A Shape Collection
CS 5001 Intensive Foundations of Computer Science

1. Goals

- Use functions in a useful way to reduce code
- Use functions to enhance work you've already been doing

2. In Recitation:

We'll spend some more time this week learning Python syntax, learning how to repeatedly execute a block of code, and creating functions with more complex parameter sets.

In addition, we'll focus on overall code organization. How you write and organize your code makes a big difference in how easy it is to write, modify, debug, and re-use. For example, we will be grouping functions that do similar things in the same file, we will have only one copy of each function, and we will be adding comments to our code. Good code organization saves you time and effort.

Objective 1 : Set up your workspace

Create a new working directory for your project. Note that spaces in filenames can be annoying when using a terminal, so you may want to use something like project3, project03, or 03project. Note that the last of those options means you can type 03 then hit tab to complete the directory name using tab-completion.

Use the cd command on the terminal to set your working directory to your project 2 directory. Then create a new file (e.g. lab3.py) and save it in your working directory.

Objective 2 : Create comments

Starting with this project, comments will be an important and essential part of your code. There are three types of comments you should be using regularly.

1. At the top of each code file, include a header: your name, a date, the class, and an overall purpose for the file.
2. Each function you write should have a comment or docstring associated with it that indicates the purpose of the function and, ideally, the purpose and legal values for each parameter. A function comment generally goes just above the def keyword and uses the standard # symbol. A docstring goes on the line after the def keyword and is identified by triple quotes ''' or """. A triple quoted string can span multiple lines.
3. If you have a large function (e.g. more than 20 lines of code) it can be helpful to put comments inside your function that indicate the purpose of important blocks of code.

```
#
# Fall 2020, CS 5001
# Lab 2 code file
# Practicing good code style and more complex
functions

# this is a regular function comment
# this function returns the result of adding a and b
using the + operator
# a and b can be both numbers or both strings
def someFunction(a, b):
    return a + b

def anotherFunction(a, b):
    '''this is an example of a docstring, note the
triple quotes to start and end
this function returns the result of using the *
operator on the a and b parameters
a and b can be both numbers or an int and a str'''
    return a*b

# main code goes here
```

Putting comments in your code is something you should do for yourself as much as for other people. Comments do three things for you.

1. First, they help you to understand what each function or large code block is doing. It lets you think about the code in terms of what you want it to do rather than the specifics of syntax. Experienced coders often write out the comments for their code before writing any actual code. Identifying mismatches between the comments and the code helps you identify possible bugs in your code.
2. Second, they help you remember the purpose of each function and each function parameter so you use them correctly later on. This helps you avoid making errors.
3. Third, they make it possible for other people to read and understand your code. Remember, CS is a collaborative discipline, so you will often be working on code bases in parallel with others who may need to edit or use your code.

Objective 3: Create the appropriate program structure

Up until now we've just been putting code in the file and letting it run. This isn't how we should be organizing our code. Typically we'll have a set of functions ending with a main function. The main function represents the entry point into our program. After your initial comment block, create a main function to show where the program starts.

```
def main():
    command
    command
main()
```

Objective 4: Import the turtle package

From now on we'll be using the import turtle method of importing packages. This creates the name turtle in the global symbol table, and all of the turtle functions are contained in a separate symbol table which we can access by using the prefix turtle. before calling the turtle functions.

Objective 5: Write a goto function

Make a function that sends the turtle to a particular location without drawing anything. Name the function goto and give it two parameters to hold the new x location and y location. Add an appropriate comment or docstring. Include default values of 0,0 so calling it with no arguments returns it to the origin.

Once you have written the function, test it by using the goto function and drawing some lines in different locations. Finish your code file by calling turtle.mainloop().

Please start to follow a standard structure for your code. From top to bottom your code file should have the header, import statements, functions, and then your main function.

Do all your testing in the main function.

Objective 6: Write a block function

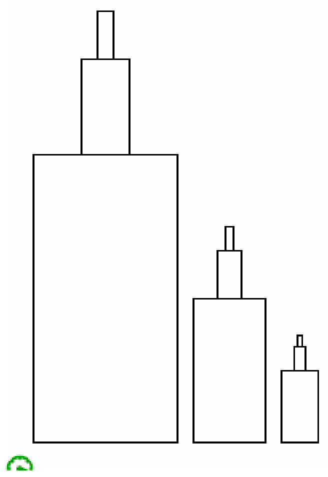
Write a function that has parameters for the location (x, y) and the size (width, height) of a rectangle. The function should draw a block of the specified size with the lower left corner of the block at position (x, y). The steps needed are to first call your goto function to position the turtle, then use forward and left turn instructions to draw the block.

