Introducing Python Via Turtle Graphics

John M. Morrison

November 30, 2020

Contents

1 Getting Started

You can proceed in a couple of different ways. An easy way to get going in Python is to to to the site `http://trinket.io` and make yourself an account. This provides a Python programming environment that requires no installation; all you need is a browser. All of the examples here can be run in this environment.

If you are running a PC or a Mac, you can install and run Python on your machine by going to the site `http://www.python.org` and following the instructions there. You should install Python3 and Tk. The Tk machinery provides the graphics environment your turtles will inhabit and draw and paint in.

2 Introduction

Our purpose here is to provide a brief introduction to Python using turtle graphics. No prior knowledge of Python is assumed. We will emphasize the notions of objects and properties. The intent here is to introduce computing to the student as a visual enterprise. A lot of our discussion will revolve around colors and graphics. These will be used to motivate the idea of bases and numbers.

We begin by introducing the creation of turtles and their worlds.

3 Creating a Window

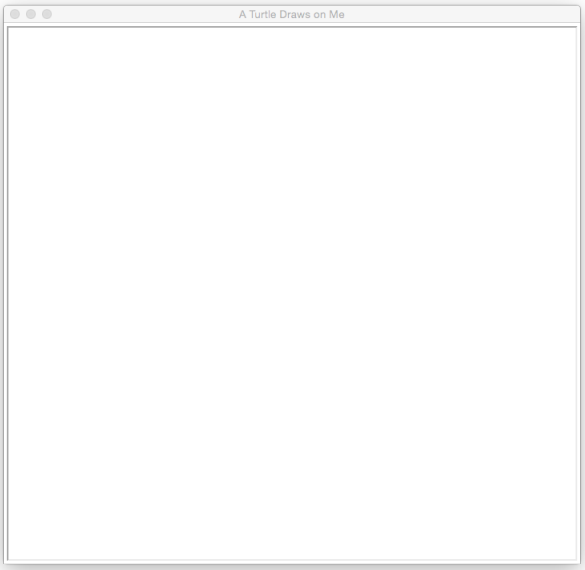
Let us get started by creating a window. The essential ingredients are a `turtle` and a `screen`. A turtle moves about in a window and draws on the screen. Here

is our very first program that just pops up a window.

```
import turtle

window = turtle.Screen()
window.title("A Turtle Draws on Me")
window.exitonclick()
```

Now run your program and you will get a window like this to pop up on your screen. This is a `Screen` object.



Let us now march through this code and see what it does. The statement

```
import turtle
```

tells Python you want to use the `turtle` library. This library allows you to create screens and turtles. It also allows you to change the properties of the objects it creates for you. The next line

```
window = turtle.Screen()
```

created the screen. Think of it as saying "`turtle` library, make me a `Screen` and name it `window`. The next line

```
window.title("A Turtle Draws on Me")
```

placed a title in the window's title bar. The `.window` method tells the window to put whatever title you give it into `window`'s title bar. Finally

```
window.exitonclick()
```

causes the new window to go a away when you click the mouse in its content pane, i.e. the main contents of the window that excludes the title bar and any other toolbars. You should run this program yourrself. Try changing the title.

Programming Exercise Make a copy of `popWindow.py`. We are going to experiment a little with the window we created. Add this line to the code.

```
window.bgcolor("orange")
```

Experiment with colors. Can you make a list of ten different colors that work?

4 Creating a Turtle

Turtles know how to draw. A turtle has a pen and it draws a line one pixel wide as he moves along. As a first simple example, we will draw a square. Run this program to see it happen.

```
import turtle

window = turtle.Screen()
window.title("Yertle draws a square")
yertle = turtle.Turtle()  #Make a turtle.
yertle.forward(50)        #Move the turtle 50 pixels.
yertle.left(90)           #Make the turtle turn left 90 degrees
yertle.forward(50)
yertle.left(90)
yertle.forward(50)
yertle.left(90)
yertle.forward(50)
yertle.left(90)
yertle.hideturtle()       #Hide the turtle.
window.mainloop()         #Keep window running until you click it away
```

Now let us understand what happens. First of all you will see text after a pound sign (#). Any text on a line after a pound sign is ignored by Python. It

is simply a *comment*, or a note we are making to other human readers of our programs. Comments also serve to remind us of what we did so we do not see it later and have to re-puzzle out its meaning.

