13A. Lists of Numbers

Topics:
- Lists of numbers
- List Methods:
  - Void vs Fruitful Methods
- Setting up Lists
  - A Function that returns a list

We Have Seen Lists Before

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

```
MyColor = [.3, .4, .5]
DrawDisk(0, 0, 1, FillColor = MyColor)
```

MyColor is a list.

A list of numbers is a way of assembling a sequence of numbers.

Terminology

\[ x = [3.0, 5.0, -1.0, 0.0, 3.14] \]

How we talk about what is in a list:

- 5.0 is an item in the list \( x \).
- 5.0 is an entry in the list \( x \).
- 5.0 is an element in the list \( x \).
- 5.0 is a value in the list \( x \).

Get used to the synonyms.

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 = len(x)
```

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]
```

The Entries in a List are Accessed Using Subscripts

The following would assign the value of -1.0 to the variable \( a \):

```
x = [3.0, 5.0, -1.0, 0.0, 3.14]
a = x[2]
```
Lists are Similar to Strings

A string is a sequence of characters.
A list of numbers is a sequence of numbers.

Visualizing Lists

Informal:

Formal:

A state diagram that shows the "map" from indices to elements.

Lists vs. Strings

There are some similarities, e.g., subscripts

But there is a huge difference:

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

Exactly what does this mean?

Strings are Immutable

Before:

After: 'str' object does not support item assignment

Lists ARE Mutable

Before:

After: You can change the values in a list
Lists ARE Mutable

Before $x[1:3] = [100,200]$

After $x: 3\ 100\ 200\ 7$

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

Let's see what they do through examples...

List Methods: append

Before: $x: 3\ 5\ 1\ 7$

$x.append(100)$

After: $x: 3\ 5\ 1\ 7\ 100$

Use append when you want to "glue" an item on the end of a given list.

List Methods: extend

Before: $x: 3\ 5\ 1\ 7$

$t = [100,200]$

$x.extend(t)$

After: $x: 3\ 5\ 1\ 7\ 100\ 200$

Use extend when you want to "glue" one list onto the end of another list.

List Methods: insert

Before: $x: 3\ 5\ 1\ 7$

$i = 2$

$a = 100$

$x.insert(i,a)$

After: $x: 3\ 5\ 100\ 1\ 7$

Use insert when you want to insert an item into the list. Items get "bumped" to the right if they are at or to the right of the specified insertion point.

List Methods: sort

Before: $x: 3\ 5\ 1\ 7$

$x.sort()$

After: $x: 1\ 3\ 5\ 7$

Use sort when you want to order the elements in a list from little to big.
**List Methods: sort**

<table>
<thead>
<tr>
<th>Before: x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.sort(reverse=True)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>After: x</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Use sort when you want to order the elements in a list from big to little.

**Void Methods**

When the methods `append` `extend` `insert` `sort` are applied to a list, they affect the list but they do not return anything like a number or string. They are called "void" methods.

Void methods return the value of `None`. This is Python’s way of saying they do not return anything.

**Void Methods/Functions**

The graphics procedures `DrawDisk`, `DrawRect`, etc., are examples of void functions. They also return the value None. But we were never tempted to do something like this:

```
C = DrawDisk(0,0,1)
```

With lists, however, it is tempting to do something like this:

```
newValue = 10
y = x.append(newValue)
```

So we have to be careful!

**Fruitful) List Methods**

When these methods are applied to a list, they actually return something:

- `pop`
- `count`

Let’s see what they do through examples...

**The List Method pop**

<table>
<thead>
<tr>
<th>Before: x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m = x.pop(i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>After: x</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>m:    1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use pop when you want to remove an element and assign it to a variable.
The List Method count

Before:  
\[
\begin{array}{ccc}
0 & 1 & 2 \\
3 & 7 & 1 \\
\end{array}
\]

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

After:  
\[
\begin{array}{ccc}
0 & 1 & 2 \\
3 & 7 & 1 \\
\end{array}
\]

Use count when you want to compute the number of items in a list that have a value.

