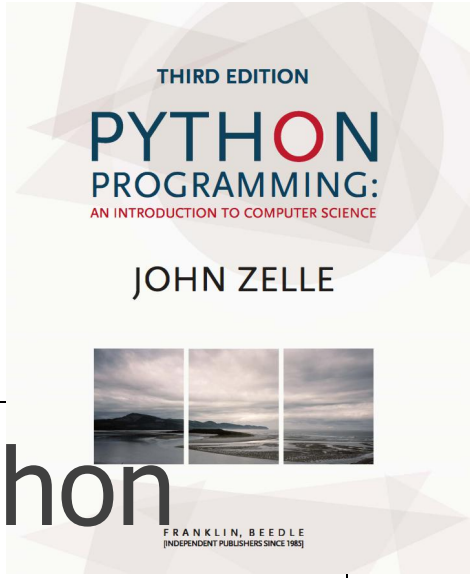


An Introduction to Python Programming

Chapter 4:Objects and Graphics

Objectives

- To be able to create objects in programs and call appropriate methods to perform graphical computations.
- To understand the fundamental concepts of computer graphics.
- To be able to write simple interactive graphics programs using the graphics library.

Overview

- Basically we viewed the data as passive entities combined via active operations.
- object-oriented (OO) approach
- Graphical programming

Overview

- Most of the applications probably have a so-called graphical user interace (GUI).
- Python comes with its own standard GUI module called *Tkinter*
- we will use a graphics library (graphics . py)

The Object of Objects

- Object-oriented is to view a complex system as the interaction of simpler objects.
- Objects can know & do
- .Objects interact by sending each other messages.
- A message is simply a request for an object to perform one of its operations.

The Object of Objects

- Suppose we want to develop a data processing system for a college or university.
- Objects :Students,Each course，etc.
  - **Student** : name, ID number, courses taken, campus address, home address, GPA;printCapusAddress()
  - **Course**:instructor,students,prerequisites,when,where;addStudent()
  - **Instructors:**rooms, times……

Simple Graphics Programming

- You will need a copy of the file ***graphics . py***.
- Import the graphics commands into an interactive Python session.

```
>>> import graphics
>>>
```

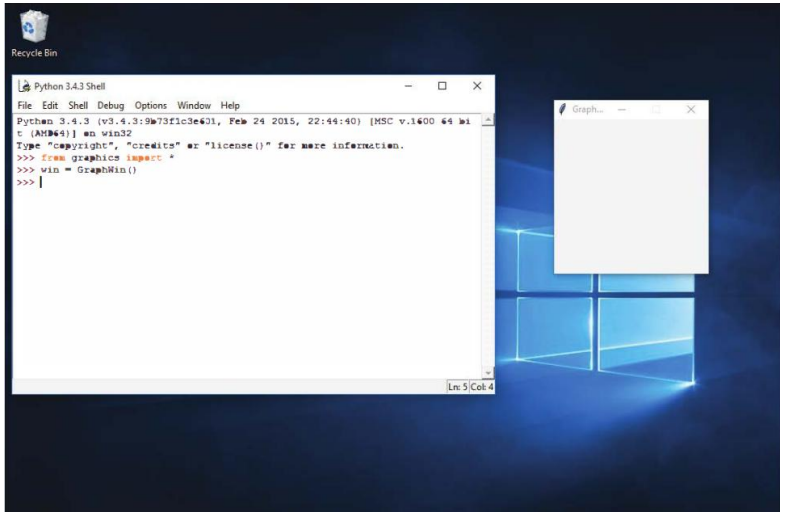
- Create a place on the screen.

```
>>> win = graphics.GraphWin()
>>>
```

Simple Graphics Programming

- The object **GraphWin** have been assigned to the variable called **win**.
- We can now manipulate the **window object** through this **variable**.