Delete or comment out the goto test code in main and instead make a couple of calls to the block function to check that it is working properly. Make sure you vary all of the parameters when writing test code.

Objective 7: Write a function to create a tower out of blocks

Plan out a city tower created from three blocks. Use the geometric thinking process we'll go through together to develop your code.

Delete your old test code and make several calls to your tower function with different parameters to test your code.



Try adding some randomness

If you have time and feel like doing some more exploration, import the random module and try using it and a loop to generate multiple towers in random locations and with randomized sizes.

Add the ability to create a filled or non filled block by adding a boolean variable to your drawblock function

Submit this final code under ICE recitation work

3. Lab Assignment instructions

The goal of this assignment is to give you more practice in creating and using functions and function parameters in Python. You'll continue to create shapes using turtle graphics.

The end result of the assignment should be a representation of a scene depicting something that you associate with Maine. The scene can be as simple or complex as you like. What is important is to properly structure your code and include good comments.

Remember that you need to write the code yourself. You can discuss the assignment and ideas regarding the assignment with others in natural language, but not Python. If you have questions about your Python code (other than very simple syntax questions), please ask one of the TAs or the professor. Please see the syllabus for guidelines on collaboration and attribution.

For this project, you will be submitting your report report03.pdf, your shape library shapelib.py file, a file main.py with your shape test scene, and a file scene2.py that creates a second scene. If you create additional scenes or separate files for your extensions, please call them extension1.py, extension2.py, and so on.

1. Set up your shape library code

    We are going to start by creating a set of functions that we can reuse in a larger program. Some of this will come from previous assignments or from the recitation session. Make sure you do a good job on this file as it will be used over and over.

    From recitation you should have the following functions:

    ```
    goto(x,y)

    block(x,y,length,width)

    main()
    ```

    Go ahead and add one new one

    ```
    draw_circle(x,y,radius)
    ```

This function accepts an x y value and a radius. Uses your previously created goto function to move to a specific location and then draws a circle at that location with the sent radius.

Test this function from your main function, and then comment out the testing code.

Now we are going to turn this into a library file. Simply remove the main method or comment it out and make sure the only statements at top level are functions.

2. Setup your driver file

Now create a main.py file that imports your shapelib.py functions. This is often called a driver file. It is the starting point for your code execution.

To do this, create a file called main.py in your text editor, and at the top (after your name, date, and description) import your shapelib file.

```
import shapelib
```

The import statement lets you call any functions in your shapelib.py file, such as block, using the syntax shapelib.block.

If you don't feel like writing shapelib. whenever you use it, you can rename shapelib as something more concise like slib, if you wish in the import statement. Use the following syntax to choose a different name for the module you are importing.

```
import shapelib as slib
```

Import the turtle package as well. If you feel like it, you can rename the turtle package to something simpler.

3. Test your main code

In your main.py file, make a function called scene1. Put a single call to the scene1 function at the end of the file, followed by turtle.mainloop()

Write some python code in your scene1 function that calls one or more functions from your shapelib.py file to draw some shapes. Note that to call a function from your shapelib file you have to prepend shapelib. (or slib if you used a different name) in front of the function name. For example, shapelib.block(10, 10, 20,30)would be how you would call the block function.

Run your main.py file and see if it successfully draws shapes from your shapelib.py file. You don't need to save this image, as you'll be replacing the code in the scene1 function later on.

4. Write functions to draw basic shapes

In your shapelib.py file, make at least 2 more functions for drawing basic shapes like the block function from the lab. Basic shapes should take in at least an x location, y location, and size information. For example, you could make a triangle function or a hexagon function with the size value being the length of a side, in pixels.

Include the ability to have a shape filled or not filled depending on a sent boolean value. For example, `shaplib.block(10,-10, 10,10,True)` would create a solid square and `shaplib.block(10,-10, 10,10,False)` would create an empty square.

Each of your shapes should draw properly no matter where it is drawn on the screen, what the size parameter is, or what the orientation of the turtle happens to be when the shape function executes. (There are ways to check the turtle orientation or just reset it back to what you expect.)Test this out for all of your functions.

Demonstrating that your code works properly is an important part of writing software that people will use. Encapsulate your test code in a function like test1.

5. Write functions that make more complex things

In the shapelib.py file, make 2 functions that draw objects/things that relate to your scene. You can pick small objects, or big things. The objects should be built from several of the simpler shapes (triangles, blocks) from your library. For example, you could have a chair, tree, or house function.

When you define these aggregate shape functions, they should all take at least three parameters: x, y, and scale. The first two (x, y) will define the starting location for the shape and the third should define the scale of the shape. A scale factor of 1.0 should draw the shape in its smallest possible size.

Feel free to add additional parameters as you like to control other aspects of the shape. As with the basic shapes, your shapes should draw properly at any scale, location, or turtle orientation.

