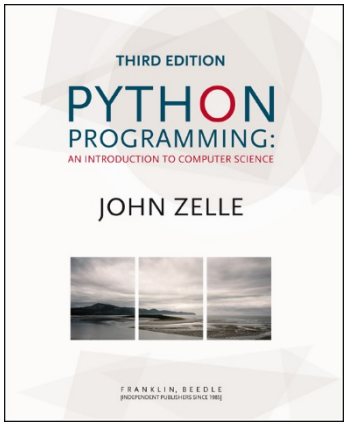
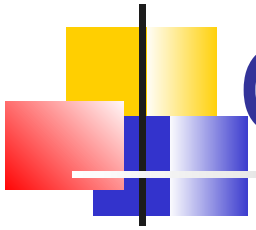
Python Programming: An Introduction to Computer Science

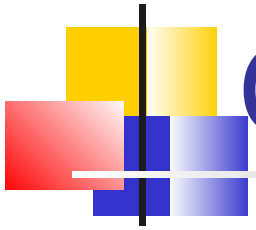


Chapter 4
Objects and Graphics



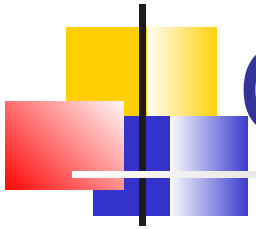
Objectives

- To understand the concept of objects and how they can be used to simplify programs.
- To be 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.



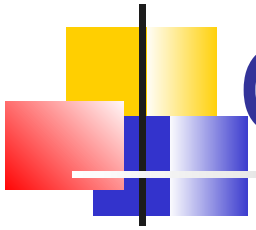
Objectives

- 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.



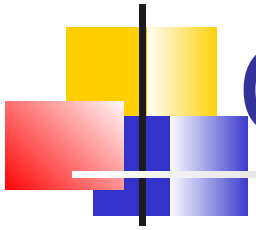
Objectives

- To be able to write simple interactive graphics programs using the graphics library.



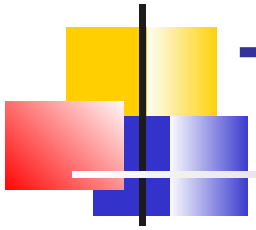
Overview

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



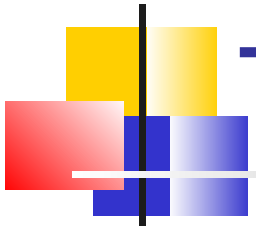
Overview

- 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's a graphics library (graphics.py) written specifically to go with this book. It's based on Tkinter.



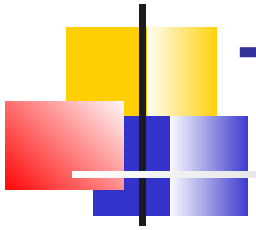
The Object of Objects

- Basic idea – view a complex system as the interaction 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.



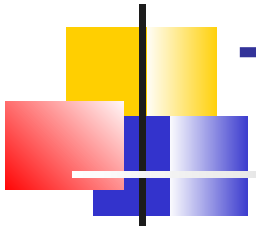
The Object of Objects

- 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.



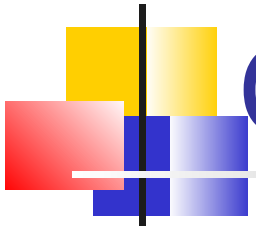
The Object of Objects

- The student object would contain data like:
  - Name
  - ID number
  - Courses taken
  - Campus Address
  - Home Address
  - GPA
  - Etc.



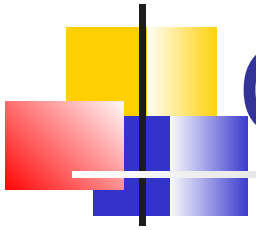
The Object of Objects

- 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.



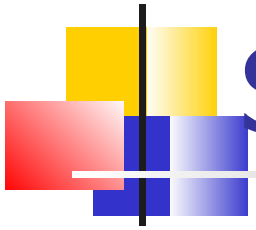
Object of Objects

- 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



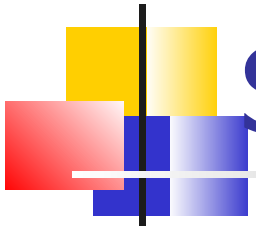
Object of Objects

- Sample Operation
  - `addStudent`
  - `delStudent`
  - `changeRoom`
  - `Etc.`



Simple Graphics Programming

- This chapter uses the `graphics.py` library supplied with the supplemental materials.
- Two location choices
  - In Python's Lib directory with other libraries
  - In the same folder as your graphics program



