Functional Programming: Basic Exercises

1.Make a function called composed Value that takes two functions f1 and f2 and a value and returns f1(f2(value)), ie, the first function called on the result of the second function called on the value.

function square (x) { return (x*x); }

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

composedValue (square, double, 5); --> 100 // i.e. square (double (5))

Ans:-

```
function composedValue(f1, f2, value) {
    return f1(f2(value));
}
function square(x) {
    return x * x;
}
function double(x) {
    return x * 2;
}
console.log(composedValue(square, double, 2));
```

```
> node "c:\Users\ASUS\Desktop\CAPGEMINI\NODE JS TRAINING\4 JAVASCRIPT\Javascript Assignment -2\1JS Composed Value Function.js"
16
```

2. Make a function called compose that takes two functions f1 and f2 and returns a new function that, when called on a value, will return f1(12(value)). Assume that f1 and f2 each take exactly one argument.

var f1 compose (square, double);

f1(5): → 100

f1(10): → 400

var f2 compose (double, square);

f1(5): → 50

f1(10): → 200

Ans:-

```
function compose(f1, f2) {
    return function (x) {
        return f1(f2(x));
    };
}

function square(x) {
    return x * x;
}

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

var f1 = compose(square, double);
var f2 = compose(double, square);

console.log(f1(5));
console.log(f1(10));
console.log(f2(5));
console.log(f2(10));
```

```
>node "c:\Users\ASUS\Desktop\CAPGEMINI\NODE JS TRAINING\4 JAVASCRIPT\Javascript Assignment -2\2JS Composed2.js"
100
400
50
200
```

3. Make a function called "find" that takes an array and a test function, and returns the first element the array that "passes" (returns non-false for) the test. Don't use map, filter, or reduce.

function isEven (num) { return (num%2 == 0);

isEven (3) --> false

isEven (4) --> true

find([11, 3, 5, 4, 2], isEven); → 4

Ans:-

```
function find(arr, f) {
    for (var i = 0; i < arr.length; i++) {
        if (f(arr[i])) {
            return arr[i];
        }
    }
}
function isEven(num) {
    return (num % 2 == 0);
}
```

```
console.log(isEven(3));
console.log(isEven(4));
console.log(find([1, 3, 5, 4, 2], isEven));
```

```
> node "c:\Users\ASUS\Desktop\CAPGEMINI\NODE JS TRAINING\4 JAVASCRIPT\Javascript Assignment -2\3JS Even.js"
false
true
4
```

4. Recent JavaScript versions added the "map" method of arrays, as we saw in the notes and used in the previous set of exercises. But, in earlier JavaScript versions, you had to write it yourself Make a function called "map" that takes an array and a function, and returns a new array that is the result of calling the function on each element of the input array. Don't use map, filter, or reduce

map ([1, 2, 3, 4, 5], square): --> [1, 4, 9, 16, 25]

map [1, 4, 9, 16, 25], Math.sqrt);-> (1, 2, 3, 4, 5]

Hint: remember the push method of arrays.

Ans:-

```
function map(arr, f) {
    var outArr = [];
    for (var i = 0; i < arr.length; i++) {
        outArr[i] = f(arr[i]);
    }
    return outArr;
}

function square(x) {
    return x * x;
}

console.log(map([1, 2, 3, 4, 5], square));
console.log(map([1, 4, 9, 16, 25], Math.sqrt));
```

```
> node "c:\Users\ASUS\Desktop\CAPGEMINI\NODE JS TRAINING\4 JAVASCRIPT\Javascript Assignment -2\4JS Map.js"
[ 1, 4, 9, 16, 25 ]
[ 1, 2, 3, 4, 5 ]
```

Functional Programming: Advanced Exercises

1. Make a "pure" recursive version of find. That is, don't use any explicit loops (eg for loops or the forEach method), and don't use any local variables (eg, var x………) inside the functions. Hint: remember the slice method of arrays.

function isEven (num) {return (num%2 ==0); }

isEven (3) --> false

isEven (4) --> true

Find[1, 3, 5, 4, 2], isEven); → 4

Ans:-

```
function find(arr, f) {
  if (arr.length > 0) {
      if (f(arr[0])) {
          return arr[0]
      }
      return find(arr.slice(1), f);
  }
}

function isEven(num) {
  return (num % 2 == 0);
}

console.log(isEven(3));
console.log(isEven(4));
console.log(find([1, 3, 5, 4, 2], isEven));
```

false

true

4

2. Make a “pure" recursive version of map. Hint: remember the slice and concat methods of arrays.

map (1, 2, 3, 4, 5), square); → [1, 4, 9, 16, 25]

map ([1, 4, 9, 16, 25, Math.sqrt); →(1, 2, 3, 4, 5]

Ans:-

```
var newArr = [];
        function map(arr, f) {
            if (arr.length > 0) {
                newArr.push(f(arr[0]));
                map(arr.slice(1), f);
            }
        }

        function square(x) {
            return x * x;
        }

        map([1, 2, 3, 4, 5], square);
        map([1, 4, 9, 16, 25], Math.sqrt);

console.log(newArr);
```

```
[
  1, 4, 9, 16, 25,
  1, 2, 3,  4,  5
]
```

3.JavaScript lets you define anonymous functions and call them right on the spot. For example, (function(x) (return xx, X5) returns 25. Also, if you concatenate a string with a function, the result is a string that looks more or less like the function definition. For example

function square (x) { return x * x; } “square is” + square

➔ “square is function square (x) (return x * x; }”

Use these ideas to make an anonymous function call that outputs a string, where inside that string is exactly what was typed in as the function call. Le, you go to the Firebug console and type in

(function(...) (something)) (blah)

and get back

(function () (nomeshing)) (biab)

The return value should be exactly what you typed in, except that it has quotes around it, and it is OK if the whitespace (spaces, carriage returns) in the return value is not exactly the same as in the input. To make it even harder, you are not allowed to use arguments callee or the arguments array at all. It can be done with function", "return" a variable name, parens, curly braces, and double quotes: no obscure JavaScript feature (or anything else at all) is needed. The answer is short, but this is a very tricky problem

Ans:-

```
setTimeout(function () {
    console.log('Execute later after 10 second')
}, 10000); //It is in milliseconds
```

```
> Execute later after 10 second
```