More Built-in Container Classes

- Container Class `dict`
- Container Class `set`

User-defined indexes and dictionaries

Goal: a container of employee records indexed by employee SS#

```
>>> employee[987654321]
['Yu', 'Tsun']
>>> employee[864209753]
['Anna', 'Karenina']
>>> employee[100010010]
['Hans', 'Castorp']
```

Problems:

- the range of SS#s is huge
- SS#s are not really integers

Solution: the dictionary class `dict`

key	value
'864-20-9753'	['Anna', 'Karenina']
'987-65-4321'	['Yu', 'Tsun']
'100-01-0010'	['Hans', 'Castorp']

A dictionary contains (key, value) pairs

```
>>> employee = {
        '864-20-9753': ['Anna',
'Karenina'],
        '987-65-4321': ['Yu', 'Tsun'],
        '100-01-0010': ['Hans', 'Castorp']}
>>> employee['987-65-4321']
['Yu', 'Tsun']
>>> employee['864-20-9753']
['Anna', 'Karenina']
```

A key can be used as an index to access the corresponding value

Properties of dictionaries

Dictionaries are not ordered

```
>>> employee = {
        '864-20-9753': ['Anna',
'Karenina'],
        '987-65-4321': ['Yu', 'Tsun'],
        '100-01-0010': ['Hans', 'Castorp']}
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-
20-9753': ['Anna', 'Karenina'], '987-65-
4321': ['Yu', 'Tsun']}
>>>
```


Properties of dictionaries

Dictionaries are not ordered

Dictionaries are mutable

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

```
>>> employee = {
        '864-20-9753': ['Anna',
'Karenina'],
        '987-65-4321': ['Yu', 'Tsun'],
        '100-01-0010': ['Hans', 'Castorp']}
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-
20-9753': ['Anna', 'Karenina'], '987-65-
4321': ['Yu', 'Tsun']}
>>> employee['123-45-6789'] = 'Holden
Cafield'
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-
20-9753': ['Anna', 'Karenina'], '987-65-
4321': ['Yu', 'Tsun'], '123-45-6789':
'Holden Cafield'}
>>> employee['123-45-6789'] = 'Holden
Caulfield'
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-
20-9753': ['Anna', 'Karenina'], '987-65-
4321': ['Yu', 'Tsun'], '123-45-6789':
'Holden Caulfield'}
```

Properties of dictionaries

Dictionaries are not ordered

Dictionaries are mutable

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

```
>>> employee = {[1,2]:1, [2,3]:3}
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    employee = {[1,2]:1, [2,3]:3}
TypeError: unhashable type: 'list'
```

The empty dictionary is `{}`

Dictionary keys must be immutable

Dictionary operators

Class `dict` supports some of the same operators as class `list`

```
>>> days = {'Mo':1, 'Tu':2, 'W':3}
>>> days['Mo']
1
>>> days['Th'] = 5
>>> days
{'Mo': 1, 'Tu': 2, 'Th': 5, 'W': 3}
>>> days['Th'] = 4
>>> days
{'Mo': 1, 'Tu': 2, 'Th': 4, 'W': 3}
>>> 'Fr' in days
False
>>> len(days)
4
```

Class dict does not support all the operators that class list supports

- + and * for example

Dictionary methods

Operation	Explanation
`d.items()`	Returns a view of the (key, value) pairs in `d`
`d.keys()`	Returns a view of the keys of `d`
`d.pop(key)`	Removes the (key, value) pair with key `key` from `d` and returns the value
`d.update(d2)`	Adds the (key, value) pairs of dictionary `d2` to `d` (overwrites)
`d.values()`	Returns a view of the values of `d`

```
>>> 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()
dict_items([('Fr', 5), ('W', 3), ('Th',
4), ('Mo', 1), ('Tu', 2)])
>>> days.keys()
dict_keys(['Fr', 'W', 'Th', 'Mo', 'Tu'])
>>> >>> vals = days.values()
>>> vals
dict_values([5, 3, 4, 1, 2])
>>> for val in vals:
        print(val, end=' ')


5 3 4 1 2
>>>
```


