Assignment 2

COMP9021, Trimester 1, 2019

1. General matter

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

- design and implement an interface based on the desired behaviour of an application program;
- practice the use of Python syntax;
- develop problem solving skills.

1.2. Submission. Your program will be stored in a file named `tangram.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 April 28, 11:59pm.

1.3. Assessment. The assignment is worth 10 marks. It is going to be tested against a number of input files. For each test, the automarking script will let your program 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.

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. Background

The game of tangram consists in creating shapes out of pieces. We assume that each piece has its own colour, different to the colour of any other piece in the set we are working with. Just for reference, here is the list of colours that are available to us (you will not make use of this list):

https://www.w3.org/TR/2011/REC-SVG11-20110816/types.html#ColorKeywords

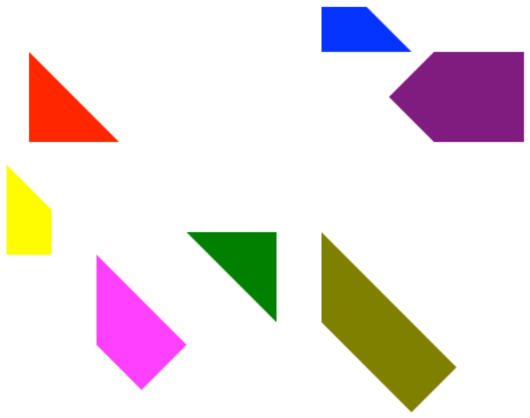
A representation of the pieces will be stored in an .xml file thanks to a simple, fixed syntax.

2.1. **Pieces.**

Here is an example of the contents of the file pieces_A.xml, typical of the contents of any file of this kind (so only the number of pieces, the colour names, and the various coordinates can differ from one such file to another–we do not bother with allowing for variations, in the use of space in particular).

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <path d="M 50 50 L 50 90 L 90 90 z" fill="red"/>
    <path d="M 160 170 L 160 130 L 120 130 z" fill="green"/>
    <path d="M 200 30 L 180 30 L 180 50 L 220 50 z" fill="blue"/>
    <path d="M 40 100 L 40 140 L 60 140 L 60 120 z" fill="yellow"/>
    <path d="M 210 70 L 230 90 L 270 90 L 270 50 L 230 50 z" fill="purple"/>
    <path d="M 180 130 L 180 170 L 220 210 L 240 190 z" fill="olive"/>
    <path d="M 100 200 L 120 180 L 80 140 L 80 180 z" fill="magenta"/>
</svg>
```

Opened in a browser, pieces_A.xml displays as follows:



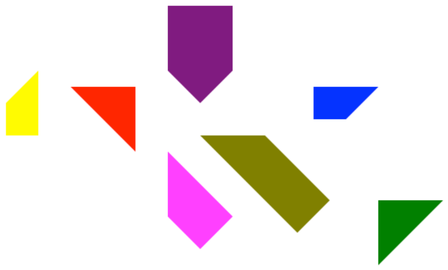
Note that the coordinates are nonnegative integers. This means that the sets of pieces we consider rule out those of the traditional game of tangram, where $\sqrt{2}$ is involved everywhere...

We require every piece to be a **convex** polygon. An .xml file should represent a piece with n sides ($n \geq 3$) by an enumeration of n pairs of coordinates, those of consecutive vertices, the first vertex being arbitrary, and the enumeration being either clockwise or anticlockwise.

The pieces can have a different orientation and be flipped over. For instance, the file pieces_AA.xml whose contents is

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <path d="M 50 50 L 50 90 L 90 90 z" fill="red"/>
    <path d="M 160 170 L 160 130 L 120 130 z" fill="green"/>
    <path d="M 200 30 L 180 30 L 180 50 L 220 50 z" fill="blue"/>
    <path d="M 40 100 L 40 140 L 60 140 L 60 120 z" fill="yellow"/>
    <path d="M 210 70 L 230 90 L 270 90 L 270 50 L 230 50 z" fill="purple"/>
    <path d="M 180 130 L 180 170 L 220 210 L 240 190 z" fill="olive"/>
    <path d="M 100 200 L 120 180 L 80 140 L 80 180 z" fill="magenta"/>
</svg>
```

and which displays as



represents the same set of pieces (the fact that the latter appear as smaller than the former is just due to the different scaling of the included pdf's; the sizes of the pieces are actually the same in terms of the coordinates of their vertices).

