Due: Oct. 14, 2022 at 11:55pm

Overview

This lab is an introduction into lab procedure and a review of Python. The purpose of this assignment is:

- An overview of using the `unittest` module in Python to test your code.
- Learn how to write test cases in Python.
- Learn how to use exceptions in Python.
- Learn how to submit your work using Gradescope.

You should implement the two functions (lcm and gcd) yourself. You may not use an external math library to compute them.

Description

Begin with the template code in `lab0.py`, which outlines a class named `mathOps`. `mathOps` stores two integers, `u` and `v`, and provides two functions to compute operations on those integers:

- `gcd` computes the greatest common divisor of `u` and `v`.
- `lcm` computes the least common multiple of `u` and `v`.

You need to write enough test cases to ensure that your code performs properly in a comprehensive set of situations. For example, listed below are some possible inputs to be used in the test cases. A few test cases are given to you in the file `test_lab0.py`. Please make sure you test your program runs properly for all the possible edge cases, including malicious inputs like strings or floating point numbers (instead of integers), extremely large numbers like 2^{100}, etc. You should make sure the program ends gracefully in these situations, throwing proper exceptions and not crashing.

Please implement Euclid's algorithm for computing the GCD. Your implementation should run in $O(\log(u+v))$ time. (This should automatically be the case, unless you do something unusually inefficient.) LCM can be done using GCD as a subroutine, so it will also run in $O(\log(u+v))$ time.

Example inputs and outputs:

u	v	gcd	lcm
29	31	1	899
0	45	45	Exception
45	0	45	Exception
5	Inf	Exception	Exception

The exceptions for the last example should indicate that one of the inputs is infinite. The exceptions for the two middle examples should indicate that the LCM is undefined in those cases.

You should not change the name of any of the classes, files, or existing methods. Arguments and return types for public methods should also be left unchanged.

Testing

We strongly suggest that while you create each individual class, you also write a robust set of tests. That is, if you write the **Stack** class, then immediately afterwards you should write tests for **Stack**. This makes it easier to isolate edge cases and helps prevent problems later on. **test_lab1.py** has some test cases to start you off but you should expand on the test file. Some things to keep in mind while making tests:

— `dequeue() / pop()` on an empty queue/stack should raise an exception.

— Capital letters and spaces are valid inputs.

At a minimum your code should be tested with the following minimal test set:

— pushing one item to an empty stack

— popping one item from an empty stack (should raise exception for `AttributeError`)

— popping one item from a stack with one item

— enqueueing one item to an empty queue

— dequeueing one item from an empty queue (should raise exception for `AttributeError`)

— `isPalindrome('TaCo CaT')` returns True

Files to Submit

You should submit the **lab0.py** with the filled in methods. You should also submit the **test_lab0.py** file with all the test cases that you had written to test your code. These two files should be compressed and uploaded in **.zip** format under the filename "submission.zip". Please make sure you select the **lab0.py** and **test_lab0.py** files and compress them. Do not compress the folder named submission directly which will result in another subfolder named submission being created after converting to zip format. The screenshot is an example link on how to do achieve this. Once you submit your files to Gradescope, you will get the results and the score for your assignment. If any of the test cases fail, you need to modify your code and resubmit until all the test cases pass in order to get full credit.

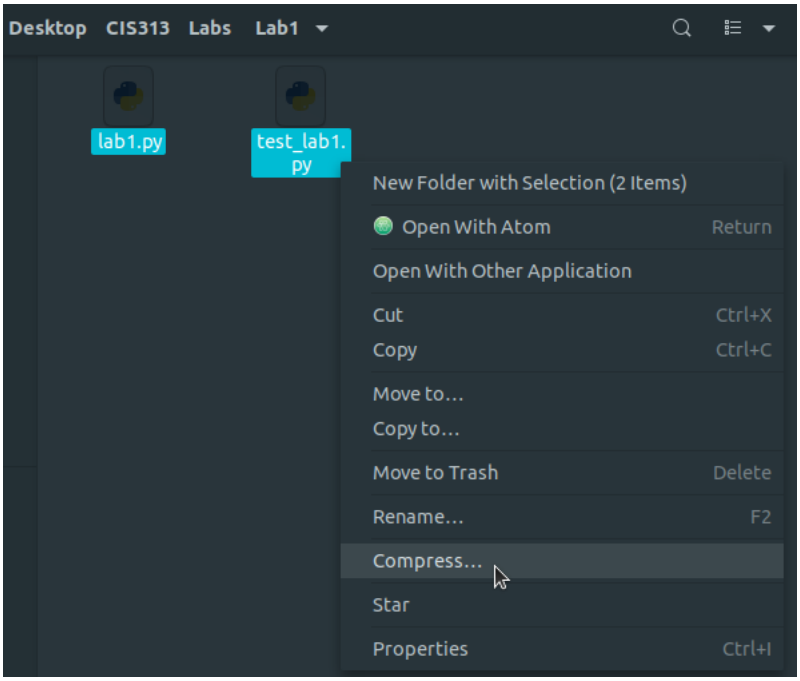
Grading

Typically, half of your grade will be determined by attempting to implement the correct data structure, and half will be on correctness and style (style will be discussed in labs). This assignment only has the correctness component. There will be 20 points possible for this assignment. Since this is the first assignment, I will try to provide as much assistance as possible during the labs and office hours to make sure you get full credit and all your doubts are clarified regarding submission and writing test cases.

Very important things to keep in mind:

Facts about LCM and GCD and things that I have assumed:

- GCD of 0 and 0 is 0



- GCD of 0 and any other number (say 45) is the number (45).
- The absolute value for negative numbers ($|-6| = 6$) should be used for calculating LCM or GCD.
- LCM of 0 and any other number is undefined (should raise an exception)
- LCM and GCD of infinity and any other number is undefined (should raise an exception)
- If floating point numbers are used, the ceiling value is used. (e.g.: 39.7 is rounded to 40)

Your code will be tested against the following invalid values:

- Strings
- Infinity
- Very large numbers
- Zero and negative numbers

So it is of paramount importance that you are aware of raising exceptions and using the `try` and the `except` block in Python. You should be able to raise some inbuilt exceptions like `OverflowError`, `TypeError`, `ValueError`, etc. in Python. The descriptions of the types of `Exceptions` can be found in `Exceptions_list.pdf`. You need not write your own exceptions for this assignment. This might seem challenging initially but we are there to provide as much help as possible. And we will do our best to ensure that you score full 20 points in this sample assignment. Please get as much assistance during the office hours as possible. Good luck!!

Note: Passing the test cases carries some points. And some points will be allocated for style. If you use proper variable names and your code is comfortable to read, you will probably get full points for style. Only in rare circumstances will you lose points for style.