CS1010E-PE1-2022 - Practice

Programming Methodology (National University of Singapore)

CS1010E Practical Exam I

Instructions:

- In Examplify, in order to view this exam paper in full screen and to copy&paste some of its content to another file, you need to click on the "setting" clog-like button at the right side, then click on the "Print" button to display the document using PDF reader. Then, you can copy-&-paste the code template (in the last page) frm this printed document to your respective py files.
- You ***cannot import*** any additional packages or functions, or you will get zero mark.
- This PE consists of **THREE** parts and each part should be submitted separately in Examplify and Cousremology. It's better for you to organize your files into `part1.py`, `part2.py` and `part3.py` and each file is independent, e.g. you cannot expect that the answer in `part2.py` calls a function in `part1.py` that is not in `part2.py`. If you want to reuse some function(s) from another file/part, you need to copy them (as well as their supporting functions) into that file/part. For example, you just need to copy the function you want to call in `part1.py` into `part2.py`.
- You should save an **exact** copy of your Examplify submission in your computer for submitting in Coursemology again ***after*** the PE. Any differences between Exemplify and Coursemology submissions will be severely penalized, except for indentation differences. If we found out that there are differences between the two copies, the copy on Examplify will be marked instead of the Coursemology copy.
- You are allowed to write extra functions to structure your code better. However, remember to submit them together mentioned in the point above.
- No mark will be given *if your code cannot run*, namely any syntax errors or crashes.
  - We will just grade what you have submitted and we will **not** fix your code.
  - Comment out any part that you do not want.
- Workable code that can produce correct answers in the given test cases will only give you *partial* marks. Only good and efficient code that can pass ALL test cases will give you full marks. Marks will be deducted if your code is unnecessarily long, hard-coded, or in poor programming style, including irrelevant code or test code.
- You must use the same function names as those in the template given.
- Your code should be efficient and able to complete *each* example function call in this paper within 1 second.
- In all parts, you should **return** values instead of **printing** your output. In another word, you should not need any "`print()`" in this entire PE for submission.
- You should either delete all test cases before submission or comment them out by adding # before the test cases. If you submit more code than required, it will result in **penalty** because your code is "unnecessarily long".
- Reminder: Any type of plagiarism like copying code with modifications will be caught, that include adding comments, changing variable names, changing the order of the lines/functions, adding useless statements, or restructuring your code, etc.

Part 1 Sum of Power Series (15 + 15 = 30 marks)

In this part, we want to compute the sum of k integers with power d:

$$\sum_{i=1}^{k} i^{d}$$

For example, if k = 4 and d = 3, the sum is:

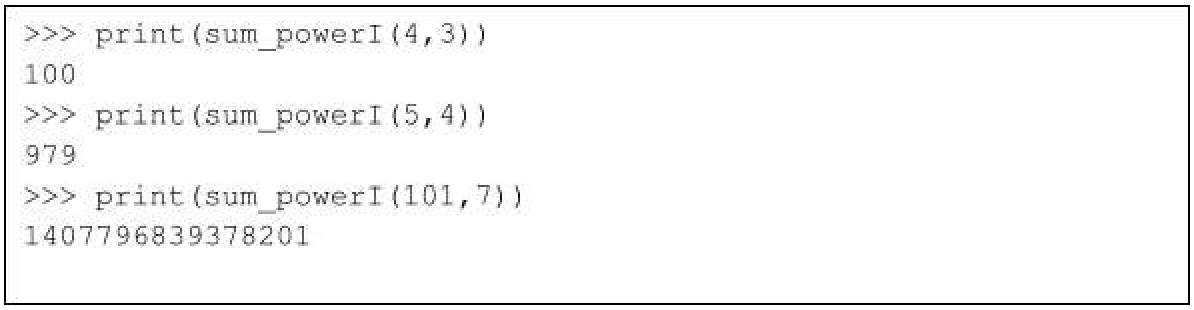
$$\sum_{i=1}^{4} i^{3} = 1^{3} + 2^{3} + 3^{3} + 4^{3} = 1 + 8 + 27 + 64 = 100$$

Another example, if k = 5, d = 4, the sum is:

$$\sum_{i=1}^{5} i^{4} = 1^{4} + 2^{4} + 3^{4} + 4^{4} + 5^{4} = 1 + 16 + 81 + 256 + 625 = 979$$

Task 1 Iterative Sum of Powers (15 marks)

Write an ***iterative*** version of the function `sum_powerI(k,d)` return an integer of the sum of the integers i to power d for i from 1 to k. Your function cannot be recursive in this task.



```
>>> print(sum_powerI(4,3))
100
>>> print(sum_powerI(5,4))
979
>>> print(sum_powerI(101,7))
1407796839378201
```

You cannot use any recursive call in this task.

