Computational Thinking with Programming

@cse_bennett @csebennett

Overview

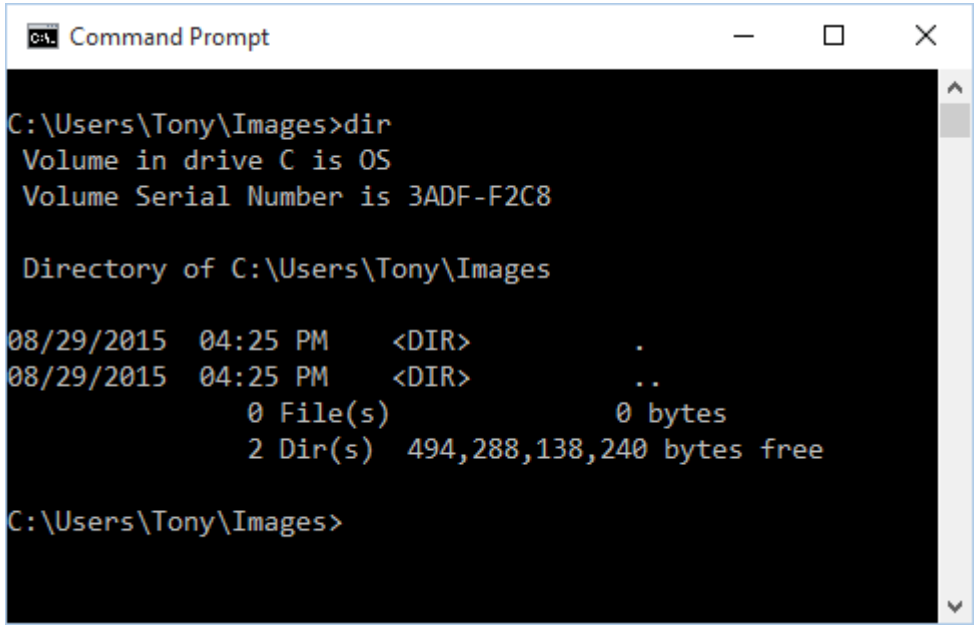
- Each data type can represent a certain set of values, and each had a set of associated operations.
- The traditional programming view is that data is passive – it's manipulated and combined with active operations.
- Modern computer programs are built using an *object-oriented* approach.
- Most applications you're familiar with have *Graphical User Interfaces* (GUI) that provide windows, icons, buttons and menus. These were provided as classes to the programmer – they did NOT have to write everything from scratch to make an interface.
- This is why people like OOP – the details can be ignored, the objects do things for you!

Graphical User Interfaces

- User Interface: the part of the computer with which the user interacts
- Command line interface: displays a prompt and the user types a command that is then executed
- Graphical User Interface (GUI): allows users to interact with a program through graphical elements on the screen

