Vahé Karamian

Python Programming

CS-110

CHAPTER 4

Objects and Graphics

OBJECTIVES

- To understand the concepts of objects and how they can be used to simplify programming.
- To become familiar with the various objects available in the graphics library.
- To be able to create objects in programs and call appropriate methods to perform graphical computations.

OBJECTIVES

- To understand the fundamental concepts of computer graphics, especially the role of coordinate systems and coordinate transformations.
- To understand how to work with both mouse- and text-based input in a graphical programming context.
- To be able to write simple interactive graphics programs using the graphics library.

OVERVIEW

- Each data type can represent a certain set of values, and each has a set of associated operations.
- The traditional programming view is that data is passive. It’s manipulated and combined with active operations.

OVERVIEW

- Modern computer programs are built using an object-oriented approach.
- Most applications you're familiar with have Graphical User Interfaces (GUI) that provide windows, icons, buttons and menus.
- There is a graphics library (graphics.py) written specifically to go with this book. It's based on Tkinter.

THE OBJECT OF OBJECTS

- Basic Idea – View a complex system as the interactions of simpler objects. An object is a sort of active data type that combines data and operations.
- Objects *know stuff* (contain data) and they can *do stuff* (have operations).
- Objects interact by sending each other *messages*.

THE OBJECT OF OBJECTS

- Suppose we want to develop a data processing system for a college or university.
- We must keep records on students who attend the school. Each student will be represented as an object.

THE OBJECT OF OBJECTS

- The Student object would contain data like:
  - Name
  - ID Number
  - Courses Taken
  - Campus Address
  - Home Address
  - GPA
  - Etc…

THE OBJECT OF OBJECTS

- The Student object should also respond to requests.
- We may want to send out a campus-wide mailing, so we'd need a campus address for each student.
- We could send the *printCampusAddress* to each student object. When the student object receives the message, it prints its own address.

THE OBJECT OF OBJECTS

- Objects may refer to other objects.
- Each course might be represented by an object:
  - Instructor
  - Student Roster
  - Prerequisite Courses
  - When and where the class meets.

THE OBJECT OF OBJECTS

- Sample Operations
  - addStudent
  - delStudent
  - changeRoom
  - Etc…

SIMPLE GRAPHICS PROGRAMMING

- For this course, we will be using a *graphics.py* library supplied with the book.
- How to install it
  - First get a copy of the file graphics.py
  - Place the file in the following directory:
    - Python's Lib directory
      - C:\Python31\Lib\site-packages
    - Or same folder as your graphics program

