13B. Loops and Lists

Topics:
- Functions that return more than 1 thing
- Nested Loops
- Map

Computing the Diameter of a Cloud of Points

500 Points. Which two are furthest apart and what is their separation?

Same Problem:
What's the Biggest Number in This Table?

Which two cities are furthest apart and what is their separation?

It Will Have Three Functions

MakeCloud(n, sigma)
This generates two lists x and y that define the coordinates of the points in the cloud.

Diameter(x, y)
This will compute the diameter of the cloud using the (x, y) coordinates of its points.

ShowCloud(x, y)
This will use SimpleGraphics to display the cloud and highlight the "diameter points".

The Function MakeCloud

from random import normalvariate as randn
def MakeCloud(n, sigma):
x=[]
y=[]
for k in range(n):
r = randn(0, sigma)
x.append(r)
r = randn(0, sigma)
y.append(r)
return (x, y)

MakeCloud Returns Two Lists

from random import normalvariate as randn
def MakeCloud(n, sigma):
x=[]
y=[]
for k in range(n):
r = randn(0, sigma)
x.append(r)
r = randn(0, sigma)
y.append(r)
return (x, y)
MakeCloud Returns Two Lists

```
>>> (x,y) = MakeCloud(3,1)
>>> print x
>>> print y
[-2.328, -0.044, -0.241]
[ 2.737,  2.078, -1.272]
```

Note the parentheses

MakeCloud

```
from random import normalvariate as randn
def MakeCloud(n,sigma):
    x=[]
y=[]
    for k in range(n):
        r = randn(0,sigma)
x.append(r)
    r = randn(0,sigma)
y.append(r)
    return x,y
```

Old Stuff

```
x and y start out as empty lists.
Repeatedly generate a random number and append to x
Ditto for y
```

The Diameter Function: What It Computes

```
The "diameter points" and the distance between them
```

Input: lists x and y that define the yellow dots

Diameter: Formal Specs

```
def Diameter(x,y):
    """ Returns (d,imax,jmax) where d is a float that is the diameter of a cloud of points defined by lists x and y. imax and jmax are ints that are the indices of the diameter points.

    The diameter of a cloud of points is the maximum distance between any two points in the cloud. The two points for which this occurs are called diameter points.

    PreC: x and y are lists of floats with the same length.
    """
    d = 0
    n = len(x)
    for i in range(n):
        for j in range(n):
            dx = x[i]-x[j]
            dy = y[i]-y[j]
            dij = sqrt(dx**2+dy**2)
            if dij>d:
                d = dij
                imax = i
                jmax = j
    return (d,imax,jmax)
```

Diameter: The Implementation

```
def Diameter(x,y):
    d = 0
    n = len(x)
    for i in range(n):
        for j in range(n):
            dx = x[i]-x[j]
            dy = y[i]-y[j]
            dij = sqrt(dx**2+dy**2)
            if dij>d:
                d = dij
                imax = i
                jmax = j
    return (d,imax,jmax)
```

Nested Loops

```
In this situation we have a loop whose body contains a loop

for blahblahblah

and contains a loop.
```
Nested Loops: A Simple Example

```
for i in range(2):
    for j in range(3):
        print i, j
        print 'Inner'
    print 'Outer'
```

Execute the loop body with i=0

```
for i in range(2):
    for j in range(3):
        print i, j
        print 'Inner'
    print 'Outer'
```

Execute the loop body with i=1

```
for i in range(2):
    for j in range(3):
        print i, j
        print 'Inner'
    print 'Outer'
```

Go to the next statement after the loop body.
Nested Loops: A Simple Example

```python
for i in range(2):
    for j in range(3):
        print i, j
        print 'Inner'
    print 'Outer'
```

Go to the next statement after the loop body.

Back to Diameter

When developing nested-loop solutions, it is essential to apply the methodology of step-wise refinement, perhaps preceded by a small example.

Aspects of our problem

- Must check all possible pairs of points.
- Look at their separation distance.
- What’s the largest among these distances?