Simple Graphics Programming

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

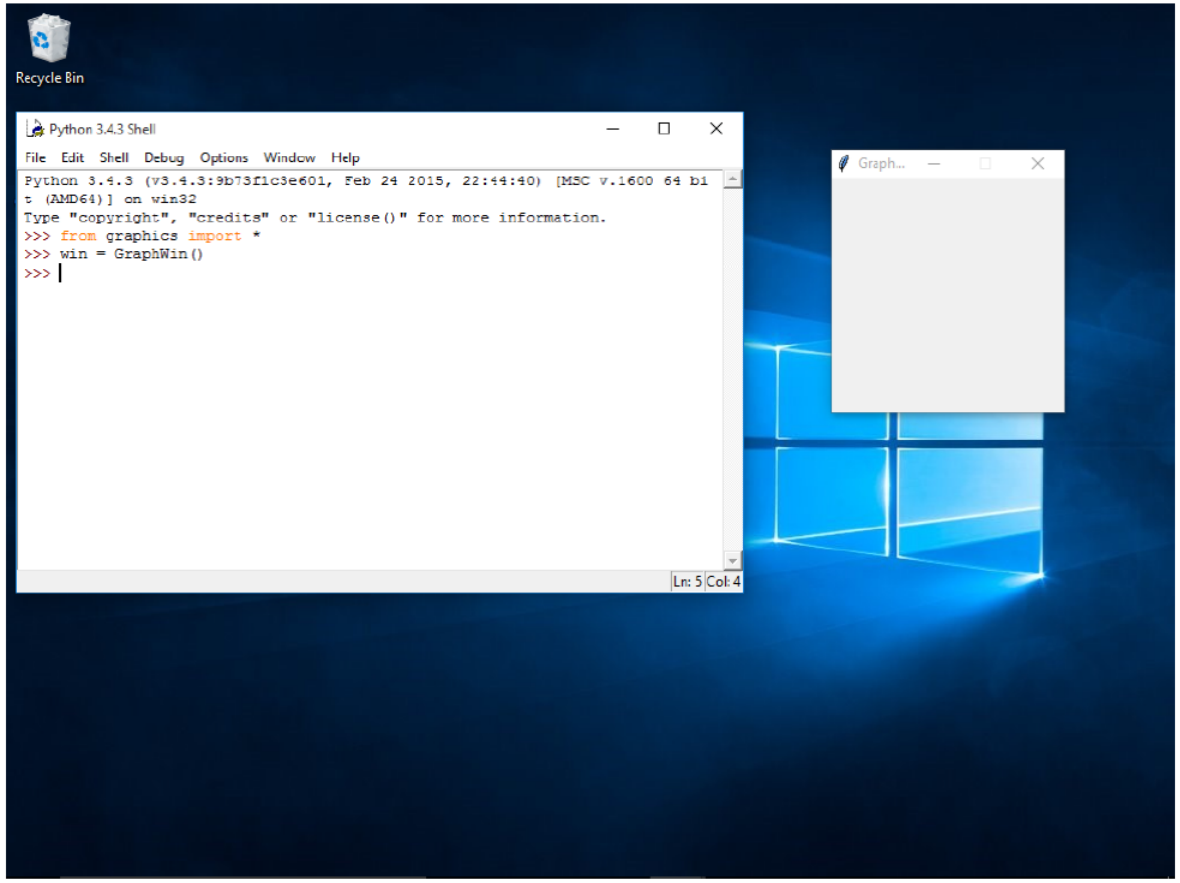
```
>>> import graphics
```

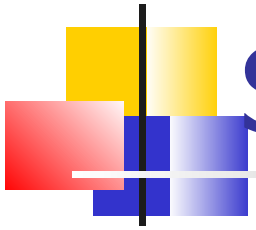
- A graphics window is a place on the screen where the graphics will appear.

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

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

Simple Graphics Programming

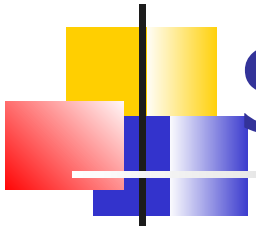




Simple Graphics Programming

- GraphWin is an object assigned to the variable win. We can manipulate the window object through this variable, similar to manipulating files through file variables.
- Windows can be closed/destroyed by issuing the command

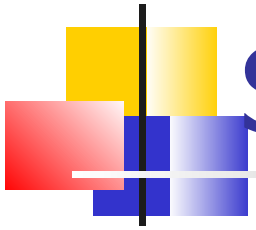
```
>>> win.close()
```



Simple Graphics Programming

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

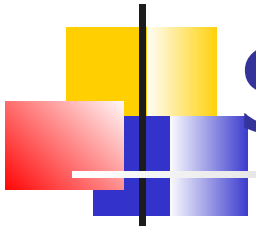
  The "from" statement allows you to load specific functions from a library module. "*" will load all the functions, or you can list specific ones.



Simple Graphics Programming

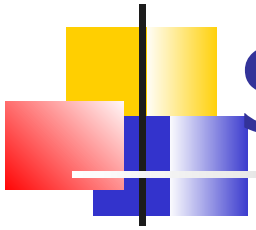
- Doing the import this way eliminates the need to preface graphics commands with `graphics.`

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



Simple Graphics Programming

- 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 library has a number of predefined routines to draw geometric shapes.

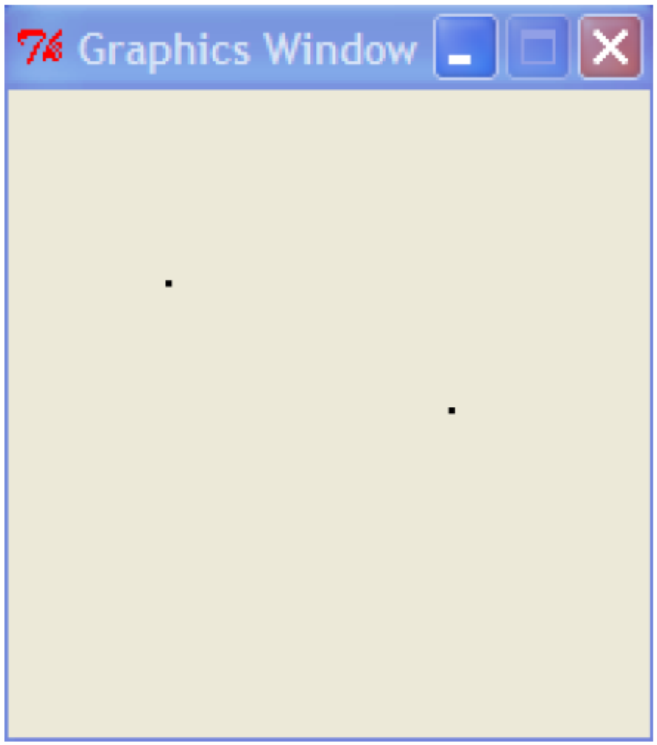


Simple Graphics Programming

- 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 of the point 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.
- Lower right corner is (199, 199)

