Turtle Graphics

For this assignment, you'll use Python's turtle graphics library module to draw interesting designs (and practice with functions).

Part 1

Try this in the shell:

```
>>> from turtle import *
>>> t = Turtle()
>>> t.forward(100)
>>> t.forward(100)
```

Notice that the effect of the two calls to `forward` is different. The first draws a line beginning at the center of the window; the second draws a line beginning where the first left off. This illustrates a fundamental difference between the graphics programs you wrote for chapter 4 and a turtle graphics program. The turtle (represented by a triangle in the window) is an object whose *state*, dependent on past commands, determines what future commands will do. The turtle "remembers" where it is and what direction it's pointing; it does whatever comes next relative to its current position and direction.

Using the commands `reset`, `forward` and `right`, you can clear the window and draw a square.

```
>>> t.reset()
>>> t.forward(100)
>>> t.right(90)
>>> t.forward(100)
>>> t.right(90)
>>> t.forward(100)
>>> t.right(90)
>>> t.forward(100)
>>> t.right(90)
```

If you don't like typing so much, use a loop:

```
>>> t.reset()
>>> for i in range(4):
...     t.forward(100)
...     t.right(90)
```

What if you want to make lots of squares? Define a function. (Switch from the shell to the editor at this point.)

```
# turtle_functions.py
# David Owen

from turtle import *

def square(turtle):

    for i in range(4):
        turtle.forward(100)
        turtle.right(90)

def main():
    t = Turtle()
    t.reset()
    square(t)

if __name__ == "__main__":
    main()
```

Notice that it's `main`'s turtle—not a copy—that is affected by the code in `square`. `Turtle()` is a call to a *constructor*; it creates a new Turtle object. So `t` in `main` and `turtle` in `square` are two variables referencing the same object.

Here's another version of `square`. This one allows you to specify the length of a side.

```
def square(turtle, sideLength):

    for i in range(4):
        turtle.forward(sideLength)
        turtle.right(90)
```

Based on this example, write a function called `triangle` that draws an equilateral triangle, allowing the size to be specified when the function is called.

```
def triangle(turtle, sideLength):

    # Fill in code here to make the turtle
    # draw an equilateral triangle...
```

Once you get `triangle` working, write a function that draws a regular polygon:

```
def polygon(turtle, numSides, sideLength):

    # Fill in code here to make the turtle
    # draw a regular polygon...
```

Part 2

If you look up turtle graphics (or Logo, the programming language first used for turtle graphics) online, you will probably see a star that looks something like this:

(star.png goes here.)

It's a bunch of squares...draw one, turn, draw another, turn...until the turtle's gone all the way around the circle. As explained above, you can use a loop to draw a square. You can use a "nested" loop, that is, a loop inside another loop, to draw a star made out of squares:

```
for i in range(18):

    for j in range(4):
        t.forward(100)
        t.right(90)

    t.right(20)
```

Better yet, you can use your `square` function inside a loop:

```
for i in range(18):
    square(t, 100)
    t.right(20)
```

Based on these examples, write a function called `polygonStar`:

```python
def polygonStar(turtle, numPolygons, numSides, sideLength):

    # Draw a Logo-esque star design made out
    # of polygons...
```

Part 3

In addition to position and direction, you can change a turtle's state so that it moves without drawing or draws different colored lines. Read through the following code (don't run it yet!) and try to predict what it will do. For each line, think about whether that line might change the turtle's state, tell it to do something, or both.

```python
for i in range(10):
    t.penup()
    t.forward(100)
    t.right(90)
    t.pendown()
    t.pencolor("red")
    t.forward(100)
    t.right(90)
    t.pencolor("blue")
    t.forward(50)
    t.penup()
    t.forward(50)
    t.right(90)
    t.forward(100)
    t.right(90)
    t.right(36)
```

Using a combination of pen commands, drawing commands and turning commands, create your own interesting image(s) to share with the class. (Feel free also to experiment with any other commands you read about on the documentation page for Python's turtle module.)

Be ready to show off one or two of your best images at the beginning of the next class (for participation points).