### RIT Dubai – Fall 2024

### Activity 4: Test-Driven Development of a Polygon Class in Python

**Introduction:**  
In this assignment, you will develop a Python class called *Polygon* using a Test-Driven Development (TDD) approach. TDD is a software development process where tests are written before implementing functionality. This will help you think critically about each component of your code and ensure that it works as expected from the beginning.

### Learning Goals:

* Implement a class in Python with appropriate methods, attributes, and accessors.
* Use TDD to iteratively develop functionality.
* Write pytest tests to validate class methods and attributes.

### Instructions:

You will develop the *Polygon* class and its methods through a series of small steps, starting with basic class attributes, constructor, getters, and setters. For each task, start by writing tests to define the expected behaviour, and then implement the necessary functionality to pass those tests.

### Assignment Breakdown:

#### Step 1: Define Class Members, Constructor, Getters, and Setters (20 Points)

1. **Create a Class and Constructor**
   * **Task 1.1**: In polygon.py, define a class called Polygon.
   * **Task 1.2**: Add the following attributes:
     + name (string): Name of the polygon (e.g., "Triangle", "Square", "Hexagon").
     + sides (list of floats): A list representing the lengths of each side of the polygon (e.g., [3, 3, 3] for an equilateral triangle).
   * **Task 1.3**: Initialize these attributes in the \_\_init\_\_ method.

1. **Add Getters and Setters**
   * **Task 1.4**: Create getter and setter methods for each attribute:
     + get\_name and set\_name for name
   * get\_sides and set\_sides for sides

1. **Write and Run Tests for Initialization and Accessors**
   * **Task 1.5**: In test\_polygon.py, write a test function (test\_polygon\_initialization) to verify that a Polygon object is correctly initialized with values for name and sides.
   * **Task 1.6**: Add individual tests for each getter and setter method to ensure they return and modify the correct values.
   * **Task 1.7**: Run the tests to confirm everything is working as expected.

#### Step 2: Implement Equality (\_\_eq\_\_) and Inequality (\_\_ne\_\_) Methods (15 Points)

1. **Write Tests for Equality and Inequality**
   * **Task 2.1**: In test\_polygon.py, add tests for the \_\_eq\_\_ and \_\_ne\_\_ methods:
     + **Equality Test** (test\_polygon\_equality): Verify that two polygons with identical attributes (same name and same sides) are considered equal.
   * **Inequality Test** (test\_polygon\_inequality): Verify that polygons with different attributes (different names or different sides) are not equal.

1. **Implement Equality and Inequality Methods**
   * **Task 2.2**: Implement the \_\_eq\_\_ method in polygon.py to compare Polygon objects by name and sides.
   * **Task 2.3**: Implement the \_\_ne\_\_ method, which should return the opposite of \_\_eq\_\_.
   * **Task 2.4**: Run the tests and make sure they pass.

#### Step 3: Implement String Representation (\_\_str\_\_) Method (15 Points)

1. **Write a Test for the \_\_str\_\_ Method**
   * **Task 3.1**: In test\_polygon.py, write a test (test\_polygon\_str) to confirm the string representation of a polygon. Format it as:
     + "<name> with sides: <sides>" (e.g., "Triangle with sides: [3, 3, 3]")
2. **Implement the \_\_str\_\_ Method**
   * **Task 3.2**: Implement the \_\_str\_\_ method in polygon.py to match the specified format.
   * **Task 3.3**: Run the test to verify that it passes.

#### Step 4: Implement the calculate\_circumference Method (20 Points)

1. **Write Tests for Circumference Calculation**
   * **Task 4.1**: In test\_polygon.py, write tests for the calculate\_circumference method for specific polygons as sum of all sides (list of floats)
   * **Task 4.2**: Use pytest.approx for floating-point comparisons
2. **Implement the calculate\_circumference Method**
   * **Task 4.3**: In polygon.py, implement calculate\_circumference to return the circumference by summing the values in the sides list.
   * **Task 4.4**: Run the tests to confirm that the calculate\_circumference method works as expected.

#### Step 5: Create a Main Script (10 Points)

1. **Create a main Function to Demonstrate the Class**
   * **Task 5.1**: Inside polygon.py, define a main function that:
     + Instantiates three Polygon objects: a triangle, a square, and a hexagon with appropriate side lengths.
     + Prints each object’s string representation and calculated circumference.
   * **Task 5.2**: Add if \_\_name\_\_ == "\_\_main\_\_": at the bottom of polygon.py to run the main function when the script is executed directly.
2. **Grading Criteria:**

| **Component** | **Description** | **Points** |
| --- | --- | --- |
| **Class and Constructor** | Properly defined attributes, getters, and setters | 20 |
| **Equality and Inequality** | Implemented \_\_eq\_\_ and \_\_ne\_\_ methods | 15 |
| **String Representation** | Correctly formatted \_\_str\_\_ method | 15 |
| **Circumference Calculation** | Correct implementation of calculate\_circumference | 20 |
| **Main Script** | Demonstration of class usage | 10 |
| **Code Quality and Comments** | Code organization and clarity, github usage | 10 |
| **Presentation** | Verbal presentation of implementation | 10 |
| **Total** |  | **100** |

***Note: For late submission 10% will be deducted for each day. In case any AI tool is used, 100% of exam mark will be deducted. Each team member must present the code.***