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

# Lab Assignment 2

Instructor: John C.S. Lui and S.H. Or Due: 23:59 on Fri. Oct. 5, 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. Your code should only contain specified functions. Please delete all the debug statements (e.g. print) before submission.

## Exercise 1 (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 are shown as below:

```
1 -> I, 2 -> II, 3 -> III, 4 -> IV,
5 -> V, 6 -> VI, 7 -> VII, 8 -> VIII, 9 -> IX
```

Now you need to find the conversion rules from decimal to Roman numerals by reading the Wikipedia page[4]. For example, values larger than 10 and less than 99 are treated in exactly the same way

[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] `https://en.wikipedia.org/wiki/Roman_numerals`

with values less than 10, except that `X`, `L`, and `C` are used instead of `I`, `V`, and `X`, e.g., 70 is written as `LXX`, and 74 is written as `LXXIV`.

Write a function `roman_number` in the script `p1.py` that takes a decimal integer as an argument and *return the string that holds the Roman numerals* whose value is equivalent to the decimal integer. Your function only needs to process the integers in the range $[1, 9999]$. For this exercise, you don't need to check whether `n` is an integer. For integers less than 1 or greater than 9999, the function should return "`Number is out of range`". The prototype of the function `roman_number` is given as follows:

```
def roman_number(n):
    # your statement follows
    # ...
    return roman_string
```

**Testing**: Suppose you saved your script `p1.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. 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.roman_number(24))
'XXIV'
>>> print(p1.roman_number(10000))
'Number is out of range'
```

**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 Python2:
>>> reload(p1)
# For Python3:
>>> from importlib import reload
>>> reload(p1)
```

## Exercise 2 (20 marks)

Python allows recursive function, i.e., a function that can call itself. As we known, if $x$ is a number and $n$ is a positive integer, the quantity $x^n$ can be computed by multiplying $x$ for $n$ times. A much faster algorithm would use the following observations. If $n$ is 0, then $x^n$ is 1. If $n$ is even, then $x^n$ is equal to $(x \times x)^{n/2}$. If $n$ is odd, $x^n$ is equal to $x * x^{n-1}$.

Using the observations above, write a *recursive function* `recursive_pow` that calls itself in the script `p2.py` to compute $x^n$. The prototype of the function `recursive_pow` is given as follows: (Do not use the built-in functions `pow` or `math.pow`.) For this exercise, you can assume that `n` is a positive integer.

```
def recursive_pow(x, n):
    # your statement follows
    # ...
    return value # value is equal to x to the power n.
```

**Testing**: Suppose you saved your script `p2.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. 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.recursive_pow(3, 5))
243
```

## Exercise 3 (10 marks)

English letter frequency obeys an interesting distribution. Now we want to count the frequency of each letter in a string. Write a function `letter_count(string)` in the script `p3.py`. The function takes a string as an argument and returns a list of tuples `(char, count)`, where `char` is the character that is in `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.

You can use a dictionary to count the number of occurrences of each letter, then use `sorted()` function to sort the dictionary by its keys. The prototype of the function `letter_count` is given as follows:

```
def letter_count(string):
    # your statement follows
    # ...
    return result
```

**Testing**: Suppose you saved your script `p3.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. 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
>>> p3.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)]
```

## Exercise 4 (10 marks)

Please use *list comprehension* to write function `divisible_sublist(list1, d1, d2)` in the script `p4.py` which takes a list of numbers `list1`, and two integers `d1` and `d2` as arguments and return a list of numbers in `list1` that are divisible by `d1` *or* `d2`. You can assume that `list1` is a non-empty list and `d1, d2` are two positive integers. The prototype of the function `divisible_sublist` is given as follows:

```
def divisible_sublist(list1, d1, d2):
    # your statement follows
    # ...
    return list2  # a list of numbers that are divisible by d1 or d2.
```

**Testing**: Suppose you saved your script `p4.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. 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
>>> print(p4.divisible_sublist([21, 25, 9, 16, 28], 3, 4))
[21, 9, 16, 28]
```

## Exercise 5 (20 marks)

Write a group of required functions for rectangle processing in the script `p5.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 `rectangle` should be a tuple `(h, w)`, where the numeric arguments `h` and `w` are the width and height of the rectangle. The input `rectangle` is considered valid if and only if it is a tuple with two positive number.
- Implement the `check_valid(rectangle)` function and return the Boolean value `True` or `False` to indicate whether the input is valid or not.
- Implement the `is_square(rectangle)` function and return the Boolean value `True` or `False` to indicate whether the input `rectangle` is a square.
- Implement the `diagonal_len(rectangle)` function to return the numerical value of the length of the diagonal of the input `rectangle`.
- Implement the `height(rectangle)` and `width(rectangle)` functions to return the numerical value of the hight and width of the input `rectangle`.
- Implement the `area(rectangle)` and `perimeter(rectangle)` functions to return the numerical value of the area and perimeter of the input `rectangle`.

**Testing**: Suppose you saved your script `p5.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. 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
>>> r1 = (4, 3)
>>> p5.is_square(r1)
False
>>> p5.height(r1)
4
>>> p5.width(r1)
3
```

```
>>> p5.area(r1)
12
>>> p5.perimeter(r1)
14
>>> r2 = (-1, 3)
>>> p5.check_valid(r2)
False
```

## Exercise 6 (20 marks)

Write a group of required functions for text processing in the script `p6.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 string.
- Implement the `vowel_capitalization(test_string)` function and return the string with the vowels (a, e, i, o, u) 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 lowest index in `test_string` where substring `sub` is found. If not found, it returns -1.

**Testing**: Suppose you saved your script `p6.py` in `C:\Users\`*`USERNAME`*`\Documents\lab2`. You should test your script `p6.py` in the Python shell with

```
>>> import sys
>>> sys.path.append(r"C:\Users\USERNAME\Documents\lab2")
>>> import p6
>>> test_str = "Alice was born in 1996.  "
>>> p6.count_alphabet(test_str)
14
>>> p6.vowel_capitalization(test_str)
'AlIcE wAs bOrn In 1996.  '
>>> p6.concat(test_str, "She is 22 now.  ")
'Alice was born in 1996.  She is 22 now.  '
>>> p6.search(test_str, "born")
10
>>> p6.search(test_str, "now")
-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. 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>_lab2` if you are in a two-person group or `<student-id1>_lab2` if not.
   - No later than 23:59 on Fri. Oct. 5, 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/Anygroup who is caught plagiarizing would get 0 score!

5. If you have any problem with respect to lab 2, please email to twliu@cse.cuhk.edu.hk.