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



Simple Graphics Programming

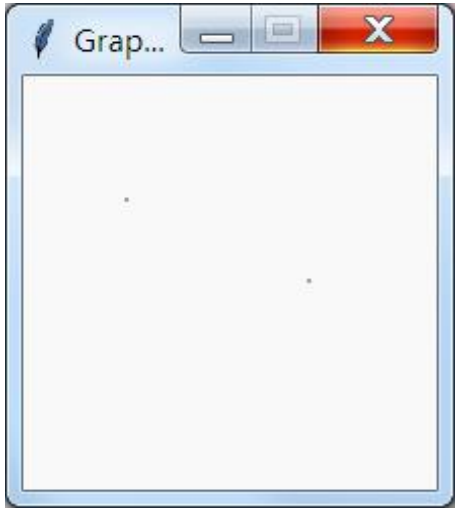
- We can create a GraphWin more simply:

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

- The simplest object in the graphics module is a **Point**,which represent a location in a GraphWin.
- Graphics programmers locate the point (0, 0) in the upper-lef
- corner of the window.
- The default size of GraphWin is 200 x 200.

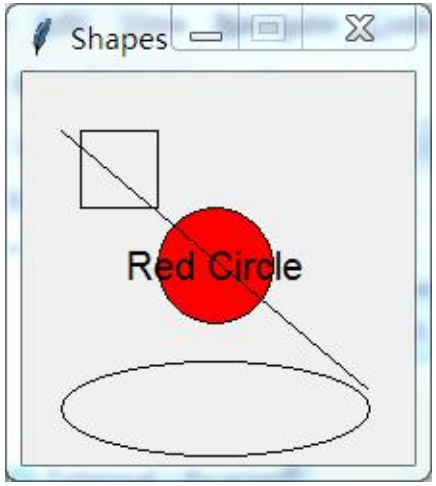
Simple Graphics Programming

```
>>> p = Point(50,60)
>>> p.getX()
50
>>> p.getY()
60
>>> win = GraphWin()
>>> p.draw(win)
>>> p2 = Point(140,100)
>>> p2.draw(win)
```



Simple Graphics Programming

```
# Open a graphics window
win = GraphWin('Shapes')
# draw a red circle, centered at point (100,100) with radius 30
center = Point(100, 100)
circ = Circle(center, 30)
corc.setFill('red')
circ.draw(win)
# Put a textual label in the center of the circle
label= Text(center, "Red Circle")
label.daw(win)
# Draw a square using a Rectangle object
rect = Rectangle(Point(30,30), Point(70,70))
rect.daw(win)
# Draw a line segent using a Line object
line= Line(Point(20,30), Point(180, 165))
line.draw(win)
# Draw a oval using the Oval Object
oval= Oval(Point(20,150),Point(180,199))
oval.draw(win)
```



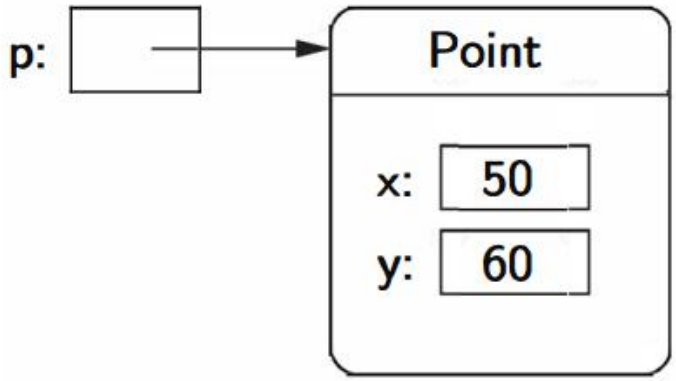
Using Graphical Objects

- Different kinds of *Objects: GraphWin, Point, Circle, Oval, Line, Text, and Rectagle.* These are *instances* of *classes*.
  - For example: Fido is a dog. Fido is an instance of this class, we expect certain things. Fido has four legs, a tail, a cold, wet nose, and he barks.

Using Graphical Objects

- Often, a constructor is used.

```
p = Point(50,60)
```



- Points also contain other information such as their color and which window they are drawn in. These are set to default values when the Point is created.