Dictionary methods

The containers returned by `d.items()`, `d.keys()`, and `d.values()` (called views) can be iterated over

```
>>> days = {'Mo':1, 'Tu':2, 'W':3}
>>> type(days.keys())
<class 'dict_keys'>
>>> type(days.items())
<class 'dict_items'>
>>> type(days.values())
<class 'dict_values'>
>>>
```

Dictionary vs. multi-way `if` statement

Uses of a dictionary:

- container with custom indexes
- alternative to the multi-way `if` statement

```
def complete(abbreviation):
    'returns day of the week corresponding to abbreviation'

    if abbreviation == 'Mo':
        return 'Monday'
    elif abbreviation == 'Tu':
        return 'Tuesday'
    elif
        ......
    else: # abbreviation must be Su
        return 'Sunday'
```

```
def complete(abbreviation):
    'returns day of the week corresponding to abbreviation'

    days = {'Mo': 'Monday', 'Tu':'Tuesday', 'We': 'Wednesday',
            'Th': 'Thursday', 'Fr': 'Friday', 'Sa': 'Saturday',
            'Su':'Sunday'}

    return days[abbreviation]
```


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

```
>>> 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, 95, 90, 95, 100, 100]
              ^   ^    ^    ^   ^   ^   ^
```

counters

95	96	100	85	90
3	1	1	1	1

```
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
```


Exercise

Implement function `wordcount()` that takes as input a text—as a string— and prints the frequency of each word in the text; assume there is no punctuation in the text.

```
>>> text = 'all animals are equal but some animals are more equal than other'
>>> wordCount(text)
all      appears 1 time.
animals  appears 2 times.
some     appears 1 time.
equal    appears 2 times.
but      appears 1 time.
other    appears 1 time.
are      appears 2 times.
than     appears 1 time.
more     appears 1 time.
>>>
```

Exercise

Implement function `wordcount()` that takes as input a text—as a string— and prints the frequency of each word in the text; assume there is no punctuation in the text.

```
def wordCount(text):
    'prints frequency of each word in text'

    wordList = text.split()  # split text into list of words

    counters ={}             # dictionary of counters
    for word in wordList:
        if word in counters: # counter for word exists
            counters[word] += 1
        else:                # counter for word doesn't exist
            counters[word] = 1

    for word in counters:    # print word counts
        if counters[word] == 1:
            print('{:8} appears {} time.'.format(word, counters[word]))
        else:
            print('{:8} appears {} times.'.format(word, counters[word]))
```


Exercise

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

```
>>> 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:
```

Exercise

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

```
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.')
```


Sets

- It is used to store an unordered collection of items, with no duplicate items allowed.
- The items must be immutable objects.
- The set type supports operators that implement the classical set operations:
  - set membership
  - Intersection
  - Union
  - symmetric difference
  - And many others.
- It is also useful for duplicate removal.

Sets definition

- Defined using { } brackets.
- Duplicate items are ignored
- Empty set can not be declared using { }

```
>>> phonebook1 = {'123-45-67', '234-56-78', '345-67-89'}

>>> phonebook1
{'123-45-67', '234-56-78', '345-67-89'}

>>> type(phonebook1)
<class 'set'>

>>> phonebook1 = {'123-45-67', '234-56-78', '345-67-89',
'123-45-67', '345-67-89'}

>>> phonebook1
{'123-45-67', '234-56-78', '345-67-89'}

>>> set1 = {}
>>> type(set1)
<class 'dict'>

>>> set1 = set()
>>> type(set1)
<class 'set'>
```


Sets for removing duplicates

Uses of a sets:

- container with unordered collection of unique items
- Removing duplicates from a list

Problem: removing duplicate entries from a list of ages of students in a class

```
>>> ages = [23, 19, 18, 21, 18, 20, 21, 23, 22, 23, 19, 20]
>>> ages = list(set(ages))
>>> ages
[18, 19, 20, 21, 22, 23]
```

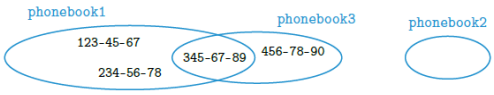
Solution: Convert the list into a set, removing any duplicate entries

There is, however, *one major caveat*: The elements have been reordered.

