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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Amsterdam</td>
<td>Berlin</td>
<td>Bordeaux</td>
<td>Brussels</td>
<td>Copenhagen</td>
<td>Dublin</td>
<td>Lisbon</td>
<td>London</td>
<td>Madrid</td>
<td>Milan</td>
<td>Munich</td>
<td>Paris</td>
<td>Rome</td>
<td>Zurich</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>550.594</td>
<td>1084.367</td>
<td>204.7</td>
<td>765.456</td>
<td>946.404</td>
<td>2254.519</td>
<td>476.014</td>
<td>1733.664</td>
<td>1071.746</td>
<td>820.188</td>
<td>503.852</td>
<td>1657.65</td>
<td>815.784</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Berlin</td>
<td>661.304</td>
<td>0</td>
<td>1634.132</td>
<td>764.707</td>
<td>379.95</td>
<td>1506.491</td>
<td>2604.284</td>
<td>1036.101</td>
<td>2333.429</td>
<td>1033.596</td>
<td>592.566</td>
<td>1053.617</td>
<td>1513.741</td>
<td>644.044</td>
</tr>
<tr>
<td>4</td>
<td>Bordeaux</td>
<td>1084.547</td>
<td>1630.61</td>
<td>0</td>
<td>890.136</td>
<td>1785.177</td>
<td>1444.687</td>
<td>1174.092</td>
<td>975.717</td>
<td>733.237</td>
<td>1018.437</td>
<td>1294.774</td>
<td>562.998</td>
<td>1658.036</td>
<td>1021.859</td>
</tr>
<tr>
<td>5</td>
<td>Brussels</td>
<td>217.37</td>
<td>767.381</td>
<td>891.025</td>
<td>0</td>
<td>906.03</td>
<td>775.414</td>
<td>2061.177</td>
<td>306.244</td>
<td>1590.322</td>
<td>881.246</td>
<td>794.539</td>
<td>310.51</td>
<td>1467.16</td>
<td>624.274</td>
</tr>
<tr>
<td>6</td>
<td>Copenhagen</td>
<td>766.378</td>
<td>381.165</td>
<td>1785.864</td>
<td>906.197</td>
<td>0</td>
<td>1546.681</td>
<td>2556.016</td>
<td>1177.511</td>
<td>2485.161</td>
<td>1414.722</td>
<td>1080.951</td>
<td>1205.349</td>
<td>2011.726</td>
<td>1185.689</td>
</tr>
<tr>
<td>7</td>
<td>Dublin</td>
<td>939.73</td>
<td>1459.75</td>
<td>1439.475</td>
<td>763.049</td>
<td>1640.41</td>
<td>0</td>
<td>2609.627</td>
<td>453.606</td>
<td>2139.772</td>
<td>1641.326</td>
<td>1554.938</td>
<td>863.552</td>
<td>2227.14</td>
<td>1383.364</td>
</tr>
<tr>
<td>8</td>
<td>Lisbon</td>
<td>2251.11</td>
<td>2797.07</td>
<td>1171.514</td>
<td>2056.699</td>
<td>2951.741</td>
<td>2611.451</td>
<td>0</td>
<td>2142.281</td>
<td>626.064</td>
<td>2150.158</td>
<td>2448.688</td>
<td>1749.502</td>
<td>2536.253</td>
<td>2165.753</td>
</tr>
<tr>
<td>9</td>
<td>London</td>
<td>476.973</td>
<td>1038.94</td>
<td>978.668</td>
<td>303.242</td>
<td>1173.603</td>
<td>455.078</td>
<td>2146.82</td>
<td>0</td>
<td>1677.965</td>
<td>1180.519</td>
<td>1094.131</td>
<td>402.745</td>
<td>1766.323</td>
<td>927.557</td>
</tr>
<tr>
<td>11</td>
<td>Milan</td>
<td>1074.297</td>
<td>1035.63</td>
<td>1019.438</td>
<td>905.961</td>
<td>1415.052</td>
<td>1672.432</td>
<td>2152.663</td>
<td>1202.042</td>
<td>1680.336</td>
<td>0</td>
<td>492.732</td>
<td>847.819</td>
<td>584.634</td>
<td>279.263</td>
</tr>
<tr>
<td>13</td>
<td>Paris</td>
<td>502.799</td>
<td>1048.75</td>
<td>583.225</td>
<td>303.367</td>
<td>1203.429</td>
<td>869.622</td>
<td>1753.377</td>
<td>400.452</td>
<td>1292.512</td>
<td>848.439</td>
<td>830.414</td>
<td>0</td>
<td>1418.908</td>
<td>653.606</td>
</tr>
<tr>
<td>15</td>
<td>Zurich</td>
<td>821.854</td>
<td>845.704</td>
<td>1021.829</td>
<td>653.218</td>
<td>1186.023</td>
<td>1419.699</td>
<td>2189.521</td>
<td>949.309</td>
<td>1668.309</td>
<td>279.652</td>
<td>315.164</td>
<td>653.299</td>
<td>865.456</td>
<td>0</td>
</tr>
</tbody>
</table>