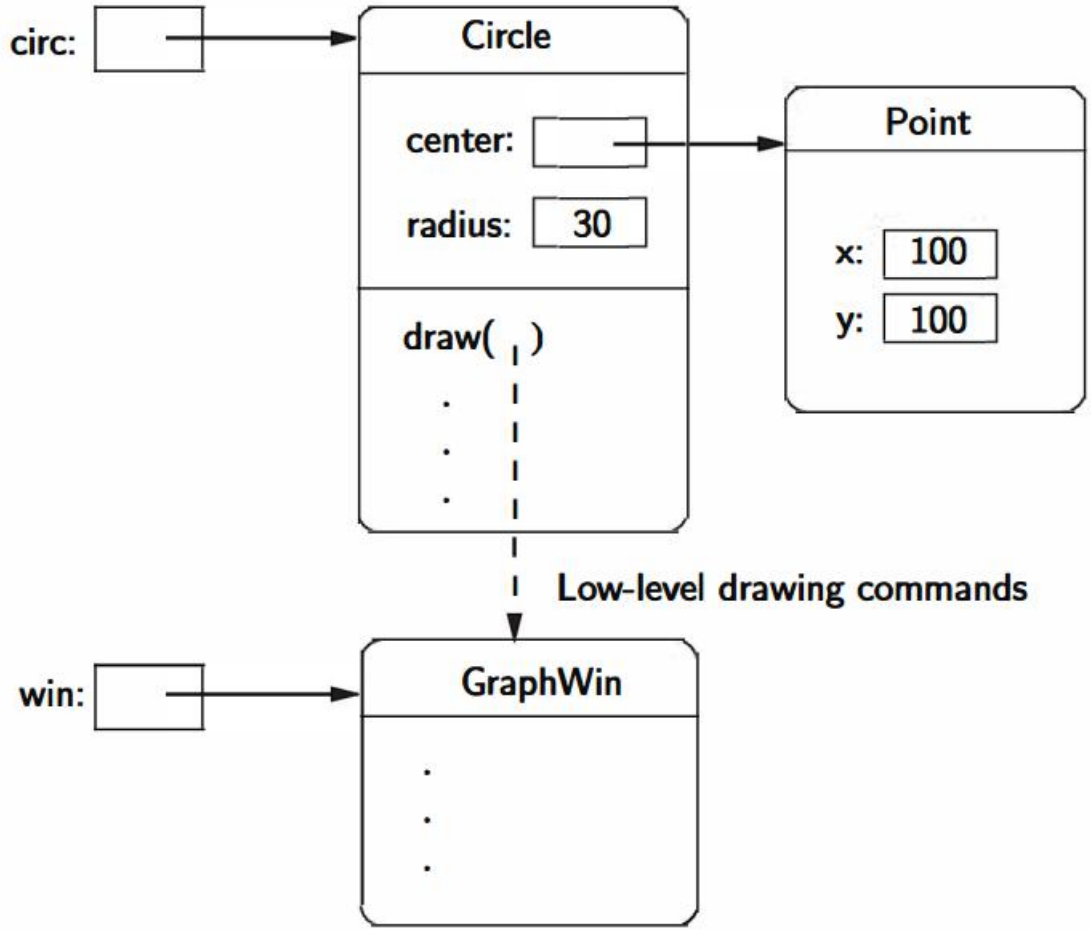
Using Graphical Objects

- To perform an operation on an object, we send the object a message (called method, sometimes called accessors or mutators).

```
>>> p . getX ()
>>> p . getY ()
>>> p . move ( 10 , 0)
```