The pieces can overlap, but that does not concern us. In practice, we will just use representations where the pieces do not overlap as that allows us to visualise the pieces properly when we open the corresponding .xml file, but it is just for convenience and irrelevant to the tasks we tackle.

2.2. **Shapes.**

A representation of a shape is provided thanks to an .xml file with the same structure, storing the coordinates of the vertices of just one polygon.

The file shape_A_1.xml whose contents is

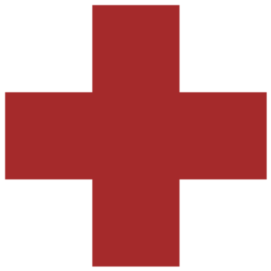
```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <path d="M 30 20 L 110 20 L 110 120 L 30 120 z" fill="grey"/>
</svg>
```

and which displays as

is such an example. The file shape_A_2.xml whose contents is

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <path d="M 50 10 L 90 10 L 90 50 L 130 50 L 130 90 L 90 90 L 90 130...
                       ...L 50 130 L 50 90 L 10 90 L 10 50 L 50 50 z" fill="brown"/>
</svg>
```

and which displays as



is another such example.

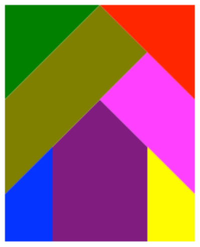
Contrary to pieces, shapes are not assumed to be convex polygons. Still they are assumed to be **simple** polygons (the boundary of a simple polygon does not cross itself; in particular, it cannot consist of at least 2 polygons that are connected by letting two of them just “touch” each other at one of their vertices–*e.g.*, two rectangles such that the upper right corner of one rectangle is the lower left corner of the other rectangle; that is not allowed).

Whereas you will have to check that the representation of the pieces in an `.xml` file satisfies our constraints, you will not have to do so for the representation of a shape; you can assume that any shape we will be dealing with satisfies our constraints.

2.3. **Tangrams.** The first shape can be built from our set of pieces, in many ways. Here is one, given by the file `tangram_A_1_a.xml` whose contents is

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <path d="M 30 60 L 30 20 L 70 20 z" fill="green"/>
    <path d="M 50 120 L 50 80 L 70 60 L 90 80 L 90 120 z" fill="purple"/>
    <path d="M 30 100 L 90 40 L 70 20 L 30 60 z" fill="olive"/>
    <path d="M 70 60 L 110 100 L 110 60 L 90 40 z" fill="magenta"/>
    <path d="M 50 120 L 30 120 L 30 100 L 50 80 z" fill="blue"/>
    <path d="M 110 20 L 70 20 L 110 60 z" fill="red"/>
    <path d="M 110 100 L 110 120 L 90 120 L 90 80 z" fill="yellow"/>
</svg>
```

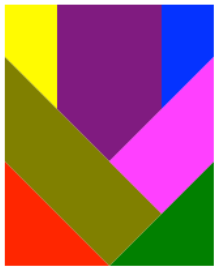
and which displays as follows.



Here is another one, given by the file `tangram_A_1_b.xml` whose contents is

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <path d="M 30 20 L 50 20 L 50 60 L 30 40 z" fill="yellow"/>
    <path d="M 50 60 L 50 20 L 90 20 L 90 60 L 70 80 z" fill="purple"/>
    <path d="M 70 120 L 110 80 L 110 120 z" fill="green"/>
    <path d="M 90 20 L 110 20 L 110 40 L 90 60 z" fill="blue"/>
    <path d="M 30 120 L 30 80 L 70 120 z" fill="red"/>
    <path d="M 70 120 L 30 80 L 30 40 L 90 100 z" fill="olive"/>
    <path d="M 70 80 L 110 40 L 110 80 L 90 100 z" fill="magenta"/>
</svg>
```

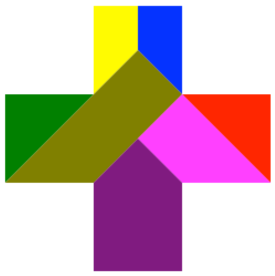
and which displays as follows.



The second shape can also be built from our set of pieces, in many ways. Here is one, given by the file tangram_A_2_a.xml whose contents is

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <path d="M 30 60 L 30 20 L 70 20 z" fill="green"/>
    <path d="M 50 120 L 50 80 L 70 60 L 90 80 L 90 120 z" fill="purple"/>
    <path d="M 30 100 L 90 40 L 70 20 L 30 60 z" fill="olive"/>
    <path d="M 70 60 L 110 100 L 110 60 L 90 40 z" fill="magenta"/>
    <path d="M 50 120 L 30 120 L 30 100 L 50 80 z" fill="blue"/>
    <path d="M 110 20 L 70 20 L 110 60 z" fill="red"/>
    <path d="M 110 100 L 110 120 L 90 120 L 90 80 z" fill="yellow"/>
</svg>
```

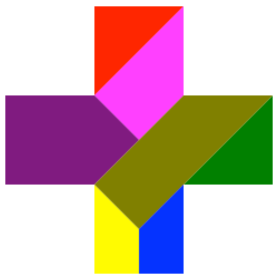
and which displays as follows.



