**JS**

**DEVELOPER**

**NOTES**

**👨🏻‍💻 Name: Mr. Rupesh Lal**

**🌐 Website: www.itmodem.com**

**📞 WhatsApp: +91 7870886355**

**💼 Profession: Software Engineer**

**📘 JavaScript Notes Outline – *It Modem Solution***

**Chapter 0: Introduction to Programming**

* **What is Programming?**
* **Why Learn Programming?**
* **What is JavaScript?**
* **Basic Programming Concepts**
  + **Statements**
  + **Variables**
  + **Data Types**
  + **Functions**
* **Tools Required**
* **Hello World Program**
* **How JavaScript Code Runs**
* **Summary Table**

**Chapter 1: Variables and Data Types**

* **Declaring Variables (var, let, const)**
* **Variable Scope**
* **Primitive Data Types**
  + **Number, String, Boolean, Undefined, Null, Symbol, BigInt**
* **Non-Primitive Data Types**
  + **Object, Array, Function**
* **typeof Operator**
* **Constants**
* **Summary Table**

**Chapter 2: Expressions and Conditions**

* **Expressions**
  + **Arithmetic, String, Logical, Comparison**
* **Conditional Statements**
  + **if**
  + **if-else**
  + **else-if Ladder**
  + **Ternary Operator**
  + **switch**
* **Logical Operators Summary**
* **Comparison Operators Summary**

**Chapter 3: Loops and Functions**

* **Loops**
  + **for, while, do-while**
  + **for…of, for…in**
  + **break, continue**
* **Functions**
  + **Declaration, Expression, Arrow Function**
  + **Default Parameters**
  + **Return Values**
  + **Scope, Nested Functions**
* **Summary Table**

**Chapter 4: Strings in JavaScript**

* **What is a String?**
* **String Length and Indexing**
* **Looping Through Strings**
* **String Immutability**
* **Common String Methods**
* **Template Literals**
* **Escape Characters**
* **String Comparisons**
* **String Repeat**
* **Summary Table**

**Chapter 5: Arrays in JavaScript**

* **What is an Array?**
* **Accessing & Modifying Elements**
* **Array Length**
* **Looping Through Arrays**
  + **for, for…of, forEach**
* **Array Methods**
  + **push, pop, shift, unshift**
  + **indexOf, includes, splice, slice**
  + **concat, join, sort, reverse**
* **Multidimensional Arrays**
* **Summary Table**

**Chapter 6: JavaScript in the Browser**

* **JavaScript Engine Overview**
* **Developer Tools**
* **<script> Tag Usage**
* **console Object Methods**
  + **log, warn, error, table, etc.**
* **Browser Interaction Methods**
  + **alert, prompt, confirm**
* **Window Object & BOM**
* **DOM Introduction**

**Chapter 7: Walking the DOM**

* **DOM Overview & Node Types**
  + **Text, Element, Comment**
* **Auto-Correction in DOM**
* **Traversing the DOM**
* **Working with Children**
* **DOM Collections**
* **Element-Only Navigation**
* **Working with Tables**
* **Searching the DOM**
* **Methods: matches, closest, contains**

**Chapter 8: Events and Other DOM Properties**

* **console.dir(), tagName, nodeName**
* **innerHTML, outerHTML, textContent**
* **Hidden Elements**
* **Attributes & data-\***
* **Inserting & Removing Elements**
* **className and classList**
* **setTimeout() and setInterval()**
* **Event Handling**
  + **Inline, addEventListener, removeEventListener**
  + **event object**

**Chapter 9: Callbacks, Promises, and Async/Await**

* **Asynchronous Nature**
* **Callbacks**
  + **Error handling, Callback Hell**
* **Promises**
  + **Creating, Consuming, Chaining**
  + **Promise.all, race, allSettled, any**
* **async/await with Error Handling**
* **Custom Errors and finally**

**Chapter 10: Object-Oriented Programming (OOP)**

* **OOP Concepts**
* **Prototype Inheritance**
* **Classes and Objects**
* **Constructor Method**
* **Inheritance and extends**
* **Method Overriding and super**
* **Static Methods**
* **Getters and Setters**
* **instanceof Operator**

**Chapter 11: Advanced JavaScript**

* **IIFE (Immediately Invoked Function Expressions)**
* **Destructuring (Array & Object)**
* **Spread Syntax**
* **Hoisting (var, let, const)**
* **Scope Types**
  + **Global, Function, Block**
* **Summary Table**

**Chapter 12: JavaScript Cheatsheet**

* **Variables and Data Types**
* **Expressions and Conditions**
* **Arrays and Strings**
* **Loops and Functions**
* **JavaScript in the Browser**
* **DOM Traversal**
* **Event Handling**
* **Callbacks, Promises, Async/Await**
* **OOP Overview**
* **Advanced JS (IIFE, Destructuring, Spread, Hoisting)**

**Chapter 13: Final Summary**

* **Summary of all key concepts from each chapter**
* **Quick syntax and explanation examples**

**📘 Chapter 0 – Introduction to Programming**

**🔹 What is Programming?**

Programming is the process of writing instructions that a computer can execute to perform specific tasks. These instructions are written using **programming languages**.

**🔹 Why Learn Programming?**

* Automate repetitive tasks
* Create websites, apps, games
* Develop problem-solving skills
* High demand career skill

**🔹 What is JavaScript?**

JavaScript is a high-level, interpreted programming language used to create dynamic and interactive web content.

*// This is a simple JavaScript program*

console.log("Hello, world!"); *// Prints message to the browser console*

**🔹 Basic Programming Concepts**

**1. Statements**

A statement is a single line of code that performs an action.

let name = "Alice"; *// Variable declaration statement*

console.log(name); *// Output statement*

**2. Variables**

Used to store data.

let age = 20; *// 'let' creates a variable called age*

**3. Data Types**

Different kinds of data that can be stored in variables.

let number = 10; *// Number*

let message = "Hi"; *// String*

let isLoggedIn = true; *// Boolean*

**4. Functions**

Blocks of reusable code.

function greet() {

console.log("Welcome to JavaScript!"); *// This message is printed when the function is called*

}

greet(); *// Call the function*

**🔹 Tools Required**

* A browser (Chrome, Firefox, etc.)
* A text editor (VS Code, Sublime Text, etc.)
* Knowledge of HTML & CSS (for web development)

**🔹 Hello World Program**

console.log("Hello, world!"); *// Classic first program in any language*

**🔹 How Code Runs in JavaScript**

1. Code is written in a file (.js) or inside the <script> tag in HTML.
2. The JavaScript engine in the browser reads and executes it.