Using Graphical Objects

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



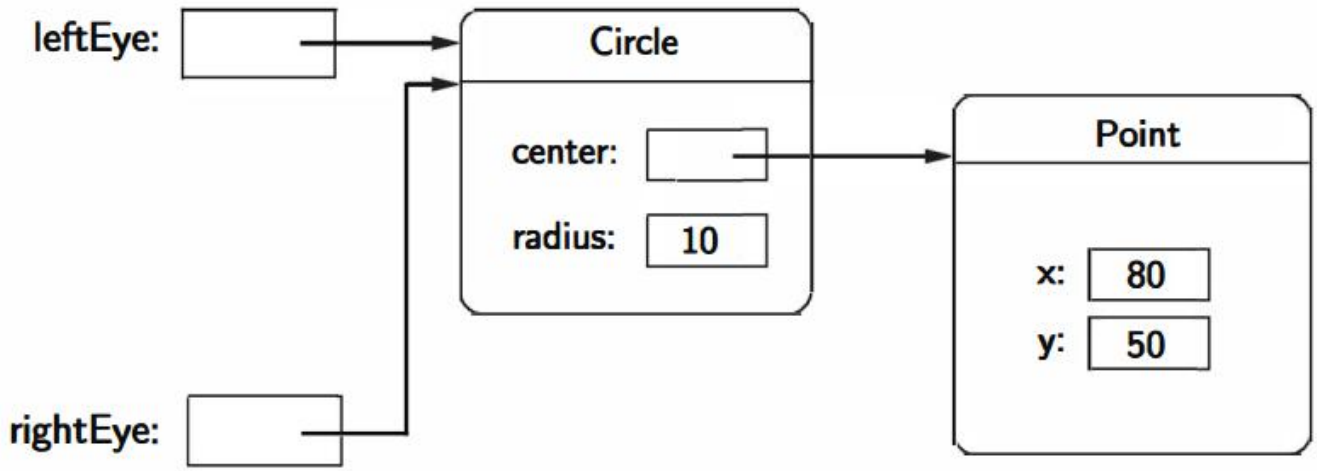
Using Graphical Objects

- We are trying to draw a smiley face.

```
## Incorrect way to create two circles.
leftEye = Circle(Point(80, 50), 5)
leftEye.setFill('yellow')
leftEye.setOutline('red')
rightEye = leftEye
rightEye.move(20,0)
```

This doesn't work.
Just aliasing

Using Graphical Objects



The String Data Type

- The graphics library provides a better solution.

```
from graphics import *
# Open a graphics window
win = GraphWin('Smile')
leftEye = Circle (Point (80 , 50) , 5)
leftEye . setFill ('yellow')
leftEye . setOutline ('red')
rightEye = leftEye . clone ( ) # rightEye is a exact copy of the left
rightEye . move (20 , 0)
leftEye.draw(win)
rightEye.draw(win)
input()
```