SIMPLE GRAPHICS PROGRAMMING

- Since this is a library, we need to import the graphics commands

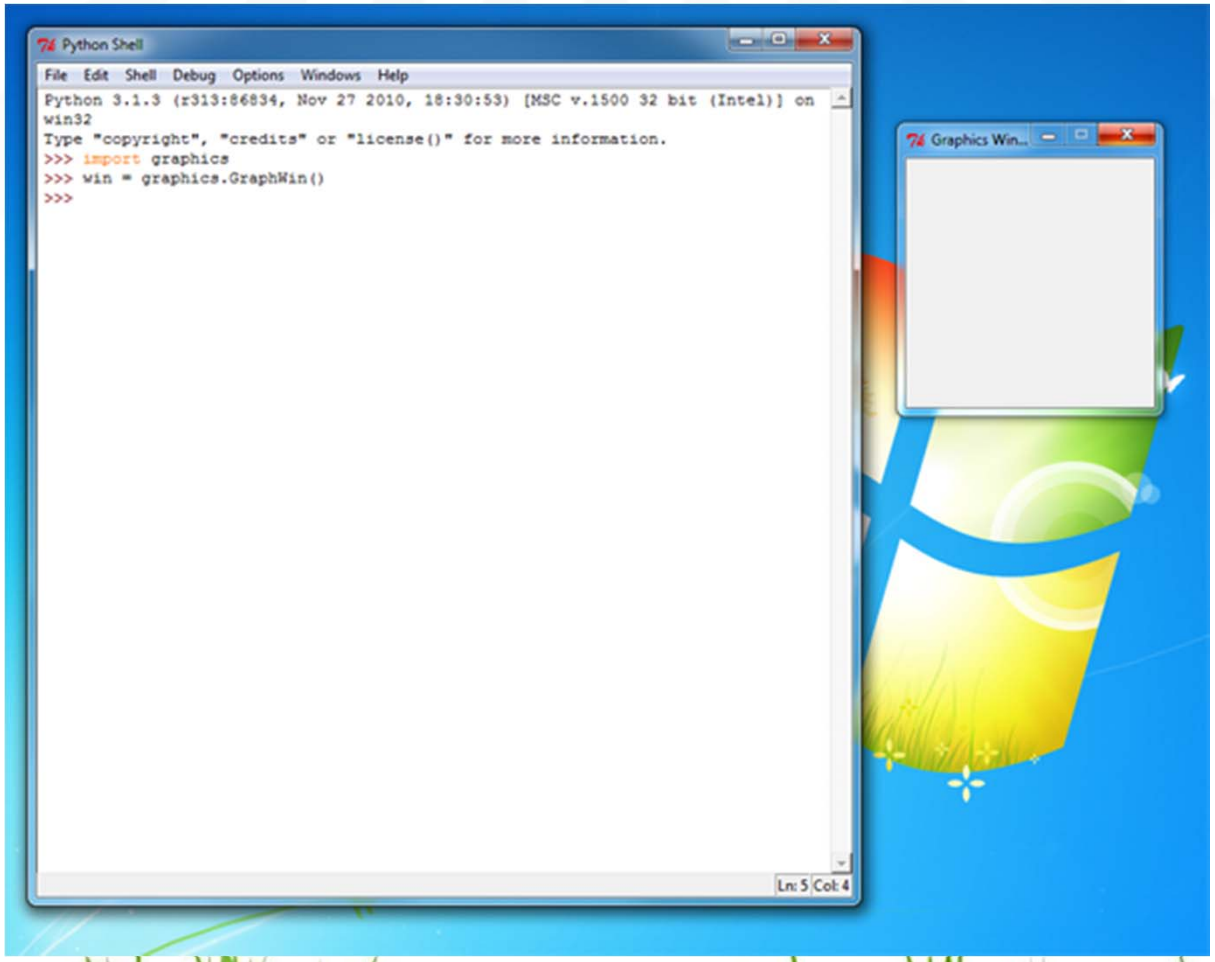
  >>>import graphics

- A graphics window is a place on the screen where the graphics will appear.

  >>>win = graphics.GraphWin()

- This command creates a new window titled "Graphics Window"

SIMPLE GRAPHICS PROGRAMMING



SIMPLE GRAPHICS PROGRAMMING

- *GraphWin* is an object assigned to the variable *win*. We can manipulate the window object through this variable.
- Windows can be closed/destroyed by issuing the command.

  ```
  >>> win.close()
  ```

- It's tedious to use the *graphics.* notation to access the graphics library routines.

  ```
  >>> from graphics import *
  ```

  – The "from" statement allows you to load specific functions from a library module. "*" will load all the functions, or you can list specific ones.

SIMPLE GRAPHICS PROGRAMMING

- Doing the import this way eliminates the need to preface graphics commands with graphics.

```
>>> from graphics import *
>>> win = GraphWin()
```

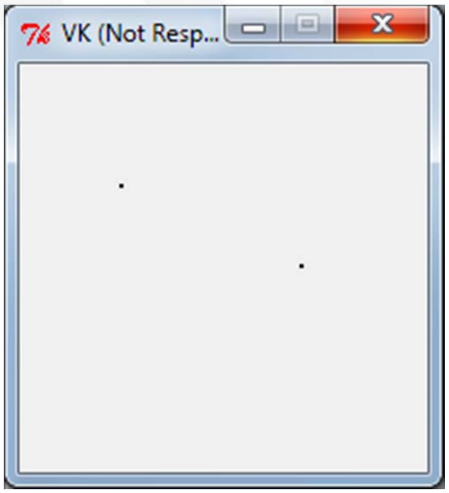
- A graphics window is a collection of points called pixels (picture elements)
- The default GraphWin is 200 pixels tall by 200 pixels wide (40,000 pixels total)
- One way to get pictures into the window is one pixel at a time, which would be tedious.
- The graphics object has a number of predefined routines to draw geometric shapes.

