### 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]
У	=	<b>x</b> [1:3]
z	=	x[:3]
W	=	x[3:]

Is same as:

$$\mathbf{x} = [10, 40, 50, 30, 20] \mathbf{y} = [40, 50] \mathbf{z} = [10, 40, 50] \mathbf{w} = [30, 20]$$

### 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

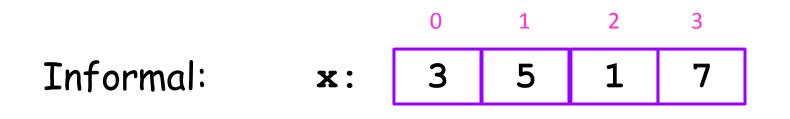
Right now we are dealing with lists of numbers.

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

$$A = [1.0, True, 'abc', 4.6]$$

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



Formal:  $x \rightarrow 0 \rightarrow 3$   $1 \rightarrow 5$   $2 \rightarrow 1$ A state diagram that shows the "map" from indices to  $x \rightarrow 0 \rightarrow 3$   $1 \rightarrow 5$   $2 \rightarrow -2$   $3 \rightarrow -2$  $3 \rightarrow -2$ 

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

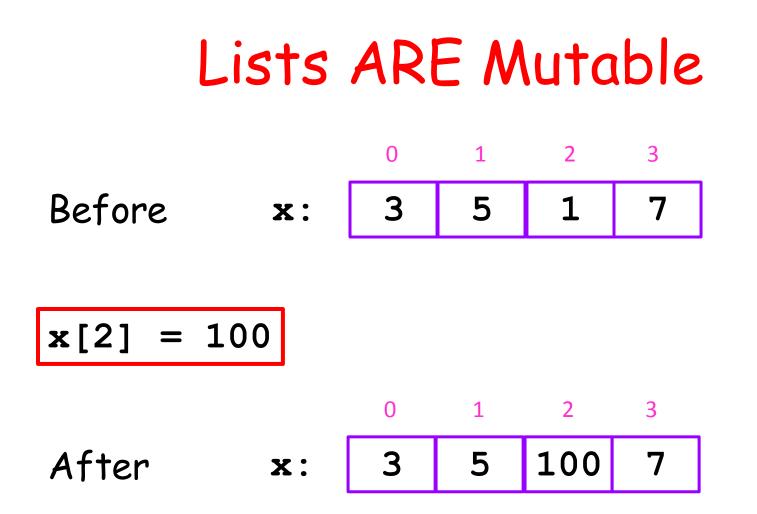
Before s: *`a' `b' `c' `d'* 

$$s[2] = 'x'$$

After

TypeError: 'str' object does not support item assignment

You cannot change the value of a string



You can change the values in a list

#### Lists ARE Mutable Before **x** : x[1:3] = [100,200]100 200 After **x**:

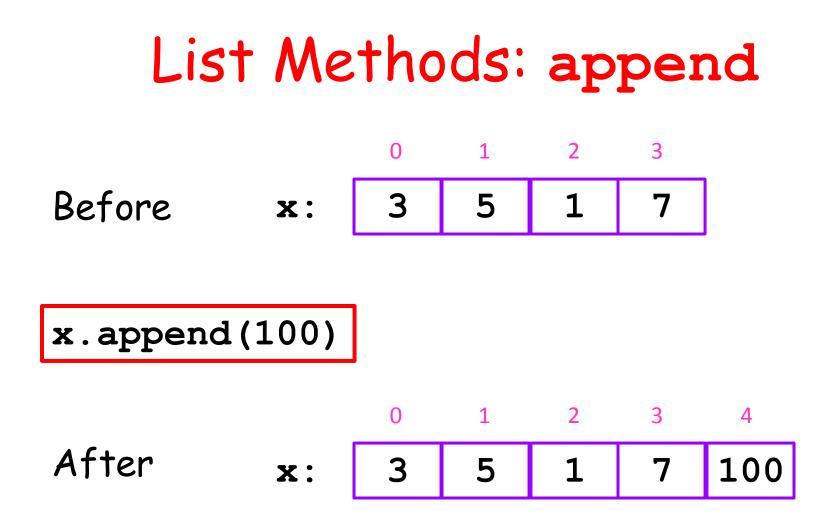
You can change the values in a list

### List Methods

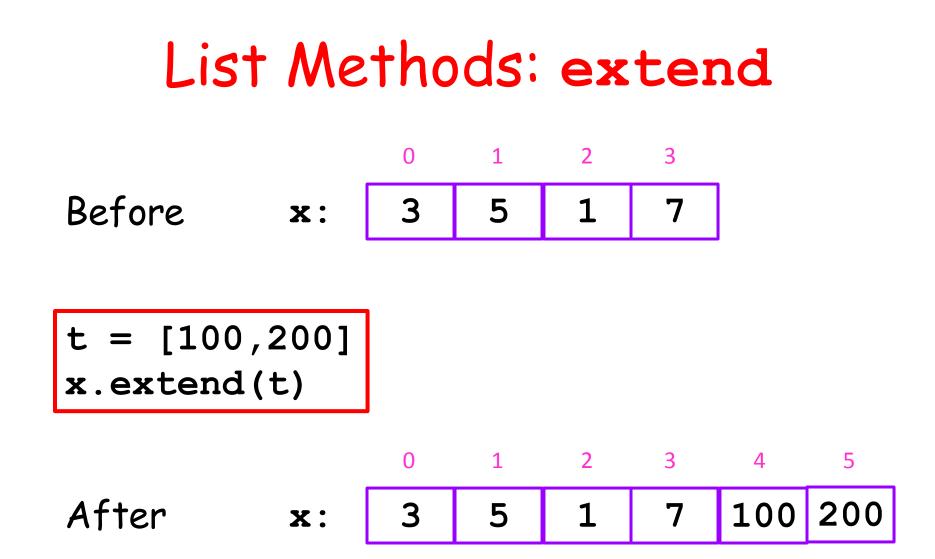
When these methods are applied to a list, they affect the list.

append extend insert sort

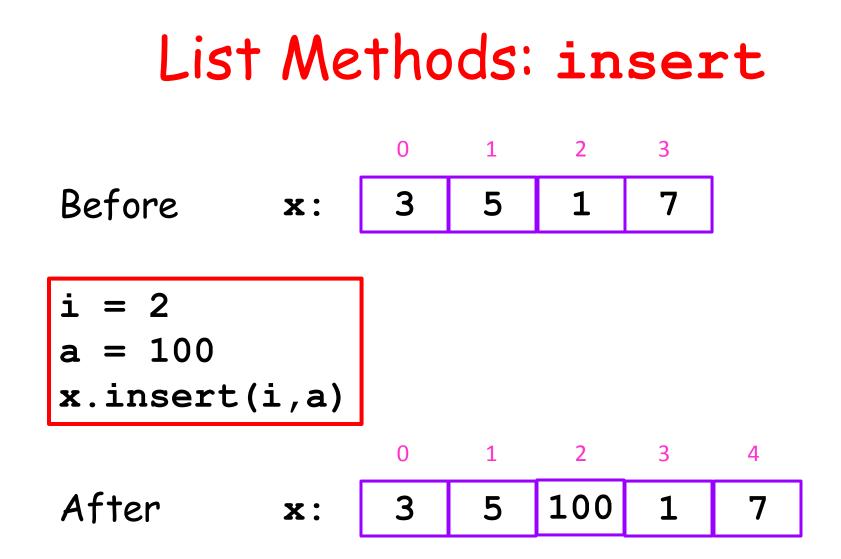
They do not return anything. Actually, they return **None** which is Python's way of saying they do not return anything.



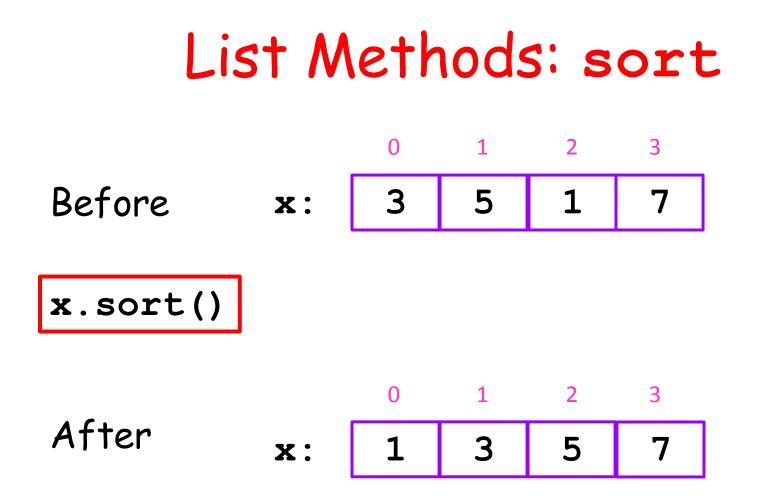
When you want to add an element on the end of a given list.



