CSC148 - Object-Oriented Programming

Part 1: Counter class

1. Consider the implementation of a `Counter` class below:

```
class Counter:
    initial: int
    current: int
    increment: int

    def __init__(self, init: int, inc: int) -> None:
        """Initialize a new counter with the given <init> as initial number, and <inc> as increment value.
        """
        initial = init
        increment = inc
        current = initial
```

   If we try to do the following lines of code, we observe that this code raises an error.

```
>>> c = Counter(0, 2)
>>> c.initial
ERROR ...
```

   Explain two things:

   (i) What the initializer's implementation actually does.

   (ii) What error is raised when we run the above code, and why (i.e., be more specific than just `ERROR ...`).

2. Complete a correct `Counter` class in the file provided - `counter.py`.

3. Here's an incorrect implementation of another method we want to add to Counter:

```
def change_incrementor(self, new_inc: int) -> None:
    old_current = self.current
    old_initial = self.initial
    self = Tweet(old_current, new_inc)
    self.initial = old_initial
```

   When we print the code below, we do not seem to get the expected value!

```
>>> c = Counter(10, 2)
>>> c.change_incrementor(5)
>>> print(c.increment)  # Prints '2', not '5'
```

   Explain what the issue is. You may want to try drawing out memory model diagrams to back up your explanations.

4. Implement the correct `change_incrementor` method in your `counter.py` file

Part 2: Twitter classes

1. Read the documentation and complete the given methods in `Tweet` and `User` class in the given `tweet.py` file.
2. Suppose we want another `User` method that will record the fact that the user follows another user with a given userid. We'll follow a version of the *Function Design Recipe* to do so.
   (a) Write a header and docstring for this method. Don't forget to include type annotations for the parameters and return value.
   (b) Then, write down a sample call to this method. (Do *not* worry about how you'll implement it yet!) You can add this sample call as an example to your function's docstring.
   (c) Decide what attribute(s) you will use to store information about who this user follows. Then, update the class docstring and attribute type annotations (within the existing code) to record your decisions.
   (d) Finally, implement your new method according to its docstring.
3. Remember that a tweet's contents must be within 280 characters long. Add this in as a representation invariant within the appropriate class docstring.
4. Create an InvalidTweetException class, and add checks to raise the exception in the appropriate places within the existing classes.

Part 3: Tournament class

Consider the `Tournament` class in the given `tournament.py` file, that records game outcomes and reports statistics.

1. Get familiar with the instance attributes. What value should `t.team_stats` have after executing the code in the example use shown in the class docstring?
2. Implement the method `Tournament.record_game` according to its docstring.
3. Read the `Tournament` initializer method. Lines 2, 3 and 4 create and assign an initial value to attribute `teams`. Would it make any difference if instead we simply wrote `self.teams = teams` instead? Explain. (Consider drawing a memory model diagram, in case that might help!)
4. Implement the method `Tournament.best_percentage` according to its docstring.