SIMPLE GRAPHICS PROGRAMMING

- The simplest object is the *Point*.
- Like points in geometry, point locations are represented with a coordinate system *(x, y)*; where x is the horizontal location and y is the vertical location.
- The origin *(0, 0)* in a graphics window is the upper left corner.
- x values increase from left to right, y values from top to bottom.

SIMPLE GRAPHICS PROGRAMMING

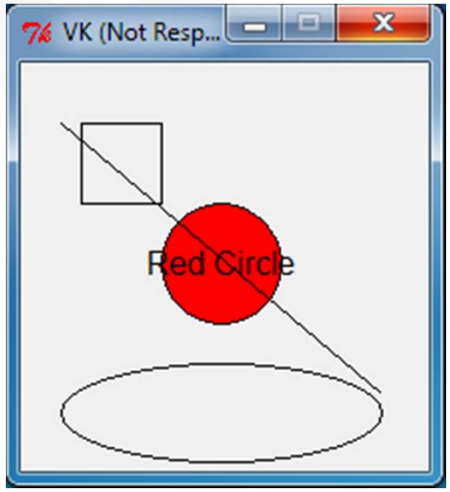
Source Code	Output

>>> from graphics import *
>>> p1 = Point(50, 60)
>>> p2 = Point(140, 100)
>>> p1.getX()
50
>>> p1.getY()
60
>>> win = GraphWin()
>>> p1.draw(win)
>>> p2.draw(win)
>>> win.close()
	

SIMPLE GRAPHICS PROGRAMMING

Source Code	Output

```
>>> center = Point(100, 100)
>>> circle = Circle(center, 30)
>>> circle.setFill('red')
>>> win = GraphWin("VK", 200, 200)
>>> circle.draw(win)
>>> label = Text(center, "Red Circle")
>>> label.draw(win)
>>> rect = Rectangle(Point(30,30),
Point(70,70))
>>> rect.draw(win)
>>> line = Line(Point(20,30),
Point(180,165))
>>> line.draw(win)
>>> oval = Oval(Point(20,150),
Point(180,199))
>>> oval.draw(win)
```



USING GRAPHICAL OBJECTS

- Computation is performed by asking an object to carry out one of its operations.
- In the previous example we manipulated
  - GraphWin
  - Point
  - Circle
  - Oval
  - Line
  - Text
  - Rectangle.
- These are examples of classes.

USING GRAPHICAL OBJECTS

- Each object is an *instance* of some class, and the *class* describes the properties of the instance.
- If we say BMW is a car, we are actually saying that BMW is a specific car manufacturer in a larger class of all cars.
- BMW is an instance of the car class.

USING GRAPHICAL OBJECTS

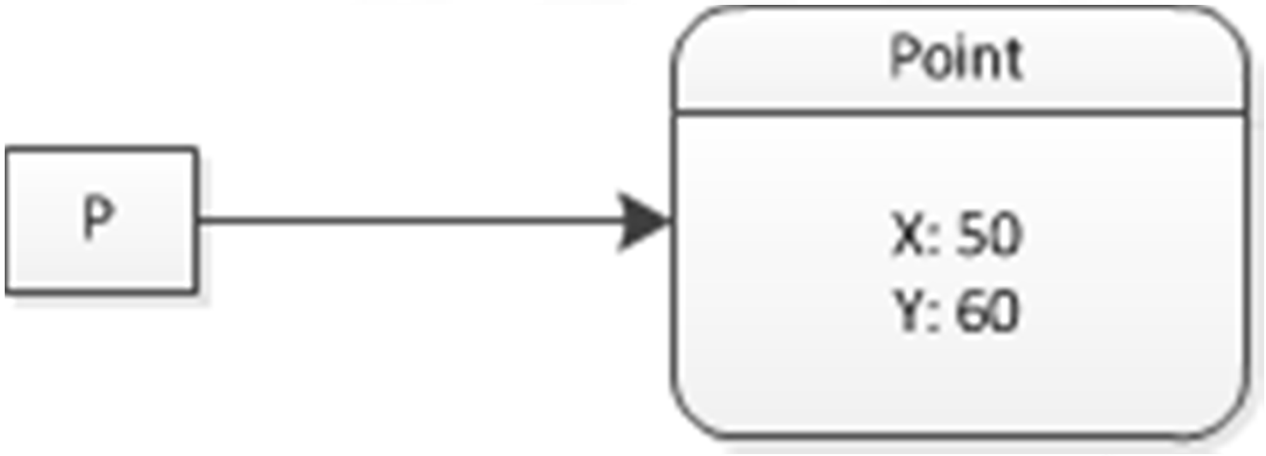
- To create a new instance of a class, we use a special operation called a constructor.
  - <class name>(<param1>,<param2>,…)
- <class name> is the name of the class we want to create a new instance of, e.g., Circle or Point.
- The parameters are required to initialize the object. For example, Point requires two numeric values.

