20. Introduction to Classes

Topics:
- Class Definitions
- Constructors
- Example: The class Point
- Functions that work with Point Objects
- Defining methods

What a Simple Class Definition Looks Like

class Point:
    ""
    Attributes:
        x: float, the x-coordinate of a point
        y: float, the y-coordinate of a point
    ""
    def __init__(self, x, y):
        self.x = x
        self.y = y
    
A class can be used to "package" related data.

One Reason for classes: They Elevate the Level Thinking

>>> P = Point(2,1)
>>> Q = Point(6,4)
>>> d = P.Dist(Q)
>>> print d
5

Here, Dist is a method and P.Dist(Q) says "compute and return the distance from point P to point Q."

Classes and Types

Recall that a type is a set of values and operations that can be performed on those values.

The four basic "built-in" types:
- int, float, str, bool

Classes are a way to define new types.

Examples

By suitably defining a rectangle class, we could say something like

```python
if R1.intersect(R2):
    print 'Rectangles R1 and R2 intersect'
```
Examples

By suitably defining a polynomial class, we could perform operations like

\[ p = q + r \]

where \( q \) and \( r \) are polynomials that are added together to produce a polynomial \( p \).

How to Define a Class

A Point Class

```python
class Point(object):
    
    Attributes:
    x: float, the x-coordinate of a point
    y: float, the y-coordinate of a point
    
def __init__(self, x, y):
        self.x = x
        self.y = y
```

This special function, called a constructor, does the packaging.

A Point Class

```python
class Point(object):
    
    Attributes:
    x: float, the x-coordinate of a point
    y: float, the y-coordinate of a point
    
def __init__(self, x, y):
        self.x = x
        self.y = y
```

The name of this class is "Point"

The "__init__" Function

```python
def __init__(self, x, y):
    
    Attributes:
    x: float, the x-coordinate of a point
    y: float, the y-coordinate of a point
    
    PreC: x and y are floats
    
    self.x = x
    self.y = y
```

That's a double underscore: `__init__`
The "__init__" Function

```python
def __init__(self, x, y):
    """ Creates a Point object
    PreC: x and y are floats
    """
    self.x = x
    self.y = y
```

"self" is always the first argument for any method defined in a class.

Calling the Constructor

```
>>> a = 3
>>> b = 4
>>> Q = Point(a, b)
```

This creates a `Point` object.

The constructor returns a reference, in effect, the red arrow.
**Objects: The Folder Metaphor**

In the office, manila folders organize data.

Objects organize data.

A point object houses float variables $x$ and $y$, called the attributes, where $(x,y)$ is the point.

**A point object houses float variables $x$ and $y$, called the attributes, where $(x,y)$ is the point.**

**Visualizing a Point Object**

Accessing Attributes

```
>>> a = 3
>>> b = 4
>>> Q = Point(a,b)
```

$Q$ is a variable and can "show up" in all the usual places, i.e., in an assignment statement.

```
Q.x = Q.x + 5
```

Seems that we can print an object!
The "__str__" function

```python
def __str__(self):
    return '(%6.3f,%6.3f)' %(self.x,self.y)
```

This "double underscore" function is part of the class definition.
Whenever a statement like

```python
print P
```

is encountered, then P is "pretty printed" according to the format rules.

Two Examples

A function that returns a Point Object:

```python
RandomPoint(Lx,Rx,Ly,Ry)
```

A function that has input parameters that are Point objects:

```python
Midpoint(P,Q)
```

Two Examples

A function that returns a Point Object:

```python
RandomPoint(Lx,Rx,Ly,Ry)
```

A function that has input parameters that are Point objects:

```python
Midpoint(P,Q)
```

Computing a Random Point

```python
def RandomPoint(Lx,Rx,Ly,Ry):
    """ Returns a point that is randomly chosen from the square Lx<=x<=Rx, Ly<=y<=Ry.
    PreC: Lx and Rx are floats with Lx<Rx
    Ly and Ry are floats with Ly<Ry
    """
    x = randu(Lx,Rx)
    y = randu(Ly,Ry)
    P = Point(x,y)
    return P
```

calling the constructor

Computing a Midpoint

```python
def Midpoint(P1,P2):
    """ Returns a point that is the midpoint of the line segment that connects P1 and P2.
    PreC: P1 and P2 are points.
    """
    xm = (P1.x + P2.x)/2.0
    ym = (P1.y + P2.y)/2.0
    Q = Point(xm,ym)
    return Q
```

calling the constructor

Computing a Midpoint

```python
def Midpoint(P1,P2):
    """ Returns a point that is the midpoint of the line segment that connects P1 and P2.
    PreC: P1 and P2 are points.
    """
    xm = (P1.x + P2.x)/2.0
    ym = (P1.y + P2.y)/2.0
    Q = Point(xm,ym)
    return Q
```

calling the constructor

Methods

Methods are functions that are defined inside a class definition.
We have experience using them with strings

```python
s.upper(),s.find(s1),s.count(s2),
s.append(s2), s.split(c), etc
```

and lists

```python
L.append(x),L.extend(x),L.sort(),etc
```
Methods

Now we show how to implement them.

We will design a method for the Point class that can be used to compute the distance between two points.

