Objects and Classes

APRIL 23RD 2012
Object-Oriented Programming: Example

• Suppose your program needs to maintain millions of polygons.
• This is something that graphics programs might have to do because complicated scenes are often constructed using polygons.
• Each polygon has a number of attributes:
  ○ Number of points (vertices) in the polygon,
  ○ List of the vertices in the polygon in clockwise order,
  ○ Colors of the vertices and colors of the line segments (edges) connecting consecutive vertices,
  ○ Whether the interior is transparent or not....
An object-oriented programming language allows us to package all of these attributes of a polygon together into an object.

We could then also define functions (or methods) that operate on the polygon object.

For example:
- deleteVertex, addVertex
- rotatePolygon, translatePolygon,
- ...
We have already seen examples of built-in objects in Python: strings, lists, etc.

**Example:**

```python
L = [3, 2, 9]
L.append(10)
```

This defines an *instance* of an object called L of *class* list. Then it applies the *method* `append` to L.

L is a “package” consisting of the list items along with other information about the list (e.g., its length).
Is this just new jargon for stuff you know already?

- To some extent, the answer is yes.
- Specifically:
  - class = data type,
  - object = variable,
  - method = function
- So by defining a class, you are essentially extending the language by defining a new data type.
- **Example:** By defining a class called `polygon` you have created a new data type called `polygon`. You can then objects (variables) of class (type) `polygon`.
Motivation

- Efficiency, with respect to running time and memory usage is one important focus of programmers.
- Another important focus is *maintainability*.
- As software sizes grow into millions of lines (e.g., Microsoft Windows OS) of code we want to ensure:
  - Smooth transition from one version to the next
  - Smooth transition when software engineers leave the project and new engineers join the project
- Object-oriented programming is one approach to programming in a disciplined manner.
Motivation

- By defining the class `polygon` and methods that operate on instances of the `polygon` class, you are making a commitment that:
  - Objects of the polygon class can be accessed using a certain syntax (e.g., `P.deleteVertex(q)`).
  - The methods have certain specified behaviors.
- The internal implementation of the class might change a lot over time, but the *interface* and external behavior remains largely static.
- This means that other code that depends on the `polygon` class will not suddenly stop working because the `polygon` class is now behaving differently.
Objects, classes, etc. as a formal notion in programming were introduced in the 1960s in a programming language called Simula 67.

*SmallTalk* was designed in 1970s at Xerox Parc and it refined the notions introduced in Simula 67.

In the 90s, object-oriented programming reached a wide audience with the introduction of *C++* and then *Java*.

Object oriented programming languages seem nicely suited for programming Graphics User Interfaces (GUIs). With the rise of GUIs, object-oriented programming languages continued to stay popular.

Now we have “hybrid” languages such as Python that allow different styles of programming (procedural, functional, object-oriented, etc.)
Defining a class in Python: Example 1

- I want to define a class called `point`.
- Each instance of this class is an object representing a point in 2-dimensional Euclidean space.
- I would like to use it as follows:
  ```python
  p = point(10, 15)
  p.translateX(5)
  ```
- In this example, we create an instance of the `point` class called `p`, whose x-coordinate is 10 and whose y-coordinate is 15.
- Then we apply the `translateX` method to this point to horizontally translate it 5 units.
Defining the `point` class

- The definition of the class starts with the header:
  ```python
class point():
  ```
- In the *body* of the class we will define a bunch of methods.
- One of these methods is special – it is called an *initializing method* or a *constructor* for the class.
# Definition of the point class

class point():

    # This is the initializing method or constructor for the class.  
    # Most classes will have one or more constructor methods.  
    # Examples: p = point(5, 7) will call this method to construct  
    # an instance p of the point class.  
    def __init__(self, a, b):
        self.x = a  
        self.y = b
The name of method is __init__. You will use the same name no matter what class you are defining.

It takes three parameters:
- self: this is a mandatory first parameter that refers to the object being constructed.
- a, b: these values are stored as the x-coordinate and y-coordinate of the point being constructed.

In the class methods we can use self.x and self.y to refer to coordinates of the point on which the method is being applied.
Two “translate” methods

# This translates the point horizontally by a units.  
# This is called as: p.translateX(20)

def translateX(self, a):
    self.x = self.x + a

# This translates the point vertically by a units.  
# This is called as: p.translateY(-10)

def translateY(self, a):
    self.y = self.y + a
# This returns the Euclidean distance between current point
# and the point given as an argument.
# This is called as: p.distance(point(10, 15))

def distance(self, p):
    return math.sqrt((self.x - p.x)**2 + (self.y - p.y)**2)