Introduction to Objects
Drawing using graphics.py

14 Oct 2010
CMPT140
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

What's on for today

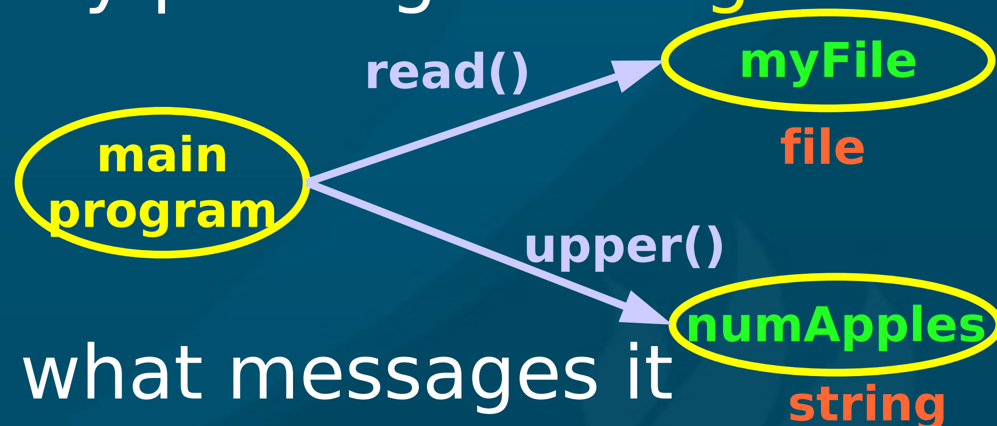
- Object-oriented programming
- Zelle's graphics.py library
  - Creating a window (GraphWin)
  - Creating objects and drawing them
- Mutators and accessors (set/get methods)
- Computing & Society essay paper
  - Topic due in 2.5 weeks
  - Paper due last day of our class

Object-oriented programming

- Procedural paradigm: “recipe” list of actions
  - Focus is on the procedures (verbs)
  - Variables, data structures get passed into procedures
  - **e.g.: string.capwords('hello')**
- Object-oriented paradigm: collections of objects
  - Focus is on the data (nouns)
  - Messages get passed between objects
  - Procedures are methods belonging to objects
  - **e.g.: 'hello'.upper()**

Everything is an object

- In object-orientation, all data are objects:
  - Variables, procedures, even libraries
- We make things happen by passing messages between objects
  - **myFile.read(16)**
  - **appleName.upper()**



- The object itself defines what messages it accepts: these are called its methods
  - e.g., files have read(), write(), etc. strings have upper(), len(), etc.

Methods and attributes

- Everything you can do with an object is encapsulated in its object definition
- Objects have attributes (local variables)
- Objects have methods (functions)
  - A collection of methods defines an interface
- Example: design an ADT for a Student:
  - Attributes: data stored with each Student
    - Name, ID#, phone #, GPA, course list, ….
  - Methods: operations involving a Student:
    - Register for course, change major, call dad for $$, …

Classes and instances

- We define (declare) object classes (types).
  A class is a user-defined type, containing:
  - Attributes: data stored in each object
  - Methods: operations involving the object
    - Constructor method: sets up new object (“factory”)
    - Destructor method: destroys an object cleanly
- Instantiate a new object using the constructor:

```
joe = Student()
```

- e.g.: joe is a variable of type Student
  - joe is the instance; Student is the class

Zelle's graphics.py library

- Download from textbook resources (or our class python examples directory)
  - Put in same directory as your Python code
- Import everything in the library (not ideal...):

```
from graphics import *
```

- Instantiate a new window object:

```
win = GraphWin("My Window", 400, 300)
```

  - Constructor func makes new window objects
  - Parameters to the constructor for GraphWin: window title, width (px), height (px)

Classes in graphics.py

- GraphWin: represents a window for drawing
- Point: represents a *(x,y)* location (units: pixels)
  - Can also be drawn as a dot in the window
- Circle: takes a Point for centre, and a radius
- Rectangle: takes two Points (opposite corners)
- Oval: takes two Points (like Rectangle)
- Line: takes two Points
- Text: takes a centre Point, plus a string

Drawing objects

- Create a Point and a Circle:

```
pt1 = Point(100, 100)
cir = Circle(pt1, 50)
```

- Draw an object by calling its .draw() method:

```
cir.draw(win)            # specify which window
```

