\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CSC121 PYTHON Programming**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lesson 09 Objects and Their Use

# **Objectives**

In this lesson, students will learn:

- What objects are

- How to use objects in a program

- How to invoke an object’s methods

- How to use Turtle Graphics

# **9.1 Overview**

Programmers always look for a better way to build large programs. Just like many complex machines, large programs are built by putting multiple parts together. In Lessons 07 and 08 we learned how to design and write programs with multiple functions. In programming, that approach is called **procedural programming**, in which procedures, also known as routines or functions, are the building blocks for a large program. When we use procedural programming to design a program, we think about what tasks the program is expected to perform and write functions to complete these tasks. During the execution of the program, functions are called to run at the moment they are needed to perform the tasks they are designed to do.

Procedural programming was the dominant programming paradigm in the 1970s and 80s. Gradually it was replaced by a new programming paradigm, object oriented program, because procedural programming has some limitations. First of all, procedural programming concentrates on steps, but does little in organizing data. A function is a series of steps. Breaking up a program into multiple functions is just breaking up a large number of steps into multiple groups. It does not do much in organizing variables and data.

**Object oriented programming** was developed in the 1980s and has replaced procedural programming as the dominant programming paradigm in the software industry. Similar to procedural programming, the fundamental principle of object oriented programming is still building large programs by parts. However, the building block in object oriented programming is **objects**, which is bigger than functions. An object is a collection of variables, and the functions that are designed to handle these variables. Variables belonged to an object are the **attributes** of the object. Functions belonged to an object are considered as controlling the **behavior** of the object. By using objects rather than functions as building blocks, we are using larger parts to build a program. This makes it possible to create larger and more complex programs. Many of the most widely used programming languages, including Java, C++, C#, Python, PHP, Ruby, Perl, Delphi, Objective-C, Swift, Common Lisp, and Smalltalk, all support object-oriented programming.

The details of object oriented programing are discussed in Chapter 10 of the textbook and will be covered in later lessons. This lesson, which is based on textbook Chapter 6, will only introduce the concept of object briefly and concentrate on using objects in programs.

# **9.2 Objects**

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which contain data and functions. Many data of an object are stored in **instance variables**, while functions that belong to an object are often known as **methods**. Most methods are designed to handle the object’s data, including storing data in instance variables, retrieving data from them and using them for calculations and other types of processing.

Let’s look at an example. Suppose we need a program to handle the grades of the students in a course. Each student has a few data items: name, project score, midterm score and final score. To keep things simple, let’s say there are only 3 students in the course. Since there are 3 students and each student has 4 data items, altogether we need 12 variables to store all the data. If we do not use object oriented programming, we need to create 12 variables somewhere in the program, probably in the main function, to store the data. These 12 variables may not be organized in any way, except that we may name them systematically.

If object oriented programming is used, we will create 3 student objects, and each student object will have four instance variables: name, midterm, final and project. This naturally groups the variables by their owner (i.e. which student the variables belong to). In addition, let’s say these objects have two methods: the calc\_avg method that calculates and returns the average of midterm, final and project scores, and the display\_info method that displays name, midterm score, final score and project score. These methods are tied to the variables they are designed to handle because both the methods and the variables belong to the student objects.

Since the focus of this lesson is on using objects, you will not see how to define objects until a later lesson. However, we will show you the syntax of accessing the instance variables and invoking the methods.

To access an instance variable of an object, we use a dot notation system: the object’s name first, followed by a dot, and then the variable’s name. Suppose we have created three student objects named student1, student2 and student3. The following statement stores 85 in the instance variable project of student2:

student2.project = 85

Similarly, to store 91 in midterm score of student3, we write:

student3.midterm = 91

To display the final exam score of student1, we write:

print(student1.final)

To calculate student2’s average score, we write:

avg = (student2.midterm + student2.final + student2.project)/3

The dot notation is also used in invoking an object’s methods. For example, to invoke the calc\_avg method of student1, which calculates and returns the average score for student1, we write:

avg = student1.calc\_avg()

Similarly, we can use the following syntax to invoke the display\_info method of student1:

student1.display\_info()

In summary, whenever you need to access an instance variable of an object or invoke a method of an object, use the dot notation.

# **9.3 Turtle Graphics**

The textbook uses Turtle Graphics as an example to show how to use objects in a program. Turtle Graphics is a popular way for introducing programming to kids. It was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966. With Turtle Graphics, a program can create a graphics window and draw various graphical designs. Python provides the capability of Turtle Graphics in the turtle Python standard library module.

In Turtle Graphics, graphical designs are drawn by moving a “turtle” inside the graphics window. The paths the turtle travels can turn into lines. With careful planning, these lines can form beautiful graphical designs. The turtle is similar to a mouse pointer, except that while you can change the position of the mouse pointer by moving the mouse by hand, you must use program instructions to change the position of the turtle.

A turtle is an object in a Python program. It has variables for its attributes (e.g. x-coordinate, y-coordinate, direction, visibility, etc.), and it has a set of methods to control its movement. Turtle graphics is introduced here as an example to show how to use objects and its methods in a program.

