ES6 ReactJS hands\_on\_document :

1)List the features of ES6

**1. let and const**

* let provides **block scope** variable declaration (unlike var which is function scoped).
* const is used to declare **constants** (variables that can't be reassigned).

let age = 25;

const name = "John";

**2.Arrow Functions ()=>**

* Shorter syntax for function expressions.
* Does **not bind its own this**, making it useful in callbacks and React components.

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

**3)Template Literals**

* Easier string interpolation using backticks (`) instead of +.

const greeting = `Hello, ${name}!`;

**4) Default Parameters**

* Function parameters can have **default values**.

function greet(name = "Guest") {

console.log(`Hello, ${name}`);

}

**5) Destructuring**

* Extract values from arrays or objects into variables.

const user = { name: "Alice", age: 22 };

const { name, age } = user;

**6. Spread and Rest Operators ...**

* Spread: Expand arrays/objects.
* Rest: Combine multiple parameters into one.

const nums = [1, 2, 3];

const moreNums = [...nums, 4, 5];

function sum(...args) {

return args.reduce((a, b) => a + b);

}

2) Explain JavaScript let?

let is a keyword introduced in **ES6 (ECMAScript 2015)** to declare **block-scoped variables**.

Unlike var, which is function-scoped and hoisted with default undefined, let gives you **more predictable and safer behavior** in modern JavaScript.

**Key Features of let:**

1. **Block Scope**  
   Variables declared with let are **only accessible inside the block** ({}) where they are defined.
2. **No Hoisting like var**  
   let declarations are hoisted but stay in the **Temporal Dead Zone** until the line where they're declared — accessing them before that gives a ReferenceError.
3. **Can be Reassigned**  
   You **can update** the value of a let variable.
4. **Cannot be Redeclared** in the same scope  
   Redeclaring the same variable with let in the same scope will throw an error.
5. **Used for safer loop counters**  
   Especially useful in for loops because it avoids closure issues that happen with var.

Example:

{

let a = 10;

console.log(a); // 10

}

console.log(a); // ReferenceError: a is not defined

3) Identify the differences between var and let?

**Difference Between var and let**

| **Feature** | **var** | **let** |
| --- | --- | --- |
| **Scope** | Function-scoped | Block-scoped (inside {} only) |
| **Hoisting** | Hoisted and **initialized** with undefined | Hoisted but **not initialized** (in TDZ) |
| **Temporal Dead Zone (TDZ)** | Not affected by TDZ | Exists in TDZ until declared |
| **Re-declaration** | Allowed within the same scope | Not allowed in the same scope |
| **Global Object Property** | Creates a property on the global object (e.g., window.varName) | Does **not** create a global object property |
| **Use in Loops** | May cause issues with closures (e.g., for loops) | Safer in loops due to block scope |

4) Explain JavaScript const

const (short for **constant**) is a keyword introduced in **ES6 (ECMAScript 2015)** used to **declare variables whose value cannot be reassigned** after their initial assignment.

It is **block-scoped**, similar to let, but with stricter behavior.

**🔹 Key Features of const (Interview-Ready Summary):**

1. **Block Scope**  
   Like let, const is **scoped to the nearest {} block** (not function/global like var).
2. **Cannot be Reassigned**  
   Once a value is assigned to a const variable, **it cannot be changed or reassigned**.
3. **Does Not Mean Immutable**  
   For **objects and arrays**, the variable name can’t be reassigned, but the **contents can be modified**.
4. **Must be Initialized**  
   You **must assign a value** when declaring a const. Otherwise, you'll get a SyntaxError.
5. **Cannot be Redeclared**  
   A const variable **cannot be declared again in the same scope**.

**Example :**

const PI = 3.14159;

console.log(PI); // 3.14159

PI = 3.14; // TypeError: Assignment to constant variable

5)Explain ES6 class fundamentals

ES6 (ECMAScript 2015) introduced a new syntax to define **classes** in JavaScript. It provides a cleaner and more understandable way to implement **object-oriented programming** concepts like **encapsulation, inheritance, and method reuse**. Although JavaScript is prototype-based, the class syntax is just syntactic sugar over the existing prototype-based system.

