CS303E: Elements of Computers and Programming

Turtle Graphics

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

Last updated: April 21, 2025 at 13:38

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. *You will be using classes already defined for you.*

Turtle Graphics

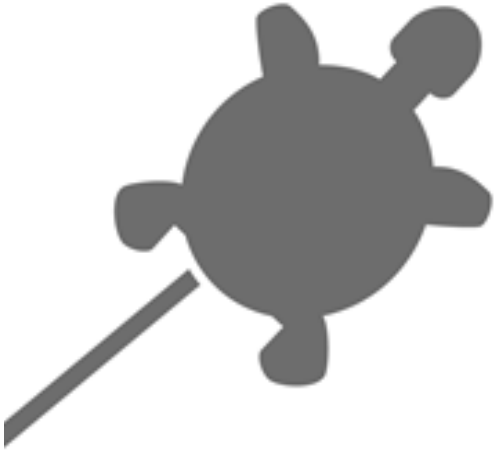
Turtles are just Python objects, so you can use any Python constructs in turtle programs: selection, loops, recursion, etc.

Turtles are objects that move about on a screen (window).

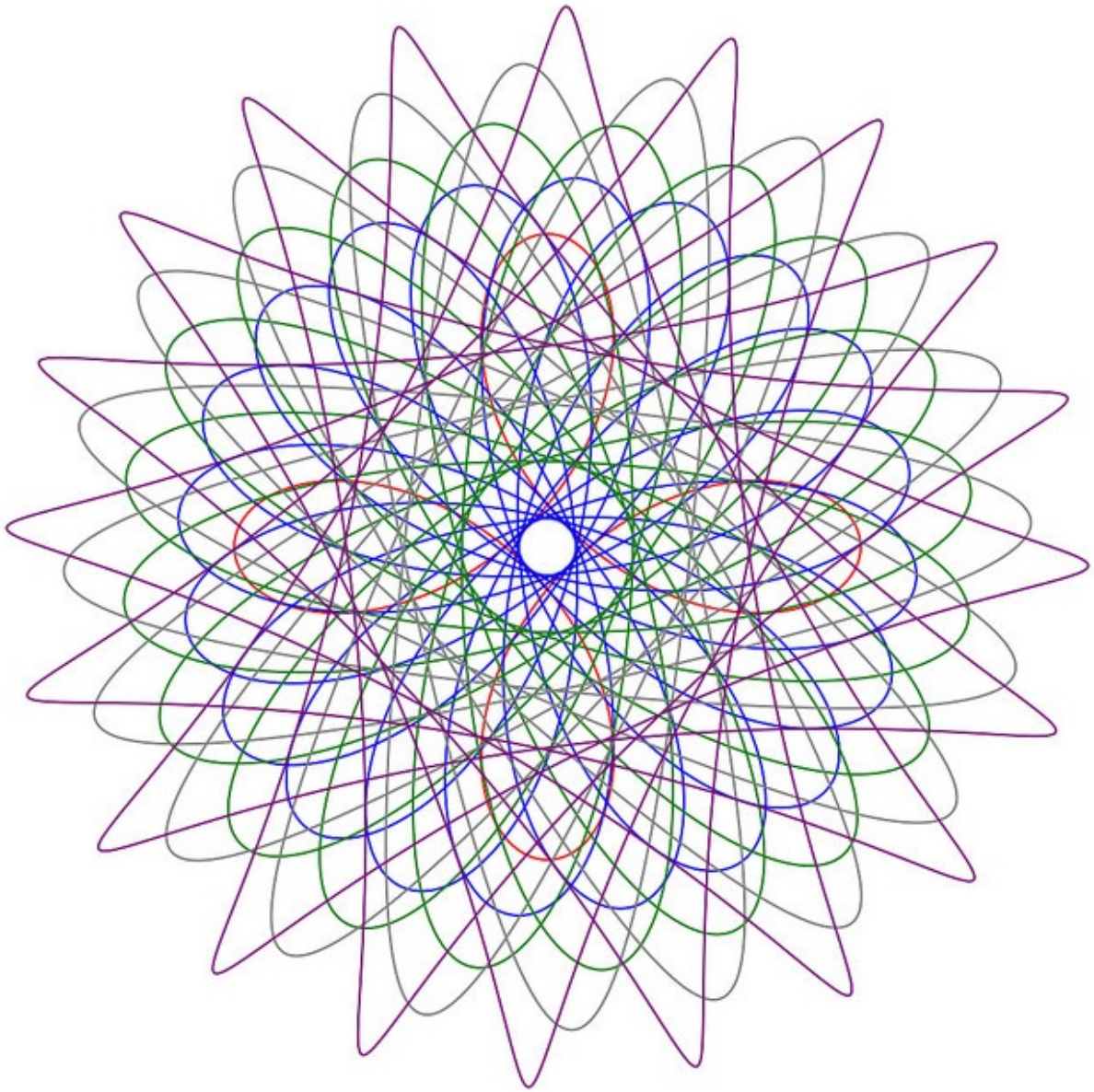
Various methods allow you to direct the turtle's movement.

The turtle's tail can be up or down. When it is down, the turtle draws on the screen as it moves.

