Dictionaries: Keeping track of pairs

- Class dict
- Class tuple

The dictionary: An associative data type

- Many things in the world come in pairs, where Thing1 is associated with Thing2. For example
  - A word and its definition
  - A person and the person's birthday
  - A living thing and its species
- The Python class dictionary models such relationships flexibly and efficiently

```
>>> definitions = {'modest':'unassuming', 'eat':'chew and swallow'}
>>> bdays = {'Napoleon':'15 Aug 1769', 'Lady Gaga':'Mar 28 1986'}
>>> species = {'Fido':'dog', 'Finny':'catfish', 'Pepe Le Pew':'skunk'}
```

Dictionary syntax

- The syntax of a dictionary:
  - A dictionary is enclosed in curly brackets. The empty dictionary is denoted {}.
  - A dictionary is a collection of key:value pairs. Each key is separated from its value by a colon (':').
  - Key:value pairs are separated by commas.
- Permitted types of keys and values:
  - A key can be any immutable type – for example, a string, a number or a tuple. A key cannot be a mutable type, such as a list or a dictionary.
    - A dictionary may contain keys of different types, but …
    - Each key in a dictionary must be unique
  - A value can be any type we have learned – a string, a number, a list – even another dictionary.

Dictionary: Basic operations

```
>>> # create an empty dictionary
>>> d = {}

>>> # add a key:value pair, using index and assignment operators
>>> d['CC'] = 'computing'

>>> # look up the value of a key using the index ([]) operator
>>> d['CC']
'computing'

>>> # assigning a new value
>>> d['CC'] = 'College of Computing'
>>> d['CC']
'College of Computing'

>>> # remove a key:value pair and return its value
>>> whatIsCC = d.pop('CC')
>>> whatIsCC
'College of Computing'
>>> d
{}
```

Dictionary: More things you can do

```
>>> # find the size (number of key:value pairs) of a dictionary
>>> d = {1:'for the money', 'two':'for the show'}
>>> len(d)
2

>>> # test for the membership (presence) of a key
>>> 1 in d
True
>>> 2 in d
False

>>> # incorporate dictionary d2 into dictionary d
>>> d2 = {'Twain':'Mark'}
>>> d.update(d2)
>>> d
{1: 'for the money', 'two': 'for the show', 'Twain': 'Mark'}

>>> # note that dictionary d2 is not changed
>>> d2
{'Twain':'Mark'}
```

Iterating over a dictionary

- Iterate over dictionary keys like this

```
>>> # Use the dictionary method d.keys()
>>> d = {'tree':'oak', 'fish':'guppy', 'dog':'boxer', 'cat':'tabby'}
>>> for thing in d.keys():
        if 't' in thing:
                print(thing, end=' ')

tree cat
```

Note: for thing in d: is the same as for thing in d.keys()

- Iterate over dictionary values like this

Use the dictionary method d.values()

```
>>> for value in d.values():
        if 'y' in value:
                print(value, end=' ')

guppy tabby
```

Dictionary properties: Order and speed

Over the years, many great ideas have combined together to produce the modern implementation of Python dictionaries.

- Prior to Python 3.7 dictionary order wasn't guaranteed
- As of Python 3.7 dictionary order is guaranteed to be insertion order
- Dictionary lookup (amazingly!) does not get slower as the dictionary gets bigger

Inserting ten million elements may take a few seconds

```
>>> d = {}
>>> for i in range(10000000):
        d[i] = str(i)
```

Retrieving one element from among 10,000,000 is immediate

```
>>> d[5000000]
'5000000'
```

The tuple as a dictionary key

- A tuple is similar to a list, but it is immutable. A tuple is delimited by parentheses where a list has square brackets

This is a list of int's

```
>>> intList = [1, 2, 3]
```

Attempting to use a list as a key will result in an error

```
>>> d = {intList: "some int's"}
Traceback (most recent call last):
  File "<pyshell#193>", line 1, in <module>
    d = {intList:"some int's"}
TypeError: unhashable type: 'list'
```

Solution: Use a tuple as a key instead

```
>>> intTuple = (1, 2, 3) # or intTuple = tuple(intList)
>>> d = {intTuple: "some int's"}
>>> d
{(1, 2, 3): "some int's"}
```

Example: Keeping count(s)

Problem: you want to count how many times each item in a list occurs, but you don't know the items in advance. For example, for this list

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

a table representation looks like this

100	3
96	1
95	3
90	1
85	1

Counter variables don't work well for this problem

- until you see the list, you don't know what counters to create
- you'd like to be able to access the count of an item via the item itself

Solution: create a dictionary in which each distinct item is a key and its count is the associated value

A dictionary as a collection of counters

```
def frequency(itemList):
    '''Compute the frequency of each item in
    itemList. Return the result as a dictionary
    in which each distinct item in itemList is a
    key and the item's frequency (count) is the
    associated value.'''

    counters = {}

    for item in itemList:
        # if the item hasn't been seen before
        if item not in counters:
            # add the item to the dictionary
            # and set its count to 1
            counters[item] = 1
        # otherwise, increment the count of item
        else:
            counters[item] += 1

    return counters
```

A dictionary as a collection of lists

```
wordOccurrences = [('rabbbit',1),('Alice',1),
('rabbbit',4),('Alice',7),('Alice',10)]

def makeIndex(wordPageList):
    '''Given a list of word/page tuples, compute and
    return an index of the pages on which words occur.'''
    wordIndex = {}
    for item in wordPageList:
        word = item[0]; page = item[1]
        # if the word hasn't been seen before
        if word not in wordIndex:
            # add the word to the dictionary
            wordIndex[word] = [page]
        # otherwise, add the page
        else:
            wordIndex[word].append(page)
    return wordIndex

aliceIndex = makeIndex(wordOccurrences)
# Returns {'rabbbit': [1, 4], 'Alice': [1, 7, 10]}
```