Lesson Content

OT Reading Material

VI Video Course Lesson 12

JavaScript: Advance Functions

Introduction to Advanced Functions

- **Objective:** In this lesson, we will explore advanced function concepts in JavaScript: callbacks, arrow functions, higher-order functions, and timing functions. By the end of this lesson, you'll be able to apply these concepts to write cleaner, more flexible code.

---

1. Callbacks: Passing Functions as Arguments

- **Objective:** In this section, we will explore callbacks—functions that are passed as arguments to other functions. They allow us to perform actions at specific points in our code and are particularly useful for asynchronous operations.

Step 1: Define the `processData` function and a simple callback

- **Explanation:** Callbacks are functions passed as arguments to other functions. Let's define a `processData` function that processes a number and takes a callback function to handle the result.

```
function processData(number, callback) {
  let result = number + 5;  // Process the number
  callback(result);  // Call the callback function with the
}
```

Step 2: Create a simple callback function to log the result

- **Explanation:** Next, we'll create a callback function `logResult` that logs the result of the processed number.

```
function logResult(result) {
  console.log('Processed result: ' + result);
}
```

Step 3: Call the `processData` function with the callback

- **Explanation:** Now, let's call the `processData` function, passing `logResult` as the callback function.

```
processData(10, logResult);  // Output: Processed result: 15
```

2. Arrow Functions: Writing Cleaner Functions

- **Objective:** Arrow functions are a more concise way to write functions, especially useful for simple functions and callbacks.

Step 1: Write a traditional function to double a number

- **Explanation:** Let's start by writing a traditional function to double a number.

```
function double(number) {
  return number * 2;
}
```

Step 2: Convert to an arrow function

- **Explanation:** Arrow functions provide a shorter syntax. Let's convert the function into an arrow function.

```
const double = (number) => number * 2;
```

Step 3: Test the arrow function

- **Explanation:** Test the arrow function by calling it and logging the result.

```
console.log(double(4));  // Output: 8
```

3. Higher-Order Functions: Functions That Take Functions

- **Objective:** Higher-order functions are functions that take one or more functions as arguments or return functions. We'll explore this concept through examples like `map`, `filter`, and `forEach`.

Step 1: Using map to create a new array where each element is doubled

- **Explanation:** `map` is used to transform an array into another array with modified values.

```
const numbers = [1, 2, 3, 4];
const doubled = numbers.map((num) => num * 2);
console.log(doubled);  // Output: [2, 4, 6, 8]
```

Step 2: Using `filter` to return only even numbers from an array

- **Explanation:** `filter` creates a new array with elements that satisfy a given condition. Here, we'll filter out only the even numbers from an array.

```
const numbers = [1, 2, 3, 4];
const evens = numbers.filter((num) => num % 2 === 0);
console.log(evens);  // Output: [2, 4]
```

Step 3: Using `forEach` to log each element of the array

- **Explanation:** `forEach` performs a given action on each element of an array.

```
const numbers = [1, 2, 3, 4];
numbers.forEach((num) => console.log(num));  // Output: 1 2
```

4. Timing Functions: Delaying or Repeating Actions

- **Objective:** Timing functions like `setTimeout` and `setInterval` are used to delay or repeat actions after a specified amount of time.

Step 1: Using `setTimeout` to delay an action

- **Explanation:** `setTimeout` delays the execution of a function by a specified amount of time.

```
setTimeout(() => {
  console.log('This message is shown after 3 seconds');
}, 3000);
```

- **Explanation:** `setTimeout` takes two arguments: the function to execute and the delay in milliseconds (3000ms = 3 seconds).

Step 2: Using `setInterval` to repeat an action

- **Explanation:** `setInterval` repeatedly calls a function at the specified interval.

```
setInterval(() => {
  console.log('This message is shown every 2 seconds');
}, 2000);
```

- **Explanation:** `setInterval` works similarly to `setTimeout`, but it repeats the function at regular intervals (2000ms = 2 seconds).

Exercises

Instructions for Students:

I.
Exercise 1:
Write a function `applyDiscount` that takes an array of prices and a callback function. The callback should apply a discount to each price. After the discount is applied, log the final prices.

II.
Exercise 2:
Write a function `squareAndPrint` that takes a number and uses an arrow function to square the number and log the result.

III.
Exercise 3:
Use `map` to take an array of strings and return an array where each string is reversed.

IV.
Exercise 4:
Use `filter` to return only the odd numbers from an array of integers.

V.

Exercise 5:

Write a function that uses `setTimeout` to log "Task completed!" after a 5-second delay. Additionally, set up a `setInterval` to log "Still waiting..." every 2 seconds.

Mark Lesson As Complete

Cape - Mark Complete