Camel case = used for variable,function,object = Example– myFunction
Pascal case = used for constructor and class = example= Class

The following table outlines the differences between the `var`, `let`, and `const` keywords in JavaScript:

Feature	var	let	const
Hoisted?	Yes (undefined)	No (Temporal Dead Zone)	No (Temporal Dead Zone)
Reassignment?	Yes	Yes	No
Redeclaration?	Yes	No	No
Scope	Function-scoped	Block-scoped	Block-scoped

Callback Functions

A function passed as an argument to another function.

```
function fetchData(callback) {
    setTimeout(() => {
        callback("Data received");
    }, 2000);
}

fetchData((message) => console.log(message));
```

Async,await,promise

```
function userLogin(username, password) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            if (username === "saurabh" && password === "1234") {
                resolve("Login successful! Welcome, Saurabh.");
            } else {
                reject("Invalid username or password.");
            }
        }, 3000);
    });
}

// Calling the function
userLogin("saurabh", "1234")
    .then(message => console.log(message))
    .catch(error => console.log(error));
```

Here's a revised version of the provided JavaScript code, along with explanations:

```
□function userLogin(username, password) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (username === "saurabh" && password === "1234") {
        resolve("Login successful! Welcome, Saurabh.");
      } else {
        reject("Invalid username or password.");
      }
    }, 3000);
  });
}


// Usage
userLogin("saurabh", "1234")
  .then(message => console.log(message))
  .catch(error => console.log(error));
```

□**Explanation:**

- **`userLogin` Function:** This function simulates a user login process. It takes a `username` and `password` as input.
- **Promise:** It returns a Promise, which represents the eventual result of the login attempt.
- **`setTimeout`:** The `setTimeout` function simulates a delay of 3 seconds (3000 milliseconds), mimicking a real login process that might take time.
- **Resolve and Reject:**
    - If the provided `username` and `password` are correct ("saurabh" and "1234" in this example), the Promise is resolved with a success message.
    - If the credentials are incorrect, the Promise is rejected with an error message.
- **`.then` and `.catch`:**
    - The `.then` method is used to handle the successful login scenario. It takes a callback function that receives the success message and logs it to the console.
    - The `.catch` method is used to handle the failed login scenario. It takes a callback function that receives the error message and logs it to the console.

1. Concatenation (Do strings ko jodna)

```
let str1 = "Hello";
let str2 = "World";
let result = str1 + " " + str2; // "Hello World"
console.log(result);

// Template literals (Modern way)
let result2 = `${str1} ${str2}`;
console.log(result2);
```

2. Find a Letter or Word in a String (Ek character ya word dhundhna)

Single Character or Word

```
let text = "JavaScript Programming";
console.log(text.includes("Java"));  // true
console.log(text.indexOf("P"));      // 11 (First occurrence)
console.log(text.indexOf("X"));      // -1 (Not found)
```

Multiple Occurrences Find Karna

```
let positions = [];
let pos = text.indexOf("a");

while (pos !== -1) {
   positions.push(pos);
   pos = text.indexOf("a", pos + 1); // Next occurrence
}
console.log(positions); // [1, 3, 18]
```

3. Replace a Character or Word in a String

```
let sentence = "I love JavaScript";
let newSentence = sentence.replace("JavaScript", "Python");
console.log(newSentence); // "I love Python"

// Multiple replacements using regex
let newSentence2 = sentence.replace(/o/g, "0");
console.log(newSentence2); // "I l0ve JavaScript"
```

4. Traverse a String (Ek ek character access karna)

```
let str = "Hello";
for (let char of str) {
   console.log(char);
}
```

5. Join Strings using a Delimiter

```
let words = ["Hello", "World", "JavaScript"];
let joinedStr = words.join("-");
console.log(joinedStr); // "Hello-World-JavaScript"
// filter , split join
let messyString = "   Hello    World   JavaScript   ";
let cleanString = messyString.split(" ").filter(word => word !== "").join(" ");
console.log(cleanString);
```

split(" ")	Convert string to array	["", "", "Hello", "", "", "World", "", "JavaScript", ""]
filter(word => word !== "")	Remove empty strings	["Hello", "World", "JavaScript"]
join(" ")	Convert array back to string	"Hello World JavaScript"

Arrays in JavaScript

(a) Simple Array Creation

```
let arr = [10, 20, 30, 40, 50];
console.log(arr);
// Output: [10, 20, 30, 40, 50]
```

(b) Using Spread Operator (...) khasiyat: Yeh original array modify nahi karta.

```
let arr =[1,2,3];
let arr2= [...arr,5,6];
console.log(arr2)
```

(c) fill() Method se Array Create Karna

```
let arr = new Array(5);
console.log(arr); //properly initialized nahi hain.
fill(0)
let arr = new Array(4).fill(0);
console.log(arr);
fill(value,start,end)end is excluded
let arr =[1,2,3,4,5,6,7,8,9];
console.log(arr.fill(0,2,5)); // [1,2,0,0,0,5,6,7,8,9]
```

new Array(5).fill() → Ek properly initialized array banata hai.
.map(() => Math.random()) → Har element ke liye ek random number generate karta hai.

fill(value)	Poore array ko ek hi value se bhare	new Array(5).fill(0)	[0, 0, 0, 0, 0]
fill(value, start, end)	Sirf specific indexes fill kare	arr.fill(0, 2, 6)	[1, 2, 0, 0, 0, 0, 7, 8, 9]
fill(null).map(...)	Unique objects banane ke liye	new Array(3).fill(null).map(() => ({}))	[{}, {}, {}]
fill().map(...)	Random values array banane ke liye	new Array(5).fill().map(() => Math.random())	[0.5, 0.7, 0.2, 0.8, 0.3]

