National University of Singapore
School of Computing
CS1010X: Programming Methodology
Semester II, 2019/2020

# Mission 1
# Rune Reading

Release date: 18 January 2020
**Due: 26 January 2020, 23:59**

## Required Files

- mission01-template.py
- runes.py
- graphics.py
- PyGif.py

## Background

You are a newly enrolled wizard in the Python Institute for Mages (PIM) where only a small group of gifted humans are taught the sacred Python magic passed down from the ancient Python Gods. In this mission, you are expected to demonstrate your investigative skills. The ability to think and reason are necessary qualities to master Python magic.

The grandmaster wizard has brought all your fresh wizards to the secret cave at the back of PIM, where you will need to open three doors, to lead you another step closer towards the inner sanctum of PIM.

In the lecture, we demonstrated how Python can be used to generate runes. Now, you get to try your hand at drawing them. To do this, first open the file `runes.py` or `mission01-template.py` in IDLE and run the module. You need to ensure that all 4 files: `runes.py, graphics.py, PyGif.py, mission01-template.py` are in the same directory. A new window, called the viewport should appear. In `runes.py`, we defined the viewport (where your runes are going to be drawn) and the four runes discussed in class `rcross_bb`, `sail_bb`, `corner_bb`, and `nova_bb`. For example, you can display `rcross_bb` in the viewport with the following command:

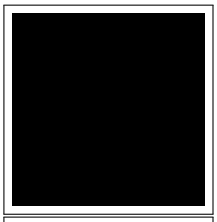
```
show(rcross_bb)
```

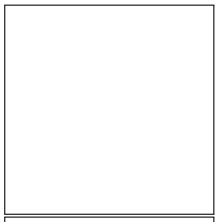
Don't forget to clear the viewport when necessary with `clear_all()`.

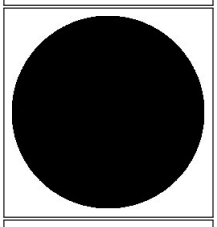
Also defined in `runes.py` are the following functions as discussed in class:

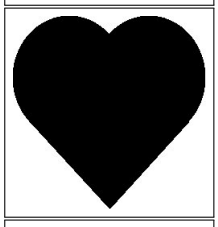
- `stack`
- `stack_frac`
- `quarter_turn_right`
- `eighth_turn_left`
- `flip_horiz`
- `flip_vert`
- `turn_upside_down`
- `quarter_turn_left`
- `beside`
- `make_cross`
- `repeat_pattern`
- `stackn`

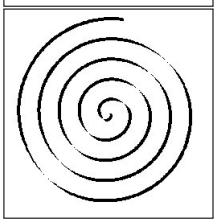
In addition to the ones you saw in lecture, we have also defined a few more basic runes that you can use:

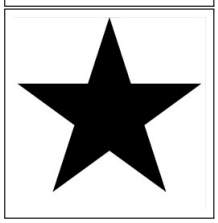
`black_bb`: 

`blank_bb`: 

`circle_bb`: 

`heart_bb`: 

`ribbon_bb`: 

`pentagram_bb`: 

In writing rather large, complex programs, one does not often understand (or even see) every single line of code, since such programs are usually written by several people, each in charge of smaller components. The key idea in functional abstraction is that as the programmer, you need not understand how each function works. All you need to know is what each function does and its signature (such as what parameters to pass). More specifically, **you need not read the code for the predefined functions listed above**. You may refer to the lecture slides to understand what arguments each function requires and its corresponding effect. Now we will use these functions as primitive building blocks to draw our own pictures.

This mission consists of **two** tasks.

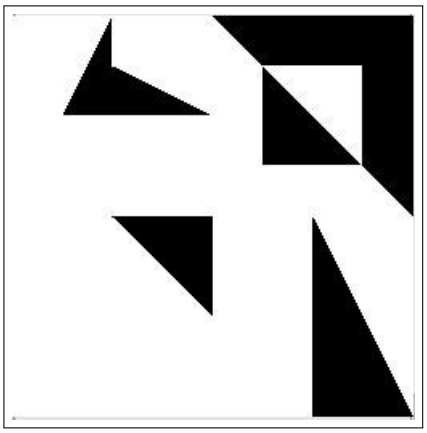
## Task 1: Mosaic (4 marks)

On the first door you find 4 basic runes and one complex rune separated by an empty space. Clearly, your task would be to fill in this space with a descriptive function for the manipulation of the 4 basic runes to create the complex one.

Write a function `mosaic` that takes four runes as arguments and arranges them in a $2 \times 2$ square, starting with the top-right corner, going clockwise. In particular, the command

```
show(mosaic(rcross_bb, sail_bb, corner_bb, nova_bb))
```

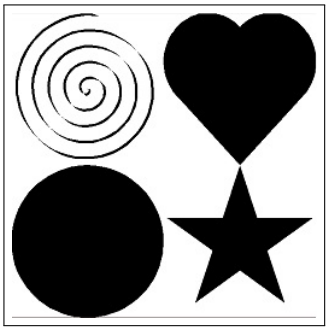
should draw the following:



and

```
show(mosaic(heart_bb, pentagram_bb, circle_bb, ribbon_bb))
```

should draw the following:



Note: the white pentagon inside `pentagram_bb` might not show up on your computer.

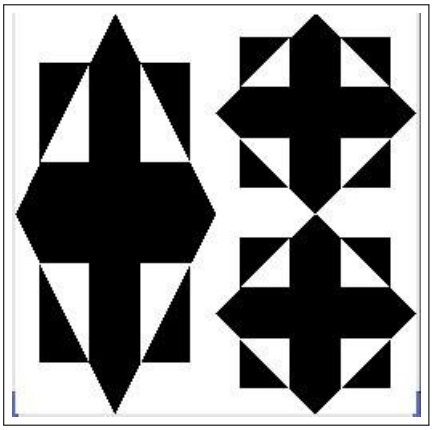
## Task 2: Simple Fractal (4 marks)

On the second door, you find 2 runes displayed in a similar fashion. The only difference would be that the complex rune now exhibits a different layout.

Write a function `simple_fractal` that takes a rune as an argument and returns a rune consisting of the original rune and a pair of runes stacked on top of each other on its right. For example, the following command:

```
show(simple_fractal(make_cross(rcross_bb)))
```

should draw the following:



and

```
show(simple_fractal(heart_bb))
```

should draw the following: