dictionaries

February 10, 2015

Contents

Dictionaries

0.0.1 Data types: Story so far

- Basic types
  - Integers int:
    . . . , -2, -1, 0, 1, 2, 3,. . .
  - Floating point numbers float:
    0.05, 3.1415, 2.0, 62.8318, . . .
  - Strings str:
    “hello”, ’John Doe’, . . .
- Compound data types (data that contains other data):
  - Lists list:
    [1, 2, 3], [’alice’, ’bob’, 7]
  - Dictionaries dict:
  - others (np.array, tuple, set)

0.0.2 Problems with lists

Suppose we store and retrieve age of Bob and Alice:

```
In [ ]: ages = [['bob', 23], ['alice',25]]
        print(ages[0])
        print(ages[0][1])
```

Have to remember Bob is index 0 and Alice is index 1

- Not going to work when there are lots of people.
- Would be much easier to not need to remember.

Instead use dictionaries

```
In [ ]: ages = {'bob': 23, 'alice': 25}
        print(ages['bob'])
        print(ages['alice'])
```

What are dictionaries?

- Values in a dictionary can be found using a key.
  - Values in a dictionary can be anything.
  - Keys can be anything that is immutable.
    - e.g., strings, integers, floats, tuples, sets.
- A dictionary uses curly braces: {}

```
In [ ]: empty_dict = {}
        dict1 = {100: 'hundred', 1: 'one', 10: 'ten'}
        dict2 = {'bob':['england', -10],
                 'alice':['england', -5],
                 'mallory': ['usa',100]}
```

Dictionaries may be visualised as a mapping from a key to a value.

```
In [ ]: dict = {'a': 'alpha', 'g': 'gamma', 'o':
        'omega'}
```

0.0.3 Things dictionaries can do

- Dictionaries are mutable. You can change them.
- Keys find associated values quickly.
- Associations between keys and values can be added, deleted and changed.
- Dictionaries are often useful in Python code:
  - Efficiently associate a key with a value.

0.0.4 Accessing values

```
In [ ]: print(dict1[100])
        print(dict2['bob'])
        print(dict2['mallory'][1])
```

0.0.5 Length and missing items

The number of items in a dictionary can be determined using the len() function

```
In [ ]: len(dict1)
```

Trying to retrieve an item using a key that is not in the dictionary throws an KeyError:

```
In [ ]: print(dict1)
        dict1[1000] # 1000 is not a key
```

0.0.6 Keys and values

A list of keys in a dictionary can be found using the .keys() member function:

```
In [ ]: print dict1.keys()
```

Likewise, a list of values in a dictionary can be found using the .values() member function:

```
In [ ]: print dict1.values()
```

You can also get both together, using the .items() member function:

```
In [ ]: print dict1.items() # returns a list of tuples
        print dict2.items() # returns a list of immutable lists
```

0.0.7 Testing for keys

One can efficiently test if a key is in a dictionary: use the in operator.

```
In [ ]: dict1
        print 100 in dict1
        print 1000 in dict1
        if 100 in dict1:
            print 'the dictionary contains 100'
```

0.0.8 Adding items

```
In [ ]: dict1[1000] = 'thousand' # adding item to dict1
        print 1000 in dict1 # testing for key

        dict2['ezekiel'] = ['france', 50] # adding item to dict2
        print dict2['ezekiel'][0]
        print dict2
```

You can add items by using a key that has not been used yet.

0.0.9 Modifying items

Items can be modified from a dictionary just like in a list:

```
In [ ]: dict2['bob'] = ['england', -5]
        print dict2['bob']
        dict2['bob'][1] = -15
        print dict2['bob']
```

0.0.10 Removing items

Similarly, items can be removed from a dictionary using the del operator:

```
In [ ]: 'mallory' in dict2
        del dict2['mallory']
        'mallory' in dict2
        dict2['mallory']
```

0.0.11 Iterating over items

Two common ways of iterating over the items of a dictionary:

```
In [ ]: for key in dict1:
            print key, dict1[key]

        for key, value in dict1.items():
            print key, value
```

Note that the order in which items are visited in a dictionary is arbitary!

```
In [ ]: for key, value in dict2.items():
            print key, value
```

Dictionaries are **not** ordered. List are ordered.

0.0.12 Computational efficiency

- Memory used by dictionaries is proportional to the number of items in a dictionary.
- Dictionaries in python are a data structure known as hash tables.
- Inserting a new item and looking up items in a dictionary by the key is O(1) on average.
- Removing any item from a dictionary is close to O(1) in practice.
- But:
  - The order of data items is not fixed.
  - Only immutable objects may be used as keys.
  - Any object may be stored as a value in a dictionary under a key.

0.0.13 Exercices

To come...