Please sit with your choice of a NEW robot partner

- Can be same old partner if you wish, but I suggest someone new so that you get to know more classmates
- Get a robot. If you need a new locker combination, just ask.

Check out today's projects from SVN:
Session08a-Files
Session08b-ObjectsAndGraphics

FILES, OBJECTS AND GRAPHICS

Outline

- Files
  - Review, with examples, of file *open/close* and *reading/writing*
  - Practice: *goRobotGo and wordCount* modules from *Session08a-Files*
- Two models of software design:
  - Procedural model
  - Object-oriented model
    - What is an object?
- Graphics
  - Creating and using objects
  - Interactive graphics
  - Coordinate systems
- Practice Objects and Graphics
  - *alienFace*, *clickMe* and *plotPoints* modules from *Session08b-ObjectsAndGraphics*

File Processing – Manipulating data stored on disk

- *Open* file
  - For reading or writing
  - Associates file on disk with a *file variable* in program
  - Examples:

```
inFile = open("blah", 'r')
outFile = open("foo", 'w')
```

Overwrites (!) the file if it exists, creates it if it doesn't.

- *Manipulate* file with operations on the file variable
  - *Read* or *write* information

See next slide for details

- *Close* file
  - Causes final "bookkeeping" to happen
  - Example: `inFile.close()`

Note: disks are slow, so writes to the file are often kept in a ***buffer*** in memory until we close the file or otherwise "flush" the buffer.

Questions about how to *open* and *close* a file?

Q1-3

Simplest example of writing to a file

```
def writeDataSimply(outputFilename, maxToWrite):
    ''' Writes 1 .. maxToWrite to the file with the given name.
     Puts a space after each number.'''

    outputFile = open(outputFilename, 'w')

    for k in range(1, maxToWrite + 1):
        outputFile.write(str(k) + " ")

    outputFile.close()
```

Open the file for writing

The *write* method takes a *string*.

Close the file when writing is finished

Questions about how to *write* to a file?

Q4

Simplest example of reading numbers from a file

```
def readDataSimply(inputFilename):
    ''' Reads the data in a file, which should be numbers separated
    by spaces and/or newlines.  Returns the sum of the numbers.'''

    inputFile = open(inputFilename, 'r')

    total = 0
    for line in inputFile:
        numbers = line.split()
        for number in numbers:
            total = total + eval(number)

    inputFile.close()
    return total
```

Open the file for reading

Each *line* in the file variable (here called *input*) is a *string*.
This loop goes through the file line by line.

Split the string at spaces, to get a *list* of strings.
Assumes data is separated by spaces.

For each *string* in the list, evaluate it. That converts it to a number.
Assumes that all the data items are numbers.

Close the file when reading is finished

Questions about how to *read* from a file?
There are other ways to read, but this pattern will do for now.

Q5-6

Practice at reading from a file

- Check your answers to Quiz problems 4 and 5 by comparing them to the:
  - *writeListToFile* and
  - *readDataIntoList*

  functions in the *fileReadingAndWritingExample* module of the *Session08a-Files* project that you checked out today

Robots – more practice at reading from a file

- Do the TODO's in the *goRobotGo* module from the *Session08a-Files* project that you checked out today.
  - First, with your instructor, review the TODO's in that file; they specify what you are to accomplish
  - Then, working with your robot partner:
    - One of you: implement the robot *turn* and *move* functions, as specified in the module. First review the description in the PyCreate documentation of the *go* and *stop* functions; they are what you will need, with a *sleep* for the right amount of time (which you'll have to calculate) in between.
    - The other: implement the file-handling and the calls to *turn* and *move*, as specified by the TODO's in the module
      - Use what you just learned in quiz Question 5 about reading numbers from a file.
      - To get 4 numbers from a line in the file, note how you solved a similar problem in quiz Question 6.
    - Whoever finishes first, help the other. Combine your work by emailing it to each other or whatever. Be sure that you list BOTH authors at the top of the file.
  - If you finish the *goRobotGo* problem, begin the rest of Homework 8.