Which two cities are furthest apart and what is their separation?
Example 2: Sorting a List of Numbers

Before:

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

After:

\[ x \rightarrow \begin{array}{cccccc}
10 & 20 & 40 & 50 & 60 & 80 \\
\end{array} \]
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).
Example 1: Computing the Diameter of a Cloud of Points

We will develop a module PointCloud
It Will Have Three Functions

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

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

\textbf{ShowCloud}(x, y) \\
This will use \texttt{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)
```

New Feature

The normal distribution
Generating floats from the Normal Distribution
Generating floats from the Normal Distribution

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

$$x = \text{random.normalvariate}(\mu, \sigma)$$

assigns to $x$ a “random” float sampled from the normal distribution with mean $\mu$ and standard deviation $\sigma$.
Generating floats from the Normal Distribution

This is a histogram of the numbers this generates:

```python
for k in range(10**6):
    r = randn(0,1)
```

Mean = 0
Standard deviation = 1
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

```python
>>> (x, y) = MakeCloud(3, 1)
>>> print x
>>> print y

[-2.328, -0.044, -0.241]
[ 2.737,  2.078, -1.272]
```

Note the parentheses
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
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. """
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

```python
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'

Execute the loop body with i=0

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Inner
Nested Loops: A Simple Example

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

Execute the loop body with i=1

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

`Inner`
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=1
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.
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>10</td>
</tr>
<tr>
<td>(x[2], y[2])</td>
<td>(x[0], y[0])</td>
<td>9</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.: $9 = 3 \times 3$
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>9</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....
First Solution

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

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

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

```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]
```
Map: Apply a Function to Each Element in a List

This:

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

Is equivalent to this:

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

Assuming that \( x \) is an initialized list of nonnegative numbers.
Map: Formal Syntax

