15. Functions and Lists

Topics:
- Subscripting
- Map
- Searching a list
- Example 1: Clouds of points
- Example 2: Selection Sort

Example 1: Computing the Diameter of a Cloud of Points

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

Example 1: Computing the Diameter of a Cloud of Points

Example 2: Sorting a List of Numbers

Before:

| x | 50 | 40 | 10 | 80 | 20 | 60 |

After:

| x | 10 | 20 | 40 | 50 | 60 | 80 |

Example 1: Computing the Diameter of a Cloud of Points

Example 2: Sorting a List of Numbers

Same Problem:

What's the Biggest Number in This Table?

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

Sorting Algorithms

There are many sorting algorithms:
- Selection Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Quick Sort

A great venue for practicing list-based computing and for studying such things as efficiency and recursion (which we will do later).
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 simpleGraphicsE to display the cloud and highlight the "diameter points".

The Function MakeCloud

```python
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 ($x$, $y$)
```

Generating floats from the Normal Distribution

This is a histogram of the numbers this generates:

![Histogram](image)

New Feature

A function that returns more than one thing.

Note the parentheses

MakeCloud Returns Two Lists

```python
from random import normalvariate as randn
def MakeCloud($n$, $\sigma$):
x=[]
y=[]
for $k$ in range($10^6$):
    $r$ = randn(0,1)
Mean = 0
Standard deviation = 1
```

Generating floats from the Normal Distribution

If $\mu$ and $\sigma$ (positive) are floats, then

```
x = random.normalvariate($\mu$, $\sigma$)
```

assigns to $x$ a "random" float sampled from the normal distribution with mean $\mu$ and standard deviation $\sigma$. 

Generating floats from the Normal Distribution

```
Mean = 0
Standard deviation = 1
```
**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]
```

**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**

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

**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)
```

**New Feature**

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

Execute the loop body with i=1

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:

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

Number of possibilities: 9 = 3x3

Suppose There Are 3 points:

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

Number of possibilities: 9 = 3x3

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

\[
\begin{align*}
&d = 0 \\
n &= \text{len}(x) \\
&\text{for } i \text{ in range}(n): \\
&\quad \text{for } j \text{ in range}(n): \\
&\quad \quad \# \text{ Examine the distance from} \\
&\quad \quad \# \text{(x}[i],y[i]) \text{ to } (x[j],y[j]) \\
\end{align*}
\]

Third Solution

\[
\begin{align*}
&d = 0 \\
n &= \text{len}(x) \\
&\text{for } i \text{ in range}(n): \\
&\quad \text{for } j \text{ in range}(n): \\
&\quad \quad \text{dx} = x[i]-x[j] \\
&\quad \quad \text{dy} = y[i]-y[j] \\
&\quad \quad \text{dij} = \sqrt{\text{dx}^2+\text{dy}^2} \\
&\quad \quad \# \text{ Compare dij to d revising} \\
&\quad \quad \# \text{ the latter if necessary} \\
\end{align*}
\]

Fourth Solution

\[
\begin{align*}
&d = 0 \\
n &= \text{len}(x) \\
&\text{for } i \text{ in range}(n): \\
&\quad \text{for } j \text{ in range}(n): \\
&\quad \quad \text{dx} = x[i]-x[j] \\
&\quad \quad \text{dy} = y[i]-y[j] \\
&\quad \quad \text{dij} = \sqrt{\text{dx}^2+\text{dy}^2} \\
&\quad \quad \text{if } \text{dij}>d: \\
&\quad \quad \quad d = \text{dij} \\
&\quad \quad \quad \text{imax} = i \\
&\quad \quad \quad \text{jmax} = j \\
&\quad \text{return } (d,\text{imax},\text{jmax}) \\
\end{align*}
\]

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

Fourth Solution

\[
\begin{align*}
&d = 0 \\
n &= \text{len}(x) \\
&\text{for } i \text{ in range}(n): \\
&\quad \text{for } j \text{ in range}(n): \\
&\quad \quad \text{dx} = x[i]-x[j] \\
&\quad \quad \text{dy} = y[i]-y[j] \\
&\quad \quad \text{dij} = \sqrt{\text{dx}^2+\text{dy}^2} \\
&\quad \quad \text{if } \text{dij}>d: \\
&\quad \quad \quad d = \text{dij} \\
&\quad \quad \quad \text{imax} = i \\
&\quad \quad \quad \text{jmax} = j \\
&\text{return } (d,\text{imax},\text{jmax}) \\
\end{align*}
\]