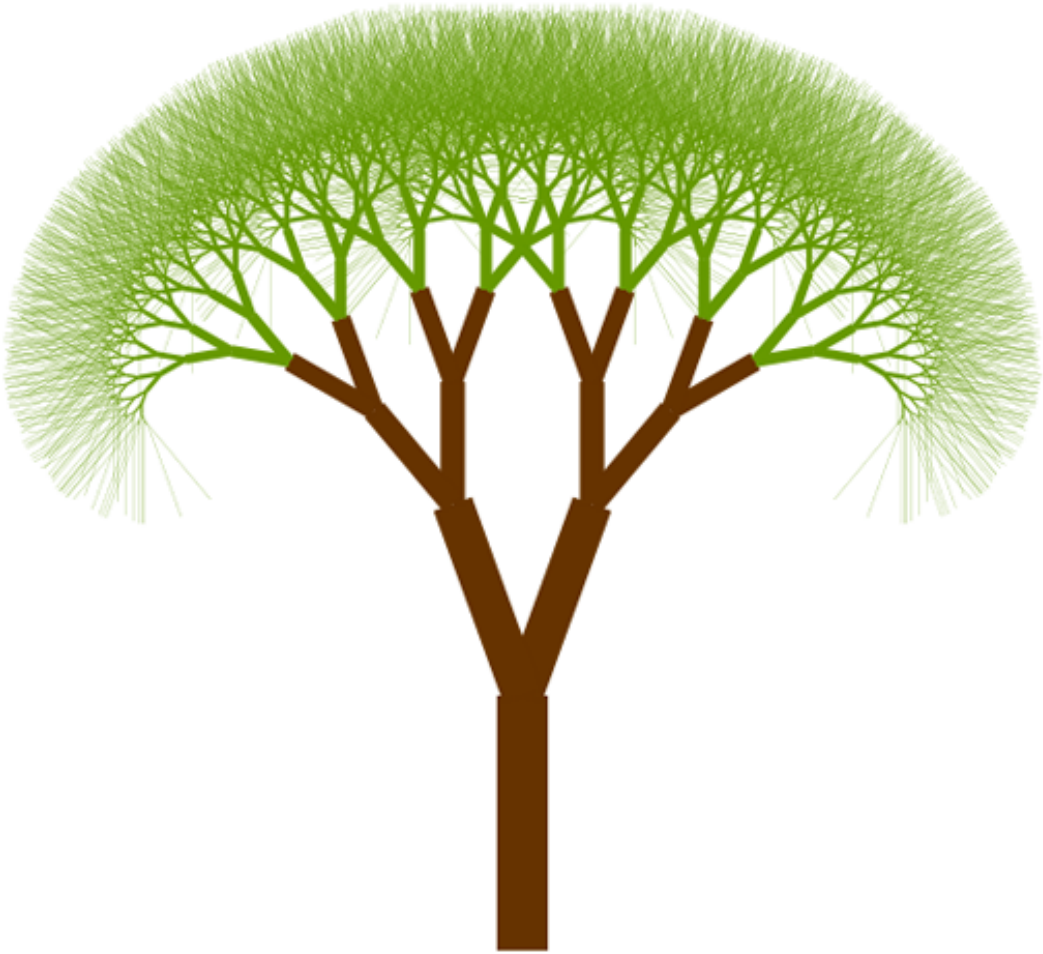
You can draw some pretty awesome images!