USING GRAPHICAL OBJECTS

- P = Point(50,60)
- The constructor for the Point class requires two parameters, the x and y coordinates for the point.
- These values are stored as instance variables inside of the objects.

USING GRAPHICAL OBJECTS

- Only the most relevant instance variables are shown (others include the color, window they belong to, etc…)



USING GRAPHICAL OBJECTS

- To perform an operation on an object, we send the object a message. The set of messages an object responds to are called the *methods* of the object.
- Methods are like functions that live inside the object.
- Methods are invoked using dot-notation:

  <object>.<method-name>(<param1>,<param2>,…)

USING GRAPHICAL OBJECTS

- P.getX() and P.getY() return the x and y values of the point.
- Routines like these are referred to as accessors because they allow us to access information from the instance variables of the object.

USING GRAPHICAL OBJECTS

- Other methods change the *state* of the object by changing the values of the object's instance variable.
- *move(dx, dy)* moves the object *dx* units in the x direction and *dy* units in the y direction.
- Move also erases the old image and draw it in its new position. Methods that change the state of an object are called *mutators*.

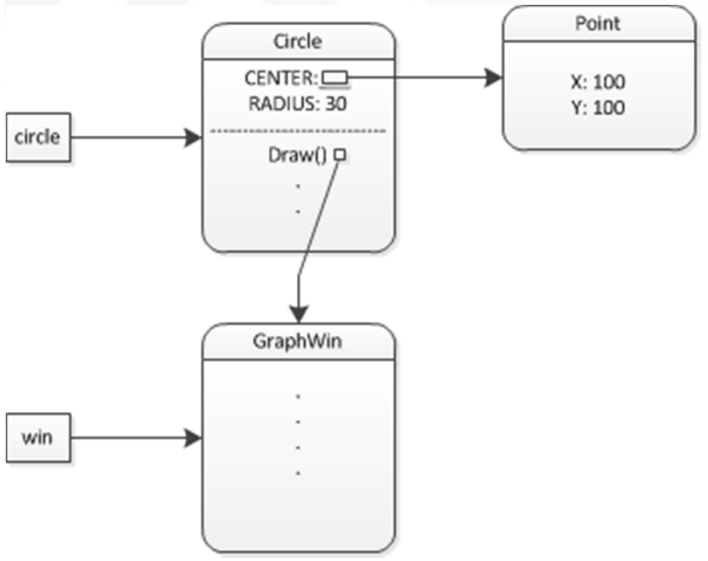
USING GRAPHICAL OBJECTS

```
>>>circle = Circle(Point(100,100), 30)
>>>win = GraphWin()
>>>circle.draw(win)
```

- The first line creates a circle with radius 30 centered at (100,100)
- We use the Point constructor to create a location for the center of the circle.
- The last line is a request to the Circle object circle to draw itself into the GraphWin object win.

USING GRAPHICAL OBJECTS

The draw method uses information about the center and radius of the circle from the instance variable.



USING GRAPHICAL OBJECTS

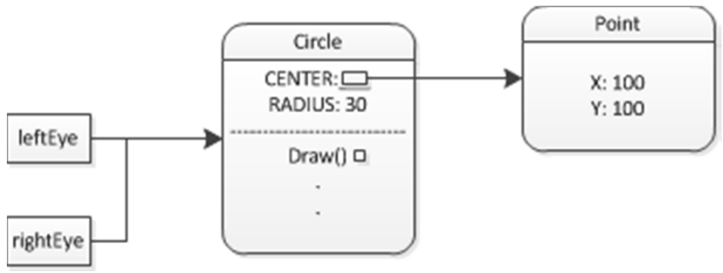
- It's possible for two different variables to refer to the same object.
- Changes made to the object through one variable will be visible to the other!

```
>>>leftEye = Circle(Ponit(80,50), 5)
>>>leftEye.setFill('yellow')
>>>leftEye.setOutline('red')
>>>rightEye = leftEye
>>>rightEye.move(20, 0)
```

- The idea is to create the left eye and copy that to the right eye which gets moved 20 units.

