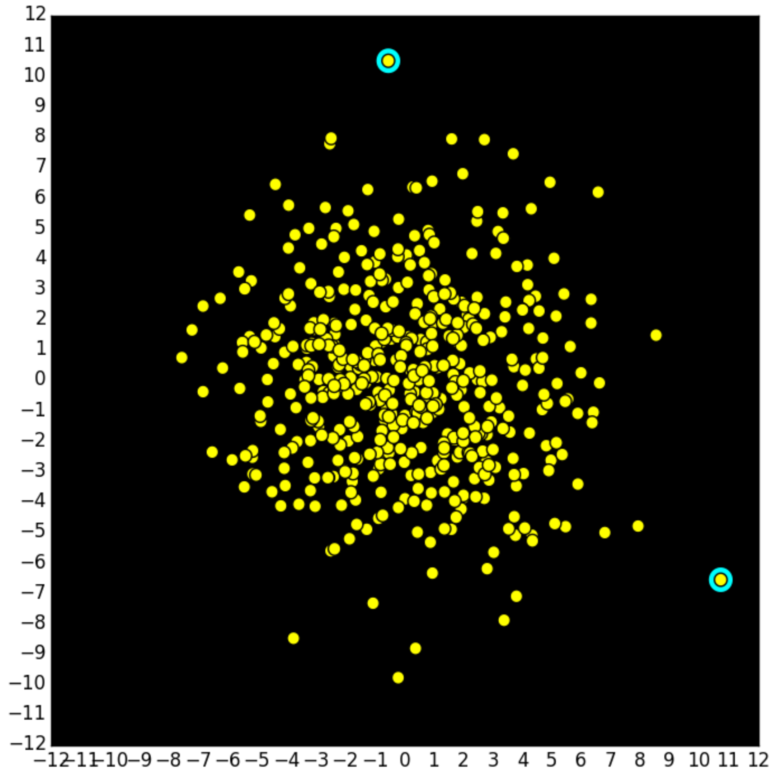
13B. Loops and Lists

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?

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		**Amsterdam**	**Berlin**	**Bordeaux**	**Brussels**	**Copenhagen**	**Dublin**	**Lisbon**	**London**	**Madrid**	**Milan**	**Munich**	**Paris**	**Rome**	**Zurich**
2	**Amsterdam**	0	650.594	1084.367	204.7	766.456	946.404	2254.519	476.014	1783.664	1071.746	820.188	503.852	1657.55	818.784
3	**Berlin**	651.304	0	1634.132	764.787	379.95	1506.491	2804.284	1036.101	2333.429	1033.586	582.566	1053.617	1513.741	844.044
4	**Bordeaux**	1084.547	1630.51	0	890.135	1785.177	1444.887	1174.092	975.717	703.237	1018.437	1284.774	582.938	1508.036	1021.859
5	**Brussels**	207.37	767.381	891.025	0	908.03	775.414	2061.177	306.244	1590.322	881.246	784.539	310.51	1467.05	628.274
6	**Copenhagen**	768.376	381.155	1785.864	906.197	0	1646.681	2956.016	1177.511	2485.161	1414.722	1080.551	1205.349	2011.726	1185.589
7	**Dublin**	939.78	1499.75	1439.475	769.049	1640.41	0	2609.627	453.606	2138.772	1641.326	1554.938	863.552	2227.14	1388.364
8	**Lisbon**	2251.111	2797.07	1171.514	2056.699	2951.741	2611.451	0	2142.281	626.064	2150.158	2448.668	1749.502	2535.253	2185.753
9	**London**	478.973	1038.94	978.668	308.242	1179.603	455.078	2148.82	0	1677.965	1180.519	1094.131	402.745	1766.323	927.557
10	**Madrid**	1782.485	2328.44	702.888	1588.073	2483.115	2144.045	625.192	1673.655	0	1581.588	1978.157	1280.876	1966.683	1669.123
11	**Milan**	1074.297	1035.63	1019.438	905.951	1415.052	1672.432	2152.653	1202.042	1580.336	0	492.726	847.819	584.634	279.263
12	**Munich**	822.285	582.946	1282.395	783.498	1078.905	1559.472	2450.087	1090.302	1976.382	490.983	0	828.256	929.685	314.143
13	**Paris**	502.799	1048.75	583.225	308.387	1203.429	869.622	1753.377	400.452	1282.522	848.469	830.414	0	1418.908	653.608
14	**Rome**	1660.357	1514.24	1509.825	1492.011	1976.829	2257.272	2540.524	1788.102	1968.207	586.94	930.682	1431.299	0	865.323
15	**Zurich**	821.854	845.704	1021.829	653.218	1186.023	1419.699	2189.521	949.309	1668.309	279.652	315.164	653.299	865.456	0
16															

Distances / Times

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)
```

The normal distribution

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)
```

New Feature

A function that returns more than one thing.

