Data Collections
Zelle - Chapter 11

Charles Severance - www.dr-chuck.com

Textbook: Python Programming: An Introduction to Computer Science, John Zelle

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 over written

```
$ python
Python 2.5.2 (r252:60911, Feb 22 2008, 07:57:53)
[GCC 4.0.1 (Apple Computer, Inc. build 5363)] on darwin
>>> x = 2
>>> x = 4
>>> print x
4
```

What is a Collection?

- A collection is nice because we can put more than one value in them and carry them all around in one convenient package.
- We have a bunch of values in a single "variable"
- We do this by having more than one place "in" the variable.
- We have ways of finding the different places in the variable

A Story of Two Collections..

- List
  - A linear collection of values that stay in order

- Dictionary
  - A "bag" of values, each with its own label

The Python List Object

```
>>> grades = list()

>>> grades.append(100)
>>> grades.append(97)
>>> grades.append(100)

>>> print sum(grades)
297

>>> print grades
[100, 97, 100]

>>> print sum(grades)/3.0
99.0
>>>
```

The grades variable will have a list of values.

Append some values to the list.

Add up the values in the list using the sum() function.

What is in the list?

Figure the average...

```
>>> print grades
[100, 97, 100]

>>> newgr = list(grades)

>>> print newgr
[100, 97, 100]

>>> newgr[1] = 85

>>> print newgr
[100, 85, 100]

>>> print grades
[100, 97, 100]
```

What is in grades?

Make a copy of the entire grades list.

Change the second new grade (starts at [0])

The original grades are unchanged.

Looking in Lists...

- We use square brackets to look up which element in the list we are interested in.
- grades[2] translates to “grades sub 2”
- Kind of like in math x_2

```
>>> print grades
[100, 97, 100]

>>> print grades[0]
100

>>> print grades[1]
97

>>> print grades[2]
100
```

Why lists start at zero?

- Initially it does not make sense that the first element of a list is stored at the zeroth position
  - grades[0]
- Math Convention - Number line
- Computer performance - don't have to subtract 1 in the computer all the time

Elevators in Europe!

Fun With Lists

- Python has many features that allow us to do things to an entire list in a single statement
- Lists are powerful objects

```
>>> lst = [ 21, 14, 4, 3, 12, 18]
>>> print lst
[21, 14, 4, 3, 12, 18]
>>> print 18 in lst
True
>>> print 24 in lst
False
>>> lst.append(50)
>>> print lst
[21, 14, 4, 3, 12, 18, 50]
>>> lst.remove(4)
>>> print lst
[21, 14, 3, 12, 18, 50]
```

```
>>> print lst
[21, 14, 3, 12, 18, 50]
>>> print lst.index(18)
4
>>> lst.reverse()
>>> print lst
[50, 18, 12, 3, 14, 21]
>>> lst.sort()
>>> print lst
[3, 12, 14, 18, 21, 50]
>>> del lst[2]
>>> print lst[3, 12, 18, 21, 33]
```

More functions for lists

```
>>> a = [ 1, 2, 3 ]
>>> print max(a)
3
>>> print min(a)
1
>>> print len(a)
3
>>> print sum(a)
6
>>>
```

http://docs.python.org/lib/built-in-funcs.html

```
>>>print lst
[3,12,14,18,21,33]
>>>for  xval in lst:
...     print xval
...
3
12
14
18
21
33
>>>
```

Looping through Lists

List Operations

Operator	Meaning
<seq> + <seq>	Concatenation
<seq> * <int-expr>	Repetition
<seq>[ ]	Indexing
len(<seq>)	Length
<seq>[:]	Slicing
for <var> in <seq>:	Iteration
<expr> in <seq>	Membership check (Returns a Boolean)

Method	Meaning
<list>.append(x)	Add element x to end of list.
<list>.sort()	Sort (order) the list. A comparison function may be passed as parameter.
<list>.reverse()	Reverse the list.
<list>.index(x)	Returns index of first occurrence of x.
<list>.insert(i,x)	Insert x into list at index i.
<list>.count(x)	Returns the number of occurrences of x in list.
<list>.remove(x)	Deletes the first occurrence of x in list.
<list>.pop(i)	Deletes the ith element of the list and returns its value.

Quick Peek: Object Oriented

<nerd-alert>

What "is" a List Anyways?

- A list is a special kind of variable
- Regular variables - integer
  - Contain some data
- Smart variables - string, list
  - Contain some data and capabilities

```
>>> i = 2
>>> i = i + 1
>>> x = [1, 2, 3]
>>> print x
[1, 2, 3]
>>> x.reverse()
>>> print x
[3, 2, 1]
```

When we combine data + capabilities - we call this an "object"

One way to find out Capabilities

Method	Meaning
<list>.append(x)	Add element x to end of list.
<list>.sort()	Sort (order) the list. A comparison function may be passed as parameter.
<list>.reverse()	Reverse the list.
<list>.index(x)	Returns index of first occurrence of x.
<list>.insert(i,x)	Insert x into list at index i.
<list>.count(x)	Returns the number of occurrences of x in list.
<list>.remove(x)	Deletes the first occurrence of x in list.
<list>.pop(i)	Deletes the ith element of the list and returns its value.