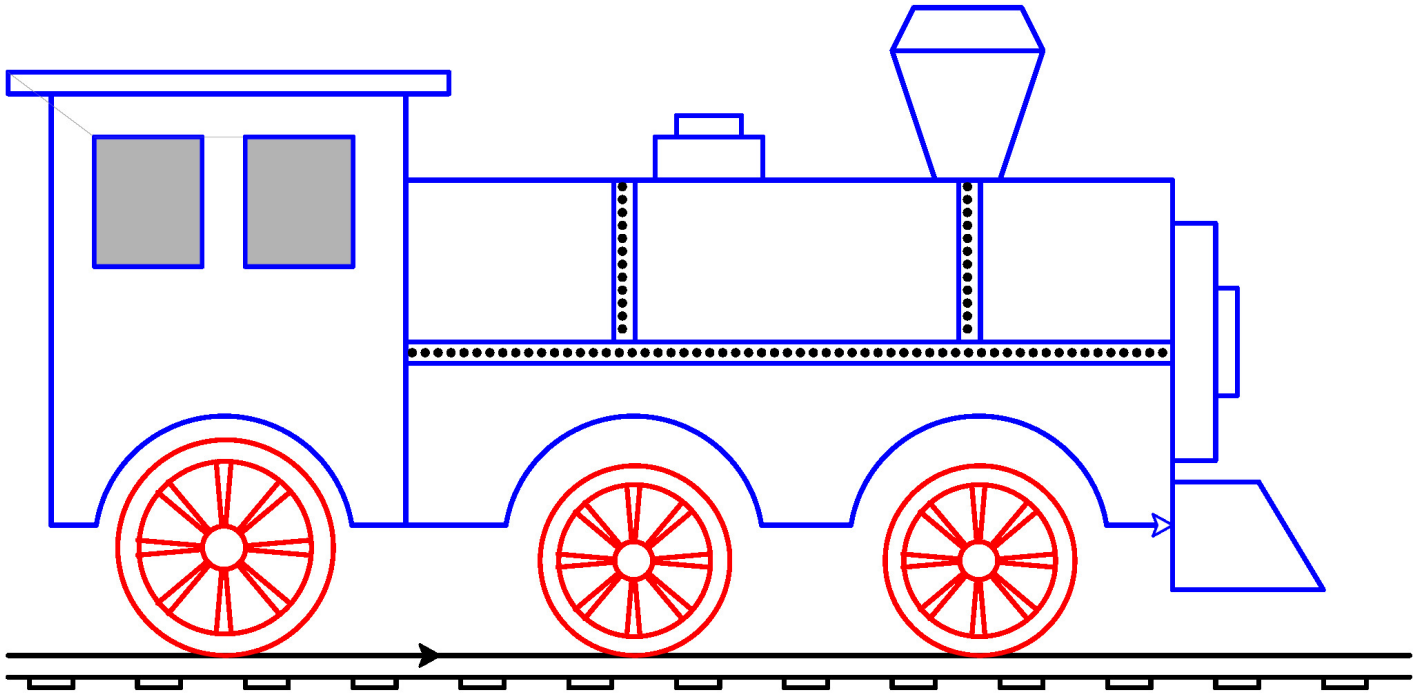
A Turtle Drawing



A Turtle Drawing

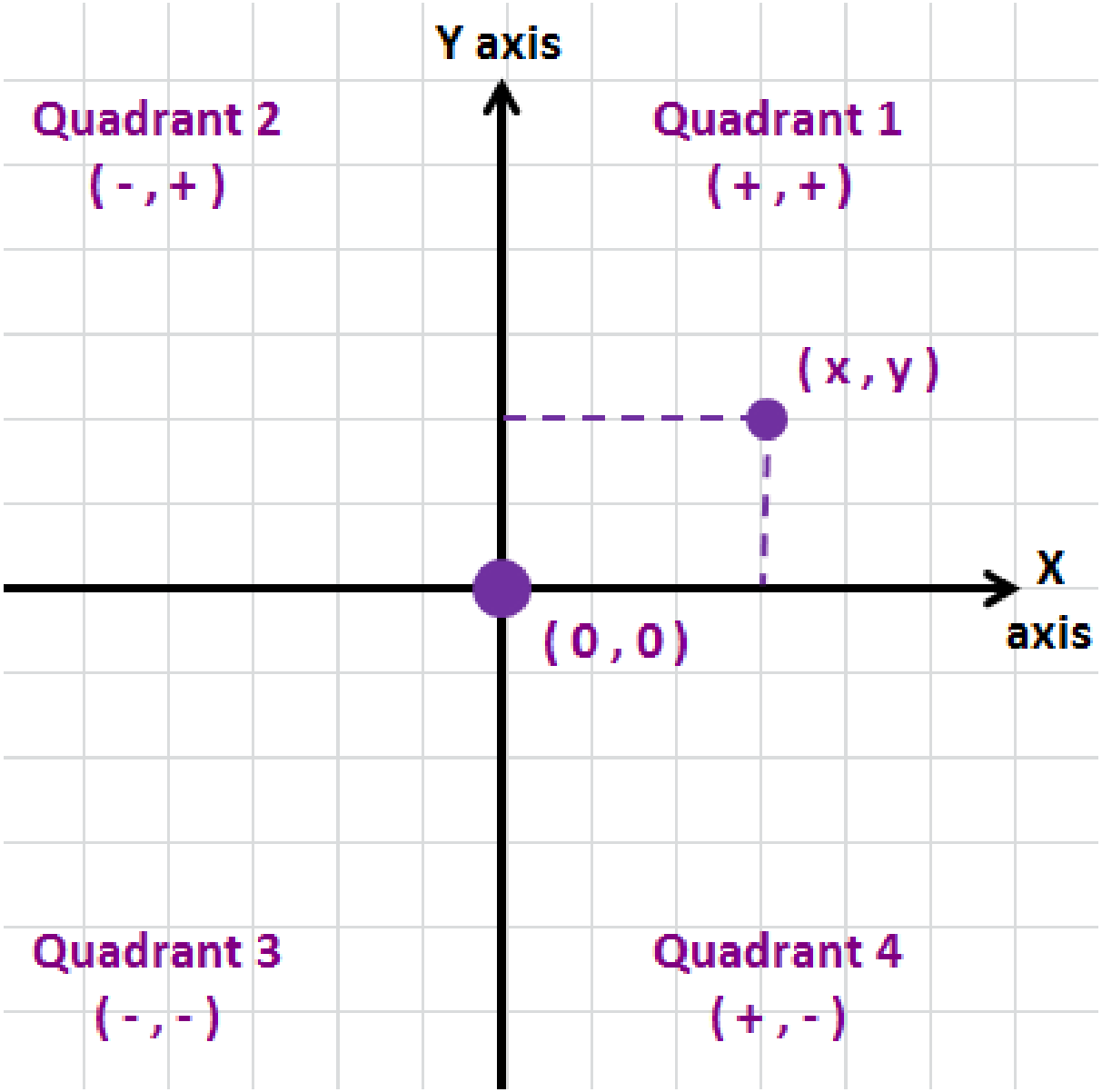


A Turtle Drawing: I Drew This One



A version of this picture was published in: William D. Young. "Modeling and Verification of a Simple Real-Time Gate Controller," in Michael Hinchey and Jonathan Bowen, editors, *Applications of Formal Methods*, Prentice-Hall Series in Computer Science, 1995, pp. 181–202.

Coordinate Grid



The Turtle's Data

Like all Python classes, the `turtle` class defines data and methods.

The data (attributes) of the turtle consists of:

Position: denoted by its current x and y coordinates; the units are pixels.

Heading: denoted by an angle in degrees. East is 0 degrees. north is 90 degrees; west is 180 degrees; south is 270 degrees.

Color: the color can be set to 2^{24} ($\sim$ 16.8 million) colors.

Width: the width of the line drawn as the turtle moves (initially 2 pixels).

Down: a Boolean attribute indicating whether the turtle's tail is down.

