Theory Class | JavaScript Interviews

 Higher Order Functions | `map` , `filter` , `reduce`

- A higher order function is a function that takes a function as an argument or returns a function as a result.
- Few other functions : `find`, `findIndex`, `forEach`, `some` and `every`.
- `Arrow functions` are also higher-order functions.

Map function | `map()`

> Example to illustrate the concept of `higher-order function : map()`

```
let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

function squarer(x) {
  // callback function
  return x * x;
}

let squared = arr.map(squarer);
console.log(squared);
```

> Output :

```
$ node lecture-017/map.js
[
   1,  4,  9, 16,  25,
  36, 49, 64, 81, 100
]
```

- map() takes a callback function as an argument
- map() is an array function that expects a callback function as an argument
- map() will return a new array with the results of the callback function for each element in the original array

> Example :

```
let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let cubed = arr.map((x) => x * x * x);
console.log(cubed);
```

- operator `=>` is called the arrow operator

> Output:

```
$ node lecture-017/map.js
[
    1,    8,  27,  64,
  125,  216, 343, 512,
  729, 1000
]
```

- When we write a self defined higher-order function, we call it as `polyfill`.
- map() is a polyfill for `Array.prototype.map()`
- map() `does not tamper the original array`
- map() `returns a new array`

> Quick Task : Use `map() to create a new array of strings of the names`

```
let names = ["John", "Mary", "Mike", "Suzy"];
let nameStrings = names.map(function (x) {
  return x;
});
console.log(nameStrings);
```

> Output :

```
$ node lecture-017/map.js
[ 'John', 'Mary', 'Mike', 'Suzy' ]
```

Split function | `split()`

- `split()` is a string method that splits a string into an array of substrings based on the delimiter.
- split() splits a string into an array of substrings based on the separator string you provide as an argument.
- The separator string can be a character, a string, or a regular expression.
- If the separator is not specified, the string is split on every character.
- If the separator is an empty string (""), the string is split on every character.

> Examples :

```
let str = "pepcoder";
let parts = str.split("c");

let str2 = "Hello World";
let parts2 = str2.split(" ");
```

```
console.log(parts);
console.log(parts2);
```

> Output :

```
$ node lecture-017/splitJoin.js
[ 'pep', 'oder' ]
[ 'Hello', 'World' ]
```

Join function | `join()`

- join function reverses the process of split()
- join() takes an array and joins the elements into a string
- it takes an optional separator argument which is used to separate the elements of the array
- if the separator is not specified, the array elements are `separated by commas`

> Example :

```
let joinedStr = parts.join("c"); // "pepcoder"
let joinedStr2 = parts2.join(" "); // "Hello World"

console.log(joinedStr);
console.log(joinedStr2);
```

> Output :

```
$ node lecture-017/splitJoin.js
[ 'pep', 'oder' ]
[ 'Hello', 'World' ]
pepcoder
Hello World
```

Task using `map()` and `split()`

> Use map to take out first name and last name from the nameArr and store it in a new array.

```
let nameArr = ["Milind Mishra", "Rajesh Kumar", "Raju Kumar", "Madan Mishra"];

let firstNames = nameArr.map((x) => x.split(" ")[0]);
let lastNames = nameArr.map((x) => x.split(" ")[1]);
console.log(firstNames);
console.log(lastNames);
```

> Output :

```
$ node lecture-017/map.js
[ 'Milind', 'Rajesh', 'Raju', 'Madan' ]
[ 'Mishra', 'Kumar', 'Kumar', 'Mishra' ]
```

Task to convert `Dollar to Rupees` using `map()` and vice-versa

```
const transactions = [1000, 3000, 4000, 2000, -896, 3800, -4500];
const dollarToRupee = 74.5;
const rupeeToDollar = 1 / 74.5;

let rupee = transactions.map((x) => x * dollarToRupee);
let dollar = transactions.map((x) => x * rupeeToDollar);
console.log(rupee);
console.log(dollar);
```

> Output :

```
$ node lecture-017/map.js
[
    74500, 223500,
   298000, 149000,
   -66752, 283100,
  -335250
]
[
  13.422818791946309,
  40.26845637583892,
  53.691275167785236,
  26.845637583892618,
  -12.026845637583893,
  51.00671140939597,
  -60.402684563758385
]
```

Filter Function | `filter()`

- The filter() method creates a `new array with all elements that pass the test implemented by the provided function`.
- filter() expects an array and a callback function as argument.
- The callback function is called for each element in the array.

