IIT Madras
BSc Degree

Home Lesson-6.4

Lesson-6.3

Dictionaries

Pangrams and Dictionaries

Assume that we wish to compute the following mapping between letters of the English alphabet and numbers from 1 to 26:

Letter	Number
a	1
b	2
...	...
z	26

Each letter in the alphabet is mapped to a unique number from 1 to 26. In the table given above, the mapping is a simple linear mapping: `a` is mapped to `1`, `b` to 2 and so on. This mapping can be computed in the most uninteresting and lousy way given below:

```
mapping = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5,
           'f': 6 , 'g': 7, 'h': 8, 'i': 9, 'j': 10,
           'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15,
           'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20,
           'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25,
           'z': 26
          }

for letter, count in mapping.items():
    print(letter, count)
```

Phew! I typed the whole thing out. It took me two dull minutes and I learned nothing new at the end of the exercise. My fingers cursed me for the mechanical key-presses while my eyes chided me for staring at the screen without blinking. Besides, the last letter of the alphabet was quite annoyed at being left alone in the last row with no company, while every other letter got to share line-space with four other letters!

Let us try a round about but interesting way. Consider the following line:

> the quick brown fox jumps over the lazy dog

This sentence is called a pangram. A pangram is a sentence that uses all the letters of the alphabet. Does that ring any bell?

```
pangram = 'the quick brown fox jumps over the lazy dog'
words = pangram.split(' ')          # get list of words in the sentence
letters = ''.join(words)            # join the words back; eliminates spaces
sorted_letters = sorted(letters)    # sort letters
mapping, count = dict(), 0
for letter in sorted_letters:
    # check if letter is not present in dict
    # to avoid counting same letter multiple times
    if letter not in mapping:
        count += 1
        mapping[letter] = count     # map the letter to count

for letter, count in mapping.items():
    print(letter, count)
```

Plenty of things to learn from those 14 lines of code. Not all diversions are bad. Now that we have an interesting dictionary in place, let us jump into some methods that are bundled along with `dict`.

Dictionary Methods

We have already seen `keys` and `items`. Both these are methods that return a view object over which we can iterate. According to the Python documentation, "a view object provides a dynamic view on the dictionary's entries, which means that when the dictionary changes, the view reflects these changes."

```
keys = mapping.keys()
print(keys)
```

This gives the following output:

```
dict_keys(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'])
```

Using the `list` function, both the `keys` and `items` views can be converted into lists:

```
keys_list = list(mapping.keys())
print(keys)
items_list = list(mapping.items())
print(items)
```

The output is as follows:

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
[('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5), ('f', 6), ('g', 7), ('h',
8), ('i', 9), ('j', 10), ('k', 11), ('l', 12), ('m', 13), ('n', 14), ('o',
15), ('p', 16), ('q', 17), ('r', 18), ('s', 19), ('t', 20), ('u', 21), ('v',
22), ('w', 23), ('x', 24), ('y', 25), ('z', 26)]
```

`keys_list` is a list of keys in the dictionary `mapping`. `items_list` is a list of tuples, where each tuple is a key-value pair. Another useful method is `values`. This returns a view on the values:

```
view = mapping.values()
view_list = list(view)
```

All three views - `keys, items, values` - support membership tests:

```
print('a' in mapping.keys())
print(1 in mapping.values())
print(('a', 1) in mapping.items())
```

All three return `True`. Membership tests for keys can be done in a simpler way:

```
print('a' in mapping)
print('x' in mapping)
print('ab' not in mapping)
```

Note that we dropped the `keys` method and it still worked! Now, to delete a key from a dictionary, we use the familiar `pop` method:

```
mapping['ab'] = 3           # some noise added to mapping
value = mapping.pop('ab')
print(value)
print('ab' not in mapping)
```

If `key` is a key in a dictionary `D`, `D.pop(key)` removes the key `key` in `D` and returns the value associated with it. Removing a key naturally removes the value associated with it. Dictionaries are aristocratic data structures: keys are higher up in the hierarchy and values depend on the keys for their existence.