Lab 1 - Environment set up
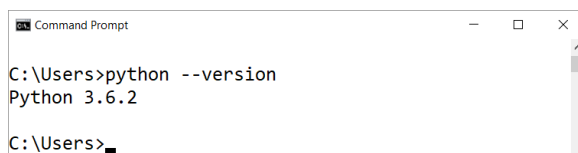Due Date: 08/30/2019 11:59 pm

Overview
This assignment will get you started with the basic tools needed to complete homework and some exercises. The main goal of this assignment is to ensure that you have correctly installed Python 3 in your computer. In addition, it will give you practice (or remind you) using a text editor to write and run Python programs.
IMPORTANT: You must use the command line and a text editor to edit, run, and debug your Python code.

Section 1. Setting Up Your Computer
You are a CMPSC 132 student who needs to install Python, configure it for command line use, and learn how to use a text editor to create and edit Python source code. You are allowed to use other Pyhton IDEs, however, all lectures, coding and review session execute code using this set up.

1. Download the latest Python 3 installer for your operating system at https://www.python.org/downloads.

2. Once Python is installed, you need to ensure that you can run it from the command line. To accomplish this, the python executable should be on your PATH. If you run Linux or Mac OS, it occurs automatically and you probably would not need help with this. If you run Windows, make sure you select the option "Add to PATH", otherwise you may have to do this:
   - If your shell is open, close it.
   - Go to the control panel, however you do that on your version of Windows.
   - Go to System and select "Advanced System Settings" where you will click on the button labelled "Environment Variables"
   - In the Environment Variables dialog find the Path variable under system variables (not user variables). Select it and click the "Edit" button below.
   - In the Edit dialog box, click on New and type the location of your Python installation (you accepted the defaults when you installed Python, make sure to read where Python would be installed, if you installed Python in a customized location, add that location to the PATH, i.e C:\Python3X\).

3. Check your Python installation. Mac users open a terminal and type `python3 --version`, Windows users open the command prompt (*cmd*) and at the command prompt type `python --version`. You should get a response like `Python 3.6`. To ensure Python was sucessfully added to the PATH variable, type python (python3 for Mac users)

```
C:\Users>python --version
Python 3.6.2

C:\Users>
```

```
C:\Users>python
Python 3.6.2 |Anaconda, Inc.| (default, Sep 19 2017, 08:03:39) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

4. Download and install a programmer's text editor. You may end up trying out several over the course of the semester before you settle on one. SublimeText (http://www.sublimetext.com/) is a good choice for beginners (recommended). Emacs is a popular choice among young programmers (http://www.gnu.org/software/emacs/), but it takes time to get used to it.

5. From the terminal (command prompt in Windows), create a directory for your coursework somewhere in your hard disk (i.e. cmpsc132) using the command `mkdir cmpsc132`. Enter the new folder using the command `cd cmpsc132` and create a subdirectory named `lab1`. Mac users can use the command `pwd` to get the path of the current working directory

```
Command Prompt                                          —    □    ×
C:\Users\Griselda>cd desktop

C:\Users\Griselda\Desktop>mkdir cmpsc132

C:\Users\Griselda\Desktop>cd cmpsc132

C:\Users\Griselda\Desktop\cmpsc132>mkdir lab1

C:\Users\Griselda\Desktop\cmpsc132>_
```

6. Enter to the *lab1* folder and create the file *output.txt* (Mac users write '> output.txt', Windows users write 'type nul > output.txt', no quotes)

```
Command Prompt                                          —    □    ×
C:\Users\Griselda\Desktop\cmpsc132>cd lab1

C:\Users\Griselda\Desktop\cmpsc132\lab1>type nul > output.txt

C:\Users\Griselda\Desktop\cmpsc132\lab1>_
```
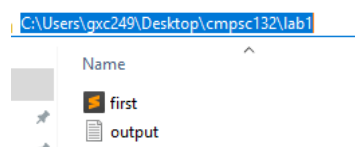
Section 2. Python Program

1. Open your text editor and create a file in your *lab1* directory named `first.py` and enter the Python program given bellow:

```python
def square(x):
    y = x * x
    return y

toSquare = 10
result = square(toSquare)
print("The result of", toSquare, "squared is", result)
```

2. Save the file. On the terminal (command line), type:
Mac users: `python3 first.py`
Windows users: `python first.py`
to run the program and see its output. Make sure you are in your *lab1* directory.