The line

```
yertle = turtle.Turtle()  #Make a turtle.
```

makes a new turtle whose name is `yertle`. We will use `yertle`'s name to communicate with him.

A turtle can move backwards or forwards. You must tell it how many pixels to move. As the turtle moves, its pen draws on the screen. The line

```
yertle.forward(50)
```

is saying, "`yertle`, move yourself forward 50 pixels." The turtle `yertle` has the behavior `forward`. You just have to tell the turtle how far to move in pixels. Turtles can turn between moves. We specify this by giving the turtle an angle in degrees. This line

```
yertle.left(90)            #Make the turtle turn left 90 degrees
```

tells `yertle` to turn 90 degrees to his left. Turning is done from the turtle's eyes point of view. Naturally, there is a similar behavior `right` that can cause a turtle to turn right a specified number of degrees.

Programming Exercises

1. Put a comment in front of the line

   ```
   yertle.hideturtle()
   ```

   so you can see what the turtle looks like (it's not much).
2. Write a program called `hexagon.py` that has a turtle draw a hexagon with sides of length 100 pixels.
3. Write a program called `square1.py` that draws a square by having the turtle turn right instead of left. Can you also have a square drawn by having the turtle move backwards.
4. Make a program with turtle `yertle` in it. Move the `yertle` 50 pixels, then do this

   ```
   yertle.penup()
   ```

   Move him 50 pixels again, then do this

   ```
   yertle.pendown()
   ```

   and move him 50 pixels again. What happened? What is the result of `penup()` and `pendown()`

5 Screen Cöordinates

The screen has cördiantes. We can move the the turtle around and have him report his location. Let us make a program that does that called `coordinated.py`.

```
import turtle

window = turtle.Screen()
reporter = turtle.Turtle()
reporter.write(reporter.position())
reporter.penup()
reporter.forward(100)
reporter.write(reporter.position())
reporter.left(90)
reporter.forward(100)
reporter.write(reporter.position())
window.mainloop()
```

Run this program. You can see that the origin $(0, 0)$ is in the middle of the screen. The x and y coördinates work like they do in a math class. The unit of measure is a pixel. Now make this program, `northeast.py`.

```
import turtle
window = turtle.Screen()
bishop = turtle.Turtle()
bishop.penup()
bishop.left(45)
bishop.forward(100)
bishop.write(bishop.position())
window.mainloop()
```

Notice that the `bishop`'s (yeah.... he moves diagonally) coördiantes are now $(70.71, 70.71)$. This is unsurprising if you think about 45-45-90 triangles and the fact that

$$\frac{100}{\sqrt{2}} = 70.71,$$

to two decimal places.

Programming Exercises Make a turtle called warpDrive. Then do this.

```
warpDrive.setposition(100,100)
warpDrive.write(warpDrive.position())
warpDrive.forward(100)
```

```
warpDrive.write(warpDrive.position())
warpDrive.home()
warpDrive.write(warpDrive.position())
```

What does `setpositon` do? Why did we name the turtle `warpDrive`?

6 Turtles are Smart

Turtles are examples of Python objects. An object is a piece of data stored in memory. Objects have three important properties.

1. *state* An object has things it knows. For example, turtle knows if its pen is up or down. It knows its position and athe direction it faces to. It knows the color that is now loaded in its pen.
2. *identity* An object *is*. Objects are self-aware and know they are different from other objects. We will see next that we can have several turtles in a screen, each doing its own thing. The turtles will know they are different from each other and will act when their name is used.
3. *behavior* A turtle can put its pen up. It can put is pen down. We also saw a turtle can change the color its pen is drawing. It can move; we must tell it how many pixels to move and whether to move forward or backward. We can also tell it to appear anywhere on the screen. It can turn through any angle we specify.

The screen is an object, too. A screen knows its background color, and it can change its background color to other colors.

