Name: Evan Leglar	**Lab Time** Tuesday 12:00 PM

Names of people you worked with:
• N/A

Websites you used:
•
See comments section for each problem for sites used as reference

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	5 hrs

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.

Turn .zip files of Python code to Canvas or your assignment will not be graded

BEFORE YOU BEGIN

1) Make a new project for this lab

Learning Objectives:

You should be able to do the following:

- Write a function in Python
- Write an in-built test function

Thoughts (come back and read these after you've read the problems):

1. You should the Python value of pi in this lab. Use either `import math` or `from math import pi`.

2. We're going to be sticklers with grading for this lab. If your code doesn't run, or gives the wrong answer, you'll end up getting zero points for it. This might seem harsh, but we're trying to reinforce the importance of writing to a particular specification.

3. We haven't really talked about strings yet. However, you should be able to work out the syntax from the following example function, which prints out the characters in a string, one per line. Strings are iterables, and a for loop will go through them one character at a time. You'll probably want to use a for loop in your answer to the last question.

```
def foo(t):
    for c in t:
        print c
```

4. For all of the code you write for this lab, you should write your own testing code to try to make sure that the functions do the right thing. You might spend more time writing the testing code than the code for the functions themselves; this is OK. We're going to talk more about writing testing code during lectures, but you should try to think of some test cases for each of your functions, and write code to make sure that they do the right thing.

5. There are example tests in Files->Code->Labs->Lab 1->lab1_grader.py. You should check your code with this file. The tests in this file are NOT complete. Use them as examples for writing your own tests, but do NOT copy them verbatim into your code.

    1. A side note – this is a script that is meant to be used to **run** your code from a separate script, hence the `from volumes import cylinder_volume` line at the top of the test function. When you're writing a test function inside of your own code you don't need to do this because those functions are already defined.

6. The built-in Python function for greatest common denominator and the one you're going to write have the same name, which is a problem. You should use the `from ... import ... as ...` form of the import statement to help you out here.

7. To test your code, we're going to use a statement like from `gcd import gcd` to import your function. This should not run your tests, and your function should serve as a drop-in replacement for the Python `gcd` function.

8. We're intentionally not giving you a grading script for gcd for this lab, since we want you to try to write your own tests. For the other functions, you can find an incomplete grading script (not all cases are tested) in the files section.

Grading Checkpoints

1. A cylinder volume function that works on valid inputs [1 point], and deals with invalid inputs appropriately [1 point].
2. A torus volume function that works on valid inputs [1 point], and deals with invalid inputs appropriately [1 point].
3. A working `close` function [1 point].
4. A working letter-counting function [1 point].
5. A working greatest common denominator function [2 points]
6. Working test code for the greatest common denominator [1 point], using the Python `gcd` [1 point], and a large number of random number pairs [1 point], that does not run tests when the code is imported into other files [1 point].

Remember to hand in the .zip of the python files (including grader.py) into canvas and this file (as a PDF) to Gradescope.

Problem 1 volumes

1. We're going to start by taking the code you wrote for calculating the volume of a cylinder and putting it into a function, so that you can reuse it. Create a file called `volumes.py`, and write a function called `cylinder_volume` that takes a radius and a height (in that order) and returns the volume of the corresponding cylinder. In addition to the function, you should write some code to test that the function is returning the correct values. This can be done by printing the value returned by your function for arguments that you define. This code should go after the `if __name__ == '__main__':` line.

2. Modify your code so that it deals correctly with arguments that are incorrect. For volume calculations, this basically means negative values. If you get bad function arguments, then the function should raise an ValueError exception, like we showed in class.

3. Do the same for the torus code, putting it in the same file, in a function called torus_volume, which takes the minor radius and major radius (in that order) as arguments.

Comments for grader/additional information (if any)

There are two functions, cylinder_volume and torus_volume which take in 2 arguments each. The output can be found under the if __name == "__main__": code

volumes.py

```
#!/usr/bin/env python
from math import pi


def cylinder_volume(radius, height):
    if radius <= 0:
        raise ValueError
    if height <= 0:
        raise ValueError
    else:
        return pi * radius**2 * height


def torus_volume(r_inner, r_outer):

    r_maj = (r_outer + r_inner) / 2
    r_min = (r_outer - r_inner) / 2

    if r_outer < 0:
        raise ValueError
    if r_inner < 0:
```

```
        raise ValueError
    if r_maj < 0:
        raise ValueError
    if r_min < 0:
        raise ValueError
    else:
        return 2 * (pi ** 2) * r_maj * (r_min ** 2)


if __name__ == '__main__':
    print(cylinder_volume(4, 2))
    print(pi * 4**2 * 2)
    print(torus_volume(3, 4))
    print(17.27)  # volume of torus from torus.py tested values of
r_minor = 3, r_major = 4
```

Tests

```
For cylinder_volume, need to check that radius and height are
positive.
Then, check the function output against known values based on the
parameters.
```

Console output

```
/Applications/miniconda3/bin/python "/Users/evanleglar/Desktop/ME499
Python/Lab 1/volumes.py"
100.53096491487338
100.53096491487338
17.271807701906376
17.27

Process finished with exit code 0
```

Problem 2 close

Write a function called `close` in a file `close.py`, that takes three numbers as arguments. It returns `True` if the absolute difference between the first two numbers is less than the third number. For example `close(1, 2, 0.5)` would return `False`, but `close(1, 2, 3)` would return `True`.