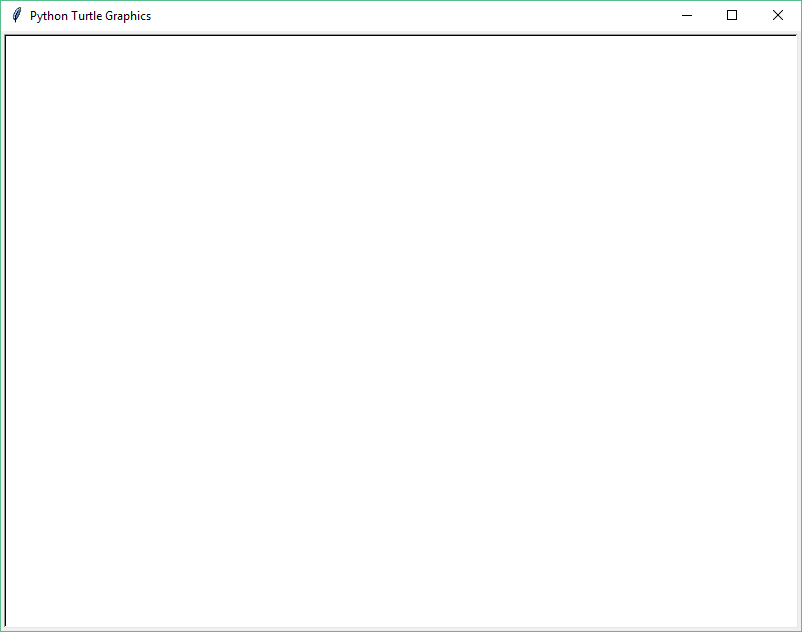
## **9.3.1 Creating a Turtle Graphics Window**

First of all, we need to create and show a Turtle Graphics window. This can be done by the following Python statements:

**import** turtle  
turtle.setup(800, 600)  
  
*# insert statements here to create and use a turtle*turtle.exitonclick()

The first statement imports the Turtle Graphics module. We will discuss what modules are and how to use them in a later lesson.

The second statement creates an 800 by 600 graphics window. You should see the following window when you run the program above:



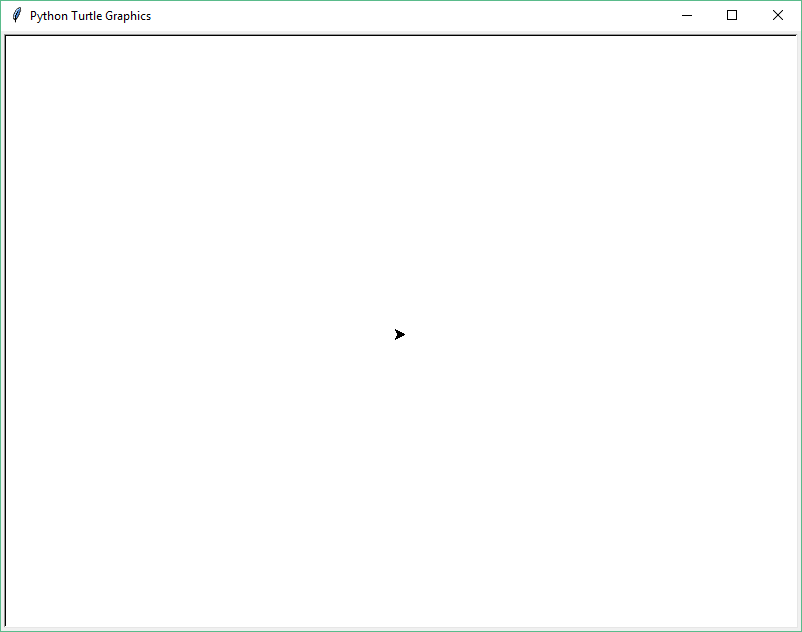
The size of this window is 800 pixels horizontally by 600 pixels vertically. Figure 6-18 on page 217 of the textbook shows the coordinate system of this graphics window. The coordinates of the center is (0, 0). The upper left corner is (-400, 300), upper right corner is (400, 300), lower left corner is (-400, -300), lower right corner is (400, -300). This window will remain open until you click anywhere in it. The last statement in the program above closes the window whenever it is clicked.

## **9.3.2 Creating a Turtle Object**

Let’s add a statement to the program above to create a Turtle object.

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle() *# create turtle object*turtle.exitonclick()

If you run the program, you will find an arrowhead added at the center of the graphics window. This arrow head represents a turtle. Its default position is (0, 0), i.e., the center of the window.



## **9.3.3 Moving Turtle Object to a New Position**

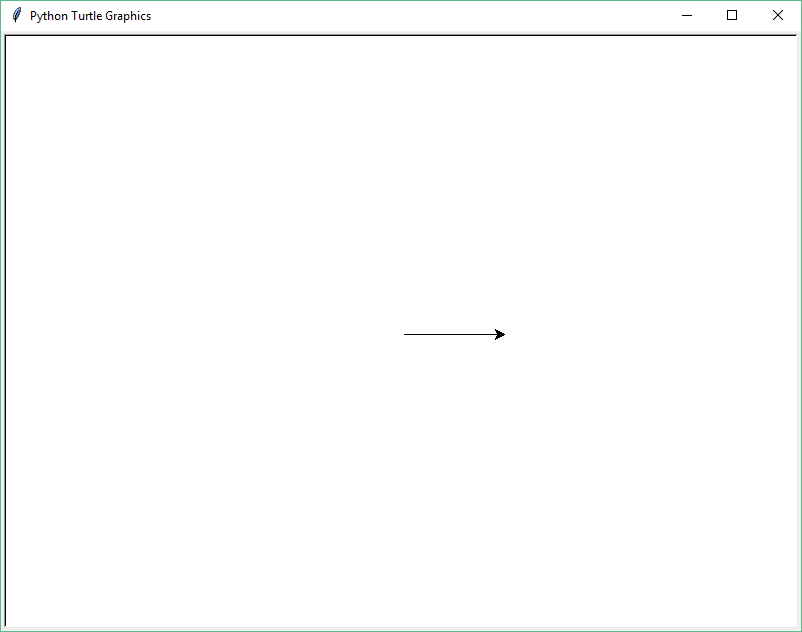
Let’s add another statement to draw a line.

