|  |  |
| --- | --- |
| **Lab 7: Graphics** | Name: Lex Baker  Date: 10/4/22 |

1. **Turtle Graphics** is a graphics package that comes with Python. It is similar to Blockly and some students may have used similar packages in the past.   
   1. Type the following statements into the python shell and write below what each command does.

>>> from turtle import \* **imports turtle like the functions are defined within the code**

>>> window = Screen(); **creates a screen and names it windows**

>>> window.setup(400,200); **initializes the screen to 400x200 pixels**

>>> pet = Turtle( );  **creates a Turtle object named pet**

>>> pet.shape(“turtle”); **changes the shape of the Turtle object**

>>> pet.forward(100); **moves the turtle object forward 100 px**

>>> pet.right(90); **rotates turtle object right by 90 degrees**

>>> pet.backward(50); **moves turtle object backwards 50 px**

Notice that both your turtle and the window are objects and setup, forward, shape, right, and backward are all methods these objects call. Here is a list of all the methods you can call on your turtles:<https://docs.python.org/3.3/library/turtle.html>.

* 1. Now open up your editor and add the appropriate header. Here are a few new commands to try out. Write the following code in your file and figure out what each command does.

from turtle import \*

canvas = Screen(); # create screen object named canvas

canvas.setup(500,500); # set screen size to 500x500 pixels

t = Turtle(); **# creates turtle object set to t**

t.color("green"); **# changing drawing color to green**

t.circle(100); **# draws a circle with radius 100**

t.penup(); **# stops drawing**

t.goto(0,-50); **# moves the turtle to the middle of the screen horizontally, and 50 pixels below the middle vertically.**

t.pendown(); **# allows for drawing again**

t.write("Circle"); **# writes the word “circle”**

canvas.exitonclick(); **# waits to end screen until user clicks**

Alter the code so that your turtle draws the circle in black and then prints the word circle in red. Now make the turtle draw the circle in the lower right part of the screen instead of the upper middle.

**from turtle import \***

**canvas = Screen(); # create screen object named canvas**

**canvas.setup(500,500); # set screen size to 500x500 pixels**

**t = Turtle();**

**t.color("black");**

**t.penup()**

**t.right(90)**

**t.forward(100)**

**t.pendown()**

**t.circle(100);**

**t.penup();**

**t.goto(0,-50);**

**t.pendown();**

**t.color("red")**

**t.write("Circle");**

**canvas.exitonclick();**

* 1. You can use these turtles along with regular python commands (if, while, etc.) to create polygons as well. Write a turtle program to draw a regular, n-sided polygon. The program should let the user choose how many sides and the length of each side. Getting user input must be done in a robust way. Use a while loop to instruct the turtle to draw the polygon.



**length = int(input("Length of the sides: "))**

**sides = int(input("Number of sides: "))**

**degPerAngle = 180 - (((sides - 2) \* 180) / sides)**

**print(degPerAngle)**

**canvas = Screen()**

**canvas.setup(500,500)**

**t = Turtle()**

**i = 0**

**while i < sides:**

**t.forward(length)**

**t.left(degPerAngle)**

**i += 1**

**canvas.exitonclick()**

1. **Shapely** is a python package dealing with geometric objects. It treats shapes as objects instead of just viewing the path of the turtle object, as is done in turtle graphics. You may need to install the package before you use it. HINT: Use the Tools -> Manage Packages menu options on the Thonny toolbar to find and install the Shapely package.
   1. Shapely has built in objects that represent points, lines, and polygons. Type the following into the command line of Thonny and see what happens.

>>> from shapely.geometry import Polygon, Point, LineString;

>>> p1 = Point(0,1);

>>> p2 = Point(1,2);

>>> p1.distance(p2);

>>> x1 = LineString([(0,0), (1,1)]);

>>> x2 = LineString([(1,0),(0,1)]);

>>> x1.intersection(x2);

What do the methods distance and intersection do?

**Distance calculates the distance between two Point objects**

**Intersection calculates at what point two LineStrings intersect**

* 1. To create triangles and other polygons, you list the starting vertex followed by each of the other vertices in order. Notice that the default for a polygon object is to automatically close itself. Type the following lines of code at the command line.

