More Built-in Container Classes

- Container Class `dict`
- Container Class `set`
- Container Class `tuple`
- Encoding of String Characters
- Randomness and Random Sampling
Dictionaries

Lists associate indices (integers) with values

```python
>>> lst = ['csc', '241', 'section', '405']
>>> lst[2]
'section'
```

Sometimes we wish to associate non-integer “keys” with values.
Dictionaries

Solution: the dictionary class `dict`

A dictionary contains `(key, value)` pairs

A key can be used as an index to access the corresponding value, And may be something other than an integer
Dictionaries

A key can be used as an index to access the corresponding value,
And may be something other than an integer

```python
>>> ssns = dict()
>>> ssns[‘john smith’] = ‘111-11-1111’
>>> ssns[‘mary jones’] = ‘222-22-2222’
>>> ssns[‘jane johnson’] = ‘333-33-3333’
>>> ssns[‘mary jones’]
‘222-22-2222’
```
Properties of dictionaries

Not ordered

Mutable

• new (key,value) pairs can be added
• the value corresponding to a key can be modified

The empty dictionary is {} 

Dictionary keys must be immutable (for example, keys should not be lists)
Initialization

>>> ssns = {'john smith': '111-11-1111',
         'mary jones': '222-22-2222',
         'jane johnson': '333-33-3333'}

>>> ssns
{'john smith': '111-11-1111', 'jane johnson': '333-33-3333', 'mary jones': '222-22-2222'}

Order is unimportant

>>> ssns['mary jones'] = '444-44-4444'

>>> ssns['mary jones']
'444-44-4444'
Iteration

Unless you specify otherwise, dictionary iteration is over the **keys**

```
>>> list(ssns)
['john smith', 'jane johnson', 'mary jones']
```

```
>>> items = [ ]
>>> for item in ssns:
    items.append(item)
>>> items
['john smith', 'jane johnson', 'mary jones']
```
## Dictionary methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>d.items()</code></td>
<td>Returns a view of the (key, value) pairs in <code>d</code></td>
</tr>
<tr>
<td><code>d.keys()</code></td>
<td>Returns a view of the keys of <code>d</code></td>
</tr>
<tr>
<td><code>d.pop(key)</code></td>
<td>Removes the (key, value) pair with key <code>key</code> from <code>d</code> and returns the value</td>
</tr>
<tr>
<td><code>d.update(d2)</code></td>
<td>Adds the (key, value) pairs of dictionary <code>d2</code> to <code>d</code></td>
</tr>
<tr>
<td><code>d.values()</code></td>
<td>Returns a view of the values of <code>d</code></td>
</tr>
</tbody>
</table>

```python
donext >>> days
{'Mo': 1, 'Tu': 2, 'Th': 4, 'W': 3}
>>> days.pop('Tu')
2
>>> days
{'Mo': 1, 'Th': 4, 'W': 3}
>>> days2 = {'Tu':2, 'Fr':5}
>>> days.update(days2)
>>> days
{'Fr': 5, 'W': 3, 'Th': 4, 'Mo': 1, 'Tu': 2}
>>> days.items()
>>> days.keys()
>>> for val in vals:
    print(val, end=’ ‘)
5 3 4 1 2
```
## Dictionary vs. multi-way if statement

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`def complete(abbreviation):</td>
<td>'returns day of the week corresponding to abbreviation'</td>
</tr>
<tr>
<td>if abbreviation == 'Mo':</td>
<td></td>
</tr>
<tr>
<td>return 'Monday'</td>
<td></td>
</tr>
<tr>
<td>elif abbreviation == 'Tu':</td>
<td></td>
</tr>
<tr>
<td>return 'Tuesday'</td>
<td></td>
</tr>
<tr>
<td>else: # abbreviation must be Su</td>
<td></td>
</tr>
<tr>
<td>return 'Sunday'</td>
<td></td>
</tr>
<tr>
<td>```</td>
<td></td>
</tr>
<tr>
<td>def complete(abbreviation):</td>
<td></td>
</tr>
<tr>
<td>'returns day of the week corresponding to abbreviation'</td>
<td></td>
</tr>
<tr>
<td>```</td>
<td></td>
</tr>
<tr>
<td>days = {'Mo': 'Monday', 'Tu': 'Tuesday', 'We': 'Wednesday',</td>
<td></td>
</tr>
<tr>
<td>'Th': 'Thursday', 'Fr': 'Friday', 'Sa': 'Saturday',</td>
<td></td>
</tr>
<tr>
<td>'Su': 'Sunday'}</td>
<td></td>
</tr>
<tr>
<td>return days[abbreviation]</td>
<td></td>
</tr>
</tbody>
</table>
Dictionary as a container of counters

Uses of a dictionary:
- container with custom indexes
- alternative to the multi-way if statement
- container of counters

Problem: computing the number of occurrences of items in a list

```python
>>> grades = [95, 96, 100, 85, 95, 90, 95, 100, 100]
>>> frequency(grades)
{96: 1, 90: 1, 100: 3, 85: 1, 95: 3}
``` 

Solution: Iterate through the list and, for each grade, increment the counter corresponding to the grade.