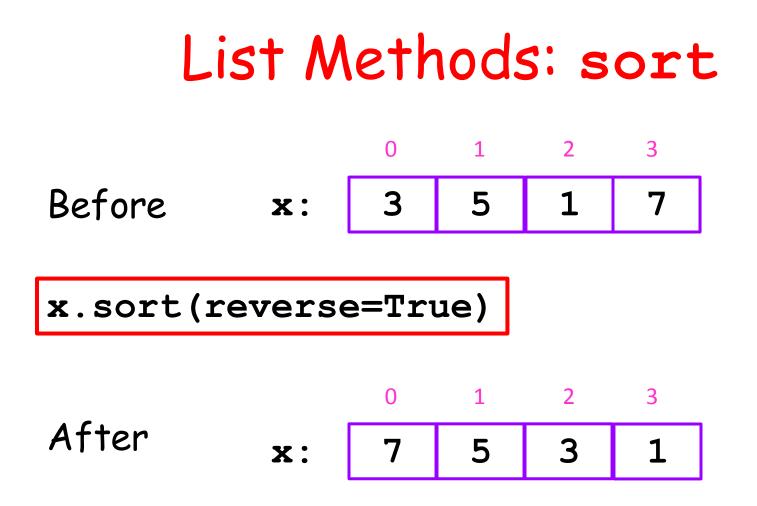
When you want to add one list onto the end of another list.



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].



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



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

>>> x = [10,20,30]
>>> y = x.append(40)
>>> print x
[10, 20, 30, 40]
>>> print y
None

x.append(40) does something to x.

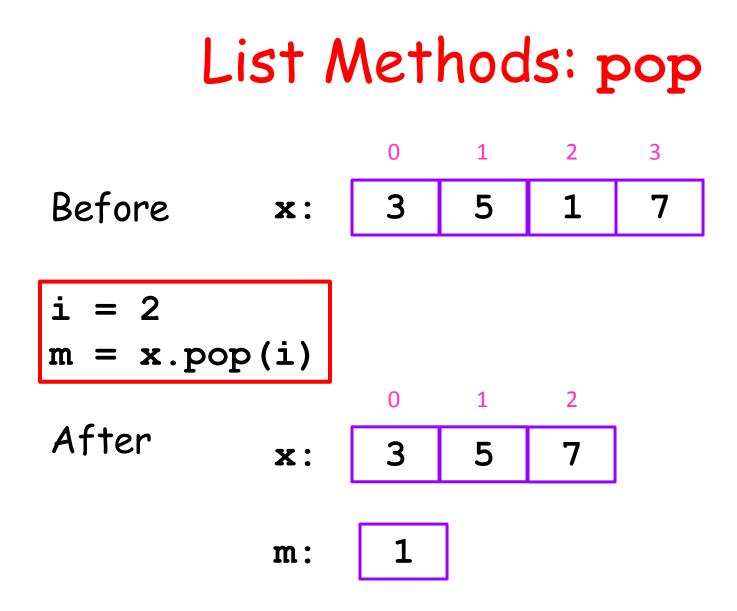
In particular, it appends an element to x

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:

pop

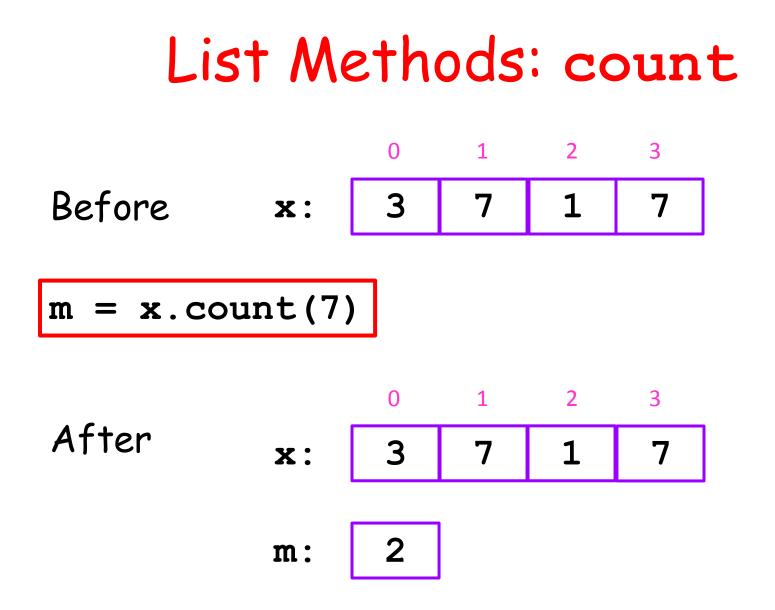


When you want to remove the ith element and assign it to a variable.