*<!-- Example of embedding JavaScript in HTML -->*

<!DOCTYPE html>

<html>

<head><title>My First JS Page</title></head>

<body>

<script>

alert("Welcome to JavaScript!"); *// Pops up a message on page load*

</script>

</body>

</html>

**🔹 Summary**

| **Concept** | **Description** |
| --- | --- |
| Programming | Writing code to solve problems |
| JavaScript | Scripting language for the web |
| Statement | Line of code that performs an action |
| Variable | Stores data |
| Function | Block of code that does a task |

🎯 Start experimenting by writing small scripts. Build logic step by step. The key is **practice**!

**📘 Chapter 1 – Variables and Data Types in JavaScript**

**🔹 What are Variables?**

Variables are containers for storing data values.

**1. Declaring Variables**

var x = 5; *// 'var' is the old way of declaring variables*

let y = 10; *// 'let' is the modern and recommended way*

const z = 15; *// 'const' declares a constant (value cannot change)*

**🚨 Variable Scope:**

* **var**: Function-scoped
* **let / const**: Block-scoped

**🔹 Data Types in JavaScript**

JavaScript has dynamic types, meaning a variable can hold any type of data.

**1. Primitive Data Types**

**✅ Number**

let age = 25; *// Integer*

let price = 99.99; *// Floating point number*

**✅ String**

let name = "Alice"; *// String with double quotes*

let greeting = 'Hello'; *// String with single quotes*

**✅ Boolean**

let isOnline = true;

let hasAccount = false;

**✅ Undefined**

let score;

console.log(score); *// undefined since no value is assigned*

**✅ Null**

let discount = null; *// Intentional absence of any value*

**✅ Symbol (ES6+)**

const id = Symbol("userID");

**✅ BigInt (For very large numbers)**

const bigNumber = 1234567890123456789012345678901234567890n;

**2. Non-Primitive Data Types**

**✅ Object**

let person = {

name: "John",

age: 30

}; *// Object with key-value pairs*

**✅ Array**

let colors = ["red", "green", "blue"];

**✅ Function**

function greet() {

console.log("Hello, world!");

}

**🔹 typeof Operator**

Used to check the type of a variable.

console.log(typeof 123); *// number*

console.log(typeof "abc"); *// string*

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

console.log(typeof null); *// object (this is a known quirk)*

**🔹 Constants with const**

* Cannot be reassigned
* Must be initialized during declaration

const PI = 3.14;

**🧠 Summary Table**

| **Data Type** | **Example Value** | **Description** |
| --- | --- | --- |
| Number | 100, 12.5 | All numeric values |
| String | “hello” | Sequence of characters |
| Boolean | true / false | Logical values |
| Undefined | undefined | Declared but not assigned |
| Null | null | No value |
| Object | { key: value } | Collection of key-value pairs |
| Array | [1, 2, 3] | Ordered list |
| Function | function() {} | Block of code |
| Symbol | Symbol(“desc”) | Unique identifier (ES6) |
| BigInt | 123n | Large integers |

**📘 Chapter 2 – Expressions and Conditions in JavaScript**

**✨ Expressions**

An **expression** is a piece of code that produces a value.

**1. Arithmetic Expressions**

let x = 5 + 2; *// 5 + 2 = 7*

let y = x \* 3; *// 7 \* 3 = 21*

console.log(y); *// Output: 21*

**2. String Expressions**

let greeting = "Hello" + " " + "World";

console.log(greeting); *// Output: Hello World*

**3. Logical Expressions**

let a = true && false; *// false*

let b = true || false; *// true*

console.log(a, b); *// Output: false true*

**4. Comparison Expressions**

console.log(5 > 3); *// true*

console.log(5 == "5"); *// true (loose equality)*

console.log(5 === "5"); *// false (strict equality)*

**❓ Conditions**

Conditions are used to make decisions in code.

**1. if Statement**

let age = 18;

if (age >= 18) {

console.log("You are eligible to vote.");

}

**2. if-else Statement**

let marks = 40;

if (marks >= 50) {

console.log("Pass");

} else {

console.log("Fail");

}

**3. else-if Ladder**

let score = 75;

if (score >= 90) {

console.log("Grade: A");

} else if (score >= 75) {

console.log("Grade: B");

} else if (score >= 60) {

console.log("Grade: C");

} else {

console.log("Grade: F");

}

**4. Ternary Operator (Shorthand for if-else)**

let result = (marks >= 50) ? "Pass" : "Fail";

console.log(result); *// Output: Fail*

**5. switch Statement**

let day = 3;

switch (day) {

case 1:

console.log("Monday");

break;

case 2:

console.log("Tuesday");

break;

case 3:

console.log("Wednesday");

break;

default:

console.log("Another Day");

}

**🔄 Logical Operators Summary**

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| && | AND | true && false → false |
| || | OR | true |
| ! | NOT | !true → false |

**✅ Comparison Operators Summary**

| **Operator** | **Meaning** | **Example** | **Result** |
| --- | --- | --- | --- |
| == | Equal | 5 == “5” | true |
| === | Strictly Equal | 5 === “5” | false |
| != | Not Equal | 5 != 3 | true |
| !== | Strict Not Equal | 5 !== “5” | true |
| > | Greater Than | 5 > 3 | true |
| < | Less Than | 5 < 3 | false |
| >= | Greater or Equal | 5 >= 5 | true |
| <= | Less or Equal | 5 <= 4 | false |

**📘 Chapter 3 – Loops and Functions in JavaScript**

**🔁 Loops in JavaScript**

Loops are used to execute a block of code multiple times.

