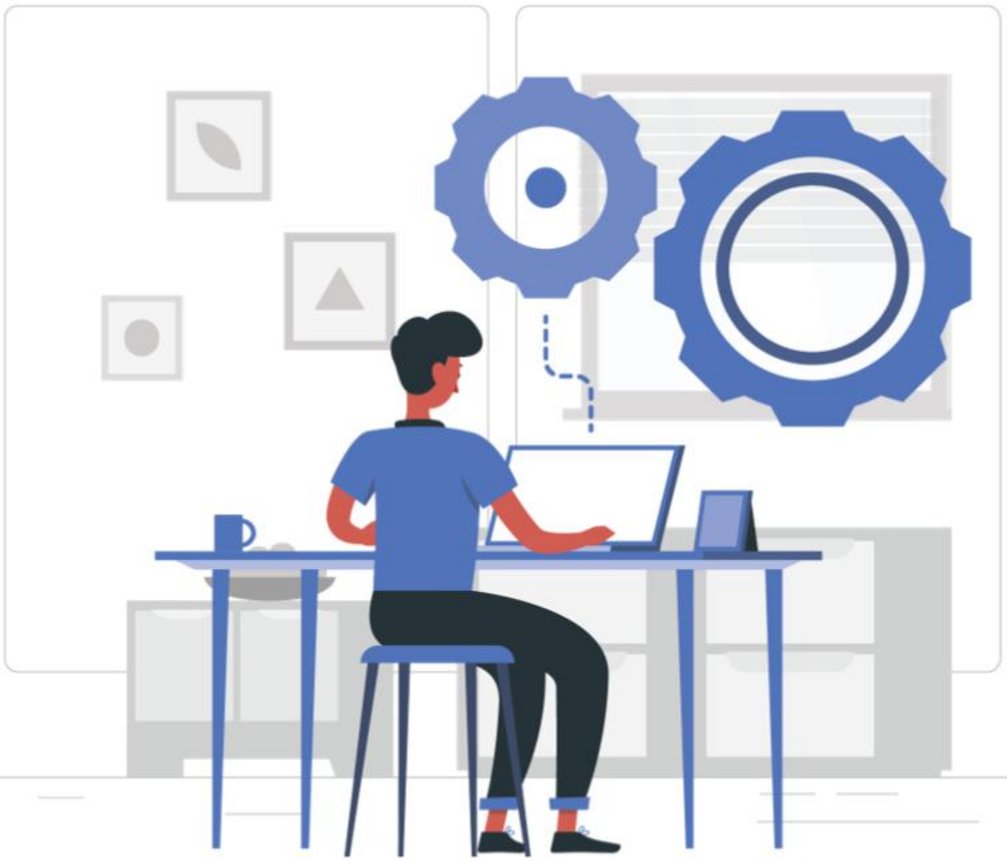

Practice
Implement Unit Testing Using Mocha and Chai

Practice

- Practice: Enable testing in the Solution to Calculate Area

Points to Remember

- Write and execute one test case at a time.
  - Proceed with the next case only if the currently written test code succeeds.
- In addition to the specified test requirements, if required, more test cases can be added, but it should be ensured that they pass before the solution is submitted.

Context

Ron has hens, ducks and cows in his barn that he wants to renovate. He thinks that creating fences to keep the animals separate will help him feed the animals more easily .

For this, he plans to divide the area into three parts. He wants to have a square area for the hens, a circular area for the ducks, and a rectangular area for the cows.

The solution for the problem exists but is not test enabled.

PRACTICE

Practice: Enable Testing in the Solution to Calculate Area

Write test cases to enable testing in the solution for calculating areas.

The test code should ensure the required functions:

1. exist
2. return expected results for the valid inputs provided
3. return error messages for the insufficient / invalid and incorrect inputs provided.

If required, modify the solution code to ensure all the test requirements are satisfied.

Steps

- Step 1: Open the file **`calculate-area-solution.js`** located inside the **`solution`** folder of the **`fe-c2-cs3-area-practice`** folder to copy the solution.
- Step 2: Copy the solution for this practice from the solution code developed for the `practice 1` of `Sprint 2 - Implement Modular Programming using Functions` into the **`calculate-area-solution.js`** file.
  - The solution should contain definitions of below functions to calculate area of square, rectangle and circle respectively:
    - `areaOfSquare()` - accepts one input for side value and returns area of square.
    - `areaOfRectangle()` - accepts two inputs for length and breadth values and returns area of rectangle.
    - `areaOfCircle()` - accepts one input for radius value and returns area of circle.

