

# Functional Programming For React

Today's Quote By Tanay

Functions are first class citizens

Declarative Programming

Why FP?

How to do Functional Programming?

Immutability

Properties

Question

Pure Functions

3 rules

Question

Higher Order Functions

Properties

Example

Question

ES6 Methods

[ ].map(fn)

[ ].filter(fn)

[ ].reduce(fn, [initialValue])

Properties

HW: Can you write your own reduce using for loop.

HW Questions

IMP: My mistake

HW Answers

Currying

Properties

When to use currying?

Logger Question

Use Case

Composition

Example 1:

Callback  $\Rightarrow$  Call me later, this is not callback.

Example 2:

Question

My Answer (Wrong ;-)\_

Tanay's Answer

THE ONE HOMEWORK

## Today's Quote By Tanay

Be lazy  $\Rightarrow$  No knowledge by reading or writing  $\Rightarrow$  No wealth  $\Rightarrow$  No Friends

## Functions are first class citizens

1. It can be assigned to variables
2. sent to other functions as arguments
3. can be added to objects
4. can be added to arrays as well
5. can be returned from other functions

NOTE: Anything which we can do with variable, function can do the same.

## Declarative Programming

1. FP is a part of a larger programming paradigm
2. Instead of saying how it should happen, what should happen.
3. We have seen how React abstracts away DOM creation.
4. React is an awesome explanation of DP.

## Why FP?

1. Does not depend on global variables
  - a. Hence, less bugs
2. More code reuse
3. Easier to understand and read
4. Super easy to test

# How to do Functional Programming?

## Immutability

### Properties

1. Do not change any data. Always return a new copy
2. In Javascript, function arguments are references to the actual data.
  - a. if you change the parameter itself (as with `num` and `obj2`), that won't affect the item that was fed into the parameter. But if you change the *internals* of the parameter, that will propagate back up

```
function changeStuff(a, b, c) { a = a * 10; b.item = "changed"; c =  
{item: "changed"}; } var num = 10; var obj1 = {item: "unchanged"};  
var obj2 = {item: "unchanged"}; changeStuff(num, obj1, obj2);  
console.log(num); console.log(obj1.item); console.log(obj2.item);
```

### Question

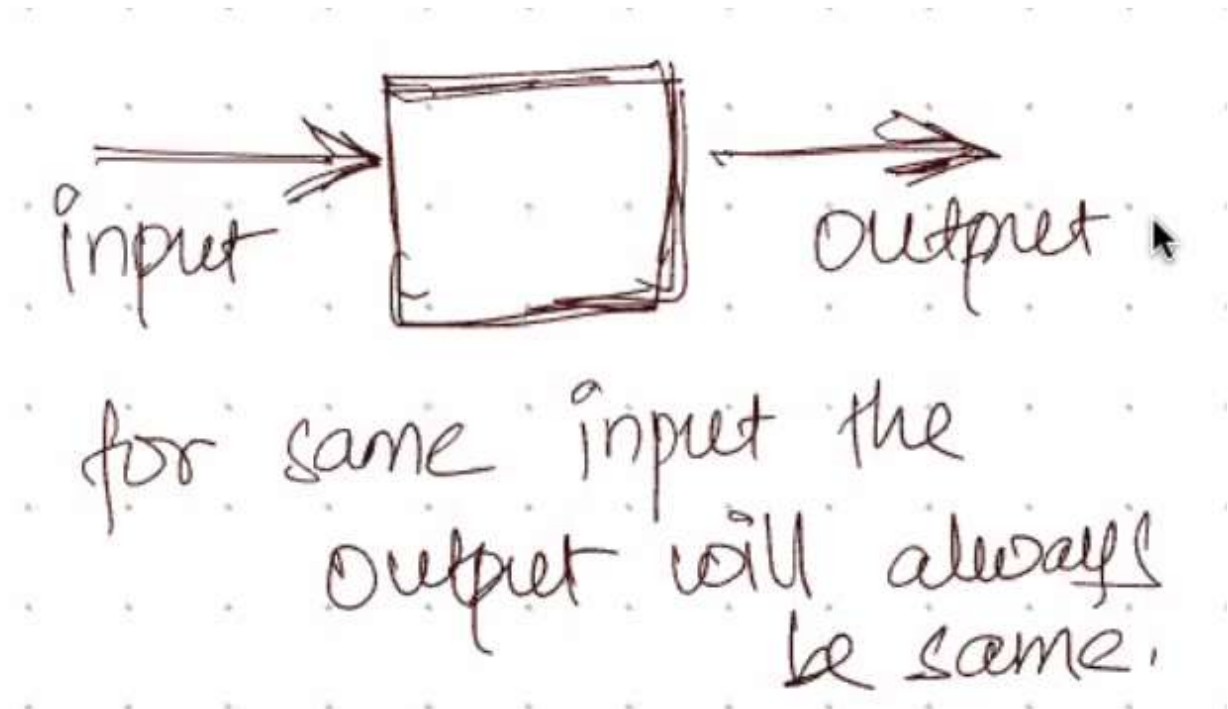
1. Take an object with your mother's name and your age. Now, create an obj for your sibling by age difference.

```
const tanay = { mother: "x", age: 22 }; NOTE: "=" creates a reference.  
Deep Copy and Shallow Copy are different. const tanvi = tanay; tanvi.age  
= tanay.age - 4; console.log(tanay === tanvi);
```