For a quick refresher, check out thinking geometrically.

In both of your new functions, use one or more of the basic shape functions to draw the shape. For example, a house shape might incorporate the block and triangle functions. Test out your new functions before proceeding. Encapsulate this code in a function like test2.

6. Write a function to draw a scene

Using the simple and aggregate shapes you created in the above steps, create a scene of something you associate with Maine. This top level code should all be inside your scene1 function.

In your scene, make use of the fact that your functions can draw the shapes with different sizes in different locations. You can also use the random package to create more creative shapes in random locations. If you want to look ahead, you can use for loops to reduce your code.

Take a screenshot to put in your report.

7. Write a function to draw a second scene

Write a function scene2 that makes a second scene. Be creative!

You can write this function in a second file like scene2.py or include it in your main.py file. In either case, a single call to the function should draw the entire scene. When your program starts, describe both scenes and ask the user which scene they want rendered.

Take a screenshot to put in your report.

Note : Screen Captures

On Windows, use the key combination alt-PrtScn or alt-fn-PrtScn to capture a single window. Your system should save the image to Pictures/Screenshots.

On a Mac, use cmd-shift-4 to get into screen capture mode. Then you can either click and drag to select part of your screen, or you can tap the spacebar and then click to capture a single window. The Mac saves screenshots to the Desktop.

4. **Extensions**

Projects are your opportunity to learn by doing. They are also an opportunity to explore and try out new things. Rather than list out every activity for a project, the defined part of each project will be about 85% of the assignment. You are free to stop there and hand in your work. If you do all of the required tasks and do them well, you will earn a B/B+.

The following are a few suggestions on things you can do as extensions to this assignment. You are free to choose other extensions, and we are happy to discuss ideas with you. Be sure to save and include pictures of your extensions in your report.

- Create some additional functions that draw different shapes.
- Add additional parameters to your functions, possibly even optional ones.
- Write a function that will draw an N-gon. It should have two parameters: the distance of a side, and the number of sides.
- Build a taller hierarchy of functions that builds shapes from your aggregate shapes.
- Add some randomness in color or in the objects you create.
- Creative or complex scenes above and beyond the basic expectations count as extensions.

5. Report

Each week you will need to submit a report along with your code. Your report is primarily intended to demonstrate that you have completed all the tasks and to show your images. There are six sections that need to be included in your report. The report is worth 4 points of your project grade. You may use a word processor of your choice, but please submit your report as a pdf with the name **report03.pdf** along with your code.

1. **Problem Description (Optional)**
   Write a brief summary of the project in your own words. It should be no more than 4-5 sentences at most. Each project will have both a CS purpose (e.g. using parameters and functions) and an application (e.g. making images of a particular type).
2. **Screenshots**
   Include all of your required images in your report. For each image, include a few sentences describing something you find interesting. This could be something about your code or something about the image itself.
3. **Extensions**
   Describe any extensions you did for this project so that you get credit for them. Include images or outputs that demonstrate your extension works.
4. **Reflection**
   Write a brief description (1-3 sentences) of what you learned. Think about the answer to this question in terms of the stated purpose of the project. What are some specific things you had to learn or discover in order to complete the project?
5. **Acknowledgements**
   Provide a list of people you worked with, including TAs, professors, and other students in the course. List also any resources that you used. This includes textbooks, lecture notes, python documentation, library documentation, websites, or forums.
6. **Grading Statement** (optional) Using the rubric, what grade do you think you deserve and why?

6. Submission

You should be submitting at the least:

- Report03.pdf
- driver.py
- shapeLib.py

Submit your project code on canvas as Lab3_”Your_name”.zip

When you submit, double-check the following.

- [ ] Is your name and an appropriate header at the top of each Python file?
- [ ] Does every function have a comment or docstring specifying what it does?
- [ ] Does your report have all six sections completed?
- [ ] Is your report a pdf document?

7. Rubric

	Possible	Given
Shape Library		
goto, block, main (from recitation)	1	0
draw_circle	1	0
basic shape function 1	1	0
basic shape function 2	1	0
object function 1	2	0
object function 2	2	0
functions have fill options	1	0
Driver		
scene1 function	3	0
scene 2 function	3	0
tests created for each function	3	0
Misc		
Report (all or nothing)	4	0
Code Quality (correct indentation, comment blocks, variable naming, etc)	4	0
Not included in total possible:		
Shape library not used as a separate file	-15	0
Extensions (Not calculated without report)	4	0
Creative or went above and beyond	4	0
Code does not compile	-30	0
Late penalty	-6	0
Not implemented as requested	-30	0
TOTAL POINTS POSSIBLE out of 30	26	0