**Project 2: Turtle Drawing**

*In the beginning of the semester we drew interesting shapes on code.org. In this project, we will use the turtle module (*[*https://docs.python.org/3.4/library/turtle.html*](https://docs.python.org/3.4/library/turtle.html)*) to draw shapes both iteratively and recursively. In this project, we will use loops, functions, conditions, and variables to draw pictures with the python turtle package.* ***You may work with a partner****; make sure both students are clearly identified in the comments of each program.*

*An important skill as a computer scientist is the ability to learn how to use an existing API library by reading documentation, finding good API usage examples on the internet, and adapting those examples to meet your needs. In this project, you will begin building these skills to help you make the transition from the classroom to practice.*

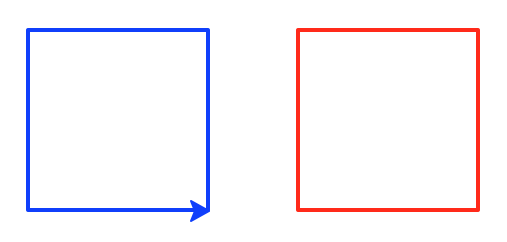
**Suggested Steps**

Create a folder project2-uLogin1\_uLogin2, replacing the uLogins with the e-mail ids of you and your partner.

**Learn how to work with turtles**

1. Create a python program where you can play with turtles. Draw a line on the screen using the python turtle package. I learned how to make turtle programs by using these examples: <http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html>  
     
   A basic turtle program looks like the following:  
     
   import turtle  
   turtle.shape("turtle") # *optional*turtle.speed(0) # *optional*  
     
   turtle.forward(90)  
   turtle.left(90)  
   turtle.forward(90)  
     
   turtle.Screen().exitonclick()

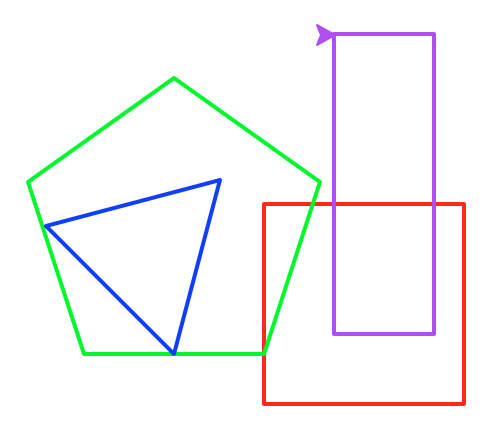
The first & last lines should be in ***every*** turtle program you write.   
***DO NOT NAME YOUR PROGRAM turtle.py!***

1. Draw a square on the screen. Change its color. Make the line thicker by changing the pensize or width. Change the speed the turtle draws.
2. It’s possible to move the turtle without drawing. Draw two squares on the screen that don’t touch. For example:  
   

