JavaScript

Functions:

A function having a name added to its declaration is called a named function. Whenever we want to call a function, we will enter the function name and added parentheses at the end of the function name.

To find sum of two numbers, we will create a function add.

```
// named function

function add(num1, num2) {
    console.log('Calculating Sum')
    return num1 + num2
}
```

add(10, 20) will return the value so to reach/catch it we can write it as;

```
var sum = add(10, 20)
console.log(sum)
```

Output: 30

In the function we are passing two numbers (num1 & num2). But in JavaScript we can pass the function itself. So

```
function add(num1, num2) {
    console.log('Calculating Sum')
    return num1 + num2
}
function specialFunction(passedValue) {
    console.log('This is coming from function named specialFunction')
    console.log('passedValue is ', passedValue)
}
specialFunction()
```

Since we have not passed any value in specialFunction(), when we call the function, the output shows:

```
This is coming from function named specialFunction
passedValue is undefined.
```

But when we pass in value 10 to specialFunction(10)

Output:

```
This is coming from function named specialFunction
passedValue is 10.
```

Now, if we pass the add () function into the specialFunction as a parameter, then

```
function add(num1, num2) {
    console.log('Calculating Sum')
    return num1 + num2
}
function specialFunction(passedValue) {
    console.log('This is coming from function named specialFunction')
    console.log('passedValue is ', passedValue)
}
specialFunction(add)
```

OUTPUT:

```
This is coming from function named specialFunction
passedValue is f (num1, num2) {
     console.log('Calculating Sum')
     return num1 + num2
}
```

If we create a new function here as subtract() and pass it as parameter to the special function then,

```
function subtract(num1, num2) {
    console.log('Calculating Difference')
    return num1 - num2
}
function add(num1, num2) {
    console.log('Calculating Sum')
    return num1 + num2
}
function specialFunction(params) {
    console.log('This is coming from function named specialFunction')
    console.log('param')
}
specialFunction(subtract)
```

Output:

```
This is coming from function named specialFunction
passedValue is f (num1, num2) {
    console.log('Calculating Difference)
    return num1 - num2
}
```

As we are passing the value, we can also pass a value as 123 (an integer) and then

```
function specialFunction(passedVlaue) {
    console.log('This is coming from function named specialFunction')
    console.log('passedVlaue is', passedVlaue)
    console.log(passedVlaue + 10)
    console.log('passedVlaue became', passedVlaue)
}
specialFunction(123)
```

Output:

passedValue is 123

133

passedValue became 123 // we have not made changes to the passedValue or incremented (passedValue = passedValue + 10) it, so the passedValue would remain same.

Now, if we pass a function add(), the passedValue will be referring to the add() function. So, the passedValue parameter in my specialFunction parameter would become a function. Since it is a function we can call that function.

```
function add(num1, num2) {
    console.log('Calculating Sum')
    return num1 + num2
}

function specialFunction(passedVlaue) {
    console.log('This is coming from function named specialFunction')
    console.log('passedVlaue is', passedVlaue)
    var result = passedVlaue(30, 40)
    console.log(result)
}
specialFunction(add)
```

Output:

This is coming from function named specialFunction
passedVlaue is ***ƒ add(num1, num2) {***
console.log('Calculating Sum')
return num1 + num2
}
Calculating Sum
70

```
function specialFunction(passedVlaue) {
    console.log('This is coming from function named specialFunction')
    console.log('passedVlaue is', passedVlaue)
    var result = passedVlaue(30, 40)
    console.log(result)
}
// specialFunction(add)
var dumy = specialFunction(subtract(30, 40))
```

This will give you an error stating that passedValue is not a function because passedValue is -10. So, in the line ***`var result = passedValue (30, 40)`***, passedValue is actually called as a function, but the passedValue typeof is integer.

Array Methods:

```
//  Array Methods
var arr = [123, 332, 20, 50]

// to loop we will use for loop.
for (var index = 0; index < arr.length; index++) {
    // const element = array[index]
    console.log(arr[index]) // 0 1 2 3 4...
}
```

Output:

123
332
20
50

We loop over arrays using for loop. In the same way,

`index = 0` → speaks about what should be the starting value

`index < arr.length` → speaks about what should be the condition to iterate

`index++` → how the value should change.

forEach:

forEach() calls a provided callback function once for each element in an array in ascending order. It is not invoked for index properties that have been deleted or are uninitialized.

Callback is invoked with three arguments.

1. The value of the element
2. The index of the element
3. The array object being tranversed

`forEach()` method executes a provided function once for each array element. It is defined on the array itself. This is a function and it needs a callback function (a function passed as parameter).

```
//  Array Methods
var arr = [123, 332, 20, 50]

// to loop we will use for loop.

// for (var index = 0; index < arr.length; index++) {
//  const element = array[index]
//  console.log(arr[index]) // 0 1 2 3 4...
// }

function abc() {
    console.log('Executing')
}

arr.forEach(abc)
```

Output:

4 Executing

The reason it is showing ‘4’ is for every item in the given array. So, forEach() function will call your function ‘arr.length’ times.

To access the elements in the array, forEach() function passes some parameters our function.