Strategic use of cloning

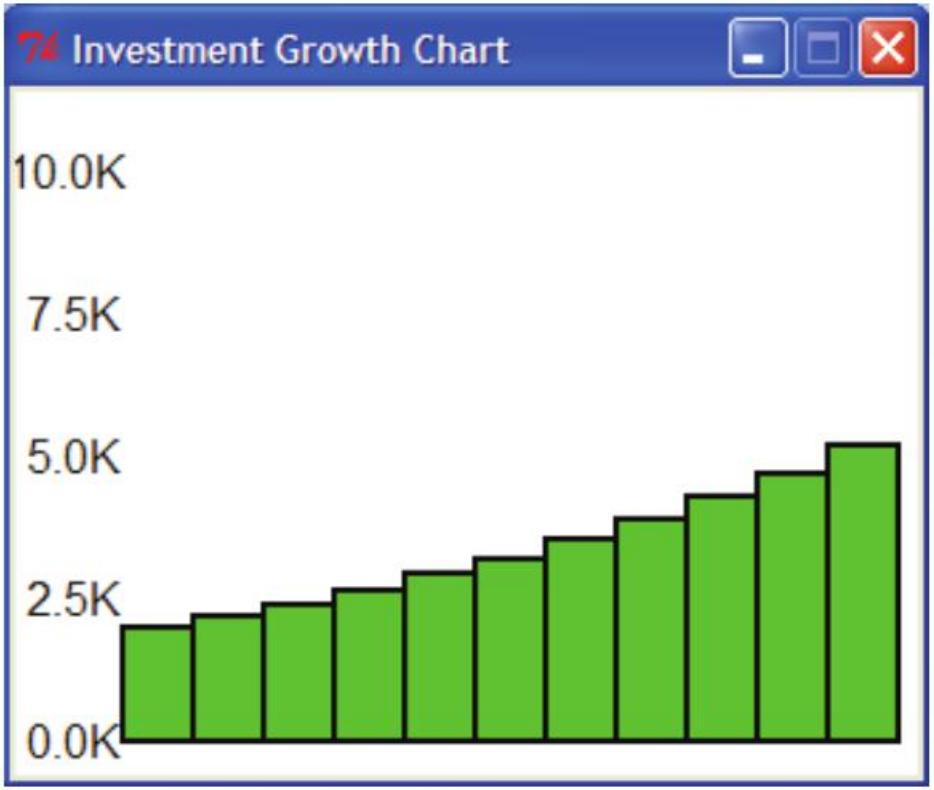
Graphing Future Value

- **A picture is worth a thousand words!**
- Programming with graphics requires careful planning.
  - Take the **futval . py** as an example.

```
Print an introduction
Get value of principal and apr from user
Create a GraphWin
Draw scale labels on left side of window
Draw bar at position 0 with height corresponding to principal
For successive years 1 through 10
    Calculate principal = principal * (1 + apr)
    Draw a bar for this year having a height corresponding to principal
Wait for user to press Enter.
```

Graphing Future Value

years	value
0	$2,000.00
1	$2,200.00
2	$2,420.00
3	$2,662.00
4	$2,928.20
5	$3,221.02
6	$3,542.12
7	$3,897.43
8	$4,287.18
9	$4,715.90
10	$5,187.49



Graphing Future Value

- How to draw a bar with height corresponding to $3221.02?
- Let's start with the *size* of the GraphWin.

```
win = GraphWin("Investment Growth Chart", 320, 240)
```

- Assume the maximum scale is $10,000 , with the five labels "0.0K" to "10.0K" .

Graphing Future Value

- Drawing labels just like that:

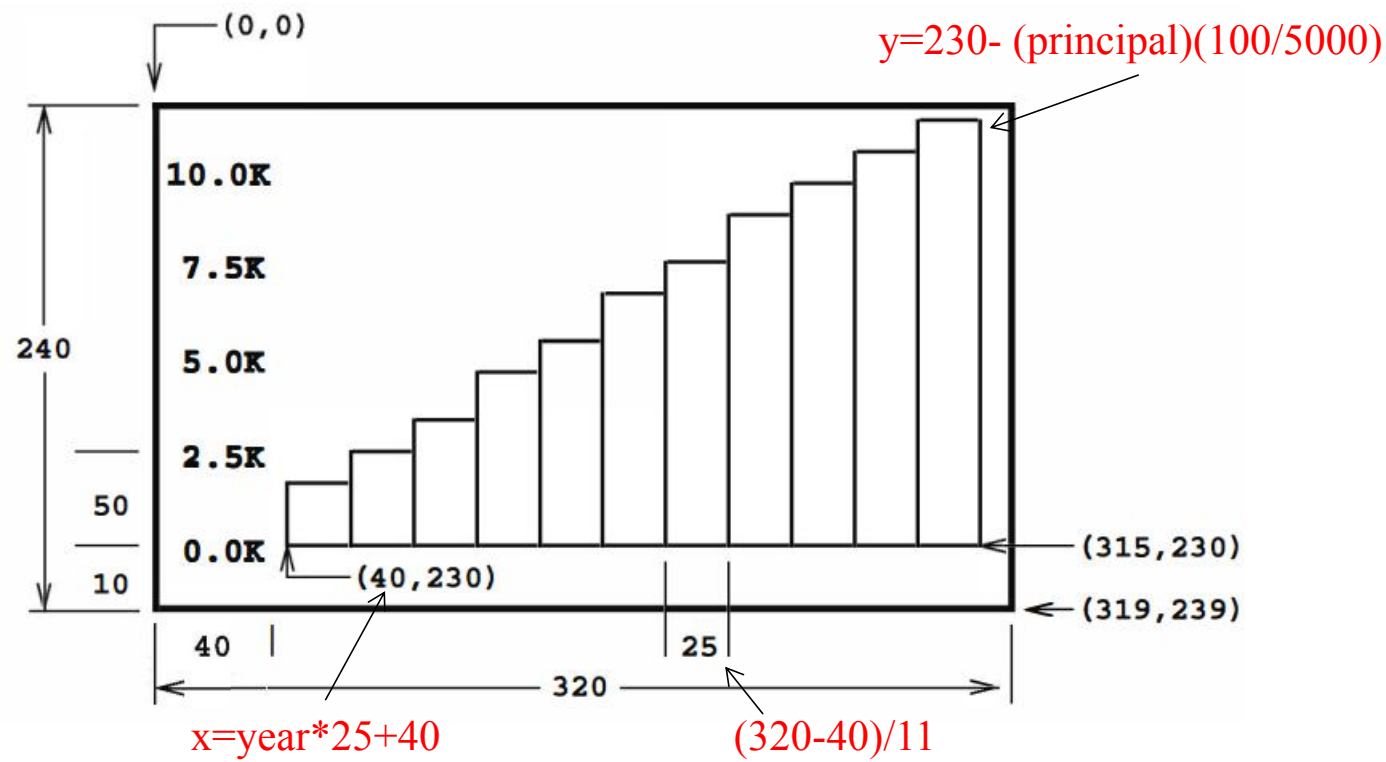
```
Draw scale labels on left side of window
```

