# Computer Science I — CSCI-141

# ShapyTurtle — Lab 4

09/11/2020

## 1 Problem

*ShapyTurtle* is an interpreter for a 'shorthand language' that expresses turtle programs compactly as a *string*. **This interpreter is defined by a function that takes a program given as a string argument.**

Table 1: ShapyTurtle Commands

| Command | Effect |
|---|---|
| `<#` | Turn left # of degrees. |
| `>#` | Turn right # of degrees. |
| `S#` | Make a square with side length #. |
| `T#` | Make a equilateral triangle with side length #. |
| `C#` | Make a circle with radius #. |
| `F#` | Move forward distance #. |
| `B#` | Move backward distance #. |
| `U` | Pick the pen up. |
| `D` | Put the pen down. |
| `R#h,#w` | Make a rectangle with height #h and width #w, where the #h and #w are integral numeric values |
| `P#s,#l` | Make a polygon with s sides with side length #l, where the #s and #l are integral numeric values |
| `G#x,#y` | GoTo location #x, #y, where the #x and #x are integral numeric values |
| `Z#` | Change the color of the turtle pen based on #, which has a value from this table<br>0: red<br>1: blue<br>2: green<br>3: yellow<br>4: brown<br>other: black |

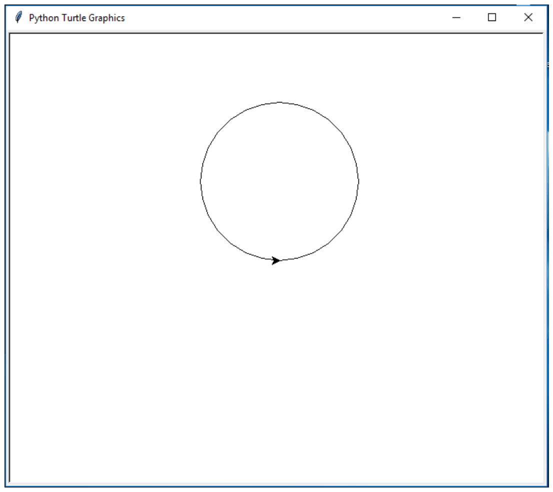
Note that all closed shapes are drawn from the turtle's current position. The turtle turns first before drawing the first side of the shape. Therefore the very last thing the turtle does is to draw the final side of the shape. It then follows that the turtle finishes in the same position and orientation from where it started.

Except for the `Z` command, all numerical arguments (#, #l, #x, ...) can be one or more numeric digits. The `Z` command takes exactly one numeric digit.

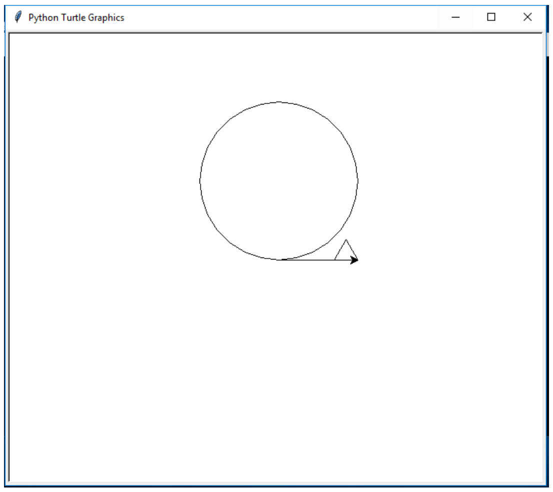
The ShapyTurtle instructions in Table 1 cause the turtle to draw or change its state. The interpreter of a ShapyTurtle program first clears the canvas and places the turtle starting at the origin, facing East (0° to the horizontal) with the turtle pen down before executing the program string.

All ShapyTurtle instructions begin at the current turtle orientation established by the initial default or by the execution of a prior command. Closed form commands return to the orientation at which they started.

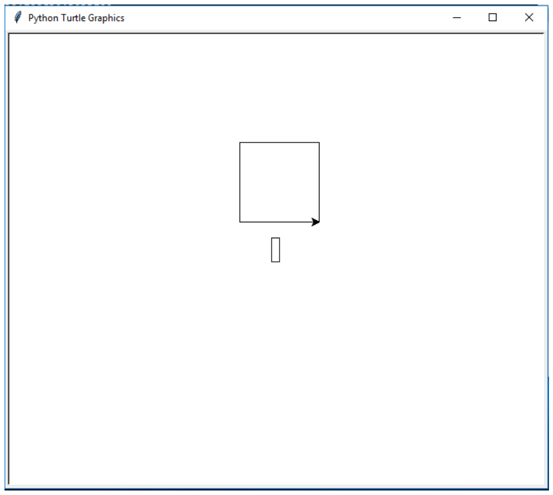
Below are some sample ShapyTurtle programs.



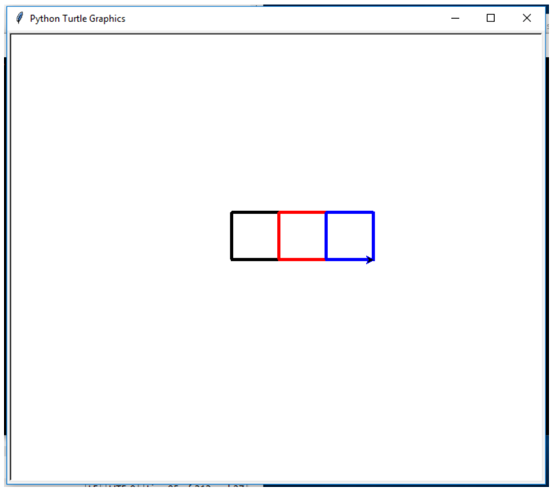
`C100`



`C100F100T30`



`R30,10UG50,50DS100`



`S60F60Z0S60F60Z1S60`

## 2 Problem-solving Session (15%)

Problem-solving involves teamwork as determined by the instructor. Below are the problems to answer during problem-solving.