Shalllow and deep

Shallow -

```
let original = [1, 2, [3, 4]];
let shallowCopy = [...original];
console.log(shallowCopy);//[ 1, 2, [ 3, 4 ] ]
shallowCopy[2][0] = 99;//first[first_array_index][second_array_index][third_array_index]
console.log(shallowCopy);//[ 1, 2, [ 99, 4 ] ]
```

Deep

```
let deepCopy = JSON.parse(JSON.stringify(original));
deepCopy[2][0] = 3;
console.log(original); // [1, 2, [99, 4]]
console.log(deepCopy); // [1, 2, [3, 4]]
```

Read element of array

```
let arr = [10, 20, 30, 40];

arr.forEach((num) => {
   if (num === 30) {
      // yahan break nahi lag sakta
   }
   console.log(num);
});
```

Update, Delete, Find

1️⃣Update an Array

```
let arr = [1,2,4];
arr[2]=99;
console.log(arr);
```

2️⃣Delete an Element

```
let arr = [10, 20, 30];
arr.splice(1, 1); // 1st index se ek element hata diya
console.log(arr); // [10, 30]
```

3️⃣ Find an Element

let arr = [10, 20, 30];
let found = arr.find(num => num > 15);
console.log(found); // 20 (Pehla match milega bas) // filter use krnege to pura array dekhega

`filter()`, `map()`, `reduce()`, `join()`

1️⃣ `filter()` (Condition based filter karna)

let arr = [10, 20, 30, 40, 50];
let filtered = arr.filter(num => num > 25);
console.log(filtered); // [30, 40, 50]

2️⃣ `map()` (Modify karke naya array banata hai) operation perform krna

let arr = [10, 20, 30];
let squared = arr.map(num => num * num);
console.log(squared); // [100, 400, 900]

3️⃣ `reduce()` (Ek hi value me compress karna) multiplication div plus minus

Let arr = [1,2,3,4];
Let sum = arr.reduce((acc,num)=>arr+num,0);
console.log(sum);// 10

2️⃣ Objects in JavaScript

(A) Create Object with Methods

```
let person = {
   name: "John",
   age: 25,
   greet: function() {
      console.log(`Hello, my name is ${this.name}`);
   }
};

person.greet(); // Hello, my name is John
console.log(person['age']);
```

(C) Traversing an Object

```
for (let key in person) {
   console.log(key, person[key]);
}name John
age 25
greet [Function: greet]
```

(D) Updating Object with Spread Operator

```
let obj = {
  Name: "saurabh",
  Age: 25
};

let updt = { ...obj, Age: 24 };

console.log(updt);
```

For delete

```
delete person.age;
console.log(person);
```

Call:

1. Function Borrowing

Kabhi kabhi ek object ka function doosre object me use karna hota hai.

```
let person1 = {
  name: "Saurabh",
  sayName: function () {
    console.log(this.name);
  }
};

let person2 = { name: "Punya" };

// Function borrowing using call()
person1.sayName.call(person2);
// Output: Punya
```

Bind:

```
let user = {
  name: "Saurabh",
  greet: function () {
    console.log(`Hello, ${this.name}`);
  }
};

// Using bind to ensure 'this' refers to user
setTimeout(user.greet.bind(user), 1000);
// Output after 1 second: Hello, Saurabh
```

🔹 **Bina `bind()` ke, `this window` (or `undefined` in strict mode) ho jata.**

Apply:

🔹 Yeh `apply()` ka ek common use case hai, kyunki `Math.max()` arguments leta hai, array nahi.

```
let numbers = [10, 5, 20, 8];

let maxNum = Math.max.apply(null, numbers);
console.log(maxNum); // Output: 20
```

3️⃣ Map in JavaScript

(A) Create a Map

```
let map = new Map([
   ["name", "Saurabh"],
   ["age", 25]
]);
```

2️⃣ Insert/Update Values

```
map.set("city", "Varanasi");
console.log(map);
```

3️⃣ Get Value Using Key map.get()

4️⃣ Check If a Key Exists map.has()

5️⃣ Delete a Key map.delete()

6️⃣ Traversing a Map

```
for(let [key,value] of map){
   console.log(`${key}:${value}`);
   }
```

(E) Sorting a Map (Ascending & Descending)

```
let map = new Map([[2, "B"], [1, "A"], [3, "C"]]);

let sortedAsc = new Map([...map.entries()].sort((a, b) => a[0] - b[0]));
console.log(sortedAsc);
```

1️⃣ Understanding `Map` Creation

Map automatically keys ko sorted order me store nahi karta!
Agar `console.log(map);` karein, toh output insertion order me aayega:
Converting `Map` to Array for Sorting`.entries()` method `Map` ke key-value pairs ka iterator return karta hai.
🔹 **`[...map.entries()]` likhne se yeh iterator ek array me convert ho jata hai.`.sort()` ek array sorting method hai jo compare function leta hai.**

4️⃣ Set in JavaScript

let set = new Set();
set.add(10);
set.add(20);
set.add(10); // Duplicate ignore ho jayega
console.log(set);

(B) Check if a Value Exists

console.log(set.has(20)); // true

(C) Traverse a Set

```
for (let value of set) {
   console.log(value);
}
```

(D) Delete a Value

set.delete(10);
console.log(set);