becomes a sequence of steps:

```
Draw label " 0.0K" at (20, 230)
Draw label " 2.5K" at (20, 180)
Draw label " 5.0K" at (20, 130)
Draw label " 7.5K" at (20, 80)
Draw label "10.0K" at (20, 30)
```

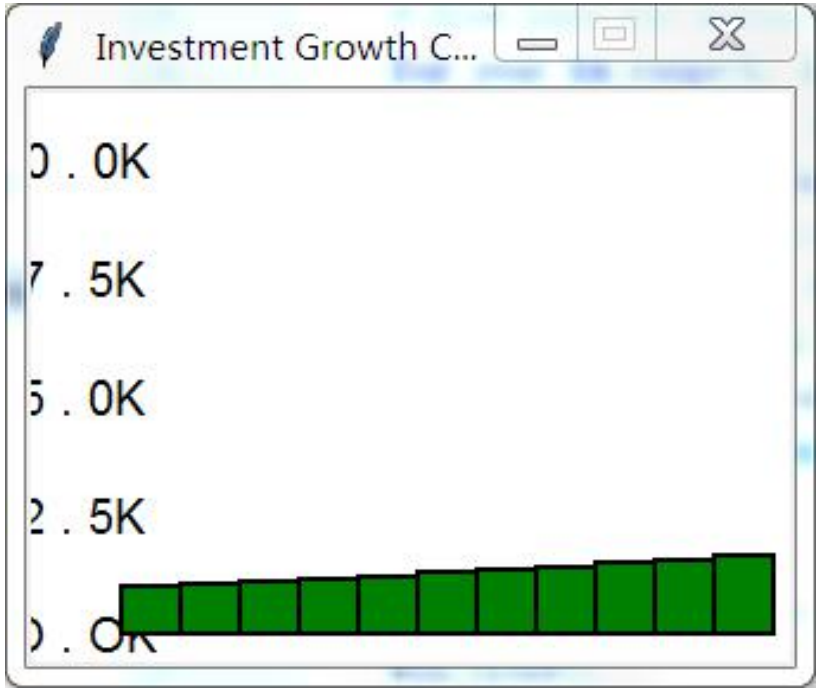
Graphing Future Value

- The next step is drawing the bar that corresponding to the amount of the principal.



Graphing Future Value

- The source code is in P102.



Choosing Coordinates

- Suppose we just want to divide the window into nine equal squares, **tic-tac-toe fashion**.
- The size of the window can be changed by changing parameters in *GraphWin()*.

Choosing Coordinates

- Apply this idea to our graphing future value program.
  - We want to go fom 0 through 10 (representing years) in the x dimension, and fom 0 to 10,000 (representing dollars) in the y dimension.

```
win = GraphWin("Investment Growth Chart", 320, 240)
win.setCoords(-1.75,-200, 11.5, 10400)
```

  - Each bar starts at the given year and a baseline of 0, and grows to the next year and a height equal to principal.

```
bar = Rectangle(Point(year, 0), Point(year+1, principal))
```

Interactive Graphics

- In a GUI envronment：
  - Applications use a technique called ***event-driven*** programming.
  - When the user moves the mouse, clicks a button, or types a key on the keyboard, this generates an ***event***.
  - It's hard to figure out "who's in charge" at any given moment.
  - The graphics module hides the underlying event-handling mechanisms and provides a few simple ways of getting user input in a GraphWin.

