

JavaScript Notes :

1. let and const (Block Scope)

let and const are used to declare variables in JavaScript. They follow block scope, which means they work only inside the { } where they are created.

- let → value can be changed
- const → value cannot be reassigned

Example:

```
{  
  let a = 10;  
  const b = 20;  
  console.log(a, b);  
}  
  
// a and b are not accessible here
```

2. Data Types

Data types decide what kind of value a variable can store.

Primitive Data Types

They store single values.

- Number
- String
- Boolean
- Undefined
- Null

Example:

```
let age = 22;  
let city = "Pune";
```

Reference Data Types

They store multiple values.

- Object
- Array
- Function

Example:

```
let user = { name: "Siddhi", age: 22 };
```

3. Truthy & Falsy Values

In JavaScript, some values act like true and some act like false in conditions.

Falsy values:

false, 0, "", null, undefined, NaN

Example:

```
if(0) {  
    console.log("True");  
} else {  
    console.log("False");  
}
```

4. Operators

Operators are symbols used to perform operations.

- === checks value and type
- !== checks not equal
- && logical AND
- || logical OR
- ?: ternary operator

Example:

```
let age = 20;  
  
let result = age >= 18 ? "Adult" : "Minor";
```

5. Conditional Statements

Conditional statements help the program take decisions.

Example:

```
if(age > 18) {  
    console.log("Eligible for voting");  
} else {  
    console.log("Not eligible");  
}
```

6. Function Declaration & Expression

Functions are reusable blocks of code.

Function Declaration

```
function add(a, b) {  
    return a + b;  
}
```

Function Expression

```
const add = function(a, b) {  
    return a + b;  
};
```

7. Arrow Functions

Arrow functions are a shorter way to write functions.

Example:

```
const square = n => n * n;
```

8. Default Parameters

Default parameters are used when no value is passed.

Example:

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

```
greet();
```

9. Return Values

return sends the result back from a function.

Example:

```
function sum(a, b) {  
    return a + b;  
}
```

10. Object Creation & Access

Objects store data in key-value pairs.

Example:

```
let student = {  
    name: "Siddhi",  
    marks: 85  
};  
  
console.log(student.name);
```

11. Array Basics

Arrays store multiple values in one variable.

Example:

```
let fruits = ["Apple", "Banana", "Mango"];
```

12. Array Methods

Array methods help work with arrays easily.

map()

```
let nums = [1, 2, 3];  
let doubled = nums.map(n => n * 2);
```

filter()

```
let result = nums.filter(n => n > 1);
```

find()

```
nums.find(n => n === 2);
```

reduce()

```
nums.reduce((a, b) => a + b);
```

some() / every()

```
nums.some(n => n > 2);  
nums.every(n => n > 0);
```

13. Destructuring

Destructuring extracts values from objects or arrays.

Example:

```
let user = { name: "Siddhi", age: 22 };
```

```
let { name, age } = user;
```

14. Spread Operator (...)

Spread operator copies values.

Example:

```
let arr1 = [1, 2];
```

```
let arr2 = [...arr1, 3];
```

15. Rest Operator (...)

Rest operator collects values into an array.

Example:

```
function total(...nums) {  
  return nums.length;  
}
```

16. Template Literals

Template literals are used for dynamic strings.

Example:

```
let name = "Siddhi";  
console.log(`Hello ${name}`);
```

17. import / export

Used to share code between files.

Example:

```
export function add() {}  
import { add } from './math.js';
```

18. ES6 Classes

Classes are blueprints for creating objects.

Example:

```
class User {  
  constructor(name) {  
    this.name = name;  
  }  
}
```

19. Constructor

Constructor runs automatically when an object is created.

Example:

```
let u1 = new User("Siddhi");
```

20. extends & super

Used to inherit properties from another class.

Example:

```
class Admin extends User {  
  constructor(name) {  
    super(name);  
  }  
}
```

21. this Keyword

this refers to the current object.

Example:

```
console.log(this.name);
```

22. Callbacks

Callbacks are functions passed to another function.

Example:

```
setTimeout(() => {
```

```
console.log("Done");
}, 1000);
```

23. Promises

Promises handle asynchronous tasks.

Example:

```
let p = new Promise((resolve, reject) => {
  resolve("Success");
});
```

24. async / await

Makes async code easier to read.

Example:

```
async function fetchData() {
  let res = await fetch(url);
}
```

25. try...catch

Used to handle errors.

Example:

```
try {
  error();
} catch (e) {
  console.log(e);
}
```

26. Scope

Scope decides where variables can be used.

Example:

```
let x = 10; // global
```

27. Closures

Closures allow a function to remember outer variables.

Example:

```
function outer() {  
  let x = 10;  
  return function inner() {  
    console.log(x);  
  };  
}
```

28. Immutability

Immutability means not changing original data.

Example:

```
let newUser = { ...user, age: 23 };
```

29. Higher-Order Functions

Functions that take other functions as input.

Example:

```
nums.map(n => n * 2);
```

30. DOM Basics

DOM allows JavaScript to interact with HTML.

Example:

```
document.getElementById("title");
```

31. Event Handling

Event handling responds to user actions.

Example:

```
button.addEventListener("click", show);
```

32. Event Bubbling

Event moves from child to parent element.

Example:
Click on button → div → body

33. LocalStorage & SessionStorage

Used to store data in browser.

Example:

```
localStorage.setItem("name", "Siddhi");
```

34. JSON

JSON is used to send and receive data.

Example:

```
JSON.stringify(obj);
```

```
JSON.parse(data);
```