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:

```
MyColor = [.3,.4,.5]
DrawDisk(1,2,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 = 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`:

```
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**

```
s: 'x' 'L' '1' '?' 'a' 'C'
x: 3 5 2 7 0 4
```

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.

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 
\[ s = ['a', 'b', 'c', 'd'] \]

After

\[ s[2] = 'x' \]

TypeError: 'str' object does not support item assignment

You cannot change the value of a string.

Lists ARE Mutable

Before 
\[ x = [3, 5, 1, 7] \]

\[ x[2] = 100 \]

After 
\[ x = [3, 5, 100, 7] \]

You can change the values in a list.

Lists ARE Mutable

Before 
\[ x = [3, 5, 1, 7] \]

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

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`  

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: [3, 5, 1, 7]
```

After:  

```
x.append(100)
```

When you want to add an element 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]
```

When you want to add 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, 100, 5, 1, 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:  

```
x: [3, 5, 1, 7]
```

```
x.sort()
```

After:  

```
x: [1, 3, 5, 7]
```

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

x.append(40) does something to x.
In particular, it appends an element to x.
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:

```python
pop
```

List Methods: pop

When you want to remove the ith element and assign it to a variable.

```
Before  x: 3 5 1 7
i = 2
m = x.pop(i)
After   x: 3 5 7
m: 1
```

List Methods: count

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

```python
count
```

```
Before  x: 3 7 1 7
m = x.count(7)
After   x: 3 7 1 7
m: 2
```

List Methods: count

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

Built-In Functions that Can be Applied to Lists

```python
len
```

returns the length of a list

```python
sum
```

returns the sum of the elements in a list provided all the elements are numerical.
### 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 Comma!

### 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

Roll a dice one million times. Record the outcomes in a list.

```python
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
```

```
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)
```
A Random Walk

```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)
```

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.

Example:

\[
\begin{array}{cccc}
D1: & 2 & 1 & 5 & 4 \\
D2: & 3 & 3 & 4 & 2 \\
D: & 5 & 4 & 9 & 6 \\
\end{array}
\]

How to Do It

\[
N = 1000000 \\
D1 = \text{randiList}(1,6,N) \\
D2 = \text{randiList}(1,6,N) \\
D = [] \\
\text{for } k \text{ in range}(N): \\
\quad \text{TwoThrows = D1}[k] + D2[k] \\
\quad D.\text{append}(\text{TwoThrows})
\]

How It Works

\[
\begin{array}{cccc}
k \rightarrow & 0 & \text{D1:} & 2 & 1 & 5 & 4 \\
N \rightarrow & 4 & \text{D2:} & 3 & 3 & 4 & 2 \\
\text{TwoThrows \rightarrow} & 5 & D: & [ ] \\
\end{array}
\]
### How It Works

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

#### Example

<table>
<thead>
<tr>
<th>k →</th>
<th>1</th>
<th>N →</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwoThrows →</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
TwoThrows = D1[1] + D2[1]
D: 5
```

<table>
<thead>
<tr>
<th>k →</th>
<th>2</th>
<th>N →</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwoThrows →</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
D: 5 4
```

<table>
<thead>
<tr>
<th>k →</th>
<th>3</th>
<th>N →</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwoThrows →</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
D: 5 4 9
```

<table>
<thead>
<tr>
<th>k →</th>
<th>3</th>
<th>N →</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwoThrows →</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
D: 5 4 9
```
How It Works

```
D1: 2 1 5 4
D2: 3 3 4 2
TwoThrows = D1[k] + D2[k]
D.append(TwoThrows)
```

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

All Done!

D: 5 4 9 6

Now Let's Record all the 2-Throw Outcomes

```
count = [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]
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.

Suppose: `i --> 7`

```
count = [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]
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

```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: count](image)

Sample Results, \( N = 10000 \)

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