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle() *# create Turtle object*turtle1.setposition(100, 0) *# move turtle1 to 100, 0*turtle.exitonclick()

The statement

turtle1.setposition(100, 0)

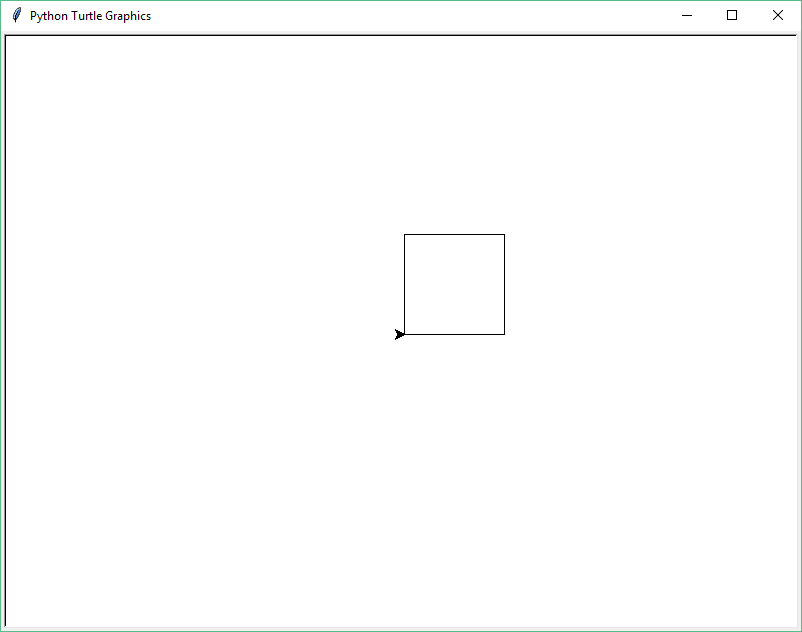
invokes the setposition method of turtle1. This method changes the x and y coordinates of turtle1, and the Turtle graphics system redraws the arrowhead at the new location. It also draws a line to connect the old location (0, 0) and new location (100, 0).



Let’s add three more statements to call the setposition method three more times.

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
turtle1.setposition(100, 0)  
turtle1.setposition(100, 100)  
turtle1.setposition(0, 100)  
turtle1.setposition(0, 0)  
turtle.exitonclick()

This program draws four lines which form a square:

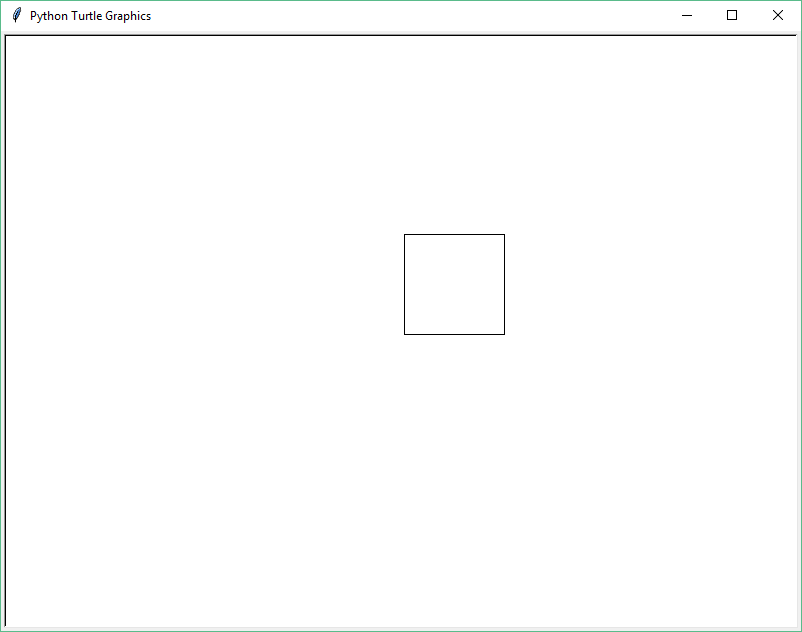


## **9.3.4 Hiding the Arrowhead**

We can also add a statement to call the hideturtle method of turtle1, which hides the arrowhead, if we want to.

