# Assignment Seven

## Please read:

- This is your seventh assignment which you need to submit on the LMS. This is an assignment on Testing.
- Work on the questions given below and be ready with your solution in CodeSandbox. You have to submit your CodeSandbox link below on this page.
- Late Submission: The deadline to submit this assignment is 19th February 2024, 6:00PM IST.

## Important Instructions:

1. Make sure that you follow the rules of unit testing and write separate tests for each test case. This will help you build your testing muscle.
2. Do not copy from someone else as that would be cheating.

## challenge

https://codesandbox.io/s/testing-assignment-8xdfkk

## map utility exercises

### 1. capitalizeValues

Write test cases for a function called `capitalizeValues`, which takes an array of strings as input and returns a new array with each string capitalized.

| Test Case | Input | Expected Output |
|---|---|---|
| Capitalize all strings in the array | ['apple', 'banana', 'cherry'] | ['Apple', 'Banana', 'Cherry'] |
| Handle empty input array | [] | [] |
| Capitalize strings with spaces | ['hello world', 'goodbye space'] | ['Hello world', 'Goodbye space'] |

| Test Case | Input | Expected Output |
| --- | --- | --- |
| Original array remains unchanged | ['abc', 'def'] | (Original array remains unchanged) |
| Capitalize strings with special characters | ['!@#', '$%^'] | ['!@#', '$%^'] |
| Capitalize and check mixed case strings | ['loRem', 'IpSum', 'DoLor'] | ['Lorem', 'Ipsum', 'Dolor'] |

```
export function capitalizeValues(arr) {
  return arr.map((str) => {
    const firstChar = str.charAt(0).toUpperCase()
    const restOfStr = str.slice(1).toLowerCase()
    return firstChar + restOfStr
  })
}
```

COPY

### 2. findDuplicates

Write test cases for a function called `findDuplicates`, which takes an array of numbers and returns an array containing all the duplicate numbers present in the input array.

| Test Case | Input | Expected Output |
| --- | --- | --- |
| Find duplicates in an array | [1, 2, 3, 2, 4, 5, 3, 6, 7] | [2, 3] |
| Handle array with no duplicates | [9, 8, 7, 6, 5, 4, 3, 2, 1] | [] |
| Find duplicates with negative numbers | [1, -2, 3, -2, 4, 5, 3, -6, 7] | [-2, 3] |
| Find duplicates with floating-point numbers | [1.5, 2.3, 1.5, 4.8, 2.3] | [1.5, 2.3] |
| Handle empty input array | [] | [] |

```
export function findDuplicates(arr) {
  const uniqueNumbers = new Set()
  const duplicates = new Set()

  for (const num of arr) {
    if (uniqueNumbers.has(num)) {
      duplicates.add(num)
    } else {
      uniqueNumbers.add(num)
    }
  }

  return Array.from(duplicates)
}
```

COPY

## filter utility exercises

### 1. filterUniqueStrings

Write tests for `filterUniqueStrings`, a function that takes an array of strings as input and returns a new array containing only the unique strings (removing duplicates).

| Test Case | Input | Expected Output |
| --- | --- | --- |
| Filters out duplicate strings | ['apple', 'banana', 'cherry', 'apple', 'date', 'banana'], | ['apple', 'banana', 'cherry', 'date'] |
| Handles empty input array | [], | [] |
| Handles input with all unique strings | ['one', 'two', 'three'], | ['one', 'two', 'three'] |
| Handles input with one repeated string | ['apple', 'apple', 'apple'], | ['apple'] |
| Checks if the output array contains only unique strings | ['apple', 'banana', 'cherry', 'apple', 'date', 'banana'], | All elements should be unique |
| Checks if the output array length is correct | ['apple', 'banana', 'cherry', 'apple', 'date', 'banana'], | Should have length 4 |
| Checks if the function throws an error with invalid input | 'invalid', | Should throw an error |

```
function filterUniqueStrings(strings) {
  return strings.filter((str, index, arr) => arr.indexOf(str) === index)
}
```

COPY

## 2. filterByProperty

Write tests for `filterByProperty`, a function that takes an array of objects and a property name as input, and returns a new array containing only the objects that have the specified property.

