CSCI 2040: Introduction to Python **2020-2021 Term 1**

Lab Assignment 2

Instructor: John C.S. Lui and S.H. Or Due: 23:59 on Wednesday. Oct. 21, 2020

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. Only **Python 3.x** is acceptable. You need to specify your python version as the first line in your script. For example, if your scripts are required to run in **Python 3.6**, the following line should appear **in the first line of your scripts**:

```
#python_version == '3.6'
```

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 `\\ntsvr1\userapps` and click "Finish". Then open the "S:" drive, open the **Python3** folder, and click the "IDLE (Python 3.7 64-bit)" 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. Your code should only contain specified functions. Please delete all the debug statements (e.g. print) before submission.

Exercise 1 (20 marks)

Please use *list comprehension* to write function `check_sublist(list1, d1, d2)` in the script `p1.py` which takes a list of numbers `list1`, and two integers `d1` and `d2` as arguments and return `lista`: a list of numbers in `list1` that are larger than `d1*d2`, `listb`: a list of numbers in `list1` that are smaller than `d1*d2`, `listc`: a list of numbers in `list1` that are smaller than `d1` *or* `d2`. You can assume that `list1` is a non-empty list and `d1, d2` are two positive integers. Please make sure that your test code is not allowed in the file, and the prototype of the function `check_sublist` is given as follows:

```
def check_sublist(list1, d1, d2):
  # your statement follows
  # ...
  return lista, listb, listc
```

[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.

Testing: Suppose you saved your script `p1.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. In IDLE, you should test your script `p1.py` in the Python shell with

```
>>> import sys
>>> sys.path.append(r"C:\Users\USERNAME\Documents\lab2")
>>> import p1
>>> print(p1.check_sublist([21, 25, 4, 6, 28], 3, 7))
([25, 28], [4, 6], [4, 6])
```

Note: if you editted your script file in the testing procedure, you need to **reload** the imported module before you call any functions. E.g.,

```
# For Python3:
>>> from importlib import reload
>>> reload(p1)
```

Exercise 2 (20 marks)

The numeric system represented by Roman numerals is based on the following seven symbols (with corresponding Arabic values):

Symbol	I	V	X	L	C	D	M
Value	1	5	10	50	100	500	1000

The correspondence between the first nine (Arabic) decimal numbers and the Roman numerals and other basic combiniations are shown as below:

Symbol	I	II	III	IV	V	VI	VII	VIII	IX
Value	1	2	3	4	5	6	7	8	9
Symbol	X	XX	XXX	XL	L	LX	LXX	LXXX	XC
Value	10	20	30	40	50	60	70	80	90
Symbol	C	CC	CCC	CD	D	DC	DCC	DCCC	CM
Value	100	200	300	400	500	600	700	800	900
Symbol	M	MM	MMM	MMMM	M*5	M*6	M*7	M*8	M*9
Value	1000	2000	3000	4000	5000	6000	7000	8000	9000

where `M*5=MMMMM` and so on. For example:

```
LXXIV=L+XX+IV=50+20+4=74
CMXCIX=CM+XC+IX=900+90+9=999
MMMMMMMDCCLXVI=MMMMMMM+DCC+LX+VI=7000+700+60+6=7766
```

Write a function `roman_to_decimal` in the script `p2.py` that takes *a Roman numerals string* as an argument and *return a decimal integer* whose value is equivalent to its corresponding Roman numerals. Your function only needs to process the string in the range [I, MMMMMMMMMCMXCIX], i.e. $[1, 9999]$. For this exercise, you don't need to check whether `str` is a correct Roman numberal string. For the corresponding integer less than 1 or greater than 9999, the function should return "`-1`". The prototype of the function `roman_to_decimal` is given as follows:

```
def roman_to_decimal(str):
    # your statement follows
```

```
    # ...
    return n
```

Testing: Suppose you saved your script `p2.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. In IDLE, you should test your script `p2.py` in the Python shell with

```
>>> import sys
>>> sys.path.append(r"C:\Users\USERNAME\Documents\lab2")
>>> import p2
>>> print(p2.roman_to_decimal('CMLV'))
955
>>> print(p2.roman_to_decimal('XCVI'))
96
```

Exercise 3 (20 marks)

Python allows recursive function, i.e., a function that can call itself. As we know, in a Fibonacci sequence, each number is the sum of the two preceding ones[4].

$$x_{n+1} = x_n + x_{n-1}.$$

In this lab, we want to compute the Fibonacci number. When a newly computed number is larger than 1000, the method stops and outputs the number.

For example, starting from $x_1 = 50$ and $x_2 = 60$, we obtain a sequence:

$$50, 60, 110, 170, 280, 450, 730, 1180$$

And finally the algorithm returns 1180.