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
turtle1.setposition(100, 0)  
turtle1.setposition(100, 100)  
turtle1.setposition(0, 100)  
turtle1.setposition(0, 0)  
turtle1.hideturtle()  
turtle.exitonclick()

The turtle icon is hidden after the square is drawn:



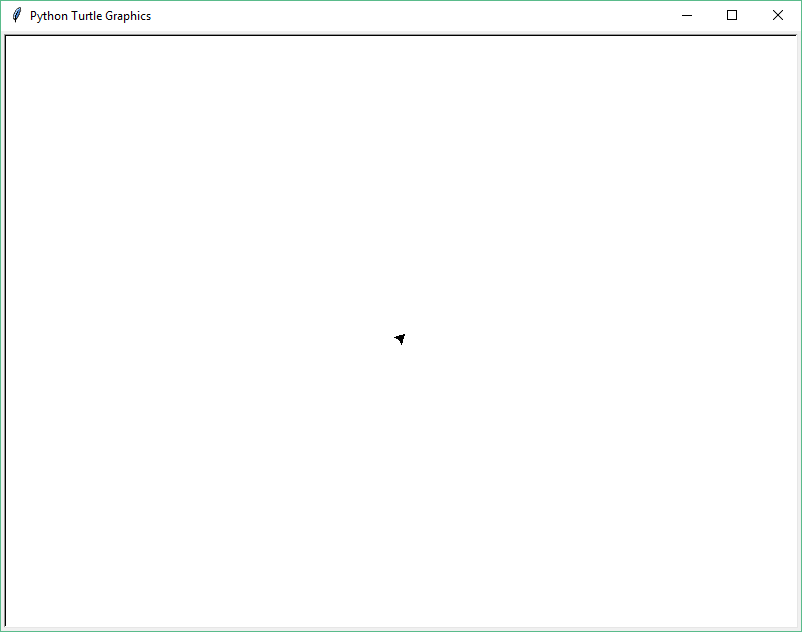
So far we have seen two methods of turtle objects: setposition and hideturtle. In fact, turtle objects have many more methods. Let’s look at a few more.

## **9.3.5 Changing Direction of the Arrowhead**

The left method turns the arrowhead counter-clockwise by a specified angle. For example, turtle1.left(45) turns the icon counter-clockwise by 45 degree.

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
turtle1.left(45)  
turtle.exitonclick()

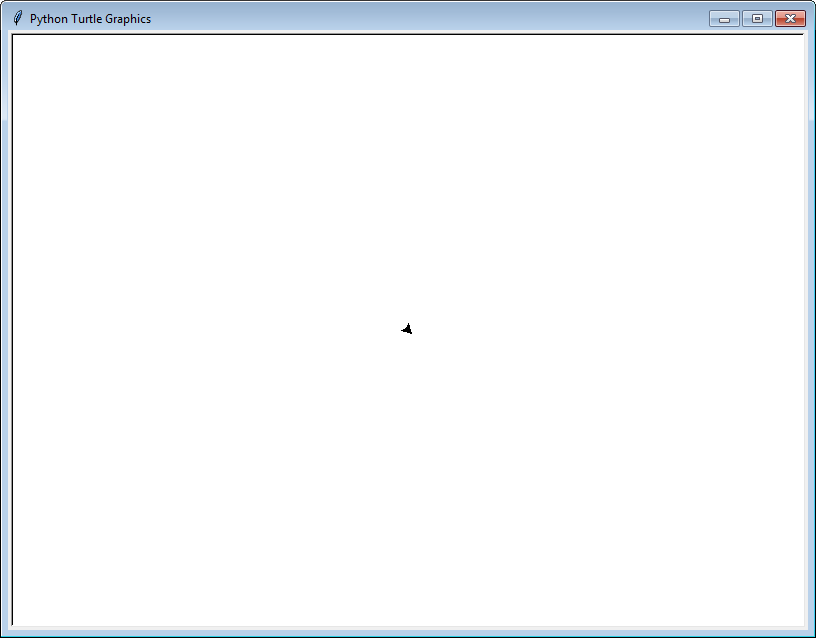
Graphics window:



Similarly, turtle1.right(45) turns the arrowhead clockwise by 45 degree.

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
turtle1.right(45)  
turtle.exitonclick()

Graphics window:

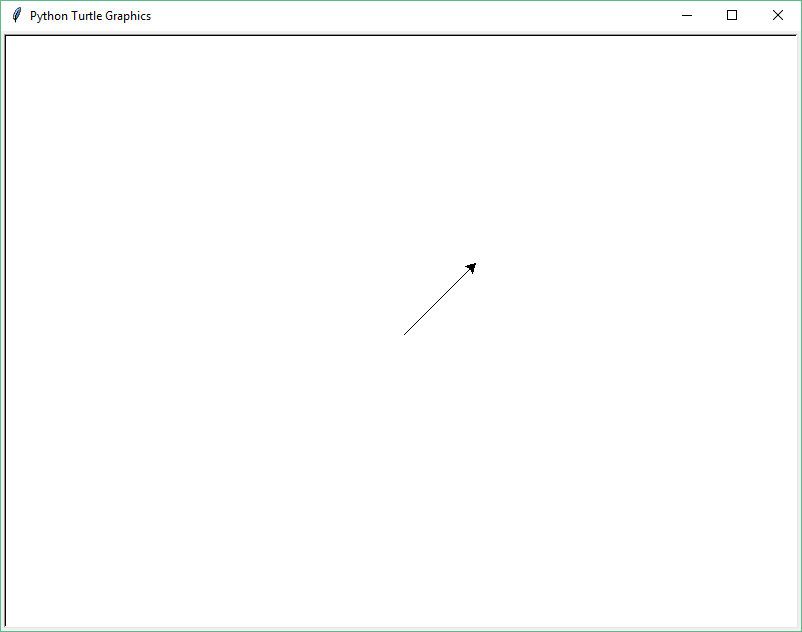


## **9.3.6 Moving the Arrowhead Forward and Backward**

The forward method moves the arrowhead forward. For example, turtle1.forward(100) moves the arrowhead 100 pixels forward.

