Functional Programming Exercises

section 1: Immutability

understanding

- Do not change any data; always return a new copy
- In JavaScript, function arguments are references to actual data

```
> const anObj = { name: "tanay", age: 31 }
< undefined
> const anObjAfterBday = { name: "tanay", age: 32 }
< undefined
> const anObjAfterBday2 = anObj;
< undefined
> anObjAfterBday2.age = 32;
< 32
> anObj
< ▶ {name: 'tanay', age: 32}
>
```

challenge

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

   Solution:

```
// code example
const tanay = { mother: 'Kanak', age: 32 }
const tanvi = { ...tanay, age: tanay.age - 4 }

console.log(tanvi) // { mother: "Kanak", age: 28 }
console.log(tanay) // { mother: "Kanak", age: 32 }
tanay === tanvi // false

// What NOT to do
const tanay = { mother: 'Kanak', age: 32 }
const tanvi = tanay // make a copy of tanay for tanvi
tanvi.age = tanay.age - 4 // then subtract the age for her
console.log(tanvi.age) // 28
```

```
console.log(tanvi) // { mother: "Kanak", age: 28 }
console.log(tanay) // { mother: "Kanak", age: 28 }
// We see here that tanay's object also changed. We don't want this.
// This copy method does not work.
```

COPY

```
> const mySis = { ...anObj, age: anObj.age - 4 }
< undefined
> mySis
< ▶ {name: 'tanay', age: 28}
> anObj
< ▶ {name: 'tanay', age: 32}
> const myself = {...anObj}
< undefined
> myself
< ▶ {name: 'tanay', age: 32}
> anObj
< ▶ {name: 'tanay', age: 32}
> myself.age = 33;
< 33
> myself
< ▶ {name: 'tanay', age: 33}
> anObj
< ▶ {name: 'tanay', age: 32}
> const howDoWeCreateANewObj = { ...anObj }
```

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

Solution:

```
// code example

const colors = ['red', 'yellow', 'blue', 'green', 'orange']
const moreColors = [...colors, 'pink', 'purple']

colors === moreColors // false

console.log(colors) // ["red", "yellow", "blue", "green", "orange"]
console.log(moreColors) // ['red', 'yellow', 'blue', 'green', 'orange', 'pink', 'purple']

// What NOT to do

const colors = ['red', 'yellow', 'blue', 'green', 'orange']
```

```
const moreColors = colors
moreColors[5] = 'pink'
moreColors[6] = 'purple'
console.log(colors) // ['red', 'yellow', 'blue', 'green', 'orange', 'pink', 'purple']

console.log(moreColors) // ['red', 'yellow', 'blue', 'green', 'orange', 'pink', 'purple']
```

COPY

section 2: Pure functions

understanding

- For the same input, the output will always be the same

```
> function wishMeBirthday(name){ console.log(`Happy Birthday ${name}`)}
<· undefined
> wishMeBirthday("tanay")
  Happy Birthday tanay                                    VM3262:1
<· undefined
> wishMeBirthday("tanvi")
  Happy Birthday tanvi                                    VM3262:1
<· undefined
> wishMeBirthday("tanay")
  Happy Birthday tanay                                    VM3262:1
<· undefined
>
```

- Three rules

1. At least one argument
2. return a value or other function
3. Should not mutate any of its arguments —> Make a new copy

challenge

1. Write a function birthday() to take a person's name and age in an object and then increase the age by 1. Return the updated object.

```
const birthday = (person) => ({ ...person, age: person.age + 1 })

const person = { mother: 'Kanak', age: 32 }
const tanayAfterBirthday = birthday(person)
```

```
console.log(tanayAfterBirthday) // { mother: "Kanak", age: 33 }

person === tanayAfterBirthday // false
```

COPY

```
> const happyBirthday = ({ name, age }) => ({ name, age: age + 1 });
< undefined
> const happyBirthday2 = person => ({...person, age: person.age + 1
  });
< undefined
> happyBirthday({name:"tanay", age: 31})
< ▶ {name: 'tanay', age: 32}
> // happyBirthday2 is what we do.
< undefined
```

```
> const happyBirthday = ({ name, age }) => ({ name, age: age + 1 });
< undefined
> const happyBirthday2 = person => ({...person, age: person.age + 1
  });
< undefined
> happyBirthday({name:"tanay", age: 31})
< ▶ {name: 'tanay', age: 32}
> happyBirthday("name", 20) // not passing an object
```

Live Challenge

1. Write an ES6 function increaseStock() to take a products's name and quantity in an object and then increase the quantity by 5.