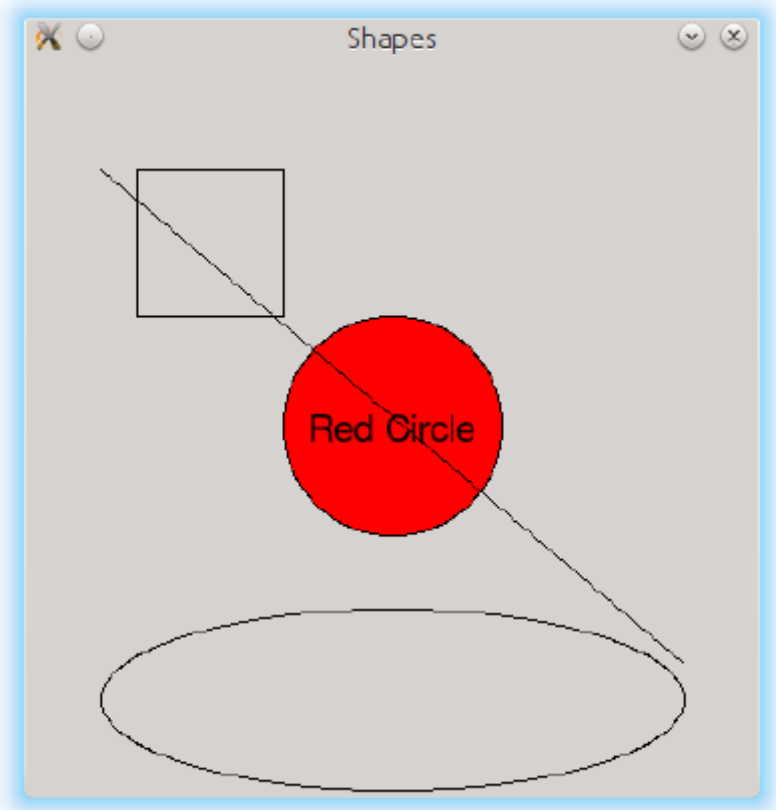
Simple Graphics Programming

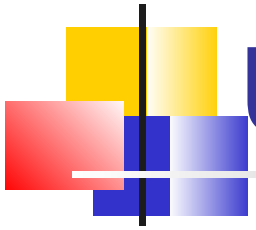
```
>>> p = Point(50, 60)
>>> p.getX()
50
>>> p.getY()
60
>>> win = GraphWin()
>>> p.draw(win)
>>> p2 = Point(140, 100)
>>> p2.draw(win)
```