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
turtle1.left(45)  
turtle1.forward(100)  
turtle.exitonclick()

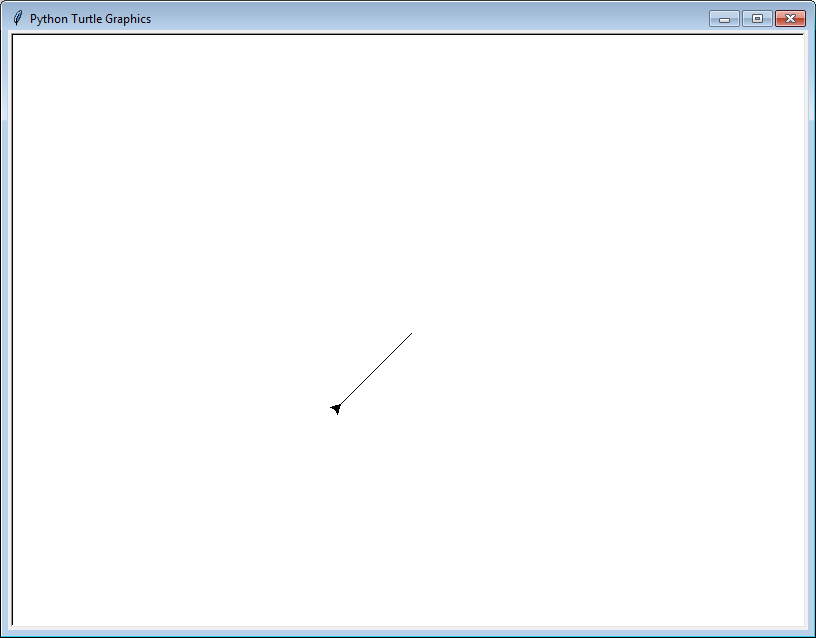
Graphics window:



Similarly, turtle1.backward(100) will move the turtle icon 100 pixels backward.

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
turtle1.left(45)  
turtle1.backward(100)  
turtle.exitonclick()

Graphics window:

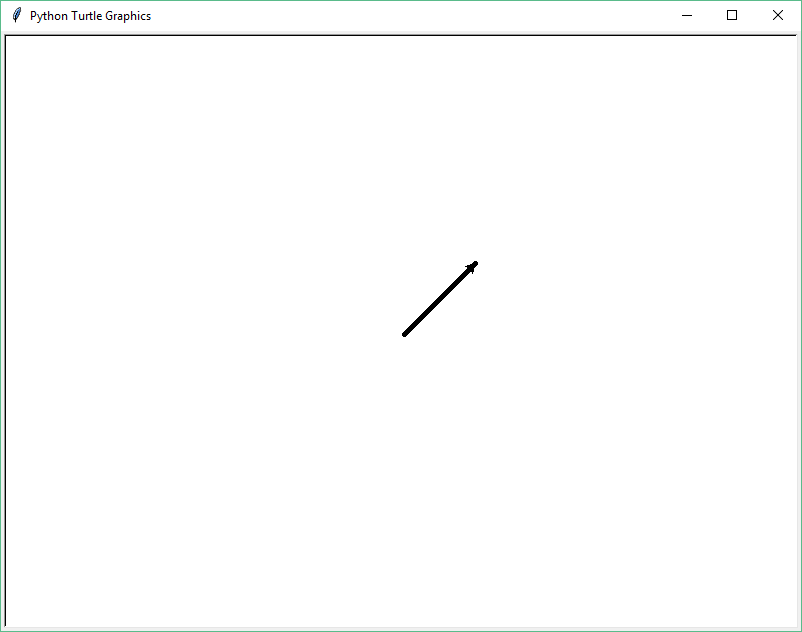


## **9.3.7 Controlling the Pen**

When lines are drawn, the default pen size of 1 is used. We can control pen size by using pensize method. For example, turtle1.pensize(5) changes pen size to 5.

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
turtle1.pensize(5)  
turtle1.left(45)  
turtle1.forward(100)  
turtle.exitonclick()

