

CS313E: Elements of Software Design

Turtle Graphics

Dr. Bill Young
 Department of Computer Sciences
 University of Texas at Austin

Last updated: October 13, 2011 at 17:05

Python is an *object-oriented* language. That implies a certain approach to thinking about problems.

Conceptualize any problem in terms of the relevant “objects”—data structures consisting of data fields and methods together with their interactions.

Programming techniques may include features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance.

Object Orientation

The basic idea of object oriented programming (OOP) and also of abstract data types is to view the world as a collection of objects.

Each object has:

- some *data* that it maintains characterizing it's current state;
- a set of actions (*methods*) that it can perform.

The programmer interacts with these objects by invoking their methods, which may:

- update the state of the object,
- query the object about its current state,
- compute some function of the state and externally provided values,
- some combination of these.

Turtle Graphics

Turtle graphics was first developed as part of the children's programming language Logo in the late 1960's. It exemplifies OOP extremely well.

There are various versions of Turtle Graphics defined for Python. The version we're going to use is defined in Chapter 7 of your textbook. *It is not the version from Lambert's earlier book.*

Objects are turtles that move about on a screen (window). The turtle's tail can be up or down. When it is down, the turtle draws on the screen as it moves.

A turtle has a given position designated by its (x, y) coordinates, with $(0, 0)$ being in the middle of the window. The state of the turtle consists of:

- Position:** denoted by its current x and y coordinates
- Heading:** denoted by an angle in degrees. East is 0 degrees; north is 90 degrees; west is 180 degrees; south is 270 degrees.
- Color:** initially blue, the color can be set to 16 million colors
- Width:** the width of the line drawn as the turtle moves (initially 2 pixels)
- Down:** an attribute indicating that the turtle's tail is down.

Some of the methods are listing on p. 250 of your textbook.

- `t = Turtle()` create a new Turtle object and open its window
- `t.home()` move the turtle to $(0, 0)$, pointing east
- `t.down()` lower the tail
- `t.up()` raise the tail
- `t.setheading(d)` change heading to direction d
- `t.left(d)` turn left d degrees
- `t.right(d)` turn right d degrees

- `t.forward(n)` move in the current direction n pixels
- `t.goto(x, y)` move to coordinates (x, y)
- `t.position()` return the current position
- `t.heading()` return the current direction
- `t.isdown()` return True if the pen is down
- `t.pencolor(r, g, b)` change the color to the specified RGB value
- `t.width(width)` change the linewidth to $width$ pixels
- `t.width()` return the width of t 's window in pixels

Because the window goes away immediately after the program terminates, it may be hard to see the result unless you delay things. Use the `time` module to do that.

The following will cause the program to pause for 10 seconds. You can set the time to any amount you like.

```
import time
time.sleep(10) # wait 10 seconds
```

```

from turtle import *
import math
import time

def drawSquare(turtle, x, y, length=100):
    """Draws a square with the given turtle, an
    upper-left corner point (x, y), and a side's length"""
    turtle.up()
    turtle.goto(x, y)
    turtle.setheading(270)
    turtle.down()
    for count in range(4):
        turtle.forward(length)
        turtle.right(90)

ttl = Turtle()
ttl.pencolor('red')
drawSquare(ttl, -50, -50, 100)
ttl.pencolor('blue')
drawSquare(ttl, 50, 50, 100)
time.sleep(20)

```

```

import math
def drawCircle(turtle, x, y, radius=100):
    """Draw a circle centered on (x, y), with
    the specified radius."""
    turtle.up()
    turtle.goto(x + radius, y)
    turtle.setheading(90)
    arc = (2 * math.pi * radius) / 360
    turtle.down()
    for i in range(360):
        turtle.forward(arc)
        turtle.left(1)
    turtle.up()
    turtle.goto(x, y)

```

```

def drawDonut(turtle, x, y):
    """Draw 36 circles in a donut shape. Each
    circle has radius 45. Figure is centered
    on (x, y)."""
    turtle.up()
    turtle.setheading(0)
    direction = turtle.heading()
    turtle.goto(x, y)
    for i in range(36):
        turtle.setheading(direction)
        turtle.forward(55)
        x1, y1 = turtle.position()
        turtle.down()
        drawCircle(turtle, x1, y1, 45)
        turtle.up()
        turtle.goto(x, y)
        direction += 10

```

What's on this slide may not quite work on all versions of turtle graphics.

Colors are in the RGB system, using a triple: (R, G, B) . Each element in the triple is an intensity from 0 to 255, indicating the contribution of R (red), G (green), and B (blue). For example:

black	(0,0,0)
red	(255,0, 0)
green	(0, 255, 0)
blue	(0, 0, 255)
gray	(127, 127, 127)
white	(255, 255, 255)
burnt orange	(255, 125, 25)

This is a nice website that allows you to find the RGB values for various colors: www.colorschemer.com/online.html.

Turtles have two “colormodes” and you’ll get an error if you try to do some things in the wrong mode. The modes are 1 and 255. In mode 255, use triples of range $0 \leq c \leq 255$. In mode 1, use percentages (range $0 \dots 1$).

```
>>> t = Turtle()
>>> t.pencolor(127, 127, 127)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    ....
    raise TurtleGraphicsError("bad color sequence: %s" % str(color))
turtle.TurtleGraphicsError: bad color sequence: (127, 127, 127)

>>> t.pencolor(0.5, 0.5, 0.5)
>>> t.screen.colormode(255)
>>> print (t.screen.colormode())
255
>>> t.pencolor(127, 127, 127)
>>> t.screen.colormode(1)
```

Write Turtle graphics functions that will do the following:

- ❶ draw a cube;
- ❷ draw a regular polygon with k sides and radius r (distance from center to one of the vertices);
- ❸ draw m concentric circles;
- ❹ draw the UT Tower.