Computer Science 1 — CSci 1100
Lecture 21 — Classes, Part 1

Overview

- Define our own types and associated functions
- Encapsulate data and functionality
- Raise the "level of abstraction" in our code
- Make code easier to write and test
- Reuse code

Potential Examples

In each of these, think about what data you might need to store to represent the "object" and what functionality you might need to apply to the data.

- Date
- Time
- Point
- Rectangle
- Student
- Animal
- Molecule

An Example from an Earlier Homework

- Think about how difficult it was to keep track of the information about each restaurant in HW 3.
- You had to:
  - Remember the indices of (a) the restaurant name, (b) the latitude and longitude, (c) the type of restaurant, (d) the address, etc.
  - Form a separate list inside the list for the ratings.
  - Write additional functions to exploit this information
- If we used a class to represent each restaurant:
  - All of the information about the restaurant would be stored and accessed as named attributes
  - Information about the restaurants would be accessed through functions that we write for the class.

Point2d Class

- Simplest step is to just tell Python that `Point2d` will exist as a class, deferring the addition of information until later.

```
class Point2d(object):
    pass
```

- The Python reserved word `pass` says that this is the end of the class definition.
  - We will not need this later when we put information into the class.

Attributes

- Classes do not get interesting until we put something in them.
- The first thing we want is variables so that we can put data into a class.
  - In Python these variables are often called *attributes*.
  - Other languages call them *member variables*.
- We will see three different ways to specify attributes.

Assigning Attributes to Each Instance

- Points have an `x` and a `y` location, so we can write, for example,

```
p = Point2d()
p.x = 10
p.y = 5
dist_from_origin = sqrt(p.x**2 + p.y**2)
```

- We have to do this for each class instance.
- This is prone to mistakes:
  - Could forget to assign the attributes
  - Could accidentally use different names for what is intended to be the same attribute.
- Example of an error

```
q = Point2d()
q.x = -5
dist_from_origin = sqrt(q.x**2 + q.y**2)    # q.y does not exist
```

Defining the Attributes Inside the Class

- The simplest way to make sure that all variables that are instances of a class have the appropriate attributes is to define them inside the class.
- For example, we could redefine our class as

```
class Point2d(object):
    x = 0
    y = 0
    pass
```

- All instances of `Point2d` now have two attributes, `x` and `y`, and they are each initialized to 0.
- We no longer need the `pass` because there is now something in the class.

Defining the Attributes Through An Initializer / Constructor

- We still need to initialize `x` and `y` to values other than `0`:

```
p = Point2d()
p.x = 10
p.y = 5
```

- What we'd really like to do is initialize them at the time we actually create the `Point2d` object:

```
p = Point2d(10,5)
```

- We do this through a special function called an *initializer* in Python and a *constructor* in most other programming languages.
- Inside the class this looks like

```
class Point2d(object):
    def __init__( self, x0, y0 ):
        self.x = x0
        self.y = y0
```

- Our code to create the point now becomes

```
p = Point2d(10,5)
```

- Notes:
  - Python uses names that start and end with two `'_'` to indicate functions with special meanings. We'll see more soon.
  - The name `self` is special notation to indicate that the object itself is passed to the function.
- If we'd like to initialize the point to $(0, 0)$ without passing these values to the constructor every time then we can specify default arguments

```
class Point2d(object):
    def __init__( self, x0=0, y0=0 ):
        self.x = x0
        self.y = y0
```

allowing the initalization

```
p = Point2d()
```

Methods — Functions Associated with the Class

- We create functions that operate on the class objects inside the class definition:

```
import math

class Point2d(object):
    def __init__( self, x0, y0 ):
        self.x = x0
        self.y = y0

    def magnitude(self):
        return math.sqrt(self.x**2 + self.y**2)

    def dist(self, o):
        return math.sqrt( (self.x-o.x)**2 + (self.y-o.y)**2 )
```

these are called *methods*

- This is used as

```
p = Point2d(0,4)
q = Point2d(5,10)
len = q.magnitude()
print "Magnitude %2f" %len
print "Distance is %2f", %p.dist(q)
```

- Note that with the above definition of `Point2d`, the following is illegal

```
q = Point2d(5,10)
len = magnitude(q)
print "Magnitude %2f" %len
```

Exercise

1. Another special method/function is `__str__` which converts an object to a string. Write method `__str__` for the `Point2d` which produces the following

```
>>> q = Point2d(5,10)
>>> print str(q)
(5,10)
```

2. Write a new `Point2d` method called `scale` that takes as an input argument a single value and multiples both the `x` and `y` attributes by this value.