| Test Case | Input |
| --- | --- |
| Filters objects with specified property | `[{ name: 'apple', color: 'red' }, { color: 'yellow' }, { name: 'cherry', color: 'red'` 'name' |
| Handles empty input array | `[]`, 'name' |
| Handles objects with property having falsy values | `[{ name: 'apple', color: 'red' }, { size: 0 }, { name: 'cherry', color: 'red' }]`, 'si |
| Checks if the output array contains only objects with the specified property | `[{ name: 'apple', color: 'red' }, { color: 'yellow' }, { name: 'cherry', color: 'red'` 'name' |
| Checks if the | 'invalid', 'name' |

| Test Case | Input |
| --- | --- |
| function throws an error with invalid input | |

```
function filterByProperty(objects, property) {
  return objects.filter((obj) => obj.hasOwnProperty(property))
}
```

COPY

# reduce utility exercises

## 1. sumNestedArrays

Write tests for `sumNestedArrays`. It takes a nested array of numbers and returns the sum of all the numbers using the `reduce` method.

| Test Case | Input | Expected Output |
| --- | --- | --- |
| Sum numbers in a nested array | [[1, 2], [3, 4, 5], [6]] | 21 |
| Sum numbers in an empty nested array | [] | 0 |
| Ensure original array remains unchanged | [[1, 2], [3, 4, 5], [6]] | (Original array remains unchanged) |
| Check if the output is a number | [[1, 2], [3, 4, 5], [6]] | Should be a number |

```
function sumNestedArrays(arr) {
  return arr.reduce(
    (sum, curr) => sum + curr.reduce((innerSum, num) => innerSum + num, 0),
    0,
  )
}
```

COPY

## 2. calculateFactorial

Write test cases for `calculateFactorial`. It takes a positive integer as input and returns the factorial of that integer using the `reduce` method.

Factorial of a non-negative integer `n` is the product of all positive integers less than or equal to `n`.

| Test Case | Input | Expected Output |
| --- | --- | --- |
| Calculate factorial of 0 | 0 | 1 |
| Calculate factorial of 1 | 1 | 1 |
| Calculate factorial of 5 | 5 | 120 |
| Calculate factorial of 10 | 10 | 3628800 |
| Check if the output is a number | 5 | Should be a number |
| Handle negative input | -5 | Should throw an error |

```
function calculateFactorial(n) {
```

```
  if (n < 0) {
    throw new Error('Input must be a non-negative integer.')
  }
  return n === 0
    ? 1
    : Array.from({ length: n }, (_, i) => i + 1).reduce(
        (product, num) => product * num,
        1,
      )
}
```

COPY

# find utility exercises

## 1. findLongestWord

Write test cases for `findLongestWord`. This function takes a string as input and returns the longest word from the string using the `split` method to break the string into an array and the `find` method to find the longest word.

| Test Case | Input | Expected Output |
| --- | --- | --- |
| Find longest word in a sentence | "The quick brown fox jumps over the lazy dog." | "quick" |
| Find longest word in a sentence with a single word | "Hello" | "Hello" |
| Find longest word in an empty string | "" | undefined |
| Check if the output is a string | "The quick brown fox jumps over the lazy dog." | Should be a string |

```
export const findLongestWord = (sentence) => {
  if (sentence.trim() === '') {
    return undefined
  }

  const words = sentence.split(' ')
  return words.find(
    (word) => word.length === Math.max(...words.map((word) => word.length)),
  )
}
```

COPY

## 2. findLastNegativeNumber

Write tests for `findLastNegativeNumber`. This function takes an array of numbers as input and returns the last negative number from the array using the `reverse` method and the `find` method.

| Test Case | Input | Expected Output |
| --- | --- | --- |
| Find last negative number | [3, -7, -2, 9, -5] | -5 |

| Test Case | Input | Expected Output |
| --- | --- | --- |
| Find last negative number in an array with no negative numbers | [3, 7, 2, 9, 5] | undefined |
| Find last negative number in an array with decimal numbers | [3.5, -7.2, -2.1, 9.7, 5.3] | -2.1 |
| Check if the output is a number | [3, -7, -2, 9, -5] | Should be a number |
| Check if the function throws an error with invalid input | 'invalid' | Should throw an error |

```
function findLastNegativeNumber(arr) {
  return arr.reverse().find((element) => element < 0)
}
```