Turtle Methods

Many turtle methods are listing in your textbook (pages 81, 83) and online; Google "python turtle graphics."

t = Turtle()	create a new Turtle object and open its window
t.home()	move the turtle to $(0, 0)$, pointing east
t.pendown()	lower the tail (`t.down()` also works)
t.penup()	raise the tail (`t.up()` also works)
t.pensize(k)	set linewidth to k pixels
t.setheading(d)	change heading to direction d
t.left(d)	turn left d degrees
t.right(d)	turn right d degrees
t.speed(n)	set how fast the turtle moves (0 .. 10)
t.setx(n)	set the turtle's x coordinate, leave y unchanged
t.sety(n)	set the turtle's y coordinate, leave x unchanged

Turtle Methods

t.forward(n)	move in the current direction n pixels
t.backward(n)	move in the reverse direction n pixels
t.goto(x, y)	move to coordinates (x, y)
t.position()	return the current position at a tuple (x, y)
t.heading()	return the current direction (angle)
t.isdown()	return True if the pen is down
t.pencolor(r, g, b)	change the color to the specified RGB value or named color
t.write(s, font)	write a message to the screen (you can specify font and size, e.g., "font=('Arial', 8, normal)"

Keeping it On Screen

Because the window goes away immediately after the program terminates, it may be hard to see the result unless you delay things. You can use `turtle.done()` for that. Note that this is a *class method* rather than an *instance method*; that means that you need to use the class name `turtle.done()`, not the instance `leonardo.done()`.

turtle.done() make the screen persist until you close it

The turtle itself will appear on your screen as a small arrowhead. You can decide whether to show or hide the turtle.

t.hideturtle() make the turtle invisible

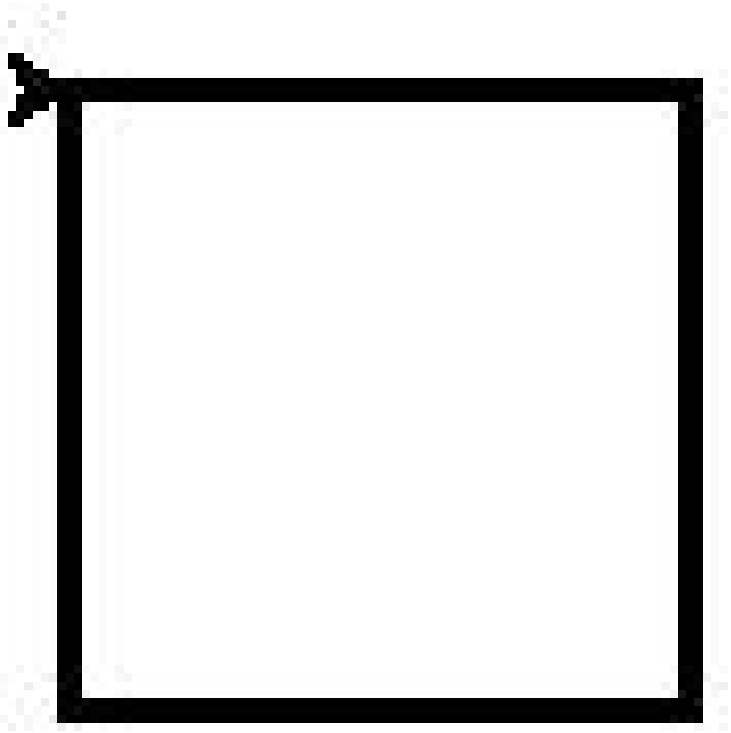
t.showturtle() make the turtle visible

t.isvisible() return True if the turtle is visible

A Turtle Function: Draw Square

```
import turtle
def drawSquare (ttl, x, y, length):
    """Draws a square using turtle ttl, with upper left
       corner at (x, y), and side of length"""
    ttl.penup()                 # raise the pen
    ttl.goto(x, y)              # move to starting position
    ttl.setheading(0)           # point turtle east
    ttl.pendown()               # lower the pen
    for count in range(4):      # draw 4 sides:
        ttl.forward(length)     #   move forward length;
        ttl.right(90)           #   turn right 90 degrees
    ttl.penup()                 # raise the pen

Bob = turtle.Turtle()           # our turtle is named Bob
Bob.speed(10)                   # make Bob crawl fast
Bob.pensize(3)                  # line width of 3 pixels
drawSquare( Bob, 0, 0, 100 )    # draw a square at (0,0)
                                #   with side length 100
turtle.done()                   # keep drawing showing
                                # note, it's a class method
```

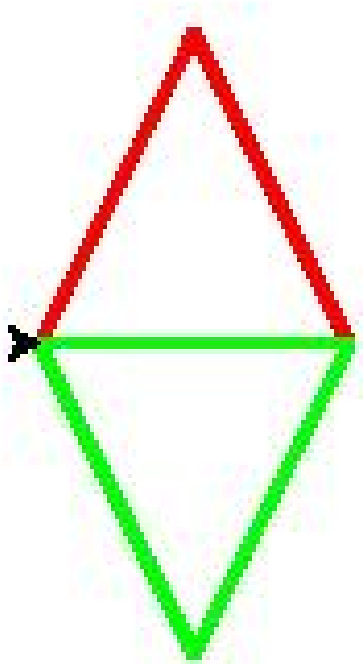
What the Turtle Drew



Draw Some Triangles

```
import turtle

def drawTriangle( ttl, x1, y1, x2, y2, x3, y3 ):
    ttl.penup()
    ttl.goto( x1, y1 )
    ttl.pendown()
    ttl.goto( x2, y2 )
    ttl.goto( x3, y3 )
    ttl.goto( x1, y1 )
    ttl.penup()

Tom = turtle.Turtle()           # our turtle is named Tom
Tom.speed(10)                   # make Tom crawl fast
Tom.pensize(3)                  # line width of 3 pixels

Tom.pencolor(1, 0, 0)           # set pen to red
drawTriangle( Tom, 0, 0, 50, 100, 100, 0 )
Tom.pencolor(0, 1, 0)           # set pen to green
drawTriangle( Tom, 0, 0, 50 ,-100, 100, 0 )
Tom.hideturtle()
```

What the Turtle Drew



Let's Take a Break

Colors

I'm not sure this works on all versions of turtle graphics.