### List Methods: count

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

count

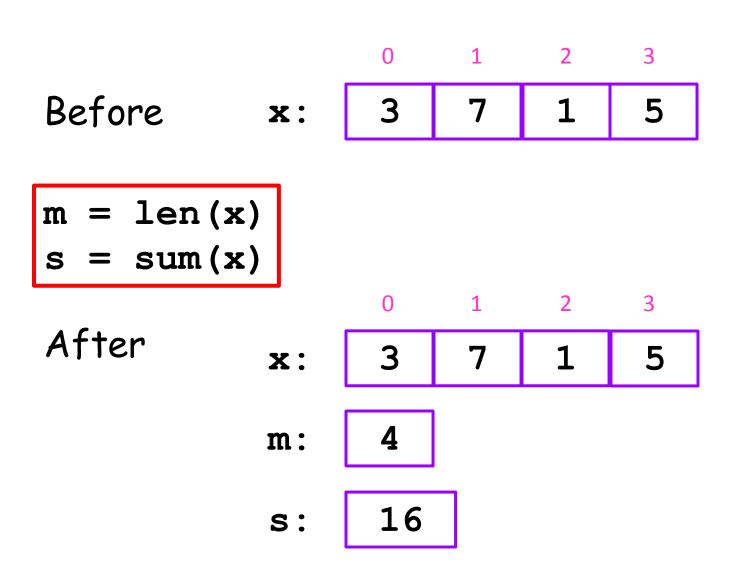


When you want to sort the elements in a list from big to little.

### 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



### 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

$$\mathbf{x} = [10, 40, 50, 30, 20]$$

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

```
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

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

### A Random Walk

```
from random import randint as randi
\mathbf{x} = [0]
\mathbf{k} = \mathbf{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

```
from random import randint as randi
\mathbf{x} = [0]
\mathbf{k} = \mathbf{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]:

x = [0] x.append(10)

This is not OK: x = 0 x.append(10)

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

### Be Careful About Types

### 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
    II II II
    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
```

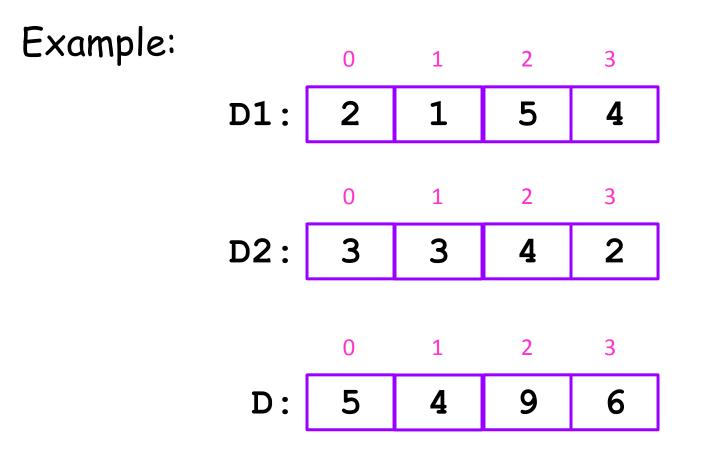
# **Outcomes from Two Dice Rolls**

Roll a pair of dice N times

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

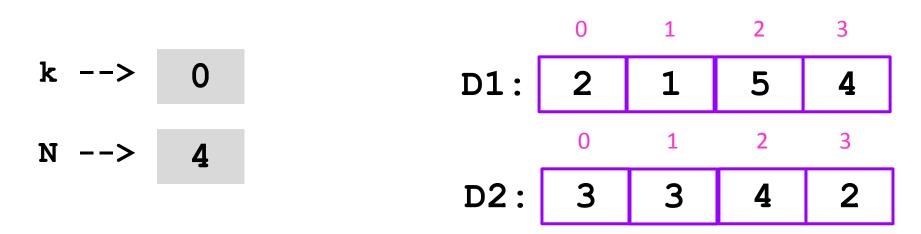
Then using those two lists, create a third list that is the sum of the outcomes in another list.