Procedural versus object-oriented

- In the ***procedural model***, a program
  - is seen as a list of tasks (subroutines, functions) to perform
  - with each task itself broken down into subtasks, and so forth.
    - We call this ***procedural decomposition***.
- Many (most?) modern computer programs are built using an ***object-oriented (OO) model***, in which:
  - A program is viewed as a collection of ***interacting objects***
  - See next slide for definition of ***object***.

Both models are valuable. In this course, you will learn:

- how to apply the procedural model and
- how to *use* objects
  (with how to *design* objects left to CSSE 220).

There are other programming paradigms in addition to the two listed here, e.g. *functional programming* and *logic programming*.

What are objects?

- Data types for numbers and Boolean are *passive*
  - Each is a single piece of data. E.g. *108* or *False*.
  - Each is passive. You can do things to a number (like adding them), but numbers can't do things of themselves.
- An **object** is an *active* data type
  - **Knows stuff.** And thus can be an aggregate of stuff.
  - **Can do stuff.** And thus is active.
- Example of an object: the ***body*** is an object that has a brain, lung, hands that have fingers, …
  - And the body can ask its heart to beat, its finger to point, etc.

Q10

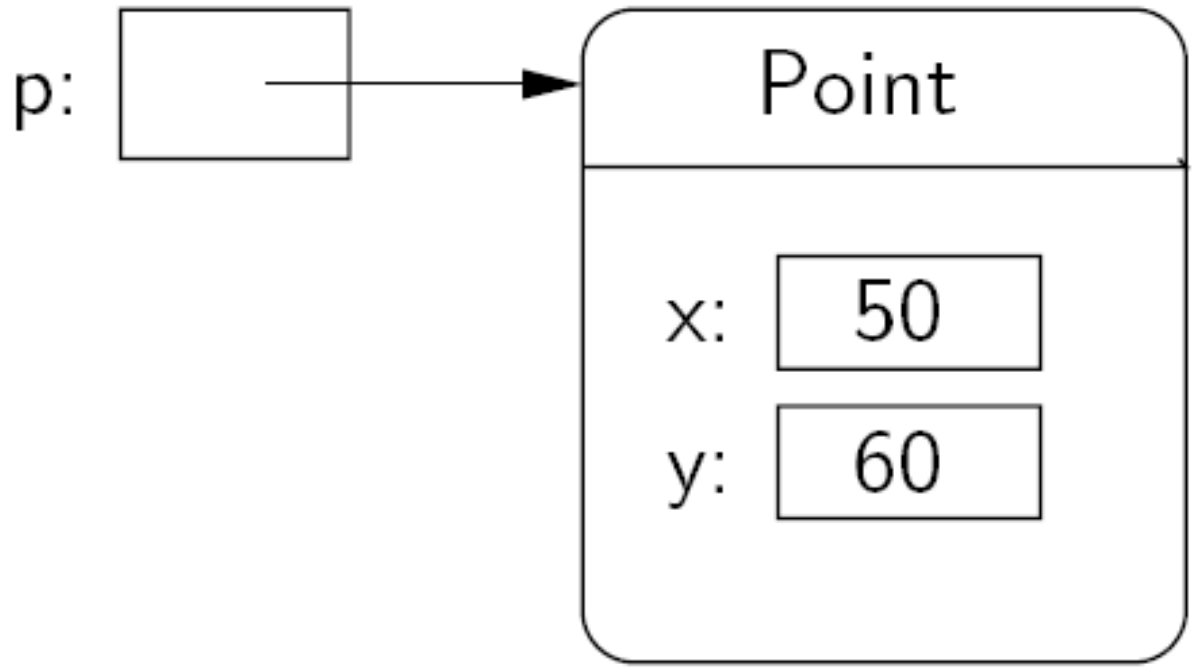
How do objects interact?

