13. Lists of Numbers

Topics:
- Lists of numbers
- Lists and Strings
- List Methods
- Setting up Lists
- Functions that return a list
We Have Seen Them Before

Recall that the rgb encoding of a color involves a triplet of numbers:

\[
\text{MyColor} = [.3,.4,.5]
\]

\[
\text{DrawDisk}(1,2,\text{color=MyColor})
\]

It is a way of assembling a collection of numbers.
A List has a Length

The following would assign the value of 5 to the variable \( n \):

\[
x = [3.0, 5.0, -1.0, 0.0, 3.14]
\]
\[
n = \text{len}(x)
\]
The Entries in a List Can Be Accessed Using Subscripts

The following would assign the value of $-1.0$ to the variable $a$:

```plaintext
x = [3.0, 5.0, -1.0, 0.0, 3.14]
a = x[2]
```
A List Can Be Sliced

This:
```
x = [10,40,50,30,20]
y = x[1:3]
z = x[:3]
w = x[3:]
```

Is same as:
```
x = [10,40,50,30,20]
y = [40,50]
z = [10,40,50]
w = [30,20]
```
Lists Seem to Be Like Strings

A string is a sequence of characters.

A list of numbers is a sequence of numbers.
Lists in Python

Right now we are dealing with lists of numbers.

But in general, the elements in a list can have arbitrary type:

\[ A = [1.0, \text{True}, 'abc', 4.6] \]

The operations on lists that we are about to describe will be illustrated using lists of numbers. But they can be applied to any kind of list.
Visualizing Lists

Informal: x:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

Formal: x ----> 0 ------> 3
1 ------> 5
2 ------> 1
3 ------> 7

A state diagram that shows the “map” from indices to elements.
Lists Vs Strings

There are some similarities.

But there also a huge difference:

1. Strings are **immutable**. They cannot be changed.
2. Lists are **mutable**. They can be change.
Strings are Immutable

Before

s: ['a', 'b', 'c', 'd']

s[2] = 'x'

After

TypeError: 'str' object does not support item assignment

You cannot change the value of a string
Lists ARE Mutable

Before

\[ x: \begin{array}{c}
0 & 1 & 2 & 3 \\
3 & 5 & 1 & 7 \\
\end{array} \]

\[ x[2] = 100 \]

After

\[ x: \begin{array}{c}
0 & 1 & 2 & 3 \\
3 & 5 & 100 & 7 \\
\end{array} \]

You can change the values in a list
Lists ARE Mutable

Before

\[ x[1:3] = [100,200] \]

After

You can change the values in a list
List Methods

When these methods are applied to a list, they affect the list.

- append
- extend
- insert
- sort

They do not return anything. Actually, they return `None` which is Python’s way of saying they do not return anything.
List Methods: append

Before

\[ x: \begin{array}{cccc} 0 & 1 & 2 & 3 \\ 3 & 5 & 1 & 7 \end{array} \]

\[ x = \text{append}(100) \]

After

\[ x: \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 \\ 3 & 5 & 1 & 7 & 100 \end{array} \]

When you want to add an element on the end of a given list.
List Methods: `extend`

<table>
<thead>
<tr>
<th>Before</th>
<th>x:</th>
<th>After</th>
<th>x:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

When you want to add one list onto the end of another list.
List Methods: insert

Before

\[ x: 3 \quad 5 \quad 1 \quad 7 \]

\[ i = 2 \]
\[ a = 100 \]
\[ x.insert(i,a) \]

After

\[ x: 3 \quad 5 \quad 100 \quad 1 \quad 7 \]

When you want to insert an element into the list. Values in \( x[i:] \) get “bumped” to the right and the value \( a \) becomes the new value of \( x[i] \).
List Methods: sort

Before  

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
3 & 5 & 1 & 7 \\
\end{array}
\]

\textbf{x.sort()}

After  

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
1 & 3 & 5 & 7 \\
\end{array}
\]

When you want to sort the elements in a list from little to big.
List Methods: sort

Before

```
x:    3 5 1 7
```

```
x.sort(reverse=True)
```

After

```
x:    7 5 3 1
```

When you want to sort the elements in a list from big to little.
Back to the “Void Business”

These methods do not return anything:

append    extend    insert    sort

So watch its

```python
>>> x = [10, 20, 30]
>>> y = x.append(40)
```

In particular, it appends an element to `x`

```python
>>> print x
[10, 20, 30, 40]
>>> print y
None
```