Using the observations above, write a *recursive function* `fibo` that calls itself in the script `p3.py` to compute the Fibonacci number. Suppose $x1$ and $x2$ are two initial Fibonacci numbers which are integers and larger than 0, the prototype of the function `fibo` is given as follows:

```
def fibo(x1, x2):
    # x1 and x2 are initial numbers
    # your statement follows
    # ...
    return value # value is the smallest Fibonacci number which is larger than 1000
```

Testing: Suppose you saved your script `p3.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. In IDLE, you should test your script `p3.py` in the Python shell with

```
>>> import sys
>>> sys.path.append(r"C:\Users\USERNAME\Documents\lab2")
>>> import p3
>>> val = p3.fibo(50, 60)
>>> print(val)
1180
```

[4] `https://en.wikipedia.org/wiki/Fibonacci_number`.

Exercise 4 (20 marks)

Write a group of required functions for triangle processing in the script `p4.py`. If you want to calculate a square root, please use `math.sqrt()`, since the test script use this function to generate the standard answer.

- The input `triangle` should be a tuple `(a,b,c)`, where the numeric arguments `a`, `b` and `c` are sides long of the triangle.
- Implement the `check_invalid(triangle)` function and return the Boolean value `True` if the input triangle is not valid, otherwise `False`. The input `triangle` is considered valid if and only if it is a tuple with three positive numbers and any two sides of a triangle must be greater than the length of the third side.
- Implement the `is_isosceles_triangle(triangle)` function and return the Boolean value `True` or `False` to indicate whether the input `triangle` is valid and is a isosceles one.
- Implement the `area(triangle)` and `perimeter(triangle)` functions to return the numerical value of the area and perimeter of the input `triangle`. (*Hint: triangle's area can be calculated by Heron's formula:* $T = \sqrt{s(s-a)(s-b)(s-c)}$*, where* s *is half of its perimeter.*)

Testing: Suppose you saved your script `p4.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. In IDLE, you should test your script `p4.py` in the Python shell with

```
>>> import sys
>>> sys.path.append(r"C:\Users\USERNAME\Documents\lab2")
>>> import p4
>>> t1 = (5, 5, 6)
>>> p4.is_isosceles_triangle(t1)
True
>>> p4.area(t1)
12.0
>>> p4.perimeter(t1)
16
>>> t2 = (3, 6, 1)
>>> p4.check_invalid(t2)
True
```

Exercise 5 (20 marks)

Write a group of required functions for text processing in the script `p5.py`.

- The input `test_string` should be a single string.
- Implement the `count_alphabet(test_string)` function and return the number of alphabetic characters (`a-z` and `A-Z`) in the `test_string`.
- Implement the `vowel_capitalization(test_string)` function and return the string with the vowels (`a, e, i`) capitalized.
- Implement the `concat(test_string, new_string)` function and return a string that is the concatenation of `test_string` and `new_string`.

- Implement the `search(test_string, sub)` function and return the highest index in `test_string` where substring `sub` is found. If not found, it returns -1.
- Implement the `letter_count(test_string)` function and return a list of tuples `(char, count)`, where `char` is the character that is in `test_string` and `count` is the number of times `char` appears:
  - You should only care about English characters *a–z* and *A–Z*.
  - The counting is *case-insensitive*. You should regard uppercase letters as lowercase letters.
  - The output list should be sorted by `char` in *alphabetic* order.

Testing: Suppose you saved your script `p5.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. In IDLE, you should test your script `p5.py` in the Python shell with

```
>>> import sys
>>> sys.path.append(r"C:\Users\USERNAME\Documents\lab2")
>>> import p5
>>> test_str = "Alice was born in 1996 and born in London."
>>> p5.count_alphabet(test_str)
29
>>> p5.vowel_capitalization(test_str)
'AlIcE wAs born In 1996 And born In London.'
>>> p5.concat(test_str, " She is 22 now.")
'Alice was born in 1996 and born in London.  She is 22 now.'
>>> p5.search(test_str, "born")
27
>>> p5.search(test_str, "now")
-1
>>> p5.letter_count("The quick brown fox jumps over the lazy dog")
[('a', 1), ('b', 1), ('c', 1), ('d', 1), ('e', 3), ('f', 1), ('g', 1), ('h', 2), ('i',
1), ('j', 1), ('k', 1), ('l', 1), ('m', 1), ('n', 1), ('o', 4), ('p', 1), ('q', 1),
('r', 2), ('s', 1), ('t', 2), ('u', 2), ('v', 1), ('w', 1), ('x', 1), ('y', 1), ('z',
1)]
```

Submission rules

1. Please name the ***`functions`*** and *script files* with the **exact** names specified in this assignment and test all your scripts. Any script that has any *wrong name or 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>_lab2.zip`

   For example, `1155012345_1155054321_lab2.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_lab2.zip`.
3. Upload the zip file to your blackboard (`https://blackboard.cuhk.edu.hk`),
   - Only one member of each group needs to upload *the archive file.*

- *Subject of your file* should be `<student-id1>_<student-id2>_lab2` if you are in a two-person group or `<student-id1>_lab2` if not.
- No later than 23:59 on Wednesday. Oct. 21, 2020

4. Students in the same group would get the same marks. Marks will be deducted if you do not follow the submission rules. Anyone/Anygroup who is caught plagiarizing would get 0 score!