- Objects interact by sending each other messages
  - Message: request for object to perform one of its operations
  - Example: the brain can ask the feet to walk
  - In Python, messages happen *via* method calls.
- `>>> win = GraphWin()  # constructor`
- `>>> p = Point(50, 60)  # constructor`
- `>>> p.getX()  # accessor method`
- `>>> p.getY()  # accessor method`
- `>>> p.draw(win)  # method`

Q11

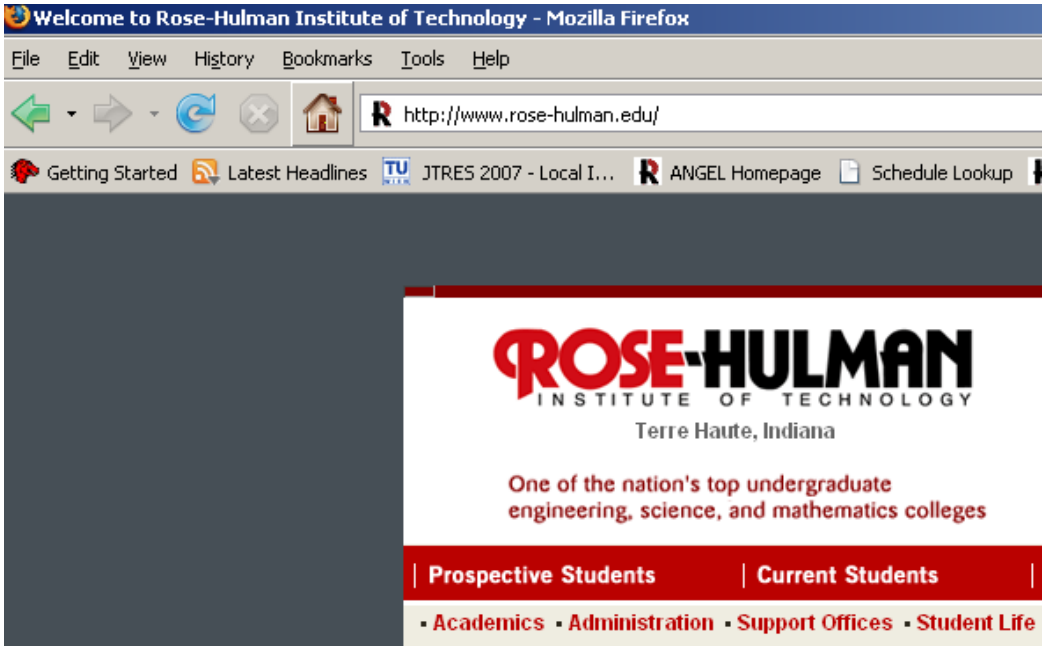
How do objects interact? Point

p = Point(50, 60)



Simple graphics programming

- Graphics is fun and provides a great vehicle for learning about objects
- Computer Graphics: study of graphics programming
- Graphical User Interface (GUI)



Q12

You choose how to import

- Must import the graphics library before accessing it

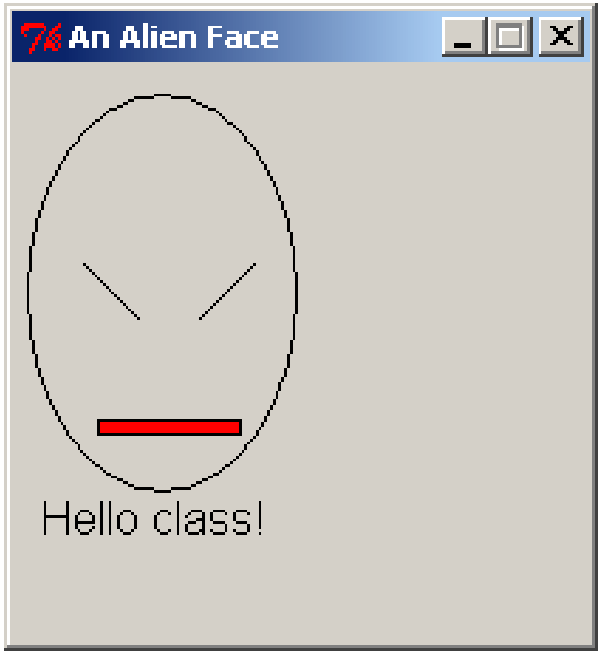
```
import zellegraphics
win = zellegraphics.GraphWin()
```

