Class 03

MERN STACK DEVELOPMENT

MODULE B

Exploring the Pinnacle of Full Stack Development with MERN

Today’s Topics

- Ternary Operator
- Spread Operator
- Rest Operator
- Arrow Function
- Array methods (loops)
    - ForEach
    - map

Ternary Operators

The ternary operator (**? :**) is a concise way to write conditional statements in JavaScript. It provides a shorthand for simple **if-else conditions**, making code more readable and compact.

if statement

syntax

```
const score = 75;
let result;

if (score >= 70) {
            result = "Pass";
}
else {
            result = "Fail";
}
```

Ternary Operator

syntax

```
const score = 75;
const result = score >= 70 ? "Pass" : "Fail";
```

Spread Operators

The Spread Operator, introduced in ECMAScript 6 (ES6), is a powerful tool for array and object manipulation in JavaScript. Represented by three dots (...), this operator facilitates concise and efficient operations, allowing the expansion of iterable elements in various contexts.

Array spread

syntax

```
const numbers = [1, 2, 3];

const newNumbers = [...numbers, 4, 5];
```

Object spread

syntax

```
const person = { name: 'Alice', age: 25 };

const updatedPerson = {
            ...person,
            age: 26,
            job: 'Developer‘
            };
```

Rest Operators

The Rest Operator, introduced in ECMAScript 6 (ES6), is a crucial tool in JavaScript for handling variable numbers of function arguments or array/object elements. Marked by three dots (...), the Rest Operator enhances code flexibility, allowing functions to accept a variable number of parameters and efficiently capture remaining elements in arrays or objects. Its introduction contributes to more versatile and expressive JavaScript programming, particularly in scenarios requiring the management of dynamic and variable-length inputs.

syntax

```
// In Function Parameters
const exampleFunction = (param_1, param_2, ...rest) => {
…
};

// In Array Destructuring
const [index_0, index_1, ...rest_of_array] = some_array;

// In Function Parameters
const { first_name, last_name, ...rest_of_object } = some_object;
```

Arrow function

An arrow function in JavaScript is a concise way to define a function using a streamlined syntax introduced in ECMAScript 6 (ES6). Unlike traditional function expressions, arrow functions provide a shorter and more readable syntax, making code more expressive.

Traditional Function Expression

syntax

```
const add = function (a, b) {
        return a + b;
};
```

Arrow Function

syntax

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

forEach – Array Method

The **forEach** method is a built-in function in JavaScript designed specifically for iterating over elements within an array. Its primary purpose is to execute a provided callback function once for each element present in the array, in ascending order. This functionality makes it a powerful and expressive tool for performing actions on every item in an array without the need for traditional for loops.

The syntax of the forEach method is straightforward. It takes a callback function as an argument, which will be executed once for each element in the array. The callback function typically accepts three parameters**: the current element, the index of the current element,** and **the array itself.**

syntax

```
array.forEach( (element, index, array) => {
                // Code to be executed for each element
                console.log(element);

 });
```

map – Array Method

The **map** method is a fundamental array method in JavaScript designed for transforming each element of an array based on a provided callback function. It creates a new array with the results of applying the callback function to each element of the original array, without modifying the original array.

The syntax of the **map** method is straightforward. It takes a callback function as its argument, which is applied to each element of the array. The result of the callback function becomes the corresponding element in the new array.

syntax

```
// Using the map method to modify the 'name' property in each object
const modifiedArray = originalArray.map((obj)=> {

Creating a new object with the 'name' property capitalized
return { ...obj, name: obj.name.toUpperCase() };

});
```

That's it for today!

Keep coding, stay curious, and enjoy your JavaScript journey!