Javascript Higher-Order Functions

Higher-order Funcitons

Higher order functions are functions that operate on other functions, either by taking them as arguments or by returning them. In simple words, *A Higher-Order function is a function that receives a function as an argument or returns the function as output*.

For example, Array.prototype.map, Array.prototype.filter and Array.prototype.reduce are some of the Higher-Order functions built into the language.

Array.prototype.map

The map() method creates a new array by calling the callback function provided as an argument on every element in the input array.
The map() method will take every returned value from the callback function and creates a new array using those values.
The callback function passed to the map() method accepts 3 arguments: element, index, and array.

Let’s look at some examples:

Array.prototype.map

```
const arr1 = [1, 2, 3];
const arr2 = arr1.map(function(item) {
  return item * 2;
});
console.log(arr2);
```

We can make this even shorter using the arrow function syntax:

```
const arr1 = [1, 2, 3];
const arr2 = arr1.map(item => item * 2);
console.log(arr2);
```

Array.prototype.filter

The filter() method creates a new array with all elements that pass the test provided by the callback function. The callback function passed to the filter() method accepts 3 arguments: element, index, and array.
Let’s look at some examples:

Array.prototype.filter

We can make this even shorter using the arrow function syntax:

```
const persons = [
  { name: 'Peter', age: 16 },
  { name: 'Mark', age: 18 },
  { name: 'John', age: 27 },
  { name: 'Jane', age: 14 },
  { name: 'Tony', age: 24},
];
const fullAge = persons.filter(person => person.age >= 18);
console.log(fullAge);
```

Array.prototype.reduce

The reduce method executes the callback function on each member of the calling array which results in a single output value. The reduce method accepts two parameters: 1) The reducer function (callback), 2) and an optional initialValue.

The reducer function (callback) accepts four parameters: accumulator, currentValue, currentIndex, sourceArray.

Array.prototype.reduce

If an initialValue is provided, then the accumulator will be equal to the initialValue and the currentValue will be equal to the first element in the array.

If no initialValue is provided, then the accumulator will be equal to the first element in the array and the currentValue will be equal to the second element in the array.

Array.prototype.reduce

We can make this even shorter using the arrow function syntax:

```
const arr = [5, 7, 1, 8, 4];
const sum = arr.reduce(function(accumulator, currentValue)
 {
  return accumulator + currentValue;
});
// prints 25
console.log(sum);
```