   Solution:

```
const increaseStock = (product) => ({
  ...product,
  quantity: product.quantity + 5,
})

const product = { name: 'ruled notebook', quantity: 20 }
const inventoryStock = increaseStock(product)
console.log(inventoryStock) // { name: "ruled notebook", quantity: 25 }
```

COPY

section 2: Most used utility functions

understanding

- Can take functions as arguments
- Return Functions
- Or both

exercise 01: map

understanding

```
> const printToConsole = data => console.log(data);
< undefined
> printToConsole("tanay")
  tanay                                                    VM5709:1
< undefined
> const colors = ["red", "blue", "green", "yellow", "orange"];
< undefined
> const printColorToConsole = color => console.log(`Color is:
  ${color}`)
< undefined
> printColorToConsole(colors[0])
  Color is: red                                            VM6069:1
```

```
> printColorToConsole(colors[1])
  Color is: blue                                           VM6069:1
< undefined
> // taken a function, and we are passing one value of an array
< undefined
> // starting from index 0
< undefined
> // const colorsMessage = ["Color is: red", "Color is: blue",
  "Color is: green"
< undefined
```

```
> const convertColorToColorMessage = color => `Color is: ${color}`
< undefined
> colorsMessage.push(convertColorToColorMessage(colors[0]))
< 1
> colorsMessage
< ▶ ['Color is: red']
> colorsMessage.push(convertColorToColorMessage(colors[1]))
< 2
> colorsMessage
< ▶ (2) ['Color is: red', 'Color is: blue']
> colorsMessage.push(convertColorToColorMessage(colors[2]))
< 3
> colorsMessage
< ▶ (3) ['Color is: red', 'Color is: blue', 'Color is: green']
> // to create a new array -- a function which takes one element at
  a time, and return a new value. And then we have to call that
  function with one element, starting from index 0.
< undefined
```

To return a new array with modification on items

```
// syntax
map(functionThatTakesOneElementAtATime) // (element) => { /* … */ }
map(orFunctionThatTakesOneElemenAtATimeAndIndex) // (element, index) => { /* … */ }
function(item,index)
// item ⇒ One item of the array
// index ⇒ index of the current item
// returns ⇒ new item for the new array
```

COPY

challenge

Live Code Example

1. Given an array of numbers, return a new array with square root of each number in it.

   Solution:

   ```
   // code example
   const numbers = [1, 4, 9]
   const roots = numbers.map((num) => Math.sqrt(num))
   console.log(roots) // [1, 2, 3]
   ```

   COPY

Live Challenges

1. Write an ES6 function that takes an array of numbers and returns an array with the square of each element using the map method.

   Solution:

```
const squareNumbers = (numbers) => numbers.map((number) => number * number)
const numbers = [1, 2, 3, 4, 5]
console.log(squareNumbers(numbers))
// Output:[1, 4, 9, 16, 25]
```

COPY

```
> const squareNum = num => num * num ;
< undefined
> squareNum(2)
< 4
> squareNum(9)
< 81
> const square = numbers => numbers.map(squareNum)
< undefined
> square([1,2,3,4,5])
< ▶ (5) [1, 4, 9, 16, 25]
```

2. Write an ES6 function that takes an array of strings and returns an array with the length of each string using the map method.

   Solution:

```
const getLength = (stringsArray) => stringsArray.map((str) => str.length)
const stringsArray = ['neoG', 'coding', 'programming']
console.log(getLength(stringsArray))
// Output: [4, 6, 11]
```

COPY

```
> const wordLength = word => word.length
< undefined
> wordLength("tanay")
< 5
> wordLength("neoG")
< 4
> const getLength = words => words.map(wordLength)
< undefined
> getLength(["tanvi", "is", "awesome"])
< ▶ (3) [5, 2, 7]
>
```

will it work with objects?

