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

Ans :-

1. var

Scope

- **Function-scoped** (or global-scoped if declared outside a function).
- Ignores block scope (if, for, {} blocks).

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

Hoisting

- **Hoisted and initialized to undefined**.
- You can use the variable before its declaration (but it will be undefined).

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

---

2. let

Scope

- **Block-scoped**.
- Exists only within the block {} where it is declared.

```
if (true) {
  let y = 20;
}
console.log(y); // ReferenceError
```

Hoisting

- **Hoisted but not initialized**.
- Lives in the **Temporal Dead Zone (TDZ)** from the start of the block until the declaration is reached.
- Accessing it before declaration throws an error.

```
console.log(b); // ReferenceError

let b = 5;
```

---

3. const

Scope

- **Block-scoped**, same as let.

```
if (true) {
  const z = 30;
}

console.log(z); // ReferenceError
```

Hoisting

- **Hoisted but not initialized** (also subject to the **Temporal Dead Zone**).
- Must be initialized at the time of declaration.

```
const c; // SyntaxError
```

In Practice

- Avoid var in modern JavaScript.
- Use **const** by default.
- Use **let** only when you need to reassign a variable.

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

Ans:-

Control Flow Statements in JavaScript

Control flow statements determine the order in which statements are executed in a program. JavaScript provides several types of control flow mechanisms.

---

1. Conditional Statements

These execute code based on conditions.

a) if, else if, else

```
if (score > 90) {
  grade = "A";
} else if (score > 75) {
  grade = "B";
} else {
  grade = "C";
}
```

b) switch

Used when comparing one value against multiple cases.

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

2. Looping Statements

Loops execute a block of code repeatedly while a condition is true.

a) for Loop

- **Used when the number of iterations is known.**
- **Combines initialization, condition, and update in one place.**

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

b) while Loop

- **Used when the number of iterations is unknown.**
- **Condition is checked before each iteration.**

```
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

c) do-while Loop

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

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

Key Differences Between for, while, and do-while

Feature	for Loop	while Loop	do-while Loop
Condition Check	Before each iteration	Before each iteration	After each iteration
Guaranteed Run	No	No	Yes (at least once)
Best Use Case	Known number of iterations	Unknown number of times	Must run once minimum
Syntax Style	Compact	Simpler, flexible	Similar to while

---

3. Jump / Branching Statements

These alter the normal flow of execution.

- **break – exits a loop or switch**
- **continue – skips the current iteration**
- **return – exits a function and returns a value**

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

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

Ans:-

Document Object Model (DOM)

The Document Object Model (DOM) is a programming interface that represents an HTML document as a structured tree of objects.
Each HTML element becomes a node in this tree, allowing JavaScript to access, modify, add, or delete elements and their content dynamically.

In simple terms, the DOM lets JavaScript interact with and manipulate a web page after it has loaded.

Selecting Elements in the DOM

JavaScript provides several methods to select HTML elements.

1. getElementById()

Selects an element using its id.

const title = document.getElementById("heading");

---

2. getElementsByClassName()

Selects all elements with a given class name (returns an HTMLCollection).

const items = document.getElementsByClassName("list-item");

---

3. getElementsByTagName()

Selects elements by tag name.

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

---

4. querySelector()

Selects the first matching element using CSS selectors.

```
const firstItem = document.querySelector(".list-item");
```

---

5. querySelectorAll()

Selects all matching elements (returns a NodeList).

```
const allItems = document.querySelectorAll(".list-item");
```

Modifying Content Using innerText and innerHTML

1. innerText

- **Changes or returns only the visible text content of an element.**
- **Ignores HTML tags.**
- **Safer because it does not render HTML.**

```
const heading = document.getElementById("heading");
heading.innerText = "Welcome to JavaScript";
```

Output:

```
<h1>Welcome to JavaScript</h1>
```

---

2. innerHTML

- **Changes or returns HTML content, including tags.**
- **Can insert new elements.**
- **Less safe if used with untrusted input (risk of XSS).**

```
const container = document.getElementById("box");
container.innerHTML = "<strong>Hello World</strong>";
```

Output:

```
<div><strong>Hello World</strong></div>
```

Difference Between innerText and innerHTML

Feature	innerText	innerHTML
Content Type	Plain text only	Text + HTML tags
HTML Parsing	❌ No	✅ Yes
Security	Safer	Risky if misused
Use Case	Display text	Insert structured content

Conclusion

- **The DOM allows JavaScript to interact with web pages dynamically.**
- **Elements can be selected using methods like getElementById() and querySelector().**
- **innerText is used for plain text changes.**
- **innerHTML is used when inserting or modifying HTML structure.**