Assignment 2

COMP9021, Trimester 1, 2025

1 General matters

1.1 Aim

The purpose of the assignment is to:

- develop your problem solving skills;
- design and implement the solutions to problems in the form of medium sized Python programs;
- design and implement an interface based on the desired behaviour of an application program;
- organise code into classes and implement special methods;
- possibly practice using the `re` and `numpy` modules;
- optionally practice the design and implementation of search techniques, with recursion as a good approach.

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, 11:59am.

1.3 Assessment

The assignment is worth 13 marks, plus 5 bonus marks. It is going to be tested against a number of inputs. For each test, the automarking script will let your program run for 30 seconds.

Assignments can be submitted up to 5 days after the deadline. The maximum mark obtainable reduces by 5% per full late day, for up to 5 days. Thus if students A and B hand in assignments worth 12 and 11, both two days late (that is, more than 24 hours late and no more than 48 hours late), then the maximum mark obtainable is 11.7, so A gets $\min(11.7, 12) = 11.7$ and B gets $\min(11.7, 11) = 11$.

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

Important notice:

- The bonus mark is an option **only for programs submitted by the deadline, NOT for programs submitted within 5 days after the deadline**.
- The final mark for the course is capped to 100; the bonus mark cannot let you score more than 100 in the course.

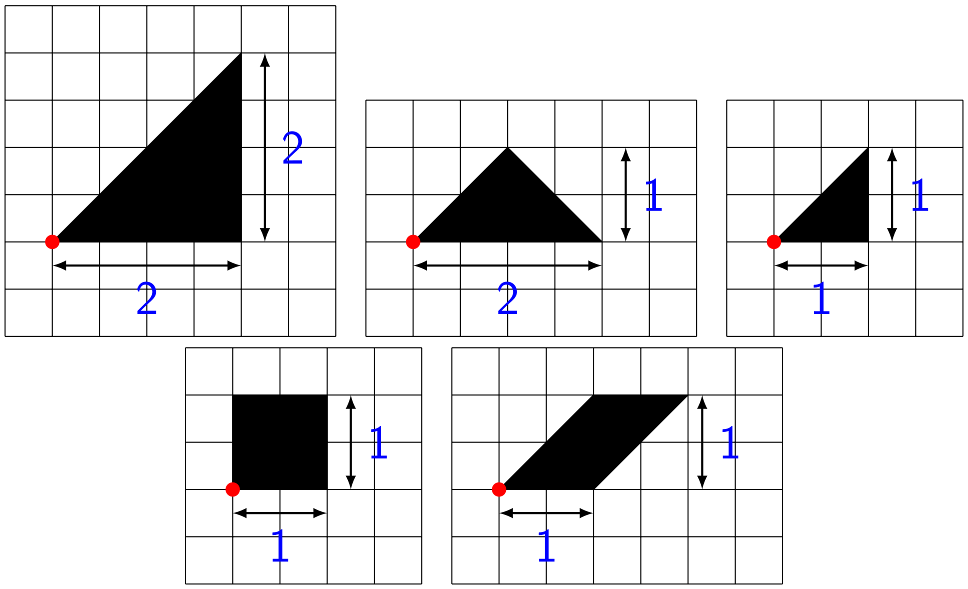
1.4 Reminder on plagiarism policy

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 Tangram puzzles and the TangramTikz package

See Tangram for an overview on Tangram puzzles. All shapes under consideration consist of a unique connected component without inside hole.

We play with 2 large triangles, a medium triangle, 2 small triangles, a square and a parallelogram of the following sizes in centimetres, first positioned as shown, the red dot playing the role of an "anchor".



Each piece can be flipped about the vertical line that goes through the anchor, flipped about the horizontal line that goes through the anchor, and rotated about its anchor (as if it was a pivot) by a multiple of 45 degrees, before being moved to let the anchor be positioned at given x- and y-coordinates.

We work with `.tex` files, any of which can be given as argument to `pdflatex` to generate a `.pdf` file.

- Here is a `.tex` file for the solved Tangram of a kangaroo, and here is the associated `pdf` file.
- Here is a `.tex` file for the solved Tangram of a cat, and here is the associated `pdf` file.
- Here is a `.tex` file for the solved Tangram of a goose, and here is the associated `pdf` file.

