Core JavaScript .

1. let and const (Block Scope)

`let` and `const` are used to declare variables with block-level scope.

- A variable declared with `let` can be reassigned later.
- A variable declared with `const` cannot be reassigned after initialization.
- Both exist only inside the block where they are declared.

Written example:
If a variable is declared inside an `if` block using `let`, it cannot be accessed outside that `if` block. If a variable is declared using `const` for storing a user object, the object’s properties can change, but the variable itself cannot point to a new object.

2. Data Types (Primitive & Reference)

JavaScript data types are categorized based on memory behavior.

Primitive Types

Primitive values are stored directly and copied independently.

Written example:
If one variable stores the number 10 and another variable copies it, changing the second variable does not affect the first.

Reference Types

Reference values store memory addresses, not actual data.

Written example:
If two variables refer to the same object and one variable updates a property, the change is visible through the other variable as well.

3. Truthy & Falsy Values

JavaScript automatically converts values into true or false when used in conditions.

- Falsy values represent absence or emptiness.
- All other values are treated as truthy.

Written example:
An empty string in a condition behaves like false, but a string containing `"0"` behaves like true.
An empty array still behaves like true in a conditional statement.

4. Operators (===, !==, &&, ||, ?:)

1. Strict Equality Operator (===)

The strict equality operator checks **two things at the same time**:

1. The value
2. The data type

If either one is different, the result is false.

It does **not perform type conversion**.

Written examples:

- Comparing the number five and the string five results in false because the types are different.
- Comparing two numbers with the same value results in true.
- Comparing two different objects always results in false, even if they look identical, because objects are compared by reference.

Why this matters:
This operator prevents hidden bugs caused by automatic type conversion.

2. Strict Inequality Operator (!==)

The strict inequality operator is the opposite of strict equality.

It returns true when:

- Values are different, or
- Data types are different

No type conversion is performed.

Written examples:

- A number and a string with the same visible value are considered not equal.
- Two values of different types always result in true.

Why this matters:
This operator makes inequality checks explicit and predictable.

3. Logical AND Operator (&&)

The logical AND operator checks **multiple conditions**.

It returns true **only if all conditions are true**.

If the first condition is false, the remaining conditions are not checked (short-circuit behavior).

Written examples:

- A user is allowed access only if they are logged in **and** have admin rights.
- If the first condition fails, the second condition is ignored.

Why this matters:
Understanding short-circuiting helps prevent unnecessary operations and bugs.

4. Logical OR Operator (||)

The logical OR operator checks **alternative conditions**.

It returns true if **at least one condition is true**.

If the first condition is true, the remaining conditions are skipped.

Written examples:

- A default username is used if the entered name is empty.
- Access is granted if the user is an admin **or** a manager.

Why this matters:
OR is often used for fallback values and default logic.

5. Ternary Operator (? :)

The ternary operator is a **compact conditional expression**.

It evaluates a condition and returns:

- One value if the condition is true
- Another value if the condition is false

It replaces simple if–else statements.

Written examples:

- If a user’s age is 18 or above, return “Adult”; otherwise return “Minor”.
- Display “Logged In” or “Guest” based on authentication status.

Why this matters:
Ternary improves readability only when used for **simple decisions**.
Nested ternary operators are a sign of poor judgment.

5. Conditional Statements (if, ternary)

Conditional statements control execution flow based on conditions.

Written example:
If a user’s balance is greater than zero, allow the transaction; otherwise, block it.
A ternary condition can be used to display “Login Successful” or “Login Failed”.

Functions

6. Function Declaration & Expression

Functions define reusable logic.

- Function declarations are available before execution starts.
- Function expressions behave like normal variables.

Written example:
A declared function can be called even before it appears in the file.
An expressed function cannot be used before assignment.

7. Arrow Functions

Arrow functions provide concise syntax and do not create their own `this`.

Written example:
In an object method, using an arrow function causes `this` to refer to the outer scope instead of the object itself.

8. Default Parameters

Default parameters ensure functions behave correctly when arguments are missing.

