Introduction to Computer Programming for Non-Majors

CSC 2301, Fall 2015

Chapter 4

Guoliang Liu

The Department of Computer Science
Review of Chapter 3

• Numeric Data Types
  – Int (while numbers)
  – Float (with fractions)
• The data type of an object determines what values it can have and what operations can be performed on it.
• How can we tell \textit{which is which}?
  – A numeric literal without a decimal point produces an int value
  – A literal that has a decimal point is represented by a float (even if the fractional part is 0)
• Why do we need two number types?
  – Values that represent counts can’t be fractional (you can’t have 3 \( \frac{1}{2} \) quarters)
  – Most mathematical algorithms are very efficient with integers
  – The float type stores only an approximation to the real number being represented!
  – Since floats aren’t exact, use an int whenever possible!
Review of Chapter 3

- Operations on ints produce ints, operations on floats produce floats (except for //).

- Using the Math Library
  - Besides (+, -, *, /, //, **, %, abs), we have lots of other math functions available in a math library.
  - http://docs.python.org/library/math.html
  - A library is a module with some useful definitions/functions.
  - http://docs.python.org/library/
  - Import math
To access the `sqrt` library routine, we need to access it as `math.sqrt(x)`.

Using this dot notation tells Python to use the `sqrt` function found in the `math` library module.

To calculate the root, you can do:

```python
discRoot = math.sqrt(b*b - 4*a*c)
```
Understand numbers

• Understand the transformation between different number systems.

• Most common ones:
  – Binary, octal, decimal, hexadecimal.
  – How to change between them?
  – Functions to use:
    • `bin()`, `oct()`, `int()`, `hex()`
Cont… Type Conversion

• In *mixed-typed expressions* Python will convert ints to floats.
  – 3.0 + 5
• Sometimes we want to control the type conversion. This is called *explicit typing*.
• Different from round()
• Int(3.6) → 3
• Round(3.6) → 4
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.
Objectives (cont.)

- 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 (cont.)

- To be able to write simple interactive graphics programs using the graphics library.
What are objects?

- A kind of **Data Type** that programmer can define
- Objects combine data and operations together
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.
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.

• Get an basic understanding with objects.
• We will Learn objects further in chapters 10 and 12.
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)**. It’s based on **Tkinter**.
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
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

- *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.
• Doing the import this way eliminates the need to preface graphics commands with \texttt{graphics}.

\begin{verbatim}
>>> from graphics import *
>>> win = GraphWin()
\end{verbatim}
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.
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

- 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**.
  \[<\text{class-name}>\!(<\text{param1}>,<\text{param2}>,\ldots)\]

• \(<\text{class-name}>\) 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

```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.
• 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.

```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 (the same object) to the right eye which gets moved 20 units.
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`. 

`rightEye` and `leftEye` refer to the same circle!
Using Graphical Objects

```
lftEye:

Circle

  center:
  radius: 10

Point

  x: 80
  y: 50
rghtEye:
```
Using Graphical Objects

• How to generate two circles?
• 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!**

```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)
```
Reviews of questions

• How do you know you are creating an object (instance of a class), Point(x, y) vs print(x, y)? Which creates objects?
  – Find the class definition.
  – Whether it can be assigned to a variable
  – If it is an object, then can you use id() or hex(id()) to get its address.
  – Naming convention: A class is usually starts with uppercase letter, while function starts with lowercase.

• How many objects are created below?
  – (1) X= Point(10,10); Y= X;
  – (2) X= Point(10,10); Y= Point(10,10);
  – Note: use id() to see
Exercise

- Download and put graphics.py in your home python directory if necessary.
- Please refer to “graphics documentation” for function documentation on course website.
- Draw two circles and one square in a window
- Connect the center of two circles using a straight line
  - No submission needed.
  - For example:
Assignment 2

- Download and put graphics.py in your home python directory if necessary.
- Download assignment2 from the course website:
- Due: 1:00pm, Sep 17th, 2015.
- Upload to desire2learn assignment2 folder.