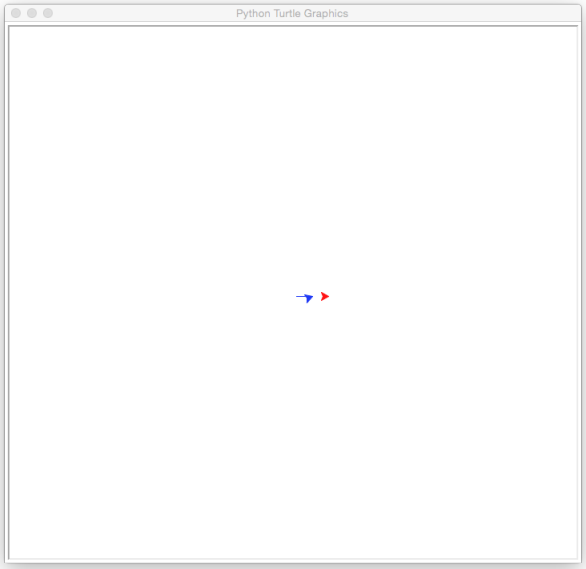
Now let us try an example where we make two turtles and have them both draw on the screen. We create two turtles. One uses red ink and alternately raises and lowers its pen. The other turns and draws, keeping it pen down and drawing in blue. Run this and it is not very interesting.

```
import turtle
window = turtle.Screen()
dotty = turtle.Turtle()
bender = turtle.Turtle()

dotty.color("red")
bender.color("blue")
#basic motion
dotty.forward(20)
dotty.penup()
dotty.forward(20)
```

```
dotty.pendown()
bender.forward(20)
bender.left(20)
window.mainloop()
```

Running the program reveals this.



Now let us copy and past and see it execute several times.

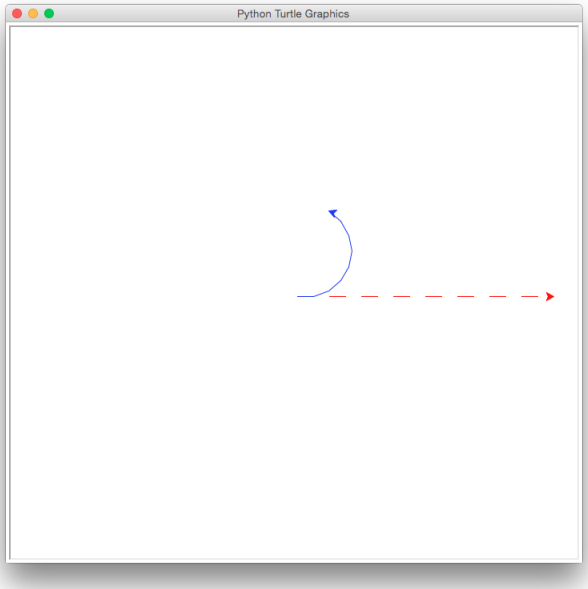
```
import turtle
window = turtle.Screen()
dotty = turtle.Turtle()
bender = turtle.Turtle()
dotty.color("red")
bender.color("blue")
##basic motion
dotty.forward(20)
dotty.penup()
dotty.forward(20)
dotty.pendown()
bender.forward(20)
bender.left(20)
##repeat
dotty.forward(20)
```

```
dotty.penup()
dotty.forward(20)
dotty.pendown()
bender.forward(20)
bender.left(20)
#repeat
dotty.forward(20)
dotty.penup()
dotty.forward(20)
dotty.pendown()
bender.forward(20)
bender.left(20)
#repeat
dotty.forward(20)
dotty.penup()
dotty.forward(20)
dotty.pendown()
bender.forward(20)
bender.left(20)
#repeat
dotty.forward(20)
dotty.penup()
dotty.forward(20)
dotty.pendown()
bender.forward(20)
bender.left(20)
#repeat
dotty.forward(20)
dotty.penup()
dotty.forward(20)
dotty.pendown()
bender.forward(20)
bender.left(20)
#repeat
dotty.forward(20)
dotty.penup()
dotty.forward(20)
dotty.pendown()
bender.forward(20)
bender.left(20)
#repeat
dotty.forward(20)
dotty.penup()
dotty.forward(20)
dotty.pendown()
bender.forward(20)
```

```
bender.left(20)

window.mainloop()
```

Run this (ungainly) code and see this.