**1. for Loop**

*// Print numbers from 1 to 5*

for (let i = 1; i <= 5; i++) {

console.log(i); *// Output: 1 2 3 4 5*

}

**2. while Loop**

*// Print numbers from 1 to 5 using while loop*

let i = 1;

while (i <= 5) {

console.log(i);

i++;

}

**3. do…while Loop**

*// The loop executes at least once before checking the condition*

let j = 1;

do {

console.log(j);

j++;

} while (j <= 5);

**4. for…of Loop (used with arrays and iterable objects)**

const fruits = ["apple", "banana", "cherry"];

for (let fruit of fruits) {

console.log(fruit); *// Output: apple, banana, cherry*

}

**5. for…in Loop (used for object keys)**

const person = {name: "Rupesh", age: 19};

for (let key in person) {

console.log(key + ":" + person[key]);

*// Output: name:Rupesh, age:19*

}

**🔂 break and continue**

*// break example*

for (let i = 1; i <= 5; i++) {

if (i === 3) break;

console.log(i); *// Output: 1 2*

}

*// continue example*

for (let i = 1; i <= 5; i++) {

if (i === 3) continue;

console.log(i); *// Output: 1 2 4 5*

}

**🔧 Functions in JavaScript**

Functions are blocks of code designed to perform a particular task.

**1. Function Declaration**

function greet(name) {

console.log(`Hello, ${name}!`); *// Template literal used for string interpolation*

}

greet("Rupesh"); *// Output: Hello, Rupesh!*

**2. Function Expression**

const square = function(num) {

return num \* num;

};

console.log(square(4)); *// Output: 16*

**3. Arrow Function**

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

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

**4. Default Parameters**

function welcome(name = "Guest") {

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

}

welcome(); *// Output: Welcome, Guest*

**5. Returning Values**

function multiply(a, b) {

return a \* b; *// Returns the product of a and b*

}

console.log(multiply(3, 4)); *// Output: 12*

**6. Function Scope**

function testScope() {

let local = "I'm local";

console.log(local); *// Accessible here*

}

testScope();

*// console.log(local); // Error: local is not defined (outside the function scope)*

**7. Nested Functions**

function outer() {

function inner() {

console.log("Inside inner function");

}

inner();

}

outer();

**✅ Summary Table**

| **Loop/Function Type** | **Description** |
| --- | --- |
| for, while, do-while | Repeats block based on condition |
| for…of | Iterates over iterable objects like arrays |
| for…in | Iterates over object properties |
| Function Declaration | Named reusable code block |
| Arrow Function | Short syntax for functions |
| Function Expression | Anonymous function stored in variable |

**📘 Chapter 4 – String in JavaScript**

**1. What is a String?**

* A string is a sequence of characters used to represent text.
* Strings are enclosed in single quotes (') or double quotes (").

let str1 = 'Hello';

let str2 = "World";

**2. String Length**

* Returns the number of characters in a string.

console.log(str1.length); *// Output: 5*

**3. Accessing Characters in a String**

* You can access characters using their index.

console.log(str1[0]); *// Output: H*

**4. Looping Through a String**

for (let char of str1) {

console.log(char); *// Logs each character of the string*

}

**5. String Immutability**

* Strings are immutable, meaning their values cannot be changed directly.

str1[0] = 'h';

console.log(str1); *// Output: Hello (no change)*

**6. Useful String Methods**

**toUpperCase() and toLowerCase()**

console.log(str1.toUpperCase()); *// Output: HELLO*

console.log(str2.toLowerCase()); *// Output: world*

**trim() – Removes whitespace from both ends**

let messy = " spaced text ";

console.log(messy.trim()); *// Output: "spaced text"*

**slice(start, end) – Extracts part of a string**

console.log(str1.slice(1, 4)); *// Output: ell*

**substring(start, end) – Similar to slice**

console.log(str1.substring(0, 3)); *// Output: Hel*

**replace() – Replaces part of a string**

let text = "I like cats.";

console.log(text.replace("cats", "dogs")); *// Output: I like dogs.*

**includes() – Checks if string contains a substring**

console.log(text.includes("cats")); *// Output: true*

**indexOf() and lastIndexOf()**

let str = "banana";

console.log(str.indexOf("a")); *// Output: 1*

console.log(str.lastIndexOf("a")); *// Output: 5*

**split() – Converts string into an array**

let sentence = "This is a test.";

console.log(sentence.split(" ")); *// Output: ["This", "is", "a", "test."]*

**concat() – Joins two or more strings**

let full = str1.concat(" ", str2);

console.log(full); *// Output: Hello World*

**7. Template Literals**

* Use backticks (`) and ${} to embed expressions.

let name = "Rupesh";

let greeting = `Hello, ${name}!`;

console.log(greeting); *// Output: Hello, Rupesh!*

**8. Escape Characters**

* Use backslash \ to escape characters in strings.

let quote = 'It\'s a nice day';

console.log(quote); *// Output: It's a nice day*

**9. Comparing Strings**

console.log("apple" === "apple"); *// true*

console.log("apple" > "Apple"); *// true (case-sensitive comparison)*

**10. String Repeat**

console.log("ha".repeat(3)); *// Output: hahaha*

**🔁 String Methods Summary Table**

| **Method** | **Description** |
| --- | --- |
| length | Returns length of string |
| toUpperCase() | Converts to uppercase |
| toLowerCase() | Converts to lowercase |
| trim() | Removes whitespace |
| slice() | Extracts part of string |
| replace() | Replaces part of string |
| includes() | Checks if substring exists |
| indexOf() | Finds first occurrence |
| lastIndexOf() | Finds last occurrence |
| split() | Converts to array |
| concat() | Joins strings |
| repeat() | Repeats string |

**📘 Chapter 5 – Arrays in JavaScript**

**1. What is an Array?**

* An array is a collection of elements stored in a single variable.
* Elements can be of any type: numbers, strings, objects, or even other arrays.