USING GRAPHICAL OBJECTS

- The assignment rightEye = leftEye makes rightEye and leftEye refer to the same circle!
- The situation where two variables refer to the same object is called *aliasing*.



USING GRAPHICAL OBJECTS

- There are two ways to get around this.
- We could make two separate circles, one for each eye:

```
>>>leftEye = Circle(Point(80,50), 5)
>>>leftEye.setFill('yellow')
>>>leftEye.setOutline('red')
>>>rightEye = Circle(Point(100,50), 5)
>>>rightEye.setFill('yellow')
>>>rightEye.setOutline('red')
```

USING GRAPHICAL OBJECTS

- The graphics library has a better solution.
- Graphical objects have a clone method that will make a copy of the object!

```
>>>leftEye = Circle(Point(80,50), 5)
>>>leftEye.setFill('yellow')
>>>leftEye.setOutline('red')
>>>rightEye = leftEye.clone()
>>>rightEye.move(20, 0)
```

INTERACTIVE GRAPHICS

- In a GUI environment, users typically interact with their applications by clicking a button, choosing items from menus, and typing information into on-screen text boxes.
- Event-driven programming draws interface elements (widgets) on the screen and then waits for the user to do something.

INTERACTIVE GRAPHICS

- An event is generated whenever a user moves the mouse, clicks the mouse, or types a key on the keyboard.
- An event is an object that encapsulates information about what is happening in the system.
- The even object is sent to the appropriate part of the program to be processed, for example, a button event.
- The graphics module hides the underlying, low-level window management and provides two simple ways to get user input in a *GraphWin*.

GETTING MOUSE CLICKS

- We can get graphical information from the user via the *getMouse* method of the *GraphWin* class.
- When *getMouse* is invoked on a GraphWin, the program pauses and waits for the user to click the mouse somewhere in the window.
- The spot where the user clicked is returned as a *Point*.

GETTING MOUSE CLICKS

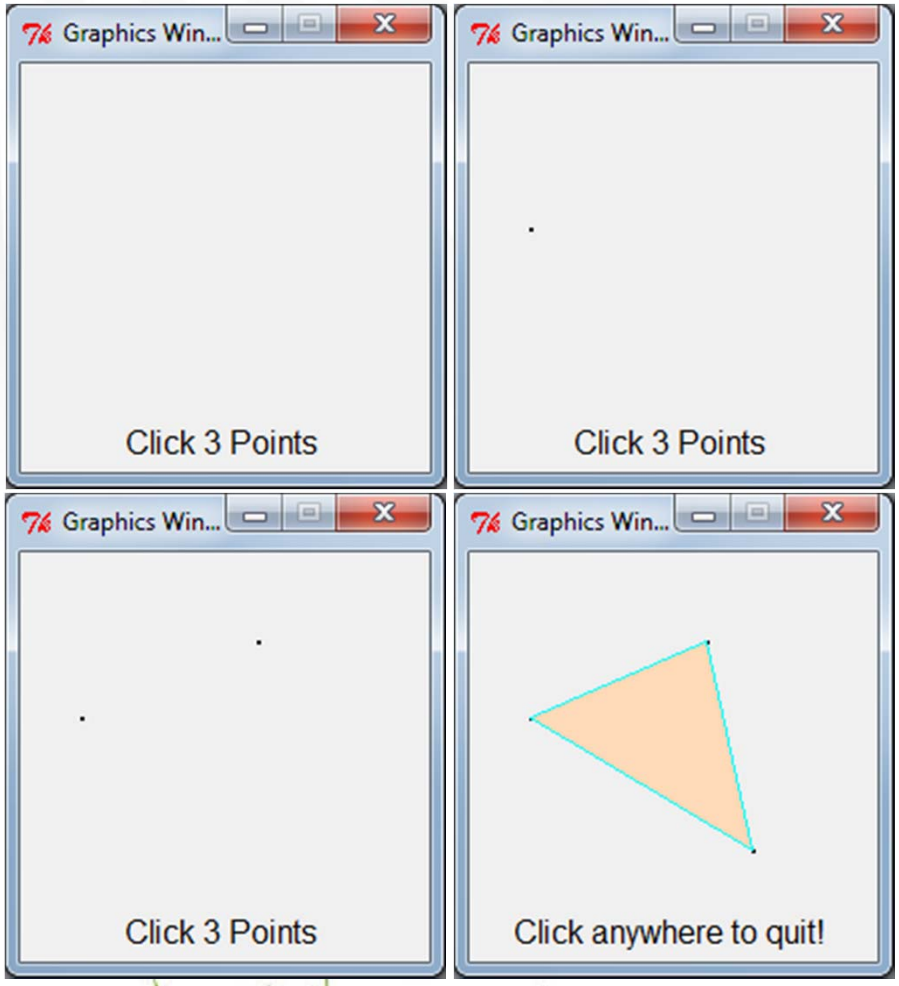
- The following code will return the coordinates of a mouse click:

```
>>>from graphics import *
>>>win = GraphWin()
>>>p = win.getMouse()
>>>print("You clicked (", p.getX(),", ", p.getY(),")")
```