Problems:
- impossible to create counters before seeing what’s in the list
- how to store grade counters so a counter is accessible using the corresponding grade

Solution: a dictionary mapping a grade (the key) to its counter (the value)
Dictionary as a container of counters

Problem: computing the number of occurrences of items in a list

```
>>> grades = [95, 96, 100, 85, 90, 95, 100, 100]
```

```
def frequency(itemList):
    "returns frequency of items in itemList"
    counters = {}
    for item in itemList:
        if item in counters:
            # increment item counter
            counters[item] += 1
        else:
            # create item counter
            counters[item] = 1
    return counters
```
Exercises

1. Using a dictionary, write a function `convertGrade`, which takes a letter grade (possibly with ‘+’ or ‘-’ after it) and returns the corresponding numeric value for the grade.

2. Write a function `wordCount`. It takes the name of a file as a parameter, and prints a list of the words in the file and their frequencies. In the output, line up the words and their frequencies in two separate columns. Be sure to remove all punctuation and ignore case.

3. Write a function `mostFrequentWord`. It returns the most frequently used word in a file.
Built-in class tuple

The class `tuple` is the same as class list ... except that it is immutable

```python
>>> lst = ['one', 'two', 3]
>>> lst[2]
3
>>> lst[2] = 'three'
>>> lst
['one', 'two', 'three']

>>> tpl = ('one', 'two', 3)
>>> tpl
('one', 'two', 3)
>>> tpl[2]
3
>>> tpl[2] = 'three'
Traceback (most recent call last):
  File "<pyshell#131>", line 1, in <module>
    tpl[2] = 'three'
TypeError: 'tuple' object does not support item assignment
```

Why do we need it? Sometimes, we need to have an “immutable list”.

- For example, when we need a dictionary that has lists as keys
Exercise

Implement function `lookup()` that implements a phone book lookup application. Your function takes, as input, a dictionary representing a phone book, mapping tuples (containing the first and last name) to strings (containing phone numbers).

```python
def lookup(phonebook):
    """implements interactive phone book service using the input phonebook dictionary""
    while True:
        first = input('Enter the first name: ')
        last = input('Enter the last name: ')
        person = (first, last)  # construct the key

        if person in phonebook:  # if key is in dictionary
            print(phonebook[person])  # print value
        else:  # if key not in dictionary
            print('The name you entered is not known.')

>>> phonebook = {
    ('Anna', 'Karenina'): '(123)456-78-90',
    ('Yu', 'Tsun'): '(901)234-56-78',
    ('Hans', 'Castorp'): '(321)908-76-54'}

>>> lookup(phonebook)
Enter the first name: Anna
Enter the last name: Karenina
(123)456-78-90
Enter the first name:
```
The built in class set represents a mathematical set
  • an unordered collection of non-identical items
  • supports operations such as set membership, set union, set intersection, set difference, etc

Example application: remove duplicates from a list
  • WARNING: the order of the items in the list changes
## Set Operators

<table>
<thead>
<tr>
<th>Operation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s == t</code></td>
<td>True if sets <code>s</code> and <code>t</code> contain the same elements, False otherwise</td>
</tr>
<tr>
<td><code>s != t</code></td>
<td>True if sets <code>s</code> and <code>t</code> do not contain the same elements, False otherwise</td>
</tr>
<tr>
<td><code>s &lt;= t</code></td>
<td>True if every element of set <code>s</code> is in set <code>t</code>, False otherwise</td>
</tr>
<tr>
<td><code>s &lt; t</code></td>
<td>True if <code>s &lt;= t</code> and <code>s != t</code></td>
</tr>
<tr>
<td>`s</td>
<td>t`</td>
</tr>
<tr>
<td><code>s &amp; t</code></td>
<td>Returns the intersection of sets <code>s</code> and <code>t</code></td>
</tr>
<tr>
<td><code>s - t</code></td>
<td>Returns the difference between sets <code>s</code> and <code>t</code></td>
</tr>
<tr>
<td><code>s ^ t</code></td>
<td>Returns the symmetric difference of sets <code>s</code> and <code>t</code></td>
</tr>
</tbody>
</table>

```python
>>> ages = {28, 25, 22}
>>> ages2 = {25, 22, 23}
>>> 28 in ages
True
>>> len(ages2)
3
>>> ages == ages2
False
>>> {22, 25} < ages2
True
>>> ages <= ages2
False
>>> ages | ages2
{22, 23, 25, 28}
>>> ages & ages2
{25, 22}
>>> ages - ages2
{28}
>>> ages ^ ages2
{28, 23}
```
set methods

```
>>> ages
{28, 25, 22}
>>> ages2
{25, 22, 23}
>> ages.add(30)
>>> ages
{25, 28, 30, 22}
>>> ages.remove(25)
>>> ages
{28, 30, 22}
>>> ages.clear()
>>> ages
set()
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.add(item)</code></td>
<td>add item to set <code>s</code></td>
</tr>
<tr>
<td><code>s.remove(item)</code></td>
<td>remove item from set <code>s</code></td>
</tr>
<tr>
<td><code>s.clear()</code></td>
<td>removes all elements from <code>s</code></td>
</tr>
</tbody>
</table>

Note that sets are **mutable**
Exercise

1. Write a function `wordsInFile` which is passed the name of a file as a parameter and returns a set containing the words that appear in a file.

2. Write a function `wordsInCommon` which is passed the names of two files as parameters and returns a set containing the words that appear in both files.