

8 – Working with Lists

(part of Python collection types)

(Course: Python Programming)

Overview of (object) data types in Python

Name	Type	Description
Integers	int	Whole numbers, such as: 3 300 200
Floating point	float	Numbers with a decimal point: 2.3 4.6 100.0
Strings	str	Ordered sequence of characters: "hello" 'Sammy' "2000" "楽しい"
Lists	list	Ordered sequence of objects: [10,"hello",200.3]
Dictionaries	dict	Unordered Key:Value pairs: {"mykey" : "value" , "name" : "Frankie"}
Tuples	tup	Ordered immutable sequence of objects: (10,"hello",200.3)
Sets	set	Unordered collection of unique objects: {"a","b"}
Booleans	bool	Logical value indicating True or False

What is Not a “Collection”?

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

```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```

A List 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 one convenient package.

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

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

List Constants

- **List** constants are surrounded by square brackets and the elements in the list are separated by commas
- A **list** 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, 98.6])
['red', 24, 98.6]
>>> print([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

We Already Use Lists!

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

5

4

3

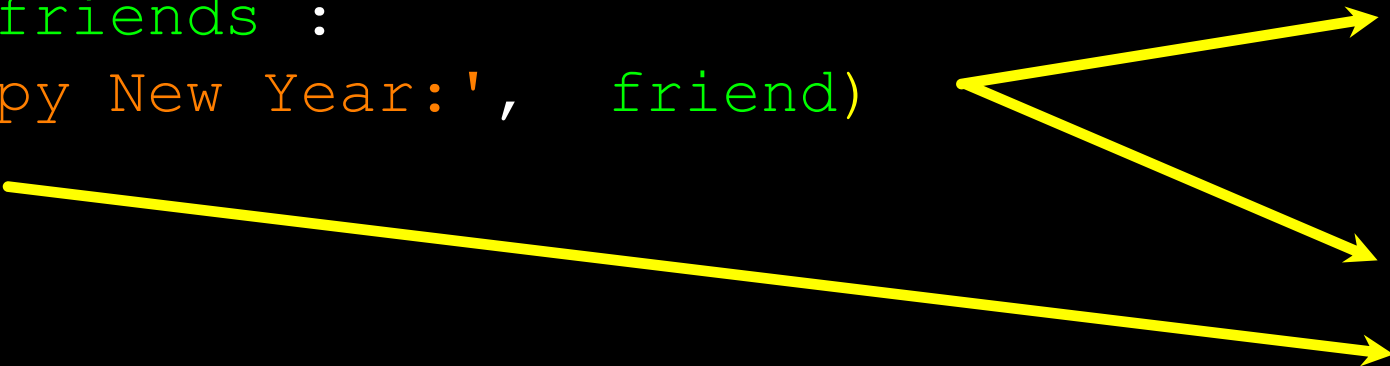
2

1

Blastoff!

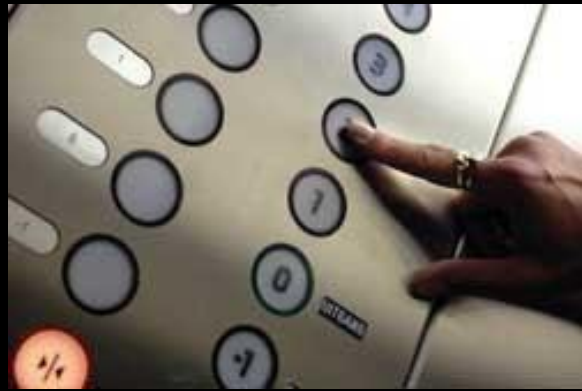
Lists and Definite Loops - Best Pals

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



Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!

```
z = ['Joseph', 'Glenn', 'Sally']  
for x in z:  
    print('Happy New Year:', x)  
print('Done!')
```



Looking Inside Lists

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

Joseph	Glenn	Sally
0	1	2

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


Lists are Mutable

- Strings are “immutable” - we cannot change the contents of a string - we must make a new string to make any change
- Lists are “mutable” - we can change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not 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?

- The `len()` function takes a `list` as a parameter and returns the number of `elements` in the `list`
- Actually `len()` tells us the number of elements of any set or sequence (such as a string...)

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
>>>
```

Using the range Function

- The range function returns a list of numbers that range from zero to one less than the parameter
- We can construct an index loop using for and an integer iterator

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

A Tale of Two Loops...

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

for friend in friends :
    print('Happy New Year:', friend)

for i in range(len(friends)) :
    friend = friends[i]
    print('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 +

We can create a new list
by adding two 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 :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41, 12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

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

List Methods

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

<http://docs.python.org/tutorial/datastructures.html>

Building a List from Scratch

- We can create an empty **list** and then add elements using the **append** method
- The **list** stays in order and new elements are **added** at the 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?

- Python provides two **operators** that let you check if an item is in a list
- These are logical operators that return **True** or **False**
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

Lists are in Order

- A **list** can hold many items and keeps those items in the order until we do something to change the order
- A **list** can be **sorted** (i.e., change its order)
- The **sort** method (unlike in strings) 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 built? These are much simpler.

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

```
total = 0
count = 0
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    total = total + value
    count = count + 1
```

```
average = total / count
print('Average:', average)
```

Enter a number: 3

Enter a number: 9

Enter a number: 5

Enter a number: done

Average: 5.66666666666667

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

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

Best Friends: Strings and Lists

```
>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print(stuff)
['With', 'three', 'words']
>>> print(len(stuff))
3
>>> print(stuff[0])
With
```

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

Split breaks a string into parts and produces a list of strings. We think of these as words. We can **access** a particular word or **loop** through all the words.

```
>>> line = 'A lot of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>> print(len(thing))
1
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
>>>
```

- When you do not specify a **delimiter**, multiple spaces are treated like one delimiter
- You can specify what **delimiter** character to use in the **splitting**

From 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])
```

Sat

Fri

Fri

Fri

...

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

The Double Split Pattern

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

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

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


The Double Split Pattern

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

```
words = line.split()
```

```
email = words[1]
```

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

The Double Split Pattern

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

```
words = line.split()  
email = words[1]  
pieces = email.split('@')
```

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

The Double Split Pattern

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

```
words = line.split()  
email = words[1]  
pieces = email.split('@')  
print(pieces[1])
```

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

List Summary

- Concept of a collection
- Lists and definite loops
- Indexing and lookup
- List mutability
- Functions: len, min, max, sum
- Slicing lists
- List methods: append, remove
- Sorting lists
- Splitting strings into lists of words
- Using split to parse strings



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here