Suppose There Are 3 points

```plaintext
<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Dist</th>
</tr>
</thead>
</table>
| (x[0], y[0])  | (x[0], y[0])  | 0
| (x[0], y[0])  | (x[1], y[1])  | 7   |
| (x[0], y[0])  | (x[2], y[2])  | 9   |
| (x[1], y[1])  | (x[0], y[0])  | 7   |
| (x[1], y[1])  | (x[1], y[1])  | 0   |
| (x[1], y[1])  | (x[2], y[2])  | 10  |
| (x[2], y[2])  | (x[0], y[0])  | 9   |
| (x[2], y[2])  | (x[1], y[1])  | 10  |
| (x[2], y[2])  | (x[2], y[2])  | 0   |
```

Number of possibilities: 9 = 3x3

Suppose There Are 3 points

```plaintext
<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Dist</th>
</tr>
</thead>
</table>
| (x[0], y[0])  | (x[0], y[0])  | 0
| (x[0], y[0])  | (x[1], y[1])  | 7   |
| (x[0], y[0])  | (x[2], y[2])  | 9   |
| (x[1], y[1])  | (x[0], y[0])  | 7   |
| (x[1], y[1])  | (x[1], y[1])  | 0   |
| (x[1], y[1])  | (x[2], y[2])  | 10  |
| (x[2], y[2])  | (x[0], y[0])  | 9   |
| (x[2], y[2])  | (x[1], y[1])  | 10  |
| (x[2], y[2])  | (x[2], y[2])  | 0   |
```

Number of possibilities: 9 = 3x3

First Solution

```python
d = 0
n = len(x)
for i in range(n):
    # Examine the distance from
    # (x[i],y[i]) to every other point
```

And now, stepwise refinement in action....
Second Solution

d = 0
n = len(x)
for i in range(n):
    for j in range(n):
        # Examine the distance from
        # (x[i],y[i]) to (x[j],y[j])

Third Solution

d = 0
n = len(x)
for i in range(n):
    for j in range(n):
        dx = x[i]-x[j]
dy = y[i]-y[j]
dij = sqrt(dx**2+dy**2)
        # Compare dij to d revising
        # the latter if necessary

Fourth Solution

d = 0
n = len(x)
for i in range(n):
    for j in range(n):
        dx = x[i]-x[j]
dy = y[i]-y[j]
dij = sqrt(dx**2+dy**2)
        if dij>d:
            d = dij
            imax = i
            jmax = j
return (d,imax,jmax)

Fourth Solution

d = 0
n = len(x)
for i in range(n):
    for j in range(n):
        dx = x[i]-x[j]
dy = y[i]-y[j]
dij = sqrt(dx**2+dy**2)
        if dij>d:
            d = dij
            imax = i
            jmax = j
return (d,imax,jmax)

Next Up: ShowCloud

ShowCloud: Specs

def ShowCloud(x,y):
    """
    Displays a point cloud
defined by x and y and highlights
the two points that define
its diameter.

PreC: x and y are lists of
    floats with the same length.
    """
First: How Big a Window?

New Feature: map

```python
xMax = max(map(abs,x))
yMax = max(map(abs,y))
M = max(xMax, yMax)
MakeWindow(1.1*M, bgcolor=BLACK)
```

Idea: look at the x and y coordinates of the points and see how big they can be.

Map: Apply a Function to Each Element in a List

Example. Apply the absolute value function to every list element:

```python
>>> x = [10, -20, -40]
>>> x = map(abs,x)
>>> print x
[10, 20, 40]
```

Example. Apply the floor function to every list element:

```python
>>> x = [11.3, 12.4, 15.0]
>>> x = map(math.floor,x)
>>> print x
[11.0, 12.0, 15.0]
```

This:

```
y = []
for k in range(len(x)):
y.append(math.sqrt(x([k])))
```

Is equivalent to this:

```
y = map(math.sqrt,x)
```

Assuming that x is an initialized list of nonnegative numbers

Map: Formal Syntax

```python
map(          ,             )
```

The name of a function that returns a value. Every element in the list must satisfy its precondition.

The name of a list.

Now, Back to ShowCloud
First: How Big a Window?

```python
xMax = max(map(abs, x))
yMax = max(map(abs, y))
M = max(xMax, yMax)
MakeWindow(1.1*M, bgcolor=BLACK)
```

```python
x = [-19, 12, -4]
max(map(abs, x))
>>> 19
```

Next, Use DrawDisk For Each Point

```python
r = M/50;
(d, i, j) = Diameter(x, y)
for k in range(len(x)):
    if k == i or k == j:
        DrawDisk(x[k], y[k], 2*r, FillColor=CYAN)
        DrawDisk(x[k], y[k], r, FillColor=YELLOW)
```
i and j are the indices of the diameter points.

Before they are displayed, we paint a larger cyan dot.