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

console.log(fruits); *// Output: ["apple", "banana", "mango"]*

**2. Accessing Array Elements**

* Elements are accessed using index, starting from 0.

console.log(fruits[0]); *// Output: "apple"*

**3. Modifying Array Elements**

fruits[1] = "orange";

console.log(fruits); *// Output: ["apple", "orange", "mango"]*

**4. Array Length**

* Returns the number of elements in the array.

console.log(fruits.length); *// Output: 3*

**5. Looping Through Arrays**

**Using for loop:**

for (let i = 0; i < fruits.length; i++) {

console.log(fruits[i]);

}

**Using for…of:**

for (let fruit of fruits) {

console.log(fruit);

}

**Using forEach:**

fruits.forEach(function(fruit) {

console.log(fruit);

});

**6. Array Methods**

**push() – Add to end**

fruits.push("grapes");

console.log(fruits); *// ["apple", "orange", "mango", "grapes"]*

**pop() – Remove from end**

fruits.pop();

console.log(fruits); *// ["apple", "orange", "mango"]*

**shift() – Remove from start**

fruits.shift();

console.log(fruits); *// ["orange", "mango"]*

**unshift() – Add to start**

fruits.unshift("pineapple");

console.log(fruits); *// ["pineapple", "orange", "mango"]*

**indexOf()**

let index = fruits.indexOf("mango");

console.log(index); *// Output: 2*

**includes()**

console.log(fruits.includes("banana")); *// Output: false*

**splice() – Add/remove elements at specific index**

fruits.splice(1, 1, "kiwi");

console.log(fruits); *// ["pineapple", "kiwi", "mango"]*

**slice() – Returns a portion of the array**

let newFruits = fruits.slice(1, 3);

console.log(newFruits); *// ["kiwi", "mango"]*

**7. Concatenation of Arrays**

let moreFruits = ["papaya", "guava"];

let allFruits = fruits.concat(moreFruits);

console.log(allFruits);

**8. Array to String**

console.log(fruits.join(", ")); *// Output: pineapple, kiwi, mango*

**9. Sorting Arrays**

let numbers = [5, 3, 8, 1];

numbers.sort();

console.log(numbers); *// Output: [1, 3, 5, 8]*

**Custom Sort Function**

numbers.sort((a, b) => b - a);

console.log(numbers); *// Output: [8, 5, 3, 1]*

**10. Reversing Arrays**

numbers.reverse();

console.log(numbers); *// Output: [1, 3, 5, 8] → [8, 5, 3, 1]*

**11. Multidimensional Arrays**

let matrix = [

[1, 2],

[3, 4],

[5, 6]

];

console.log(matrix[1][0]); *// Output: 3*

**12. Important Array Methods Summary**

| **Method** | **Description** |
| --- | --- |
| push() | Add element to end |
| pop() | Remove element from end |
| shift() | Remove element from start |
| unshift() | Add element to start |
| indexOf() | Find index of an element |
| includes() | Check if array contains value |
| splice() | Add/remove elements |
| slice() | Return part of an array |
| concat() | Merge arrays |
| join() | Convert array to string |
| sort() | Sort elements |
| reverse() | Reverse order of array |

**📘 Chapter 6 - JavaScript in the Browser**

**1. What is JavaScript Engine?**

* A JavaScript engine is a program that executes JavaScript code.
* Examples:
  + **V8** – Used in Chrome and Node.js
  + **SpiderMonkey** – Used in Firefox
  + **JavaScriptCore** – Used in Safari

JavaScript code → Parsed → Compiled → Machine code → Executed

**2. Developer Tools (F12)**

* Press **F12** or **Right-click > Inspect** to open Developer Tools.
* Common tabs:
  + **Console** – Logs, errors, interaction
  + **Elements** – Inspect HTML/CSS
  + **Sources** – Debugging
  + **Network** – Monitor requests
  + **Application** – Cookies, local storage, etc.

**3. The <script> Tag**

* Used to include JavaScript in HTML.

<script src="script.js"></script>

* Place just before </body> for best performance.

**4. console Object Methods**

Useful for debugging JavaScript:

**4.1 console.assert()**

Logs an error if the assertion is false.

console.assert(2 + 2 === 5, "Assertion failed");

**4.2 console.clear()**

Clears the console.

console.clear();

**4.3 console.log()**

Displays general messages.

console.log("Hello, world!");

**4.4 console.table()**

Displays data as a table.

console.table([{name: "Alice", age: 20}, {name: "Bob", age: 25}]);

**4.5 console.warn()**

Shows a warning message.

console.warn("This is a warning!");

**4.6 console.error()**

Displays error messages.

console.error("This is an error message!");

**4.7 console.info()**

Displays informational messages.

console.info("This is some info.");

**5. Interaction: alert(), prompt(), confirm()**

**alert()**

Displays an alert box.

alert("Welcome to the site!");

**prompt()**

Asks user for input.

let name = prompt("What is your name?");

**confirm()**

Asks for confirmation.

let isSure = confirm("Are you sure?");

**6. Window Object**

* Global object representing the browser window.
* All global variables/functions are properties of window.

console.log(window.innerWidth); *// Gets browser width*

**6.1 BOM (Browser Object Model)**

* Provides interaction with the browser (not DOM).
* Includes:
  + navigator – browser info
  + screen – screen resolution
  + location – URL info
  + history – browsing history
  + alert, prompt, confirm

console.log(window.location.href); *// Current page URL*

**6.2 DOM (Document Object Model) – In Depth**

* Represents HTML as a tree of nodes.
* Allows dynamic content manipulation.

const title = document.querySelector("h1");

title.textContent = "Changed via JS";

🧠 Practice tip: Use developer tools actively to debug, inspect and interact with your code.

**📘 Chapter 7 - Walking the DOM (In-depth Notes)**

**1. What is DOM?**

* **DOM (Document Object Model)** is a programming interface for web documents.
* It represents the page as a **tree structure** where each node is an object representing a part of the document.

<body>

<div>Hello <b>world</b></div>

</body>

* The above becomes:

body

└── div

├── Text: "Hello "

└── b

└── Text: "world"

**2. Main Types of Nodes in the DOM Tree**

**2.1 Text Nodes**

* Contain plain text.

let textNode = document.body.firstChild; *// Text node: "\n"*

**2.2 Element Nodes**

* Represent HTML elements like <div>, <p>, etc.

let elementNode = document.body; *// Element node: <body>*

**2.3 Comment Nodes**

* Represent HTML comments.

