# CS 251 OUTLAB 3 - Python

## General Instructions

- Make sure you know what you write, you might be asked to explain your code at a later point in time.
- The submission will be graded automatically, so stick to the naming conventions strictly.

## Submission Instructions

```
<rollno1>-<rollno2>-outlab3/
|
├── ring.py
├── q1a.py
├── q1b.py
├── q1c.py
├── q2.py
├── q3.py
├── q4.py
├── q5.py
├── q6.py
├── … one additional .py file if you need
└── references.txt
```

**Your submission directory should be called <rollno1>-<rollno2>-outlab3. These should be sorted lexicographically.**
**For example:**
**17D070059-180070035-outlab3**

**Compress it into a tarball, strictly with the following command:**

```
tar -zcvf 17D070059-180070035-outlab3.tar.gz
17D070059-180070035-outlab3
```

**Of course, replace our roll numbers with yours! We will provide a script for you to verify that you conform to the submission format.**

**You have to submit on Moodle. The link is up, please find the script along with that. Submit once per pair, from the account of the student with the lexicographically smaller roll number, i.e. rollno1. In our example, 17D070059 will submit.**

## Problem 1 [20 marks]
### Task A: Ring around the Roses

In your Discrete Structures course, you must have studied “modulo division” and the related arithmetic. The structure this arithmetic is defined over is said to be a “ring”, i.e. it is equipped with definitions of a sum and a product, which is distributive over the sum.

In the ring of integers modulo 7 (here 7 is called the *characteristic*), 2+3=5; but 5+4=2. Also, 2*3=6; 5*3=1. Of course, in these rings, the additive identity is 0, the multiplicative identity is 1. When the characteristic is *prime,* the multiplicative inverse of all numbers except 0 is defined. This means we can define subtraction and division as well and what we have here is a *field*.

You need to implement the **RingInt** class: an instance of this class must have a <u>characteristic</u> (i.e. it is in the ring of integers modulo what) and a <u>value</u>. If the characteristic is 7, possible values are 0, 1, 2, 3, 4, 5, 6. We will denote them as 0[7], 1[7], …, 6[7]. Define the __str__ method for the class!

Overload the +, -, *, /, **, == operators for the RingInt class. These are only defined when the characteristics of the operands are equal! Division by 0 (or a divisor that has a common factor with the characteristic) is *undefined*. Wherever defined, operations **must** return a RingInt. When the operations are undefined, your code must raise a *ValueError.* (Read more about Exception Handling in Python).

5[7]/6[7] = 2[7]

2[7]**1 = 2[7]
2[7]**2 = 4[7]
2[7]**3 = 1[7]

(notice that the exponent is a regular integer)

Define this class in file **ring.py**

Now we come to our original task. You already have studied about series expansion. However, here we're going to deal with the ring of integers modulo n ($Z_n$). Your first task would be to compute the sum of the following series upto k terms in a series given the value of k and x. Make use of the class you defined for this part

1. $S = \sum_{i=0}^{k-1} \frac{x^i}{i!} \quad n > k$

2. $S = {}^{x}C_0 \times ({}^{x+1}C_0 + {}^{x+1}C_1) \times \dots ({}^{x+k-1}C_0 + \dots {}^{x+k-1}C_{k-1}) \;, \; n \geq x + k$

3. $S = 1^x + 2^x + 3^x \dots$ upto $k$ terms $n > x, \; k$

where $\quad {}^{n}C_r = \frac{n!}{n!(n-r)!}$ each multiplication and division operation being defined in $Z_N$

The k, x and n values and the series number (1, 2, or 3) are provided in **input.txt *in that order*** separated by a space. Each line of **output.txt** should contain the corresponding sum of the series. If the sum is undefined, write "UNDEFINED" to the file. Your program should take **2** command line arguments containing the names input and output files respectively. It is preferable to have a separate function for each series. Invoke your program like:

```
python3 q1a.py -inp <input-file> -out <output-file>
```

**Note:** The ordering of the arguments may vary. E.g:

```
python3 q1a.py -out <output-file> -inp <input-file>
```

This should provide the same output.

**Do not use any library support for this question**

**Hint:** Use `argparse` for handling command line arguments

Sample **input.txt**

```
------------------------
4 2 3 3
5 1 8 1
```

Sample **output.txt**

```
--------------------------
0[3]
UNDEFINED
```

Sample **input.txt**

```
------------------------
5 3 11 1
3 2 25 2
3 2 15 2
```

Sample **output.txt**

```
-------------------------
2
19
UNDEFINED
```

## Task B: Erroneous messages

Sara and Dana are sisters living in different cities. They are stuck in a pandemic, but they desperately need to communicate with each other. The only mode of communication between them is through email. They want to share some secret information through *codes*. A code is of the following form:

$$C = \$(a_1, a_2, \dots a_l)\#(b_1, b_2, \dots b_m)\$, s.t\ \forall i,\ a_i \in Z_N,\ \forall j,\ b_j \in Z_N\ ,\ l > 0,\ m > 0$$

Each code has an unique property: $\sum_{i=1}^{l} i \times a_i = 0 \bmod n$ and $\sum_{i=1}^{m} i \times b_i = 0 \bmod n$. For example, if N = 3, a valid code can be $\$(1, 1\ 0)\#(2, 1, 1, 2)\$$ .

