LAB ASSIGNMENT-3

1. Explain the differences between var, let, and const with respect to scope and hoisting.

⇨ **Difference between var, let, and const in JavaScript (Scope & Hoisting)**

In JavaScript, var, let, and const are used to declare variables, but they differ significantly in terms of **scope**, **hoisting**, and **reassignment**. Understanding these differences helps in writing safer and more predictable code.

---

1. Scope

var

- var has **function scope**.
- If declared outside a function, it becomes **globally scoped**.
- It **does not support block scope** (like if, for, {}).

Example:

```
if (true) {
  var x = 10;
}
console.log(x); // Output: 10
```

Here, x is accessible outside the if block because var ignores block scope.

---

let

- let has **block scope**.
- It is only accessible inside the block {} where it is declared.

Example:

```
if (true) {
  let y = 20;
```

```
}
console.log(y); // Error: y is not defined
```

---

const

- const also has **block scope**, just like let.
- Variables declared with const **cannot be reassigned**.

Example:

```
if (true) {
  const z = 30;
}
console.log(z); // Error: z is not defined
```

---

2. Hoisting

What is Hoisting?

Hoisting is JavaScript's default behavior of moving variable declarations to the **top of their scope** before code execution.

Hoisting with var

- var declarations are **hoisted and initialized with undefined**.
- Accessing before declaration does **not cause an error**.

Example:

```
console.log(a); // Output: undefined
var a = 5;
```

Hoisting with let

- let is **hoisted but not initialized**.
- Accessing it before declaration causes a **ReferenceError** due to the **Temporal Dead Zone (TDZ)**.

Example:

```
console.log(b); // ReferenceError
let b = 10;
```

Hoisting with const

- const is also **hoisted but not initialized**.
- Must be **declared and initialized at the same time**.

Example:

```
console.log(c); // ReferenceError
const c = 15;
```

---

3. Reassignment

Keyword	Reassignment Allowed
var	Yes
let	Yes
const	No

Example:

```
const pi = 3.14;
pi = 3.14159; // Error: Assignment to constant variable
```

---

4. Summary Table

Feature	var	let	const
Scope	Function	Block	Block
Hoisting	Yes (initialized as undefined)	Yes (TDZ)	Yes (TDZ)
Reassignment	Yes	Yes	No
Must initialize	No	No	Yes

Conclusion

- var is **older** and can cause unexpected bugs due to lack of block scope.
- let is preferred for variables that may change.
- const is best for values that should remain constant.
- Modern JavaScript recommends using **let and const instead of var**.

2. Describe the various Control Flow statements in JavaScript, specifically highlighting the difference between for, while, and do while loops.

⇨ **Control Flow Statements in JavaScript:-**

Control flow statements in JavaScript determine the **order in which the program executes statements**. By default, JavaScript executes code from top to bottom, but control flow statements allow developers to make decisions, repeat actions, and control execution based on conditions.

Types of Control Flow Statements in JavaScript:-

1. Conditional Statements (Decision Making)

These statements execute code based on conditions.

a) if statement

Executes code if the condition is true.

```
let age = 18;
if (age >= 18) {
  console.log("Eligible to vote");
}
```

b) if...else statement

Provides an alternative block when the condition is false.

```
let marks = 40;
if (marks >= 50) {
  console.log("Pass");
} else {
  console.log("Fail");
}
```

c) else if ladder

Used to check multiple conditions.

```
let grade = 85;
if (grade >= 90) {
  console.log("A");
} else if (grade >= 75) {
  console.log("B");
} else {
  console.log("C");
}
```

d) switch statement

Used when there are multiple fixed values.

```
let day = 2;
switch (day) {
  case 1: console.log("Monday"); break;
  case 2: console.log("Tuesday"); break;
  default: console.log("Invalid day");
}
```

---

2. Looping Statements (Iteration)

Looping statements are used to **execute a block of code repeatedly** as long as a condition is true.

*** Difference between for, while, and do...while Loops**

a) for Loop

- Used when the **number of iterations is known**.
- Initialization, condition, and update are written in one line.

Syntax:

```
for (initialization; condition; increment/decrement) {
  // code
}
```

Example:

```
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
```

---

b) while Loop

- Used when the **number of iterations is not known in advance**.
- Condition is checked **before** executing the loop body.