It will be used like this:

\[
\text{delta} = P.\text{Dist}(Q)
\]

A Point Class Method: Dist

class Point(object):
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def Dist(self,other):
        """Returns distance from self to other.
        PreC: other is a point
        """
        dx = self.x - other.x
        dy = self.y - other.y
        d = sqrt(dx**2+dy**2)
        return d

Using the Dist Method

Let's create two point objects and compute the distance between them. This can be done two ways...

```python
>>> P = Point(3,4)
>>> Q = Point(6,8)
>>> deltaPQ = P.Dist(Q)
>>> deltaQP = Q.Dist(P)
>>> print deltaPQ, deltaQP
5.0 5.0
```

Method Implementation:

Syntax Concerns

class Point(object):
    def Dist(self,other):
        """Returns distance from self to other.
        PreC: other is a point
        """
        dx = self.x - other.x
        dy = self.y - other.y
        d = sqrt(dx**2+dy**2)
        return d

How to Think “Method”

Think like this: "We are going to apply the method dist to a pair of Point objects, self and other."

Method Implementation:

Syntax Concerns

class Point(object):
    def Dist(self,other):
        """Returns distance from self to other.
        PreC: other is a point
        """
        dx = self.x - other.x
        dy = self.y - other.y
        d = sqrt(dx**2+dy**2)
        return d

Two Facts:

Indentation is important.
A class method is part of the class definition.
Let's see what happens when we execute the following:

\[
P = \text{Point}(3,4) \\
Q = \text{Point}(6,8) \\
D = P.\text{Dist}(Q)
\]

```
class Point(object):
    def Dist(self, other):
        """ Returns distance from self to other. """
        dx = self.x - other.x
        dy = self.y - other.y
        d = sqrt(dx**2 + dy**2)
        return d
```

Think of \texttt{self} and \texttt{other} as input parameters.
Visualizing a Method Call

\[ P = \text{Point}(3, 4) \]
\[ Q = \text{Point}(6, 8) \]
\[ D = P.\text{Dist}(Q) \]

\[ dx = \text{self}.x - \text{other}.x \]
\[ dy = \text{self}.y - \text{other}.y \]
\[ d = \sqrt{(dx^2 + dy^2)} \]
\[ \text{return } d \]

dx: -3 dy: -4 self other

P
Q
Dist
Point
Point
\[
\begin{array}{cc}
3 & 6 \\
4 & 8 \\
\end{array}
\]

Visualizing a Method Call

\[ P = \text{Point}(3, 4) \]
\[ Q = \text{Point}(6, 8) \]
\[ D = P.\text{Dist}(Q) \]

\[ dx = \text{self}.x - \text{other}.x \]
\[ dy = \text{self}.y - \text{other}.y \]
\[ d = \sqrt{(dx^2 + dy^2)} \]
\[ \text{return } d \]

d: 5 dx: -3 dy: -4 self other

P
Q
Dist
Point
Point
\[
\begin{array}{cc}
3 & 6 \\
4 & 8 \\
\end{array}
\]

Checking Things Out

```python
>>> P1 = RandomPoint(-10,10)
>>> P2 = RandomPoint(-10,10)
>>> M = Midpoint(P1,P2)
>>> print M.Dist(P1)
4.29339610681
>>> print M.Dist(P2)
4.29339610681
```

Summary: Base Types vs Classes

**Base Types**
- Built into Python
- Instances are values
- Instantiate w/ Literals
- Immutable

**Classes**
- Defined in Modules
- Instances are objects
- Instantiate w/ constructors
- Mutable
A Note on Copying an Object

There is a difference between creating an alias and creating a genuine second copy of an object.

This Does Not Create a Copy…

```python
>>> Q = Point(3,4)
>>> P = Q
```

It creates an alias, not a copy.

This Does Create a Copy…

```python
>>> Q = Point(3,4)
>>> P = Point(Q.x, Q.y)
```

And This Also Creates a Copy…

```python
>>> Q = Point(3,4)
>>> P = copy(Q)
```

The function `copy` must be imported.

The Module `copy`

```python
from copy import copy
```

Import this function and use it to make copies of objects.

`deepcopy` is another useful function from this module—more later.

Using `copy`

```python
>>> Q = Point(3,4)
>>> P1 = copy(Q)
>>> P1.x = 5
>>> print P1
( 5.000, 4.000)
>>> print Q
( 3.000, 4.000)
```

We are modifying `P1`, but `Q` remains the same.
Methods vs Functions

It is important to understand the differences between methods and functions, i.e., how they are defined and how they are invoked.

A Function that Returns the Distance Between Two Points

```
def Dist(P1,P2):
    """ Returns the distance from P1 to P2.
    PreC: P1 and P2 are points
    """
    d = sqrt((P1.x-P2.x)**2+(P1.y-P2.y)**2)
    return d
```

Methods and (Regular) Functions

```
def Dist(self,other):
    dx = self.x - other.x
    dy = self.y - other.y
    D = sqrt(dx**2+dy**2)
    return D
```

```
def Dist(P,Q):
    dx = P.x - Q.x
    dy = P.y - Q.y
    D = sqrt(dx**2+dy**2)
    return D
```

```
>>> P = Point(3,4)
>>> Q = Point(6,8)

>>> P.Dist(Q)
5.0
```

```
>>> P = Point(3,4)
>>> Q = Point(6,8)

>>> Dist(Q,P)
5.0
```