Day 1 - JavaScript Basics

Syntax, Variables, Data Types, Arrays, Objects

📌 JavaScript Syntax

What is JavaScript?

JavaScript is a programming language that makes web pages interactive and dynamic.

How JavaScript Executes

- **V8 Engine** - Runs JavaScript code
- **DOM** - Document Object Model (HTML structure)
- **CSSOM** - CSS Object Model (Style structure)

Declaration vs Initialization vs Expression vs Statement

```javascript
// Declaration - Creating a variable
var a;

// Initialization - Assigning a value
var a = 10;

// Expression - Always produces a value
var result = 5 + 10; // Expression: 5 + 10 = 15

// Statement - Performs an action
if (result > 10) {
  console.log("Greater than 10");
}
```

Key Difference:

- **Expression** = Has a value (can store, update, reuse)
- **Statement** = Performs an action (no value)

🔤 Variables

Variables are containers that store data values.

Three Types of Variables

Keyword	Scope	Re-declare?	Re-assign?	Hoisting
var	Function/Global	✅ Yes	✅ Yes	undefined
let	Block	❌ No	✅ Yes	TDZ*
const	Block	❌ No	❌ No	TDZ*

*TDZ = Temporal Dead Zone

VAR - Function Scope

javascript

```javascript
// Global scope
var name = "John";

function test() {
  var age = 25; // Function scope
  console.log(age); // 25
}

console.log(name); // John
// console.log(age); // Error: age is not defined

// Re-declaration allowed
var name = "John";
var name = "Jane"; // ✅ Works
console.log(name); // Jane

// Re-assignment allowed
var count = 10;
count = 20; // ✅ Works
console.log(count); // 20

// Function scope example
function varExample() {
  var x = 10;

  if (true) {
    var x = 20; // Same variable (function scope)
    console.log(x); // 20
  }

  console.log(x); // 20 (changed)
}
```

LET - Block Scope

javascript

```
// Global scope
let name = "John";

// Block scope
if (true) {
  let age = 25;
  console.log(age); // 25
}

// console.log(age); // Error: age is not defined

// Re-declaration NOT allowed
let name = "John";
// let name = "Jane"; // ❌ Error: Identifier 'name' has already been declared

// Re-assignment allowed
let count = 10;
count = 20; // ✅ Works
console.log(count); // 20

// Block scope example
function letExample() {
  let y = 10;

  if (true) {
    let y = 20; // Different variable (block scope)
    console.log(y); // 20
  }

  console.log(y); // 10 (unchanged)
}

// Another block scope example
{
  let blockVar = "inside block";
  console.log(blockVar); // inside block
}
// console.log(blockVar); // Error: not defined
```

CONST - Block Scope, Cannot Reassign

javascript

```
// Global scope
const PI = 3.14159;

// Re-declaration NOT allowed
const name = "John";
// const name = "Jane"; // ❌ Error

// Re-assignment NOT allowed
const count = 10;
// count = 20; // ❌ Error: Assignment to constant variable

// Empty declaration NOT allowed
// const x; // ❌ Error: Missing initializer

// Must initialize when declaring
const age = 25; // ✅ Correct

// Block scope
if (true) {
  const temp = 100;
  console.log(temp); // 100
}
// console.log(temp); // Error: not defined

// IMPORTANT: Objects and Arrays can be modified
const person = {
  name: "John",
  age: 30
};

person.name = "Jane"; // ✅ Works (modifying property)
person.age = 31; // ✅ Works
console.log(person); // { name: "Jane", age: 31 }

// person = {}; // ❌ Error: Cannot reassign

const colors = ["red", "green", "blue"];
colors.push("yellow"); // ✅ Works (modifying array)
console.log(colors); // ["red", "green", "blue", "yellow"]

// colors = []; // ❌ Error: Cannot reassign
```