Here is another one, given by the file tangram_A_2_b.xml whose contents is

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <path d="M 30 20 L 50 20 L 50 60 L 30 40 z" fill="yellow"/>
    <path d="M 50 60 L 50 20 L 90 20 L 90 60 L 70 80 z" fill="purple"/>
    <path d="M 70 120 L 110 80 L 110 120 z" fill="green"/>
    <path d="M 90 20 L 110 20 L 110 40 L 90 60 z" fill="blue"/>
    <path d="M 30 120 L 30 80 L 70 120 z" fill="red"/>
    <path d="M 70 120 L 30 80 L 30 40 L 90 100 z" fill="olive"/>
    <path d="M 70 80 L 110 40 L 110 80 L 90 100 z" fill="magenta"/>
</svg>
```

and which displays as follows.



3. First task (3 marks)

You have to check that the pieces represented in an `.xml` file satisfy our constraints. So you have to check that each piece is convex, and if it represents a polygon with n sides ($n \geq 3$) then the representation consists of an enumeration of the n vertices, either clockwise or anticlockwise. Here is the expected behaviour of your program.

```
$ python3
...
>>> from tangram import *
>>> file = open('pieces_A.xml')
>>> coloured_pieces = available_coloured_pieces(file)
>>> are_valid(coloured_pieces)
True
>>> file = open('pieces_AA.xml')
>>> coloured_pieces = available_coloured_pieces(file)
>>> are_valid(coloured_pieces)
True
>>> file = open('incorrect_pieces_1.xml')
>>> coloured_pieces = available_coloured_pieces(file)
>>> are_valid(coloured_pieces)
False
>>> file = open('incorrect_pieces_2.xml')
>>> coloured_pieces = available_coloured_pieces(file)
>>> are_valid(coloured_pieces)
False
>>> file = open('incorrect_pieces_3.xml')
>>> coloured_pieces = available_coloured_pieces(file)
>>> are_valid(coloured_pieces)
False
>>> file = open('incorrect_pieces_4.xml')
>>> coloured_pieces = available_coloured_pieces(file)
>>> are_valid(coloured_pieces)
False
```

Note that the function `are_valid()` does not print out `True` or `False`, but returns `True` or `False`.

4. Second task (3 marks)

You have to check whether the sets of pieces represented in two .xml files are identical, allowing for pieces to be flipped over and allowing for different orientations. Here is the expected behaviour of your program.

```
$ python3
...
>>> from tangram import *
>>> file = open('pieces_A.xml')
>>> coloured_pieces_1 = available_coloured_pieces(file)
>>> file = open('pieces_AA.xml')
>>> coloured_pieces_2 = available_coloured_pieces(file)
>>> are_identical_sets_of_coloured_pieces(coloured_pieces_1, coloured_pieces_2)
True
>>> file = open('shape_A_1.xml')
>>> coloured_pieces_2 = available_coloured_pieces(file)
>>> are_identical_sets_of_coloured_pieces(coloured_pieces_1, coloured_pieces_2)
False
```

Note that the function `identical_sets_of_coloured_pieces()` does not print out `True` or `False`, but returns `True` or `False`.

5. Third task (4 marks)

You have to check whether the pieces represented in an .xml file are a solution to a tangram puzzle represented in another .xml file. Here is the expected behaviour of your program.

```
$ python3
...
>>> from tangram import *
>>> file = open('shape_A_1.xml')
>>> shape = available_coloured_pieces(file)
>>> file = open('tangram_A_1_a.xml')
>>> tangram = available_coloured_pieces(file)
>>> is_solution(tangram, shape)
True
>>> file = open('tangram_A_2_a.xml')
>>> tangram = available_coloured_pieces(file)
>>> is_solution(tangram, shape)
False
```

Note that the function `is_solution()` does not print out `True` or `False`, but returns `True` or `False`.