Classes and Modules

- Each class should generally be put into its own module, or several closely-related classes should be combined in a single module.
- During lecture we will place the code for `Point2d` into its own module.
- Doing so is good practice for languages like C++ and Java, where classes are placed in separate files.
- Testing code can be included in the module or placed in a separate module.

Operators and Other Special Functions

- We'd like to write code that uses our new objects in the most intuitive way possible.
- For our point class, this involves use of operators such as

```
p = Point2d(1,2)
q = Point2d(3,5)
r = p+q
s = p-q
t = -s
```

- Notice how in each case, we work with the `Point2d` variables (objects) just like we do with int and float variable (objects).
- We implement these by writing the special functions `__add__`, `__sub__`, and `__neg__`
- For example, inside the `Point2d` class we might have

```
def __add__(self,other):
     return Point2d(self.x + other.x, self.y+other.y)
```

- Similarly, we can define boolean operators such as `==` and `!=` through the special functions `__eq__` and `__neq__`

Exercise

1. Write the implementation of `__sub__` for `Point2d`
2. Write the implementation of `__neg__` for `Point2d`
3. Write the implementation of `__mul__` for `Point2d`
    - This function should be like the `scale` function, except it should create a new object instead of modifying an existing one.
4. Write the implementation of `__eq__` for `Point2d`

When to Modify, When to Create New Object

- Some methods, such as `scale`, modify a single `Point2d` object
- Other methods, such as our operators, create new `Point2d` objects without modifying existing ones.
- The choice between this is made on a method-by-method basis by thinking about the meaning — the *semantics* — of the behavior of the method.

Programming Conventions

- Don't create attributes outside the class.
- Don't directly access or change attributes except through class methods.
    - Languages like C++ and Java have constructions that enfore this.
    - In languages like Python it is not a hard-and-fast rule.
- Class design is often most effective by thinking about the required methods rather than the required attributes.
    - As an example, we rarely think about how the Python `list` and `dict` classes are implemented.

Time Example

- In the remainder of the lecture, we will work through an extended example of a `Time` class
- By this, we mean the time of day, measured in hours, minutes and sections.
- We'll brainstorm some of the methods we might need to have.
- We'll then consider several different ways to represent the time internally:
    - Hours, minutes and seconds
    - Seconds only
    - Military time
- Despite potential internal differences, the methods — or at least the way we call them — will remain the same
    - This is an example of the notion of *encapsulation*, which we will discuss more in Lecture 22
- At the end of lecture, the resulting code will be posted and exercises will be generated to complete the class definition.

Summary

- Define new types in Python by creating classes
- Classes consist of *attributes* and *methods*
- Attributes should be defined and initialized through the special method call `__init__`. This is a *constructor*
- Other special methods allow us to create operators for our classes.
- We looked at a *Point2d* and *Time* example.

Practice Problem

Create a `TimeOfDay` class that provides the following methods:

- `__init__` from an initial number of seconds, minutes and hours, in that order (this is a different order from what was suggested during lecture), with initial values of each being 0. In other words, the default time is midnight.
- `__add__` which adds two `TimeOfDay` objects
- `get_hour`, which returns the current hour, in the range (1-12)
- `get_minute`, which returns the current hour.
- `is_am` which returns `True` if and only if the time is before noon.
- `__str__` which converts the time to hours, minutes and seconds, with a label of `am` or `pm`. This is actually a bit of a challenge to make look right, and far beyond what we might ask you on Test 3.

Write code to test each. Many more methods can be added, and we will in Lecture 22, but this is enough for now. The `TimeOfDay` class should have just one attribute, which is the number of seconds in the day since midnight. This attribute's value will be in the range 0 to $3,600 \times 24$.