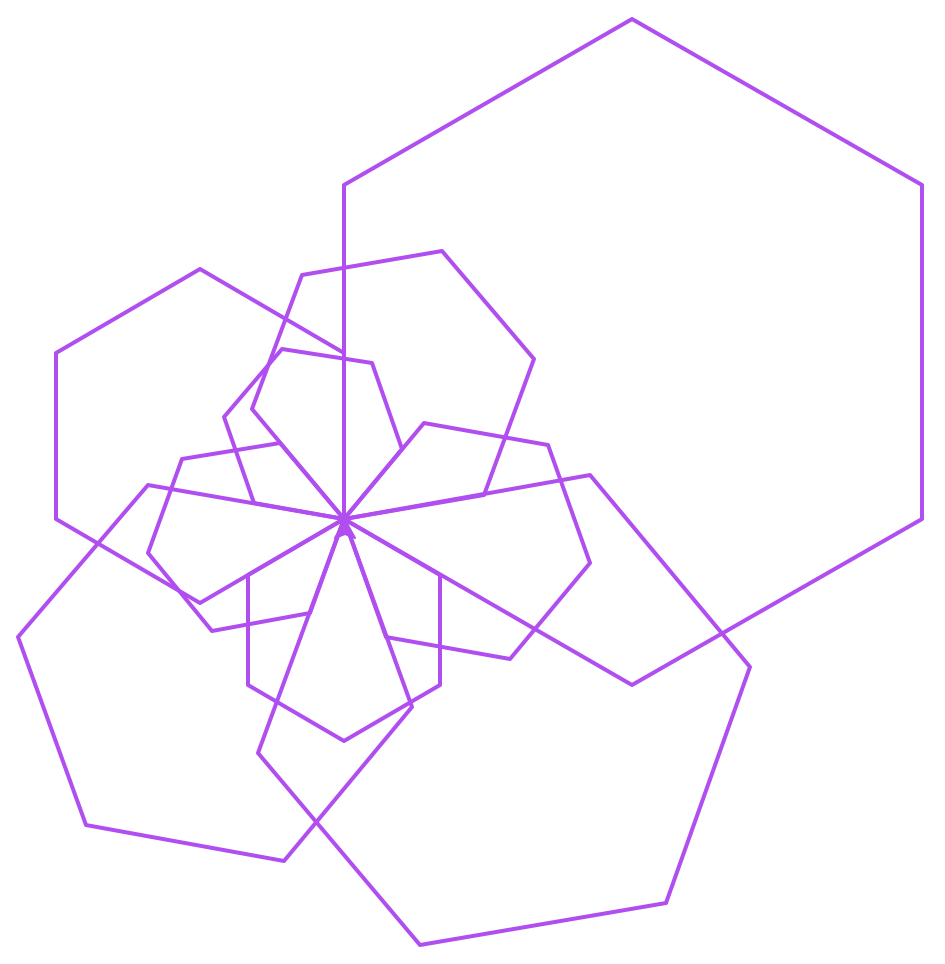
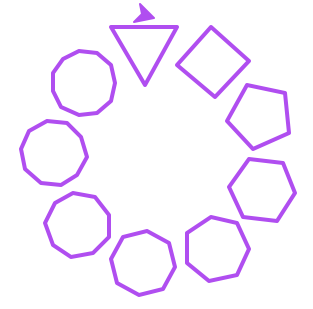
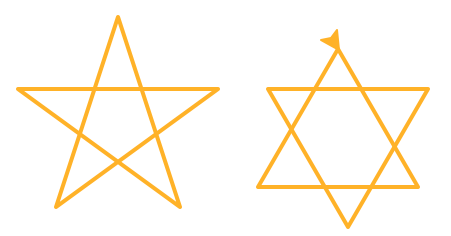
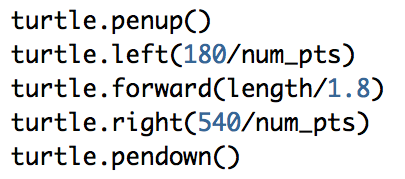
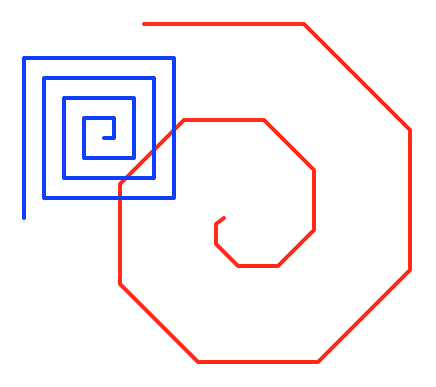