1. What is the *image* drawn by the interpreter for these *program strings*:

   (a) ‘`<120F30<120F30<120F30`’

   (b) ‘`<90F77>45F77<45F77<45F77<45F77`’

   (c) ‘`F10<90F10>90F9<90F9>90F8<90F8>90F7<90F7>90`’

   Remember, the turtle starts facing East with the pen down at the origin. The origin (0,0) is at the center of the canvas, x increases to the right, and y increases upward.”

2. Write a function that takes a string and returns the index of the first non-numeric character. The string argument may be the empty string.
   Example: “123P456” would return 3.

3. Explain in words, not code, an algorithm or process that would interpret a ShapyTurtle command string. A high level explanation is all that is required.

4. Write a function to interpret and process a string containing any combination of the first three ShapyTurtle commands in Table 1. Assume the input is a string containing one or more instances of the first three ShapyTurtle drawing commands. Also assume that a function exists for each of the commands. For example, assume there is a `process_left(...)` function to interpret the `<` command. In your answer, make the appropriate function call to process the command that is found.
   There is no need to implement the command functions during problem-solving. Do not worry about error checking during problem-solving.

5. Write a function to properly interpret and process the GoTo command. How is GoTo different from other commands considered in problem-solving? What special, extra work is needed?
   Example: G100,50 ultimately would issue the turtle command `turtle.goto(100,50)`.

If finished early, notify the instructor or the SLIs.

## 3 post-PSS Implementation (10%)

For the post-PSS work on the following:

1. Write a function to interpret and process a sequence of different ShapyTurtle commands. Start with processing the first three commands and continue implementing the other commands as you progress.
2. Write a main function that prompts for the string containing ShapyTurtle commands and calls the function that interprets and processes the command string.
3. Write a function that takes a string and returns the index of the first non-numeric character. The function returns None if the string does not start with a number. The string argument may be the empty string.

To get credit, submit the work-in-progress `shapy_turtle.py` file to the post-PSS submission dropbox by the due date in MyCourses.

## 4 Implementation (75%)

Implement all ShapyTurtle commands in Table 1 for the complete program.

### 4.1 Details of Operation

The main function will prompt the user to enter a string containing ShapyTurtle commands. It will then call a ShapyTurtle interpret function passing that string in for processing. Assume that the user will enter exactly one string containing a sequence of ShapyTurtle commands. If there are no errors, the main function waits for the user to close the drawing window using `turtle.done()`.

The ShapyTurtle interpretation function only needs to check the first character of the input. It will send the input to the proper ShapyTurtle command function. If the character is not a valid ShapyTurtle command, print an error and terminate the function. It will not change the string other than moving beyond the command character. If a ShapyTurtle command returns None, which shall represent an error, the program will print the text it tried to interpret, terminate and not process the string any further.

Create working functions for processing each of the ShapyTurtle commands. When the program identifies a ShapyTurtle command, it should call a function to interpret that command substring. If it processes that command without error, then the program should advance to the next substring to interpret the next command.

**It is acceptable to use iteration, slicing, and indexing of strings.**

### 4.2 Error Handling

Each ShapyTurtle command function should handle errors. If an error occurs, your function should return None. For example, `process_c( "CC100" )` should return None because the first *C* in the string is missing its radius value.

The numbers accepted for a ShapyTurtle program are all non-negative, integral values. Input of a negative number or of a floating point number is an error.

### 4.3 Constraints and Restrictions

The use of the Python re module, or any similar library, is prohibited.

### 4.4 Tools and Tips

This section outlines some tools and tips for this assignment.

1. This assignment needs to check if a character is a numeric digit. Python has a function that checks if a string contains numeric digits. If `st` is a reference to a string, the call `st.isdigit()` will return `True` if `st` contains only numeric digits, and False otherwise.

### 4.5 Grading

One long function that does everything will result in a deduction of 50% of the implementation points.

The 75% implementation portion of the assignment will be graded as follows:

- 10%: The main function prompts the user for a string and calls an interpret function to process it as a string of ShapyTurtle commands.
- 25%: An interpreter function interprets the string argument, and calls the proper ShapyTurtle command function. Errors are handled properly, such as invalid command characters.
- 40%: A function exists to process each of the ShapyTurtle commands, and each function handles errors properly.

- **-10%** Style and Documentation:
  Points for style and documentation are now **minus points**, which means they are taken off the top of the grade. Even a perfectly working implementation will be subject to grade deductions of up to 10% if that program lacks proper style, layout and documentation. This includes other problems, including not having followed submission instructions.
  - 2 points for incomplete or missing the file docstring and content.
  - 4 points for incomplete or missing function docstrings.
  - 1 point for bad zip file name.
  - 1 point for incorrect python file name.
  - 2 points for other style and layout issues.

  Docstrings should be `"""` form and have a purpose statement, parameter descriptions, return type descriptions, and pre/post-conditions where appropriate.

### 4.6 Submission

ZIP `shapy_turtle.py` into a file named `lab04.zip` and submit to the *MyCourses assignments dropbox* by the due date for this assignment.