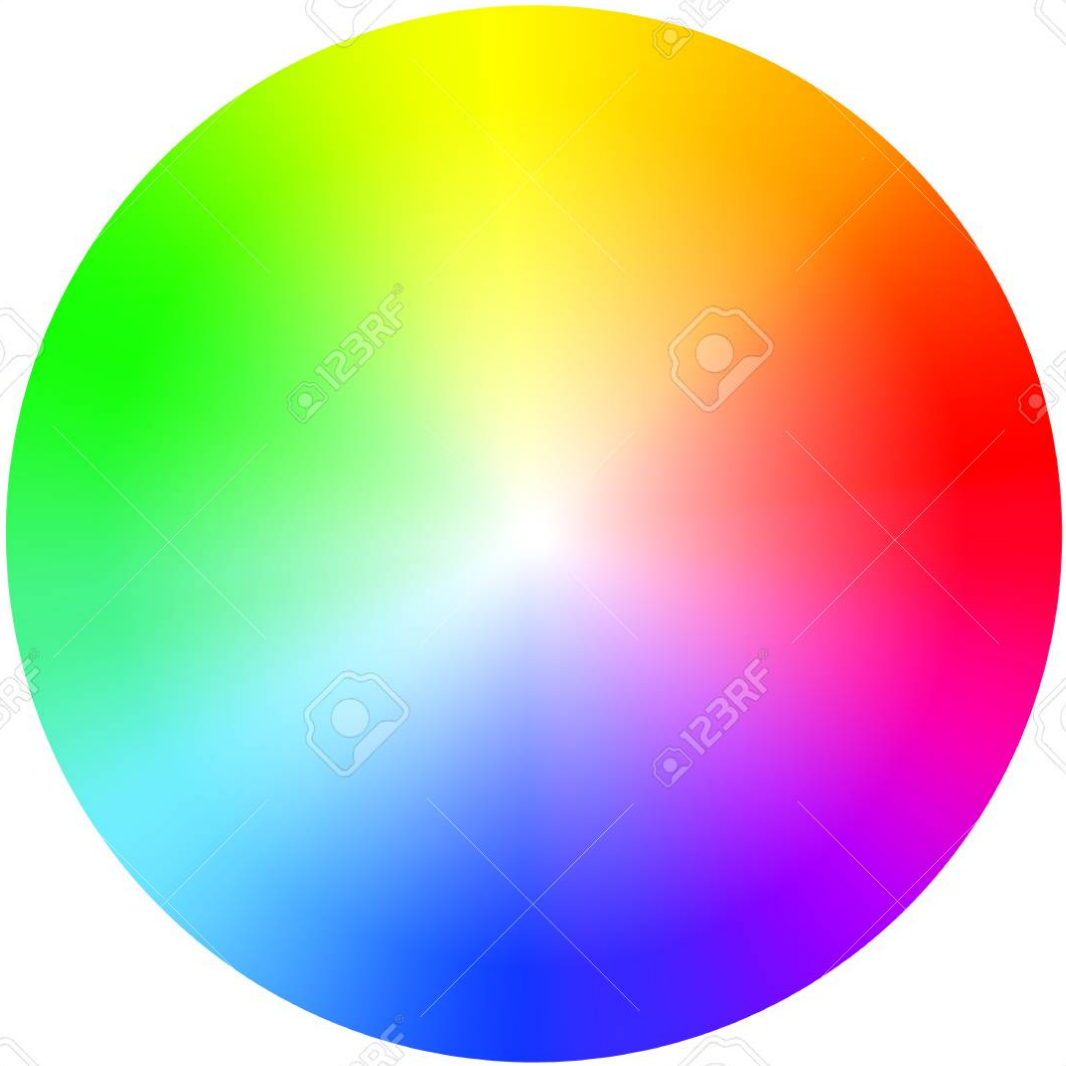
Colors are in the RGB system, using a triple: (R, G, B). Each element in the triple is an intensity from 0 to 255, indicating the contribution of R (red), G (green), and B (blue). For example:

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/color-picker`.

The *named* Python colors can be found here: `https://python-graph-gallery.com/python-colors/`.

Color Wheel



Colors

Turtles have two "colormodes" and you'll get an error if you try to do some things in the wrong mode. The modes are 1.0 and 255. In mode 255, use triples of range $0 \leq c \leq 255$. In mode 1, use triples (percentages) in range $0 \ldots 1$.

```
>>> 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)
```

Circles

You can draw circles, arcs, and dots using these functions:

t.circle(r, ext, step) draw a circle with radius r, ext (arc of circle drawn; 360 is entire circle), step (number of segments).

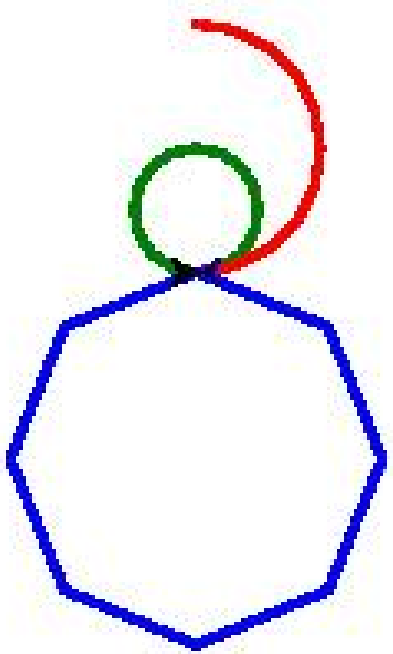
t.dot(d, color) draw a filled circle with diameter r and color

Note: the circle *is not centered* at the starting point. If you want that you could write:

```
def centeredCircle( ttl, r, x, y ):
    """ Draw a circle with radius r centered at (x, y). """
    ttl.up()                        # raise the pen
    angle = ttl.heading()           # save the current heading
    ttl.setheading(0)               # set heading east
    ttl.goto(x, y - r)              # move to bottom of circle
    ttl.down()                      # pen down
    ttl.circle(r)                   # draw the circle
    ttl.up()                        # pen up
    ttl.setheading(angle)           # restore the heading
```

