

# Python Lists

## Chapter 8

# It is a kind of Collection

A **collection** allows us to put many values in a single “**variable**”

A **collection** is nice because we can carry all **many values** around in a convenient package.

```
names = [ 'Joseph', 'Glenn', 'Sally' ]
```

```
clothes = [ 'socks', 'shirt', 'perfume' ]
```

# What is not a “Collection”

our **variables** have one value in them - when we put a new value in the **variable**, the old value is overwritten

```
Python
```

```
Python 2.5.2 (r252:60911, Feb 22 2008, 07:57:53)
```

```
>>> 4.0.1 (Apple Computer, Inc. build 5363) [on Darwin10.0.0: Tue Aug 11 22:03:10 PDT 2009; root:xnu-160.0.0/RELEASE_ARM_T8020]
```

```
>>> x = 2
```

```
>>> x = 4
```

```
>>> print x
```

# List Constants

List constants are surrounded by square brackets and the elements in the list are separated by commas

Any element can be any Python object, even another list

A list can be empty

```
>>> print [1, 24, 76]
[1, 24, 76]
>>> print ['red', 'yellow', 'blue']
['red', 'yellow', 'blue']
>>> print ['red', 24, 98.5999]
['red', 24, 98.5999]
>>> print [1, [5, 6], 7]
[1, [5, 6], 7]
>>> print []
[]
```

# We already use lists!

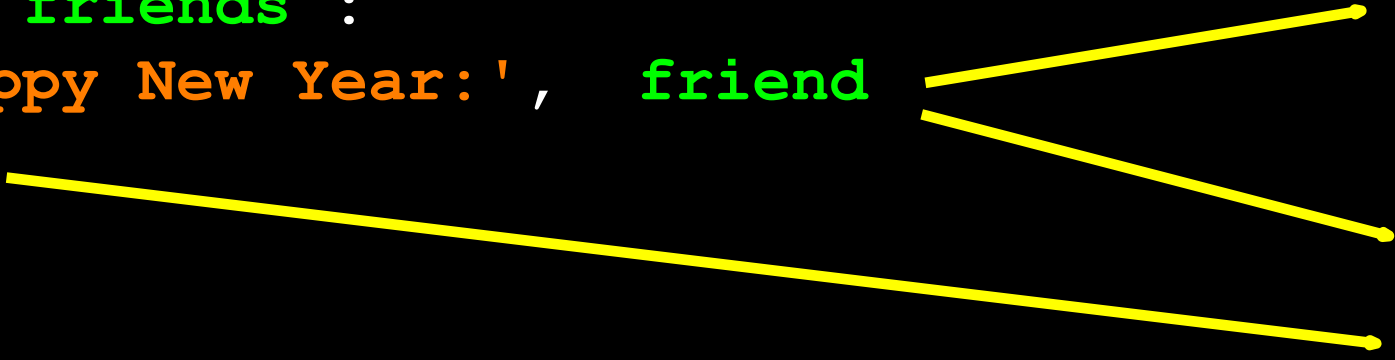
```
i in [5, 4, 3, 2, 1] :  
print i  
t 'Blastoff! '
```

5  
4  
3  
2  
1  
Blastoff!

# and definite loops - best

```
friends = ['Joseph', 'Glenn', 'Sally']  
for friend in friends :  
    print 'Happy New Year:', friend  
print 'Done!'
```

Happy New Year:  
Happy New Year:  
Happy New Year:  
Done!



# Looking Inside Lists

strings, we can get at any single element in a list using the index specified in **square brackets**

Glenn

Sally

1

2

```
>>> friends = [ 'Joseph', 'Glenn' ]
>>> print friends[1]
Glenn
>>>
```

# Lists are Mutable

are “immutable” - we  
change the contents of a  
we must make a new  
make any change

“mutable” - we can  
an element of a list  
e index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object
support item assignment
>>> x = fruit.lower()
>>> print x
banana
>>> lotto = [2, 14, 26, 41, 63]
>>> print lotto
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print lotto
[2, 14, 28, 41, 63]
```



# How Long is a List?

`len()` function takes a **list** as a parameter and returns the number of **elements** in the **list**

`len()` tells us the number of items of *any* set or sequence (a string...)

```
>>> greet = 'Hello, world!'
>>> print len(greet)
9
>>> x = [1, 2, 3, 4]
>>> print len(x)
4
>>>
```

# Using the range function

The function returns a list of numbers that range from zero up to less than the parameter

to construct an index loop and an integer iterator

```
>>> print range(4)
[0, 1, 2, 3]
>>> friends = ['Joseph', 'G1', 'G2']
>>> print len(friends)
3
>>> print range(len(friends))
[0, 1, 2]
>>>
```

# A tale of two loops...

