CW12-2: Python HTTP Server - Assertion – Testing

1. **Python HTTP Server:**

   Design a Python HTTP server that implements a simple todo program. The server should handle GET and POST requests to manage a todo list.

   a) Implement a Python HTTP server that listens on a specific port.
   b) On a GET request to the root ("/") endpoint, the server should return the current todo list as a JSON-like response.
   c) The todo list should be initially empty.
   d) The response should include the appropriate headers to indicate the content type as "application/json".
   e) On a POST request to the root ("/") endpoint, the server should create a new todo item.
   f) The server should expect the todo item as a JSON payload in the request body.
   g) The todo item should have properties such as "id," "title," and "description".
   h) Append the new todo item to the existing todo list.
   i) Test your server using a tool like **curl** or **requests** module.
   j) Make a GET request to retrieve the current todo list and ensure it is initially empty.
   k) Make a POST request to create a new todo item with appropriate JSON payload. Make another GET request to verify that the new todo item is added to the list.

2. Unit Test

1) Design a class called `EmailValidator` that validates email addresses. The class should have a method called `is_valid(email)` that takes an email address as input and returns `True` if it is a valid email address and `False` otherwise. Write unit tests for the `is_valid` method.

2) Design a class called `StudentManager` that manages a list of student records. Each student record should contain a unique ID, name, and grade. The `StudentManager` class should have the following methods:

    a) `add_student(id, name, grade)`: Adds a new student record to the list.
    b) `remove_student(id)`: Removes a student record with the given ID from the list.
    c) `get_student_grade(id)`: Returns the grade of the student with the given ID.
    d) `is_student_exist(id)`: Checks if a student with the given ID exists inthe list and returns `True` or `False` accordingly.

3. Pytest

Design a class called `BankAccount` that represents a simple bank account. The class should have the following methods:

a) `deposit(amount)`: Takes an `amount` as input and adds it to the account balance.

b) `withdraw(amount)`: Takes an `amount` as input and subtracts it from the account balance. Ensure that the account has sufficient funds before allowing the withdrawal.

c) `get_balance()`: Returns the current balance of the account.

d) `transfer(amount, recipient_account)`: Takes an `amount` as input and transfers it from the current account to the `recipient_account`.

Write tests for each of these methods using the pytest framework to ensure their correctness. Consider different scenarios, such as depositing and withdrawing various amounts, checking the balance after each operation, handling insufficient funds for withdrawals, and validating successful transfers.

4. Assertion

1) Write a function called `get_student_grade(percentage)` that takes a percentage as input and returns the corresponding grade based on the following scale: 90-100 (A), 80-89 (B), 70-79 (C), 60-69 (D), 0-59 (F). Use assertions to validate the input and raise an error if the percentage is out of range.

2) Write a function called `calculate_circle_area(radius)` that calculates the area of a circle given its radius. Use assertions to validate the input and raise an error if the radius is negative.

3) Write a function called `calculate_average(numbers)` that takes a list of numbers as input and returns their average. Use assertions to validate the input and raise an error if the list is empty.