Computer Science AP/X CSCI-140/242
Tiny Turtle Lab 1

8/23/2018

1 Requirements

You will individually implement a program, `tinyturtle.py`, that can correctly interpret the complete set of basic and enhanced TT commands. Recall from the problem solving writeup:

Table 1: Basic TT commands

TT Command	Argument	Turtle Command
`F`	###	`forward(###)`
`B`	###	`backward(###)`
`L`	###	`left(###)`
`R`	###	`right(###)`
`C`	###	`circle(###)`
`U`	None	`up()`
`D`	None	`down()`

Table 2: Enhanced TT commands

Command	Arguments	Meaning
`I`	`# ... @`	Iterate ... # times
`P`	`# ###`	Draw polygon of # sides & ### side length

Here, `#` is a single digit (`0-9`), and white space (e.g. space, newline, tab) is ignored.

1.1 Program Execution

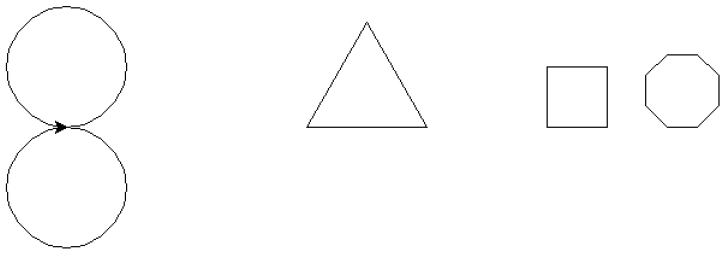
The program should execute as follows:

1. First prompt for a string containing any combination of basic and enhanced TT commands, as a single string.
2. The expanded TT program should be displayed to standard output. It should contain only the basic TT commands.
3. As each basic TT command is interpreted, the corresponding turtle command should be displayed to standard output. For example, if the basic TT command is `F100`, the output should be `forward(100)`.
4. The program should "idle on" the `mainloop` call until the user closes the drawing window.

1.2 Sample Run

```
Enter Tiny Turtle program (CMD+D or CTRL+D to terminate):
P3 100 U F200 D P4 050 U F100 D P8 025 U B500 D I2 C050 L180 @
Expanded program:
F100 L120 F100 L120 F100 L120 U F200 D F050 L090 F050 L090 F050 L090 F050 L090 U F100
D F025 L045 F025 L045 F025 L045 F025 L045 F025 L045 F025 L045 F025 L045 F025 L045 U
B500 D C050 L180 C050 L180
Evaluating...
forward(100)
left(120)
forward(100)
left(120)
forward(100)
left(120)
up()
forward(200)
down()
forward(50)
left(90)
forward(50)
left(90)
forward(50)
left(90)
forward(50)
left(90)
up()
forward(100)
down()
forward(25)
left(45)
forward(25)
left(45)
forward(25)
left(45)
forward(25)
left(45)
forward(25)
left(45)
forward(25)
left(45)
forward(25)
left(45)
forward(25)
left(45)
up()
backward(500)
down()
circle(50)
left(180)
circle(50)
left(180)
```

Using the sample input from above, the following should be graphically displayed.



2 Implementation Details

You are not allowed to use regular expressions for this lab. Everything should be done using `str` operations, e.g. slicing, concatenation, `index`, etc.

Additionally, you are not allowed to use the `global` keyword.

In order to receive full design points, you should be using functions to break down your program. It is a good idea to have one function that expands the entire set of TT commands into the basic ones, and another function for interpreting just the basic TT commands. Look at your problem solving for more hints about how you can break this down even further to promote function reuse.

Your program must be properly documented to receive full style credit. The program should have a main docstring containing your name and a description of the assignment. Each function should have a properly formatted docstring with a description, arguments, return, etc.

For the iterate command, you may assume all the arguments will be only basic TT commands. This means you do not have to deal with polygon commands, or nested loops. If you wish to implement your program so that it can handle these cases, you will receive extra credit. For example, you will receive extra credit for being able to handle:

1. `"I4 P3 100 F100 @"` - Draw 4 triangles, side by side.
2. `"I2 I4 F100 L090 @ F100 @"` - Draw 2 squares, side by side.

Regarding the polygon command, it is assumed the polygon will be drawn left to right. Also, the angle you compute to turn for each side of the polygon should be rounded to a 3 digit integer.

You do not have to deal with erroneous input. It is assumed that all commands are legal, and are all formatted correctly. You can always assume there will be a single space separating the arguments of a command, as well as the commands themselves.

3 Grading

The grade breakdown for this lab is as follows:

- Problem Solving: 15%
- In-Lab Activity: 10%
- Functionality: 50 (+10 bonus)%
  - Basic TT commands: 25%
  - Iterate command: 10%
  - Polygon command: 10%
  - Bonus: 10% - Iterate command handles polygon and nested iterate commands.
- Design: 10% - Your implementation uses functions to promote code reuse.
- Version Control: 5%
- Code Style and Documentation: 10%

4 Submission

4.1 Overview

All labs and projects will require you to use `Git` for source control, and submit on our CS machines using `try`. If you have not run through the Python setup instructions from the resources section of the course web page, you should do so now before continuing with this lab.

For this lab, you should mirror the `Git` and `try` instructions. A high level summary of the steps are:

1. In PyCharm, after creating a new project for your lab, create a local Git repository.
2. On the CS machines, create a new bare repository for this lab in your `Git` directory.
3. In PyCharm, commit and push your initial code. This is where you will connect your local code to the remote CS repository.
4. Clone your bare repository into your `Courses` directory.
5. Develop your lab in PyCharm. When ready to submit to `try`, commit and push to the bare repo.
6. In your `Courses` directory for this lab, do a `git pull` to receive the latest update of the code you committed.
7. Run `git log` to create a log file of all your commits.
8. Run the `try` command for this lab target to submit your source code and log file.

4.2 `try` Submission

The `try` command for this lab is:

```
try csapx-grd lab1-1 tinyturtle.py log.txt
```

Note, if you are in the top level directory for this lab and you created a `src` folder containing your implementation, you must specify it as `src/tinyturtle.py` when submitting through `try`.

Recall that to verify your latest submission has been saved successfully, you can run `try` with the query option, `-q`:

```
try -q csapx-grd lab1-1
```