*<!-- This is a comment -->*

let comment = document.body.childNodes[0];

console.log(comment.nodeType); *// 8*

**3. Auto Correction**

* Browsers auto-correct broken HTML.

<table>

<tr>Test</tr>

</table>

* Becomes:

<table>

<tbody>

<tr>Test</tr>

</tbody>

</table>

**4. Walking the DOM (HTML Boilerplate Explanation)**

DOM traversal lets you navigate the tree structure:

let html = document.documentElement; *// <html>*

let body = html.lastElementChild; *// <body>*

let head = html.firstElementChild; *// <head>*

**5. Children of Element**

**5.1 firstChild**

* Returns the first child node (can be text, comment, etc.)

console.log(document.body.firstChild);

**5.2 lastChild**

* Returns the last child node.

console.log(document.body.lastChild);

**5.3 childNodes**

* Returns a live NodeList of all child nodes.

for (let node of document.body.childNodes) {

console.log(node);

}

**5.4 element.hasChildNodes()**

* Checks if element has any child nodes.

console.log(document.body.hasChildNodes());

**6. DOM Collection**

* DOM collections like childNodes, getElementsByTagName, etc. are **live**, auto-updating with DOM changes.

**7. Element-Only Navigation**

let elem = document.body;

elem.firstElementChild;

elem.lastElementChild;

elem.previousElementSibling;

elem.nextElementSibling;

elem.parentElement;

**8. Table Links (Working with Tables)**

<table id="myTable">

<caption>Table Caption</caption>

<thead><tr><th>Head</th></tr></thead>

<tfoot><tr><td>Footer</td></tr></tfoot>

<tbody><tr><td>Body Row</td></tr></tbody>

</table>

let table = document.getElementById("myTable");

**8.1 table.rows – all rows**

console.log(table.rows);

**8.2 table.caption**

console.log(table.caption.textContent);

**8.3 table.tHead**

console.log(table.tHead);

**8.4 table.tFoot**

console.log(table.tFoot);

**8.5 table.tBodies**

console.log(table.tBodies);

**8.6 tbody.rows**

console.log(table.tBodies[0].rows);

**8.7 tr.cells**

let row = table.rows[0];

console.log(row.cells);

**8.8 tr.sectionRowIndex**

console.log(row.sectionRowIndex); *// Index in section*

**8.9 tr.rowIndex**

console.log(row.rowIndex); *// Index in whole table*

**8.10 td.cellIndex**

let cell = row.cells[0];

console.log(cell.cellIndex);

**9. Searching the DOM**

**9.1 document.getElementById()**

document.getElementById("myId");

**9.2 document.querySelectorAll()**

document.querySelectorAll("div.note");

**9.3 document.querySelector()**

document.querySelector("p");

**9.4 document.getElementsByTagName()**

document.getElementsByTagName("li");

**9.5 document.getElementsByClassName()**

document.getElementsByClassName("box");

**9.6 document.getElementsByName()**

document.getElementsByName("gender");

**10. Matches, Closest, and Contains Methods**

**10.1 elem.matches(css)**

* Returns true if element matches the CSS selector.

if (elem.matches(".active")) {

console.log("Active element found.");

}

**10.2 elem.closest(css)**

* Searches ancestors until it finds a match.

let section = elem.closest("section");

**10.3 eleA.contains(eleB)**

* Checks if eleA contains eleB.

console.log(div.contains(span));

✨ Practice Tip: Open DevTools and try DOM navigation in the Console tab. Try modifying and accessing nodes live!

**📚 Chapter 8 - Events and Other DOM Properties**

**1. console.dir() Function**

* Used to display an interactive list of all properties of a specified JavaScript object.
* Especially useful for examining DOM elements.

const heading = document.querySelector("h1");

console.dir(heading); *// Shows all properties and methods of the h1 element*

**2. tagName / nodeName**

* tagName returns the tag name in **uppercase** (HTML elements).
* nodeName returns the name of the **node** (not always an HTML tag).

const element = document.querySelector("div");

console.log(element.tagName); *// DIV*

console.log(element.nodeName); *// DIV*

**3. innerHTML and outerHTML**

* innerHTML: Gets or sets the **HTML inside** an element.
* outerHTML: Gets or sets the **HTML including the element** itself.

const div = document.querySelector(".content");

console.log(div.innerHTML); *// Inside content*

console.log(div.outerHTML); *// Entire div + content*

**4. textContent**

* Returns the text **without HTML tags**.

const para = document.querySelector("p");

console.log(para.textContent); *// Just the text*

**5. hidden Property**

* Use .hidden = true to hide an element.

const box = document.querySelector(".box");

box.hidden = true; *// Hides the element*

**6. Attributes**

* Access attributes using .getAttribute() and .setAttribute().

const link = document.querySelector("a");

console.log(link.getAttribute("href"));

link.setAttribute("href", "https://example.com");

**7. data-\* Attributes**

* Used to store custom data on elements.

<div data-user-id="123" data-role="admin"></div>

const el = document.querySelector("div");

console.log(el.dataset.userId); *// "123"*

console.log(el.dataset.role); *// "admin"*

**8. Inserting Methods**

* .append(), .prepend(), .before(), .after()

div.append("Text"); *// Inside at the end*

div.prepend("Start"); *// Inside at the beginning*

div.before("Above"); *// Outside before*

div.after("Below"); *// Outside after*

**9. Insert Adjacent HTML/Text/Element**

* .insertAdjacentHTML(position, html)

div.insertAdjacentHTML("beforebegin", "<p>Before div</p>");

div.insertAdjacentHTML("afterend", "<p>After div</p>");

**10. Node Removal**

* .remove() deletes an element from the DOM.

const btn = document.querySelector("button");

btn.remove(); *// Removes the button*

**11. className and classList**

* className: Gets or sets class as a string.
* classList: Useful methods: add, remove, toggle, contains

const box = document.querySelector(".box");

console.log(box.className); *// "box red"*

box.classList.add("blue");

box.classList.remove("red");

box.classList.toggle("hidden");

**12. setTimeout() and setInterval()**

* setTimeout(fn, delay): Runs once after delay.
* setInterval(fn, delay): Repeats every delay.