Comments for grader/additional information (if any)

N/A

Tests

```
For this function, there are only two output options. Therefore, a
good test is to give the function arguments which are expected to
produce a desired output. To test this function, the arguments from
the examples above were used.
```

close.py

```
#!/usr/bin/env python


def close(num_1, num_2, diff):
    if abs(num_1 - num_2) <= diff:
        return True
    else:
        return False


if __name__ == '__main__':
    print(close(1, 2, 0.5))
    print(close(1, 2, 3))
```

Console Output

```
/Applications/miniconda3/bin/python
/Users/evanleglar/Desktop/untitled/close.py
False
True

Process finished with exit code 0
```

Problem 3 words

Write a function called `letter_count` in a file `words.py` that takes two string arguments. The first string is a bunch of text, and the second is a letter. The function returns the number of occurrences of the letter in the text. Your function should be case insensitive, that is: halLway should return 2 for ('halLway','l') and ('halLway','L').

Comments for grader/additional information (if any)

https://ispycode.com/Blog/python/2016-07/Count-case-insensitive-characters-in-a-string
This website was referenced for general structure

https://www.geeksforgeeks.org/python-program-count-upper-lower-case-characters-without-using-inbuilt-functions/
This website was referenced to better understand the .upper, .lower, and other string modifiers.

words.py

```
#!/usr/bin/env python


def letter_count(str1, str2):
    str1 = str1.lower()
    str2 = str2.lower()
    return str1.count(str2)


if __name__ == '__main__':
    str1 = 'helLoworLd'
    str2 = 'L'
    print("Total number of letter " + str2 + " is " +
str(letter_count(str1, str2)))
```

Tests

```
The tests created show how the output of the function does not
changes when counting total number of a specific letter vs. counting
only a capitalized character.
```

Console Output

```
/Applications/miniconda3/bin/python "/Users/evanleglar/Desktop/ME499
Python/Lab 1/words.py"
Total number of letter L is 3

Process finished with exit code 0
```

Problem 4 gcd

Write a function that calculates the greatest common divisor of two numbers, `gcd(a, b)`. The function should use Euclid's algorithm, take two numbers as input and return their greatest common divisor. There is already a Python function to do this, called `gcd`, but you are not allowed to use this in your function; we want you to write your own. You should, however, write some testing code that compares your answer to the Python `gcd` function to make sure it gives the same answer. To do this, you should write code that generates a pair of random numbers, then make sure your answer agrees with the built-in answer. Do this for some large number of pairs of numbers (say, 1000), and put this code underneath the `if __name__ == '__main__':` line in your file (which you should call `gcd.py`). Make sure your function is called `gcd` and Python's is called something else (see `from` foo `import as` blah)

Comments for grader/additional information (if any)

https://www.geeksforgeeks.org/gcd-in-python/

The above website was used to gain understanding of what gcd recursion functions might look like. Used as structure to fit the problem.

https://stackoverflow.com/questions/30890434/how-to-generate-random-pairs-of-numbers-in-python-including-pairs-with-one-entr/45387668

implemented the Euclidean gcd function found across the different sites into my code to compare with python gcd function

Tests

```
Ran a test comparing the output of the library gcd function vs. the
Euclidean one I implemented. The output of the test was a counter for
each comparison that resulted in the same gcd calculation.
```

gcd.py

```
#!/usr/bin/env python
from math import gcd as old_gcd
from random import randint


def gcd(a, b):
    while b:
```

```
        a, b = b, a % b
    return a


if __name__ == '__main__':

    counter = 0  # how many successful trials of new gcd function

    for i in range(0, 1001):
        a, b = randint(1, 1 * 10 ** 6), randint(1, 1 * 10 ** 6)

        if gcd(a, b) != old_gcd(a, b):
            print('Function Failed')
            counter = counter - 1
            raise ValueError

        else:
            counter = counter + 1
    print(str(counter) + str(' successful gcd computations'))
```

Console Output

```
/Applications/miniconda3/bin/python "/Users/evanleglar/Desktop/ME499
Python/Lab 1/gcd.py"
1001 successful gcd computations

Process finished with exit code 0
```