Problem: Interval arithmetic

Using Python object oriented programming, write a class called `Interval` that represents a one-dimensional open interval on the real line. This main purpose of this class is to simplify overlapping continuous intervals. The code below should get you started but there are a lot of missing pieces that you will have to figure out.

The API should take a pair of integers as input and respond to the `+` operator such that

```
>>> a = Interval(1,3)
>>> b = Interval(2,4)
>>> c = Interval(5,10)
>>> a + b
Interval(1,4)
>>> b+c
[ Interval(2,4), Interval(5,10)]
```

- Note that in the case of non-overlapping intervals, the output should be a list of constituent `Intervals`. Keep in mind that these are **open** intervals. Specifically,

  ```
  >>> Interval(2,3)+Interval(1,2)
   [Interval(2,3), Interval(1,2)]
  ```

- Note that these do not produce a single interval because each interval is open (not closed). The interval endpoints can be negative also (e.g., `Interval(-10,-3)` is valid). The output **does not** have to be sorted.
- It's up to you to write the dunder functions for your object. If you do this right, you will have a very general solution to this problem.

This is where good object-oriented design pays off.

Note: Make sure to implement `__eq__` method below, to pass all the test cases in the grader.

Starter Code:

```
    # fill out the necessary methods shown below and add others if need be.

    class Interval(object):
        def __init__(self,a,b):
            """
            :a: integer
            :b: integer
            """
            assert a<b
            assert isinstance(a,int)
            assert isinstance(b,int)
            self._a = a
            self._b = b
         def __repr__(self):
            pass
         def __eq__(self,other):
            pass
         def __lt__(self,other):
            pass
         def __gt__(self,other):
            pass
         def __ge__(self,other):
            pass
         def __le__(self,other):
            pass
         def __add__(self,other):
            pass
```

Please put your Python code in a Python script file and upload it. Please retain your submitted source files! Remember to use all the best practices we discussed in class. You can use any module in the Python standard library, but third-party modules (e.g., Numpy, Pandas) are restricted to those **explicitly** mentioned in the problem description.

Tips:

- After you have submitted your file, do **not** use the browser back or reload buttons to navigate or open the page in multiple browser tabs, as this may cause your `attempts` to decrease unexpectedly. It may take up to thirty seconds for your code to be processed, so please be **patient**.

- If you find yourself back at the main page without any feedback or change in your `attempts` then it means that your code timed out or crashed in some unexpected way.
- Ensure that your development environment does not presume the existence of certain packages for the autograder. The autograder does not have anything other than the standard library and those third-party libraries **explicitly** named in the problem description.
- Do not leave extraneous statements in your code like test cases, print statements, or anything else besides what is needed to evaluate your submission because the the autograder will spend its limited time executing those lines, which may result in unexpected crashes or timeouts.

浏览... 0501.py Upload Python source code file

Correct! Back to assignments. functional points = 27 /27 and validation points = 13/13