setTimeout(() => {

console.log("Executed after 2 seconds");

}, 2000);

let count = 0;

const timer = setInterval(() => {

console.log(++count);

if (count === 5) clearInterval(timer);

}, 1000);

**13. Browser Events**

* Common events: click, mouseover, keydown, submit, etc.

button.addEventListener("click", () => {

alert("Clicked!");

});

**14. Handling Events**

* Inline in HTML: <button onclick="greet()">Click</button>
* Or using JavaScript addEventListener()

function greet() {

console.log("Hello!");

}

button.onclick = greet;

**15. addEventListener() and removeEventListener()**

function sayHello() {

console.log("Hello!");

}

button.addEventListener("click", sayHello);

button.removeEventListener("click", sayHello);

**16. The event Object**

* Automatically passed to the event handler.
* Contains info like target, type, key, clientX, clientY etc.

button.addEventListener("click", function(event) {

console.log("Event type:", event.type);

console.log("Clicked element:", event.target);

});

✅ Practice DOM events and manipulation to make your JavaScript interactive and powerful!

**📘 Chapter 9: Callbacks, Promises, and Async/Await**

**1. What is Asynchronous?**

console.log("Start"); *// Synchronous log*

setTimeout(() => console.log("Async task done"), 1000);

*// Asynchronous: Executes after 1 second, doesn't block next code*

console.log("End"); *// Prints before the async task finishes*

✅ **Key Point:** JavaScript doesn’t wait for setTimeout. It continues executing other code.

**2. Callback Functions (Explained in Depth)**

function fetchData(callback) {

setTimeout(() => {

console.log("Data fetched");

callback(); *// Executes after data is fetched*

}, 1000);

}

function displayData() {

console.log("Display data");

}

fetchData(displayData);

✅ **Key Point:** Callbacks are functions passed as arguments and executed after a task is completed.

**3. Handling Errors with Callbacks**

function fetchData(callback) {

setTimeout(() => {

const error = true;

if (error) {

callback("Something went wrong", null);

} else {

callback(null, "Data loaded");

}

}, 1000);

}

fetchData((err, data) => {

if (err) console.error(err);

else console.log(data);

});

✅ **Key Point:** First parameter is for error, second is for success.

**4. The Pyramid of Doom (Callback Hell)**

login(user => {

fetchProfile(user, profile => {

fetchPosts(profile, posts => {

console.log(posts);

});

});

});

🔺 **Problem:** Deep nesting makes the code hard to read and maintain.

**5. Introduction to Promises**

let promise = new Promise((resolve, reject) => {

setTimeout(() => resolve("Success!"), 1000);

});

✅ **Key Point:** A Promise represents a value that may be available now, later, or never.

**6. Consumers: .then() and .catch()**

promise

.then(result => console.log(result)) *// Handles resolved promise*

.catch(error => console.error(error)); *// Handles rejected promise*

**7. Promise Chaining**

new Promise(resolve => resolve(1))

.then(result => result \* 2)

.then(result => result + 1)

.then(console.log); *// Output: 3*

✅ **Key Point:** Each .then() returns a new promise, enabling chaining.

**8. Attaching Multiple Handlers**

let p = new Promise(resolve => setTimeout(() => resolve("Done"), 1000));

p.then(result => console.log("First handler:", result));

p.then(result => console.log("Second handler:", result));

**9. Promise API Methods**

Promise.all([

Promise.resolve(1),

Promise.resolve(2)

]).then(console.log); *// Output: [1, 2]*

**Methods:**

* **Promise.all([])** – Resolves when all succeed, rejects if any fail.
* **Promise.race([])** – Resolves/rejects as soon as one settles.
* **Promise.allSettled([])** – Waits for all promises, returns both success/failure.
* **Promise.any([])** – Resolves as soon as any one promise resolves.

**10. Async/Await**

async function fetchData() {

try {

let res = await fetch("https://api.example.com");

let data = await res.json();

console.log(data);

} catch (err) {

console.error("Error:", err);

}

}

fetchData();

✅ **Key Point:** async functions return a promise. await pauses until the promise settles.

**11. Error Handling in Async/Await**

async function run() {

try {

let result = await someAsyncFunction();

} catch (error) {

console.error("Caught Error:", error);

}

}

**12. Throwing Custom Errors**

function checkAge(age) {

if (age < 18) {

throw new Error("You must be at least 18.");

}

return true;

}

try {

checkAge(15);

} catch (e) {

console.error(e.message);

}

**13. The finally Clause**

fetch("/api/data")

.then(response => response.json())

.catch(error => console.error(error))

.finally(() => console.log("Request finished"));

✅ **Key Point:** finally() always runs—whether promise was resolved or rejected.

**🧠 Practice Questions**

**✅ Basic Understanding**

1. What is the main problem with callback-based code?
2. How is a promise better than a callback?
3. What are the three states of a JavaScript Promise?

**✅ Code Output Practice**

console.log("Start");

setTimeout(() => {

console.log("Middle");

}, 0);

console.log("End");

❓ What will be the output?

**✅ Fill in the Blanks**

1. A function passed to another function as an argument is called a **\_**.
2. The .then() method handles **\_**, while .catch() handles **\_**.
3. The await keyword can only be used inside **\_** functions.

**✅ Conceptual**

1. What is the difference between Promise.all() and Promise.race()?
2. Why is async/await considered syntactic sugar over promises?
3. Explain how you would handle multiple API calls and display the data only when all succeed.

**📘 Chapter 10 – Object Oriented Programming in JavaScript**

**1. What is OOP?**

* OOP (Object-Oriented Programming) is a way of structuring code using **objects** that contain both **data (properties)** and **functions (methods)**.

