DICTIONARIES

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

A dictionary holds **key-value** pairs, which are referred to as items. You will hear dictionaries referred to in different ways including: associative arrays, hashes, or hash tables.

Dictionaries are generated using comma separated items surrounded by curly braces. The item begins with a key, followed by a colon, and concluded with a value.

The pattern is **dictionary_name = {key_1: value_1, key_N: value_N}**.
In order to create an empty dictionary, use **dictionary_name = {}**.

Creating a Dictionary

Creating a dictionary is as simple as placing items inside curly braces {} separated by comma. An item has a key and the corresponding value expressed as a pair, key: value.

While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

```
# creating an empty dictionary
>>> my_dict1 = {}
>>> my_dict1
{}
>>> type(my_dict1)
<type 'dict'>

# dict with integer keys
>>> my_dict2 = {1: 'apple', 2: 'ball'}
>>> my_dict2
{1: 'apple', 2: 'ball'}

#dict with mixed keys
>>> my_dict3 = {'name': 'Vamshi', 1: [2, 4, 3]}
>>> my_dict3
{1: [2, 4, 3], 'name': 'Vamshi'}
```

We can also create a dictionary using the built-in function dict().

```
>>> my_dict4 = dict({1:'Apple', 2:'Mango'})
>>> my_dict4
{1: 'Apple', 2: 'Mango'}

#creating a dict with each item as a pair
>>> my_dict5 = dict([(1, 'Apple'), (2, 'Ball'), (3, 'Cat')])
>>> my_dict5
```

{1: 'Apple', 2: 'Ball', 3: 'Cat'}

Accessing elements of a dictionary

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method.

```
>>> print(my_dict6['name'])
John

>>> print(my_dict6.get('name'))
John
>>> print(my_dict6.get('age'))
30
```

Not only are you able to access values by key, you can also set values by key. The pattern is dictionary_name[key] = value.

```
>>> my_dict6['age'] = 32
>>> my_dict6
{'age': 32, 'name': 'John'}
```

Adding items to a dictionary

Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

Example1:

```
>>> my_dict6['address'] = 'New Jersey'  #adding address key to the existing dict
>>> my_dict6
{'age': 32, 'name': 'John', 'address': 'New Jersey'}
```

Example2:

```
>>> Contacts = {'David': '555-0123', 'Tom': '555-5678'}
>>> Contacts['Nora'] = '555-2413'   # adding Nora to the Contacts dictionary
>>> print(Contacts)
{'Nora': '555-2413', 'Tom': '555-5678', 'David': '555-0123'}
```

Removing items from a dictionary

We can remove a particular item in a dictionary by using the method pop(). This method removes as item with the provided key and returns the value.

The method, popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary. All the items can be removed at once using the clear() method.

We can also use the del keyword to remove individual items or the entire dictionary itself.

```
>>> squares = {1:1, 2:4, 3:9, 4:16, 5:25}
>>> print(squares.pop(4))  #remove a particular item
16
>>> del squares[3]  #remove a particular item
>>> squares
{1: 1, 2: 4, 5: 25}
>>> print(squares.popitem())  #remove arbitrary item
(1, 1)
>>> squares.clear()   #remove all items
>>> print(squares)
{}
```

Dictionaries can contain different datatypes

Keep in mind that the values within a dictionary do not have to be of the same data type. In the following example you'll see that while the value for the **David** key is a list, the value for the **Tom** key is a string.

```
>>> contacts = { 'David': ['555-0123', '555-0000'],
              'Tom': '555-5678'
            }
>>> print('David contacts {}'.format(contacts['David']))
David contacts ['555-0123', '555-0000']
>>> print('Tom contacts {}'.format(contacts['Tom']))
Tom contacts 555-5678
```

Finding a Key in a Dictionary:

If you would like to find out whether a certain key exists within a dictionary, use the **value in dictionary_name.keys()** syntax. If the value is in fact a key in the dictionary, **True** will be returned. If it is not, then **False** will be returned.

```
>>> if 'David' in contacts.keys():
        print("David's contact num is: {}".format(contacts['David']))


David's contact num is: ['555-0123', '555-0000']

>>> if 'Mahesh_Babu' in contacts.keys():
        print(Mahesh_Babu)


>>>
```

Take note that **'David' in contacts** evaluates to **True**, so the code block which follows the **if** statement will be executed. Since **'Mahesh_Babu' in contacts** evaluates to **False**, the code block which follows that statement will not execute.

Finding a Value in a Dictionary:

Using the **values()** dictionary method returns a list of values within the dictionary. Use the **value in list** syntax to determine if the value actually exists within the list. If the value is in the list, **True** will be returned. Otherwise **False** will be returned.

```
>>> contacts = { 'David': ['555-0123', '555-0000'], 'Tom': '555-5678' }
>>> print('555-5678' in contacts.values())
True
```

The above method is fine for finding a value of a particular key. But what if we want to find the value of a key which is a list, ie. Accessing elements of a list inside a dictionary:

```
>>> if '555-0000' in contacts['David']:
        print('True')
else:
        print('False')


True
```

Looping through a Dictionary:

```
>>> contacts = { 'David': '555-0123', 'Tom': '555-5678' }
>>> for contact in contacts:
        print("The number for {0} is {1}".format(contact, contacts[contact]))


The number for Tom is 555-5678
The number for David is 555-0123
```

Nesting Dictionaries:

Since the values contained in a dictionary can be of any data type you have the ability to nest dictionaries. In the following example, names are the keys for the **contacts** dictionary, while **phone** and **email** are the keys used within the nested dictionary. Each individual in this contact list has both a phone number and an email address.

```
>>> contacts = { 'David': { 'phone': '555-0123', 'email': 'david@gmail.com'},
                 'Tom': { 'phone': '555-5678', 'email': 'tom@gmail.com'}
               }

>>> for contact in contacts:
        print("{}'s contact info:".format(contact))
        print(contacts[contact]['phone'])
        print(contacts[contact]['email'])


Tom's contact info:
555-5678
tom@gmail.com
David's contact info:
555-0123
david@gmail.com
```