Assignment 1

COMP9021, Term 1, 2021

1. General matters

1.1. Aims. The purpose of the assignment is to:

- let you design solutions to simple problems;
- let you implement these solutions in the form of short python programs;
- practice the use of arithmetic computations, string manipulation, tests, repetitions, lists, tuples, dictionaries, reading from files.

1.2. Submission. Your programs will be stored in files named `pivoting_die.py`, `highest_scoring_words.py`, and `perimeter.py`. After you have developed and tested your program, upload it using Ed (unless you worked directly in Ed). Assignments can be submitted more than once; the last version is marked. Your assignment is due by March 29, 10:00pm.

1.3. Assessment. The first two exercises are worth 4 marks each, the third one, 5 marks. For all exercises, the automarking script will let each of your programs run for 30 seconds.

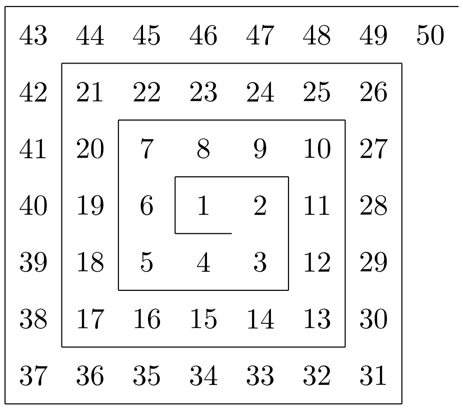
Late assignments will be penalised: the mark for a late submission will be the minimum of the awarded mark and 10 minus the number of full and partial days that have elapsed from the due date.

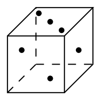
The outputs of your programs should be **exactly** as indicated.

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 routinely 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. Pivoting 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.





Write a program, stored in a file named `pivoting_die.py`, that prompts the user for a natural number N at least equal to 1, and outputs the numbers at the top, the front and the right after the die has been moved to cell N.

Here is a possible interaction.

```
$ python3 pivoting_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 pivoting_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 pivoting_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.
```

3. A word game (4 marks)

Write a program, stored in a file named `highest_scoring_words.py`, that performs the following task.

- Prompts the user to input between 3 and 10 lowercase letters (with possibly whitespace inserted anywhere); if the input contains too few or too many lowercase letters or any character which is neither a lowercase letter nor whitespace, then the program outputs an error message and exits.
- Finds in the file `wordsEn.txt`, assumed to be stored in the working directory, the words built from the letters input by the user (with the exclusion of any other character) with highest score, if any; the score of a word is defined as the sum of the values of the letters that make up that word, the value of each letter being defined as follows:

a	2	b	5	c	4	d	4	e	1	f	6
g	5	h	5	i	1	j	7	k	6	l	3
m	5	n	2	o	3	p	5	q	7	r	2
s	1	t	2	u	4	v	6	w	6	x	7
y	5	z	7								

- Outputs a specific message if there is no such word; otherwise, outputs the highest score and all words with that score, one word per line, with a different introductory message depending on whether there is a unique such word (in which case the introductory message is on the same line as the word) or at least two such words (in which case the introductory message is on a line of its own and all words are preceded with 4 spaces).

Here is a possible interaction.

