18. Lists of Objects

Topics:

Example: The class Disk
Boolean-Valued Methods
A Disk Intersection Problem
Example: The class CountyPop
Representing census-related data
Sorting a list of CountyPop objects
We would like to assemble a list whose elements are not numbers or strings, but references to objects.

For example, we have a hundred points in the plane and a length-100 list of points called ListOfPoints.

Let’s compute the average distance to (0,0).
Working with a List of Point Objects

Origin = Point(0,0)
D = 0
for P in ListOfPoints:
    D += P.Dist(Origin)
N = len(ListOfPoints)
AveDist = D/len(ListOfPoints)

A lot of familiar stuff: Running sums. A for-loop based on “in”. The len function, Etc
A List of Random Points

```python
def RandomCloud(Lx,Rx, Ly, Ry, n):
    """ Returns a length-n list of points, each chosen randomly from the rectangle Lx<=x<=Rx, Ly<=y<=Ry.
    PreC: Lx and Rx are floats with Lx<Rx, Ly and Ry are floats with Ly<Ry, and n is a positive int.
    """
    A = []
    for k in range(n):
        P = RandomPoint(Lx, Rx, Ly, Ry)
        A.append(P)
    return A
```

The append method for lists works for lists of objects.
Visualizing a List of Points

>>> P = Point(3,4); Q = Point(1,2); R = Point(9,3)
>>> L = [P, Q, R]
Visualizing a List of Points

```python
>>> P = Point(3,4); Q = Point(1,2); R = Point(9,3)
>>> L = [P, Q, R]
```

More accurate: A List of references to Point objects
Operations on a List of Points

>>> L[1].x = 100

Before
Operations on a List of Points

>>> L[1].x = 100

After
Operations on a List of Points

>>> L[1] = Point(5,5)

Before
Operations on a List of Points

>>> L[1] = Point(5,5)

After
def printCloud(A):
    """ Prints the points in A
    """
    PreC : A is a list of points.
    """
    for a in A:
        print a

Synonym for the loop:

    for k in range(len(A)):
        print A[k]
We Now Showcase the Use of Lists of Objects

Example 1. A Disk Intersection Problem

Example 2. A Census Data Problem
A Disk Intersection Problem
A Class for Representing Disks

def class Disk(object):
    
    Attributes:
    center: Point, the center of the disk
    radius: float, the radius of the disk
    
    def __init__(self,P,r):
        
        Creates a Disk object with
        center P and radius r
        PreC: P is a Point,r is a pos float
        
        self.center = P
        self.radius = r

Note that an attribute can be an object. The center attribute is a Point
When Does a Pair of Disks Intersect?

Answer: When the distance between their centers is less than the sum of their radii.
def Intersects(self, other):
    """ Returns True if self and other intersect and False otherwise.  
    PreC: self and other are Disk objects  
    """
    # The center-to-center distance:
    c1 = self.center
    c2 = other.center
    d = c1.Dist(c2)
    # The sum of the two radii
    radiusSum = self.radius + other.radius
    TheyIntersect = (radiusSum >= d)
    return TheyIntersect
An Intersection Problem

We have a 10-by-10 target

for k in range(100):
    Generate a random disk \( D \)
    Display \( D \) if it does not touch any of the previously displayed disks

Assume all the disks have radius 1 and all inside the target.
def outsideAll(D0,L):
    """ Returns True if D0 doesn't intersect any of the disks in L
    """

    PreC: D0 is a Disk and L is a list of Disks
    """

    for D in L:
        if D.Intersects(D0):
            return False

    return True
Using outsideAll

# The list of displayed disks...
DiskList = []
for k in range(100):
    D = A random disk
    if outsideAll(D, DiskList):
        # D does not intersect any
        # of the displayed disks
        ShowDisk(D, MAGENTA)
        DiskList.append(D)

nDisplayed = len(DiskList)
A Census Data Sorting Problem
What Can We Sort?

We can sort a list of numbers from small to big (or big to small).