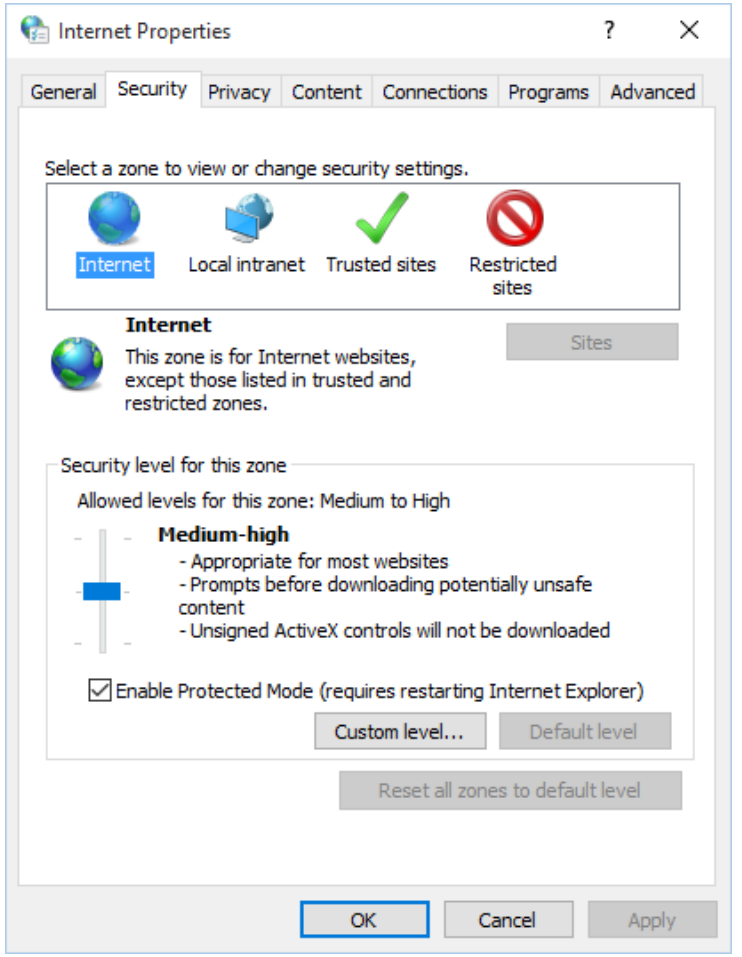
Graphical User Interfaces (cont'd.)

- A command line interface



Graphical User Interfaces (cont'd.)

- Dialog boxes: small windows that display information and allow the user to perform actions
  - Responsible for most of the interaction through GUI
  - User interacts with graphical elements such as icons, buttons, and slider bars



GUI Programs Are Event-Driven

- In text-based environments, programs determine the order in which things happen
  - The user can only enter data in the order requested by the program
- GUI environment is event-driven
  - The user determines the order in which things happen
    - User causes events to take place and the program responds to the events

Overview

- A commonly used graphics library for Python is named Tkinter. This is not a simple package to use!
- The author of a popular Python book (Zelle) wrote a library "on top of" Tkinter, to make it easier to use. It's called graphics.py
  - C:\Python32\Lib\site-packages\graphics.py
- This chapter uses the `graphics.py` library supplied with the supplemental materials.
- Two locations you can put the file
  - In Python's Lib\site-packages 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
  ```
  >>> from graphics import *
  ```
- A *graphics window* is a place on the screen where the graphics will appear.
  ```
  >>> win = GraphWin()
  ```
- This command creates a new window titled “Graphics Window.” You use the name `win` to refer to the window
- Command to install graphics on Jupiter
- !pip3 install --user http://bit.ly/csc161graphics
- https://www.rose-hulman.edu/class/csse/resources/Python/ZelleGraphics.html

Installing instructions

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 routine has a number of predefined routines to draw geometric shapes.
- https://www.cs.swarthmore.edu/~knerr/teaching/f17/topics/graphics.html
- https://www.cs.swarthmore.edu/courses/CS21Labs/s17/docs/graphics.php

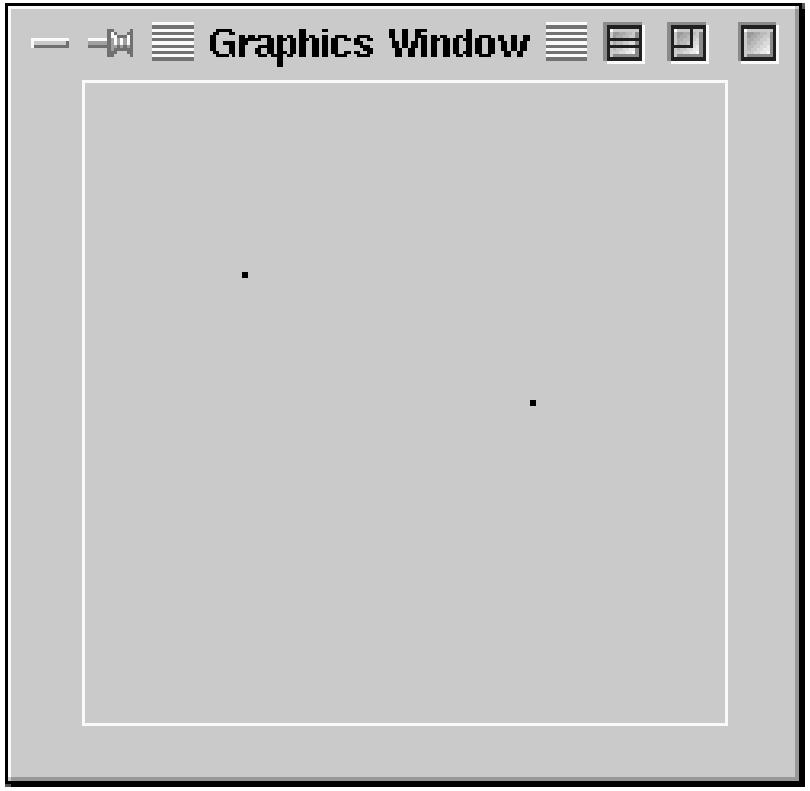
many programs related to zelly graphics library