More precisely, the `.tex` files we work with have the following structure:

```
\documentclass{standalone}
\usepackage{TangramTikz}
\begin{document}
\begin{EnvTangramTikz}
...
\end{EnvTangramTikz}
\end{document}
```

Between `\begin{EnvTangramTikz}` and `\end{EnvTangramTikz}` are 7 lines, one for each piece, that read as:

```
\PieceTangram[TangSol](...,...){...}
```

or as

```
\PieceTangram[TangSol]<...>(...,...){...}
```

`...` between the curly braces at the end is one of

- `TangGrandTri` for any of the two large triangles,
- `TangMoyTri` for the medium triangle,
- `TangPetTri` for any of the two small triangles,
- `TangCar` for the square, and
- `TangPara` for the parallelogram.

Both `...` within the pair of parentheses are for the x- and y-coordinates of the point where the anchor of the piece has to be moved, after the piece has been possibly flipped and rotated. These coordinates are of the form *a*, *a*`+sqrt(2)`, *a*`-sqrt(2)`, *a*`+`*b*`*sqrt(2)`, *a*`-`*b*`*sqrt(2)`, `sqrt(2)`, `+sqrt(2)`, `-sqrt(2)` or *a*`*sqrt(2)` where *a* and *b* are integers or floating point numbers with *b* being 0 or strictly positive. Floating point numbers have a dot in their representation. When positive, *a* can start with `+` whereas *b* can't. Floating point numbers can start with any number of `0`s, and `0` before the dot is optional in floating point numbers whose absolute value is smaller than 1 (essentially, any sequence of digits and one dot that Python accepts as a floating point literal is valid).

`...` between the less than and greater than signs is for one, two or three distinct options, consecutive options being separated by a comma. The possible options are:

- `xscale=-1`, to flip the piece so left becomes right and right becomes left,
- `yscale=-1`, to flip the piece so top becomes bottom and bottom becomes top, and
- `rotate=_` with `_` an integer that is a multiple of 45 (positive values are for anticlockwise rotation, negative values are for clockwise rotation).

The options can appear in any order. As making both flips is equivalent to rotating the piece by 180 degrees, it is always possible to use at most two options. The TangramTikz package is implemented in such a way that when a piece is both rotated and flipped, the rotation is performed before the flip, irrespective of the order in which the options are given.

In the whole `.tex` file, there can be blank lines and whitespace (space and tab characters) between tokens everywhere. Also, the leftmost occurrence of a `%` marks the beginning of a comment that runs from this symbol included all the way to the end of the physical line, including the `\n` character.

3 The tasks to perform

Your program will allow `TangramPuzzle` objects to be created from `.tex` files that you can assume are stored in the working directory, and whose contents satisfy all conditions spelled out in Section 2.

3.1 Reporting on the transformations applied to each piece (3 marks)

A `TangramPuzzle` object has a `transformations` attribute, whole value is a dictionary with 7 keys, one for each piece. The keys are `'Large triangle 1'`, `'Large triangle 2'`, `'Medium triangle'`, `'Small triangle 1'`, `'Small triangle 2'`, `'Square'` and `'Parallelogram'`. The order of the large and small triangles are given by the order they have in the `.tex` file. More generally, the keys of the dictionary are created in the order of the corresponding pieces in the `.tex` file. The value for each key is itself a dictionary whose keys are `'xflip'`, `'yflip'` and `'rotate'`. The values for `'xflip'` and `'yflip'` are `False` for at least one of them, and `True` for at most one of them, as making both flips is equivalent to rotating the piece by 180 degrees. The value for `'rotate'` is an integer between `0` included and `360` excluded.

Here is a possible interaction.

3.2 Reporting on the coordinates of the vertices (4 marks)

Printing out a `TangramPuzzle` object outputs for each piece, the coordinates of the vertices.

