1. Features of ES6 (ECMAScript 2015)

- let and const for block-scoped variables
- Arrow functions (=>)
- Template literals (`Hello ${name}`)
- Classes and inheritance
- Default, Rest, and Spread parameters
- Destructuring (array & object)
- Promises for async operations
- Map and Set objects
- Modules (import/export)
- Enhanced object literals

✅ **2. Explain JavaScript let**

- let is used to declare variables that are **block-scoped**.
- Unlike var, let does not hoist to the top of the function/block.
- It **cannot be re-declared** in the same scope.

```
let count = 10;
if (true) {
  let count = 20;
  console.log(count); // 20
}
console.log(count); // 10
```

---

3. Differences Between var and let

Feature	var	let
Scope	Function-scoped	Block-scoped ({})
Hoisting	Yes (initialized as undefined)	Yes (but not accessible before declaration – Temporal Dead Zone)
Re-declaration	Allowed	Not allowed in same block
Global Object Bind	Yes	No

---

4. Explain JavaScript const

- Declares a **constant** (block-scoped) variable.
- Must be initialized at the time of declaration.
- Value **cannot be reassigned**, but **objects/arrays can be mutated**.

```
const PI = 3.14;

const arr = [1, 2];

arr.push(3);
```

---

5. ES6 Class Fundamentals

- Introduced **class-based syntax** for object-oriented JavaScript.
- Simplifies constructor function and prototype inheritance.

```
class Person {
 constructor(name) {
  this.name = name;
 }
 greet() {
  return `Hello, ${this.name}`;
 }
}
```

```
const p = new Person('Alice');
console.log(p.greet()); // Hello, Alice
```

---

6. ES6 Class Inheritance

- Use extends to create a subclass.
- Use super() to call the parent constructor.

```
class Animal {
 constructor(name) {
  this.name = name;
 }

 sound() {
  return "Some sound";
 }
}

class Dog extends Animal {
 sound() {
  return "Bark!";
 }
}

const d = new Dog("Tommy");
console.log(d.sound()); // Bark!
```

---

7. Define ES6 Arrow Functions

- Short syntax for writing functions:
  (args) => expression
- this is **lexically bound**, i.e., it does **not** get its own this.

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

console.log(add(2, 3)); // 5
```

Note: Avoid using arrow functions when you need dynamic this.

---

8. Identify Set() and Map()

Set

- Stores **unique values** (no duplicates).
- Insertion order is preserved.

```
const nums = new Set([1, 2, 3, 2]);

nums.add(4);

console.log(nums); // Set(4) {1, 2, 3, 4}
```

Map

- Stores **key-value pairs**.
- Keys can be of any type (object, function, etc).

```
const roles = new Map();

roles.set("admin", 1);

roles.set("user", 2);

console.log(roles.get("admin")); // 1
```