Note the parentheses

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
```

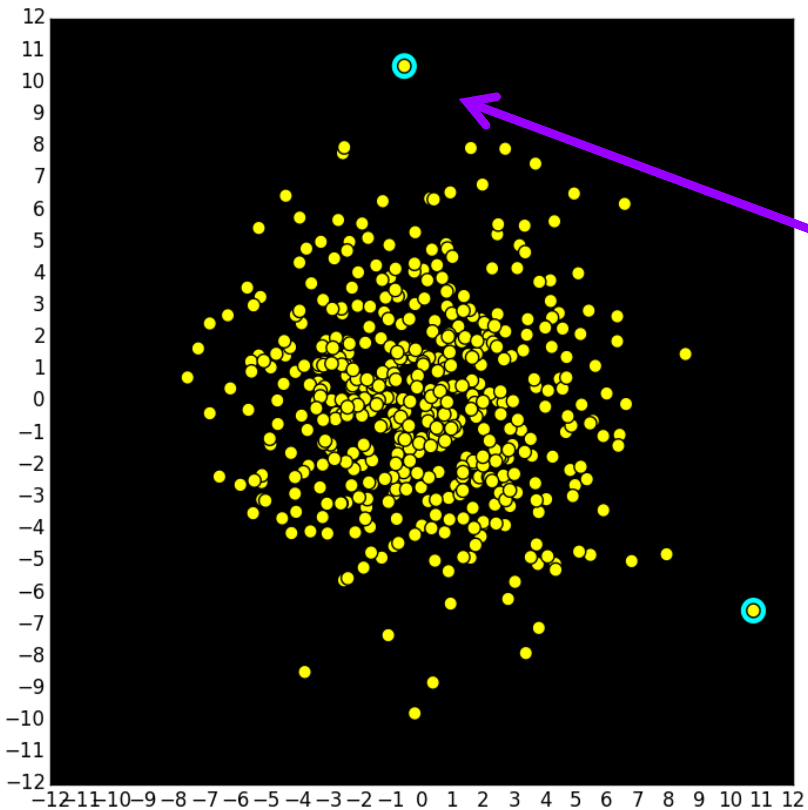
Old Stuff

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.
```

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)
```

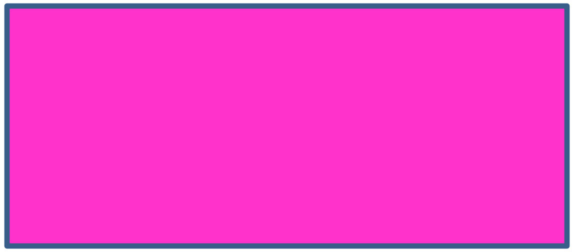
New Feature

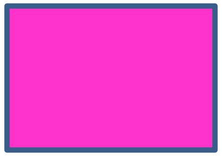
Nested Loops

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'
```

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

Nested Loops: A Simple Example

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

print 'Outer'
```

```
0  0
0  1
0  2
Inner
```

Execute the loop body with `i=0`

Nested Loops: A Simple Example

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

print 'Outer'
```

```
0 0
0 1
0 2
Inner
```

Execute the loop body with **i=1**

Nested Loops: A Simple Example

```
for i in range(2):
    for j in range(3):
        print i,j
    print ‘Inner’

print ‘Outer’
```

```
0  0
0  1
0  2
Inner
1  0
1  1
1  2
Inner
```

Execute the loop body with i=1

Nested Loops: A Simple Example

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

print 'Outer'
```

```
0  0
0  1
0  2
Inner
1  0
1  1
1  2
Inner
```

Go to the next statement after the loop body.

Nested Loops: A Simple Example

```
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.

```
0  0
0  1
0  2
Inner
1  0
1  1
1  2
Inner
Outer
```

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

From	To	Dist
(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

From	To	Dist
(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

And now, stepwise refinement in action….

First Solution

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

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

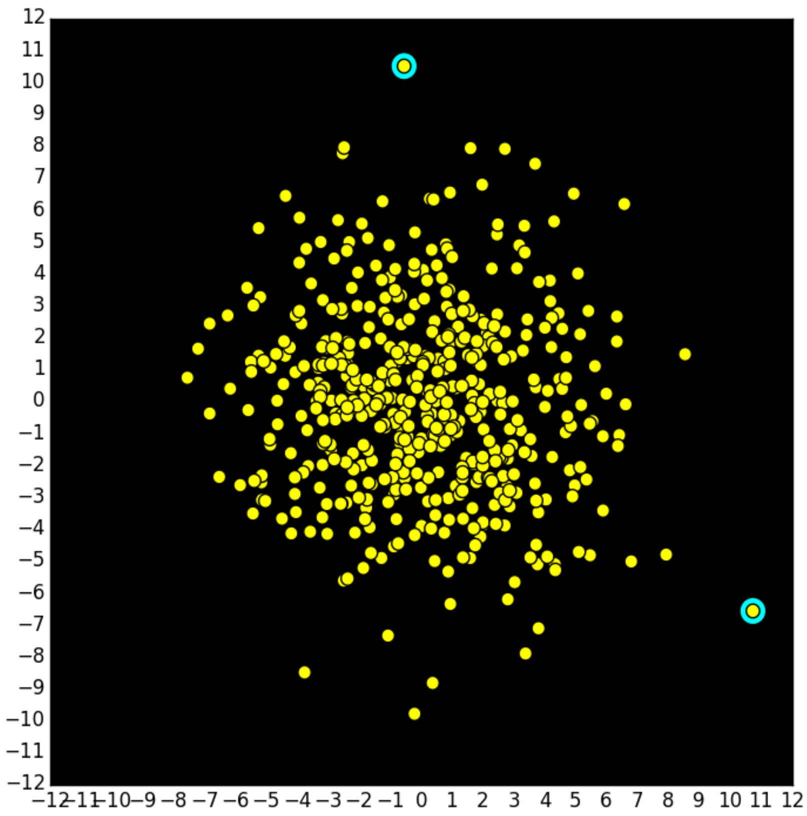
```
d = 0
n = len(x)
for i in range(n):
    for j in range(n):
        dx = x[i]-x[j]
        dy = y[i]-y[j]
        dij = sqrt(dx**2+dy**2)
        if dij>d:
            d = dij
            imax = i
            jmax = j
return (d,imax,jmax)
```

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)
```

We have to "remember" where the max separation occurs.

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

```
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

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

Map: Apply a Function to Each Element in a List

Example. Apply the floor function to every list element:

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

Map: Apply a Function to Each Element in a List

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

`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?

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

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

Next, Use DrawDisk For Each Point

```
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.