1. Global Scope

Example:

```
var a = 1;
let b = 2;
const c = 3;

function showValues() {
  console.log(a); // 1 - global var
  console.log(b); // 2 - global let
  console.log(c); // 3 - global const
}
showValues();
```

Explanation:

- `var`, `let`, and `const` declared outside functions are **globally scoped** and accessible anywhere in the script (unless shadowed).
- All three work inside functions unless redefined.

2. Function Scope

Example:

```
function test() {
  var x = 10;
  let y = 20;
  const z = 30;
}
console.log(x); // Error - not defined
console.log(y); // Error - not defined
console.log(z); // Error - not defined
```

Explanation:

- All variables inside a function (`var`, `let`, `const`) are **local to that function**.
- You cannot access them from outside.

3. Block Scope

Example:

```
if (true) {
  var a = "A";
  let b = "B";
  const c = "C";
}
console.log(a); // "A" - var leaks out
console.log(b); // Error - block scoped
console.log(c); // Error - block scoped
```

Explanation:

- `var` is **not block scoped**, so it's accessible outside the `if` block.
- `let` and `const` are **block scoped**, so they're only accessible inside the block.

4. Hoisting

Function Declaration Hoisting:

```
greet(); // Works

function greet() {
  console.log("Hello");
}
```

Variable Hoisting with `var`:

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

Variable Hoisting with `let`/`const`:

```
console.log(y); // Error - Cannot access 'y' before initialization
let y = 10;
```

Explanation:

- Function declarations are fully hoisted (definition + body).
- `var` is hoisted as `undefined`.
- `let` and `const` are hoisted but in the **Temporal Dead Zone (TDZ)** — not accessible until declared.

5. Shadowing

Example:

```
let a = "global";

function test() {
  let a = "local";
  console.log(a); // "local"
}
test();
console.log(a); // "global"
```

Explanation:

- The `a` inside the function **shadows** the global `a`.
- Changes inside the function don't affect the global variable.

6. Loop Scope with `var` vs `let`

Using `var` (bad):

```
for (var i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 100);
}
// Logs: 3, 3, 3
```

Using `let` (correct):

```
for (let i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 100);
}
// Logs: 0, 1, 2
```

Explanation:

- `var` is function scoped, so one `i` shared across iterations.
- `let` is block scoped, so new `i` for each iteration.

7. Closures and Scope

Example:

```
function createCounter() {
  let count = 0;
  return function () {
    count++;
    return count;
  };
}

const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2
```

Explanation:

- `count` is inside the `createCounter` function.
- The returned function **closes over** `count` and retains access even after `createCounter` finishes.

Summary by Variable Type

Type	Scope	Hoisted?	Block Scoped?	Re-declarable?	Temporal Dead Zone
`var`	Function	Yes (undefined)	No	Yes	No
`let`	Block	Yes (in TDZ)	Yes	No	Yes
`const`	Block	Yes (in TDZ)	Yes	No	Yes
`function`	Function or Block (ES6+)	Yes (with body)	Yes (in blocks)	Yes (in functions)	No