\texttt{map ( \text{\_\_\_\_\_\_\_\_}, \text{\_\_\_\_\_\_\_\_} )}

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)
```

```
x = [-19, 12, -4]
max(map(abs, x))
>>> 19
```
Next, Use DrawDisk For Each Point

\[ r = \frac{M}{50}; \]
\[ (d,i,j) = \text{Diameter}(x,y) \]
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:

\[ x --\rightarrow \begin{array}{cccccccc} 50 & 40 & 10 & 80 & 20 & 60 \end{array} \]

After:

\[ x --\rightarrow \begin{array}{cccccccc} 10 & 20 & 40 & 50 & 60 & 80 \end{array} \]
We Will Implement the Method of Selection Sort

At the Start:

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

High-Level:

\[
\text{for k in range(len(x)-1)}
\]

\[
\text{Swap x[k] with the smallest value in x[k:]}\]
Selection Sort: How It Works

Before:

\[ x \rightarrow \begin{array}{c} 50 \ 40 \ 10 \ 80 \ 20 \ 60 \end{array} \]

Swap \( x[0] \) with the smallest value in \( x[0:] \)
Selection Sort: How It Works

Before:

\[ x \rightarrow \begin{array}{ccccccc}
50 & 40 & 10 & 80 & 20 & 60 & x \end{array} \]

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

After:

\[ x \rightarrow \begin{array}{ccccccc}
10 & 40 & 50 & 80 & 20 & 60 & \end{array} \]
Selection Sort: How It Works

Before:

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

Swap \( x[1] \) with the smallest value in \( x[1:] \)
Selection Sort: How It Works

Before:

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

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

After:

\[ x \rightarrow \begin{array}{c|c|c|c|c|c|c} 10 & 20 & 50 & 80 & 40 & 60 \end{array} \]
Selection Sort: How It Works

Before:

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

Swap \( x[2] \) with the smallest value in \( x[2:] \).
Selection Sort: How It Works

Before:

\[ x \rightarrow \begin{array}{cccccccc}
10 & 20 & 50 & 80 & 40 & 60
\end{array} \]

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

After:

\[ x \rightarrow \begin{array}{cccccccc}
10 & 20 & 40 & 80 & 50 & 60
\end{array} \]
Selection Sort: How It Works

Before:

\[
x \rightarrow \begin{array}{ccccccc}
10 & 20 & 40 & 80 & 50 & 60
\end{array}
\]

Swap x[3] with the smallest value in x[3:]
Selection Sort: How It Works

Before:

\[ x \rightarrow \begin{array}{ccccccc} 10 & 20 & 40 & 80 & 50 & 60 \\ \end{array} \]

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

After:

\[ x \rightarrow \begin{array}{ccccccc} 10 & 20 & 40 & 50 & 80 & 60 \\ \end{array} \]
Selection Sort: How It Works

Before:

\[
x \quad \rightarrow \quad \begin{array}{cccccc}
10 & 20 & 40 & 50 & 80 & 60
\end{array}
\]

Swap \( x[4] \) with the smallest value in \( x[4:] \)
Selection Sort: How It Works

Before:

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

Swap \(x[4]\) with the smallest value in \(x[4:]\)

After:

\[
x \rightarrow \begin{array}{cccccc}
10 & 20 & 40 & 50 & 60 & 80 \\
\end{array}
\]
# Selection Sort: Recap

<table>
<thead>
<tr>
<th></th>
<th>50</th>
<th>40</th>
<th>10</th>
<th>80</th>
<th>20</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>40</td>
<td>50</td>
<td>80</td>
<td>20</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>50</td>
<td>80</td>
<td>40</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
<td>50</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>40</td>
<td>50</td>
<td>80</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
The Essential Helper Function:
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 in that satisfies 0<=i<len(x)""
```

Does not return anything and it has a list argument
How Does it Work?

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

The calling program executes `Select(a, 0)` and control passes to `Select`
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.
How Does Select Work?

If inside Select we have
\[ t = x[0]; \ x[0] = x[2]; \ x[2] = t \]
it is as if we said
\[ t = a[0]; \ a[0] = a[2]; \ a[2] = t \]
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.

\[
\begin{array}{|c|c|}
\hline
i & \rightarrow & 0 \\
\hline
x & \rightarrow & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
i & \rightarrow & 0 \\
\hline
1 & \rightarrow & 10 \\
2 & \rightarrow & 40 \\
3 & \rightarrow & 50 \\
4 & \rightarrow & 80 \\
5 & \rightarrow & 20 \\
\hline
\end{array}
\]
Let's Assume This Is Implemented

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 in that satisfies $0 \leq i < \text{len}(x)$"""
After this:

**Initialization**

Select(a,0)

Select(a,1)

Select(a,2)

Select(a,3)

Select(a,4)

Select(a,5)

The list a looks like this

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>40</td>
<td>10</td>
<td>80</td>
<td>20</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>40</td>
<td>50</td>
<td>80</td>
<td>20</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>50</td>
<td>80</td>
<td>40</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
<td>50</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>40</td>
<td>50</td>
<td>80</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
```
In General We Have This

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