CSCI 2040: Introduction to Python **2024-2025 Term 1**

Lab Assignment 2

Instructor: Dr. Yim Pan, CHUI Due: 23:59pm on Wednesday, Oct. 23

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 Community[2] installed. You don't 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.
4. Passing the test scripts we have provided does not guarantee full marks for your question as our grade scripts will test for more cases.
5. You may assume that all the corner cases we have not mentioned in this document will not appear in the hidden test cases, so you do not have to worry too much about wrong inputs unless you are required to do so.
6. 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 `divisible_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 divisible by `d1` or `d2`, `listb`: a list of numbers in `list1` that are divisible by `d1` and `d2`, `listc`: a list of numbers in `list1` that aren't divisible by `d1` and `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 lista, listb, listc
```

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.divisible_sublist([21, 25, 9, 16, 28], 3, 7))
([21, 9, 28], [21], [25, 16])
```

[1] An open data science platform powered by Python. `https://www.anaconda.com/download`

[2] A powerful Python IDE. `https://www.jetbrains.com/pycharm/download/other.html`

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

For example:
`LXXIV=L+XX+IV=50+20+4=74`
`XCVII=XC+VII=90+7=97`

Write a function `roman_to_decimal` in the script `p2.py` that takes one Roman numerals strings as an argument and return the corresponding decimal integer. Besides, you need to write a function `decimal_to_roman` in the script that takes one positive integer as an argument and return the corresponding Roman numerals string. Your function only needs to process the string in the range [I, XCIX], i.e. $[1, 99]$. For this exercise, you don't need to check whether `str` is a correct Roman numeral string. The prototype of the function `roman_to_decimal` and `decimal_to_roman` are given as follows:

```
def roman_to_decimal(str):
    # your statement follows
    # ...
    return n

def decimal_to_roman(n):
    # your statement follows
    # ...
    return str
```

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('XCVII'))
97
>>> print(p2.decimal_to_roman(88))
LXXXVIII
```

Exercise 3 (20 marks)

Python allows recursive function, i.e., a function that can call itself. As we know, calcuating an integer's square is easy, while calculating its square root is difficult, i.e. to solve $x^2 = a, \forall a \in \mathbf{N}^+$. We can apply the well-known Newton's method to compute square root[3]. This method suggests how to guess an integers' square root better iteratively:

$$x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n}).$$

For simiplity, the method stops when $|x_{n+1} - x_n| \leq 0.001$ and truncates the latest guess value to 2 decimal places (without rounding).

For example, we want to guess $\sqrt{2}$, i.e. $a = 2$. Starting from guess $x_1 = 1$, we obtain:

$$x_1 = 1$$
$$x_2 = 1.5$$
$$x_3 = 1.41666666666666666666666666666666666666666666666666666666666666675$$
$$x_4 = 1.41421568627450980392156862745098039215686274509803921568627450980392156862745$$
$$x_5 = 1.41421356237468991062629557889013491011655962211574404458490571$$

Notice $|x_5 - x_4| \leq 0.001$, we stop and guess $\sqrt{2} \approx 1.41$.

Using the observations above, write a recursive function `recursive_sqrt` that calls itself in the script `p3.py` to compute $\sqrt{x}$. The prototype of the function `recursive_sqrt` is given as follows: (Do not use the built-in functions `a**(.5)` or `math.sqrt(a)`.) For this exercise, we only consider positive integer's sqare root. As for start point x_0, you can choos any positive integer as you guess.

```
def recursive_sqrt(a, x):
    # a is an integer, x is the former guess
    # your statement follows
    # ...
    return value # value is next square root guess
```

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")
```

[3] `https://en.wikipedia.org/wiki/Newton%27s_method#Square_root_of_a_number`, for further interest: `https://math.mit.edu/~stevenj/18.335/newton-sqrt.pdf`

```
>>> import p3
>>> guess = p3.recursive_sqrt(2, 1)
>>> print("%.2f" % guess)
1.41
```

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 the sum of any two sides of a triangle must be greater than the length of the third side.
- Implement the `is_obtuse_triangle(triangle)` functionand return the Boolean value `True` if the input triangle is an obtuse one, otherwise `False`. (Hint: for an obtuse triangle, if the largest side is c, then $c^2 > a^2 + b^2$.)
- 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.)
- Implement the `outer_radius(triangle)` to return the radius of the outer circle of the input `triangle`. (Hint: the radius of the outer circle of a triangle can be calculated by formula: $R = abc/4T$, where T is the area of the triangle.)

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 = (3, 4, 5)
>>> p4.area(t1)
6
>>> p4.perimeter(t1)
12
>>> p4.is_obtuse_triangle(t1)
False
>>> p4.outer_radius(t1)
2.5
>>> 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_digit(test_string)` function and return the number of digital characters `0-9` in the `test_string`.
- Implement the `check_isogram(test_string)` function and return the bool variable to indicate whether `test_string` has duplicate letters. If `test_string` is an isogram, then the function returns `True`. If `test_string` is not an isogram, then the function returns `False`. (To avoid case sensitivity issues in the check_isogram function, first convert the input string to lowercase using Python's lower() method before checking for duplicate letters.)
- Implement the `join(original_string, inserted_list)` function and return a new string which made by joining characters in `Inserted_list` with `original_string` inserted between every element.
- 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.

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 2000 and born in hong kong."
>>> p5.count_digit(test_str)
4
>>> p5.check_isogram(test_str)
False
>>> p5.search(test_str, "born")
27
>>> p5.search(test_str, "now")
-1
>>> p5.join('-', ['a','b','c'])
'a-b-c'
```

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 23:59pm on Wednesday, Oct. 23
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!