Next Up: ShowCloud

ShowCloud: Specs

```python
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`

\[
\begin{align*}
\text{xMax} & = \max(\text{map}(\text{abs}, x)) \\
\text{yMax} & = \max(\text{map}(\text{abs}, y)) \\
M & = \max(\text{xMax}, \text{yMax}) \\
\text{MakeWindow}(1.1*M, \text{bgcolor}=\text{BLACK})
\end{align*}
\]

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:

\[
\begin{align*}
\text{>>> } x & = [10, -20, -40] \\
\text{>>> } x & = \text{map}(\text{abs}, x) \\
\text{>>> print } x \\
& [10, 20, 40]
\end{align*}
\]

Map: Apply a Function to Each Element in a List

Example. Apply the floor function to every list element:

\[
\begin{align*}
\text{>>> } x & = [11.3, 12.4, 15.0] \\
\text{>>> } x & = \text{map}(\text{math.floor}, x) \\
\text{>>> print } x \\
& [11.0, 12.0, 15.0]
\end{align*}
\]

Map: Apply a Function to Each Element in a List

This:

\[
\begin{align*}
\text{y} & = [] \\
\text{for } k \text{ in range(len(x))}{:} \\
\text{y.append(}\text{math.sqrt}(x([k])))
\end{align*}
\]

Is equivalent to this:

\[
\text{y} = \text{map}(\text{math.sqrt}, x)
\]

Assuming that \(x\) is an initialized list of nonnegative numbers.

Map: Formal Syntax

\[
\text{map(} \quad , \quad )
\]

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))
```

```python
>>> 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,color=CYAN)
        DrawDisk(x[k],y[k],r,color=YELLOW)
```

i and j are the indices of the diameter points.

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

Now, on to another example that highlights functions + lists

Example 2: Sorting a List of Numbers

Before:
```
   x -->  50 40 10 80 20 60
```

After:
```
   x -->  10 20 40 50 60 80
```

We Will Implement the Method of Selection Sort

At the Start:
```
   x -->  50 40 10 80 20 60
```

High-Level:
```
for k in range(len(x)-1)
    Swap x[k] with the smallest value in x[k:]
```
**Selection Sort: How It Works**

**Before:**
\[ x \rightarrow 50 \ 40 \ 10 \ 80 \ 20 \ 60 \]

Swap \( x[0] \) with the smallest value in \( x[0:] \)

**After:**
\[ x \rightarrow 10 \ 40 \ 50 \ 80 \ 20 \ 60 \]

**Selection Sort: How It Works**

**Before:**
\[ x \rightarrow 10 \ 40 \ 50 \ 80 \ 20 \ 60 \]

Swap \( x[1] \) with the smallest value in \( x[1:] \)

**Selection Sort: How It Works**

**Before:**
\[ x \rightarrow 10 \ 40 \ 50 \ 80 \ 20 \ 60 \]

Swap \( x[2] \) with the smallest value in \( x[2:] \)

**After:**
\[ x \rightarrow 10 \ 20 \ 50 \ 80 \ 40 \ 60 \]

**Selection Sort: How It Works**

**Before:**
\[ x \rightarrow 10 \ 20 \ 50 \ 80 \ 40 \ 60 \]

Swap \( x[3] \) with the smallest value in \( x[3:] \)

**After:**
\[ x \rightarrow 10 \ 20 \ 40 \ 80 \ 50 \ 60 \]
Selection Sort: How It Works

Before:
\[ x \rightarrow 10 \ 20 \ 40 \ 80 \ 50 \ 60 \]
Swap \( x[3] \) with the smallest value in \( x[3:] \)
After:
\[ x \rightarrow 10 \ 20 \ 40 \ 50 \ 80 \ 60 \]

Selection Sort: How It Works

Before:
\[ x \rightarrow 10 \ 20 \ 40 \ 50 \ 80 \ 60 \]
Swap \( x[4] \) with the smallest value in \( x[4:] \)
After:
\[ x \rightarrow 10 \ 20 \ 40 \ 50 \ 80 \ 60 \]

Selection Sort: How It Works

Before:
\[ x \rightarrow 10 \ 20 \ 40 \ 50 \ 80 \ 60 \]
Swap \( x[4] \) with the smallest value in \( x[4:] \)
After:
\[ x \rightarrow 10 \ 20 \ 40 \ 50 \ 80 \ 60 \]