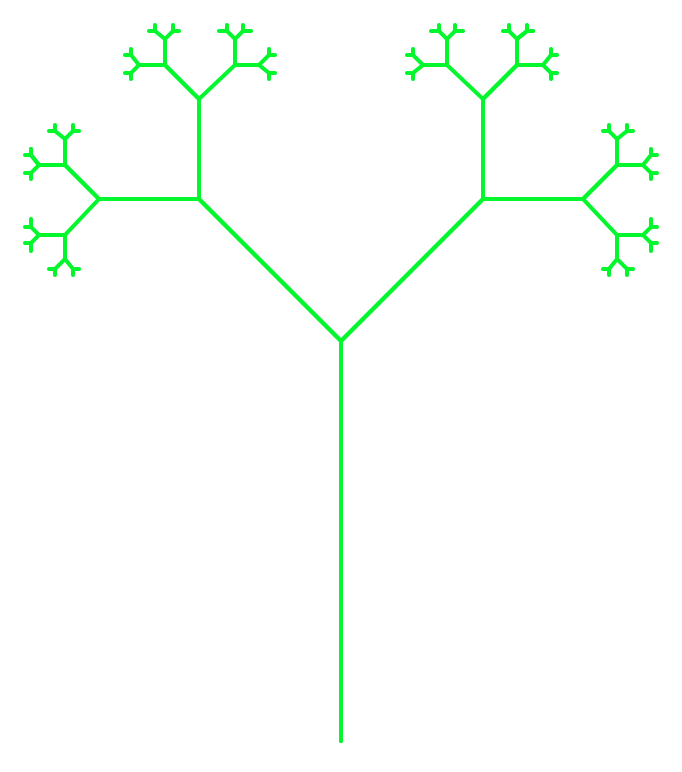
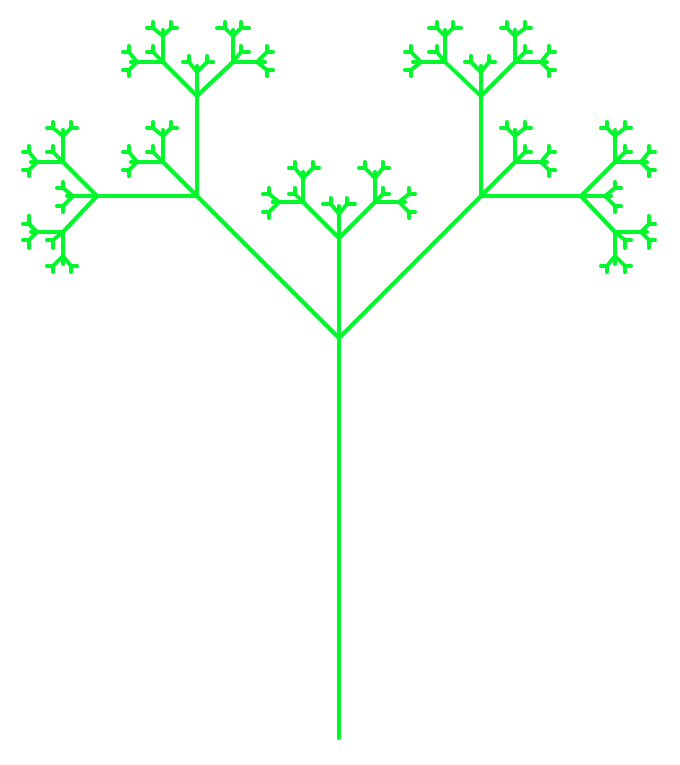
*Now you’re ready to tackle the assignment. To speed up testing, set the turtle speed to be 0. None of the following turtle programs should take user input!*