Yes, this works with array of objects. But Not for objects.map

```
> const thisIsAnArrayOfObjects = [{name:"tanay"}, {name:"tanvi"},
  {name:"akanksha"}]
< undefined
> // ["tanay", "tanvi", "akanksha']
< undefined
> const extractNameFromObj = obj => obj.name
< undefined
> extractNameFromObj({name: "tanay"})
< 'tanay'
> thisIsAnArrayOfObjects.map(extractNameFromObj)
< ▶ (3) ['tanay', 'tanvi', 'akanksha']
```

exercise 02: filter

understanding

```
> const numbers = [1, 3, 5, 2, 22, 11,9]
< undefined
> const isOdd = num => num % 2 != 0
< undefined
> isOdd(3)
< true
> // true and false are boolean value
< undefined
> // but to use filter, the callback function should return boolean
< undefined
> const oddNumberOnly = numbers.filter(isOdd)
< undefined
> oddNumberOnly
< ▶ (5) [1, 3, 5, 11, 9]
> const onlyBooleans = numbers.map(isOdd)
< undefined
> onlyBooleans
< ▶ (7) [true, true, true, false, false, true, true]
> // a filter will return an array with the values for which the
  callback function returned true
< undefined
```

remove items from an array based on a specific condition.

```
// syntax
filter((element) => {
  /* … */
})
filter((element, index) => {
  /* … */
})
array.filter(function(item, index));

// item => the current element of the array.
// index => the index of the current element
// return true if the item should be included, false otherwise
```

COPY

challenge

Live Code Example

1. Given an array, return an array with only odd numbers in it.

Solution:

```
// code example
const numbers = [3, 5, 6, 1, 2]
const isOdd = (num) => num % 2 !== 0
const oddArr = numbers.filter(isOdd)
console.log(oddArr) // [3, 5, 1]
```

COPY

Live Challenges

1. Given an array, return an array with only numbers divisible by 10.

Solution:

```
const arr = [5, 20, 15, 40, 3, 30, 11]
const divisibleBy10 = (num) => num % 10 === 0
const filteredArray = arr.filter(divisibleBy10)
console.log(filteredArray)
// Output: [20, 40, 30]
```

COPY

2. Write an ES6 function that returns an array with no number less than 10 in it.

Solution:

```
const getNumbersGreaterThanTen = (arr) => arr.filter((num) => num > 10)
const arr = [2, 20, 3, 30, 4, 40, 50, 5]
console.log(getNumbersGreaterThanTen(arr))
// Output: [20, 30, 40, 50]
```

COPY

```
> const isDivisibleBy10 = num => num % 10 === 0
<- undefined
> isDivisibleBy10(5)
<- false
> isDivisibleBy10(90)
<- true
> const numbersDivisibleBy10 = numbers.filter(isDivisibleBy10)
<- undefined
> numbersDivisibleBy10
<- ▶ (3) [20, 40, 30]
> isDivisibleBy10(numbers[0]) ?
  numbersDivisibleBy10.push(numbers[0]) : null;
<- null
> isDivisibleBy10(numbers[1]) ?
  numbersDivisibleBy10.push(numbers[1]) : null;
<- 4
> const newArrayFor10s = []
<- undefined
> isDivisibleBy10(numbers[0]) ? newArrayFor10s.push(numbers[0]) :
  null;
```

exercise 03: find

understanding

used to find the first element in an array that satisfies a certain condition.

```
// syntax
find((element) => {
  /* … */
})
find((element, index) => {
  /* … */
})
array.find(function(element, index));
// element => the current element of the array.
// index => the index of the current element
// returns the first element in the array that satisfies the condition, otherwise undefined
```

COPY

challenge

Live Code Example

1. Write an ES6 function that takes an array, and returns the first even number in the array.

Solution:

```
const numbers = [5, 12, 8, 13, 44]
const isEven = (element) => element % 2 === 0
const findEvenNumber = numbers.find(isEven)

console.log(findEvenNumber) // 12
```

COPY

Live Challenges

1. Write an ES6 function that takes an array of fruits and returns the first fruit that is of 10 characters or above.

   Solution:

