COMP9021 Principles of Programming
Term 2, 2024

Assignment 1

Worth 13marks and due Week 7 Monday @ 10am

1. General Matters

1.1 Aim

The purpose of this assignment is to:

- develop your problem-solving skills.
- design and implement the solution to a problem in the form of a **small** sized Python program.
- practice the use of **arithmetic computations**, **tests**, **repetitions**, **lists**, **dictionaries**, and **strings**.
- use **procedural** programming.

1.2 Marking

This assignment is worth **13 marks** distributed as follows:

```
Rectangles Boundary   5 marks
Moving Die            4 marks
Fishing Towns         4 marks
------------------------------------------------------------------------
Total                13 marks
```

Your program will be tested against several inputs. For each test, the auto-marking script will let your program run for **30 seconds**. The outputs of your program should be **exactly** as indicated.

1.3 Due Date and Submission

Your programs will be stored in the files **boundary.py**, **moving_die.py**, and **fishing_towns.py**. The assignment can be submitted more than once. The last version just before the due date and time will be marked (unless you submit late in which case the last late version will be marked).

Assignment 1 is due **Week 7 Monday 8 July 2024 @ 10:00am** (Sydney time).

Note that **late** submission with **5% penalty per day** is allowed **up to 5 days** from the due date, that is, any late submission after **Week 7 Saturday 13 July 2024 @ 10:00am** will be discarded.

Make sure not to change the filenames **boundary.py**, **moving_die.py**, and **fishing_towns.py** while submitting by clicking on **[Mark]** button in **Ed**. It is your responsibility to check that your submission did go through properly using **Submissions** link in Ed otherwise your mark will be **zero** for **Assignment 1**.

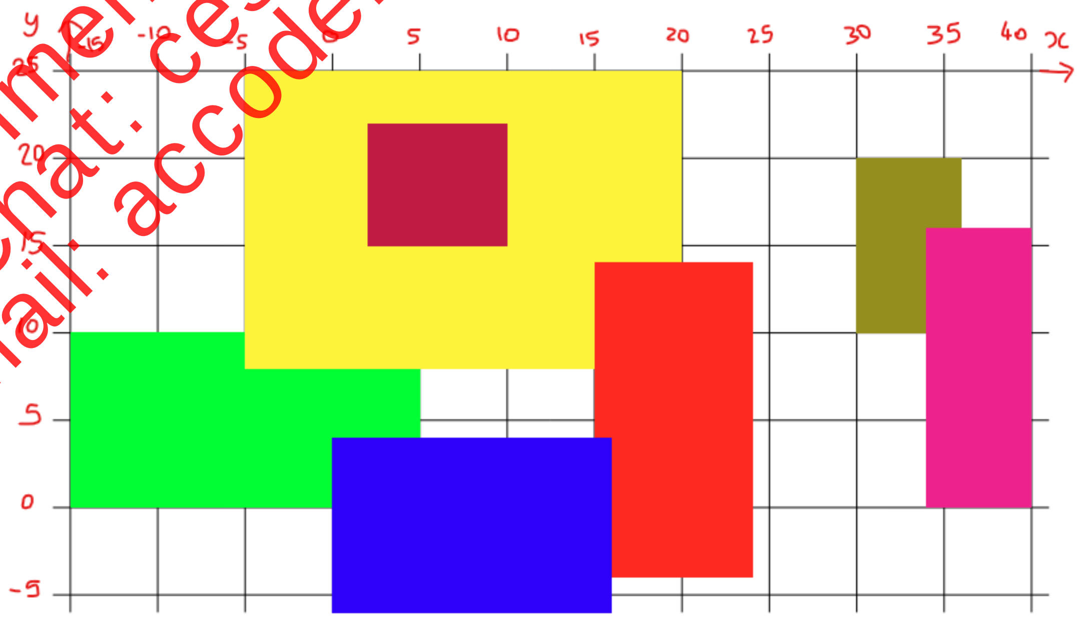
1.4 Reminder on Plagiarism Policy

You are permitted, indeed encouraged, to discuss ways to solve the assignment with other people. Such discussions must be in terms of **algorithms, not code**. But you **must implement the solution on your own**. Submissions are **scanned for similarities** that occur when students copy and modify other people's work or work very closely together on a single implementation. Severe penalties apply.

