to get or send data to a server without reloading the page.

**9. Destructuring & Spread/Rest Operators**

* **Destructuring** allows you to unpack values from arrays or objects into separate variables.
* **Spread (...)** copies or expands elements.
* **Rest (...)** collects remaining items into an array.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**10. Modules (import/export)**

* **Modules** let you split your code into multiple files and reuse them easily.
* You can **export** a function or variable from one file and **import** it into another.

**11. Arrow Functions & Default Parameters**

* **Arrow functions** are a shorter way to write functions. They don’t have their own this.
* **Default parameters** allow you to assign a default value to a function parameter if no value is provided.

**12. Optional Chaining & Nullish Coalescing**

* **Optional chaining (?.)** safely checks if a property exists before accessing it — prevents errors.
* **Nullish coalescing (??)** provides a default value only if something is null or undefined.

**13. Event Delegation & DOM Manipulation**

* **DOM Manipulation** means changing HTML elements dynamically using JavaScript.
* **Event delegation** means instead of adding event listeners to many elements, you add it to a common parent and detect which child triggered the event.

**14. Functional Programming Patterns**

* A style of coding that focuses on **pure functions** (no side effects), **immutability** (don’t change original data), and **using methods like map, filter, reduce** to transform data.

**15. Error Handling**

* Managing mistakes or unexpected behaviour in code using **try/catch** blocks.
* **Custom errors** can be created for specific situations.
* In **async code**, errors can be handled using. catch() or try/catch inside async functions.