Two 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 sum

Before:  
\[
\begin{array}{ccc}
0 & 1 & 2 \\
3 & 7 & 1 \\
\end{array}
\]

\[m = len(x)\]

\[s = sum(x)\]

After:  
\[
\begin{array}{ccc}
0 & 1 & 2 \\
3 & 7 & 1 \\
\end{array}
\]

len and sum: Common errors

Legal But Not What You Probably Expect

Legal But Not What You Probably Expect
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 requires a loop. Looking for things in a big list requires a loop. Let’s consider 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)

\[ x \] starts out as an empty list and is built up through repeated appending.

This Does Not Work

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

IndexError: list assignment index out of range

x[0] = r does not work because x is the empty list—it has no components

A List of Square Roots

from math import sqrt
x = []
N = 1000000
for k in range(N):
    s = sqrt(k)
    x.append(s)

Same idea. Create a list through repeated appending.

A Random Walk Example

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)
A Random Walk Example

```python
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)
```

Notice that `x` is initialized as a length-1 list. The robot starts at the origin.

Be Careful About Types

This is OK and synonymous with `x = [0,10]`:

```python
x = [0]
x.append(10)
```

This is not OK:

```python
x = 0
x.append(10)
```

`AttributeError: 'int' object has no attribute 'append'`

You need the square brackets. It is your way of telling Python that `x` is a list, not an int.

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

```python
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:

0 1 2 3
2 1 5 4
0 1 2 3

D2:

0 1 2 3
3 3 4 2

D:

0 1 2 3
5 4 9 6

How to Do It

N = 1,000,000
D1 = randiList(1,6,N)
D2 = randiList(1,6,N)
D = []
for k in range(N):
    TwoThrows = D1[k] + D2[k]
    D.append(TwoThrows)

How It Works

k --> 0
N --> 4

At the start of the loop

D: []

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

k --> 1
N --> 4

TwoThrows --> 5

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

k --> 2
N --> 4

TwoThrows --> 4

N = 4
D = []
for k in range(N):
    TwoThrows = D1[k] + D2[k]
    D.append(TwoThrows)
| k | 1 | D1: 0 1 2 3 | 2 1 5 4 |
| N | 4 | D2: 0 1 2 3 | 3 3 4 2 |
| TwoThrows | 4 | D: 5 4 |

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

| k | 2 | D1: 0 1 2 3 | 2 1 5 4 |
| N | 4 | D2: 0 1 2 3 | 3 3 4 2 |
| TwoThrows | 9 | D: 5 4 |

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

| k | 3 | D1: 0 1 2 3 | 2 1 5 4 |
| N | 4 | D2: 0 1 2 3 | 3 3 4 2 |
| TwoThrows | 6 | D: 5 4 9 |

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

| k | 3 | D1: 0 1 2 3 | 2 1 5 4 |
| N | 4 | D2: 0 1 2 3 | 3 3 4 2 |
| TwoThrows | 6 | D: 5 4 9 6 |

```
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)

All Done!

Tabulating Outcomes

We have simulated the rolling of a pair of dice N times.
The outcomes are recorded in the list D.

New problem:
- How many 2’s were there?
- How many 3’s were there?
- How many 12’s were there?

Tabulating Outcomes

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

Suppose:
i -> 7

then the assignment count[i] = count[i]+1
Tabulating Outcomes

```python
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
```

Before:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>count:</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

After:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>count:</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Overall...

```python
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
```

A list of counters.

Sample Results, N = 10000

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

<table>
<thead>
<tr>
<th>k</th>
<th>count[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>293</td>
</tr>
<tr>
<td>3</td>
<td>629</td>
</tr>
<tr>
<td>4</td>
<td>820</td>
</tr>
<tr>
<td>5</td>
<td>1100</td>
</tr>
<tr>
<td>6</td>
<td>1399</td>
</tr>
<tr>
<td>7</td>
<td>1650</td>
</tr>
<tr>
<td>8</td>
<td>1321</td>
</tr>
<tr>
<td>9</td>
<td>1149</td>
</tr>
<tr>
<td>10</td>
<td>820</td>
</tr>
<tr>
<td>11</td>
<td>527</td>
</tr>
<tr>
<td>12</td>
<td>292</td>
</tr>
</tbody>
</table>