2. Overlapping Rectangles (5 Marks)

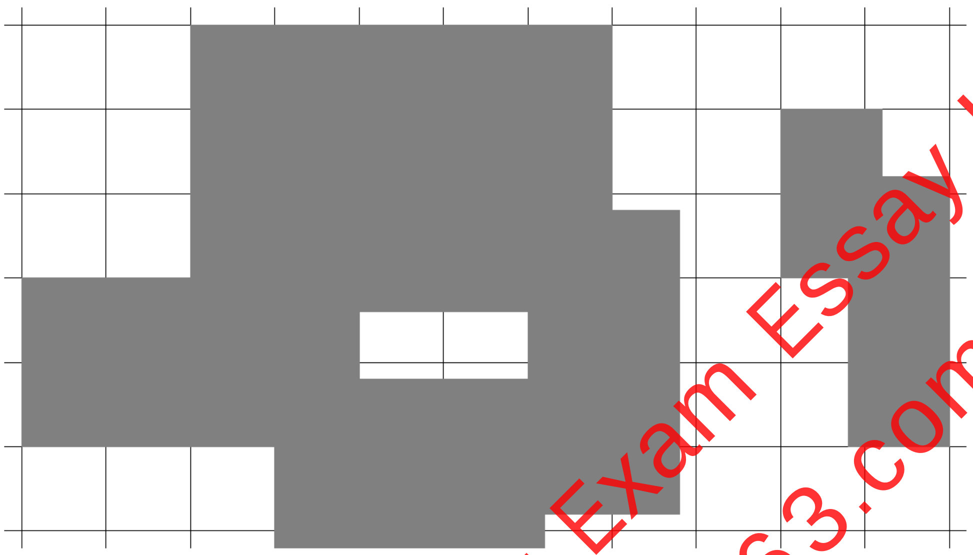
Write a program, stored in a file named **boundary.py**, that performs the following tasks:

- Prompts the user to input the name of a text file assumed to be stored in the working directory. We assume that if the name is valid then the file consists of lines all containing 4 integers separated by whitespace, of the form *x1 y1 x2 y2* where *(x1, y1)* and *(x2, y2)* are meant to represent the coordinates of two opposite corners (**lower left** and **upper right**) of a rectangle. With the provided file **rectangles_1.txt**, the rectangles can be represented as follows:



- We assume that all rectangles are distinct and either properly **overlap** or are **disjoint** (they do not touch each other by some of their sides or some of their corners).

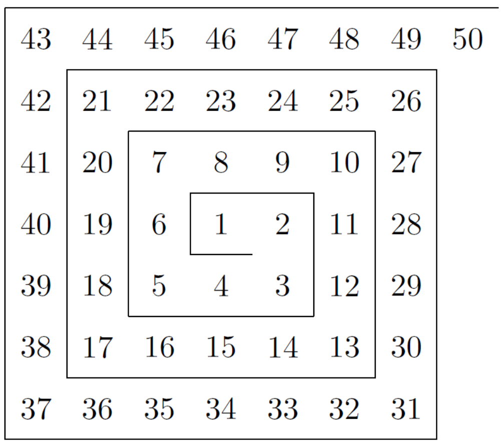
- Computes and outputs the perimeter of the boundary, so with the sample file **rectangles_1.txt**, the sum of the lengths of the (external or internal) sides of the following picture:



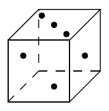
See the possible interaction with the two provided sample files in Section 5 below.