```
C:\Users\gxc249\Desktop\cmpsc132\lab1
        Name
     S  first
        output
```

3. Add the output of your program to the file *output.txt* by typing:

    python3 first.py >> output.txt (Mac)

    python first.py >> output.txt (Windows)

Go to your lab1 folder, you should have a txt file named output with the string 'The result of 10 squared is 100'

NOTE:

Mac users must run their code on the terminal using 'python3'. Using 'python' will execute the code using Python 2.7 and your code might not get full credit when graded.

Section 3. Submission's Format

1. Read the file print() vs return available in the LAB1 Assignment on Canvas.
2. Download the starter code file from the LAB1 Assignment on Canvas. Do not change the function names or given started code on your script
3. A doctest is provided as an example of code functionality. Getting the same result as the doctest does not guarantee full credit (See Appendix for a quick introduction to the doctest module). You are responsible for debugging and testing your code with enough data.
4. Each function must return the output (Do not use print in your final submission, otherwise your submissions will receive a -1 point deduction)
5. Do not include test code outside any function in the upload. Printing unwanted or ill-formatted data to output will cause the test cases to fail. Remove all your testing code before uploading your file (You can also remove the doctest).
6. Do not include the input() function in your submission.

[GOAL] Using your text editor, write the function *sumSquares*(aList). The function takes a list as an input and returns (not prints) the sum of the squares of the all the numbers which absolute value is divisible by 3. If an element in the list is not a number, the function should ignore the value and continue. When the function receives an input that is not a list, code should return an error message to the user. Hints: review the type() or isinstance() methods

Grading: For this assignment only, Gradescope will test your code with 5 different inputs for 2 points each. This will happen right after you upload your file. Make sure you follow the given instructions, otherwise, Gradescope will not show you the grading report

Deliverables

- Submit your code in a file named LAB1.py to the Lab1 GradeScope assignment before the due date, then mark the assignment as done on Canvas.

Appendix. Doctest Introduction

The doctest module is a lightweight testing framework that comes prepackaged with Python. This module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown.

We demonstrate this way of proceeding with the following simple example. We have the function countOdd, that takes a list and computes how many odd numbers are present in the list

```python
1  def countOdd(num_list):
2      odd_count = 0
3      for num in range(len(num_list)):
4          if num_list[num] % 2 != 0:
5              odd_count+= 1
6
7      return odd_count
```

The first step to setting up doctests is to write examples as interactive shell sessions into the docstrings in your module sing triple quotes. Here, countOdd has three examples given:

```python
1  def countOdd(num_list):
2      '''
3      >>> countOdd([1,1,1,2,2,5,5,7])
4      6
5      >>> countOdd([2,4,6,8])
6      0
7      >>> countOdd([1.5,2,3,3,8])
8      3
9      '''
10     odd_count = 0
11     for num in range(len(num_list)):
12         if num_list[num] % 2 != 0:
13             odd_count+= 1
14
15     return odd_count
```

To run the tests, we use doctest as the main program via the -m option to the interpreter like this:

```
> python -m doctest <file_name.py>
```

or

```
> python -m doctest -v < file_name.py >
```

The '-v' means verbose. Verbose is real handy when testing your doctests, since doctest doesn't output anything if all of the tests pass.

```
C:\Users\Griselda\Desktop>python -m doctest sumNum.py


C:\Users\Griselda\Desktop>
```


```
C:\Users\Griselda\Desktop>python -m doctest -v sumNum.py
Trying:
    countOdd([1,1,1,2,2,5,5,7])
Expecting:
    6
ok
Trying:
    countOdd([2,4,6,8])
Expecting:
    0
ok
Trying:
    countOdd([1.5,2,3,3,8])
Expecting:
    3
ok
1 items had no tests:
    sumNum
1 items passed all tests:
   3 tests in sumNum.countOdd
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
```