CMPSC-132: Programming and Computation II

Lab 0 - Environment set up

Due Date: 08/28/2021 11:59 PM

Goal: This assignment will first get you started with the tools we will use to program in this course. Additionally, we want you to get familiar with our grading tool, Gradescope.

General instructions:

- The work in this assignment must be your own original work and be completed alone.
- The instructor and course assistants are available on Teams and with office hours to answer any questions you may have. You may also share testing code on Teams.
- A doctest is provided to ensure basic functionality and may not be representative of the full range of test cases we will be checking. Further testing is your responsibility.
- Debugging code is also your responsibility.

Assignment-specific instructions:

- Read the print() vs return PDF file on the LAB 0 Canvas Assignment before starting this lab.
- A starter code file is provided on Canvas, download it before starting the assignment.
- Do not include the input() function in your submission (this applies to all assignments).

Submission format:

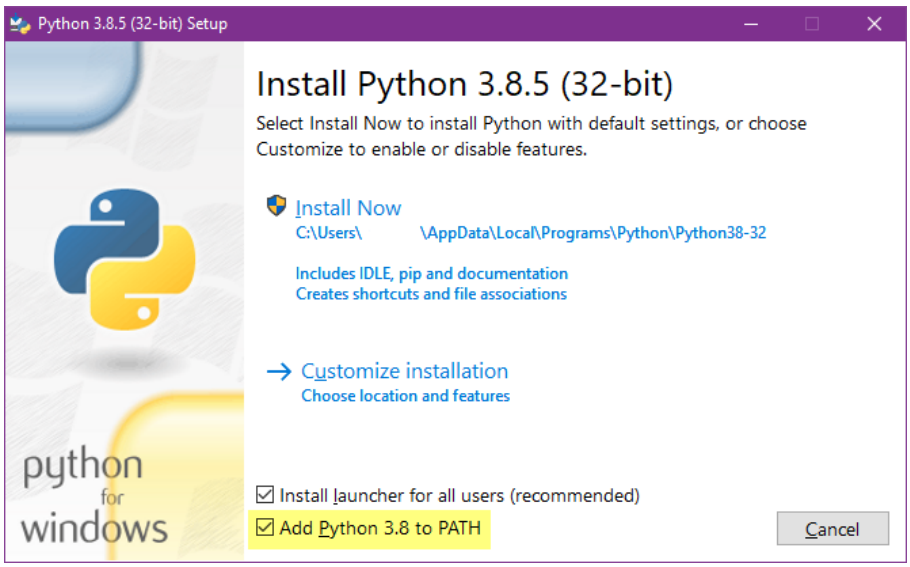
- Submit your sumSquares code from Section 2 in a file named LAB0.py to the Lab 0 Gradescope assignment before the due date.
- Afterwards, verify your Gradescope submission score and resubmit if necessary.

Section 1: Setting up your computer

You can use any programming environment you wish in this course as long as it runs Python 3.6 or later versions. We will set up the "default environment" in this section, which is what all lectures, review sessions, etc. will use.

Installing Python:

1. Download the latest version of Python 3 from https://www.python.org/downloads.
2. Install Python to your computer, selecting "Add to PATH" if on a Windows machine.



3. Open a terminal window. On MacOS, open the Terminal application. On Windows, open a command prompt window (search for `cmd`).
4. Verify the Python installation by typing `python --version`. For Mac users, please use `python3` instead of `python` (otherwise, you will be running Python2 instead of Python3).
5. If you see a response like Python 3.8.5, then Python has been installed successfully. If you don't see a response like that, try uninstalling Python and follow these steps again.

Writing and Executing Python code:

1. Download and install a programming text editor. There are many text editors, so you might end up trying a couple of them before you decide on one. In our class, some of the popular choices among students are Visual Code, Atom, Sublime Text, and replit (online based, no downloads, configurations or setups needed). Note that if you decide to use replit with

GitHub integration, your account must keep your repositories private. Keeping repositories public is a direct violation of academic integrity.

2. Create a folder for this class, preferably named cmpsc132.
3. With your text editor, create a new file named `first.py` and save it to that folder. Enter the following Python program below:

```
def square(x):
    y = x * x
    return y
toSquare = 10
result = square(toSquare)
print("The result of", toSquare, "squared is", result)
```

4. Open a terminal window and navigate to your cmpsc132 folder.
    a. Use `ls` (Mac) or `dir` (Windows) to see a list of folders from the current location.
    b. Use `cd` <foldername> to navigate down to a folder. *The cd command stands for change directory*
    c. If a folder name contains a space, surround the folder name with double quotes.
    d. Use `cd ..` to move back up a folder. (see below for an example of navigation)
5. Run your program by running the command `python first.py` (on Mac, use python3)
6. Send the output of your program to a file with the command `python first.py > output.txt` (on Mac, use python3)
7. Verify that the content of `output.txt` is the string “The result of 10 squared is 100”

