

Assignment – 2

1. Make a function called `composedValue` that takes two functions `f1` and `f2` and a value and returns `f1(f2(value))`, i.e., 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)).
```

JavaScript Code:

```
Q1 > JS index.js > ...
1   let square = (num) => {
2   |   return Math.pow(num, 2);
3   };
4
5   let double = (num) => {
6   |   return num * 2;
7   };
8
9   let composedValue = (square, double, x) => {
10  |   return square(double(x));
11  };
12
13  document.write(`<h3>Output: ${composedValue(square, double, 5)}</h3>`);
14
```

Output:

← → ↻ ⓘ 127.0.0.1:5500/Q1/index.html 🔗 ☆ 🖨️ ⚙️ 📱 👤 ⋮

Q1) Make a function called `composedValue` that takes two functions `f1` and `f2` and a value and returns `f1(f2(value))`, i.e., 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))
```

Output: 100

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(f2(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);
```

```
f2(5); --> 50
```

```
f2(10); --> 200
```

JavaScript Code:

```

Q2 > JS index.js > ...
1  let square = (num) => {
2    return Math.pow(num, 2);
3  };
4
5  let double = (num) => {
6    return num * 2;
7  };
8
9  let f1 = (square, double, x) => {
10   return square(double(x));
11 };
12
13 let f2 = (double, square, x) => {
14   return double(square(x));
15 };
16
17 document.write(`<h3>Output: f1(5) = ${f1(square, double, 5)}</h3>`);
18 document.write(`<h3>Output: f1(10) = ${f1(square, double, 10)}</h3>`);
19
20 document.write(`<h3>Output: f2(5) = ${f2(double, square, 5)}</h3>`);
21 document.write(`<h3>Output: f2(10) = ${f2(double, square, 10)}</h3>`);
22

```

Output:

← → ↻ ⓘ 127.0.0.1:5500/Q2/index.html 🔗 ☆ 🖨 ⚙ 🎵 👤 ⋮

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(f2(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);
f2(5); --> 50
f2(10); --> 200

```

Output: `f1(5) = 100`

Output: `f1(10) = 400`

Output: `f2(5) = 50`

Output: `f2(10) = 200`

3. Make a function called “find” that takes an array and a test function, and returns the first element of 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([1, 3, 5, 4, 2], isEven); --> 4
```

JavaScript Code:

```
Q3 > JS index.js > find
1  function isEven(num) {
2    |   return num % 2 == 0;
3    |   }
4
5    document.write(`<h3>Output: isEven(3) = ${isEven(3)}</h3>`);
6    document.write(`<h3>Output: isEven(4) = ${isEven(4)}</h3>`);
7
8    let arr = [1, 3, 5, 4, 2];
9
10   function find(arr, isEven) {
11     |   for (let i = 0; i < arr.length; i++) {
12     |     |   if (isEven(arr[i])) {
13     |     |     |   return arr[i];
14     |     |     }
15     |     }
16   }
17
18   document.write(
19     |   `<h3>Output: find([1, 3, 5, 4, 2], isEven) = ${find(arr, isEven)}</h3>`
20   |   );
21
```

Output:

 127.0.0.1:5500/Q3/index.html 

. Make a function called “find” that takes an array and a test function, and returns the first element of 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([1, 3, 5, 4, 2], isEven); --> 4
```

Output: isEven(3) = false

Output: isEven(4) = true

Output: find([1, 3, 5, 4, 2], isEven) = 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.

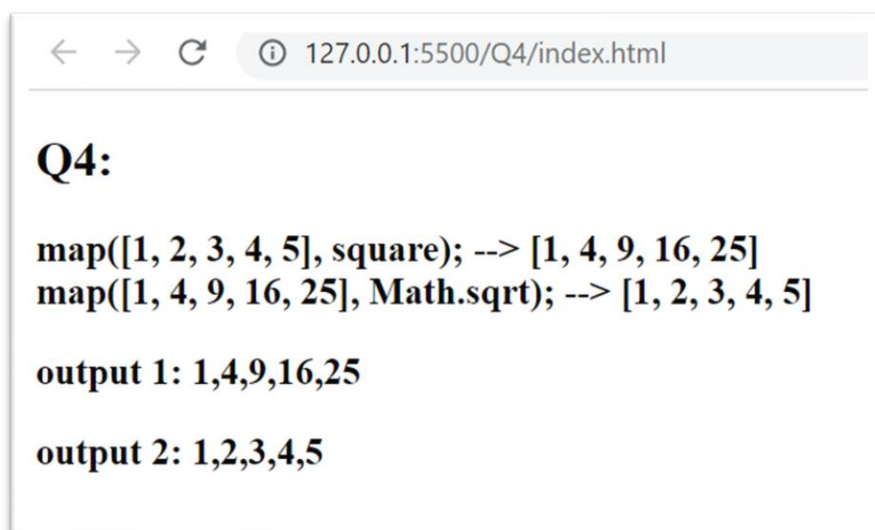
JavaScript Code:

```

Q4 > js index.js > ...
1  function square(value) {
2    return value * value;
3  }
4  function process(arr, func) {
5    var arr2 = [];
6    for (var i = 0; i < arr.length; i++) {
7      arr2.push(func(arr[i]));
8    }
9    return arr2;
10 }
11
12 const map = (arr, fun) => process(arr, fun);
13 document.write(`<h3>output 1: ${map([1, 2, 3, 4, 5], square)}</h3>`);
14 document.write(`<h3>output 2: ${map([1, 4, 9, 16, 25], Math.sqrt)}</h3>`);
15

```

Output:



Adv 1. Make a “pure” recursive version of find. That is, don’t use any explicit loops (e.g. for loops or the forEach method), and don’t use any local variables (e.g., var x = ...) inside the functions.