Buy a book and read it and carry it around with you.

Lets Ask Python...

- The dir() command lists capabilities
  - Ignore the ones with underscores - these are used by Python itself
  - The rest are real operations that the object can perform
- It is like type() - it tells us something *about* a variable

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
['__add__', '__class__', '__contains__',
'__delattr__', '__delitem__',
'__delslice__', '__doc__',
'__eq__"__setitem__', '__setslice__',
'__str__', 'append', 'count', 'extend',
'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>>
```

Try dir() with a String

```
>>> y = “Hello there”
>>> dir(y)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__',
'__lt__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__str__',
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs',
'find', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

What does x = list() mean?

- These are called "constructors" - they make an empty list, str, or dictionary
- We can make a "fully formed empty" object and then add data to it using capabilities (aka methods)

```
>>> a = list()
>>> print a
[]
>>> print type(a)
<type 'list'>
>>> b = dict()
>>> print b
{}
>>> print type(b)
<type 'dict'>
>>> a.append("fred")
>>> print a
['fred']
>>> c = str()
>>> d = int()
>>> print d
0
```

Object Oriented Summary

- Variables (Objects) contain data and capabilities
- The dir() function asks Python to list capabilities
- We call object capabilities "methods"
- We can construct fresh, empty objects using constructors like list()
- Everything in Python (even constants) are objects

Python Dictionaries

http://en.wikipedia.org/wiki/Associative_array

Dictionaries

- Dictionaries are Python's most powerful data collection
- Dictionaries allow us to do fast database-like operations in Python
- Dictionaries have different names in different languages
  - Associative Arrays - Perl / Php
  - Properties or Map or HashMap - Java
  - Property Bag - C# / .Net

http://en.wikipedia.org/wiki/Associative_array

Dictionaries

- Lists label their entries based on the position in the list
- Dictionaries are like bags - no order
- So we mark the things we put in the dictionary with a "tag"

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print purse
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print purse['candy']
3
>>> purse['candy'] = purse['candy'] + 2
>>> print purse
{'money': 12, 'tissues': 75, 'candy': 5}
```

```
>>> purse = dict()

>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75

>>> print purse
{'money': 12, 'tissues': 75, 'candy': 3}

>>> print purse['candy']
3

>>> purse['candy'] = purse['candy'] + 2

>>> print purse
{'money': 12, 'tissues': 75, 'candy': 5}
```

Lookup in Lists and Dictionaries

- Dictionaries are like Lists except that they use keys instead of numbers to look up values

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print lst
[21, 183]
>>> lst[0] = 23
>>> print lst
[23, 183]
```

```
>>> ddd = dict()
>>> ddd["age"] = 21
>>> ddd["course"] = 182
>>> print ddd
{'course': 182, 'age': 21}
>>> ddd["age"] = 23
>>> print ddd
{'course': 182, 'age': 23}
```

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print lst
[21, 183]
>>> lst[0] =23
>>> print lst
[23, 183]
```

List

Key	Value
[0]	21
[1]	183

lll

```
>>> ddd = dict()
>>> ddd["age"] = 21
>>> ddd["course"] = 182
>>> print ddd{'course': 182, 'age': 21}
>>> ddd["age"] = 23
>>> print ddd
{'course': 182, 'age': 23}
```

Dictionary

Key	Value
[course]	183
[age]	21

ddd

Dictionary Operations

Method	Meaning
`<dict>.has_key(<key>)`	Returns true if dictionary contains the specified key, false if it doesn't.
`<key> in <dict>`	Same as `has_key`
`<dict>.keys()`	Returns a list of the keys.
`<dict>.values()`	Returns a list of the values.
`<dict>.items()`	Returns a list of tuples `(key,value)` representing the key-value pairs.
`<dict>.get(<key>, <default>)`	If `key` is not in the dictionary, returns `default`; otherwise returns the value for `key`.
`del <dict>[<key>]`	Delete the specified entry.
`<dict>.clear()`	Delete all entries.

Dictionary Literals (Constants)

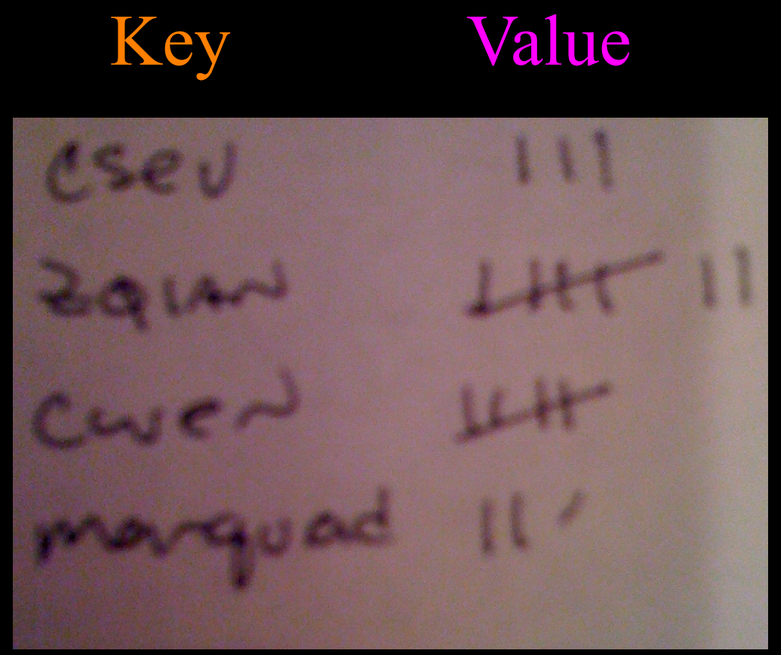
- Dictionary literals use curly braces and have a list of key : value pairs
- You can make an empty dictionary using empty curly braces

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print jjj
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print ooo
{}
>>>
```