>>> triangle = Polygon([(0,0),(1,0),(0,1)]);

>>> triangle.area;

>>> triangle.length;

>>> c = triangle.centroid;

>>> c.coords[:];

>>> c.x;

>>> c.y;

What type is c? What do c.x, c.y, and c.coords[:] do?

**C is of type Point(). C.coords[:] gives x and y coordinates of c together, and c.x and c.y give the same points, individually and respectively.**

* 1. Open up a new file in the editor of Thonny, create the points p1 and p2 from above, and use the plot function you learned in Lab 4 to plot these points. You will need to write import matplotlib.pyplot as plt at the top of your file in addition to the import shapely statement from part a) of this problem. Hints: use the x and y functions you discovered in part b) of this problem and add 'o' after the x- and y-coordinates of the point as a third argument to the plot function to print a blue dot (just as with the other plots you've done, you could replace this with a different character to see a different shape and color).

**File in submission**

* 1. Now find the file shapely\_graphics.py on Canvas and save it in your working directory. After the import statements are 3 function definitions taken from the website listed there. These functions will make it easier to plot shapes. Look at the sample code that uses the functions and try to understand how it works. (Don't worry about the syntax of the functions themselves. They will make more sense once you learn about lists.)
  2. **2-D Tranformations.** Run each of the following commands at the command line. The coordinates of the vertices of a triangle are printed out by a call to exterior.coords. What is happening in each line of code?

>>> triangle = Polygon([(0,0),(1,0),(0,1)]); **# creating triangle object assigned to variable triangle**

>>> triangle\_a = scale(triangle, xfact=2, yfact=2); **# creates triangle\_a, which is scaled by a factor of 2 from triangle**

>>> triangle\_a.exterior.coords[:]; **# returns coordinates of points on the triangle**

>>> triangle\_b = translate(triangle,3,4); **# creates triangle that is translated +3 on x-axis and +4 on y-axis from original triangle**

>>> triangle\_b.exterior.coords[:]; **# returns coordinates of points on the triangle**

>>> triangle\_c = rotate(triangle,45,(0,0), False); **# rotates original triangle by 45 degrees**

>>> triangle\_c.exterior.coords[:]; **# returns coordinates of points on the triangle**

Once you understand what is going on, go back to the editor and plot all of these triangles on the same plot (use the function plot\_polygon from part d). Add text to explain which triangle comes from which transformation.

1. **Geometry.** Create a program that will prompt the user to enter in two lengths and an angle that represent two sides of a triangle and the included angle. The program will take this information and do the following:
   1. Find and report (regular text output) the length of the third side, the measures of the other two angles and the area of the triangle.

* Use units of pixels as these are the default units for Turtle commands. Tell the user to keep their side lengths in the range of 10 to 400 pixels and their angles in the range of 1 to 179 degrees and enforce this.
* Be careful of degrees vs. radians. Python uses radians but turtles and the users use degrees.
* Use the Law of Cosines and the formula for the area of a triangle
  1. Use Turtle Graphics to draw this triangle. The triangle should be centered, that is the centroid should be at (0,0).

**Answered in python file**

* 1. Have the turtle write the coordinates of each vertex near each vertex. Also display the message “Here is your triangle!” right beneath the triangle. Hint – the turtle can tell you it’s current location. See the documentation for turtles to figure out how to ask the turtle for its coordinates.

**Answered in python file**

* 1. Extension: Draw the circumcircle and the incircle of this triangle.

1. **Geometry with Shapely.** Repeat parts b, c, (and d) of problem 3 with Shapely.  
   **Answered in python file**
2. **Tessellations**. Do each of the following exercises in both turtle graphics and Shapely. You should be able to use the exact same code for the calculations; the only difference will be drawing the graphics. Don't forget to use the transformations when plotting with Shapely.
   1. Create a program that tessellates a square around a single point. Do the same with an equilateral triangle.

**Answered in python file**

* 1. Adjust your program so it will ask for user input on the number of sides of a regular polygon. Use an if statement to check whether you can tessellate that shape or not. If so, do so. If not, tell the user what the problem is.

**Answered in python file**