Task 2 Recursive Sum of Powers (15 marks)

Write a ***recursive*** version of the function `sum_powerR(k,d)` with the same functionality in Part 1 Task 1. However, you cannot use any loops or list comprehension in this task, e.g no “while” nor “for”. You have to use recursion technique in this task.

Part 2 Counting Sheep (40 marks)

A sleeping research association is doing a research on how many sheep will someone count until he falls into sleep. We will record this counting in a string like this

‘1234567891011121314151617’

We assume a person can count clearly when he is awake. However, someone starts to fall into sleep when he is losing the counting, e.g. if someone counts

‘**12345678910111213**1516172031’

We know that he starts falling into sleep at count `13`.

Task 1 Counting Sheep (20 marks)

Write a function `count_sheep(s)` to return the integer of the last number that a person counts correctly before he fell asleep with the input string `s`. If someone cannot count correctly from the beginning, return zero. You can assume the string `s` will not contain any character other than the ten digits from 0 to 9. Some example output:

```
>>> print(count_sheep('123456789101112131516172031'))
13
>>> print(count_sheep('12356789101112l3'))
3
>>> print(count_sheep('1234567'))
7
>>> print(count_sheep('912356789101112l3'))
0
```

Task 2 Alice Counting Sheep (20 marks)

Alice in the Wonderland likes to count sheep also. However, she will be counting in the Fibonacci sequence*:

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Again, she falls asleep when she “breaks” the sequence.

Write a function `alice_count_sheep(s)` to return the integer of the last Fibonacci number that Alice counts correctly before she fell asleep with the input string `s`. Here are some sample output:

```
>>> print(alice_count_sheep('112358132021'))
13
>>> print(alice_count_sheep('112358132134'))
34
>>> print(alice_count_sheep('11235813213455891442333776109871l597'))
987
```

**The Fibonacci Sequence is the series of numbers that each number is found by adding up the two numbers before it, except the first two numbers which are 1 and 1. It’s believed that Fibonacci numbers are hidden in the book Alice's Adventures in Wonderland by Lewis Carroll.*

Part 3 Escape from Magic Castle (30 marks)

An evil black wizard trapped 99 good white wizards in his evil castle. There are 100 dungeon cells in this castle and the white wizards are locked up in the cells numbered by 1 to 99 and each of these cells is totally sealed and no one can get out of the cell, except that one of the cells, the cell numbered by 0, leads to the exit of the castle.

The castle is under the power of the black wizard that the white wizards cannot use their magic freely, except the last ultimate spell of the white wizard, namely *teleport*. However, under the enchantment of the evil black wizard, the white wizard can only teleport to one of the 100 cells. Moreover, if a white wizard is currently at cell number `x`, he can only teleport to the cell with the number:

$$\left(3x^3 + 7\right)\%100$$

Hopefully, the white wizards can teleport to the cell 0 and exit the castle. You are their only hope!

In this part, you can assume the input x is always an integer from 1 to 99, inclusively. For Tasks 2, please only submit one copy of the function `number_of_teleport(x)` only.

Task 1 Teleport (5 marks)

Write the function `teleport(x)` to return the integer of the cell that a white wizard can teleport to from cell x.

```
>>> teleport(45)
82
>>> teleport(82)
11
>>> teleport(11)
0
```

Task 2 Number of Teleport (25 marks)

If a white wizard starts from cell x, how many times does he need to teleport to cell 0? Write a function `number_of_teleport(x)` to return the integer of number of teleports he needs to exit the castle if he starts from cell x. For example, if a white wizard starts at cell 45:

$$45 \longrightarrow 82 \longrightarrow 11 \longrightarrow 0$$

Namely, he needs to teleport **3** times to exit the castle.

Unluckily, we found out that some of the wizards cannot exit the castle by teleports no matter how many times they cast the spell. (Poor wizards!) Add on the functionality of your function `number_of_teleport(x)` that will return `'Trapped'` if a white wizard cannot exit the castle from cell `x`.

```
>>> print(num_of_teleports(45))
3
>>> print(num_of_teleports(17))
5
>>> print(num_of_teleports(96))
4
>>> print(num_of_teleports(83))
'Trapped'
>>> print(num_of_teleports(39))
'Trapped'
>>> print(num_of_teleports(40))
6
```

Code Template