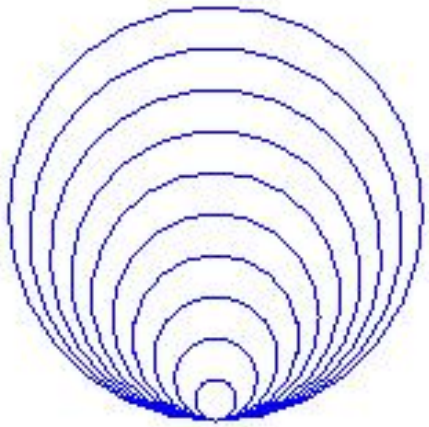
Circles

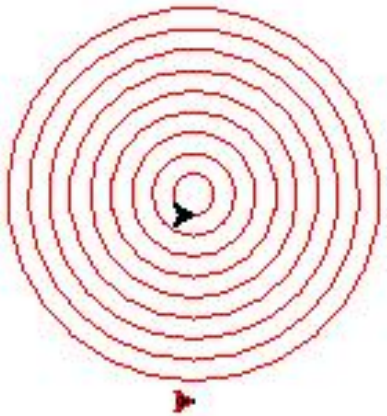
```
def drawSomeCircles(ttl):
    ttl.speed(10)
    ttl.pensize(3)      # line is 3 pixels
    ttl.up()
    ttl.home()          # go to (0, 0)
    ttl.down()
    ttl.pencolor('Green')
    ttl.circle(25)      # rad. 25 pixels
    ttl.up()
    ttl.goto(0, 0)
    ttl.pencolor('Red')
    ttl.down()
    ttl.circle(50,180)  # arc 180 deg.
    ttl.up()
    ttl.goto(0, 0)
    ttl.pencolor('Blue')
    ttl.down()
    ttl.circle(75,360,8)  # octogon
    ttl.up()

Sam = turtle.Turtle()
drawSomeCircles(Sam)
```



More with Circles

```
def tangentCircles(ttl):
    """ Print 10 tangent circles. """
    r = 10          # initial radius
    n = 10          # count of circles
    for i in range(1, n + 1, 1):
        ttl.circle(r * i)

def concentricCircles(ttl):
    """ Print 9 concentric circles. """
    r = 10          # initial radius
    for i in range(1, 10):
        ttl.circle(r * i)
        ttl.up()
        ttl.sety((r * i)*(-1))
        ttl.down()

Ben = turtle.Turtle()
Ben.up(); Ben.goto(0, 150)
Ben.down(); Ben.pencolor('Blue')
tangentCircles(Ben)
Ben.up(); Ben.goto(0, -150)
Ben.down(); Ben.pencolor('Red')
concentricCircles(Ben)
```

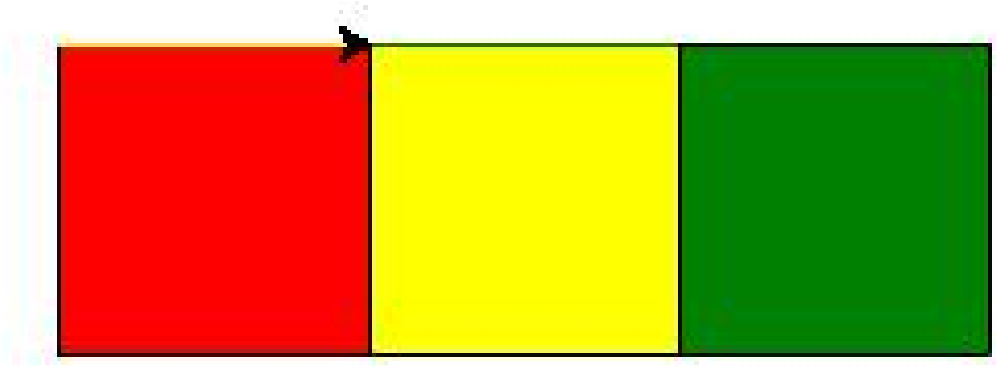




Fill Areas

If you draw a closed region, you can fill it with a specified color:

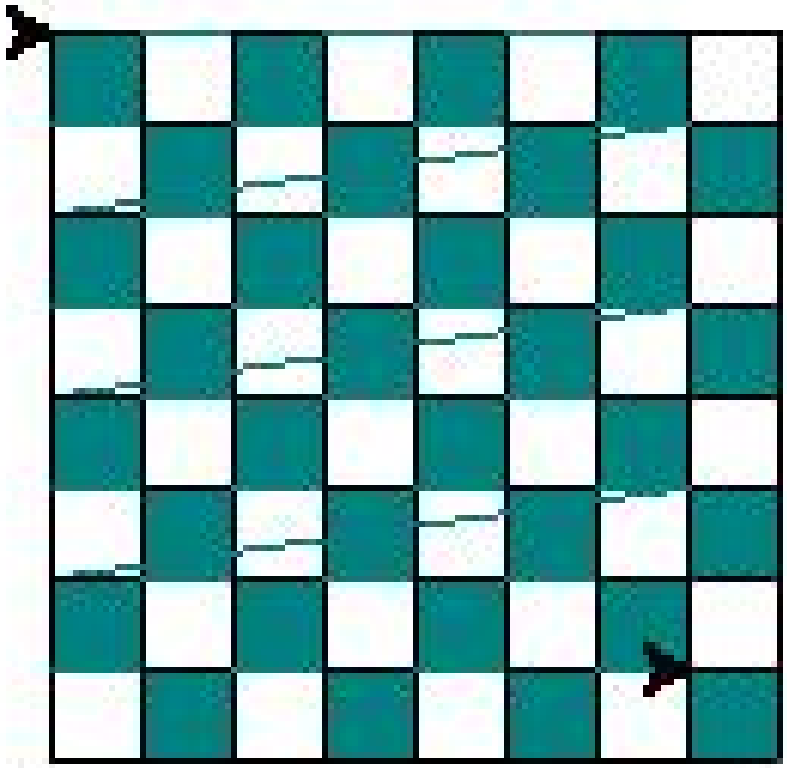
t.fillcolor()	sets the pen fill color
t.begin_fill()	call this before filling a shape
t.end_fill()	call to no longer keep filling
t.filling()	return True if filling, False otherwise