**2. Prototype**

const person = {

greet() {

console.log("Hello!"); *// Method defined in the person object*

}

};

const student = Object.create(person); *// Create a new object student with person as its prototype*

student.name = "Ravi"; *// Add a new property to student*

student.greet(); *// Access greet method from the prototype (person)*

**3. Setting Prototype**

const animal = {

speak() {

console.log("Animal speaks"); *// Method that will be inherited*

}

};

const dog = {}; *// Create an empty object*

Object.setPrototypeOf(dog, animal); *// Set animal as the prototype of dog*

dog.speak(); *// Call the method from the prototype*

**4. Classes and Objects**

class Person {

constructor(name) {

this.name = name; *// Set instance property*

}

greet() {

console.log(`Hi, I am ${this.name}`); *// Method of the class*

}

}

const john = new Person("John"); *// Create an instance of Person*

john.greet(); *// Call greet method*

**5. The Constructor Method**

class Car {

constructor(brand) {

this.brand = brand; *// Initialize the brand when object is created*

}

}

const myCar = new Car("Toyota"); *// Create a new Car object*

console.log(myCar.brand); *// Access property*

**6. Class Inheritance**

class Animal {

speak() {

console.log("Animal sound"); *// Method in base class*

}

}

class Dog extends Animal {} *// Dog class inherits from Animal*

const d = new Dog(); *// Create Dog instance*

d.speak(); *// Inherited method from Animal*

**7. The extends Keyword**

* extends creates an inheritance relationship.
* The prototype chain is automatically set up, so the child class inherits the parent class methods and properties.

**8. Method Overriding**

class Animal {

sound() {

console.log("Generic animal sound"); *// Base method*

}

}

class Cat extends Animal {

sound() {

console.log("Meow"); *// Overrides parent class method*

}

}

const kitty = new Cat(); *// Create a Cat object*

kitty.sound(); *// Calls overridden method*

**9. super Keyword**

class Vehicle {

constructor(name) {

this.name = name; *// Set name in base class*

}

}

class Bike extends Vehicle {

constructor(name, type) {

super(name); *// Call parent constructor*

this.type = type; *// Set new property in child class*

}

}

const b = new Bike("Hero", "Mountain");

console.log(b.name, b.type); *// Output: Hero Mountain*

**10. Overriding Constructor**

class Parent {

constructor() {

console.log("Parent constructor"); *// Base class constructor*

}

}

class Child extends Parent {

constructor() {

super(); *// Must call super() before using 'this'*

console.log("Child constructor"); *// Child class constructor*

}

}

new Child(); *// Output: Parent constructor \n Child constructor*

**11. Static Methods**

class MathUtil {

static square(x) {

return x \* x; *// Static method doesn't rely on object instance*

}

}

console.log(MathUtil.square(5)); *// Call directly on class, Output: 25*

**12. Getters and Setters**

class User {

constructor(name) {

this.\_name = name; *// Use underscore to show it's a private variable (by convention)*

}

get name() {

return this.\_name.toUpperCase(); *// Getter returns name in uppercase*

}

set name(value) {

if (value.length < 3) {

console.log("Name too short"); *// Validation*

} else {

this.\_name = value; *// Set the name if valid*

}

}

}

let u = new User("Sam");

console.log(u.name); *// Output: SAM*

u.name = "Al"; *// Output: Name too short*

**13. instanceof Operator**

class Animal {}

class Dog extends Animal {}

let tommy = new Dog();

console.log(tommy instanceof Dog); *// true - tommy is instance of Dog*

console.log(tommy instanceof Animal); *// true - because Dog inherits Animal*

📘 **Chapter 11 - Advanced JavaScript**

**1. IIFE (Immediately Invoked Function Expression)**

**📌 Definition:**

* A function that **executes right after it’s defined**, without being explicitly called.

**🔧 Syntax:**

(function() {

console.log("IIFE executed!");

})(); *// ← Called immediately*

**✅ Why use it?**

* Prevents **global variable pollution**
* Creates a **private scope** for variables

**💡 Example:**

(function() {

let message = "Hello from IIFE!";

console.log(message);

})();

*// console.log(message); ❌ Error: message is not defined*

**2. Destructuring**

**📌 Definition:**

* Allows extracting values from arrays or objects and assigning them to variables.

**🔹 Array Destructuring:**

const [a, b] = [10, 20];

console.log(a); *// 10*

console.log(b); *// 20*

**🔸 Object Destructuring:**

const person = { name: "Rupesh", age: 19 };

const { name, age } = person;

console.log(name); *// Rupesh*

console.log(age); *// 19*

**3. Spread Syntax (...)**

**📌 Definition:**

* Expands iterable elements or object properties.

**🔹 Arrays:**

const arr1 = [1, 2, 3];

const arr2 = [...arr1, 4, 5]; *// Clones + adds*

console.log(arr2); *// [1, 2, 3, 4, 5]*

**🔸 Objects:**

const obj1 = { a: 1, b: 2 };

const obj2 = { ...obj1, c: 3 }; *// Copies properties and adds*

console.log(obj2); *// { a: 1, b: 2, c: 3 }*

**4. Hoisting**

**📌 Definition:**

* JavaScript **hoists declarations** to the top of the current scope (function or global).

**⚠️ Example:**

console.log(x); *// undefined*

var x = 5;

**❗ Explanation:**

* var x is hoisted (moved up), but **not the value 5**.

**5. Hoisting with let and const**

**⚠️ Example:**

console.log(a); *// ❌ ReferenceError*

let a = 10;

**❗ Explanation:**

* let and const are hoisted but **not initialized**.
* They stay in the **Temporal Dead Zone (TDZ)** until the declaration line.

**6. Scope Types**

**🌐 Global Scope:**

* Declared **outside** any function/block — accessible everywhere.

var globalVar = "I am global";

**🔒 Local (Function) Scope:**

* Declared **inside** a function — accessible only within that function.

function greet() {

var localVar = "Hello";

console.log(localVar);

}

**📦 Block Scope (let and const):**

{

let blockVar = "I am block scoped";

console.log(blockVar);

}

*// console.log(blockVar); ❌ Not accessible here*

**🧠 Summary:**

| **Feature** | **Purpose** | **Key Tip** |
| --- | --- | --- |
| IIFE | Executes instantly, avoids globals | Wrap in () and invoke |
| Destructuring | Easy variable assignment from arrays/objects | Use {} for objects, [] for arrays |
| Spread ... | Clone or merge arrays/objects | Works on iterables |
| Hoisting | Moves declarations to top | let/const → TDZ |
| Scopes | Controls variable access | Use let/const over var |