Simple Graphics Programming

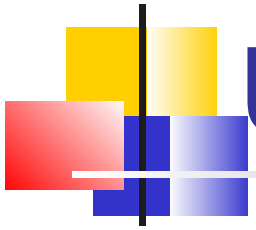
```
>>> ### Open a graphics window
>>> win = GraphWin('Shapes')
>>> ### Draw a red circle centered at point
>>> ### (100, 100) with radius 30
>>> center = Point(100, 100)
>>> circ = Circle(center, 30)
>>> circ.setFill('red')
>>> circ.draw(win)
>>> ### Put a textual label in the center of the circle
>>>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





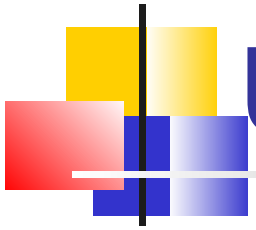
Using Graphical Objects

- 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` and `Rectangle`. These are examples of classes.



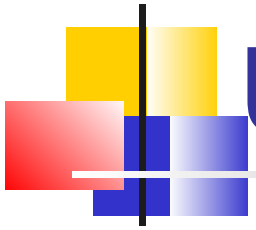
Using Graphical Objects

- Each object is an instance of some class, and the class describes the properties of the instance.
- If we say that Augie is a dog, we are actually saying that Augie is a specific individual in the larger class of all dogs. Augie is an instance of the dog 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.

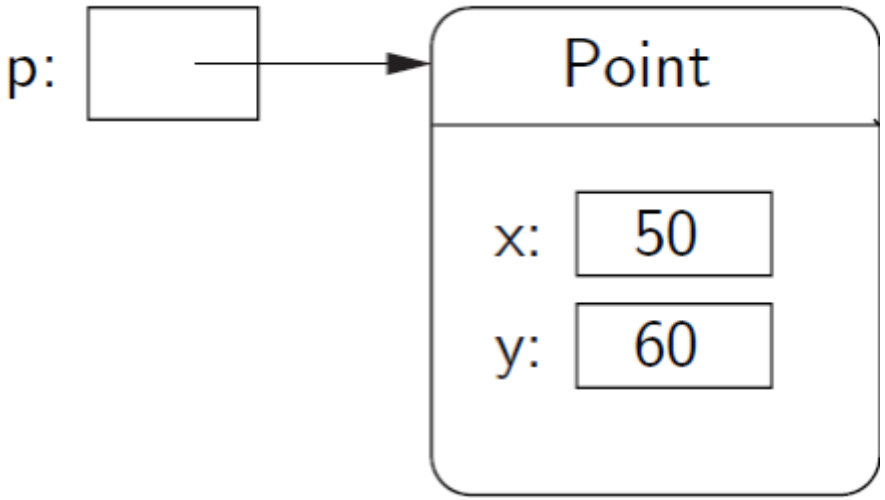


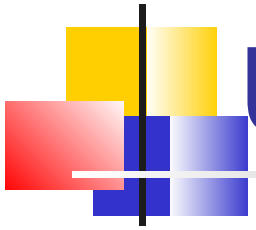
Using Graphical Objects

- `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 object.

Using Graphical Objects

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

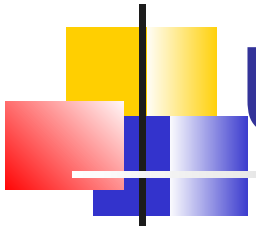




Using Graphical Objects

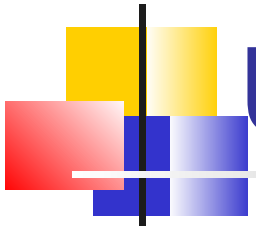
