JavaScript Higher Order Functions Review

Callback Functions and the forEach Method

- **Definition**: In JavaScript, a callback function is a function that is passed as an argument to another function and is executed after the main function has finished its execution.
- **forEach() Method**: This method is used to iterate over each element in an array and perform an operation on each element. The callback function in forEach can take up to three arguments: the current element, the index of the current element, and the array that forEach was called upon.

Example Code

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


// Result: 2 4 6 8 10
numbers.forEach((number) => {
  console.log(number * 2);
});
```

Higher Order Functions

- **Definition**: A higher order function takes one or more functions for the arguments and returns a function or value for the result.

Example Code

```
function operateOnArray(arr, operation) {
  const result = [];
  for (let i = 0; i < arr.length; i++) {
    result.push(operation(arr[i]));
  }
  return result;
}
```

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

const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = operateOnArray(numbers, double);
console.log(doubledNumbers); // [2, 4, 6, 8, 10]
```

- **map() Method**: This method is used to create a new array by applying a given function to each element of the original array. The callback function can accept up to three arguments: the current element, the index of the current element, and the array that map was called upon.

Example Code

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map((num) => num * 2);

console.log(numbers); // [1, 2, 3, 4, 5]
console.log(doubled); // [2, 4, 6, 8, 10]
```

- **filter() Method**: This method is used to create a new array with elements that pass a specified test, making it useful for selectively extracting items based on criteria. Just like the map method, the callback function for the filter method accepts the same three arguments: the current element being processed, the index, and the array.

Example Code

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const evenNumbers = numbers.filter((num) => num % 2 === 0);

console.log(evenNumbers); // [2, 4, 6, 8, 10]
```

- **reduce() Method**: This method is used to process an array and condense it into a single value. This single value can be a number, a string, an object, or even another array. The reduce() method works by applying a function to each element in the array, in order, passing the result of each calculation on to the next. This function is often called the reducer function. The reducer function takes two main parameters: an accumulator and the current value. The accumulator is where you store the running result of your operations, and the current value is the array element being processed.

Example Code

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

const sum = numbers.reduce(

 (accumulator, currentValue) => accumulator + currentValue,

 0

);


console.log(sum); // 15
```

Method Chaining

- **Definition**: Method chaining is a programming technique that allows you to call multiple methods on the same object in a single line of code. This technique can make your code more readable and concise, especially when performing a series of operations on the same object.

Example Code

```
const result = "  Hello, World!  "

 .trim()

 .toLowerCase()

 .replace("world", "JavaScript");


console.log(result); // "hello, JavaScript!"
```

Working with the sort Method

- **Definition**: The sort method is used to sort the elements of an array and return a reference to the sorted array. No copy is made in this case because the elements are sorted in place.

Example Code

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.sort();


console.log(fruits); // ["Apple", "Banana", "Mango", "Orange"]
```

If you need to sort numbers, then you will need to pass in a compare function.
The sort method converts the elements to strings and then compares their sequences of UTF-16 code units values. UTF-16 code units are the numeric values that represent the characters in the string. Examples of UTF-16 code units are the numbers 65, 66, and 67 which represent the characters "A", "B", and "C" respectively. So the number 200 appears before the number 3 in an array, because the string "200" comes before the string "3" when comparing their UTF-16 code units.

Example Code

```
const numbers = [414, 200, 5, 10, 3];


numbers.sort((a, b) => a - b);


console.log(numbers); // [3, 5, 10, 200, 414]
```

The parameters a and b are the two elements being compared. The compare function should return a negative value if a should come before b, a positive value if a should come after b, and zero if a and b are equal.

Working with the every and some Methods

- **every() Method**: This method tests whether all elements in an array pass a test implemented by a provided function. The every() method returns true if the provided function returns true for all elements in the array. If any element fails the test, the method immediately returns false and stops checking the remaining elements.

Example Code

```
const numbers = [2, 4, 6, 8, 10];

const hasAllEvenNumbers = numbers.every((num) => num % 2 === 0);


console.log(hasAllEvenNumbers); // true
```

- **some() Method**: This method checks if at least one element passes the test.
  The some() method returns true as soon as it finds an element that passes the test.
  If no elements pass the test, it returns false.

Example Code

```
const numbers = [1, 3, 5, 7, 8, 9];

const hasSomeEvenNumbers = numbers.some((num) => num % 2 === 0);


console.log(hasSomeEvenNumbers); // true
```

Assignment

Review the JavaScript Higher Order Functions topics and concepts.

Please complete the assignment

Submit

Ask for Help

Navigated to JavaScript Higher Order Functions Review