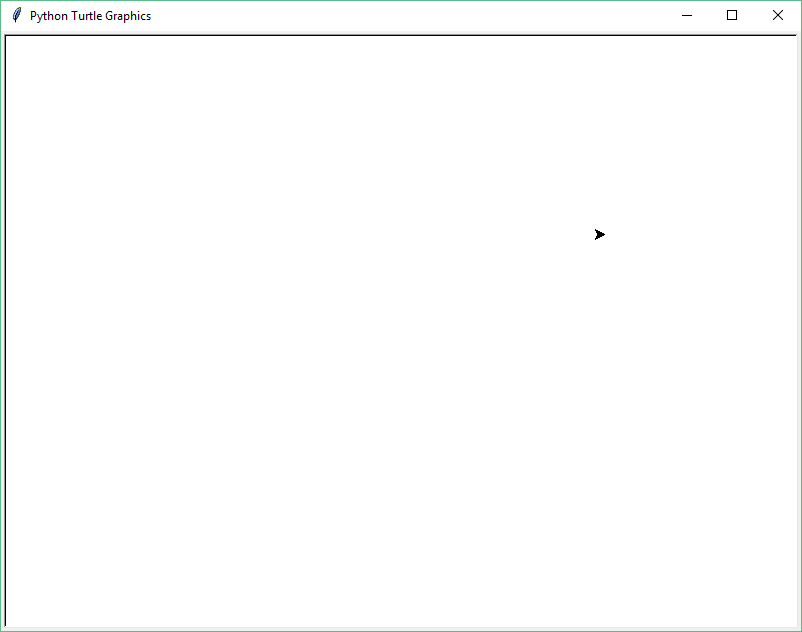
Graphics window:



If you want to move the arrowhead to a new position without drawing a line, you can use the penup method to lift the pen so it will not draw. Example:

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
turtle1.penup()  
turtle1.setposition(200, 100)  
turtle.exitonclick()

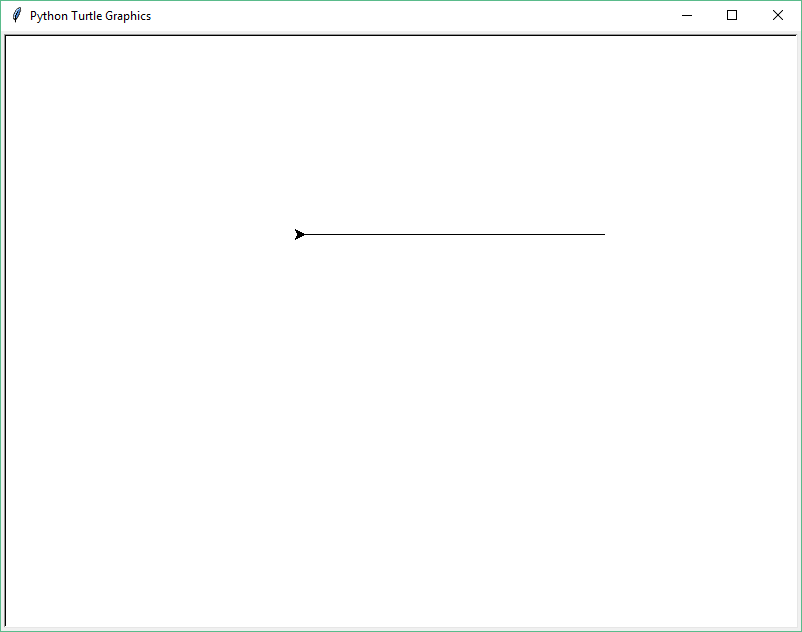
The program above moves the arrowhead from (0,0) to (200, 100) without drawing a line. Graphics window:



You can use the pendown method to change the pen from “up” to “down”. Example:

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
turtle1.penup()  
turtle1.setposition(200, 100)  
turtle1.pendown()  
turtle1.setposition(-100, 100)  
turtle.exitonclick()

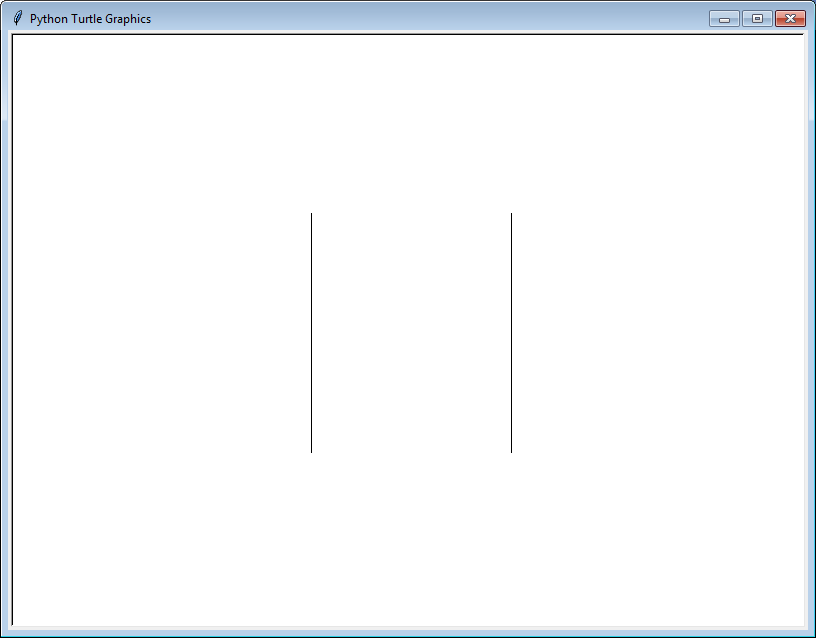
Graphics window:



Let’s look at one more example about the penup and pendown methods. The following program draws two vertical lines:

**import** turtle  
turtle.setup(800, 600)  
turtle1 = turtle.Turtle()  
  
*# draw first vertical line*turtle1.penup()  
turtle1.setposition(-100, 120)  
turtle1.pendown()  
turtle1.setposition(-100, -120)  
  