**📘 Chapter 12 - JavaScript Cheatsheet**

**1. Variables and Data Types**

*// Declaring variables*

let age = 25; *// Block-scoped variable*

const name = "John"; *// Constant value*

var city = "Delhi"; *// Function-scoped (not recommended)*

*// Data types*

let num = 42; *// Number*

let str = "Hello"; *// String*

let isTrue = true; *// Boolean*

let user = {name: "Ali"}; *// Object*

let arr = [1, 2, 3]; *// Array*

let und; *// Undefined*

let nul = null; *// Null*

**2. Expression and Condition**

*// Arithmetic Operators: + - \* / % \*\**

let result = 3 + 2 \* 4;

*// Comparison: == != === !== > < >= <=*

if (age >= 18) {

console.log("Adult");

}

*// Logical: && || !*

if (age > 18 && isTrue) {

console.log("Allowed");

}

**3. Array**

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

fruits.push("orange"); *// Add to end*

fruits.pop(); *// Remove from end*

fruits.shift(); *// Remove from start*

fruits.unshift("mango"); *// Add to start*

*// Iteration*

fruits.forEach(fruit => console.log(fruit));

**4. String**

let text = "JavaScript";

console.log(text.length); *// Length*

console.log(text.toUpperCase());

console.log(text.indexOf("S"));

console.log(text.slice(0, 4)); *// Substring*

**5. Loop and Function**

*// Loop*

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

console.log(i);

}

*// Function*

function greet(name) {

return `Hello ${name}`;

}

const greet2 = name => `Hi ${name}`;

**6. JavaScript in the Browser**

*// script tag*

<script src="script.js"></script>

*// console methods*

console.log("Log");

console.warn("Warning");

console.error("Error");

**7. Walking the DOM**

let element = document.getElementById("demo");

element.textContent = "Updated";

*// Traverse*

let parent = element.parentElement;

let children = element.children;

**8. Events and Other DOM Properties**

*// Event Handling*

button.addEventListener("click", () => {

alert("Clicked!");

});

*// Attributes*

element.setAttribute("id", "main");

console.log(element.getAttribute("id"));

**9. Callback, Promises and Async/Await**

*// Callback*

function fetchData(cb) {

setTimeout(() => cb("Data"), 1000);

}

*// Promise*

let promise = new Promise((resolve, reject) => {

resolve("Done");

});

promise.then(data => console.log(data));

*// Async/Await*

async function getData() {

let res = await fetch("/api");

let data = await res.json();

console.log(data);

}

**10. Object Oriented Programming**

class Animal {

constructor(name) {

this.name = name;

}

speak() {

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

}

}

class Dog extends Animal {

speak() {

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

}

}

let dog = new Dog("Rex");

dog.speak();

**11. Advanced JavaScript**

*// IIFE*

(function() {

console.log("IIFE executed");

})();

*// Destructuring*

let [a, b] = [1, 2];

let {x, y} = {x: 10, y: 20};

*// Spread*

let arr = [1, 2];

let newArr = [...arr, 3, 4];

*// Hoisting*

console.log(hoistedVar); *// undefined*

var hoistedVar = 10;

*// Scope*

let globalVar = "global";

{

let blockVar = "block";

}

**📘 Chapter 13 – Summary**

**0. 🚀 Introduction to Programming**

* Programming is giving instructions to a computer to perform tasks.
* JavaScript is a high-level, interpreted scripting language used primarily for web development.

console.log("Hello, world!"); *// Prints to the browser console*

**1. 📦 Variables and Data Types**

**Declaring Variables**

let name = "Alice"; *// mutable*

const age = 25; *// constant*

var city = "Delhi"; *// function-scoped (not recommended)*

**Data Types**

* String: "Hello"
* Number: 42, 3.14
* Boolean: true, false
* Null: null
* Undefined: undefined
* Object, Array, Function

let user = { name: "Bob", age: 30 }; *// object*

let scores = [10, 20, 30]; *// array*

**2. 🔣 Expressions and Conditions**

**Conditional Statements**

if (age > 18) {

console.log("Adult");

} else {

console.log("Minor");

}

**Ternary Operator**

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

**3. 🔁 Loops and Functions**

**Loops**

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

console.log(i);

}

let i = 0;

while (i < 5) {

console.log(i);

i++;

}

**Functions**

function greet(name) {

return `Hello, ${name}`;

}

const square = x => x \* x; *// Arrow function*

**4. 🔤 Strings**

let text = "Hello";

console.log(text.length); *// 5*

console.log(text.toUpperCase()); *// "HELLO"*

console.log(text.includes("e")); *// true*

**5. 📚 Arrays**

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

fruits.push("mango"); *// add*

fruits.pop(); *// remove last*

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

*// Looping through array*

fruits.forEach(fruit => console.log(fruit));

**6. 🧠 JavaScript in the DOM**

const el = document.getElementById("demo");

el.innerHTML = "Updated text";

**7. 🌳 Walking the DOM**

const root = document.body;

console.log(root.firstChild);

console.log(root.children);

**8. ⚡ Events and Other DOM Properties**

*// Add event listener*

button.addEventListener("click", () => {

alert("Button clicked");

});

*// Remove event listener*

function handler() {}

button.removeEventListener("click", handler);

**9. ⏳ Callbacks, Promises, and Async/Await**

**Callback**

function loadData(callback) {

setTimeout(() => {

callback("Data loaded");

}, 1000);

}

**Promise**

const promise = new Promise((resolve, reject) => {

setTimeout(() => resolve("Done"), 1000);

});

promise.then(result => console.log(result));

**Async/Await**

async function getData() {

const res = await fetch("https://api.example.com");

const data = await res.json();

console.log(data);

}

**10. 🧱 Object-Oriented Programming**

**Class and Object**

class Person {

constructor(name) {

this.name = name;

}

greet() {

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

}

}

const user = new Person("Alice");

user.greet();

**Inheritance**

class Employee extends Person {

constructor(name, id) {

super(name); *// Call parent constructor*

this.id = id;

}

}

**Static Methods and Getters**

class MathTool {

static add(x, y) {

return x + y;

}

}

console.log(MathTool.add(3, 5));