   ```
   const fruitArray = ['apple', 'banana', 'cherry', 'watermelon', 'pineapple']
   const findLongFruit = (arr) => arr.find((str) => str.length >= 10)

   console.log(findLongFruit(fruitArray))
   // Output: "watermelon"
   ```

   COPY

2. Given an array of objects, find the object with name “clips”.

   Solution:

   ```
   const inventory = [
     { name: 'socks', quantity: 12 },
     { name: 'shirts', quantity: 10 },
     { name: 'clips', quantity: 5 },
   ]

   // Your code
   const result = inventory.find(({ name }) => name === 'clips')
   console.log(result) // { name: "clips", quantity: 5 }
   ```

   COPY

exercise 04: reduce

understanding

```
> const sumOfNumbers = (acc, curr) => { console.log(acc, curr);
  return acc + curr }
< undefined
```

```
> let init = 0;
< undefined
> init = sumOfNumbers(init, numbers[0])
  0 5                                                    VM634:1
< 5
> init
< 5
> init = sumOfNumbers(init, numbers[1])
  5 20                                                   VM634:1
< 25
> init
< 25
> init = sumOfNumbers(init, numbers[2])
  25 15                                                  VM634:1
< 40
```

```
// manual implementation
init1 = sumOfNumbers(init0, numbers[0])
init2 = sumOfNumbers(init1, numbers[1])
init3 = sumOfNumbers(init2, numbers[2])
```

COPY

used to "reduce" an array to a single value based on an accumulator and the items in the array.

```
// syntax
const callbackFn = (accumulator, currentValue) => {
  /* … */
}
reduce(callbackFn)
reduce((accumulator, currentValue) => {
  /* … */
}, initialValue)
array.reduce(function(accumulates, currentValue), initialValue)

// accumulator => coming from the previous run
// currentValue => the current element of the array being processed.
// initialValue (optional) => the initial value of the total parameter. If this is not spec
// returns the updated value of the accumulator
```

COPY

challenge

Live Code Example

1. Given an array, find the sum of all numbers in the array using reduce() method.

```
// code example
const array = [1, 2, 3, 4]
const sumOfNumbers = (accumulator, currentValue, currentIndex) =>
  accumulator + currentValue
const sum = array.reduce(sumOfNumbers, 0)
```

```
console.log(sum) // 10
```

COPY

Live Challenges

1. Given an array, find the sum of all odd numbers in the array using reduce() method.

   Solution:

```
const array = [1, 2, 3, 4, 5, 6, 7]
const sumOfOddInd = (accumulator, currentValue, currentIndex) =>
  currentValue % 2 !== 0 ? accumulator + currentValue : accumulator

console.log(array.reduce(sumOfOddInd, 0)) // 16
```

COPY

```
> // a function which given current sum of odd numbers and a number.
  <-- two arguments. This function should return sum of odd numbers
  with the number added if the number is odd.
< undefined
> const sumOfOdds = (currentSumOfOdds, currentNumberFromTheArray) =>
  currentNumberFromTheArray % 2 != 0 ? currentSumOfOdds +
  currentNumberFromTheArray : currentSumOfOdds
< undefined
> const numOddAndEven = [1,2,3,4,5,6,7]
< undefined
> let currentSumOfOdds = 0;
< undefined
> let currentSumOfOdds2 = sumOfOdds(currentSumOfOdds,
  numOddAndEven[0])
< undefined
> currentSumOfOdds2
< 1
> let currentSumOfOdds3 = sumOfOdds(currentSumOfOdds2,
  numOddAndEven[1])
< undefined
> currentSumOfOdds3
< 1
> numOddAndEven.reduce(sumOfOdds,0)
< 16
```

2. Write a function that takes an array of objects representing books with properties title, author, and pages, and returns the total number of pages of all the books using the reduce function.

```
const books = [
  { title: 'The Alchemist', author: 'Paulo Coelho', pages: 197 },
  { title: 'To Kill a Mockingbird', author: 'Harper Lee', pages: 281 },
  { title: 'The Great Gatsby', author: 'F. Scott Fitzgerald', pages: 180 },
]
```

```
const getTotalPages = (books) =>
  books.reduce((totalPages, book) => totalPages + book.pages, 0)

console.log(getTotalPages(books)) // Output: 658
```