Set operators

The set class supports operators that correspond to the usual mathematical set operations. Some operators are the ones which are also used for lists, strings, and dictionaries

Operations supported by other containers:
in, not in, len, comparison operators



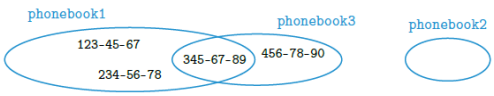

```
>>> '123-45-67' in phonebook1
True

>>> '456-78-90' in phonebook1
False

>>> '456-78-90' not in phonebook1
True

>>> len(phonebook1)
3
>>> phonebook3 = {'345-67-89','456-78-90'}
>>> phonebook1 == phonebook3
False

>>> phonebook1 != phonebook3
True
```

A set is “less than or equal to” another set if it is a subset of it, and a set is “less than another set” if it is a proper subset of it.

Set operators

The set class supports operators that correspond to the usual mathematical set operations. Some operators are the ones which are also used for lists, strings, and dictionaries

Operations supported by other containers:
in, not in, len, comparison operators




```
>>> {'123-45-67', '345-67-89'} <= phonebook1
True
```

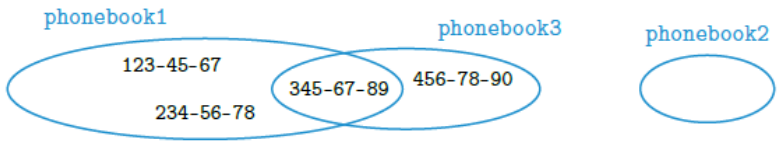
A set is “less than or equal to” another set if it is a subset of it, and a set is “less than another set” if it is a proper subset of it.

Mathematical Set operators

The mathematical set operations union, intersection, difference, and symmetric diffe
are implemented as set operators |, &, -, and ^, respectively

Each set operation takes
two sets and returns a new set.

```
>>> phonebook1 | phonebook3
{'123-45-67', '234-56-78', '345-67-89', '456-78-90'}

>>> phonebook1 & phonebook3
{'345-67-89'}

>>> phonebook1 - phonebook3
{'123-45-67', '234-56-78'}

>>> phonebook1 ^ phonebook3
{'123-45-67', '234-56-78', '456-78-90'}
```



Mathematical Set operators

The mathematical set operations union, intersection, difference, and symmetric diffe
are implemented as set operators |, &, -, and ^, respectively

Each set operation takes
two sets and returns a new set.

Operation	Explanation
`x in s`	True if x is in set s, else False
`x not in s`	False if x is in set s, else True
`len(s)`	Returns the size of set s
`s == t`	True if sets s and t contain the same elements, False otherwise
`s != t`	True if sets s and t do not contain the same elements, False otherwise
`s <= t`	True if every element of set s is in set t, False otherwise
`s < t`	True if `s <= t` and `s != t`
`s | t`	Returns the union of sets s and t
`s & t`	Returns the intersection of sets s and t
`s - t`	Returns the difference between sets s and t
`s ^ t`	Returns the symmetric difference of sets s and t

Set methods

Operation	Explanation
`S.add()`	Adds an item to a set
`S.remove()`	Removes an item from the set
`S.clear()`	Empty a set

```
>>> phonebook3.add('123-45-67')
>>> phonebook3
{'123-45-67', '345-67-89', '456-78-90'}

>>> phonebook3.remove('123-45-67')
>>> phonebook3
{'345-67-89', '456-78-90'}

>>> phonebook3.clear()
>>> phonebook3
set()
```