Example target: D:\OneDrive\2020-2021 Fall\cmpsc132

```
D:\>dir

 Directory of D:\

08/04/2020  11:47    <DIR>          OneDrive
other directories hidden

D:\>cd OneDrive

D:\OneDrive>dir

 Directory of D:\OneDrive

08/04/2020  11:47    <DIR>          .
08/04/2020  11:47    <DIR>          ..
08/10/2020  19:24    <DIR>          2020-2021 Fall
other directories hidden

D:\OneDrive>cd "2020-2021 Fall"

D:\OneDrive\2020-2021 Fall>dir

 Directory of D:\OneDrive\2020-2021 Fall

07/10/2020  19:24    <DIR>          .
07/10/2020  19:24    <DIR>          ..
08/05/2020  19:11    <DIR>          cmpsc132
other directories hidden

D:\OneDrive\2020-2021 Fall>cd cmpsc132

D:\OneDrive\2020-2021 Fall\cmpsc132>
```

Section 2: Assignment Submission

Now that you have successfully installed Python, you will use your text editor to implement the function *sumSquares*(*aList*), run a doctest, and submit your code to Gradescope.

Please note that for this assignment only, you will be able to see all the test cases your submission will be tested with on Gradescope (visible after submitting). You should be able to see the grading report after uploading your code to Gradescope. In future assignments, the visible test cases will not be representative of the final test cases. If the grading report is showing errors, it means you need to go back to your code and debug it before resubmitting.

sumSquares(*aList*): Sums the squares of all numbers in aList that are greater than 5 but less than 500. If an element in the list is not a number, simply ignore that value and continue. Do not make any assumptions about the contents of aList.

Preconditions:

Input		
list	*aList*	List might be empty and could contain data types other than *int*

Output	
int/float	The sum of the squares of the positive numbers > 5 and < 500

Error output	
None	Nothing is returned if the input is not a list. Note that the keyword None is not the string 'None'

Hints: review the type() or isinstance() methods

```
type(5.6) == float   returns True

isinstance(5.6, float)    returns True
```

Appendix: Doctests

The doctest module is a lightweight testing framework that comes prepackaged with Python. We will demonstrate how to set one up and how to execute a doctest. The doctest looks like what an interactive shell session would look like, but it is surrounded by triple quotes so that it doesn't get executed as Python code. Here we have three example tests:

example.py

```
def countOdd(num_list):
    """
        >>> countOdd([1,1,1,2,2,5,5,7])
        6
        >>> countOdd([2,4,6,8])
        0
        >>> countOdd([1.5,2,3,3,8])
        3
    """
    odd_count = 0

    for num in range(len(num_list)):
        if num_list[num] % 2 != 0:
            odd_count += 1

    return odd_count
```

To run these tests in the terminal, we need to call the module (`-m`) named `doctest`.

```
D:\OneDrive\2019-2020 Fall\cmpsc132>python -m doctest example.py
```

To run these tests in verbose mode, we can pass in the verbose (-v) option to our command.

```
D:\OneDrive\2019-2020 Fall\cmpsc132>python -m doctest -v example.py
```

Without running it in verbose mode, there will be no output if all test cases pass.

Appendix: Doctests (continued)

Running doctests in both non-verbose and verbose mode

```
D:\OneDrive\2020-2021 Fall\cmpsc132>python -m doctest example.py

D:\OneDrive\2020-2021 Fall\cmpsc132>python -m doctest -v example.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:
    example
1 items passed all tests:
   3 tests in example.countOdd
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
```

An alternate way to run the doctest is to end each file with:

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

and run example.py directly from the command line as:

```
D:\OneDrive\2019-2020 Fall\cmpsc132>python example.py
```

Or activate verbose mode as

```
D:\OneDrive\2019-2020 Fall\cmpsc132>python example.py -v
```

Appendix: Doctests (continued)

example.py

```
def countOdd(num_list):
    """
        >>> countOdd([1,1,1,2,2,5,5,7])
        6
        >>> countOdd([2,4,6,8])
        0
    """
    odd_count = 0

    for num in range(len(num_list)):
        if num_list[num] % 2 != 0:
            odd_count += 1

    return odd_count

if __name__ == "__main__":
   import doctest
   doctest.testmod()
```

Running doctests in both non-verbose and verbose mode

```
D:\OneDrive\2020-2021 Fall\cmpsc132>python example.py

D:\OneDrive\2020-2021 Fall\cmpsc132>python example.py -v
Trying:
    countOdd([1,1,1,2,2,5,5,7])
Expecting:
    6
ok
Trying:
    countOdd([2,4,6,8])
Expecting:
    0
ok
1 items had no tests:
    example
1 items passed all tests:
   2 tests in example.countOdd
2 tests in 2 items.
2 passed and 0 failed.
Test passed.
```