- Methods are functions that belong to an object
- Change colours before drawing:

```
cir.setOutline('green')      # line colour
cir.setFill('blue')          # inside colour
cir.setWidth(4)              # line width
```

Copy vs. alias

- Objects are mutable, so regular assignment makes an alias (reference to the same object):

```
cir = Circle(pt1, 50)
cir2 = cir              # cir2 is an alias of cir
cir.move( 50, 0 )       # move 50px to the right
```

  - cf. objects are passed by reference
- To make a separate copy, use .clone() method:

```
cir2 = cir.clone()
cir.move( 50, 0 )       # doesn't affect cir2
```

Writing your own classes

- Let's make a class representing a clock:
  - Specify a centre point, a radius, and angles for the big and little hands
    → these are the attributes
  - Draw the clock using a Circle and 2 Lines
    → so we want to provide a .draw() method
  - Use this class to instantiate many clocks
- Class design: name of the class, purpose,
  - Attributes (local variables), and
  - Methods (functions/operations)

Defining a class: constructor

- Declare a class using the 'class' keyword:

```
class Clock:
    def __init__( self ):
        self.centre = Point(0,0)
        self.radius = 0.
        self.hrAng = 0.
        self.mnAng = 0.
```

- __init__ is Python's name for the constructor
  - Every method has 'self' as first argument: refers to current object
- Constructor assigns default values to attributes

Constructor with parameters

- We can pass parameters to the constructor:
  - Define initial values of attributes:

```
class Clock:
    def __init__( self, c, r, h, m ):
        self.centre = c
        self.radius = r
        self.hrAng = h
        self.mnAng = m
```

- Instantiate specifying centre, radius, etc.:
  - `myClock = Clock( Point(100,100), 7, 0, 0 )`
- This now requires 4 parameters to constructor

Default parameters

- Functions may have default parameters:
  - **def double_me(x=0):**
    - **return x*2**
- Can call double_me() with 0 or 1 parameters:
  - **double_me() → returns: 0**
- Apply this to the constructor:

```
class Clock:
    def __init__( self, c, r=0., h=0., m=0. ):
        self.centre = c
        self.radius = r
        self.hrAng = h
        self.mnAng = m
```

Default params evaluate once

- Default values are evaluated once at declarat'n

```
def __init__(self, center=Point()):      # wrong!
```

  - This uses one shared Point object (alias) as the default center for every Clock!

- Use None as the default value, and instantiate a new object as the default value at run time:

```
def __init__( self, c=None, r=0., h=0., m=0. ):
    if c == None:
        c = Point(0,0)
    self.center = c
```

Listing all entities in a class

- Special Python attribute '__dict__'
- Dictionary of all entities in the object
  - For module: lists all methods, constants, etc. __module__, __doc__ (docstring)
    - import math
    - math.__dict__
    - Clock.__dict__
  - For object: lists all attributes

    myClock.__dict__: { 'center':<Point>, 'radius':7., 'hrAng': 0., 'mnAng': 0. }

Computing & Society Paper

- Computing scientist as **G**odly **C**hristian **L**eader:
  - Not just knowledge about tools, but
  - Wisdom of how to use tools
    - To serve others and
    - To give glory to God
- Write a short essay on a topic of your choosing about computers and society:
    - ~ 5 pages typed double-spaced 12pt 1in margins
    - Submit half-page topic by Tues 2 Nov
    - Paper due last day of our class (Tues 7 Dec)
      - Electronic submission (email, myCourses)

Sample paper topics

- Censorship and free speech
  - Pornography, gambling, hate groups, etc.
- Blogs: effect on politics, social interaction, etc.
- Artificial intelligence: the nature of sentience
- Violence in video games (Columbine etc.)
- Privacy: online banking, ID theft, etc.
- File sharing: BitTorrent, etc.
- Online dating (e.g. eHarmony): pros/cons
- Equity of access / rural digital divide
  - ...... or come up with your own topic!

Essay / Position Paper

- Your essay should be a position paper:
  - Topic should have at least two sides (e.g. pro/con)
  - You should state (in the introductory paragraph) what your position is (thesis)
  - You should have at least 2-3 points, each, both for and against your position
    - It is not necessary to rebut every point that contradicts your position:
    - Be honest about faults/limitations of your thesis
  - Summary intro/conclusion paragraphs
  - Proper English (spelling, grammar) is important!