- You can use the accessors like *getX* and *getY* or other methods on the point returned.

LETS BUILD A TRIANGLE

```
>>> from graphics import *
>>> def main():
    win = GraphWin()
    win.setCoords(0.0, 0.0, 10.0, 10.0)
    message = Text(Point(5,0.5), "Click 3 Points")
    message.draw(win)

    p1 = win.getMouse()
    p1.draw(win)
    p2 = win.getMouse()
    p2.draw(win)
    p3 = win.getMouse()
    p3.draw(win)

    triangle = Polygon(p1,p2,p3)
    triangle.setFill("peachpuff")
    triangle.setOutline("cyan")
    triangle.draw(win)

    message.setText("Click anywhere to quit!")
    win.getMouse()
```



LETS BUILD A TRIANGLE

NOTES:

- If you are programming in windows environment, using the .pyw extension on your file will cause the Python shell window to not display when you double click the program icon.
- There is no triangle class! Rather, we use the general polygon class, which takes any number of points and connects them into a closed shape!

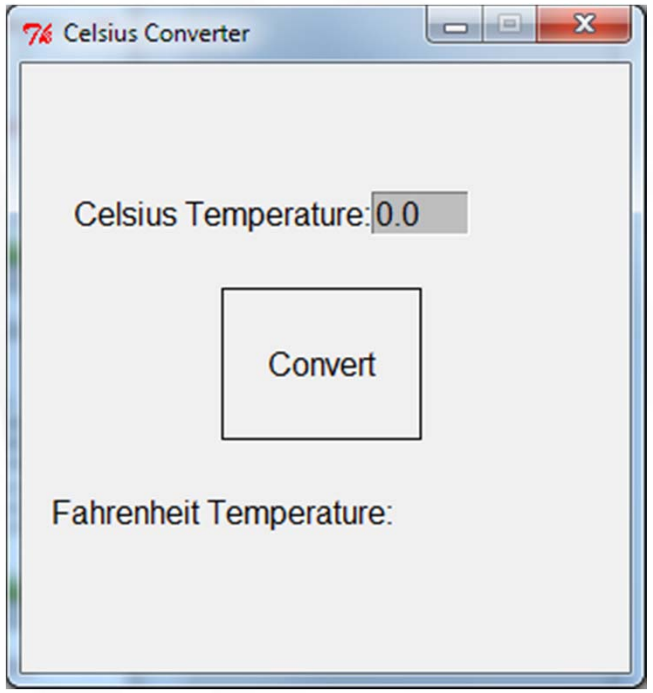
LETS BUILD A TRIANGLE

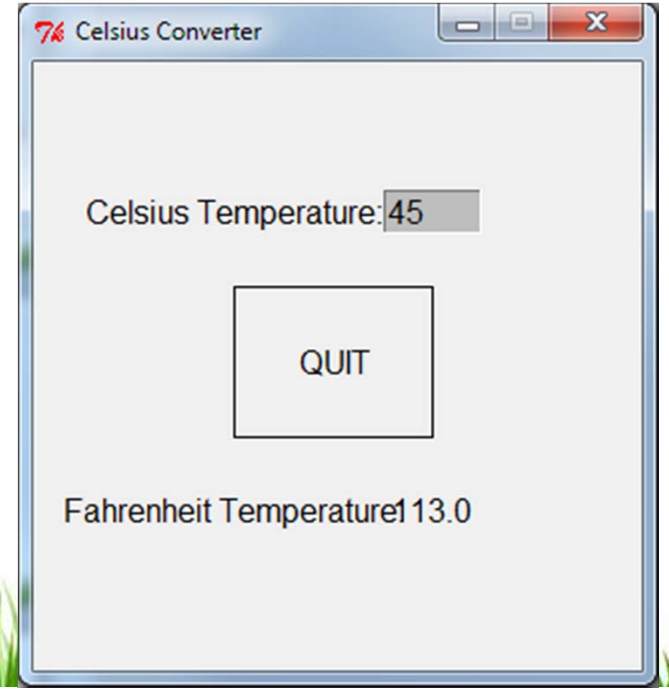
- Once you have three points, creating a triangle polygon is easy:
  - Triangle = Polygon(p1, p2, p3)
- A single text object is created and drawn near the beginning of the program.
  - Message = Text(Point(5,0.5),"Click 3 Points")
- To change the prompt, just change the text to be displayed.
  - Message.setText("Click anywhere to quit!")

HANDLING TEXTUAL INPUT

- The triangle program's input was done completely through mouse clicks.
- There is also an Entry object that can get keyboard input.
- The Entry object draws a box on the screen that can contain text.
- It understands setText and getText, with one difference that the input can be edited.

```
>>> from graphics import *
>>> def main():
    win = GraphWin()
    win.setCoords(0.0, 0.0, 3.0, 4.0)

    Text(Point(1,3), "Celsius Temperature:").draw(win)
    Text(Point(1,1), "Fahrenheit Temperature:").draw(win)

    input = Entry(Point(2,3), 5)
    input.setText("0.0")
    input.draw(win)

    output = Text(Point(2,1), "")
    output.draw(win)

    button = Text(Point(1.5,2.0), "Convert")
    button.draw(win)

    Rectangle(Point(1,1.5), Point(2,2.5)).draw(win)

    win.getMouse()

    celsius = eval(input.getText())
    fahrenheit = 9.0/5.0 * celsius + 32

    output.setText("%0.1f" % fahrenheit)

    button.setText("QUIT")

