Lab Week 14 - Inheritance

Skills needed to complete this Lab (You may not use all of these skills)

|  |
| --- |
| * Create and use classes * Inherit from a previously defined class. |

## Turtle

We’re going to use the turtle module to draw on the screen a little. Take a few minutes to get familiar with turtle.

>>> import turtle

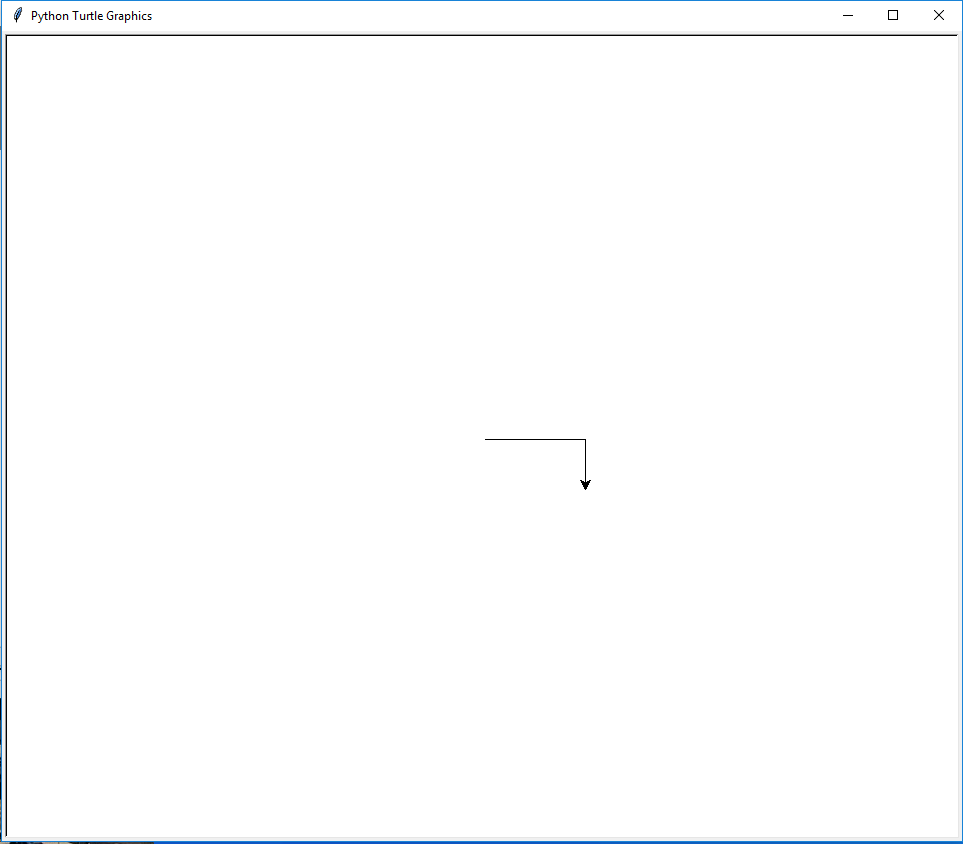
>>> turtle.forward(100)

>>> turtle.right(90)

>>> turtle.forward(50)

>>>

Once you do the first turtle.forward(), you’ll notice it opens a new window that has the line you’ve just drawn.



Experiment with turning left and right, and drawing lines. Keep in mind that when we are drawing a box, or a circle, the direction of the turtle may not be at the angle we intend.

Some of the skills we’re going to need is are

* turtle.right( angle )
* turtle.forward( distance )
* turtle.circle( radius )

In some of the classes we want to create a class that draws a filled shape. In which case we need to use turtle.fillcolor(color), turtle.begin\_fill(), and turtle.end\_fill()

>>> turtle.fillcolor("red")

>>> turtle.begin\_fill()

>>> turtle.forward(100)

>>> turtle.right(120)

>>> turtle.forward(100)

>>> turtle.right(120)

>>> turtle.forward(100)

>>> turtle.end\_fill()

>>>

## Creating our super class

Create a solution for the project and enter following

import turtle

class Point(object):

def \_\_init\_\_(self, x, y, color):

self.x = x

self.y = y

self.color = color

def draw(self):

turtle.penup()

turtle.goto(self.x, self.y)

turtle.pendown()

turtle.color(self.color)

turtle.setheading(0)

self.draw\_action()

def draw\_action(self):

turtle.dot()

Try running this program and create an instance of the point in the shell.