- To perform an operation on an object, we send the object a message. The set of messages an object responds to are called the methods of the object.
- Methods are like functions that live inside the object.
- Methods are invoked using dot-notation:

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



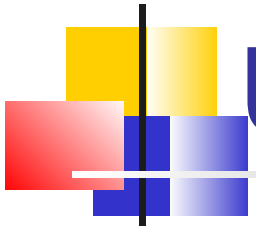
Using Graphical Objects

- `p.getX()` and `p.getY()` returns 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 variables.
- `move(dx, dy)` moves the object dx units in the x direction and dy in the y direction.
- Move erases the old image and draws it in its new position. Methods that change the state of an object are called mutators.



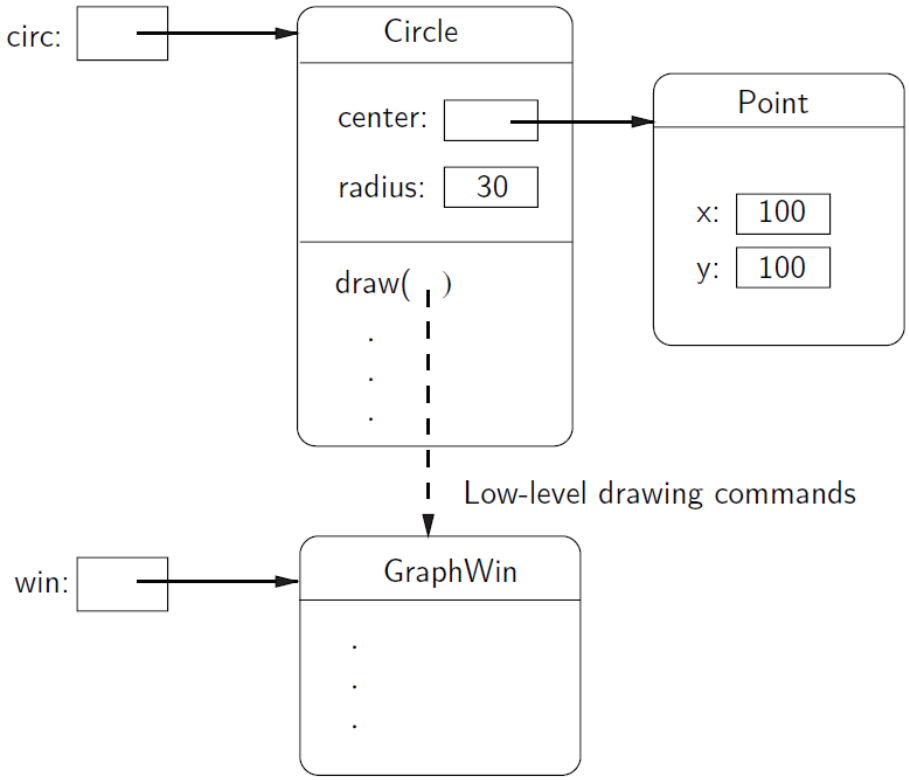
Using Graphical Objects

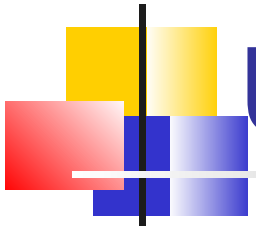
```
>>> circ = Circle(Point(100, 100), 30)
>>> win = GraphWin()
>>> circ.draw(win)
```

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



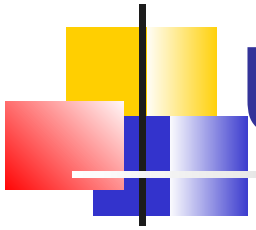


Using Graphical Objects

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

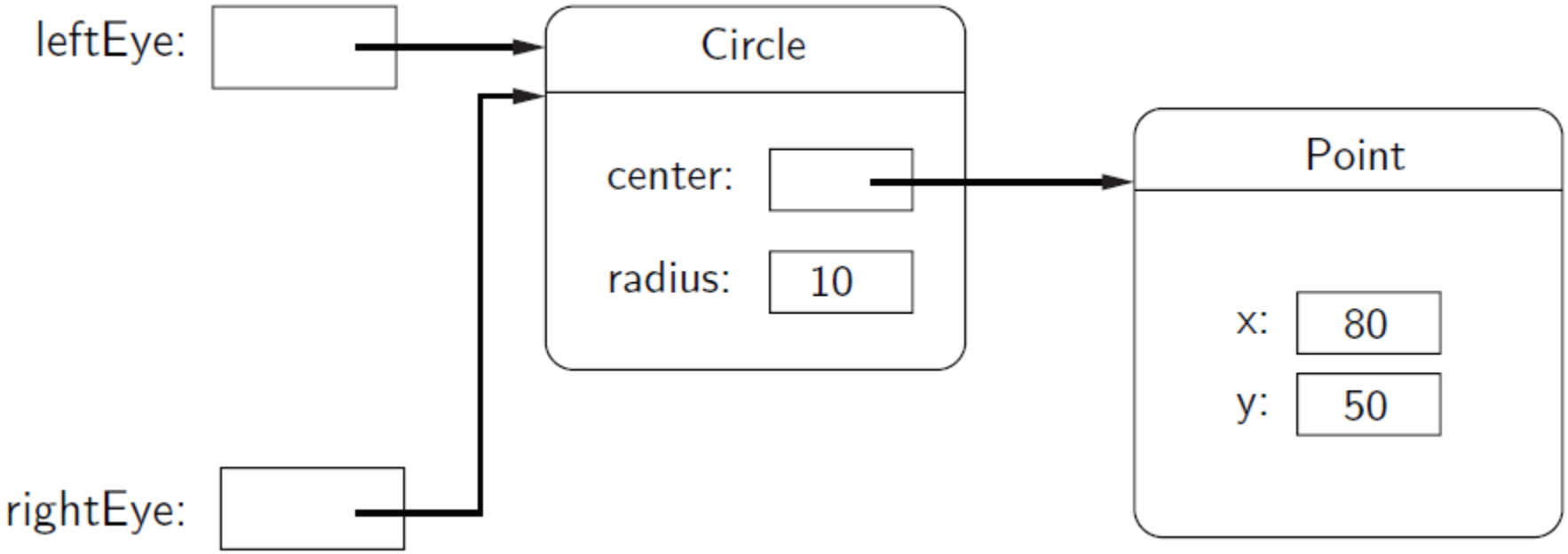
- The idea is to create the left eye and copy that to the right eye which gets moved over 20 units.

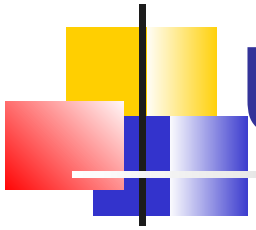


Using Graphical Objects

- 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

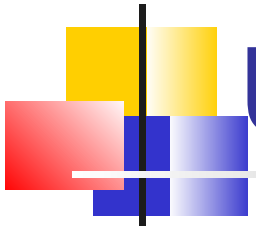




Using Graphical Objects

- 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')
```

