Data Collections
Zelle - Chapter 11
Charles Severance - www.dr-chuck.com

Textbook: Python Programming: An Introduction to Computer Science, John Zelle (www.si182.com)
Data Collections

- Sequences - 11.2
- Lists / Arrays - 11.2
- Dictionaries - 11.6
- For this lecture - avoid 11.3, 11.4, and 11.5 - these deal with classes and objects - which we will cover later
Sequences

• We already are familiar with sequences

• Strings are a sequence of characters

• The string.split("hello there bob") returns a sequence of three strings ['hello', 'there', 'bob']

• The range(5) gives us a sequence [0, 1, 2, 3, 4]

• Sequences can be indexed seq[4] and we know their length len(seq)
Sequences are structures that have lengths and we can use [position] to get at a particular element of the sequence.
• We store data in a sequence and then we can use the index [0] to retrieve the contents stored in a particular position in the list.

• A sequence is like a little file cabinet or spreadsheet with numbers as the labels to retrieve data or to look up a row.

• Always remember that the index labels start at zero, not 1.

Index | Contents
---|---
[0] | Hello
[1] | Bob

>>> import string
>>> www = string.split(zzz)
>>> print len(www)
2
>>> print www
['Hello', 'bob']
>>> print www[0]
Hello
A List of Characters

```python
>>> zzz = "Hello Bob"
>>> print len(zzz)
9
>>> print zzz[4]
'o'
```

![Indexing of the string "Hello Bob"](image)

*Figure 4.1: Indexing of the string "Hello Bob"*
A List of Strings

- The string.split() function returns a list of strings.
- We can find the first word on the line by retrieving the first element of the list that came back from split().

```python
>>> zzz = "Hello Bob"
>>> import string
>>> www = string.split(zzz)
>>> print len(www)
2
>>> print www
['Hello', 'Bob']
>>> print www[0]
Hello
```
A List of Numbers

>>> yyy = range(4)
>>> print yyy
[0, 1, 2, 3]
>>> print len(yyy)
4
>>> print yyy[2]
2

The range() function returns a list of integer numbers from zero up to range.
Sequence Operators

- Sequences operations apply to all sequences
- Not just strings

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;seq&gt; + &lt;seq&gt;</code></td>
<td>Concatenation</td>
</tr>
<tr>
<td><code>&lt;seq&gt; * &lt;int-expr&gt;</code></td>
<td>Repetition</td>
</tr>
<tr>
<td><code>&lt;seq&gt;[ ]</code></td>
<td>Indexing</td>
</tr>
<tr>
<td><code>len(&lt;seq&gt;)</code></td>
<td>Length</td>
</tr>
<tr>
<td><code>&lt;seq&gt;[ : ]</code></td>
<td>Slicing</td>
</tr>
<tr>
<td>for <code>&lt;var&gt; in &lt;seq&gt;</code>:</td>
<td>Iteration</td>
</tr>
<tr>
<td><code>&lt;expr&gt; in &lt;seq&gt;</code></td>
<td>Membership check (Returns a Boolean)</td>
</tr>
</tbody>
</table>
Slicing a List of Numbers

>>> sss = [3, 41, 12, 9, 74, 15]
>>> ttt = sss[2:5]
>>> print ttt
[12, 9, 74]

Slicing goes from beginning to one before end.
Slicing Strings

- We can also look at any continuous section of a string using a colon.
- The second number is one beyond the end of the slice - “up to but not including”.
- If a number is omitted it is assumed to be the the beginning or end.

```python
>>> greet = "Hello Bob"
>>> greet[0:3]
'Hel'
>>> greet[5:9]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
'Hello Bob'
```
Lists
Lists and Sequences

- A List is a Sequence that you can Edit and Modify

- Sometimes we actually have been using Lists without knowing it

- As an example, you can change an entry in a List but not in a Sequence

```python
>>> import string
>>> str = "Hello There Bob"
>>> www = string.split(str)
>>> print www
['Hello', 'There', 'Bob']
>>> www[0] = 'Howdy'
>>> print www
['Howdy', 'There', 'Bob']
>>> str[2] = 'x'
Traceback (most recent call last):
  File "<stdin>" , line 1, in <module>
TypeError: 'str' object does not support item assignment
```
## List Operations