Steps (cont'd)

- Step 3: Switch to command terminal and set the path to the folder **`fe-c2-cs3-area-practice`**.
- Step 4: Run the command **`npm init -y`** to transform the current folder into a new npm based JavaScript code.
  - This command will create file **`package.json`**.
- Step 5: Run the command **`npm install mocha chai`** to install the testing dependencies `Mocha` and `Chai`.
  - This command lists `mocha` and `chai` as dependencies in the **`package.json`** file.
- Step 6: Inside **`package.json`** file, modify `test` property of the `scripts` object with value given below:

```
"scripts": {
    "test": "mocha"
}
```

Steps (cont'd)

- Step 7: Write test code inside **`calculate-area-spec.js`** file located inside the **`test`** folder of the **`fe-c2-s3-testing-practice`** folder to test the above functions and ensure they:
  - are defined.
  - return expected area value for the valid inputs provided.
  - return error message if insufficient parameters are passed to the function.
  - return error message if invalid type of values are passed to the function.
  - return error message if negative values are passed to the function.
- Step 8: Refactor the solution code inside **`calculate-area-solution.js`** file to ensure the functions return proper error messages for insufficient / invalid and incorrect inputs.

Note: The existing solution might not be handling errors and hence code needs to be refactored to add error handling logic.

- Step 9: Run the command **`npm run test`** from the command terminal to execute the test code.

Solution Code Refactoring

- To check for insufficient inputs:
  - Use if statement to ensure none of the parameters is equal to “undefined” value.
- To check for invalid input type:
  - Use if statement to ensure type of each of the parameters is “number”.
- To check for non-zero or non-negative inputs:
  - Use if statement to ensure value of none of the parameters is less than or equal to zero.
- Refer to the table shown in the upcoming slide that lists down the error messages expected to be returned by the functions tested.

Error Handling Requirements

Function Under Test	Test Condition	Expected Error Message
`areaOfCircle()`	Number of parameters should not be less than 1	Insufficient Inputs
	Parameter should be of type number	Invalid Input Type, Radius Should Be A Number !!
	Parameter should not be less than or equal to zero	Incorrect Input, Radius Cannot Be Zero or Negative !!
`areaOfSquare()`	Number of parameters should not be less than 1	Insufficient Inputs
	Parameter should be of type number	Invalid Input Type, Side Should Be A Number !!
	Parameter should not be less than or equal to zero	Incorrect Input, Side Cannot Be Zero or Negative !!
`areaOfRectangle()`	Number of parameters should not be less than 2	Insufficient Inputs
	Parameters should be of type number	Invalid Input Type, Length and Breadth Should Be Numbers !!
	Parameters should not be less than or equal to zero	Incorrect Input, Length and Breadth Cannot Be Zero or Negative !!

Expected Output

```
> mocha

 solution
   ✓ should have function `calculateAreaOfSquare()`
   ✓ should have function `calculateAreaOfCircle()`
   ✓ should have function `calculateAreaOfRectangle()`

 Function calculateAreaOfSquare
   ✓ should return area of square for the valid input provided`
   ✓ should return message insufficient inputs if less than 1 parameter is passed
   ✓ should return message invalid input type if the parameter is not a number
   ✓ should return message incorrect input if the value of the parameter is less than or equal to 0

Function calculateAreaOfCircle
   ✓ should return area of circle for the valid input provided`
   ✓ should return message insufficient inputs if less than 1 parameter is passed
   ✓ should return message invalid input type if the parameter is not a number
   ✓ should return message incorrect input if the value of the parameter is less than or equal to 0

Function calculateAreaOfRectangle
   ✓ should return area of rectangle for the valid inputs provided`
   ✓ should return message insufficient inputs if less than 2 parameters are passed
   ✓ should return message invalid input types if the parameters are not of type numbers
   ✓ should return message incorrect input if the value of the parameter is less than or equal to 0


15 passing (103ms)
```