# **Outcomes from Two Dice Rolls**



# How to Do It

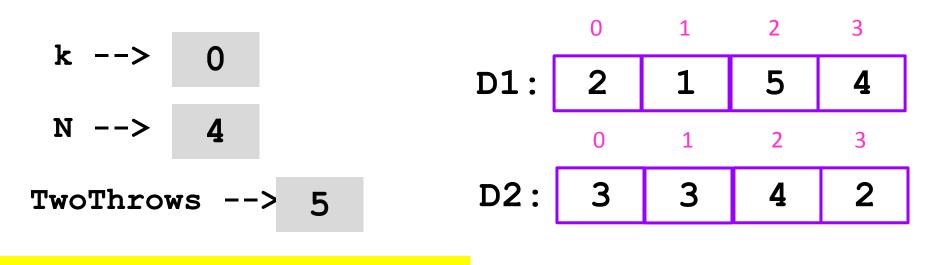
```
N = 1000000
D1 = randiList(1, 6, N)
D2 = randiList(1, 6, N)
D = []
for k in range(N):
   TwoThrows = D1[k] + D2[k]
   D.append(TwoThrows)
```



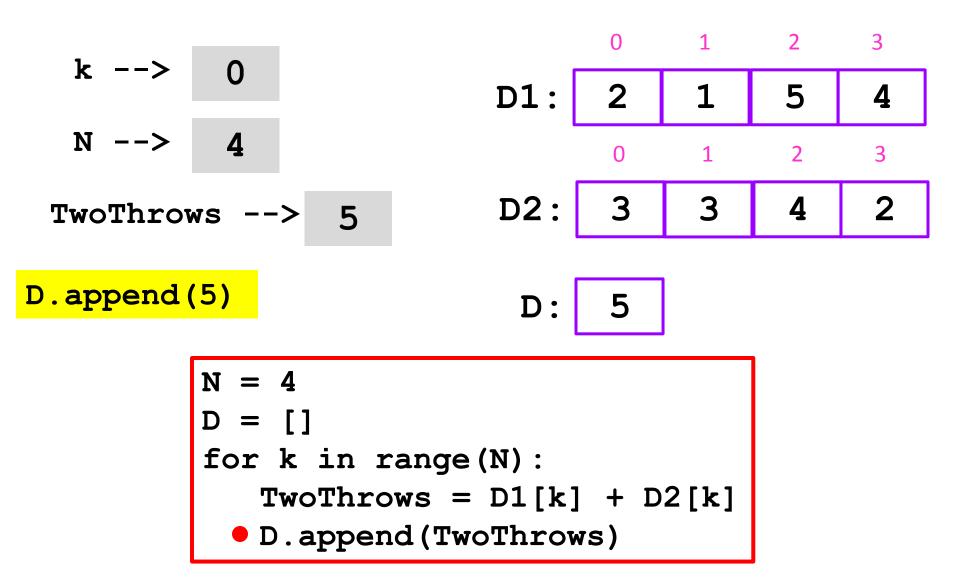
At the start of the loop

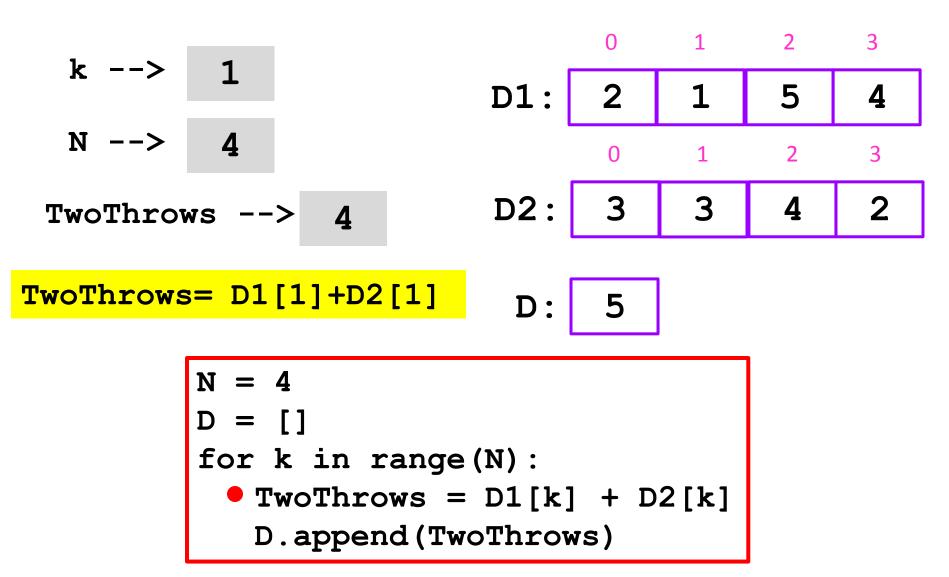
D: []