2. Take an array with 5 colors. Create another array by adding two more colors to it

```
const arr = ["a", "b", "c", "d", "e"]; const narr = [ ...arr, "f", "g" ];
```

## Pure Functions



### 3 rules

1. at least one argument
2. return a value or other function
3. should not mutate any of its argument.

### Question

Write a function birthday() to take a person's name and age in an object and increase age

```
const birthday = (personInfo) => { return { ...personInfo, age: personInfo.age + 3 }; };
```

This is wrong.. Curly Braces and return.. Not practiced ES6 ;-;;

```
const birthday = personInfo => { ...personInfo, age: personInfo.age + 3 };
```

This is wrong... We missed Parentheses in returning

```
const birthday = (personInfo) => ({ ...personInfo, age: personInfo.age + 3 });
```

## Higher Order Functions

### Properties

1. Can take functions as arguments
2. Return functions
3. Can do both.

### Example

**Array.filter** is a higher order function.

### Question

1. Write a function which can tell whether a number is less than 10 or not. Supply this function to Array.filter() to get an array with no 10s in it.

```
function isLessThan10 = (number) => (number < 10); const arr = [9, 10, 11];  
arr.filter(isLessThan10);
```

NOTE: We have abstracted away. It is declarative + more readable.

2. Given an array of numbers, return an array with all numbers incremented by 2.

```
const incrementByTwo = num => num + 2; numbers.map(incrementByTwo);
```

*Is map/filter a pure function or not? ⇒ Its pure.*

because does not mutate, creates a new array.

and returns a value/function.

*When to validate arguments in a function?*

⇒ Whenever it is coming from outside of our program, we need to validate it.

Example: Local Storage, API, etc

## ES6 Methods

### [ ].map(fn)

1. Returns a new array
2. Modifies each element ⇒ Must return a value

```
arr.map( (value, index) => );
```

### [ ].filter(fn)

Returns a new array

Condition check on each element

• filter()

fn

num  $\rightarrow$  num < 10



true



take  
in array



false



discard  
that  
value

[ ].reduce(fn, [initialValue])

Properties

1. fn  $\Rightarrow$  should be pure.



To return one final value after iterating on every item in an array.

arr.reduce(fn)

fn (accumulator, currentValue)

accumulator  $\Rightarrow$  coming from previous run

currentValue  $\Rightarrow$  value of array one time.

HW: Can you write your own reduce using for loop.