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 right to left, y values from top to bottom.
- Lower right corner is (199, 199)

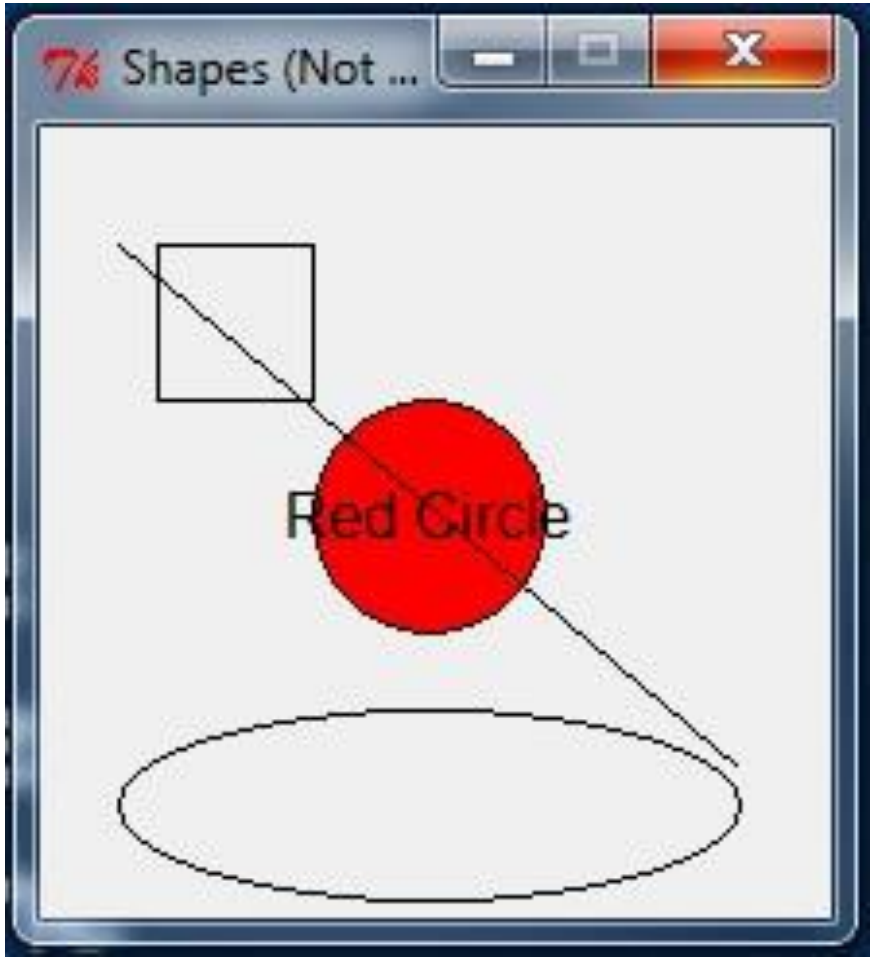
Simple Graphics Programming

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



Graphics Example

```
from graphics import *
def main():
    win = GraphWin('Shapes')
    center = Point(100, 100)
    circ = Circle(center, 30)
    circ.setFill('red')
    circ.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)
main()
```