- Pieces are listed from the ones whose leftmost topmost vertices are highest to the ones whose leftmost topmost vertices are lowest. When the heights of the leftmost topmost vertices of two pieces are the same, the piece that is to the left of the other piece is listed first. Essentially, the plane is scanned from top to bottom and from left to right and when the leftmost topmost vertex of a piece is encountered, the piece is listed.
- Coordinates have the simplest possible form, as shown in the provided examples here. Binary `+` and `-` have a space on both sides, there is no addition of 0, there is no multiplication by 0, there is no multiplication by 1 or -1, fractions are simplified, integers are used instead of fractions whose denominator would be 1, and fractions are surrounded by parentheses when followed by `√2` to avoid that it be wrongly believed that $\sqrt{2}$ is multiplied by the denominator.
- For a given piece, the enumeration of vertices proceeds clockwise starting from the leftmost topmost vertex.

3.3 Creating a file to represent the pieces of the solved puzzle (3 marks)

The `TangramPuzzle` class has a `draw_pieces()` method that takes the name of a `.tex` file as argument, to represent a solved tangram using the `tikz` package rather than the `TangramTiks` package, easily taking advantage of what has been done for the previous part. The order of the pieces, the order of the vertices of a given piece, are the same as for the previous part. The origin of the plane is represented by a red dot. The grid in the background extends in such a way that there is at least one square (of 5mm by 5mm) and strictly less than 2 squares above the topmost vertex, to the right of the rightmost vertex, below the bottommost vertex and to the left of the leftmost vertex. For example: if the x-coordinate of the rightmost vertex of the shape is equal to 3.01 or 3.5, then the grid extends to the right to an x-coordinate of 4; if the x-coordinate of the rightmost vertex of the shape is equal to 3.51 or 4, then the grid extends to the right to an x-coordinate of 4.5.

Executing

```
$ python3
...
>>> from tangram import *
>>> TangramPuzzle('kangaroo.tex').draw_pieces('kangaroo_pieces_on_grid.tex')
>>> TangramPuzzle('cat.tex').draw_pieces('cat_pieces_on_grid.tex')
>>> TangramPuzzle('goose.tex').draw_pieces('goose_pieces_on_grid.tex')
```

produces this, this and this `.tex` files, respectively, with this, this and this associated `.pdf` files.

It is advisable to make sure that the spaces in the `.tex` files produced during the interaction are exactly as shown. Still, whitespace will be ignored for assessment purposes, but of course, all other nonspace characters have to be exactly the same, character for character, with all lines in the same order.

3.4 Creating a file to represent the outline of the solved puzzle (3 marks)

The `TangramPuzzle` class has a `draw_outline()` method that takes the name of a `.tex` file as argument, to represent a solved tangram by drawing the outline of its shape as opposed to each piece, starting from the leftmost topmost vertex and moving in a clockwise direction. Straight line segments extend as much as possible between vertices. Again, the origin is represented by a red dot. Again, the grid in the background extends in such a way that there is at least one square (of 5mm by 5mm) and strictly less than 2 squares above the topmost vertex, to the right of the rightmost vertex, below the bottommost vertex and to the left of the leftmost vertex.

Executing

```
$ python3
...
>>> from tangram import *
>>> TangramPuzzle('kangaroo.tex').draw_outline('kangaroo_outline_on_grid.tex')
>>> TangramPuzzle('cat.tex').draw_outline('cat_ouline_on_grid.tex')
>>> TangramPuzzle('goose.tex').draw_outline('goose_outline_on_grid.tex')
```

produces this, this and this `.tex` files, respectively, with this, this and this associated `.pdf` files.

Again, it is advisable to make sure that the spaces in the `.tex` files produced during the interaction are exactly as shown. Still, whitespace will be ignored for assessment purposes, but of course, all other nonspace characters have to be exactly the same, character for character, with all lines in the same order.

3.5 Solving a puzzle (5 bonus marks)

A function `solve_tangram_puzzle()` takes a file name as argument and returns a `TangramPuzzle` object, making the following interaction successful:

```
$ python3
...
>>> from tangram import *
>>> solve_tangram_puzzle('kangaroo_outline_on_grid.tex').draw_pieces('solved_kangaroo.tex')
>>> solve_tangram_puzzle('cat_outline_on_grid.tex').draw_pieces('solved_cat.tex')
>>> solve_tangram_puzzle('goose_outline_on_grid.tex').draw_pieces('solved_goose.tex')
>>> ^D
$ diff kangaroo_pieces_on_grid.tex solved_kangaroo.tex
$ diff cat_pieces_on_grid.tex solved_cat.tex
$ diff goose_pieces_on_grid.tex solved_goose.tex
$
```