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;seq&gt; + &lt;seq&gt;</code></td>
<td>Concatenation</td>
</tr>
<tr>
<td><code>&lt;seq&gt; * &lt;int-exp&gt;</code></td>
<td>Repetition</td>
</tr>
<tr>
<td><code>&lt;seq&gt;[ ]</code></td>
<td>Indexing</td>
</tr>
<tr>
<td><code>len(&lt;seq&gt;)</code></td>
<td>Length</td>
</tr>
<tr>
<td><code>&lt;seq&gt;[ : ]</code></td>
<td>Slicing</td>
</tr>
<tr>
<td><code>for &lt;var&gt; in &lt;seq&gt;</code>:</td>
<td>Iteration</td>
</tr>
<tr>
<td><code>&lt;expr&gt; in &lt;seq&gt;</code></td>
<td>Membership check (Returns a Boolean)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;list&gt;.append(x)</code></td>
<td>Add element x to end of list.</td>
</tr>
<tr>
<td><code>&lt;list&gt;.sort()</code></td>
<td>Sort (order) the list. A comparison function may be passed as parameter.</td>
</tr>
<tr>
<td><code>&lt;list&gt;.reverse()</code></td>
<td>Reverse the list.</td>
</tr>
<tr>
<td><code>&lt;list&gt;.index(x)</code></td>
<td>Returns index of first occurrence of x.</td>
</tr>
<tr>
<td><code>&lt;list&gt;.insert(i,x)</code></td>
<td>Insert x into list at index i.</td>
</tr>
<tr>
<td><code>&lt;list&gt;.count(x)</code></td>
<td>Returns the number of occurrences of x in list.</td>
</tr>
<tr>
<td><code>&lt;list&gt;.remove(x)</code></td>
<td>Deletes the first occurrence of x in list.</td>
</tr>
<tr>
<td><code>&lt;list&gt;.pop(i)</code></td>
<td>Deletes the ith element of the list and returns its value.</td>
</tr>
</tbody>
</table>
>>> lll = [21, 14, 4, 3, 12, 18]
>>> print lll
[21, 14, 4, 3, 12, 18]
>>> print 18 in lll
True
>>> print 24 in lll
False
>>> lll.append(50)
>>> print lll
[21, 14, 3, 12, 18, 50]
>>> lll.remove(4)
>>> print lll
[21, 14, 3, 12, 18, 50]
>>> print lll.index(18)
4
>>> lll.reverse()
>>> print lll
[50, 18, 12, 3, 14, 21]
>>> lll.sort()
>>> print lll
[3, 12, 14, 18, 21, 50]
>>> lll[5] = 33
>>> print lll
[3, 12, 14, 18, 21, 33]

z-343
>>> print lll
[3, 12, 14, 18, 21, 33]
>>> for xval in lll :
...    print xval
...
3
12
14
18
21
33

Looping through Lists
Dictionaries - Python’s Built-In “Database”
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
  - Map or HashMap - Java

http://en.wikipedia.org/wiki/Associative_array
Keys and Values

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

```python
>>> lll = []
>>> lll.append(21)
>>> lll.append(183)
>>> print lll
[21, 183]
>>> lll[0] = 23
>>> print lll
[23, 183]
>>> ddd = { }
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print ddd
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print ddd
{'course': 182, 'age': 23}
```
>>> lll = [ ]
>>> lll.append(21)
>>> lll.append(183)
>>> print lll
[21, 183]
>>> lll[0] = 23
>>> print lll
[23, 183]

>>> ddd = { }
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print ddd
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print ddd
{'course': 182, 'age': 23}
## Dictionary Operations