It returns None and that is assigned to `y`. 
List Methods: pop

When this method is applied to a list, it affects the list but also returns something:

pop
List Methods: pop

Before

\[
\begin{array}{c|c|c|c|c}
& 0 & 1 & 2 & 3 \\
\hline
x: & 3 & 5 & 1 & 7 \\
\end{array}
\]

\[
i = 2
\]

\[
m = x.pop(i)
\]

After

\[
\begin{array}{c|c|c|c|c}
& 0 & 1 & 2 \\
\hline
x: & 3 & 5 & 7 \\
\end{array}
\]

\[
m: & 1
\]

When you want to remove the \(i\)th element and assign it to a variable.
List Methods: count

When this method is applied to a list, it returns something:

    count
List Methods: count

Before

\[ x: \begin{array}{c}
3 \\
7 \\
1 \\
7 \\
\end{array} \]

\[ m = x.count(7) \]

After

\[ x: \begin{array}{c}
3 \\
7 \\
1 \\
7 \\
\end{array} \]

\[ m: \begin{array}{c}
2 \\
\end{array} \]

When you want to sort the elements in a list from big to little.
Built-In Functions that Can be Applied to Lists

len returns the length of a list

sum returns the sum of the elements in a list provided all the elements are numerical.
len and count

Before

\[
\begin{align*}
m &= \text{len}(x) \\
s &= \text{sum}(x)
\end{align*}
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
x: & 3 & 7 & 1 & 5 \\
m: & 4 \\
s: & 16
\end{array}
\]

After
Setting Up Little Lists

The examples so far have all been small.

When that is the case, the “square bracket” notation is just fine for setting up a list.

\[ x = [10, 40, 50, 30, 20] \]

Don’t Forget the Commas!
Working with Big Lists

Setting up a big list will require a loop.

Looking for things in a big list will require a loop.

Let’s look at some examples.
A Big List of Random Numbers

from random import randint as randi
x = []
N = 1000000
for k in range(N):
    r = randi(1,6)
    x.append(r)

The idea here is to keep appending values to x, which starts out as the empty list.

Roll a dice one million times. Record the outcomes in a list.
This Does Not Work

```python
from random import randint as randi
x = []
N = 1000000
for k in range(N):
r = randi(1,6)
x[k] = r
```

```
x[k] = r
IndexError: list assignment index out of range
```
A List of Square Roots

```python
x = []
N = 1000000
for k in range(N):
    s = math.sqrt(k)
x.append(s)
```
from random import randint as randi
x = [0]
k = 0

# x[k] is robot’s location after k hops
while abs(x[k])<=10:
    # Flip a coin and hop right or left
    r = randi(1,2)
    if r==1:
        new_x = x[k]+1
    else:
        new_x = x[k]-1
    k = k+1
    x.append(new_x)
from random import randint as randi
x = [0]
k = 0
# x[k] is robot's location after k hops
while abs(x[k])<=10:
    # Flip a coin and hop right or left
    r = randi(1,2)
    if r==1:
        new_x = x[k]+1
    else:
        new_x = x[k]-1
    k = k+1
x.append(new_x)
Be Careful About Types

This is OK and synonymous with \( x = [0,10] \):

\[
\begin{align*}
  x &= [0] \\
  x &. \text{append}(10)
\end{align*}
\]

This is not OK:

\[
\begin{align*}
  x &= 0 \\
  x &. \text{append}(10)
\end{align*}
\]

AttributeError: 'int' object has no attribute 'append'
Be Careful About Types

```python
>>> x = 0
>>> type(x)
<type 'int'>
>>> x = [0]
>>> type(x)
<type 'list'>
```
Functions and Lists

Let’s start with a function that returns a list.

In particular, a function that returns a list of random integers from a given interval.

Then we will use that function to estimate various probabilities when a pair of dice are rolled.
A List of Random Integers

from random import randint as randi

def randiList(L,R,n):
    """ Returns a length-n list of random integers from interval [L,R]
    PreC: L,R,n ints with L<=R and n>=1
    """
    x = []
    for k in range(n):
        r = randi(L,R)
        x.append(r)
    return x
Outcomes from Two Dice Rolls

Roll a pair of dice N times

Store the outcomes of each dice roll in a pair of length-N lists.