The file that Dana receives from is given to you in **message.txt**. The first line of this file contains **N** (N is prime for this subtask)**.** The communication channel they are using is pretty secure, however, sometimes, one number $a_i$, or $b_i$ can change due to noise in the channel. Help Dana figure out whether the message that she receives is corrupted or not. Output to *stdout* “OK” or “CORRUPTED” depending on the message.

Usage:

```
python3 q1b.py -m <message_file>
```

Example **message.txt**

```
----------------------------------
3
Hi dear Dana. It’s great 2 finally get in touch with you. I have read the article (0, 1)
that you sent me. It’s awesome. Here is some $(1,1,0)#(2,1,1,2)$ cookies for you.
Have a gr8 day, Bye $Sara$
```

Example output **(to *stdout*)**

```
------------------------------------
OK
```

## Task C: Series revisited

Take the 1st series from Task A: $S = \sum_{i=0}^{k-1} \frac{x^i}{i!}$ **.** Instead, now we are interested in just the individual terms of the series, $a_i = \frac{x^i}{i!}$ denoting the $i^{th}$ term. Define a class `Series` having the behaviour of an iterator, i.e.define the `__iter__() and __next__()` methods.

Code stubs are provided for this question in q1c.py. Please **do not** change the stub.

## Problem 2 [15 marks]

Linus has installed his favourite Linux distro on his system. He is trying to get his head around Unix-style paths. Given an *absolute path* for a file (Unix-style), he wants to simplify it to the corresponding *canonical path*. A file or a directory can have multiple *absolute paths*, but only **1** unique *canonical path*. For example, `/usr/bin/../src/` and `/usr//src/./` both refer to the same *canonical path* `/usr/src`

In a UNIX-style file system, a period ‘.’ refers to the current directory. Furthermore, a double period ‘..’ moves the directory up a level.

The canonical path must always begin with a slash ‘/’, and there must be only a single slash ‘/’ between two directory names. The last directory name **must not** end with a trailing ‘/’. Also, the canonical path must be the shortest string representing the absolute path.

**Note:**

- All absolute and canonical paths start with /, i.e. they are w.r.t the root directory.

- All directory and file names are alphanumeric.

The input file contains a single string denoting an absolute path. The output file will have a single line of output. Run your code as:

```
python3 q2.py -inp <input-file> -out <output-file>
```

Sample **input.txt**
---------------------------------
/a/../../b/../c//.//

Sample **output.txt**
---------------------------------
/c

Sample **input.txt**
---------------------------------
/a//b////c/d//././/..

Sample **output.txt**
---------------------------------
/a/b/c

## Problem 3 [15 marks]

Matt owns a candy shop. His shop has $M$ number of candies. Each candy can be of $1,\ 2,\ ...,\ k$ types. He has a list containing the type of each candy he has in his shop.

There are $N$ children who are willing to pay $c_i$ amount of money only if they get the candy of their desired type.

Your task is to maximize how much money Matt earned.

Input Format

The file **candies.txt** contains 2 lines. The first line has $M$ , the number of candies. The second line contains the space separated integers, the $i^{th}$ of which represent the type of the $i^{th}$ candy in the shop.

The file **children.txt** contains 2 lines. The first line contains $N$ , the number of customers. There are $N$ following lines. Each of these next lines contain space separated values of the type desired by the children and the price of the candy.

**Hint:** Have a look at Python `collections`

Sample **candies.txt:**

```
--------------------------------
10
2 3 4 5 6 8 7 6 5 18
```

Sample **children.txt:**

```
--------------------------------
6
6 55
6 45
6 55
4 40
18 60
10 50
```

Sample **output (stdout)**

```
-------------------------------
210
```

Explanation:  Sell Type 6 to 1st and 3rd child, Type 4 to 4th child, Type 18 to 5th child. Type 10 is not in his shop. So, 55 + 55 + 40 + 60 = 210

```
python3 q3.py -ca <candies-file> -ch <children-file>
```

**Note:** The ordering of the arguments may vary.

## Problem 4 [25 marks]

## q4.py file is edited, please have a look.

Binary trees with which we are dealing in this question have nodes that have 2 children or no children.

**An internal node** is defined as a node in the tree which has two children - these children may themselves be a leaf or an internal node. Hence understand that by definition a leaf is not an internal node, and a node is either an internal node or a leaf.

Henceforth, a tree is said to be of size n if it has **n internal nodes. For a binary tree,** this tree will have in total **2n + 1** nodes: n + 1 of them will be leaves. A tree has at least one node as defined here - it's head.