Syntax:

```
while (condition) {
  // code
}
```

Example:

```
let i = 1;
while (i <= 5) {
  console.log(i);
  i++;
}
```

---

c) do...while Loop

- Similar to while loop, but the condition is checked **after** executing the loop body.
- The loop executes **at least once**, even if the condition is false.

Syntax:

```
do {
  // code
} while (condition);
```

Example:

```
let i = 6;
do {
  console.log(i);
  i++;
} while (i <= 5);
```

Output: 6 (runs once even though condition is false)

---

3. Loop Control Statements

a) break

Stops the loop immediately.

```
for (let i = 1; i <= 5; i++) {
  if (i === 3) break;
  console.log(i);
}
```

b) continue

Skips the current iteration.

```
for (let i = 1; i <= 5; i++) {
  if (i === 3) continue;
  console.log(i);
}
```

Comparison Table: for vs while vs do...while

Feature	for Loop	while Loop	do...while Loop
Condition Check	Before loop	Before loop	After loop
Minimum Execution	0 times	0 times	At least 1 time
Best Used When	Iterations known	Iterations unknown	Must run once
Syntax Complexity	Compact	Simple	Slightly longer

Conclusion

- Control flow statements guide the execution path of a JavaScript program.
- for loop is best when the number of iterations is fixed.
- while loop is used when the loop depends on a condition.
- do...while loop guarantees at least one execution.
- Choosing the right loop improves readability and efficiency.

3. What is the Document Object Model (DOM)? Explain how to select elements and modify their content using innerText and innerHTML.

⇨ **Document Object Model (DOM):-**

The **Document Object Model (DOM)** is a programming interface that represents an **HTML document as a tree structure of objects**. In the DOM, every HTML element, attribute, and text becomes an **object (node)** that JavaScript can access and manipulate.

Using the DOM, JavaScript can:

- Access HTML elements
- Change content and styles
- Add or remove elements
- Respond to user events (click, input, etc.)

Structure of the DOM

An HTML document is represented as a **hierarchical tree**:

```
Document
└── html
    ├── head
    └── body
        ├── h1
        └── p
```

Each part of the HTML page becomes a **node** in the DOM tree.

Selecting Elements in the DOM

JavaScript provides several methods to select HTML elements.

1. getElementById()

Selects an element using its id.

Hello

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

---

2. getElementsByClassName()

Selects elements using class name (returns HTMLCollection).

Paragraph

let elements = document.getElementsByClassName("text");

---

3. getElementsByTagName()

Selects elements using tag name.

let paragraphs = document.getElementsByTagName("p");

---

4. querySelector() and querySelectorAll()

Selects elements using CSS selectors.

let firstPara = document.querySelector("p");

let allParas = document.querySelectorAll("p");

Modifying Content Using innerText and innerHTML

Once an element is selected, its content can be modified using **innerText** or **innerHTML**.

---

innerText

- Changes **only the text content** of an element.
- Ignores HTML tags.
- Safer for displaying plain text.

Example:

<h1 id="title">Welcome</h1>

document.getElementById("title").innerText = "Hello World";

Output:

Hello World

---

innerHTML

- Changes **HTML content including tags**.
- Can add elements like , <i>, etc.
- Powerful but must be used carefully.

Example:

document.getElementById("content").innerHTML = "Bold Text";

Output:

Bold Text (in bold)

Difference Between innerText and innerHTML

Feature	innerText	innerHTML
Handles HTML tags	No	Yes
Displays text only	Yes	No
Can insert elements	No	Yes
Security	Safer	Risky if misused

Example Program Using DOM

<!DOCTYPE html>

<html>

<body>

Original Text

```
<button onclick="changeText()">Click Me</button>

<script>
function changeText() {
  document.getElementById("demo").innerHTML = "<i>Text Changed!</i>";
}
</script>

</body>
</html>
```

Conclusion

- The DOM allows JavaScript to interact with HTML dynamically.
- Elements can be selected using ID, class, tag, or CSS selectors.
- innerText is used for changing text only.
- innerHTML is used for changing or inserting HTML content.
- Proper use of DOM manipulation makes web pages interactive and dynamic.