CSCI 2040: Introduction to Python **2018-2019 Term 1**

Lab Assignment 1

Instructor: Prof. John C.S. Lui Due: 23:59 on Friday, Sept. 21, 2018

Notes

1. You are allowed to form a group of two to do this lab assignment.
2. You are strongly recommended to bring your own laptop to the lab with Anaconda[1] and Pycharm[2] installed. You don't even have to attend the lab session if you know what you are required to do by reading this assignment.
3. **Python 2.x** and **Python 3.x** are both acceptable. But you need to specify the python version in `requirements.txt`. For example, if your scripts are required to run in **Python 2.7**, the following line should appear in `requirements.txt`:

   ```
   python_version == '2.7'
   ```

4. For those of you using the Windows PC in SHB 924A (NOT recommended) with your CSDOMAIN account[3], please login and open "Computer" on the desktop to check if an "S:" drive is there. If not, then you need to click "Map network drive", use "S:" for the drive letter, fill in the path `\\ntsvr6\userapps` and click "Finish". Then open the "S:" drive, open the **Python2** folder, and click the "IDLE" shortcut to start doing the lab exercises. You will also receive a paper document and if anything has changed, please be subject to the paper.
5. The programming assignments are **graded by Python scripts on linux**! Just a reminder, the subscript we provide can be used **only on Linux** for your own test. For those who use Windows, you may not use our test scripts so you can test your solution on your own. If your script does not pass the tests of our grading script, which support both Linux and Windows, you cannot get full grades. You are able to test the input and output format for a exercise by the corresponding testing script. For example, by running `test_p1.py`, you can test `p1.py` for exercise 1. Note that the grading script will be DIFFERENT from the provided testing script. To use our testing scripts, you should install package `pexpect`, and run the testing script in the same folder of the script.

Exercise 1: The Second Smallest Number (20 marks)

`raw_input()` in **Python 2** (equivalent to `input()` in **Python 3**[4]) is a function that read a string from a user's input. The function can be used as `raw_input(['prompt'])`. For

[1] An open data science platform powered by Python. `https://www.continuum.io/downloads`
[2] A powerful Python IDE. `https://www.jetbrains.com/pycharm/download/`
[3] A non-CSE student should ask the TA for a CSDOMAIN account.
[4] Stackoverflow: What's the difference between raw_input() and input() in python3.x?

example, after executing `x = raw_input('type a number:')` and reading the user typed "one", the variable `x` will store the string `"one"`.

Write a script and save it as `p1.py` that will read four numeric values from the user's input. If the input is not a numeric value, then the user should be asked again to input a numeric value and this round does not count. After that, print the second smallest of the four inputted values, print the line `Program ends.` and exit. A sample run should be as follows: (*Hint:* use `float()` to convert a string to the corresponding numeric value)

```
Please input the first number: 4.1
Please input the second number: joke
Your input is not a number!
Please input the second number: -5
Please input the third number: 23.16
Please input the fourth number: -3.02
The second smallest value is -3.02
Program ends.
```

If you use IDLE, you can test your script `p1.py` in the Python shell with

```
>>> execfile('path/to/p1.py')
```

Exercise 2: Pyramid of A String (20 marks)

Using `while` loop or `for` loop to write a script `p2.py` that reads an integer value `n` and a string `str` from the user's input and then produces an `n`-line block, repeatedly. For the `n`-line output, the first line contains `1 str`, the second line contains `2 str`s, and so on until the `n`-th line, which contains only n `str`s. The script will keep asking the user to input an integer value `n` and a string `s` and then print the corresponding `n`-line block. The script will print the line `Program ends.` and exit if the user inputs an integer -1. For this exercise, you don't need to check whether the user's input is an integer or not. A sample run should be as follows:

```
Enter an integer: 4
Enter a string:1
1
11
111
1111

Enter an integer: 2
Enter a string:^_^
^_^
 _
^_^^_^
 _ _
```

```
Enter an integer: -1
Program ends.
```

Exercise 3: GPA Calculator (20 marks)

