MODULE 11
Compute unique words and their frequency

In the previous two modules, we count the total number of occurrences of vowels and characters. Now we will apply the same thing for words. i.e., we will find the frequency (number of occurrences) of the words in the given sentence.

Problem

Given a sentence compute the unique words and their frequency.

Attacking the Problem

In the previous two modules, we take an array with size of 10 for vowels and 52 for characters and we increase the count of corresponding indexes. But now we need to find the frequency of words. So we will find the index for the word by adding ascii values of all the characters in that word. Let's look at a small example how it works.

What is ASCII Value

***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange*. Pronounced *ask-ee*, ASCII is a code for representing English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase '*M*' is 77. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

Character to ASCII value:

```
>>> ord('a')
97
```

ASCII value to character:

```
>>> chr(97)
a
```

Finding index for word

```
>>> word = "dog"
>>> index = ord('d') + ord('o') + ord('g')
>>> print "index for dog is:", index
```

Output: this will produce:

Index for dog is: 314

So we can save the frequency of "dog" in index of 314. Let's find the index for "god"

```
>>> word = "god"
>>> index = ord('g') + ord('o') + ord('d')
>>> print "index for dog is:", index
```

Output: this will produce:

Index for god is: 314

So index for god is also 314. What should we do now? How can we store the frequencies for "dog" and "god"? In the same way, we may also face problems for other words like

- cat, act
- bare, bear
- bat, tab

Python has an excellent solution for this. We have dictionaries in python which will solve the above problem.

Dictionaries

Dictionary is a data structure in which you can refer to each value by name. This type of structure is called a mapping or hash, and the only built-in mapping type in Python is the dictionary. The values in a dictionary don't have any particular order but are stored under a key, which may be a number, a string.

The name "dictionary" should give you a clue: an ordinary book is made for reading from start to finish. If you like, you can quickly open it to any given page. Dictionaries, however (both real ones and their Python equivalent) are constructed so that you can look up a specific word (key) easily, to find its definition (value).

In a dictionary, you have an 'index' of words and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - they aren't in any specific order, either - the key does the same thing. You can add, remove, and modify the values in dictionaries.

Example: telephone book.

Some Important Points

- ✓ Hashes or dictionaries are (key, value) pairs
- ✓ Appropriate for quick lookup/search operations
- ✓ Keys of a dictionary are immutable
- ✓ Duplicate keys are not possible in dictionaries
- ✓ Assigning value to existing key will wipe out the old value

Dictionary Syntax

Dictionaries are written like this:

```
Phonebook = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

Dictionaries consist of pairs (called items) of keys and their corresponding values. In the preceding example, the names are the keys and the telephone numbers are the values. Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary (without any items) is written with just two curly braces, like this: {}.

Methods in Dictionaries

Just like the other built-in types, dictionaries have methods. While these methods can be very useful, you probably will not need them as often as the list and string methods.

clear

The clear method removes all items from the dictionary. It returns nothing (or, rather, None)

```
IDLE 1.2.1
>>> d = {}
>>> d['name'] = 'Gumby'
>>> d['age'] = 42
>>> d
{'age': 42, 'name': 'Gumby'}
>>> returned_value = d.clear()
>>> d
{}
>>> print returned_value
None
>>>
```

get

The get method is a forgiving way of accessing dictionary items. Ordinarily, when you try to access an item that is not present in the dictionary, things go very wrong:

```
IDLE 1.2.1
>>> d = {}
>>> print d['name']

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print d['name']
KeyError: 'name'
>>> print d.get('name')
None
>>>
```

As you can see, when you use get to access a nonexistent key, there is no exception. Instead, you get the value None. You may supply your own “default” value, which is then used instead of None:

```
>>> d.get('name', 'N/A')
'N/A'
```

If the key is there, get works like ordinary dictionary lookup:

```
>>> d['name'] = 'Eric'
>>> d.get('name')
'Eric'
>>>
```

has_key

The has_key method checks whether a dictionary has a given key. The expression d.has_key(k) is equivalent to k in d. The choice of which to use is largely a matter of taste.

Here is an example of how you might use has_key:

```
>>> d = {}
>>> d.has_key('name')
False
>>> d['name'] = 'Eric'
>>> d.has_key('name')
True
>>>
```

items

The items method returns all the items of the dictionary as a list of items in which each item is of the form (key, value). The items are not returned in any particular order:

```
>>> d = {'title': 'Python Web Site', 'url': 'http://www.python.org', 'spam': 0}
>>> d.items()
[('url', 'http://www.python.org'), ('spam', 0), ('title', 'Python Web Site')]
>>>
```

keys

The keys method returns a list of the keys in the dictionary.

Examples of using Dictionaries

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
```

```
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Looping Techniques in Dictionaries

When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the **`iteritems()`** method.

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.iteritems():
...     print k, v
...
gallahad the pure
robin the brave
```

When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the **`enumerate()`** function.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print i, v
...
0 tic
1 tac
2 toe
```

To loop over a sequence in reverse, first specify the sequence in a forward direction and then call the **`reversed()`** function.

```
>>> for i in reversed(xrange(1,10,2)):
...     print i
...
9
7
5
3
1
```

Steps to solve the above problem

1. First Read the given Sentence
2. Create an empty dictionary
3. Read word by word (which you already learnt in the previous modules) from the given sentence
    a. If that word is not there in Dictionary,
        i. Put that word as a key in dictionary and set value as 1
    b. If that word is already there in Dictionary,
        i. Increase the value of that word by 1
4. Repeat the above step until you finish the given sentence
5. Now print all keys and corresponding values of the dictionary