- Another way to import the graphics library

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

Using graphical objects

- Using different types of objects from the graphics library, draw the following alien face and message



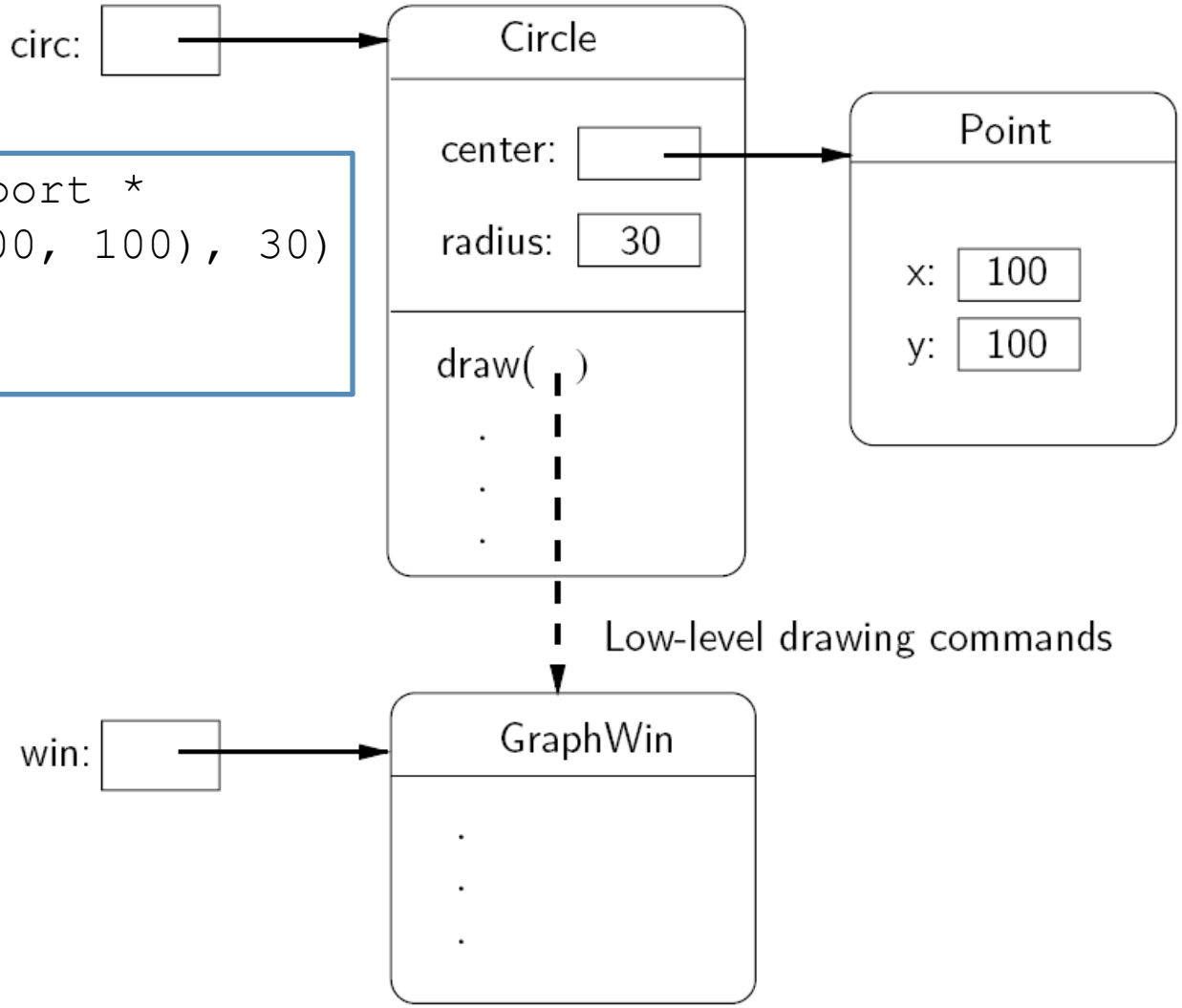
Q13

Class and object terminology

- Different types of objects
  - Point, Line, Rectangle, Oval, Text
  - These are examples of *classes*
- Different objects
  - head, leftEye, rightEye, mouth, message
  - Each is an *instance* of a class
  - Created using a *constructor*
  - Objects have *instance variables*
  - Objects use *methods* to operate on instance variables