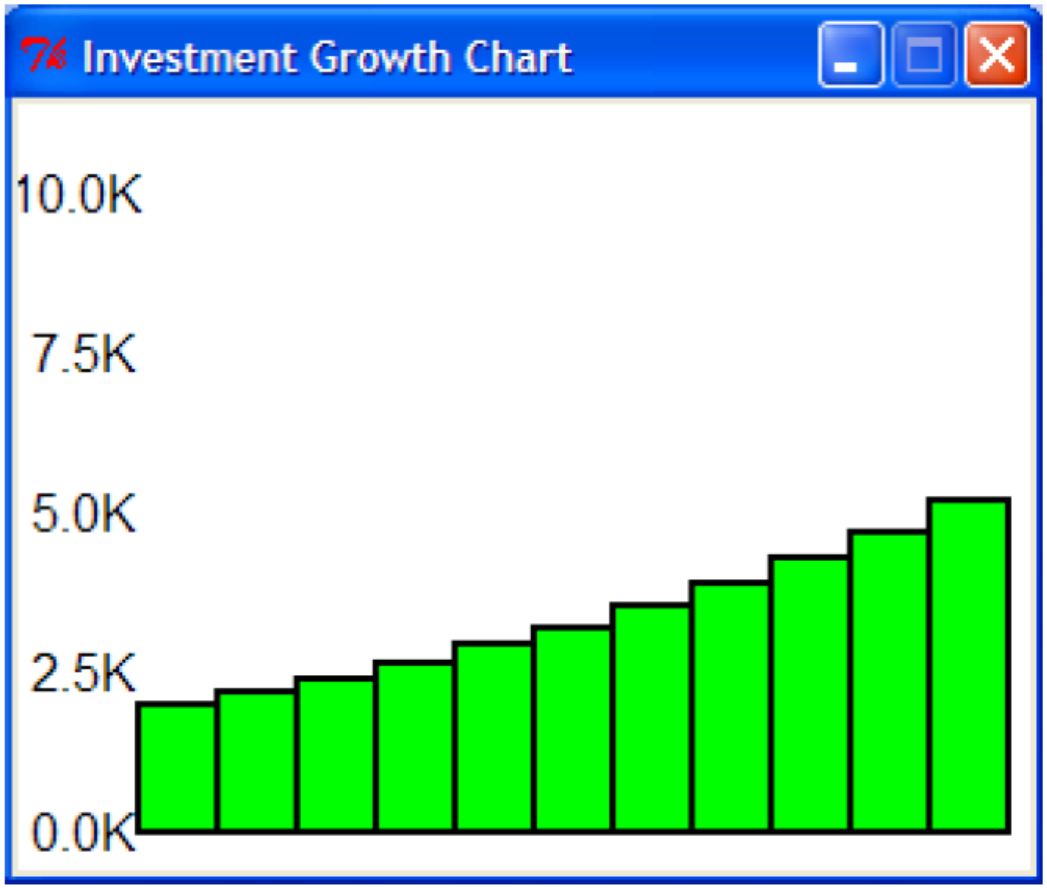


Using Graphical Objects

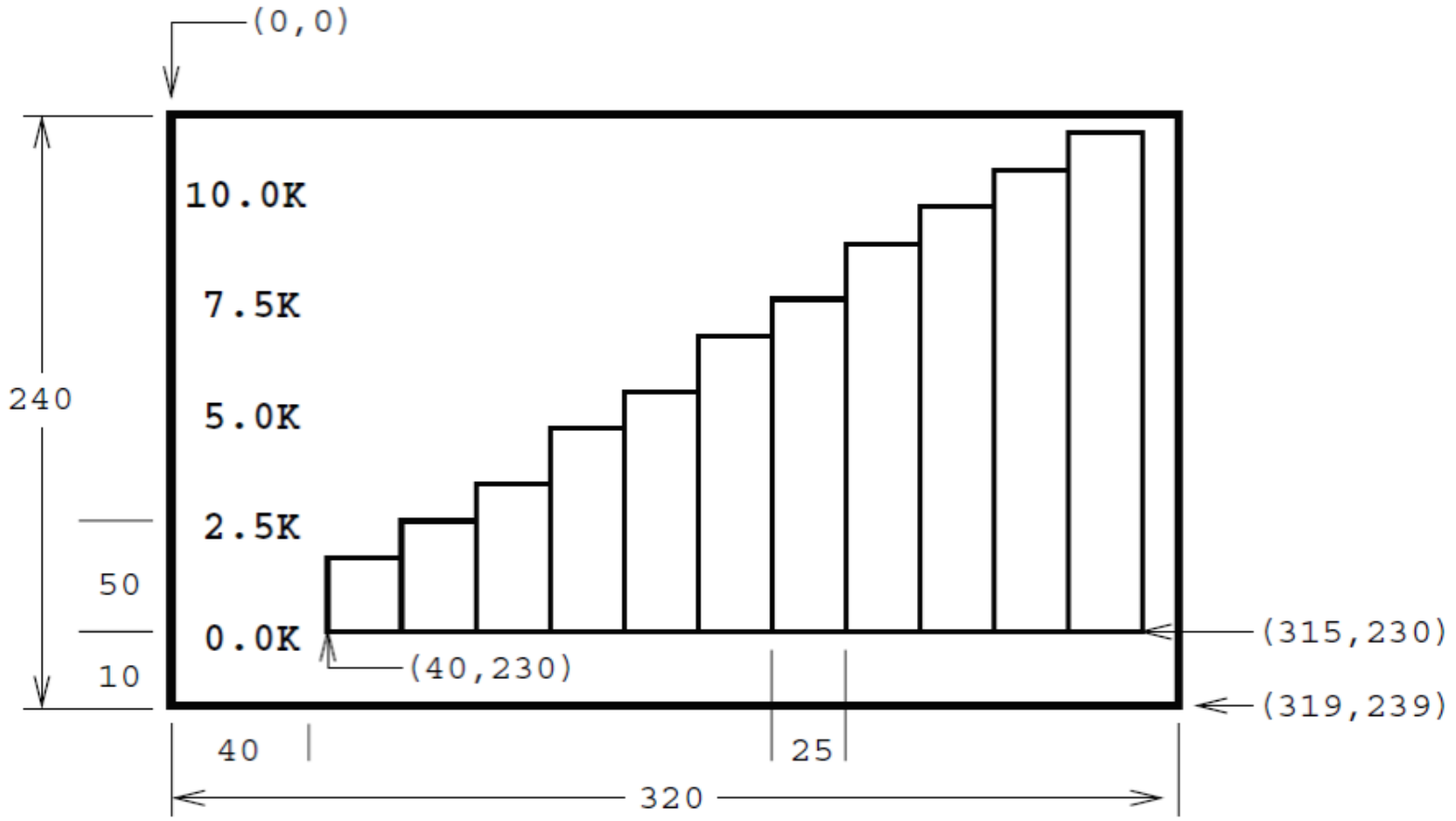
- The graphics library has a better solution. Graphical objects have a clone method that will make a copy of the object!

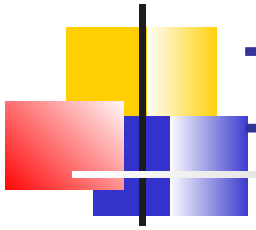
```
>>> # Correct way to create two circles, using clone
>>> leftEye = Circle(Point(80, 50), 5)
>>> leftEye.setFill('yellow')
>>> leftEye.setOutline('red')
>>> rightEye = leftEye.clone()
>>> # rightEye is an exact copy of the left
>>> rightEye.move(20, 0)
```

Graphing Future Value/ Choosing Coordinates



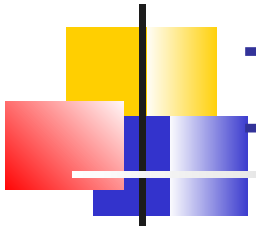
Graphing Future Value/ Choosing Coordinates





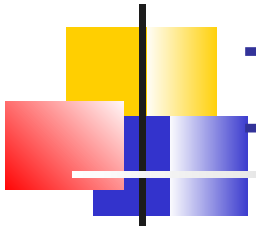
Interactive Graphics

- In a GUI environment, users typically interact with their applications by clicking on buttons, 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.



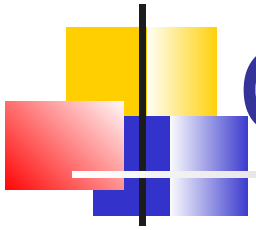
Interactive Graphics

- 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 just happened.
- The event object is sent to the appropriate part of the program to be processed, for example, a button event.



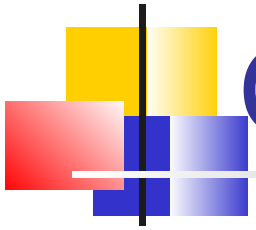
Interactive Graphics

- 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`.