Drawing a Chessboard

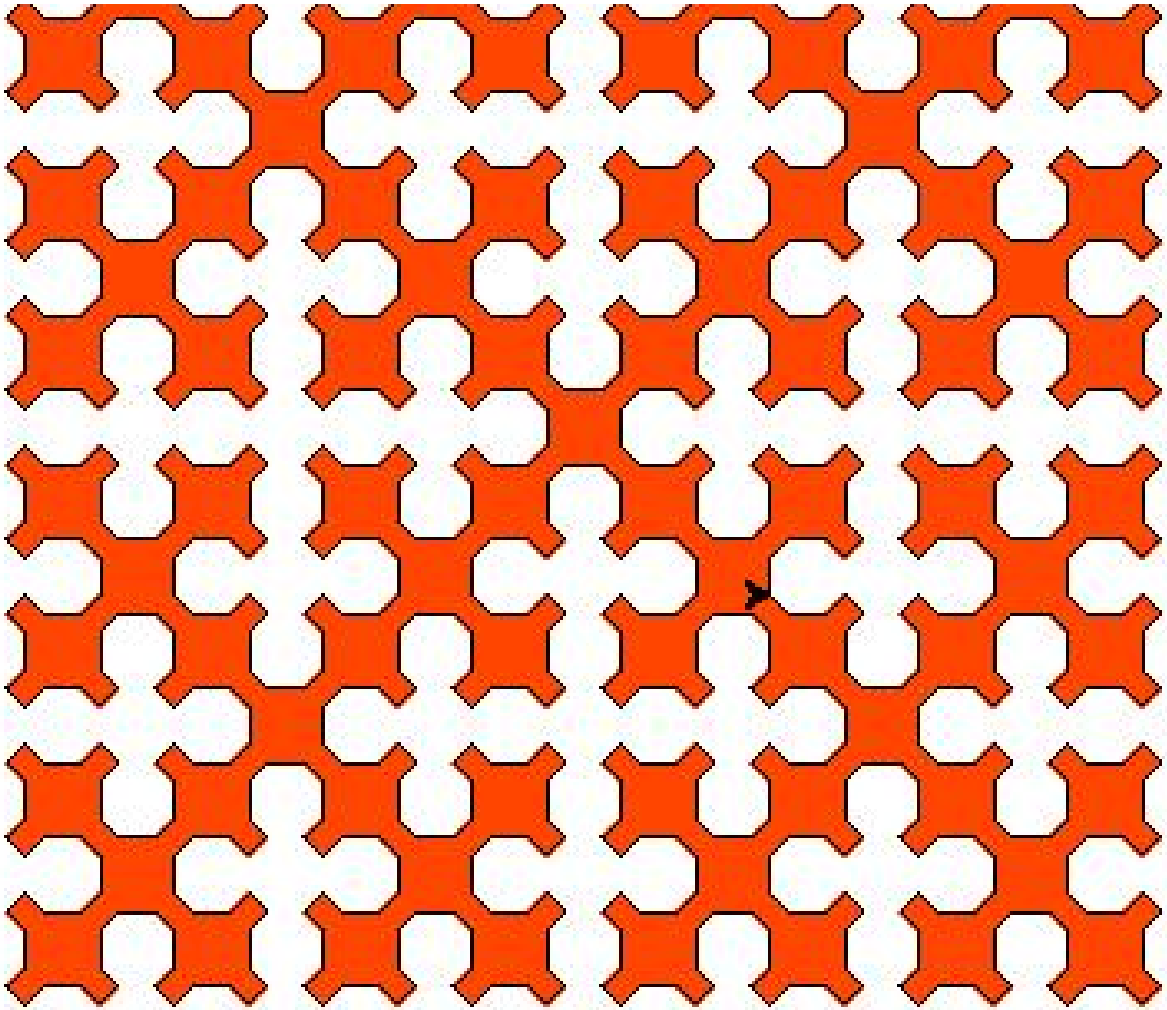
```
def maybeFillSquare (ttl, x, y, lngth, fill, color):
    """ Boolean parameter fill says whether to fill. """
    if fill:
        ttl.fillcolor( color )
        ttl.begin_fill()
        drawSquare( ttl, x, y, lngth )
        ttl.end_fill()
    else:
        drawSquare( ttl, x, y, lngth )

def drawChessboard (ttl, x, y, squaresize):
    """ Draw a teal and white chessboard. """
    fill = True
    for j in range( 8 ):
        for i in range( 8 ):
            x1 = x + i*squaresize
            y1 = y - j*squaresize
            maybeFillSquare( ttl, x1, y1, squaresize,
                             fill, 'teal')
            fill = not fill
        fill = not fill

Matt = turtle.Turtle()
drawChessboard( Matt, 0, 0, 20 )
```

Our Chessboard



I don't know why those weird lines are in there. They don't show up on the screen.