Dictionary Patterns

- One common use of dictionary is counting how often we "see" something

```
>>> ccc = dict()
>>> ccc["csev"] = 1
>>> ccc["cwen"] = 1
>>> print ccc
{'csev': 1, 'cwen': 1}
>>> ccc["cwen"] = ccc["cwen"] + 1
>>> print ccc
{'csev': 1, 'cwen': 2}
```



Dictionary Patterns

- It is an error to reference a key which is not in the dictionary
- We can use the in operator to see if a key is in the dictionary

```
>>> ccc = dict()
>>> print ccc["csev"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> print "csev" in ccc
False
```

in

print “No”

in

print “Yes”

Dictionary Counting

- Since it is an error to reference a key which is not in the dictionary
- We can use the dictionary get() operation and supply a default value if the key does not exist to avoid the error and get our count started.

```
>>> ccc = dict()
>>> print ccc.get("csev", 0)
0
>>> ccc["csev"] = ccc.get("csev",0) + 1
>>> print ccc
{'csev': 1}
>>> print ccc.get("csev", 0)
1
>>> ccc["csev"] = ccc.get("csev",0) + 1
>>> print ccc
{'csev': 2}
```

dict.get(key, defaultvalue)

What get() effectively does...

- The get() method basically does an implicit if checking to see if the key exists in the dictionary and if the key is not there - return the default value
- The main purpose of get() is to save typing this four line pattern over and over

```
d = dict()
x = d.get("fred",0)
```

```
d = dict()
if "fred" in d:
    x = d["fred"]
else:
    x = 0
```

Retrieving lists of Keys and Values

- You can get a list of keys, values or items (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print jjj.keys()
['jan', 'chuck', 'fred']
>>> print jjj.values()
[100, 1, 42]
>>> print jjj.items()
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

Looping Through Dictionaries

- We loop through the key-value pairs in a dictionary using *two* iteration variables
- Each iteration, the first variable is the key and the the second variable is the corresponding value

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for aaa,bbb in jjj.items() :
...     print aaa, bbb
...
jan 100
chuck 1
fred 42
>>>
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

Dictionary Maximum Loop

```
$ cat dictmax.py
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
print jjj

maxcount = None
for person, count in jjj.items() :
    if maxcount == None or count > maxcount :
        maxcount = count
        maxperson = person

print maxperson, maxcount
```

```
$ python dictmax.py
{'jan': 100, 'chuck': 1, 'fred': 42}
jan 100
```

None is a special value in Python. It is like the "absense" of a value. Like "nothing" or "empty".

Dictionaries are not Ordered

- Dictionaries use a Computer Science technique called "hashing" to make them very fast and efficient
- However hashing makes it so that dictionaries are not sorted and they are not sortable
- Lists and sequences maintain their order and a list can be sorted - but not a dictionary

http://en.wikipedia.org/wiki/Hash_function

Dictionaries are not Ordered

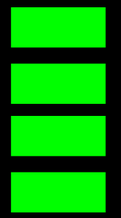
```
>>> dict = { "a" : 123, "b" : 400, "c" : 50 }
>>> print dict
{'a': 123, 'c': 50, 'b': 400}
```

Dictionaries have no order and cannot be sorted. Lists have order and can be sorted.

```
>>> lst = dict()
>>> lst.append("one")
>>> lst.append("and")
>>> lst.append("two")
>>> print lst
['one', 'and', 'two']
>>> lst.sort()
>>> print lst
['and', 'one', 'two']
>>>
```

http://en.wikipedia.org/wiki/Hash_function

Summary: Two Collections



- List
  - A linear collection of values that stay in order

- Dictionary
  - A "bag" of values, each with its own label / tag

What do we use these for?

- Lists - Like a Spreadsheet - with columns of stuff to be summed, sorted - Also when pulling strings apart - like string.split()
- Dictionaries - For keeping track of (keyword,value) pairs in memory with very fast lookup. It is like a small in-memory database. Also used to communicate with databases and web content.