Written example:
If a greeting function is called without a name, it automatically uses “Guest”.

9. Return Values

Functions may return a value or return nothing.

Written example:
A function that calculates total price returns the final amount.
A function that only logs a message returns nothing, resulting in undefined.

Objects & Arrays

10. Object Creation & Access

Objects store structured data as key–value pairs.

Written example:
A user object stores name, email, and age.
Dot notation is used when property names are fixed, while bracket notation is used when property names are dynamic.

11. Array Basics

Arrays store ordered data collections.

Written example:
A list of student names is stored in an array where each name has a fixed position index.

12. Array Methods

- `map()` transforms every element.
- `filter()` removes unwanted elements.
- `find()` returns the first matching element.
- `reduce()` combines values into one.
- `some()` checks if at least one element matches.
- `every()` checks if all elements match.

Written example:
Mapping can convert prices from dollars to rupees.
Filtering can remove inactive users.
Finding can locate a user by ID.
Reducing can calculate total cart value.
Some can check if any product is out of stock.
Every can verify if all students passed.

Modern JavaScript (ES6+)

13. Destructuring

Destructuring extracts values from objects or arrays.

Written example:
Instead of accessing user.name and user.age separately, both values are extracted at once.

14. Spread Operator

Spread operator copies or merges data.

Written example:
A new array is created by copying an existing array and adding extra elements without changing the original.

15. Rest Operator

Rest operator gathers multiple values into a single variable.

Written example:
A function that calculates total price accepts any number of item prices as input.

16. Template Literals

Template literals allow embedded variables in strings.

Written example:
A welcome message dynamically displays the user’s name inside the message.

Modules & Classes

17. import / export

Modules split code into reusable files.

Written example:
A utility file exports validation logic, which is imported into a login file.

18. ES6 Classes

Classes act as blueprints for objects.

Written example:
A `Car` class defines properties like brand and speed, which are shared by all car objects.

19. Constructor

Constructors initialize object data.

Written example:
When a new user object is created, the constructor assigns username and email immediately.

20. extends & super

Used for inheritance.

Written example:
A `Student` class inherits properties from a `Person` class and adds roll number.

21. this Keyword

`this` refers to the current execution context.

Written example:
Inside an object method, `this` refers to that object’s properties.

Asynchronous JavaScript

22. Callbacks

Callbacks handle operations that finish later.

Written example:
After fetching data from a server, a callback function runs to process the response.

23. Promises

Promises represent future results.

Written example:
A promise resolves when data is successfully loaded or rejects if an error occurs.

24. async / await

Async/await simplifies promise handling.

Written example:
An async function waits for server data before continuing execution.

25. try...catch

Handles runtime errors gracefully.

Written example:
If invalid JSON is received, the error is caught and handled without crashing the app.

Advanced JavaScript Concepts

26. Scope

Scope defines variable accessibility.

Written example:
A variable declared inside a function cannot be accessed outside that function.

27. Closures

Closures allow functions to remember outer variables.

Written example:
A counter function remembers its count value even after execution finishes.

28. Immutability

Immutability avoids modifying original data.

Written example:
Instead of changing a user’s age directly, a new user object is created with the updated age.

29. Higher-Order Functions

Functions that work with other functions.

Written example:
A function that accepts another function to process array data.

Browser & Runtime

30. DOM Basics

DOM represents HTML as objects.

Written example:
A button element is accessed and updated using JavaScript.

31. Event Handling

Events respond to user actions.

Written example:
Clicking a button triggers a function to submit a form.

32. Event Bubbling

Events propagate upward in the DOM tree.

Written example:
Clicking a list item also triggers the parent container’s click event.

33. LocalStorage & SessionStorage

Browser storage mechanisms.

Written example:
LocalStorage saves theme preference permanently.
SessionStorage saves login state until the tab is closed.

34. JSON (parse, stringify)

JSON is used for data exchange.

Written example:
Server sends user data as text, which is converted into an object for use in the application.