Python in Bioinformatics
Project 1: Creating a complement for a DNA sequence
Due date: on September 4, 2023

Create a Python project that involves creating a complement for a DNA sequence:
1. Understand the Problem:
- Familiarize yourself with the concept of DNA and the complement of a DNA sequence.
- Understand that the complement of a DNA sequence replaces each nucleotide with its complementary base (A with T, T with A, C with G, and G with C).

2. Set up the Project:
- Create a new folder for your project (preferably on GitHub but its optional)
- Open your preferred Python 3 IDE (preferably Jupyter on Anaconda)

3. Write the Complement Function:
- Define a function, such as `compute_complement(sequence)`, that takes a DNA sequence as input and returns its complement.
- Implement the logic to iterate through the sequence and replace each nucleotide with its complement.
- Ensure that the function handles both uppercase and lowercase nucleotides.
- Test your function with different DNA sequences to verify its correctness.

4. Take User Input:
- Prompt the user to enter a DNA sequence.
- Use `input()` to capture the sequence provided by the user.
- Store the user's input in a variable.

5. Call the Complement Function:
- Call the `compute_complement()` function and pass the user's input sequence as an argument.
- Capture the returned complement sequence in a variable.

6. Display the Result:
- Print the original DNA sequence and its complement to the console.
- Provide a clear message indicating that the complement has been computed successfully.

7. Test and Refine:
- Test your program with various DNA sequences, including both short and long sequences.
- Verify that the program produces the correct complement for each input.
- Refine your code if necessary, addressing any issues or bugs.

8. Add Documentation:
- Include comments in your code to explain the purpose and logic of each section.
- Write a README file that provides an overview of the project, explains how to run the program, and includes any necessary instructions.

9. Finalize and Submit:
- Review your code for any potential improvements or optimizations.
- Ensure that your code is well-structured, readable, and adheres to Python coding conventions (PEP

10)- Double-check that your project files are organized and well-documented.
- Submit your project, either as a code file or a zip archive. Name the project as such: **FirstName_FamilyName_001**

The following rubrics are for evaluating and grading this project:

1. Correctness of the Complement Function:
- Does the function produce the correct complement sequence for a given DNA sequence? (5pts)
- Are both the DNA sequence and its complement of the same length? (5pts)

2. Code Structure and Readability:
- Is the code well-structured and organized with appropriate function and variable names? (5pts)
- Are the code comments clear and descriptive, explaining the purpose and logic of the code? (5pts)
- Is the code properly indented and formatted, following Python's 3 style guidelines (PEP 8)? (5pts)

3. Function Design and Modularity:
- Does the code use function appropriately, with a separate function for computing the complement? (5pts)
- Is the function reusable, allowing for easy integration into other programs? (5pts)
- Does the code demonstrate good modular design principles, separating concerns and promoting code reusability? (5pts)

4. Error Handling and Robustness:
- Does the code handle potential errors, such as invalid input or unexpected edge cases? (5pts)
- Are appropriate error messages displayed to the user for better error handling and debugging? (5pts)

5. Efficiency and Performance:
- Is the code efficient in terms of time and space complexity? (5pts)
- Does the code avoid unnecessary computations or redundant operations? (5pts)
- Can the code handle large DNA sequences without significant performance degradation? (5pts)

6. Testing and Test Coverage:
- Has the code been thoroughly tested with different input cases, including edge cases? (5 pts)
- Are there test cases to verify the correctness of the complement function? (5 pts)
- Does the code provide adequate test coverage, ensuring that most scenarios are covered? (5 pts)

7. Documentation and Presentation:
- Is there a clear and concise documentation explaining the purpose and usage of the code? (5 pts)
- Does the project include a README file with instructions on how to use the program? (5 pts)
- Are any dependencies or external libraries clearly stated if applicable? (5 pts)

8. Extra Features or Enhancements (optional):
- Does the project include any additional features beyond the basic complement functionality? (5 pts)
- Are there any innovative or creative elements that add value to the project? (5 pts)