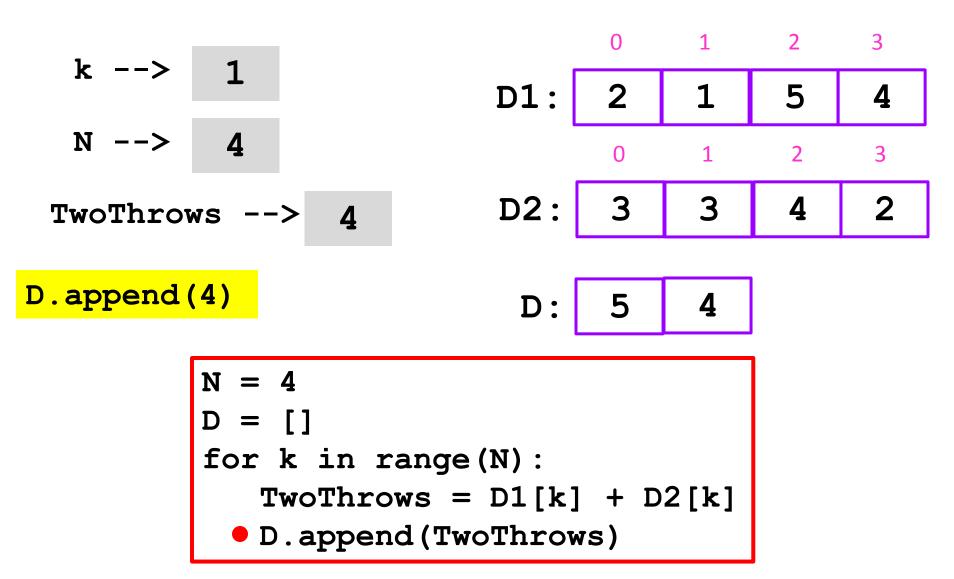
N = 4
D = []
for k in range(N):
 TwoThrows = D1[k] + D2[k]
 D.append(TwoThrows)

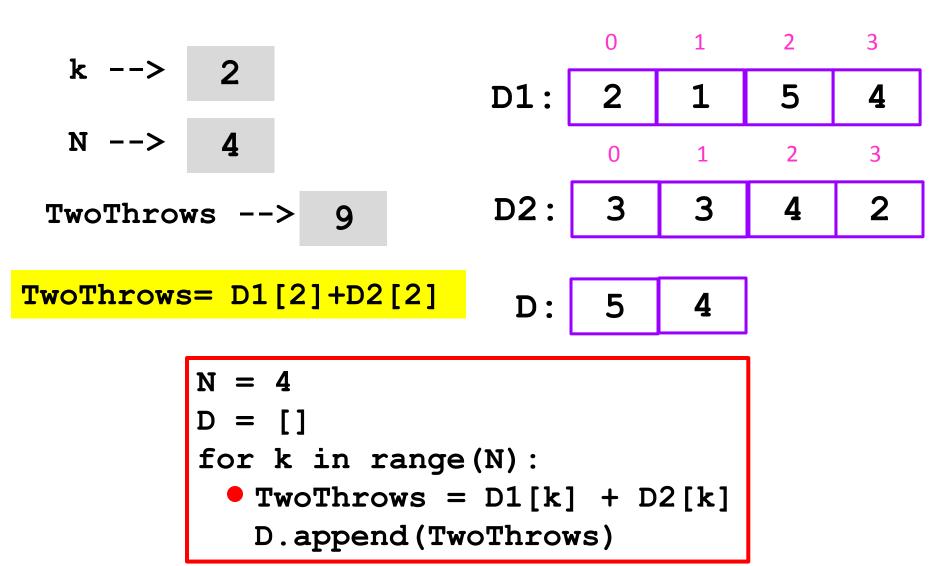


 $\frac{\text{TwoThrows} = D1[0] + D2[0]}{D: []}$ 









		0	1	2	3
k> 2	D1:	2	1	5	4
N> 4		0	1	2	3
TwoThrows> 9	D2:	3	3	4	2
	_				
D.append(9)	D:	5	4	9	
N = 4					
D = []					