```
Joseph', 'Glenn', 'Sally']
```

```
friends :  
Happy New Year:', friend
```

```
range(len(friends)) :  
friends[i]  
Happy New Year:', friend
```

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
```

```
>>> print len(friends)
```

```
3
```

```
>>> print range(len(friends))
```

```
[0, 1, 2]
```

```
>>>
```

```
Happy New Year: Joseph
```

```
Happy New Year: Glenn
```

```
Happy New Year: Sally
```

# Concatenating lists using +

create a new list by adding  
existing lists together

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
>>> print a
[1, 2, 3]
```

# Lists can be sliced using :

```
[9, 41, 12, 3, 74, 15]
```

```
[2, 3]
```

```
[5]
```

```
[2, 3, 74, 15]
```

**Remember:** *Just like in strings, the second number is “up to but not including”*

# List Methods

```
= list()  
type(x)  
'list'  
dir(x)  
'append', 'count', 'extend', 'index', 'insert',  
, 'remove', 'reverse', 'sort']
```

# Building a List from Scratch

create an empty **list**  
then add elements using  
**append** method

stays in order and  
elements are **added** at  
end of the **list**

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print stuff
['book', 99]
>>> stuff.append('cookie')
>>> print stuff
['book', 99, 'cookie']
```

# Is Something in a List?

Provides two  
operators that let you check  
if something is in a list

Logical operators  
return True or False

Does not modify the list

```
>>> some = [1, 9, 21, 10]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```



# List is an Ordered Sequence

can hold many items and  
keep those items in the order  
you do something to  
change the order

can be sorted  
(change its order)

sort method (unlike in  
Python means “sort yourself”

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> friends.sort()
>>> print friends
['Glenn', 'Joseph', 'Sally']
>>> print friends[1]
Joseph
>>>
```

# Built-in Functions and Lists

There are a number of  
functions built into Python  
that take lists as parameters

Remember the loops we  
did before these are much

```
>>> nums = [3, 41, 12, 74, 3, 154]
>>> print len(nums)
6
>>> print max(nums)
74
>>> print min(nums)
3
>>> print sum(nums)
154
>>> print sum(nums)/len(nums)
25.5
```

```
:
raw_input('Enter a number: ')
== 'done' : break
float(inp)
total + value
count + 1

total / count
age:', average
```

```
numlist = list()
while True :
    inp = raw_input('Enter a num
    if inp == 'done' : break
    value = float(inp)
    numlist.append(value)

average = sum(numlist) / len(num
print 'Average:', average
```

Enter a number: 3  
Enter a number: 9  
Enter a number: 5  
Enter a number: 0  
Average: 5.66666

# Best Friends: Strings and Lists

```
abc = 'With three words'
stuff = abc.split()
print stuff
['With', 'three', 'words']
print len(stuff)
print stuff[0]
```

```
>>> print stuff
['With', 'three', 'words']
>>> for w in stuff:
...     print w
...
With
Three
Words
>>>
```

splits a string into parts and produces a list of strings. We then iterate through the list. We can access a particular word or loop through all the words.

```
line.split()
```

```
['of', 'spaces']
```

```
first;second;third'
```

```
line.split()
```

```
ng
```

```
d;third']
```

```
(thing)
```

```
line.split(';')
```

```
ng
```

```
cond', 'third']
```

```
(thing)
```

If you do not specify a **delimiter**, multiple spaces are treated like one.

You can specify what **delimiter** character to use in the **splitting**

[stephen.marquard@uct.ac.za](mailto:stephen.marquard@uct.ac.za) Sat Jan 5 09:14:16 2008

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print words[2]
```

```
'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
words = line.split()
words[2] = 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008'
```

# The Double Split Pattern

When we split a line one way, and then grab one of the pieces and split that piece again

```
phen.marquard@uct.ac.za Sat Jan 5 09:14:16 2
```

```
line.split()  
words[1]
```

# The Double Split Pattern

When we split a line one way, and then grab one of the pieces and split that piece again

```
stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2006
```

```
lines[0].split()
```

```
words[1]
```

```
stephen.marquard@uct.ac.za
```



# The Double Split Pattern

Sometimes we split a line one way, and then grab one of the pieces and split that piece again

```
stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
line.split()
```

```
words[1]
```

```
mail.split('@')
```

```
stephen.marquard@uct.ac.za
```

```
['stephen.marquard', 'uct.ac.za']
```

# The Double Split Pattern

Sometimes we split a line one way, and then grab one of the pieces and split that piece again

stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
line.split()
```

```
words[1]
```

```
mail.split('@')
```

```
pieces[1]
```

stephen.marquard@uct.ac.za

['stephen.marquard', 'uct.ac.za']

'uct.ac.za'

# List Summary

of a collection

and definite loops

g and lookup

ability

s: len, min, max, sum

- Slicing lists

- List methods: append,

- Sorting lists

- Splitting strings into list

- Using split to parse string