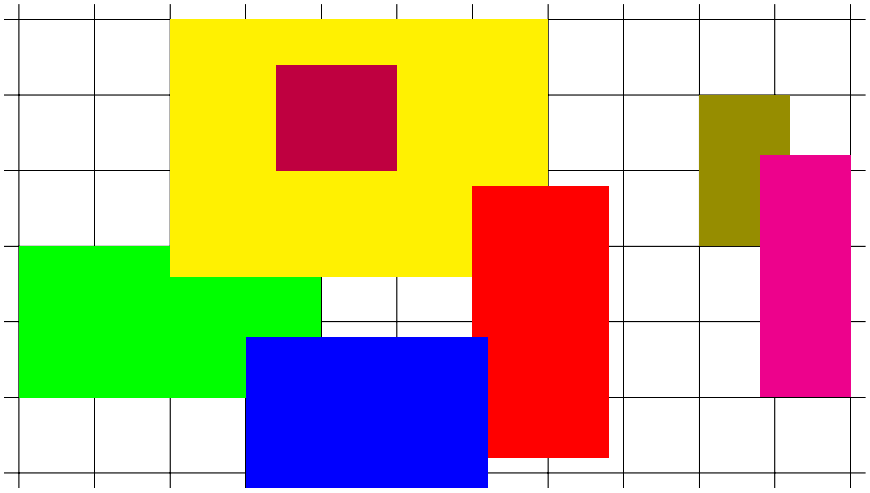
```
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: abc2ef
Incorrect input, giving up...
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: ab
Incorrect input, giving up...
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: abcdefghijk
Incorrect input, giving up...
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: zz zz zz
No word is built from some of those letters.
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: a  a   a
The highest score is 2.
The highest scoring word is a
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: a e i o u
The highest score is 8.
The highest scoring words are, in alphabetical order:
    iou
    oui
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: prmgroa
The highest score is 24.
The highest scoring word is program
```

```
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: a b  e    o    r   a t
The highest score is 16.
The highest scoring word is abator
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: r a m m o x y
The highest score is 17.
The highest scoring words are, in alphabetical order:
    mayor
    moray
    moxa
    oryx
$ python3 highest_scoring_words.py
Enter between 3 and 10 lowercase letters: eaeo rtsmn
The highest score is 17.
The highest scoring words are, in alphabetical order:
    matrons
    transom
```

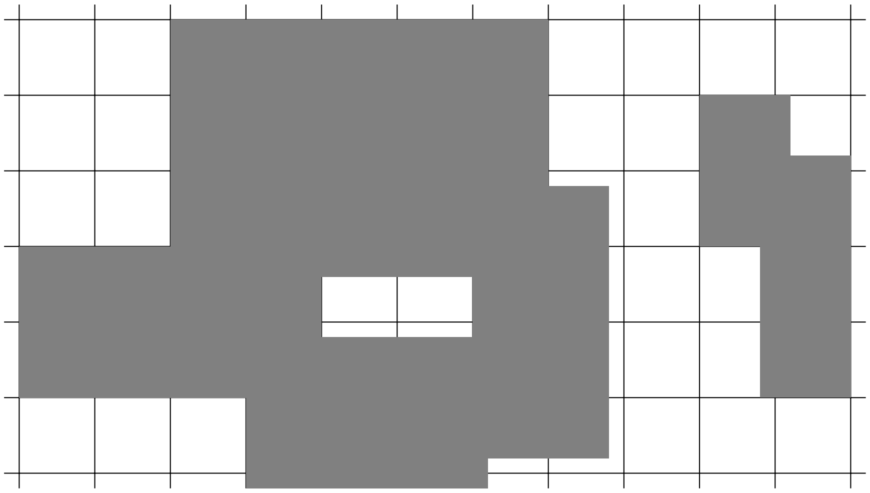
4. Overlapping photographs (5 marks)

Write a program, stored in a file named `perimeter.py`, that performs the following task.

- Prompts the user to input the name of 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 x_1 y_1 x_2 y_2 where (x_1, y_1) and (x_2, y_2) are meant to represent the coordinates of two opposite corners of a rectangle. With the provided file `frames_1.txt`, the rectangles can be represented as follows, using from top bottom the colours green, yellow, red, blue, purple, olive, and magenta.



  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 `frames_1.txt`, the sum of the lengths of the (external or internal) sides of the following picture.



Here is a possible interaction with the two provided sample files.

```
$ python3 perimeter.py
Which data file do you want to use? frames_1.txt
The perimeter is: 228
$ python3 perimeter.py
Which data file do you want to use? frames_2.txt
The perimeter is: 9090
```