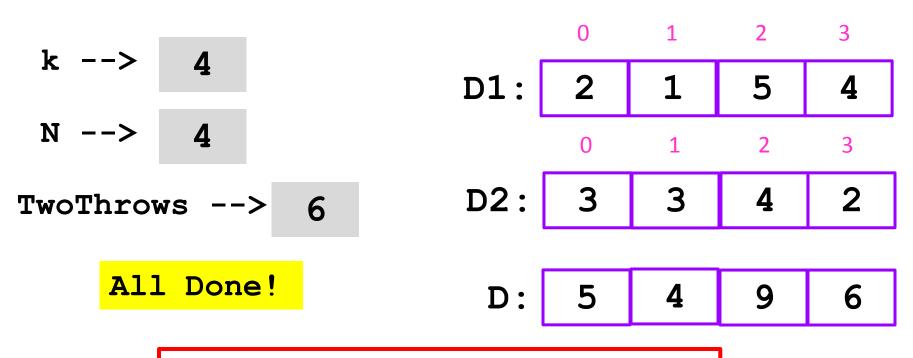
		0	1	2	3
k> 3	D1:	2	1	5	4
N> 4		0	1	2	3
TwoThrows> 9	D2:	3	3	4	2
	_				
TwoThrows = D1[3]+D2[3]	D:	5	4	9	
N = 4 $D = []$					J

۲

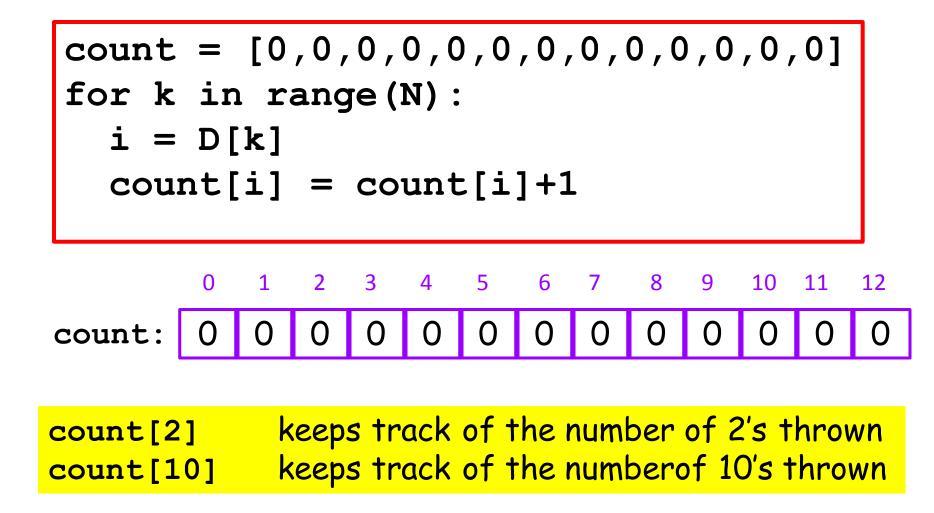
		0	1	2	3
k> 3	D1:	2	1	5	4
N> 4		0	1	2	3
TwoThrows> 6	D2:	3	3	4	2
TwoThrows = D1[3]+D2[3]	D:	5	4	9	
N = 4					I

۲

		0	1	2	3
k> 3	D1:	2	1	5	4
N> 4		0	1	2	3
TwoThrows> 6	D2:	3	3	4	2
D.append(6)	D:	5	4	9	6



### Now Let's Record all the 2-Throw Outcomes



### 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

The variable i is assigned the outcome of the k-th 2-die roll.

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:

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

effectively says count[7] = count[7]+1

count for k i = cour	in D[	ra [k]	ang	<b>je (</b> )	N):	·		·	0,0	),0	,0,	0]	
	i	>		7									
Before:	0	1	2	3	4	5				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

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

# Sample Results, N = 10000

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

k	count[k]
2	293
3	629
4	820
5	1100
6	1399
7	1650
8	1321
9	1149
10	820
11	527
12	292