>>> p = Point(-100, 100, "blue")

>>> p.draw()

>>>

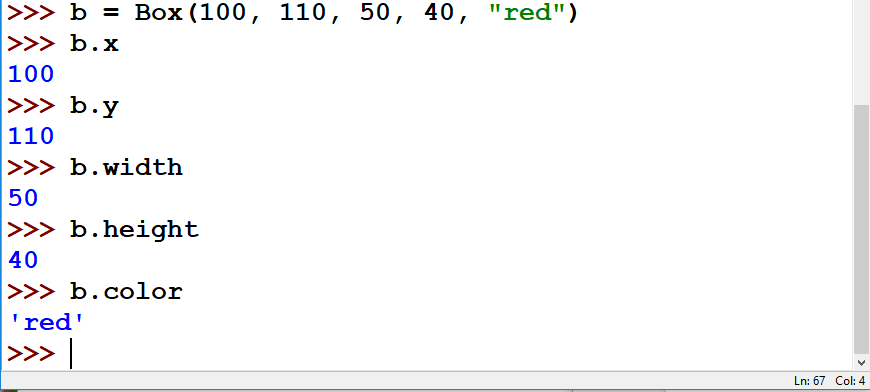
## The Box Subclass

In your solution create a new class called Box that inherits from Point

Now create an \_\_init\_\_ method that has parameters; self, x1, y1, width, height, color. Since we are inherited from point, we can have the super class setup our our instance by calling

super().\_\_init\_\_(x1, y1, color)

After that in the init you want to set the width and height for the self instance. Test your new class in the shell. Make sure you can access the attributes for the class.



You’ll see we didn’t have to repeat the actions of setting the attributes for x, y and color since we inherited from Point and called its init.

Now create a draw\_action method for how to draw the box.

def draw\_action(self):

""" Draws a box """

Remember to draw the box, the turtle will already be heading to the right. So you should move forward by the width for the instance, turn right by 90 degrees, then move forward by the height, turn right by 90 degrees again, draw a line for the width, turn right again and draw a line for the length of the height.

Run the module and test creating a box object and calling the draw method.

>>> b = Box(-100, 100, 50, 20, "blue")

>>> b.draw()

>>>

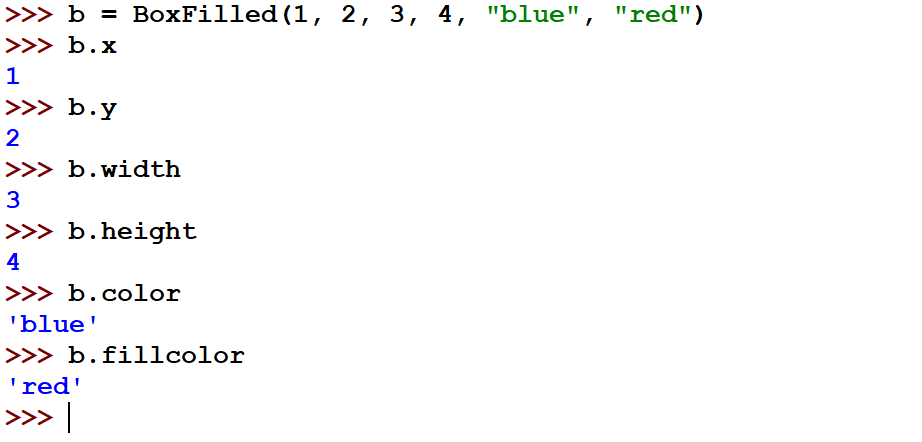
We didn’t create a draw() method for our object. When we call draw() it can’t find draw in our box class, so it checks for draw in our super class ( point ). It calls draw in the super class which draws our specific draw\_action method in Box. Again, the super class did some of the work setting our turtle object up to draw consistently and all we had to do in draw\_action was to draw the box.

## Create BoxFilled class.

Create another class in the solution called BoxFilled that inherits from Box. Then create an init method that takes x1, y1, width, height, color and fillcolor as parameters.

Call super().\_\_init\_\_() method. Remember we are inheriting from Box this time so pass it the values that Box needs to be instantiated. ( we don't’ have to pass self to the super init )

Then set the attribute for fill color in the init. Test your class in the shell.



Now create a draw\_action method for the BoxFilled class. We already know how to draw a box and have that code in our Box class. So instead of repeating that code, we’ll call our box class as shown below

Box.draw\_action(self)

Test creating a filled box in the shell

>>> b = BoxFilled(1, 2, 100, 200, "red", "Blue")

>>> b.draw()

>>>

That should have drawn the box given at x = 1, y = 2, width 100, and height 200 with a red outline. However, it didn’t draw the fill color in the box. Before calling the Box.draw\_action we need to set the fill color for turtle, then begin\_fill. After calling Box.draw\_action you just need to call end\_fill() for turtle.

Test your code in the shell.

## Creating a Circle class

Create a Circle class that inherits from the Point class. The init parameters will be x1, y1, radius and color. Make sure you call the init for the super class. You’ll have to save the radius.

In the draw\_action called turtle.circle() and pass it the radius to draw a circle.

## Creating a CircleFilled class

Create a CircleFilled class that inherits from the Circle class. The init parameters will be x1, y1, radius, color and fill color. Use the same tactics we learned from earlier to only save the new fill color in the instance and test your results.

In the draw action, you should call the Circle draw\_action method like we did with BoxFilled, but call fillcolor, begin\_fill first and then end\_fill afterwards.

Turn in your solution on Canvas.