HW Questions

a) Find the sum of all odd numbers.

b) Find the sum of all numbers at odd indices.

c) Find the biggest number in the array.

d) Find the numbers divisible by 10.

e) Return an array of numbers where odd numbers are incremented by one and even numbers are decremented by one.

f) return an object with sum of all odd numbers and even numbers separately.

a) Find the number of strings with similar number of characters.

```
const input = ["apple", "orange", "mango", "papaya"]
```

// output: { 5: 3, 6: 1 }

## Homework Questions

b) return an array with strings which have vowels.

c) Return an array of objects with key as item and value as number of characters in the string.

**# Rules**

1. Use pure functions.
2. Create functions separately for reuse.

## IMP: My mistake

### My Code

```
const sumOfAllOddNumbers = (accumulator = 0, value) => { if (value % 2)
accumulator += value; return accumulator; } [1, 3, 5, 2, 22, 11,
9].reduce(sumOfAllOddNumbers); const sumOfAllOddNumbersObject = (accumulator
= { oddSum: 0, evenSum: 0 }, value) => { if (value % 2) accumulator.oddSum +=
value; else accumulator.evenSum += value; return accumulator; } // this
doesn't work ;---; // Because, I MUTATEDDDDD... [1, 3, 5, 2, 22, 11,
9].reduce(sumOfAllOddNumbersObject);
```

### Tanay's Code:

```
const numbers = [1, 3, 5, 2, 22, 11, 9]; const oddSum = (sum, num) => num % 2 === 0 ? sum : sum + num; //now Testing becomes easier oddSum(11, 3) => 14 oddSum(2, 4) => 2 numbers.reduce(oddSum, 0); //NOTE: PURE Function inside reduce const reduceObj = (oddEvenObj, num) => num % 2 === 0 ? { ...oddEvenObj, even: oddEvenObj.even + num } : { ...oddEvenObj, odd: oddEvenObj.odd + num } const oddEvenObj = { even: 0, odd: 0 }; numbers.reduce(reduceObj, oddEvenObj);
```

## HW Answers

- Done above
- List

```
const sumAtOddIndex = (sum, value, index) => index % 2 === 1 ? sum + value : sum; [].reduce(sumAtOddIndex, 0);
```

c.

```
const biggestNumberInArr = (big, value) => Math.max(big, value); [].reduce(biggestNumberInArr, Number.NEGATIVE_INFINITY);
```

d.

```
const isDivisibleBy10 = value => value % 10 === 0; [].filter(isDivisibleBy10);
```

e.

```
const increaseOddDecreaseEven = value => value % 2 === 0 ? value - 1 : value + 1; [].map(increaseOddDecreaseEven);
```

f. Done above

a':

```
const numOfStrings = (numObj, str) => numObj.hasOwnProperty(str.length) ? {  
  ...numObj, str.length: numObj[str.length] + 1 } : { ...numObj, str.length: 1  
}; [].reduce(numOfStrings, {});
```

b':

```
const checkForVowel = (hasVowel, char) => 'aeiou'.indexOf(char) !== -1 ?  
hasVowel | true : hasVowel; const hasVowel = str => str.reduce(checkForVowel,  
false); [].filter(hasVowel);
```

c':