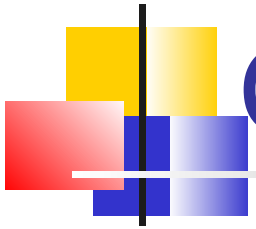


Getting Mouse Clicks

- The following code reports the coordinates of a mouse click:

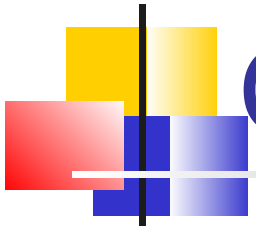
```
from graphics import *
win = GraphWin("Click Me!")
p = win.getMouse()
print("You clicked", p.getX(), p.getY())
```

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



Getting Mouse Clicks

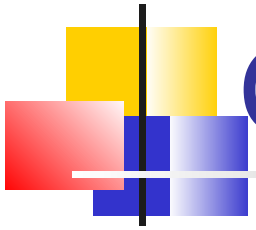
```
# triangle.pyw
# Interactive graphics program to draw a triangle

from graphics import *

def main():
    win = GraphWin("Draw a Triangle")
    win.setCoords(0.0, 0.0, 10.0, 10.0)
    message = Text(Point(5, 0.5), "Click on three points")
    message.draw(win)

    # Get and draw three vertices of triangle
    p1 = win.getMouse()
    p1.draw(win)
    p2 = win.getMouse()
    p2.draw(win)
    p3 = win.getMouse()
    p3.draw(win)
```



Getting Mouse Clicks

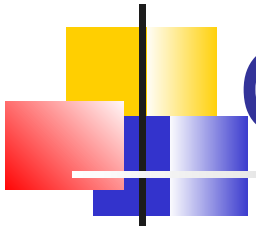
```
# Use Polygon object to draw the triangle
    triangle = Polygon(p1,p2,p3)
    triangle.setFill("peachpuff")
    triangle.setOutline("cyan")
    triangle.draw(win)

    # Wait for another click to exit
    message.setText("Click anywhere to quit.")
    win.getMouse()

main()
```

Getting Mouse Clicks



Getting Mouse Clicks

- Notes:
  - If you are programming in a 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.



Getting Mouse Clicks

- Once you have three points, creating a triangle polygon is easy:

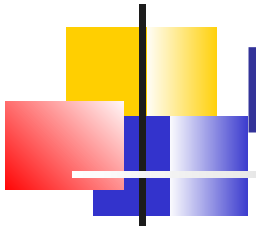
```
triangle = Polygon(p1, p2, p3)
```

- A single text object is created and drawn near the beginning of the program.

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

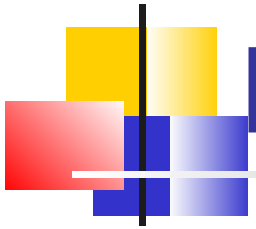
- To change the prompt, just change the text to be displayed.

```
message.setText("Click anywhere to quit.")
```



Handling Textual Input

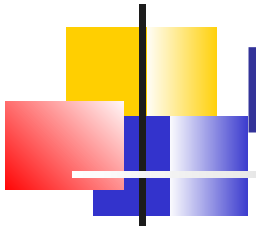
- The triangle program's input was done completely through mouse clicks.
- The `GraphWin` object provides a `getKey()` method that works like the `getMouse` method.



Handling Textual Input

```
# clickntype.py
from graphics import *
def main():
    win = GraphWin("Click and Type", 400, 400)
    for i in range(10):
        pt = win.getMouse()
        key = win.getKey()
        label = Text(pt, key)
        label.draw(win)
```

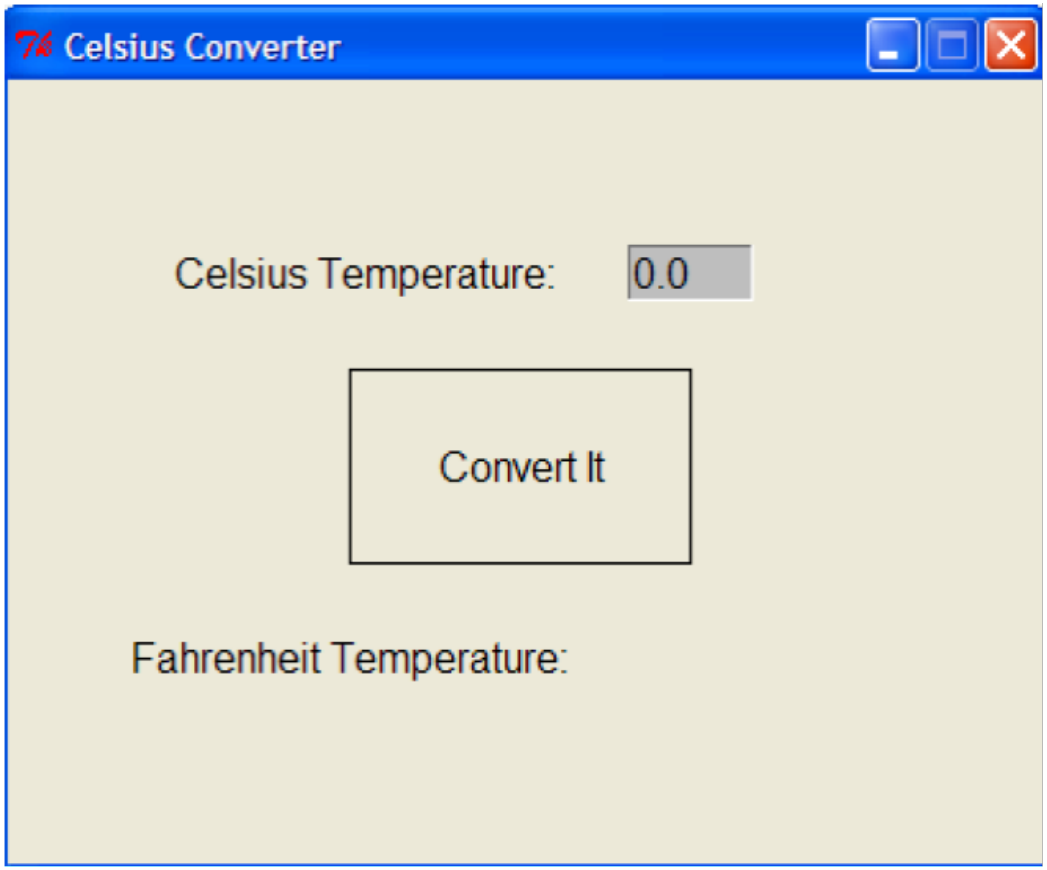
Handling Textual Input

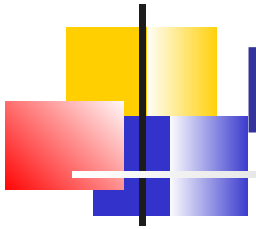


Handling Textual Input

- There's 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.