**Key Features of ES6 Classes**

1. **Simplified Syntax**:  
   ES6 provides a clean and concise syntax to create classes and objects, similar to other object-oriented languages like Java or C++.
2. **Constructor Method**:  
   Every ES6 class has a constructor() method used to initialize object properties when an object is created.
3. **Inheritance with extends**:  
   Classes can inherit from other classes using the extends keyword, promoting code reuse through hierarchical relationships.
4. **Use of super()**:  
   The super() function is used to call the constructor or methods of a parent class from a subclass.
5. **Encapsulation**:  
   Classes allow grouping related data and methods together, improving code structure and encapsulating functionality.

6)Explain ES6 class inheritance?

Inheritance is a fundamental concept of object-oriented programming where a class (called **child or subclass**) derives properties and behaviors (methods) from another class (called **parent or superclass**).

**ES6 Class Inheritance**

In **ES6 (ECMAScript 2015)**, class inheritance allows you to:

* Reuse code from an existing class.
* Create a hierarchical structure of classes.
* Add or override functionality in the derived class.

**🔹 Key Concepts in ES6 Inheritance**

1. **extends keyword**:  
   Used to create a child class that inherits from a parent class.
2. **super() function**:
   * Used to call the constructor of the parent class from the child class.
   * Must be called before using this in the child constructor.
3. **Method Overriding**:  
   A child class can define a method with the same name as one in the parent to override it.
4. **Code Reusability**:  
   Common functionality is defined in the parent class, and the child class can extend or modify it.
5. **Prototype Chain**:  
   Under the hood, ES6 classes are syntactic sugar over JavaScript’s prototype-based inheritance model.

Example:

class Animal {

constructor(name) {

this.name = name;

}

speak() {

console.log(`${this.name} makes a sound.`);

}

}

class Dog extends Animal {

constructor(name, breed) {

super(name); // Call the parent class constructor

this.breed = breed;

}

speak() {

console.log(`${this.name} barks.`);

}

}

const myDog = new Dog("Buddy", "Labrador");

myDog.speak(); // Output: Buddy barks.

7)Define ES6 arrow functions

**Arrow functions** are a concise way to write functions introduced in **ES6 (ECMAScript 2015)**. They are especially useful for writing **shorter function expressions** and also handle the this keyword differently than regular functions.

**Key Features of Arrow Functions**

1. **Shorter syntax** than traditional function expressions.
2. **Implicit return** when using a single expression (no {} block needed).
3. **Lexical this binding** – arrow functions don’t have their own this; they use this from their enclosing context.
4. **No arguments object** – they don’t have a local arguments object.
5. **Cannot be used as constructors** – you can't use new with them.

**Syntax:**

// Traditional function

function add(a, b) {

return a + b;

}

// Arrow function (equivalent)

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

8) Identify set(), map()

**Set() in ES6**

A **Set** is a built-in JavaScript object that allows you to store **unique values** of any type — primitive or object.

**Key Features of Set:**

1. **No duplicates allowed** – Each value in a Set must be unique.
2. **Maintains insertion order** – Elements are iterated in the order of insertion.
3. **Can store any type** – Numbers, strings, objects, etc.
4. **Has useful methods** – Like add(), delete(), has(), clear().
5. **Iterable** – Can be looped using for...of.

**Set – Example (Remove Duplicates)**

const numbers = [1, 2, 2, 3, 4, 4, 5];

const uniqueNumbers = new Set(numbers);

console.log(uniqueNumbers); // Output: Set(5) { 1, 2, 3, 4, 5 }

**Map() in ES6**

A **Map** is a collection of **key-value pairs** where the keys can be **of any data type**.

**Key Features of Map:**

1. **Stores data in key-value pairs**.
2. **Preserves insertion order**.
3. **Keys can be of any type** – Unlike objects where keys are always strings/symbols.
4. **Useful methods** – set(), get(), has(), delete(), clear().
5. **Better performance** than objects for frequently added/removed key-value pairs.

**Map – Example (Store Student Marks)**

const studentMarks = new Map();

studentMarks.set("Rahul", 85);

studentMarks.set("Anjali", 92);

studentMarks.set("Ravi", 78);

console.log(studentMarks.get("Anjali")); // Output: 92