*# draw second vertical line*turtle1.penup()  
turtle1.setposition(100, 120)  
turtle1.pendown()  
turtle1.setposition(100, -120)  
  
turtle1.hideturtle()  
turtle.exitonclick()

The program above first draws a vertical line at x = -100. The y coordinates of the two endpoints of this line are 120 and -120, respectively. It then draws a second vertical line at x = 100. The y coordinates of the two endpoints of this line are 120 and -120, respectively. Graphics window:

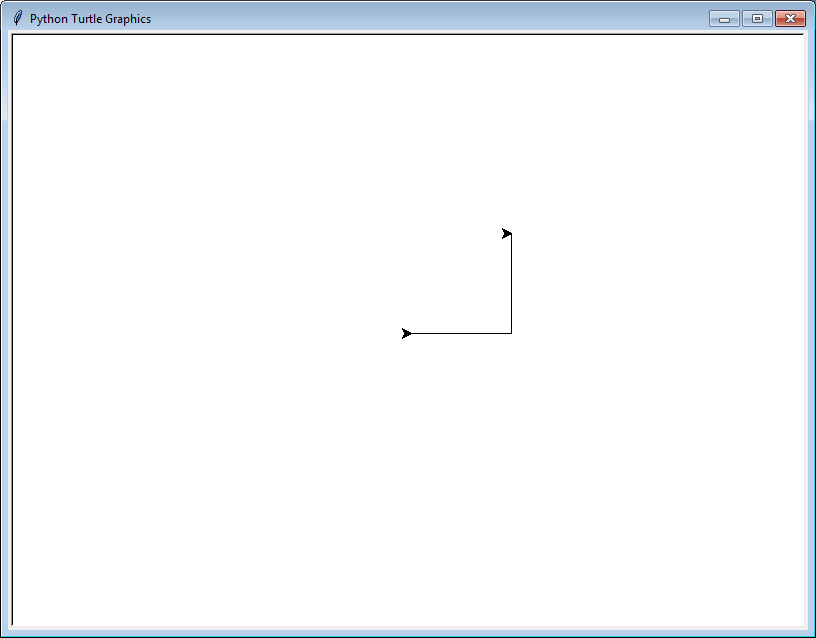


## **9.3.8 Creating Multiple Turtle Objects**

You can create multiple Turtle objects. In the following program, two Turtle objects are created.

import turtle  
turtle.setup (800, 600)  
turtle1 = turtle.Turtle()  
turtle1.setposition (100, 0)  
turtle2 = turtle.Turtle()  
turtle2.penup()  
turtle2.setposition (100, 0)  
turtle2.pendown()  
turtle2.setposition (100, 100)  
turtle.exitonclick()

Two Turtle objects named turtle1 and turtle2 are created. The setposition method of turtle1 is called to move its arrowhead to the position (100, 0). Similarly, the setposition method of turtle2 is called to move its arrowhead to the position (100, 100). The following is the graphics window:

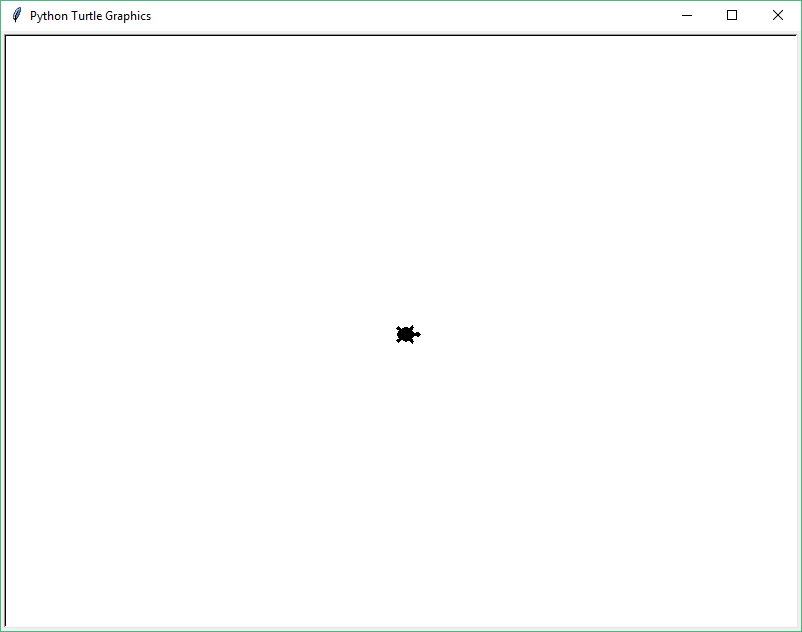


## **9.3.9 Changing Turtle Shape**

You can use the shape method to change the shape from the default arrowhead to something else. Figure 6-25 of the textbook shows available turtle shape. For example, the following program changes the shape of the turtle to “turtle”.

**import** turtle  
turtle.setup(800, 600)  
  
turtle1 = turtle.Turtle()  
turtle1.shape(**"turtle"**)  
  
turtle.exitonclick()

Graphics window:



## **9.3.10 Integrated Example**

Let use a few things we have learned in a new example. We are going to draw a “wall” horizontally. A turtle will walk toward the wall. When it hits the wall, it will turn around and walk away. The following is the Python code.