Interactive Graphics

1.Getting Mouse Clicks

- get graphical information fom the user via the ***getMouse*** method
- The spot where the user clicks is returned to the program as a **Point**.

Interactive Graphics

```
# click.py
from graphics import *

def main():
    win = GraphWin("Click Me!")
    for i in range(10):
        p = win.getMouse()
        print("You clicked at:", p.getX(), p.getY())

main()
```

Interactive Graphics

- Another class **Polygon** is used for any multi-sided, closed shape.

```
triangle = Polygon(p1, p2, p3)
```

- Also，the **Text object** is used to provide prompts.To change the prompt, just change the text that is displayed.

```
message = Text(Point(5, 0.5), "Click on three points")
message.draw(win)
message.setText("Click anywhere to quit.")
```

Interactive Graphics

2.Handling Textual Input

- The GraphWin object provides a **getKey ()** method

```
for i in range(10):
    pt = win.getMouse()
    key = win.getKey()
    label = Text(pt, key)
    label.draw(win)
```

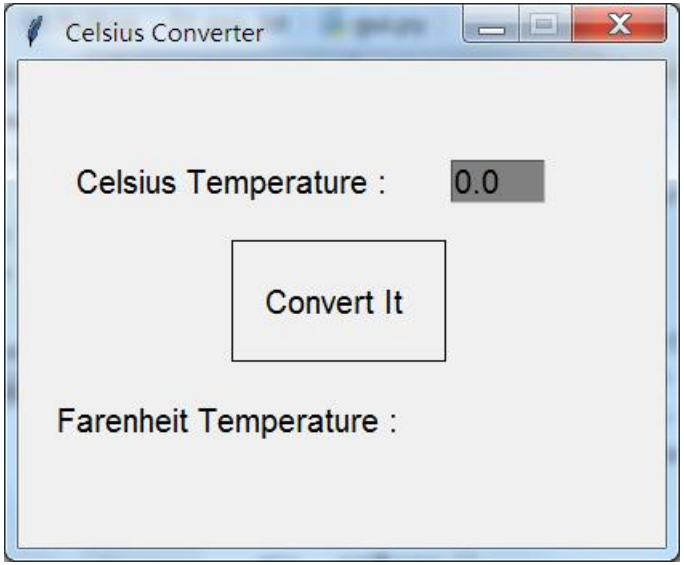
- The key that is pressed is returned as a **string** and saved as the variable key.
- *What strings are retured when you type <Shift>, <Ctrl>?*

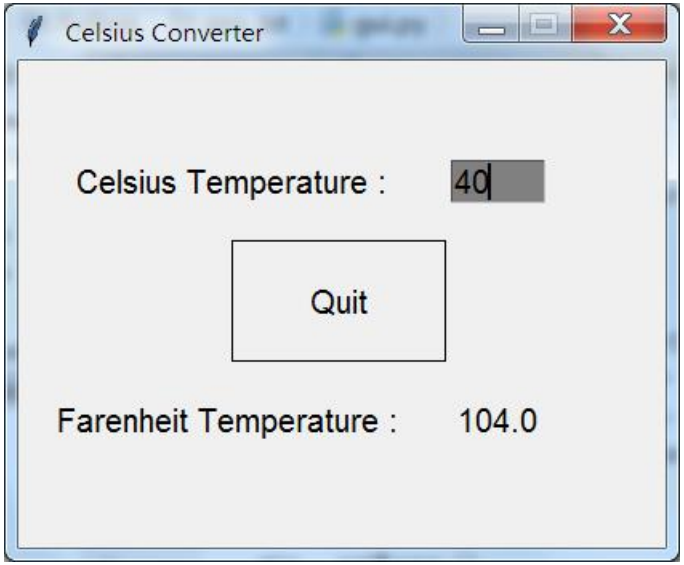
Interactive Graphics

- A **Entry object** draws a box on the screen that can contain text.
- It understands setText and getText methods.
- The contents of an Entry can be **edited** by the user.

```
# Draw the interface
Text(Point(1,3), "   Celsius Temperature:").draw(win)
Text(Point(1,1), "Fahrenheit Temperature:").draw(win)
inputText = Entry(Point(2.25, 3), 5)
inputText.setText("0.0")
```