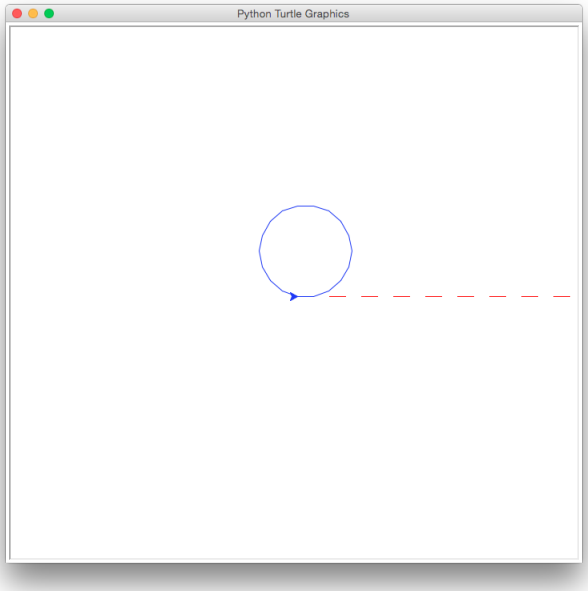


Ugh! Look at all of that repeated code! Wouldn't it be nice if we could tell code to repeat itself? We can; we will use a *for loop*. Here is how it works. We will run the basic motion 18 times; notice how we get a blue polygon as a result and notice how `dotty` runs off the screen.

```
import turtle
window = turtle.Screen()
dotty = turtle.Turtle()
bender = turtle.Turtle()
dotty.color("red")
bender.color("blue")
##basic motion
for k in range(18):
    dotty.forward(20)
    dotty.penup()
    dotty.forward(20)
    dotty.pendown()
```

```
    bender.forward(20)
    bender.left(20)
window.mainloop()
```

Run your program and you will see this.



Let us focus on the a very simple `for` loop. It looks like this.

```
for k in range(someNumber):
    statement1
    statement2

        .
        .
        .

    lastStatement
```

This loop will run the statements below it `someNumber` times. Notice how all of the statements we want repeated are indented. This indented sequence is called a *block* of code. The `for` loop runs its block of code the number of times you specify. Use your tab key to indent, or use four (or two) spaces. You must do the same thing consistently or Python will hiss error messages. Here,

it pays to be strictly consistent. A `for` loop's block of code must have at least one (indented) line. The block ends where the indentation ends.

Programming Exercises

1. Create a program called `shapes.py` Make a turtle and have him draw a square in red that is 100 pixels on a side.
2. Add a turtle, move him away from the first one to $(100, 100)$, and have him draw an equilateral triangle in blue there.
3. Make a third turtle, and have him draw a circle of radius 100. A turtle knows how to draw a circle, use its `.circle` method.
4. Make a fourth turtle, and have him draw an octagon with sides of length 50 pixels in red starting at $(-200, 200)$.
5. After the drawing is done, hide all four turtles.
6. On one of your turtles, call `.pensize(10)` on it. What does that do?

7 Being Persistent

We are going to look at this problem. Have a turtle march forward 20 pixels at a time until it gets within 30pixels of the right edge of the screen. Then have it turn left. Then have it march along 20 pixels at a time until it gets within 30 pixels top of the screen. Then end. Let us think about the things we need to know.

- How do we know when we get near the edge of the screen?
- Who knows where the screen ends?
- How do we tell the turtle to "keep marching?"

Let us begin with this program. Place it in a file called `automaton.py`. You can see that the screen knows its size; the size of the screen is part of its state.

```
import turtle

window = turtle.Screen()
checker = turtle.Turtle()
checker.write(window.window_width())
checker.forward(100)
checker.write(window.window_height())
window.mainloop()
```

You wlll see that the width of the window is written near the origin and the height is written 100 pixels away to the right.

```
width= window.window_width()
```

Let us march a turtle to the right until it gets close to the end.

```
import turtle
window = turtle.Screen()
width, height = window.screensize()
while turtle.xcor() <= width - 30:
    turtle.forward(20)
window.mainloop()
```

Run this and watch the turtle march right off the end. This is because we are already in the middle of the screen, so we should divide the width by 2.

```
import turtle
window = turtle.Screen()
rightRoom = window.window_width()
lefty = turtle.Turtle()
while lefty.xcor() <= rightRoom/2 - 30:
    lefty.forward(20)
window.mainloop()
```

Now let us have him turn left and head up

```
import turtle
window = turtle.Screen()
lefty = turtle.Turtle()
width= window.window_width()
height = window.window_height()
while lefty.xcor() <= width/2 - 30:
    lefty.forward(20)
lefty.left(90)
while lefty.ycor() <= height/2 - 30:
    lefty.forward(20)
window.mainloop()
```

Here is a little Python grammar. Python has two types of statements, boss statements and worker statements. Consider the statement

```
yertle.forward(10)
```