Then using those two lists, create a third list that is the sum of the outcomes in another list.
Outcomes from Two Dice Rolls

Example:

D1: 2 1 5 4

D2: 3 3 4 2

D: 5 4 9 6
N = 1000000
D1 = randiList(1,6,N)
D2 = randiList(1,6,N)

D = []
for k in range(N):
    TwoThrows = D1[k] + D2[k]
    D.append(TwoThrows)
At the start of the loop

N = 4
D = []

for k in range(N):
    TwoThrows = D1[k] + D2[k]
    D.append(TwoThrows)
How It Works

k --&gt; 0
N --&gt; 4
TwoThrows --&gt; 5

TwoThrows = D1[0] + D2[0]

N = 4
D = []
for k in range(N):
    TwoThrows = D1[k] + D2[k]
    D.append(TwoThrows)

D1: 2  1  5  4
D2: 3  3  4  2
D: []
How It Works

\[
\begin{align*}
N &= 4 \\
D &= [] \\
\text{for } k \text{ in range}(N): \\
\quad \text{TwoThrows} &= D1[k] + D2[k] \\
\quad \text{D.append}(\text{TwoThrows})
\end{align*}
\]
How It Works

\begin{align*}
N & \rightarrow 4 \\
D1 & \rightarrow \begin{bmatrix} 2 & 1 & 5 & 4 \end{bmatrix} \\
D2 & \rightarrow \begin{bmatrix} 3 & 3 & 4 & 2 \end{bmatrix} \\
D & \rightarrow [5]
\end{align*}

\begin{align*}
& \text{TwoThrows} = D1[k] + D2[k] \\
& \text{D.append(TwoThrows)}
\end{align*}
How It Works

N = 4
D = []
for k in range(N):
    TwoThrows = D1[k] + D2[k]
    D.append(TwoThrows)
How It Works

N = 4
D = []
for k in range(N):
    TwoThrows = D1[k] + D2[k]
    D.append(TwoThrows)
How It Works

k --> 2
N --> 4
TwoThrows --> 9

D.append(9)

N = 4
D = []
for k in range(N):
    TwoThrows = D1[k] + D2[k]
    D.append(TwoThrows)
How It Works

\[ k \rightarrow 3 \quad N \rightarrow 4 \]


\[ D = [] \]

for \( k \) in range(\( N \)):

\[ \text{TwoThrows} = D1[k] + D2[k] \]

\[ D.append(\text{TwoThrows}) \]
How It Works

\[ k \rightarrow 3 \]
\[ N \rightarrow 4 \]
\[ \text{TwoThrows} \rightarrow 6 \]

\[ \text{TwoThrows} = D1[3] + D2[3] \]

\[ N = 4 \]
\[ D = [] \]
\[ \text{for } k \text{ in range}(N): \]
  - \[ \text{TwoThrows} = D1[k] + D2[k] \]
  - \[ D.\text{append}(\text{TwoThrows}) \]
How It Works

k --> 3
N --> 4
TwoThrows --> 6

D.append(6)

N = 4
D = []
for k in range(N):
    TwoThrows = D1[k] + D2[k]
    D.append(TwoThrows)
How It Works

\[ N = 4 \]
\[ D = [] \]
\[ \text{for } k \text{ in range}(N): \]
\[ \text{TwoThrows} = D1[k] + D2[k] \]
\[ D\text{.append}(\text{TwoThrows}) \]
Now Let's Record all the 2-Throw Outcomes

count = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
for k in range(N):
    i = D[k]
    count[i] = count[i] + 1

count: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

count[2] keeps track of the number of 2's thrown

count[10] keeps track of the number of 10's thrown
Now Let’s Record all the 2-Throw Outcomes

```python
count = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] for k in range(N):
    i = D[k]
    count[i] = count[i]+1
```

The variable $i$ is assigned the outcome of the $k$-th 2-die roll.
Now Let’s Count 2-Throw Outcomes

count = [0,0,0,0,0,0,0,0,0,0,0,0,0]  
for k in range(N):  
    i = D[k]  
    count[i] = count[i]+1

Suppose:  

    i --> 7

count: 0 0 3 1 5 8 7 2 1 6 9 2 1
Now Let's Count 2-Throw Outcomes

count = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
for k in range(N):
    i = D[k]
    count[i] = count[i] + 1

Suppose i --> 7

then the assignment count[i] = count[i] + 1
Now Let's Count 2-Throw Outcomes

count = [0,0,0,0,0,0,0,0,0,0,0,0,0]
for k in range(N):
    i = D[k]
    count[i] = count[i]+1

i --> 7

Before: 0 1 2 3 4 5 6 7 8 9 10 11 12
count: 0 0 3 1 5 8 7 2 1 6 9 2 1

After: 0 1 2 3 4 5 6 7 8 9 10 11 12
count: 0 0 3 1 5 8 7 3 1 6 9 2 1
Now Let's Count 2-Throw Outcomes

count = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]  
for k in range(N):  
    i = D[k]  
    count[i] = count[i] + 1
Sample Results, N = 10000

for k in range(2,13):
    print k,count[k]