Hoisting

Hoisting is JavaScript's behavior of moving declarations to the top of their scope.

```javascript
// Example 1: var hoisting
console.log(a); // undefined (declaration hoisted)
var a = 10;
console.log(a); // 10

// Behind the scenes:
// var a; // Declaration moved to top
// console.log(a); // undefined
// a = 10; // Initialization stays

// Example 2: let hoisting
console.log(b); // ❌ ReferenceError: Cannot access 'b' before initialization
let b = 20;

// Example 3: const hoisting
console.log(c); // ❌ ReferenceError: Cannot access 'c' before initialization
const c = 30;

// Example 4: Function hoisting
sayHello(); // ✅ Works! "Hello"
function sayHello() {
  console.log("Hello");
}

// Example 5: Function expression NOT hoisted
sayHi(); // ❌ TypeError: sayHi is not a function
var sayHi = function() {
  console.log("Hi");
};
```

Interview Answer: "Hoisting moves variable and function declarations to the top of their scope during compilation. `var` is hoisted and initialized with `undefined`. `let` and `const` are hoisted but remain in the Temporal Dead Zone (TDZ) until their line is executed."

📊 Data Types

JavaScript has **2 categories** of data types:

1. **Primitive** - Store single values
2. **Non-Primitive** - Store collections/complex data

Primitive Data Types

1. String

```javascript
let firstName = "John";
let lastName = 'Doe';
let greeting = `Hello, ${firstName}!`; // Template literal

// String methods
let text = "Hello World";
console.log(text.length); // 11
console.log(text.toUpperCase()); // HELLO WORLD
console.log(text.toLowerCase()); // hello world
console.log(text.indexOf("World")); // 6
console.log(text.slice(0, 5)); // Hello
console.log(text.replace("World", "JavaScript")); // Hello JavaScript
console.log(text.split(" ")); // ["Hello", "World"]
```

2. Number

```javascript
let age = 25;
let price = 99.99;
let negative = -10;
let infinity = Infinity;

// Number methods
let num = 10.56789;
console.log(num.toFixed(2)); // "10.57"
console.log(parseInt("100")); // 100
console.log(parseFloat("10.5")); // 10.5
console.log(Math.round(num)); // 11
console.log(Math.floor(num)); // 10
console.log(Math.ceil(num)); // 11
console.log(Math.max(10, 20, 30)); // 30
console.log(Math.min(10, 20, 30)); // 10
console.log(Math.random()); // Random number between 0-1
```

3. Boolean

javascript

```
let isActive = true;
let isLoggedIn = false;

// Boolean comparison
console.log(10 > 5); // true
console.log(10 < 5); // false
console.log(10 == "10"); // true (loose equality)
console.log(10 === "10"); // false (strict equality)
```

4. Null

```javascript
let data = null; // Intentional absence of value

console.log(typeof null); // "object" (JavaScript bug)
console.log(data); // null
```

5. Undefined

```javascript
let x; // Declared but not initialized
console.log(x); // undefined

let person = {
  name: "John"
};
console.log(person.age); // undefined (property doesn't exist)

function test() {
  // No return statement
}
console.log(test()); // undefined
```

6. Symbol (ES6)

```javascript
let id1 = Symbol("id");
let id2 = Symbol("id");
console.log(id1 === id2); // false (always unique)
```

7. BigInt (ES11)

```javascript
let bigNumber = 9007199254740991n;
let anotherBig = BigInt(9007199254740991);
```

Checking Data Types

```javascript
console.log(typeof "Hello"); // string
console.log(typeof 42); // number
console.log(typeof true); // boolean
console.log(typeof undefined); // undefined
console.log(typeof null); // object (bug)
console.log(typeof []); // object
console.log(typeof {}); // object
console.log(typeof function(){}); // function
```

🗂️ Non-Primitive Data Types

1. Arrays

Arrays store multiple values in a single variable.

```javascript
// Creating arrays
let fruits = ["apple", "banana", "orange"];
let numbers = [1, 2, 3, 4, 5];
let mixed = [1, "hello", true, null, {name: "John"}];

// Accessing elements (0-indexed)
console.log(fruits[0]); // apple
console.log(fruits[1]); // banana
console.log(fruits[2]); // orange

// Array length
console.log(fruits.length); // 3

// Modifying arrays
fruits[1] = "mango";
console.log(fruits); // ["apple", "mango", "orange"]
```