    win.getMouse()
    win.close()
```





HANDLING TEXTUAL INPUT

- When run, this program produces a window with an entry box for typing in the Celsius temperature and a button to “do” the conversion.
  - The button is for show only! We are just waiting for a mouse click anywhere in the window.
- Initially, the input entry box is set to contain “0.0”.
- The user can delete this value and type in another value.

HANDLING TEXTUAL INPUT

- The program pauses until the user click the mouse – we don't care where so we don't store the point!
- The input is stored in three steps:
  - The value entered is converted into a number with eval.
  - This number is converted to degrees Fahrenheit.
  - This number is then converted to a string and formatted for display in the output text area.

CHAPTER 4 SUMMARY

- An object is a computational entity that combines data and operations. An object's data is stored in instance variables, and its operations are called methods.
- Every object is an instance of some class. It is the class that determines what method an object will have. An instance is created by calling a constructor method.

CHAPTER 4 SUMMARY

- An objects attributes are accessed vis dot notation. Generally computations with objects are performed by calling on an object's methods. Accessor methods return information about the instance variables of an object, Mutator methods change the values of instance variables.
- The graphics module provided for this class provides a number of classes that are useful for graphics programming.

CHAPTER 4 SUMMARY

- A GraphWin is an object that represents a window on the screen for displaying graphics.
- Point, Line, Circle, Rectangle, Oval, Polygon and Text are provided classes.
- An important consideration in graphical programming is the choice of an appropriate coordinate system.
- The graphics library provides a way of automating certain coordinate transformations.

CHAPTER 4 SUMMARY

- The situation where two variables refer to the same object is called aliasing.
- It can sometimes cause unexpected results. Use of the clone method in the graphics library can help prevent these situations.

CHAPTER 4 HOMEWORK

REVIEW QUESTIONS

True/False (ALL)

Multiple Choice (ALL)

Discussion Questions:

1, 2, and 3

Programming Exercises:

1, 2, 3, 6, 8, 9, and 10

DUE DATE: 02/15/2011