3. Moving Die (4 Marks)

Consider the board below, which can be imagined as being infinite, so only a small part is represented. A die is placed on cell number 1 of the board in the position indicated: 3 at the top, 2 at the front (towards cell number 4 of the board), and 1 on the right (towards cell number 2 of the board). Recall that 4 is opposite to 3, 5 is opposite to 2, and 6 is opposite to 1. The die can be moved from cell 1 to cell 2, then to cell 3, then to cell 4..., each time pivoting by 90 degrees in the appropriate direction (to the right, forwards, to the left, or backwards). For instance, after it has been moved from cell 1 to cell 2, 2 is still at the front but 6 is at the top and 3 is on the right.



43	44	45	46	47	48	49	50
42	21	22	23	24	25	26	
41	20	7	8	9	10	27	
40	19	6	1	2	11	28	
39	18	5	4	3	12	29	
38	17	16	15	14	13	30	
37	36	35	34	33	32	31	



Write a program, stored in a file named **moving_die.py**, that prompts the user for a strictly positive integer **N**, and outputs the numbers at the top, the front and the right after the die has been moved to cell N.

See the possible interaction with the three provided sample files in Section 5 below.

4. Fishing Towns (4 Marks)

Write a program, stored in a file named **fishing_towns.py**, that performs the following tasks:

- The program prompts the user to input a file name. If there is no file with that name in the working directory, then the program outputs an (arbitrary) error message and exits.
- The contents of the file consist of some number of lines, each line being a sequence of two strictly positive integers separated by at least one space, with possibly spaces before and after the first and second number, respectively, the first numbers listed from the first line to the last line forming a strictly increasing sequence. The first number represents the distance (say in kilometres) from a point on the coast to a fishing town further down the coast (so the towns are listed as if we were driving down the coast from some fixed point); the second number represents the quantity (say in kgs) of fish that has been caught during the early hours of the day by that town's fishermen. For instance, the contents of the file **towns_1.txt** can be displayed as:

```
5 70
15 100
1200 20
```

- which corresponds to the case where we have 3 towns, one situated 5 km south the point, a second one situated 15 km south the point, and a third one situated 1200 km south the point, with 70, 100 and 20 kgs of fish being caught by those town's fishermen, respectively.
- The aim is to maximise the quantity of fish available in all towns (the same in all towns) by possibly transporting fish from one town to another one, but unfortunately losing 1 kilo of fish per kilometre. For instance, if one decides to send 20 kilos of fish from the second town to the first one, then the second town ends up having 100 - 20 = 80 kgs of fish, whereas the first one ends up having 70 + 20 - (15 - 5) = 80 kgs of fish too.
- The program outputs that maximum quantity of fish that all towns can have by possibly transporting fish in an optimal way.

See the possible interaction with the two provided sample files in Section 5 below.

5. Sample Outputs (or Test Cases)

Here are a few tests together with the expected outputs. The outputs of your program should be exactly as shown:

```
$ python3 boundary.py

Which data file do you want to use? rectangles_1.txt

The perimeter is: 228
```

```
$ python3 boundary.py

Which data file do you want to use? rectangles_2.txt

The perimeter is: 9090
```

```
$ python3 moving_die.py

Enter the desired goal cell number: A

Incorrect value, try again

Enter the desired goal cell number:

Incorrect value, try again

Enter the desired goal cell number: -1

Incorrect value, try again

Enter the desired goal cell number: 0

Incorrect value, try again

Enter the desired goal cell number: 1

On cell 1, 3 is at the top, 2 at the front, and 1 on the right.
```

```
$ python3 moving_die.py

Enter the desired goal cell number: 29

On cell 29, 3 is at the top, 2 at the front, and 1 on the right.
```

```
$ python3 moving_die.py

Enter the desired goal cell number: 2006

On cell 2006, 4 is at the top, 1 at the front, and 2 on the right.
```

```
$ python3 fishing_towns.py

Which data file do you want to use? towns_1.txt

The maximum quantity of fish that each town can have is 20.
```

```
$ python3 fishing_towns.py

Which data file do you want to use? towns_2.txt

The maximum quantity of fish that each town can have is 415.
```