Hint: remember the slice method of arrays.

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

isEven(3) --> false

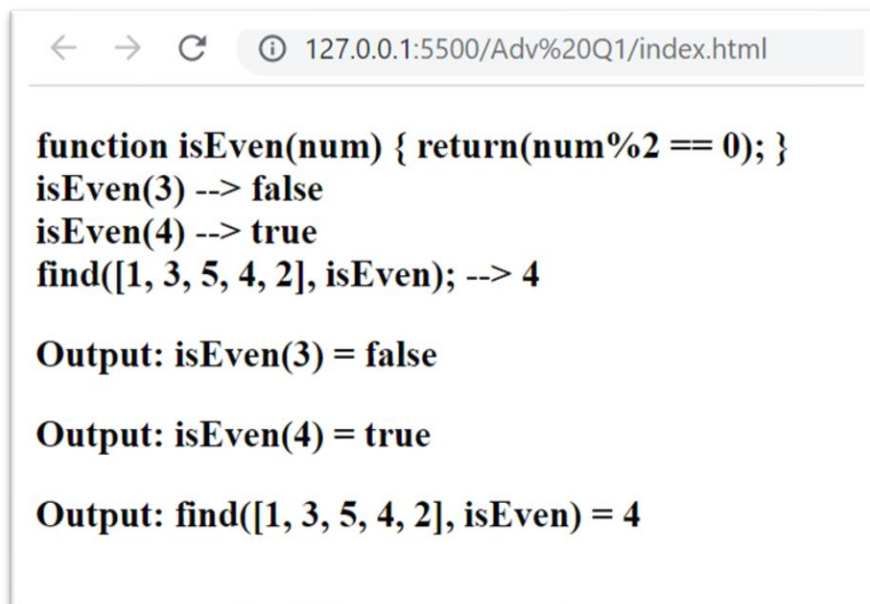
isEven(4) --> true

find([1, 3, 5, 4, 2], isEven); --> 4.

JavaScript Code:

```
Adv Q1 > Js index.js > ...
1  function isEven(num) {
2      return num % 2 == 0;
3  }
4
5  document.write(`<h3>Output: isEven(3) = ${isEven(3)}</h3>`);
6  document.write(`<h3>Output: isEven(4) = ${isEven(4)}</h3>`);
7
8  let arr = [1, 3, 5, 4, 2];
9
10 function find(arr, isEven) {
11     for (let i = 0; i < arr.length; i++) {
12         if (isEven(arr[i])) {
13             return arr[i];
14         }
15     }
16 }
17
18 document.write(
19     `<h3>Output: find([1, 3, 5, 4, 2], isEven) = ${find(arr, isEven)}</h3>`
20 );
21
```

Output:



```
function isEven(num) { return(num%2 == 0); }
isEven(3) --> false
isEven(4) --> true
find([1, 3, 5, 4, 2], isEven); --> 4

Output: isEven(3) = false

Output: isEven(4) = true

Output: find([1, 3, 5, 4, 2], isEven) = 4
```

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

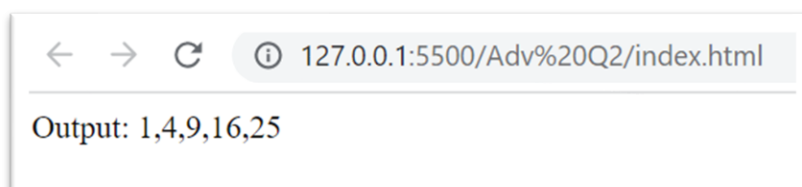
map([1, 2, 3, 4, 5], square); --> [1, 4, 9, 16, 25]

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

JavaScript Code:

```
Adv Q2 > JS index.js > ...
1  function square(num) {
2      return num * num;
3  }
4  var resList = [];
5  function map(list, fun) {
6      if (list.length == 0) {
7          return;
8      }
9      resList.push(fun(list.slice(0, 1)));
10     list = list.slice(1, list.length);
11     return map(list, fun);
12 }
13 map([1, 2, 3, 4, 5], square);
14 document.write(`Output: ${resList}`);
15
```


Output:



3. JavaScript lets you define anonymous functions and call them right on the spot. For example, `(function(x) { return x*x; })(5)` 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; }"
```

JavaScript Code:

Adv Q3 >  index.js > ...

```
1  var myFunction = function (a) {  
2    return a * a;  
3  };  
4  document.write(`<h3> Output: func(5) = ${myFunction(5)}</h3>`);  
5
```

Output:

← → ↻ ⓘ 127.0.0.1:5500/Adv%20Q3/index.html

Output: func(5) = 25