Array Methods

```
javascript
```

```
let arr = ["apple", "banana", "orange"];

// 1. push() - Add to end
arr.push("mango");
console.log(arr); // ["apple", "banana", "orange", "mango"]

// 2. pop() - Remove from end
let removed = arr.pop();
console.log(removed); // mango
console.log(arr); // ["apple", "banana", "orange"]

// 3. unshift() - Add to start
arr.unshift("grape");
console.log(arr); // ["grape", "apple", "banana", "orange"]

// 4. shift() - Remove from start
let first = arr.shift();
console.log(first); // grape
console.log(arr); // ["apple", "banana", "orange"]

// 5. splice() - Add/remove at specific position
arr.splice(1, 0, "kiwi"); // At index 1, remove 0, add "kiwi"
console.log(arr); // ["apple", "kiwi", "banana", "orange"]

arr.splice(2, 1); // At index 2, remove 1 element
console.log(arr); // ["apple", "kiwi", "orange"]

// 6. slice() - Extract portion (doesn't modify original)
let sliced = arr.slice(0, 2);
console.log(sliced); // ["apple", "kiwi"]
console.log(arr); // ["apple", "kiwi", "orange"] (unchanged)

// 7. concat() - Join arrays
let arr1 = [1, 2];
let arr2 = [3, 4];
let combined = arr1.concat(arr2);
console.log(combined); // [1, 2, 3, 4]

// 8. join() - Convert to string
let words = ["Hello", "World"];
console.log(words.join(" ")); // "Hello World"
console.log(words.join("-")); // "Hello-World"

// 9. indexOf() - Find index of element
let colors = ["red", "green", "blue"];
console.log(colors.indexOf("green")); // 1
console.log(colors.indexOf("yellow")); // -1 (not found)

// 10. includes() - Check if element exists
console.log(colors.includes("red")); // true
console.log(colors.includes("yellow")); // false

// 11. reverse() - Reverse array
let nums = [1, 2, 3, 4, 5];
nums.reverse();
console.log(nums); // [5, 4, 3, 2, 1]

// 12. sort() - Sort array
let names = ["John", "Alice", "Bob"];
names.sort();
console.log(names); // ["Alice", "Bob", "John"]

let numbers = [5, 2, 8, 1, 9];
numbers.sort((a, b) => a - b); // Ascending
console.log(numbers); // [1, 2, 5, 8, 9]
```

Array Iteration Methods

javascript

```javascript
let numbers = [1, 2, 3, 4, 5];

// 1. forEach() - Execute function for each element
numbers.forEach((num, index) => {
  console.log(`Index ${index}: ${num}`);
});

// 2. map() - Create new array with transformed elements
let doubled = numbers.map(num => num * 2);
console.log(doubled); // [2, 4, 6, 8, 10]

// 3. filter() - Create new array with filtered elements
let evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // [2, 4]

// 4. find() - Find first matching element
let found = numbers.find(num => num > 3);
console.log(found); // 4

// 5. findIndex() - Find index of first matching element
let index = numbers.findIndex(num => num > 3);
console.log(index); // 3

// 6. some() - Check if at least one element matches
let hasEven = numbers.some(num => num % 2 === 0);
console.log(hasEven); // true

// 7. every() - Check if all elements match
let allPositive = numbers.every(num => num > 0);
console.log(allPositive); // true

// 8. reduce() - Reduce array to single value
let sum = numbers.reduce((total, num) => total + num, 0);
console.log(sum); // 15
```

🏗️ Objects

Objects store data in key-value pairs.