Selection Sort: Recap

\[ \begin{array}{c|cccccc} 50 & 40 & 10 & 80 & 20 & 60 \\ 10 & 20 & 50 & 80 & 20 & 60 \\ 10 & 20 & 40 & 80 & 50 & 60 \\ 10 & 20 & 40 & 50 & 80 & 60 \\ 10 & 20 & 40 & 50 & 60 & 80 \\ 10 & 20 & 40 & 50 & 60 & 80 \end{array} \]

The Essential Helper Function: \( \text{Select}(x,i) \)

```python
def Select(x,i):
    """ Swaps the smallest value in \( x[i:] \) with \( x[i] \)
    PreC: \( x \) is a list of integers and \( i \) is an in that satisfies \( 0 \leq i < \text{len}(x) \)"
    Does not return anything and it has a list argument
```

How Does it Work?
The calling program has a list. E.g.,

\[ a \rightarrow \begin{array}{c|c}
0 & 50 \\
1 & 40 \\
2 & 10 \\
3 & 80 \\
4 & 20 \\
5 & 60 \\
\end{array} \]
How Does it Work?
The calling program executes \texttt{Select(a,0)} and control passes to \texttt{Select}

\begin{verbatim}
a --> 0 ----> 50 
 1 ----> 40 
 2 ----> 10 
 3 ----> 80 
 4 ----> 20 
 5 ----> 60 
\end{verbatim}

How Does Select Work?
- Nothing new about the assignment of 0 to \(i\).
- But there is no assignment of the list \(a\) to \(x\).
- Instead \(x\) now refers to the same list as \(a\).

\begin{verbatim}
\texttt{a --> 0 ----> 50} 
\texttt{0 ----> 1 ----> 40} 
\texttt{2 ----> 10} 
\texttt{3 ----> 80} 
\texttt{4 ----> 20} 
\texttt{5 ----> 60}
\end{verbatim}

How Does Select Work?
If inside \texttt{Select} we have
\(t = x[0]; x[0] = x[2]; x[2] = t\)
it is as if we said
\(t = a[0]; a[0] = a[2]; a[2] = t\)

\begin{verbatim}
x ------------------- 
\texttt{i --> 0} 
\texttt{0 ----> 50} 
\texttt{1 ----> 40} 
\texttt{2 ----> 10} 
\texttt{3 ----> 80} 
\texttt{4 ----> 20} 
\texttt{5 ----> 60}
\end{verbatim}

How Does Select Work?
It changes the list \(a\) in the calling program.
We say \(x\) and \(a\) are aliased. They refer to the same list.

\begin{verbatim}
x ------------------- 
\texttt{i --> 0} 
\texttt{0 ----> 10} 
\texttt{1 ----> 40} 
\texttt{2 ----> 50} 
\texttt{3 ----> 80} 
\texttt{4 ----> 20} 
\texttt{5 ----> 60}
\end{verbatim}

Let's Assume This Is Implemented

\begin{verbatim}
def Select(x,i):
    """ Swaps the smallest value in 
    \(x[i:]\) with \(x[i]\)"

    PreC: \(x\) is a list of integers and \(i\) is an in that satisfies
    \(0 \leq i < \text{len}(x)\)""
\end{verbatim}

<table>
<thead>
<tr>
<th>After this:</th>
<th>The list (a) looks like this</th>
</tr>
</thead>
</table>
| Initialization | \begin{verbatim}
50 40 10 80 20 60
\end{verbatim} |
| Select\((a,0)\)  | \begin{verbatim}
10 40 50 80 20 60
\end{verbatim} |
| Select\((a,1)\)  | \begin{verbatim}
10 20 50 80 40 60
\end{verbatim} |
| Select\((a,2)\)  | \begin{verbatim}
10 20 40 80 50 60
\end{verbatim} |
| Select\((a,3)\)  | \begin{verbatim}
10 20 40 50 80 60
\end{verbatim} |
| Select\((a,4)\)  | \begin{verbatim}
10 20 40 50 60 80
\end{verbatim} |
| Select\((a,5)\)  | \begin{verbatim}
10 20 40 50 60 80
\end{verbatim} |
In General We Have This

def SelectionSort(a):
    n = len(a)
    for k in range(n):
        Select(a,k)