**Implement the project**

1. Create a program **basic\_shapes.py** that implements the following functions. This link may be helpful: <http://www.mathsisfun.com/geometry/interior-angles-polygons.html>
   1. **rectangle(width, height)**: takes the length of the width and height as parameters.
   2. **square(side)**: takes the length of a side as a parameter. Implement this by calling the rectangle function.
   3. **triangle(side**): takes the length of a side as a parameter and draws an equilateral triangle.
   4. **hexagon(side)**: takes the length of a side as a parameter and draws an hexagon.

Test that your functions work by clearly drawing a square, rectangle, triangle, and hexagon on the screen. They shouldn’t overlap too much. For example:  
  
***Every submitted picture should look unique.***

Remember to declare all your functions at the ***top*** of the file, ***before*** you call them.

1. Next we’ll generalize our polygon drawing by writing a program **polygon.py**:
   1. **polygon(num\_sides, length)**: takes the number of sides (num\_sides) and the length of each side (length) as parameters. If the number of sides is less than 3, draw nothing.
   2. ***Hint***: To determine how far to turn your turtle, you will likely need to divide a multiple/factor of 360 (the number of degrees in a circle) by the number of sides.
   3. Test that your function works by using a loop to draw a number of shapes in an interesting pattern. Here are some examples:  
      
2. We can use a similar technique to draw stars with an arbitrary number of points. Write a program **polygram.py**:
   1. **polygram(num\_pts, length)**: takes the number of points (num\_pts) and the length of each line in the star (length) as parameters. You will need a different approach to drawing the stars depending on if there is an odd or even number of points. *Hint*: implement the odd case first. 
   2. ***Hint***: How far should your turtle turn for an odd-pointed star? Twice as much as you did for the polygon version.
   3. Even-sided stars can be drawn as two separate polygons with half the number of points. For example, a six-sided star can be drawn as two triangles. In between drawing these two polygons, you may find it useful to move your turtle:  
      
   4. Test that your function works by drawing at least an even- and odd-pointed star.
3. Let’s explore recursion by writing a program **spiral.py**:
   1. **spiral(length, angle)**: takes the length of the spiral (length) and the angle of the spiral (angle) as parameters. Implement using recursion. Think about how the parameters change at each recursive step. Should either parameter stay the same? What’s the base case?
   2. **spiral\_loop(length, angle)**: takes the length of the spiral (length) and the angle of the spiral (angle) as parameters. Implement using iteration. How should the loop change at each step/iteration?
   3. Test your functions by drawing two different spirals.
4. Create a program **tree.py** that recursively draws a tree using the following function:  
     
   def tree(size, angle):  
    if size > 2:  
    turtle.forward(size)  
    turtle.right(angle)  
    tree(size/2, angle) # call A  
    turtle.left(2\*angle)  
    tree(size/2, angle) # call B  
    turtle.right(angle)  
    turtle.back(size)  
     
   turtle.left(90)  
   tree(200, 45)  
     
   Modify the above function so the output changes from this:  
   to this: 
5. Our final recursive program does not make use of turtle. Instead, it will calculate the *nth* fibonacci number (see <http://www.mathsisfun.com/numbers/fibonacci-sequence.html>). A fibonacci number is calculated by adding the two numbers that come before in the sequence. Think about what the recursive step will look like. Assume the sequence starts at 0 (for n = 0). How many base cases are need? Write a program **fibonacci.py**:
   1. **fibonacci(n)**: takes the fibonacci sequence number as a parameter and returns the sum of the previous two terms in the sequence. This should be implemented recursively.
   2. Test that your function works by getting a number (n) as input from the user and printing the result of calling your fibonacci function.

**Grading**

You will be graded on the following:

* **Correctness:** Your code should perform as specified. Correctness will count for the largest portion of your grade.
* **Docstrings:** For each function that you design from scratch, write a good docstring (“””).
* **Internal comments:** Within functions, the more complicated parts of your code should also be described using "internal" comments.
* **Programming style:** Your variable names should be meaningful and your code as simple and clear as possible.
* **Formatting style:** Each line must be less than **80 characters** long *including spaces*. You should break up long lines using \.

**Submitting**

When complete, your project should include the following **six** python programs:

* basic\_shapes.py
* polygon.py
* polygram.py
* spiral.py
* tree.py
* fibonacci.py

Zip your project using the same steps as for the labs, giving the zip file the same name as your project1-uLogin1\_uLogin2 folder name, and submit.