In order to improve GPA (Grade Point Average), the first thing at hand is to write a script that calculate GPA quickly. With the script, a user could input the grade followed by the credit of a course. Immediately after reading the grade for a course, the GPA calculator print out the current GPA. The user could keep adding the grades for more and more courses until the user inputs -1. When the user inputs -1, the script will print the line `Program ends.` and exit. The grade and credit are separated by space. If the grade or credit inputted is negative, your script should print `Wrong input!` and continue to wait for new inputs. The GPA MUST be in the format of 2 decimal points. (*Hint:* Use `"0:.2f".format(a)` if you want to output 2 decimal points of `a`) A sample run should be as follows:

```
3.7 3
Current GPA: 3.70
3.3 2
Current GPA: 3.54
-2.5 2
Wrong input!
-1
Program ends.
```

Exercise 4: Guess My Number (20 marks)

Hint: To hide the input in Python, you could simply replace `input()` with `getpass.getpass()`.

```
import getpass
getpass.getpass()
```

After knowing that the GPA is improved, the students want to play some games to relax. They decide to play `guess my number`. Specifically, one of the player selects a integer in the range $[-100, 100]$ and report it to the computer when seeing the prompt `Player 1, write down your number secretly:`. After that, the script prompts another player for a guess with the message `Player 2, input your guess:`. At each step, the script responds one of the following 4 sentences:

```
Your guess is too low!
Your guess is too high!
You are right after trying for n times.  Program ends.
You have tried 6 times and it is still wrong!  The answer is ans and program ends.
```

Here `n` is the number of times that the user has tried to guess the random integer correctly

which is less than 6. And `ans` is the answer `Player 1` inputted. A sample run should be like as follows,

```
Player 1, write down your number secretly:
Player 2, input your guess: 0
Your guess is too low!
Player 2, input your guess: 100
Your guess is too high!
Player 2, input your guess: 12
Your guess is too high!
Player 2, input your guess: 9
You are right after trying for 4 times. Program ends.
```

Exercise 5: Palindrome Wonderland (20 marks)

In a secret place near the back hill of CUHK, there is a palindrome wonderland. People there uses palindrome as their language. A palindrome is a string that reads the same backward and forward. For example, the string `eye` and the string `able was I ere I saw elba` are both palindromes. The palindrome wonderland has an entrance guard, which only allows the people speaking palindrome to pass. You would implement such an entrance guard in `p5.py`.

The entrance guard start with the prompt `Please input a string:` that first reads a string from the user's input and check whether the input string is a palindrome or not and the scripts would not exit until a palindrome is inputted. The script will print `Welcome to the wonderland!` and the script exits if the input string is a palindrome. Otherwise, it will print `No, you must input a palindrome:` and keep asking the user to input another string if the input string is not a palindrome. (For this exercise, uppercase and lowercase letters are treated as same characters.) *Hint*: you can use `raw_input()` in Python 2 or `input()` in Python 3. A sample run should be like as follows,

```
Please input a palindrome: 210
No, you must input a palindrome: 012
No, you must input a palindrome: NBaAbn
Welcome to the wonderland!
```

Submission rules

1. Please name the script files with the **exact** names specified in this assignment and test all your scripts. Any script that has any syntax error will not be marked.
2. For each group, please pack all your script files as a single archive named as

`<student-id1>_<student-id2>_lab1.zip`

For example, `1155012345_1155054321_lab1.zip`, i.e., just replace `<student-id1>` and `<student-id2>` with your own student IDs. If you are doing the assignment alone, just leave `<student-id2>` empty, e.g, `1155012345_lab1.zip`.

3. Send the zip file to `cuhkcsci2040@gmail.com`,
   - Each group only needs to send one Email
   - Subject of your Email should be `<student-id1>_<student-id2>_lab1` if you are in a two-person group or `<student-id1>_lab1` if not.
   - No later than 23:59 on Friday, Sept. 21, 2018
4. Students in the same group would get the same marks. Marks will be deducted if you do not follow the submission rules. Anyone/Any group caught plagiarizing would get 0 score!