**Function 1: Mirror a tree [5 marks]**
Fill up the function def mirrorTree(node)using the explanation given below.
**Arguments (1):** head node of a tree.
**Return value:** Root (same as head) of newly constructed tree that's the mirror image to the given tree.
Don't print anything. Do not change the name of or arguments to this function- it will be auto-graded.

**Definition:** The mirror image of a tree is defined as given in figure 1 below. T1 is a mirror image of T2 if one child of the head node of T1 is the mirror image of the other child of the head node of T2. For head being a leaf - mirror image of a leaf head tree is itself.

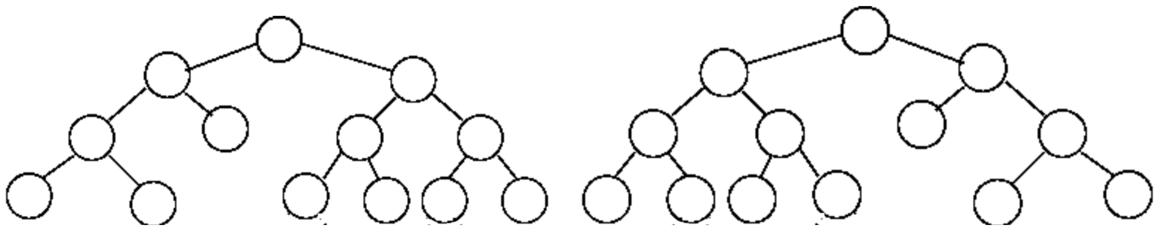

**Figure 1:** Examples of trees that are mirror images of each other, not symmetrical

**Function 2: Find all trees with 'n' internal nodes [10 marks]**
Complete function def allTrees(n)using the explanation given below.
Don't print anything. Do not change the name of or arguments to this function- it will be auto-graded.
**Arguments (1):** A single integer - number of internal nodes
**Return value:** List of heads of all unique trees that have n internal nodes. The order does not matter.
Limits - 0 <= n <= 11
Note that the list is exponential in n,
This function, on using **list comprehension**, can be completed with just 1 for loop, no nesting.
By a for loop, we mean an indented block that goes

```
for foo in bar:
     blah
# end
...
```

Hence an instance of list comprehension is not a for loop

Hence you will be awarded **full 10 marks** for correct output and 1 for loop used, **8 marks** if the output is correct but two for loops have been used (nested or not), **6 marks** if the output is correct but three loops have been used (nested or not). **5 marks** for correct output and more than 3 loops. **Zero marks** if the output is incorrect.

**Function 3: Find all trees with 'n' internal nodes that are symmetrical (i.e) they exactly match their mirror images. [10 marks]**

Complete function def allSymTrees(n)using the explanation given below.
Don't print anything. Do not change the name of or arguments to this function- it will be auto-graded.
**Arguments (1):** A single integer - number of internal nodes
**Return value:** List of nodes that are heads to symmetrical trees. Again, the order does not matter.
This will be rather simple once you have completed allTrees
**Definition of equality:** The equality of two trees is equivalent to the equality of its head node, which has been defined in the Node class. Read it carefully.
**Definition of symmetry:** if node is head of tree T, then T is symmetric iff node == mirror(node).
This is also exponential, 0 <= n <= 23
This function, on using **list comprehension**, can be completed without for loops.
Hence you will be awarded **full 10 marks** for correct output and no for loops and 7 **marks** if the output is correct but one for loop has been used (nested or not). **5 marks** for more than 1 loop. **Zero marks** if the output is incorrect.

**Hints for function 3:**

1. What is the relation between all trees and all symmetric trees?
2. Even number of internal nodes - is such a symmetric tree possible?
3. What would happen if the left child of the head node is the mirror of the right child?

Just to reinforce what you might be thinking, and to make this question slightly easier, the following are true. Use these to understand the approach for the last function.

a. $len(allTrees(n)) \ == \ len(allSymTrees(2n + 1))$
b. $len(allSymTrees(n)) \ == \ 0$

## Problem 5 [10 marks]

**Note : Don't use Numpy library.**

A square n×n matrix (n>=1) of integers can be written in Python as a list with n elements, where each element is, in turn, a list of n integers, representing a row of the matrix.

Write a function rotate(m) that takes a list representation m of a square matrix as input, and returns the matrix obtained by rotating the original matrix clockwise by 90 degrees.

**Input:**

```
1 2 3
4 5 6
7 8 9
```

**Output:**

```
7 4 1
8 5 2
9 6 3
```

## Problem 6 [15 marks]

Given L, a list that contains a list of strings, the function concatenates all the strings occurring inside this list of lists into one single string using space as a separator.

Important: You must use lambda expression and reduce to define functions which are to be called upon. NO LOOPING ALLOWED.

Example: L = [ ["this","is"], [ ["an", "interesting", "python"], ["programming", "exercise."] ] ]

Function Def: collapse(L)

Returns: "this is an interesting python programming exercise."