We can sort a list of strings from “A-to-Z” (or “Z-to-A”).

We can sort a list of objects based on an attribute if that attribute is either a number or a string.
A Sorting Problem

Suppose we have

class Student(object):
    Attributes:
        Name: string, student’s name
        GPA : float, student’s gpa

and that L is a list of Student objects...
A List of Student Objects

- **Student**
  - Name: 'Gaga'
  - GPA: 3.31
  - Index: L[0]

- **Student**
  - Name: 'Cher'
  - GPA: 4.00
  - Index: L[1]

- **Student**
  - Name: 'Adele'
  - GPA: 2.95
  - Index: L[2]
A List of Student Objects

We can sort this list based on Name or GPA.

L:


Student

Name: ‘Gaga’
GPA: 3.31

Student

Name: ‘Cher’
GPA: 4.00

Student

Name: ‘Adele’
GPA: 2.95
A List of Student Objects

1. **Student**
   - **Name:** ‘Adele’
   - **GPA:** 2.95
   - **Index:** $L[0]$

2. **Student**
   - **Name:** ‘Cher’
   - **GPA:** 4.00
   - **Index:** $L[1]$

3. **Student**
   - **Name:** ‘Gaga’
   - **GPA:** 3.31
   - **Index:** $L[2]$

*Sorted by Name*
A List of Student Objects

Name: ‘Cher’  GPA: 4.00
Name: ‘Gaga’  GPA: 3.31
Name: ‘Adele’ GPA: 2.95

Sorted by GPA

How to Do We Do This?

You have to write a “getter” function that extracts the value of the “key” attribute.

The name of this getter function is then passed as an argument to the sort method.

We illustrate the technique on a problem that involves census data.
class CountyPop(object):

    Attributes:

    Name: the name of the county (string)
    State: the name of the state (string)
    Pop2010: the 2010 population (int)
    Pop2011: the 2011 population (int)
    Pop2012: the 2012 population (int)
    Pop2013: the 2013 population (int)
    Pop2014: the 2014 population (int)
Setting Up the List of CountyPop Objects

The file `CensusData.csv` has these columns:

5. State Name
6. County Name
7. 2010 county population
10. 2011 county population
11. 2012 county population
12. 2013 county population
13. 2014 county population
TheCounties = fileToStringList('CensusData.csv')
L = []
for c in TheCounties:
    v = c.split(',', ')
    c = CountyPop(v[6], v[5], int(v[7]), int(v[10]),
                   int(v[11]), int(v[12]), int(v[13]))
    L.append(C)

The constructor sets up the Name, State, Pop2010, Pop2011, Pop2012, Pop2013, and Pop2014 attributes
Let's Sort!

```python
def getPop2014(C):
    # C is a County Object
    return C.Pop2014

if __name__ == '__main__':
    L.sort(key=getPop2014, reverse=True)
    for k in range(10):
        print L[k], L[k].Pop2014
```

This getter function grabs the 2014 population.

And here is how we tell `sort` to use it.

Printing the top ten counties in the USA in terms of population.
## Top Ten in 2014

<table>
<thead>
<tr>
<th>County</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Los Angeles County, California</td>
<td>1,011,6705</td>
</tr>
<tr>
<td>Cook County, Illinois</td>
<td>524,6456</td>
</tr>
<tr>
<td>Harris County, Texas</td>
<td>444,1370</td>
</tr>
<tr>
<td>Maricopa County, Arizona</td>
<td>408,7191</td>
</tr>
<tr>
<td>San Diego County, California</td>
<td>326,3431</td>
</tr>
<tr>
<td>Orange County, California</td>
<td>314,5515</td>
</tr>
<tr>
<td>Miami-Dade County, Florida</td>
<td>266,2874</td>
</tr>
<tr>
<td>Kings County, New York</td>
<td>262,1793</td>
</tr>
<tr>
<td>Dallas County, Texas</td>
<td>251,8638</td>
</tr>
<tr>
<td>Riverside County, California</td>
<td>232,9271</td>
</tr>
</tbody>
</table>