> Example : lets say we need to filter out all the even numbers

```
let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let evenNumbers = arr.filter(function (n) {
  return n % 2 === 0;
});
console.log(evenNumbers);
```

- whenever `filter()` spots a `true` value in the callback function, it will add that element to the new array.
- filter() works in boolean values.
- filter() will return a new array with the elements that pass the specified test, it does not modify the original array.

> Task : to filter out all profit transactions from the transactions array

```
const transactions = [1000, 3000, 4000, 2000, -898, 3800, -4500];
let profitTransactions = transactions.filter((x) => x > 0);
console.log(profitTransactions);
```

> Output :

```
$ node lecture-017/filter.js
[ 1000, 3000, 4000, 2000, 3800 ]
```

Chaining of Higher Order Functions | `filter()` and `map()`

> Task : to filter out all the females from the array and print their ages

- Using separate functions

```
let arr = [
  { name: "A", age: 14, gender: "M" },
  { name: "B", age: 34, gender: "M" },
  { name: "C", age: 24, gender: "F" },
  { name: "D", age: 44, gender: "F" },
  { name: "E", age: 44, gender: "M" },
  { name: "I", age: 28, gender: "F" },
  { name: "G", age: 36, gender: "M" },
  { name: "H", age: 47, gender: "F" },
];

// print the age of all the females "F" using filter() map() chaining

let allFemales = arr.filter(function (x) {
  if (x.gender == "F") {
    return true;
  } else {
```

```
    return false;
  }
});

console.log(allFemales);

let femaleAge = allFemales.map(function (female) {
  return female.age;
});

console.log(femaleAge);
```

> Output :

```
$ node lecture-017/filterMapChain.js
[
  { name: 'C', age: 24, gender: 'F' },
  { name: 'D', age: 44, gender: 'F' },
  { name: 'I', age: 28, gender: 'F' },
  { name: 'H', age: 47, gender: 'F' }
]
[ 24, 44, 28, 47 ]
```

- Using chaining

```
// Chaining the filter() and map() together!
// basically output of filter can be mapped

let ageOfAllFemales = arr
  .filter(function (x) {
    if (x.gender == "F") {
      return true;
    } else {
      return false;
    }
  })
  .map(function (female) {
    return female.age;
  });

console.log(ageOfAllFemales);
```

> Output :

```
$ node lecture-017/filterMapChain.js
[ 24, 44, 28, 47 ]
```

Find Function | `find()` method

-The find() method returns the value of the first element in the array that satisfies the provided testing function. -Otherwise undefined is returned. -find() expects a callback function as its argument -The callback function is called for each element of the array, and the element is passed as the first argument to the callback function. -The callback function should return a boolean value. -If the callback function returns true, find() returns the value of the element. Otherwise, it returns undefined.

```
let arr = [-1, -2, -3, -4, -5, -6, -7, -8, -9, 10];

let firstPositive = arr.find(function (x) {
  return x > 0;
});

console.log(firstPositive); // 10
```

- it just returns the first element that satisfies the condition, a value and not an array
- used to look up a value in an array or anything else that is iterable

Some Function | `some()` method

- some() function is used to `check if any of the elements in an array passes the condition`.
- some() expects a callback function as its argument
- The callback function is called for each element of the array, and the element is passed as the first argument to the callback function.
- The callback function should return a boolean value.
- If the callback function returns true, some() returns true. Otherwise, it returns false.

> Example code :

```
let arr = [-1, -2, -3, -4, -5, -6, -7, -8, -9, 10];

let firstPositive = arr.some(function (x) {
  return x > 0;
});

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

- if even one value `satisfies the condition`, it returns `true`
- it returs true of one or more than one value satisfies the condition

Every Function | `every()` method

- every() method checks if all elements in an array pass a test.
- it returns true if all elements pass the test, otherwise it returns false.

```
let a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let allPositive = a.every(function (x) {
  return x > 0;
});
console.log(allPositive); // true
```

> Next class : reduce(), forEach(), findIndex() custom polyfills implementations : map(), filter(), forEach()