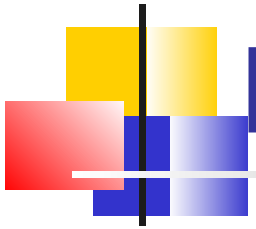
Handling Textual Input





Handling Textual Input

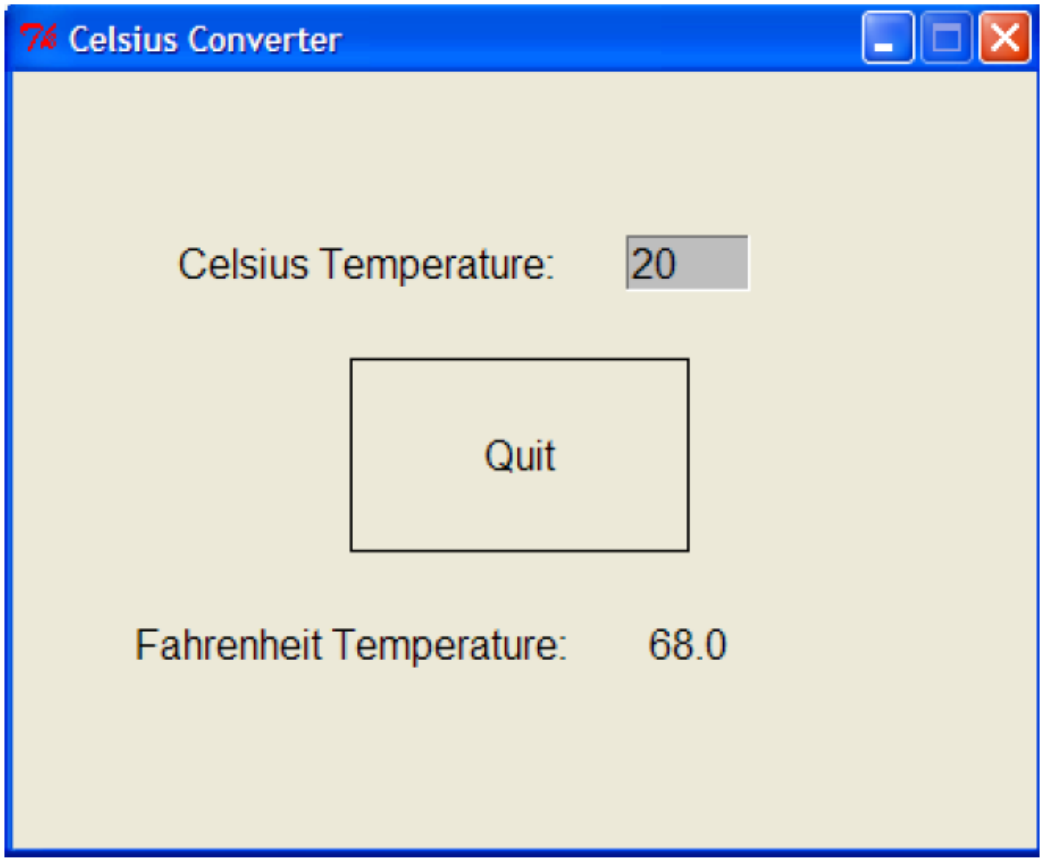
```
# convert_gui.pyw
# Program to convert Celsius to Fahrenheit using a simple
#   graphical interface.

from graphics import *

def main():
    win = GraphWin("Celsius Converter", 300, 200)
    win.setCoords(0.0, 0.0, 3.0, 4.0)

    # Draw the interface
    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 It")
    button.draw(win)
    Rectangle(Point(1,1.5), Point(2,2.5)).draw(win)
```

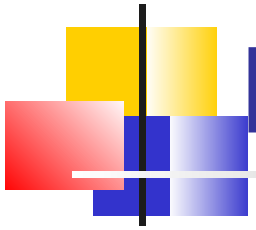


Handling Textual Input

```
    # wait for a mouse click
    win.getMouse()

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

    # display output and change button
    output.setText(fahrenheit)
    button.setText("Quit")

    # wait for click and then quit
    win.getMouse()
    win.close()

main()
```

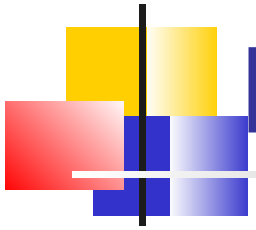
Handling Textual Input





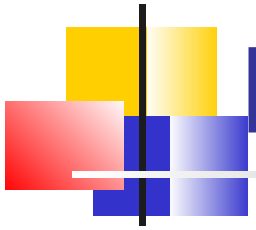
Handling Textual Input

- 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.



Handling Textual Input

- Initially, the input entry box is set to contain "0.0".
- The user can delete this value and type in another value.
- The program pauses until the user clicks the mouse – we don't care where so we don't store the point!



Handling Textual Input

- The input is processed in three steps:
  - The value entered is converted into a number with `float`.
  - This number is converted to degrees Fahrenheit.
  - This number is then converted to a string and formatted for display in the `output` text area.