COPY

# ex03: testing reducer - exercises

## 1. Bookshelf Reducer with Categories:

Description: Write test cases for the `bookshelfReducer`

- `bookshelfReducer` manages a state representing a bookshelf containing an array of books and categories.
- It handles three types of actions: `"ADD_BOOK"`, `"ADD_CATEGORY"`, and `"ASSIGN_CATEGORY"`.
- For the `"ADD_BOOK"` action, a new book is added to the books array with the specified `id`, `title`, and `author`.
- For the `"ADD_CATEGORY"` action, a new category is added to the categories array with the specified `id` and `name`.
- For the `"ASSIGN_CATEGORY"` action, the specified book is assigned to the specified category.

```
const initialState = {
  books: [],
  categories: [],
}

function bookshelfReducer(state = initialState, action) {
  switch (action.type) {
    case 'ADD_BOOK':
      return {
        ...state,
        books: [
          ...state.books,
          {
            id: action.payload.id,
            title: action.payload.title,
            author: action.payload.author,
            category: null,
          },
        ],
      }
    case 'ADD_CATEGORY':
      return {
        ...state,
        categories: [
          ...state.categories,
```

```
          { id: action.payload.id, name: action.payload.name },
        ],
      }
    case 'ASSIGN_CATEGORY':
      return {
        ...state,
        books: state.books.map((book) =>
          book.id === action.payload.bookId
            ? { ...book, category: action.payload.categoryId }
            : book,
        ),
      }
    default:
      return state
  }
}
```

COPY

## 2. Social Media Posts Reducer:

Description: Write test cases for the `postReducer`

- `postReducer` manages a state containing an array of social media posts.
- It handles four types of actions: `"CREATE_POST"`, `"EDIT_POST"`, `"DELETE_POST"`, and `"LIKE_POST"`.
- For the `"CREATE_POST"` action, a new post object is added to the posts array with the specified `id`, `author`, `content`, and initial `likes` count.
- For the `"EDIT_POST"` action, the content of the specified post is updated.
- For the `"DELETE_POST"` action, the specified post is removed from the posts array.
- For the `"LIKE_POST"` action, the number of likes for the specified post is incremented by 1.

```
const initialState = {
  posts: [],
}

function postReducer(state = initialState, action) {
  switch (action.type) {
    case 'CREATE_POST':
      const newPost = {
        id: action.payload.id,
        author: action.payload.author,
        content: action.payload.content,
        likes: 0,
      }
      return {
        ...state,
        posts: [...state.posts, newPost],
      }
    case 'EDIT_POST':
      return {
        ...state,
        posts: state.posts.map((post) =>
          post.id === action.payload.id
            ? { ...post, content: action.payload.newContent }
            : post,
        ),
      }
```

```
    case 'DELETE_POST':
      return {
        ...state,
        posts: state.posts.filter((post) => post.id !== action.payload.id),
      }
    case 'LIKE_POST':
      return {
        ...state,
        posts: state.posts.map((post) =>
          post.id === action.payload.id
            ? { ...post, likes: post.likes + 1 }
            : post,
        ),
      }
    default:
      return state
  }
}
```

COPY

## ex04: let's try TDD - exercises

📢 Make sure you write the test cases first.

### 1. Employee Management Reducer:

Description: Write test cases for the `employeeReducer`

- `employeeReducer` manages a state containing an array of employees with their names, roles, and salaries.
- It handles four types of actions: `"ADD_EMPLOYEE"`, `"UPDATE_SALARY"`, `"REMOVE_EMPLOYEE"`, and `"FILTER_EMPLOYEES"`.
- For the `"ADD_EMPLOYEE"` action, a new employee with the provided details is added to the employees array.
- For the `"UPDATE_SALARY"` action, the salary of an employee is updated.
- For the `"REMOVE_EMPLOYEE"` action, an employee is removed from the employees array.
- For the `"FILTER_EMPLOYEES"` action, employees are filtered based on their roles.

### 2. Event Booking Reducer:

Description: Write test cases for the `eventReducer`

- `eventReducer` manages a state containing a list of events with their names, dates, and attendees.

- It handles four types of actions: `"CREATE_EVENT"`, `"CANCEL_EVENT"`, `"ADD_ATTENDEE"`, and `"REMOVE_ATTENDEE"`.
- For the `"CREATE_EVENT"` action, a new event is added to the events list.
- For the `"CANCEL_EVENT"` action, an event is removed from the events list.
- For the `"ADD_ATTENDEE"` action, an attendee is added to the specified event.
- For the `"REMOVE_ATTENDEE"` action, an attendee is removed from the specified event.

  All the best. We hope you complete and submit your assignment in time.

  Click on the Share button on your CodeSandbox, then click on Copy link button and submit that link here in the submission form below. Make sure the access is public.