lec 3 summary:

☕ JavaScript Concepts Organization

The provided information covers key concepts in JavaScript, primarily focusing on **Objects**, **Arrays**, **Loops**, **Callback Functions**, and **ES6 Features** (Spread and Rest Operators).

---

1. ⚙️ JavaScript Objects

JavaScript objects are categorized into two main types:

- **Language Objects:** Built-in objects provided by the JavaScript environment (e.g., `String`, `Array`, `Math`, `Date`).
- **User-Defined Objects:** Objects created by the developer (e.g., Anonymous Objects, Classes).

---

2. 🔢 Arrays

Arrays are ordered lists used to store multiple values.

Array Creation

- Using the `Array` constructor: `let grades = new Array(1, 2, 3);`
- Using the array literal (shortcut): `let numbers = [1, 2, 3, 4];`

Array Properties and Methods

- **Common Methods:** `push`, `pop`, `unshift`, `shift`, `splice` (for search/manipulation), `join`, `indexOf`, `includes`.
- **Sparse Arrays (Worse Case):** Arrays with empty slots or large gaps between elements (e.g., `numbers[7] = 7`).
- **Checking Constructor Name:** Can be used to identify the object type (e.g., `grades.constructor.name`).

---

3. 🔁 For Loops in JavaScript

JavaScript provides several methods for iterating over arrays and objects:

- `for` **loop:** The standard loop construct for iteration based on an index:JavaScript

  ```
  // for(let i=0; i < numbers.length; i++) { ... }
  ```

- `for...in` **:** Iterates over the **enumerable properties/indices** of an object or array. **Special case** for arrays as it iterates over indices (including sparse ones).JavaScript

  ```
  // for(let index in numbers) { ... }
  // if(1 in numbers) {} // Checks if index 1 exists
  ```

- `for...of` **:** Iterates over the **values** of an iterable object (like an array).JavaScript

  ```
  // for(let item of numbers) { ... }
  ```

- **Built-in Loop (** `forEach` **):** A high-order function that executes a provided function once for each array element.

---

4. 🔗 Array Methods Using Callback Functions

Several built-in array methods utilize **callback functions** to perform operations on each element.

Iteration and Modification

- `forEach` **:** Executes a function for each element. Can be used to modify the **original array**:JavaScript

  ```
  // numbers.forEach((number, index, array) => array[index] *= 2);
  ```

  - *Task:* Multiply each item by 2:JavaScript

    ```
    // let result = [];
    // numbers.forEach((number) => { result.push(number * 2); });
    ```

- `filter` **(Custom Generic Implementation):** Creates a new array with all elements that pass the test implemented by the provided function.JavaScript

  ```
  // const filter = (array, condition) => { /* implementation */ }
  // filter([1, 2, 3, 4], function(number){ return number > 3 }); // Anonymous function
  // filter([3, 5, 1, 6, 8, 4], number => number < 3); // Arrow function
  ```

Array Transformation

- `map`: Creates a **new array** by calling a function on every element in the original array.
  - *Task:* Multiply each item by 2: `let numbers_2 = numbers.map(number => number * 2);`
  - *Task:* Calculate square root: `numbers.map(Math.sqrt);` (Recommended syntax)

Array Aggregation

- `reduce`: Executes a reducer function (that you provide) on each element of the array, resulting in a single output value.
  - *Task:* Calculate the summation:JavaScript

```
// numbers.reduce((prev, current) => prev += current, 0);
// Step 1: (0, 1) => 1
// Step 2: (1, 2) => 3
// Step 3: (3, 3) => 6
```

Sorting

- `sort`: Sorts the elements of an array *in place* and returns the sorted array. By default, it sorts alphabetically.
  - **Custom Sorting:** Requires a comparison function:JavaScript

```
// numbers.sort((a, b) => {
//     if (a > b) return 1;
//     else if (a < b) return -1;
//     else return 0;
// });
// Simplified for numbers: numbers.sort((a, b) => a - b);
```

---

5. 🛠️ ES6 Features: Spread and Rest Operators

The `...` operator has two uses depending on the context: **Spread** and **Rest**.

Spread Operator (`...`)

Expands an iterable (like an array) into its individual elements.

1. **Calling Functions:** Passes array elements as separate arguments:JavaScript

```
// Math.min(...numbers)
```

2. **Copying Arrays:** Creates a shallow copy:JavaScript

```
// let newNumbers = [...numbers];
```

3. **Concatenation/Combining:** Merges multiple arrays and/or elements:JavaScript

```
// let studentsGrades = [...numbers, ...grades, 30, 50, ...newNumbers];
```

Rest Parameters (`...`)

Collects an **indefinite number of arguments** passed to a function into an **actual array**.

JavaScript

```
// const sum = function(...input) {
//     console.log(input); // input is now a proper Array
//     return input.reduce((prev, current) => prev += current, 0);
// }
// sum(1, 2);
// sum(1, 2, 5);
```

---

6. 🧪 General Testing and Utility Functions

Type Coercion

JavaScript performs type coercion when using the `+` operator with different types:

- `numbers + 1;` (Array to String concatenation)
- `numbers + "eman";` (Array to String concatenation)
- `names + 1` (Array to String concatenation)

Falsy Check

A simple check for falsy values (`undefined` , `null` , `0` , `false` , `NaN` , `""`):

JavaScript

```
// let number; // 'number' is undefined, which is falsy
// if(!number) console.log("number is empty");
```

Utility Function Example (Case Switching)

A function to switch the case of letters in a string:

JavaScript

```
// const switchCase = (input) => {
//     // ... implementation using split, iteration, case conversion, and join
// }
```