Interactive Graphics





Graphics Module Reference

- This section provides a complete reference to the **objects and functions** provided in graphics.
- The set of objects and functions that are provided by a module is sometimes called an **Applications Programming Interace, or API.**

Graphics Module Reference

1.GraphWin Objects

- A GraphWin understands the following methods:
  - **GraphWin** (title , width , height)
  - **plot** (x , y , color) *Draws the pixel at (x, y) in the window.*
  - **plotPixel** (x , y , color) *Draws the pixel at the "raw" position (x, y),ignoring any coordinate transformations set up by setCoords.*
  - **setBackgroud** (color)
  - **close** ()
  - **getMouse** ()

Graphics Module Reference

- **checkMouse** () *Similar to getMouse, but does not pause for a user click. This is particularly useful for controlling animation loops*
- **getKey** ( )
- **checkKey**( ) *Similar to getKey, but does not pause for the user to press a key.This is particularly useful for controlling simple animation loops.*
- ***setCoords*** *(xll , yll , xur , yur)*

Graphics Module Reference

2.Graphics Objects

- The module provides:Point, Line,Circle, Oval,Rectangle, Polygon, and Text.
- All graphics objects support the following methods:
  - **setFill** (color)
  - **setOutline** (color)
  - **setWidth** (pixels)
  - **draw** (aGraphWin)
  - **undraw**()

Graphics Module Reference

- **move** (dx , dy)
- **clone** ()
- **Point Methods**:Point (x , y), getX (), getY ()
- **Line Methods**:Line (point! , point2) ,setArrow (endString),getCenter () ,getP1 () , getP2 ()
- others like **Circle Methods, Rectangle Methods, Oval Methods, Polygon Methods, Text Methods**

Graphics Module Reference

4.Displaying Images

- Most platforms will support at least PPM and GIF

images.

- Images support the generic methods move (dx , dy) , draw(graphwin) , undraw () , and clone ( ) .
- Image-specifc methods:Image (achorPoint , filename),getAnchor(),getWidth(),etc.

Graphics Module Reference

5.Generating Colors

- Colors are indicated by strings,such as "red", "purple ", "green", " cyan", etc.
- Many colors come in various shades,such as "red1 " ' "red2 " ' "red3 " ' "red4" ',look up *X11*color names on the web.
- The graphics module also provides a function for **mixing your own colors** .For example,color_rgb (255 , 0 , 0) is a bright red.

Graphics Module Reference

Color names with clashing definitions

Color name	X11 color					W3C color				
	Hex	Red	Green	Blue	Sample	Sample	Hex	Red	Green	Blue
Gray	#BEBEBE	75%	75%	75%			#808080	50%	50%	50%
Green	#00FF00	0%	100%	0%			#008000	0%	50%	0%
Maroon	#B03060	69%	19%	38%			#7F0000	50%	0%	0%
Purple	#A020F0	63%	13%	94%			#7F007F	50%	0%	50%

Colors with multiple names

W3C name[a]	Color					X11 name
	Hex	Red	Green	Blue	Sample	
Lime[10]	#00FF00	0%	100%	0%		Green
Fuchsia[10]	#FF00FF	100%	0%	100%		Magenta
Aqua[10]	#00FFFF	0%	100%	100%		Cyan

Graphics Module Reference

6.Controlling Display Updates

- When **using the graphics library inside** some interactive shells, it may be necessary to **force the window to update** in order for changes to be seen.
- The **update ()** function is provided to do this.
- The GraphWin includes a ***autoflush*** to controls this updating.

```
win = GraphWin("My Animation", 400, 400, autoflush=False)
```