Object interaction to draw a circle

```
from zellegraphics import *
circ = Circle(Point(100, 100), 30)
win = GraphWin()
circ.draw(win)
```



Interactive graphics

- *GUI*—Graphical User Interface
  - Accepts input
    - Keyboard, mouse clicks, menu, text box
  - Displays output
    - In graphical format
    - On-the-fly
- Developed using *Event-Driven Programming*
  - Program draws interface elements (*widgets*) and waits
  - Program responds when user does something

Q16

getMouse

- `win.getMouse()`
  - Causes the program to pause, waiting for the user to click with the mouse somewhere in the window
  - To find out where it was clicked, assign it to a variable:
    - `p = win.getMouse()`

Q17

Mouse Event Exercise

Together, lets' solve the following problem:

Create a program, `clickMe.py`, with a window labeled "Click Me!" that displays the message *You clicked (x, y)* the first 5 times the user clicks in the window.

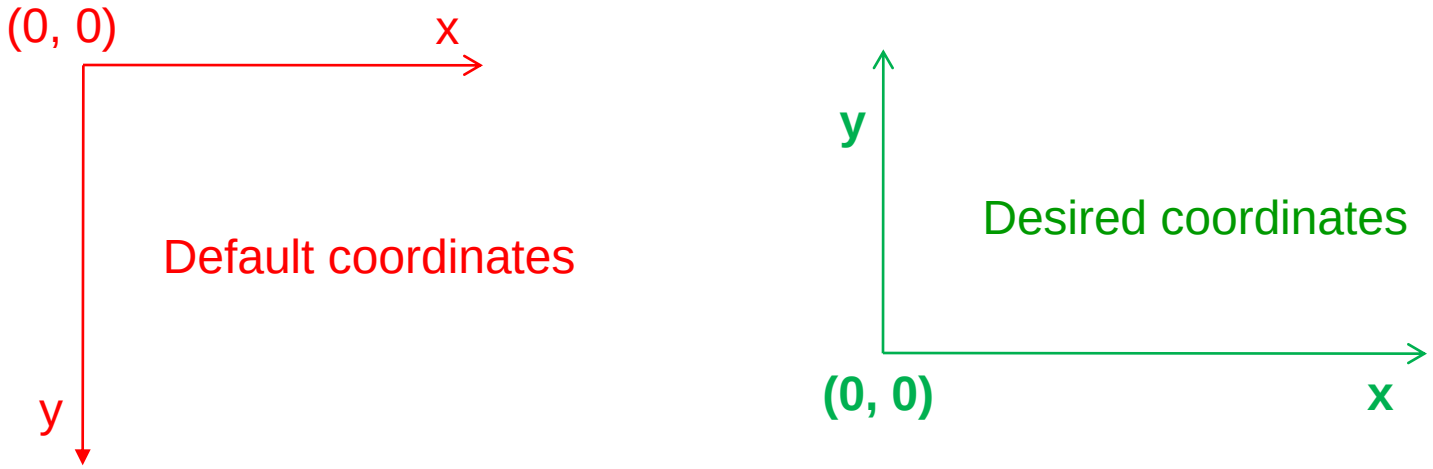
The program also draws a red-filled circle, with blue outline, in the location of each of these first 5 clicks.

The program closes the window on the 6th click

Q18

Coordinate systems

- An important use of graphics is to represent **data** visually
  - Example: a bar chart
- We really want (0,0) to be in the lower-left corner



Desired coordinate system

- `win.setCoords(x1, y1, x2, y2)` **method from** `GraphWin` **class**
  - Sets the coordinates of the window to run from (x1,y1) in the lower-left corner to (x2,y2) in the upper-right corner.