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;dict&gt;.has_key(&lt;key&gt;)</code></td>
<td>Returns true if dictionary contains the specified key, false if it doesn’t. Same as <code>has_key</code></td>
</tr>
<tr>
<td><code>&lt;key&gt; in &lt;dict&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;dict&gt;.keys()</code></td>
<td>Returns a list of the keys.</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.values()</code></td>
<td>Returns a list of the values.</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.items()</code></td>
<td>Returns a list of tuples <code>(key, value)</code> representing the key-value pairs.</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.get(&lt;key&gt;, &lt;default&gt;)</code></td>
<td>If key is not in the dictionary, returns default; otherwise returns the value for key.</td>
</tr>
<tr>
<td><code>del &lt;dict&gt;[&lt;key&gt;]</code></td>
<td>Delete the specified entry.</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.clear()</code></td>
<td>Delete all entries.</td>
</tr>
</tbody>
</table>
Dictionary Literals

• Dictionary literals use curly braces and have a list of key : value pairs

• You can make an empty dictionary using empty curly braces

```python
>>> j jj = { 'chuck' : 1, 'fred' : 42, 'jan' : 100}
>>> print j jj
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = {}
>>> print ooo
{}
>>>```
Dictionary Patterns

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

```python
>>> ccc = {}
>>> 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

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

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

dict.get(key, defaultvalue)
Retrieving lists of Keys and Values

- You can get a list of keys or values from a dictionary

```python
>>> jjj = { 'chuck' : 1, 'fred' : 42, 'jan': 100}
>>> print jjj.keys()
['jan', 'chuck', 'fred']
>>> print jjj.values()
[100, 1, 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 second variable is the corresponding value

```python
>>> 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 j jj

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

print maxperson, maxcount

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

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

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

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

http://en.wikipedia.org/wiki/Hash_function
<advanced-trick-stuff>
Dictionary Tricky / Clever Code

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

Find and print the key which corresponds to the maximum value.
Dictionary Tricky / Clever Code

```python
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print jjj.keys()[jjj.values().index(max(jjj.values()))]
jan
>>> print jjj.values()
[100, 1, 42]
>>> print max(jjj.values())
100
>>> print jjj.values().index(100)
0
>>> print jjj.keys()[0]
jan
```
$ cat dictmax.py
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
print j jj

maxcount = None
for person, count in j jj.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

Do it this way - it is clearer and makes more sense to someone reading your code.
# You can put anything into dictionaries, including lists and other dictionaries
# Make a list of lucky numbers by person
lotto = {'botimer': [1, 7, 13, 37, 42, 47]}

# The lists inside a dictionary work just like other lists
# Print my fourth lotto number
print lotto['botimer'][3]

# Set up a little address book by uniqnmae
people = {'botimer': {'firstname': 'Noah', 'lastname': 'Botimer'},
          'csev': {'firstname': 'Charles', 'lastname': 'Severance'}}

# When you have multiple levels, you can "drill" into them by using multiple indexes
# Print my first name: Noah
print people['botimer']['firstname']
</advanced-trick-stuff>
## Dictionary Operations

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;dict&gt;.has_key(&lt;key&gt;)</code></td>
<td>Returns true if dictionary contains the specified key, false if it doesn’t. Same as <code>has_key</code></td>
</tr>
<tr>
<td><code>&lt;key&gt; in &lt;dict&gt;</code></td>
<td>Same as <code>has_key</code></td>
</tr>
<tr>
<td><code>&lt;dict&gt;.keys()</code></td>
<td>Returns a list of the keys.</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.values()</code></td>
<td>Returns a list of the values.</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.items()</code></td>
<td>Returns a list of tuples <code>(key, value)</code> representing the key-value pairs.</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.get(&lt;key&gt;, &lt;default&gt;)</code></td>
<td>If key is not in the dictionary, returns default; otherwise returns the value for key.</td>
</tr>
<tr>
<td><code>del &lt;dict&gt;[&lt;key&gt;]</code></td>
<td>Delete the specified entry.</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.clear()</code></td>
<td>Delete all entries.</td>
</tr>
</tbody>
</table>
Summary

- Sequences
- Lists - Editable Sequences
- Dictionaries - Simple database within Python - maps keys to values