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*.
- 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>(<arg1>, <arg2>, …)`
- `<class-name>` is the name of the class we want to create a new instance of, e.g. Circle or Point.
- The arguments are required to initialize the object. For example, Point requires two numeric values. `p = Point(50,100)`

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 functions that live inside the object.
- Methods are invoked using dot-notation:
  `<object>.<method-name>(<arg1>, <arg2>, …)`

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 object.
- Other methods change the *state* of the object
- `p.move(dx,dy)` moves the object `p` dx units in the *x* direction and dy units 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 onto the GraphWin object win.

Using Graphical Objects

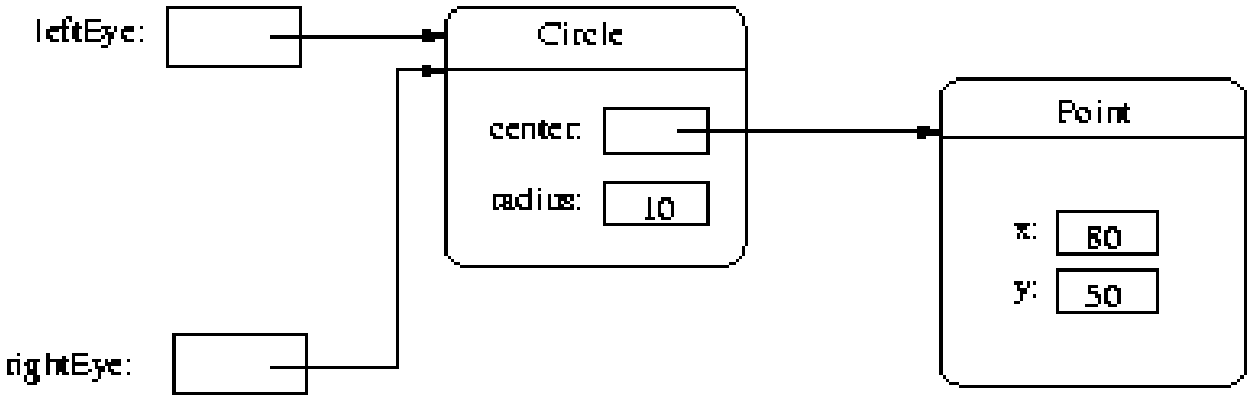
- It's possible for two different variables to refer to the same object – changes made to the object through one variable will be visible to the other.

```
>>> 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 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:

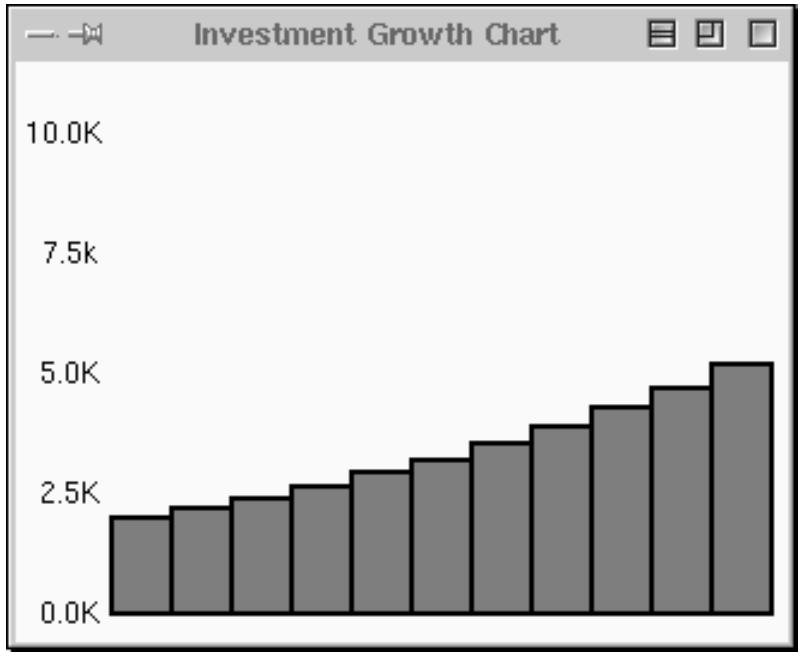
```
>>> leftEye = Circle(Point(80, 50), 5)
>>> leftEye.setFill('yellow')
>>> leftEye.setOutline('red')
>>> rightEye = Circle(Point(100, 50), 5)
>>> rightEye.setFill('yellow')
>>> rightEye.setOutline('red')
```

Using Graphical Objects

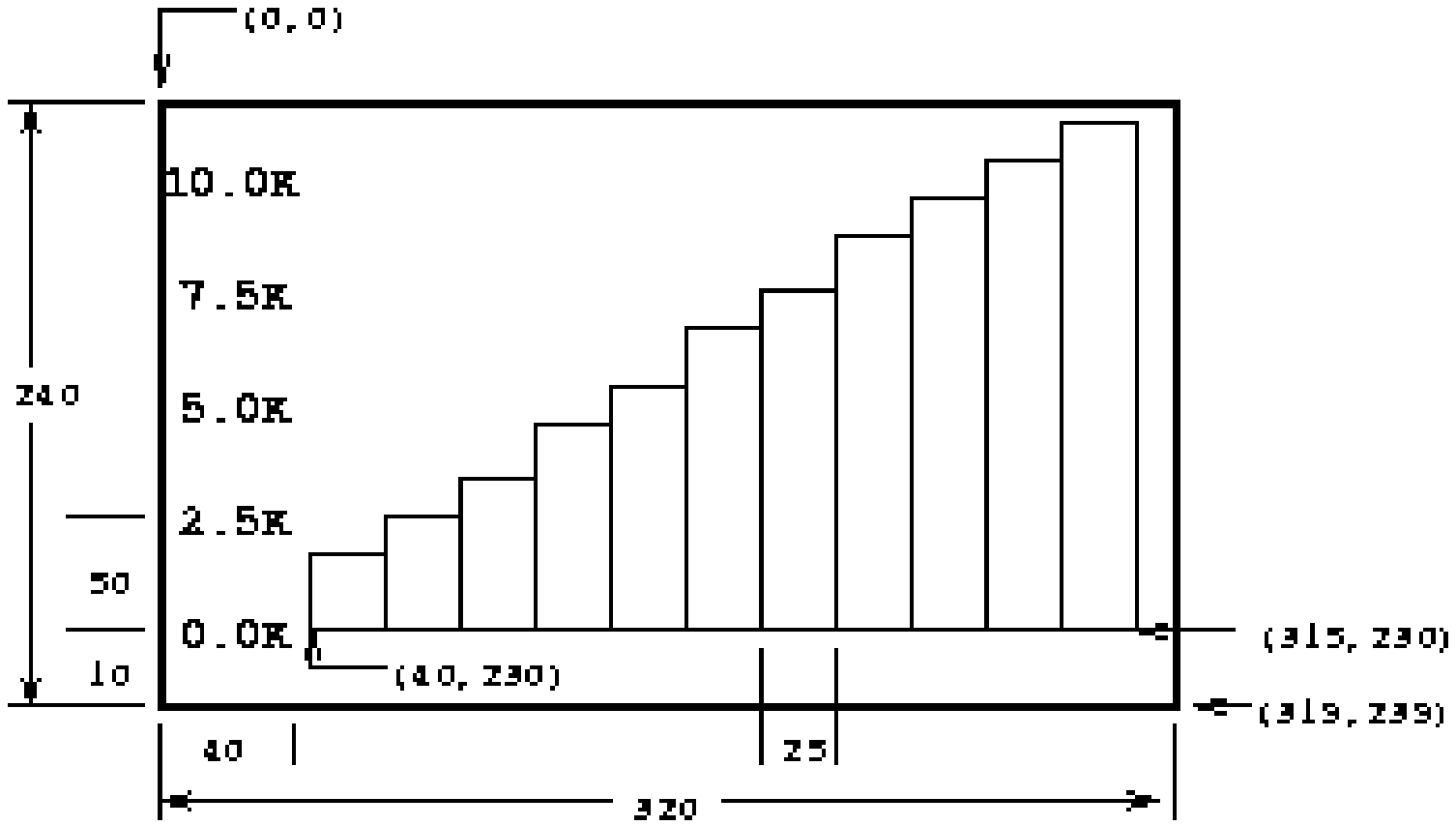
- The graphics library has a better solution. Graphical objects have a clone method that will make a copy of the object!