javascript

```javascript
// Creating objects
let person = {
  name: "John Doe",
  age: 30,
  email: "john@example.com",
  isActive: true
};

// Accessing properties
console.log(person.name); // Dot notation
console.log(person["age"]); // Bracket notation

// Adding new properties
person.phone = "123-456-7890";
person["address"] = "New York";

// Modifying properties
person.age = 31;
person["email"] = "johndoe@example.com";

// Deleting properties
delete person.isActive;

console.log(person);
```

Object Methods

javascript

```
let user = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  email: "john@example.com"
};

// 1. Object.keys() - Get all keys
console.log(Object.keys(user)); // ["firstName", "lastName", "age", "email"]

// 2. Object.values() - Get all values
console.log(Object.values(user)); // ["John", "Doe", 30, "john@example.com"]

// 3. Object.entries() - Get key-value pairs
console.log(Object.entries(user));
// [["firstName", "John"], ["lastName", "Doe"], ...]

// 4. Object.assign() - Copy/merge objects
let copy = Object.assign({}, user);
console.log(copy);

// 5. hasOwnProperty() - Check if property exists
console.log(user.hasOwnProperty("age")); // true
console.log(user.hasOwnProperty("phone")); // false

// 6. Loop through object
for (let key in user) {
  console.log(`${key}: ${user[key]}`);
}

// Object.keys with forEach
Object.keys(user).forEach(key => {
  console.log(`${key}: ${user[key]}`);
});
```

Nested Objects

javascript

```
let company = {
  name: "Tech Corp",
  address: {
    street: "123 Main St",
    city: "New York",
    country: "USA"
  },
  employees: [
    { id: 1, name: "John" },
    { id: 2, name: "Jane" }
  ]
};

// Accessing nested properties
console.log(company.address.city); // New York
console.log(company.employees[0].name); // John
```

Array of Objects

```
javascript
```

```
let students = [
  { id: 1, name: "John", grade: 85, passed: true },
  { id: 2, name: "Jane", grade: 92, passed: true },
  { id: 3, name: "Bob", grade: 58, passed: false },
  { id: 4, name: "Alice", grade: 78, passed: true }
];

// Accessing data
console.log(students[0].name); // John
console.log(students[1].grade); // 92

// Filter passed students
let passedStudents = students.filter(student => student.passed);
console.log(passedStudents);

// Map names only
let names = students.map(student => student.name);
console.log(names); // ["John", "Jane", "Bob", "Alice"]

// Find student by id
let student = students.find(s => s.id === 2);
console.log(student); // { id: 2, name: "Jane", grade: 92, passed: true }

// Calculate average grade
let totalGrade = students.reduce((sum, student) => sum + student.grade, 0);
let average = totalGrade / students.length;
console.log(average); // 78.25

// Sort by grade (descending)
let sortedByGrade = students.sort((a, b) => b.grade - a.grade);
console.log(sortedByGrade);
```

📝 Practice Questions

Variables

1. What is the difference between `var`, `let`, and `const`?
2. What is hoisting? Give an example.
3. What is the Temporal Dead Zone?
4. Can you reassign a `const` object's property?

Data Types

1. What are primitive and non-primitive data types?

2. What is the difference between `null` and `undefined`?
3. What does `typeof null` return and why?
4. What is the difference between `==` and `===`?

Arrays

1. How do you add an element to the end of an array?
2. What is the difference between `push()` and `unshift()`?
3. What is the difference between `slice()` and `splice()`?
4. How do you remove the last element from an array?

Objects

1. What are two ways to access object properties?
2. How do you add a new property to an object?
3. How do you loop through object properties?
4. What is the difference between `Object.keys()` and `Object.values()`?

✅ Day 1 Checklist

- [ ] Understand declaration vs initialization
- [ ] Know difference between var, let, const
- [ ] Understand hoisting concept
- [ ] Know all primitive data types
- [ ] Master array creation and methods
- [ ] Master object creation and methods
- [ ] Practice array of objects
- [ ] Complete practice questions

Next: Day 2 - Operators, Conditionals, Loops