- element → forEach() calls 5 times, and the first parameter is being passed as the element of the array. So for each call, subsequent element will be called
- index → the same happens for index.
- myArr → this refers to the array itself.

```
function abc(element, index, myArr) {
    // console.log('Executing')
    console.log(element)
    console.log(index)
    console.log(myArr)
}

arr.forEach(abc)
```

The above three parameters (element, index and myArr) are default parameters of **`forEach()`** function

Even / Odd

```
var numbers = [55, 12, 30, 2, 300, 555, 65, 88]

numbers.forEach(function (item, index, originalArray) {
    if (item % 2 === 0) {
        console.log(item + 'is an even number')
    } else {
        console.log(item + 'is an odd number')
    }
})
```

Output:

55 is an odd number
12 is an even number
30 is an even number
2 is an even number
300 is an even number
555 is an odd number
65 is an odd number
88 is an even number

Map()

The `map()` method **creates a new array** populated with the results of calling a provided function on every element in the calling array.

It calls a provided callback function **once for each element** in an array, in order, and constructs a new array from the results. Callback is invoked only for indexes of the array which have assigned values (including undefined).

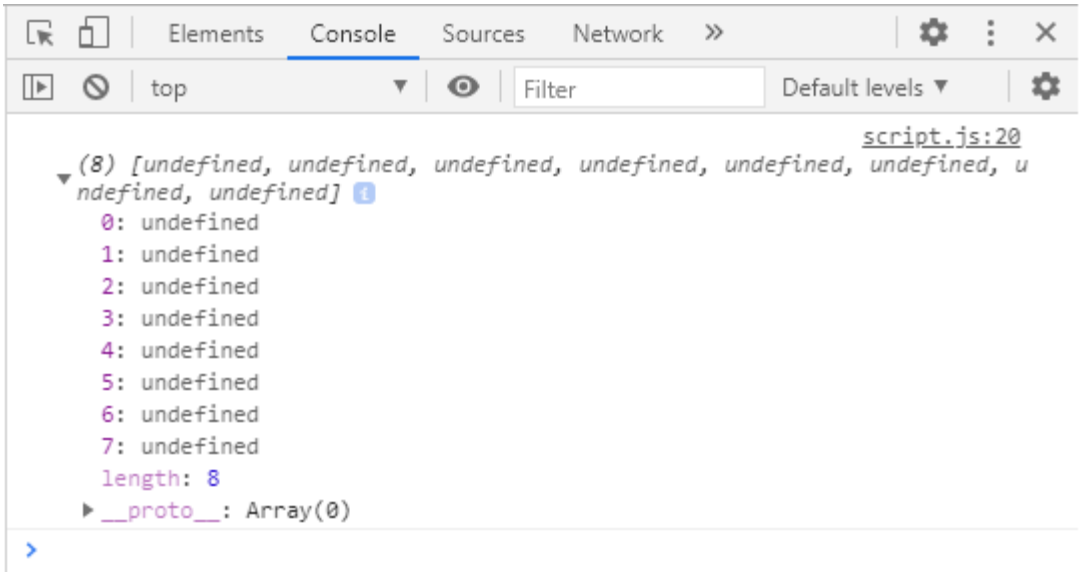
It is not called for misising elements of the array

- indexes that have never been set
- indexes which have been deleted.

```
var numbers = [55, 12, 30, 2, 300, 555, 65, 88]

var mapResult = numbers.map(function (item, index, originalArray) {
    var square = item * item
})
console.log(mapResult)
```

Output:



To get square of all the elements in the array:

```
var numbers = [55, 12, 30, 2, 300, 555, 65, 88]

var mapResult = numbers.map(function (item, index, originalArray) {
    var square = item * item
    return square
})
console.log(mapResult)
```

Output:

```
(8) [3025, 144, 900, 4, 90000, 308025, 4225, 7744]
0: 3025
1: 144
2: 900
3: 4
4: 90000
5: 308025
6: 4225
7: 7744
length: 8
__proto__: Array(0)
```

NOTE: **`forEach()`** will make changes to the array itself, but **`map()`** function gives a completely new array.

Objects:

It's representation of real world entity. It is similar to dictionary in Python. In order to design an object, which is a collection of key-value pairs

```
var person = {
    name: 'Yash',
    age: 50
    }

console.log(person)
console.log(person.name)
console.log(person.name, person.age)
```

Output:

{name: "Yash", age: 50}
Yash
Yash 50

This is a basic way of defining an object.

We can define multiple objects as well.

```
var person1 = {
    name: 'Yash',
    age: 50
}

var person2 = {
    name: 'Josh',
    age: 10
}

console.log(person1, person2)
```

We can define a function inside it as well.

```
var person1 = {
    name: 'Yash',
    age: 50,
    sayHello: function() {
        console.log('Hello')
    }
}

console.log(person1)
person1.sayHello()
```

Output:

{name: "Yash", age: 50, sayHello: *f*}

Hello

The other way to define objects is using functions.

```
function Person(passedName, PassedAge) {
    return {
        name: passedName,
        age: PassedAge
    }
}

var abc = new Person('Yash', 50)
console.log(abc)
```

Output:

{name: "Yash", age: 50}