Comp 151

Using Objects
Admin

- How did that first project go?
- New project coming
Assignment
Overview of Objects

- 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 of Objects

- 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 your 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 lecture begins to use the `graphics.py` library supplied by your book. You can get it from your books website.
  - A direct link is available from the resources page of the class website.
- Two location choices
  - In Python’s Lib directory with other libraries
  - In the same folder as your graphics program
    - On campus we have to use the second
Simple Graphics Programming

- Since this is a library, we need to import the graphics commands
  ```python
  import graphics
  ```
- A *graphics window* is a place on the screen where the graphics will appear.
  ```python
  win = graphics.GraphWin()
  ```
- This command creates a new window titled “Graphics Window.”
Simple Graphics Programming

- It’s a pain 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.

- Be careful about going whole hog on the
  - `From <package> import *`
  - “namespace collisions”
Simple Graphics Programming

- Doing the import this way eliminates the need to preface graphics commands with `graphics`.
  ```python
  from graphics import *
  win = GraphWin()
  ```
Simple Graphics Programming

- A graphics window is a collection of points called \textit{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.
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

```python
>>> 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

```python
>>> ### 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 preformed 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

- 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

- \( p = \text{Point}(50, 60) \)
  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:
  \[
  \text{<object>.<method-name>(<param1>, <param2>, ...)}
  \]
Using Graphical Objects

- \texttt{p.getX()} and \texttt{p.getY()}
  - returns the \textit{x} and \textit{y} values of the point. Routines like these are referred to as \textit{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

- 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.

```python
>>> 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 called *aliasing* in your book.
Using Graphical Objects

Diagram showing a `Circle` object with properties `center` and `radius`, and a `Point` object with coordinates `(80, 50)`.
Using Graphical Objects

- There are two ways to get around this.
- We could make two separate circles, one for each eye:
  ```python
  >>> leftEye = Circle(Point(80, 50), 5)
  >>> leftEye.setFill('yellow')
  >>> leftEye.setOutline('red')
  >>> rightEye = Circle(Point(100, 50), 5)
  >>> rightEye.setFill('yellow')
  >>> rightEye.setOutline('red')
  ```
The graphics library has a better solution. Graphical objects have a clone method that will make a copy of the object!

```python
>>> # 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)
```
In the Graphics Library

- All of the shape classes support several methods

  - setFill(color)
    - Sets the interior of the object to the given color.

  - setOutline(color)
    - Sets the outline of the object to the given color.

  - setWidth(pixels)
    - Sets the width of the outline of the object to this many pixels. (Does not work for Point.)
Graphics Library functions II

- **draw(aGraphWin)**
  - Draws the object into the given GraphWin.

- **undraw()**
  - Undraws the object from a graphics window. This produces an error if the object is not currently drawn.

- **move(dx,dy)**
  - Moves the object dx units in the x-direction and dy units in the y-direction. If the object is currently drawn, the image is adjusted to the new position.

- **clone()**
  - Returns a duplicate of the object. Clones are always created in an undrawn state. Other than that, they are identical to the cloned object.
GraphWin

- The basic graphics window – some useful methods (more later)
- GraphWin(title, width, height)
  - Constructs a new graphics window for drawing on the screen. The parameters are optional, the default title is "Graphics Window," and the default size is 200 x 200.
- setBackground(color)
  - Sets the window background to the given color. The initial background is gray. See Section 5.8.5 for information on specifying colors.
- close()
  - Closes the on-screen window.
Using Graphical Objects

```python
>>> 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.
- Circle constructor takes two params
  - First param is Point
  - Second param is int for radius
- The last line is a request to the Circle object circ to draw itself into the GraphWin object win.
Rectangle Class

- We've seen circle before
- Remember all of the general method like setFill are also available
  - Rectangle(point1, point2)
    - Constructs a rectangle having opposite corners at point1 and point2.
  - getCenter()
    - Returns a clone of the center point of the rectangle.
Oval

- **Oval(point1, point2)**
  - Constructs an oval in the bounding box determined by point1 and point2.

- **getCenter()**
  - Returns a clone of the point at the center of the oval.
Polygon

- Arbitrary polygon built out of points
- Polygon(point1, point2, point3, ...)
  - Constructs a polygon having the given points as vertices. Also accepts a single parameter that is a list of the vertices.
Lines

- **Line(point1, point2)**
  - Constructs a line segment from point1 to point2.

- **setArrow(string)**
  - Sets the arrowhead status of a line. Arrows may be drawn at either the first point, the last point, or both. Possible values of string are 'first', 'last', 'both', and 'none'. The default setting is 'none'.

Assignment

- Now let's look at your assignment.