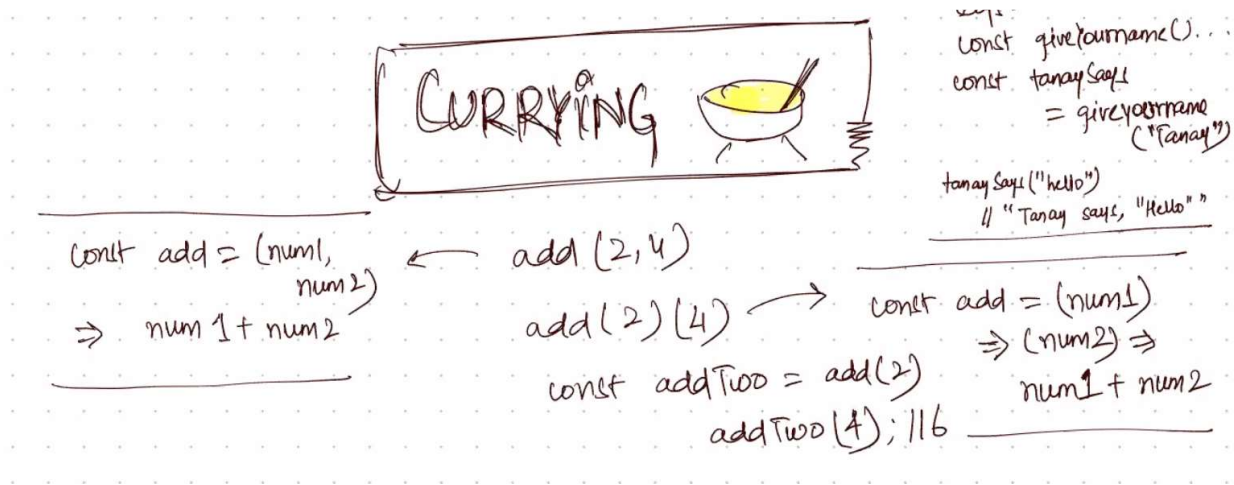
```
const createObj = str => { str: str.length }; ["tanay", "tanvi", "kishan",  
"prerana"].map(createObj);
```

# Currying

## Properties

### How this works?

Closure makes this work....



`add(2, 4)`  $\Rightarrow$  Normal people

`add(2)(4)`  $\Rightarrow$  first function is returning a function.

```
const add = num1 => num2 => num1 + num2; const fourAdded = add(4); //
Returns: (num2 => num1 + num2;) fourAdded(2); // returns 6
```

```
[[Scopes]]: Scopes[3]
0: Closure (add) {num1: 4}
1: Script {numbers: Array(7), oddEvenObj: {...}, oddSum: f, reduceObj: f, add: f}
2: Global {0: Window, window: Window, self: Window, document: document, name: '', location: Location, ...}
```

## When to use currying?

When we don't have all values of function at same time

## Logger Question

```
const nameLogger = name => msg => console.log(`${name} ${str}`); const
ritikLogger = nameLogger("ritik"); const teamB1Logger = nameLogger("Team
B1");
```

## Use Case

1. When there is logger, we want to attach id to the logger, from which place it came.  
So we use currying there to add the id to identify the place.

## Composition

Output of  $f(x)$  should go to  $g(x)$ .  $\Rightarrow g(f(x))$

### Example 1:

```
const add = (num1, num2) => num1 + num2; console.log(add(2, 3)); // Output of  
add is input of console.
```

Callback  $\Rightarrow$  Call me later, this is not callback.

### Example 2:

```
const double = num => num * 2; double(add(2, 3));
```

## Question

Write a function which can log any text with your username.

Write a function which can write any text with your userId.

Now compose both functions to give one function which can log any text with username and ID.

My Answer (Wrong ;-)\_

```
const getUserId = (userId) => userId; const getUsername = (userName)
=> userName; const logger = (userId, userName) => console.log(`${userId}
${userName}`); logger( getUserId("123"), getUsername("ritik") );
```

## Tanay's Answer

```
const logWithName = msg => `Tanay says, ${msg}`; const logWithId = msg =>
`ID: 1234 :: ${msg}`; const logWithIdAndName = msg =>
logWithId(logWithName(msg)); logWithIdAndName("We are reockstars");
```

# THE ONE HOMEWORK

Q. Write a function `compose()` which can take any number of functions and return a function which will run the given functions in order when called with an argument.



? Hint This exercise would need  
 (a) Carrying  
 (b) Reduce  
 (c) rest operator cuz you don't know how many values can be passed to compose.

ex

```
const increment = num => num+1;
const square = num => num*num;
const incrementThenSquare = compose(
  increment,
  square);
incrementThenSquare(2); //9
```

Write this function `compose` as a util so that you can use it for all your functional programming needs. 😊