```
>>> # Better 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.
- 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*.
- 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`.
- 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

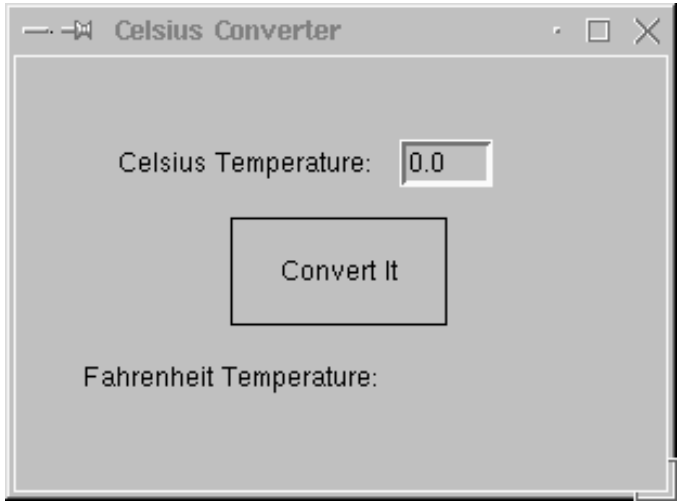
```
# triangle.pyw
# Interactive graphics program to draw a triangle
from graphics import *
def main():
    win = GraphWin("Draw a Triangle")
    win.setCoords(0.0, 0.0, 10.0, 10.0)
    message = Text(Point(5, 0.5), "Click on three points")
    message.draw(win)
    # Get and draw three vertices of triangle
    p1 = win.getMouse()
    p1.draw(win)
    p2 = win.getMouse()
    p2.draw(win)
    p3 = win.getMouse()
    p3.draw(win)
    # 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

- 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.
  - 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."
    ```

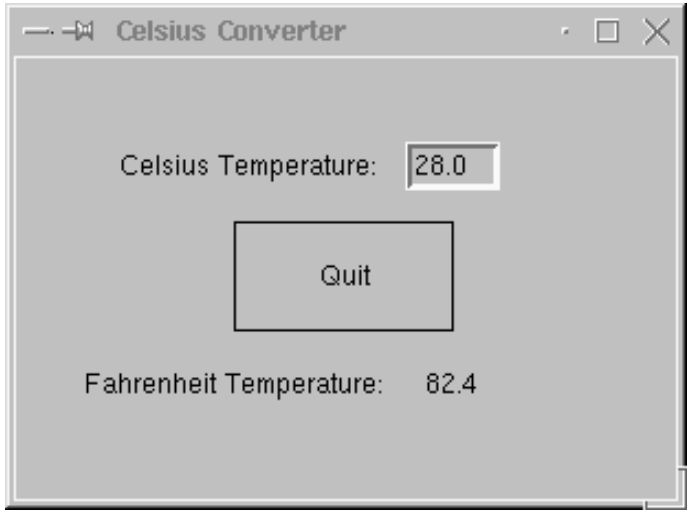
Handling Textual Input

- The triangle program's input was done completely through mouse clicks. 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

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

    # convert input
    celsius = float(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

- 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.
- 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.

Thank You