**import** turtle  
turtle.setup(800, 600)

*# set coordinates of the wall*  
left = -300  
right = 300  
top = 200  
  
turtle1 = turtle.Turtle()  
turtle1.hideturtle()  
  
*# draw a horizontal wall*turtle1.penup()  
turtle1.setposition(left, top)  
turtle1.pendown()  
turtle1.setposition(right, top)  
  
*# create a turtle*turtle2 = turtle.Turtle()  
turtle2.shape(**"turtle"**)  
turtle2.penup()  
turtle2.left(90)  
  
*# walk upward, turn around when hitting the wall***for** i **in** range(400):  
 turtle2.forward(1)  
 **if** turtle2.ycor() >= top:  
 turtle2.left(180)  
  
turtle.exitonclick()

The first part of the program draws a wall. First we move the arrowhead to (-300, 200) with the pen up. Then we draw a line by changing the position to (300, 200) with the pen down.

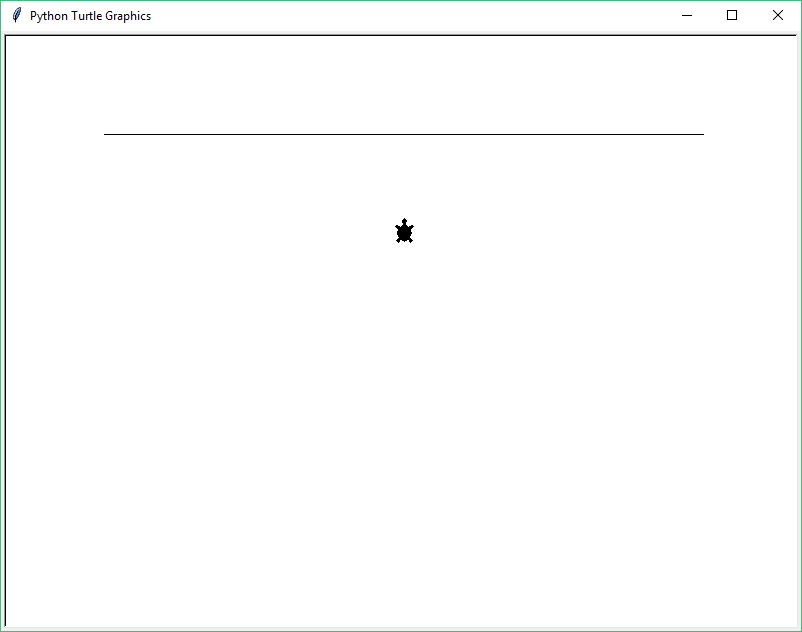
The second part of the program creates a Turtle object and changes the shape to “turtle”. We turn the turtle toward the wall by turning it 90 degree counter clockwise.

The third part of the program creates some sort of animation. We use a loop and the forward function to make the turtle walk 400 steps. When it hits the wall, it turns a round (i.e. turn 180 degree) and continue to walk.

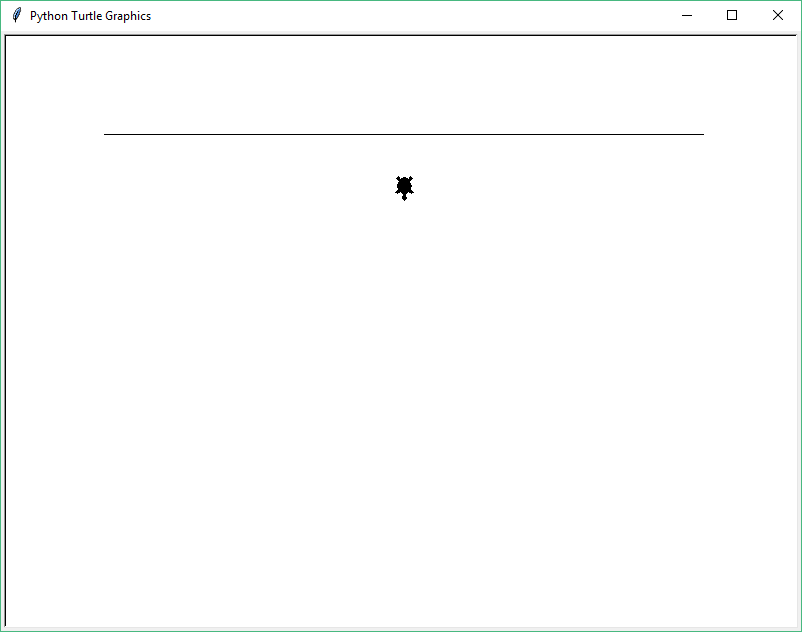
How do we check whether the turtle hits the wall? We check the y-coordinate of the turtle to see whether it is larger than the y-coordinate of the wall. Each turtle object has two methods that return its x and y coordinates: the xcor method and the ycor method. The following condition tests whether the turtle hit the wall:

**if** turtle2.ycor() >= top:

The following screenshot shows the turtle walking towards the wall:



The following screenshot shows the turtle walking away from the wall:



In summary, Turtle graphics is used to show how to create objects in a program and how to use the dot notation to invoke methods of an object. Turtle objects have a lot more methods to control color, size, visibility, shape and speed. Please read section 6.2 of the textbook chapter 6.

# **9.4 Further Reading**

Please read chapter 6 of the textbook. Section 6.1 introduces software objects, while section 6.2 introduces Turtle Graphics.