**React- Day -1 Notes:**

**ES5 vs ES6:**

**The First Main Revision ES5 (2009) The Second Revision ES6 (2015)**

**ES5 vs Es6 :**

ECMAScript stands for European Computer Manufacturers Association

**ES5: Some features**

1. Var Keyword

2. Function Declaration

3. No Arrow Functions

4. No Classes

5. No Destructuring

6. No Template Literals

7. Callbacks for Asynchronous Operations

**ES6: Some Additional features**

1. Let and Const

2. Arrow Functions

3. Classes

4. Destructuring

5. Template Literals

6. Promises for Asynchronous Operations

7. Default Parameters

**Eg:**

Class FSD {

constructor(name) {

this.name = name;

}

sayHello() {

console.log(Hello, I am ${this.name});

}

}

const [a, b] = [1, 2];

const name = 'John';

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

const fetchData = () => new Promise(resolve => setTimeout(() => resolve('Data'), 1000));

fetchData().then(data => console.log(data));

**Scoping - var vs let vs const:**

The three main variable declaration **keywords** are var, let, and const. Each has different scoping rules, affecting how and where the variable is accessible

var:

* Function Scope:
  + Variables declared with var are **function-scoped**, meaning their scope is limited to the function in which they are defined.
* Hoisting:
  + var declarations are hoisted to the **top of their scope**. The variable is moved to the top during the compilation phase, allowing you to access it before it is declared.

### let:

* Block Scope:
  + Variables declared with let are **block-scoped.** They are only accessible within the block (statements enclosed in curly braces) in which they are defined.
* Hoisting:
  + let declarations are hoisted to the **top of their block scope but are not initialized**. They remain in the "temporal dead zone" until the actual declaration is encountered.

### const:

* Block Scope:
  + Variables declared with const are **block-scoped** similar to let.
* Constant Value:
  + const is used for constant values. **Once assigned**, the value cannot be reassigned.
* Hoisting:

Like let, const declarations are hoisted to the **top of their block scope but are not initialized** until the actual declaration is encountered.

In modern JavaScript development, it is generally **recommended to use let and const over var due to the more predictable scoping behavior** and the prevention of certain issues associated with var, such as unintended global scope and **hoisting complexities**.

**arrow functions:**

(ES6), provides a concise syntax for writing anonymous functions in JavaScript. They offer a more compact and expressive way to define functions, **especially for short, one-line operations.**

**use of this keyword(lexical scoping):**

the this **keyword** is a special variable that is created for every function execution. The value of **this depends on how the function is called**. **Lexical scoping**, introduced by arrow functions, affects the behavior of this.

**template literals:**

(ES6), provides a convenient and **flexible way to create strings** in JavaScript. They allow for the embedding of expressions and **multiline strings** directly within the string literal. Template literals are enclosed by **backticks (``)** instead of single or double quotes.${}

**spread & rest prarameter:**

Spread and rest parameters are features introduced in ECMAScript 2015 (ES6)

**Spread Parameter:**

The spread syntax (...) allows an iterable (like an array or string) to be expanded **into individual elements or characters**. It is used in **function calls**, array literals, and object literals.

* **Copy** arrays and objects.
* **Concatenate** arrays.
* Pass elements of an array as individual function arguments.

**Rest Parameter:**

The rest parameter also uses the spread syntax (...), but it is used in **function parameters** to collect multiple arguments into a single array.

Capture variable numbers of **function arguments** into an array.

Gather **remaining parameters** into an array.

**array & object:**

**Array:**

Arrays are **ordered**, numerically indexed collections of values. They are used to store lists of items and provide methods for manipulating those items. Array **indices start from 0**.

const fruits = ['apple', 'orange', 'banana'];

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

**Object:**

Objects are **unordered collections of key-value pairs.** They are used to represent entities and provide a way to organize and structure data. Object keys are strings or symbols.

const person = {

name: 'John',

age: 30,

city: 'New York'

};

**Destructure:**

Destructuring assignment is a feature in JavaScript that allows you to **extract values from arrays**

const numbers = [1, 2, 3];

const [a, b, c] = numbers;

**property shorthand:**

that allows you to create objects with concise syntax by automatically **using variable names as property names**.

This shorthand is particularly useful when you want to create an object using variables with the **same name as the desired property names.**

**module import & export:**

It supports **modular programming in JavaScript.** They provide a way to **organize code into separate files or modules**, making it more maintainable, scalable, and easier to collaborate on **larger projects**.

export module a

export module b

import { add, multiply } from './moduleA';

import greet from './moduleB';

**Class in Javascript:**

It supports **object-oriented programming concepts**. While JavaScript still follows a prototype-based inheritance model, classes provide a more familiar and syntactically convenient way **to create and extend objects.**