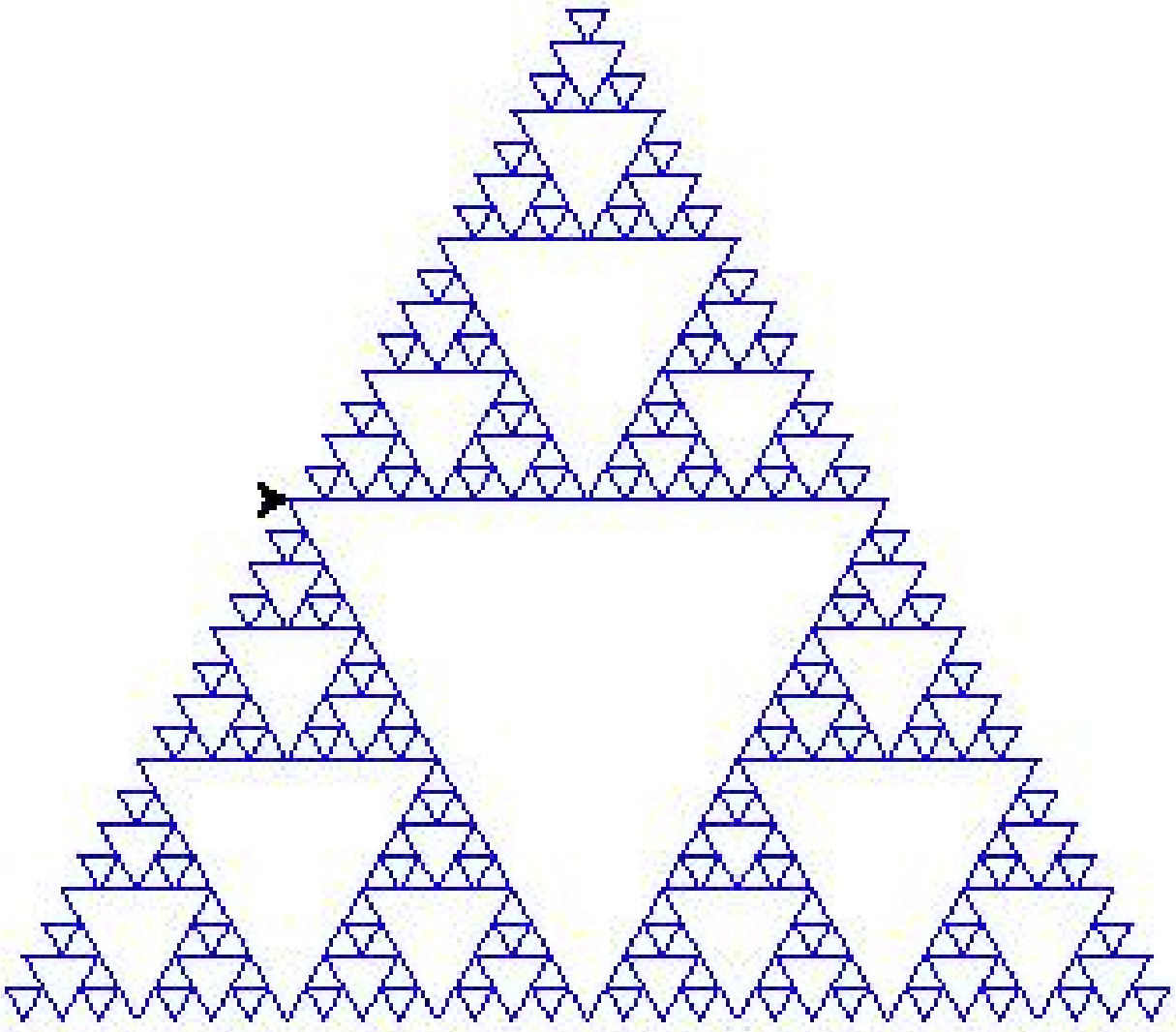
Some Complex Stuff: Sierpinski Curve



Some Complex Stuff: Sierpinski Curve

```
def oneSide( ttl, s, diag, level ):
    if (level == 0):
        return
    else:
        oneSide( ttl, s, diag, level - 1 )
        ttl.right(45); ttl.forward( diag ); ttl.right(45)
        oneSide( ttl, s, diag, level - 1 )
        ttl.left(90); ttl.forward( s );ttl.left(90)
        oneSide( ttl, s, diag, level - 1 )
        ttl.right(45); ttl.forward( diag ); ttl.right(45)
        oneSide( ttl, s, diag, level - 1 )

def sierpinski( ttl, s, level ):
    diag = s / math.sqrt(2)
    for i in range(4):
        oneSide( ttl, s, diag, level )
        ttl.right(45)
        ttl.forward( diag )
        ttl.right(45)
```

Some Complex Stuff: Fractal Triangles



Fractal Triangles

```
def drawOutwardTriangles( ttl, size ):
    if size < 10:
        return
    for i in range( 3 ):
        ttl.forward( size / 2 )
        insert( ttl, size )
        ttl.forward( size / 2 )
        ttl.right( 120 )

def insert( ttl, size ):
    ttl.left( 120 )
    drawOutwardTriangles( ttl, size / 2 )
    ttl.right( 120 )

Ken = turtle.Turtle()
Ken.color("blue")
drawOutwardTriangles( Ken, 200 )
```

Saving Your Picture

Python saves your picture as a postscript file, but you can convert it. To save your picture as a jpeg, do the following:

```
from PIL import Image

def save_as_jpg(canvas,fileName):
    # save postscipt image
    canvas.postscript(file = fileName + '.eps')
    # use PIL to convert to JPEG
    img = Image.open(fileName + '.eps')
    img.save(fileName + '.jpeg', 'jpeg')

< Your drawing functions >

ts = turtle.getscreen()
tc = ts.getcanvas()
# creates a postscript image file
# substitute your own filename
tc.postscript(file="filename.eps")
# converts to JPEG
save_as_jpg(tc,"filename")

turtle.done()
```

Some Sample Projects

Write Turtle graphics functions that will do the following:

1. draw a cube;
2. draw a regular polygon with k sides and radius r (distance from center to one of the vertices);
3. draw m concentric circles, varying the color as you go outward;
4. draw an American flag;
5. draw the UT Tower.

We're Done!