Python - Dictionary

Like the list and the tuple, dictionary is also a collection type. However, it is not an ordered sequence, and it contains key-value pairs. One or more key:value pairs separated by commas are put inside curly brackets to form a dictionary object.

Syntax:
dict = { key1:value1, key2:value2,...keyN:valueN }

The following declares a dictionary object.

```
>>> capitals={"USA":"Washington, D.C.", "France":"Paris", "India":"New Delhi"}
```

In the above example, capitals is a dictionary object. The left side of : is a key and right side of : is a value. The key should be an immutable object. A number, string or tuple can be used as key. Hence, the following definitions of dictionary are also valid:

```
>>> numNames={1:"One", 2: "Two", 3:"Three"}
>>> decNames={1.5:"One and Half", 2.5: "Two and Half", 3.5:"Three and Half"}
>>>
items={("Parker","Reynolds","Camlin"):"pen",("LG","Whirlpool","Samsung"): "Refrigerator"}
```

However, a dictionary with a list as a key is not valid, as the list is mutable:

```
>>>dict={["Mango","Banana"]:"Fruit", ["Blue", "Red"]:"Colour"}
TypeError: unhashable type: 'list'
```

But, a list can be used as a value.

```
>>>dict={"Fruit":["Mango","Banana"], "Colour":["Blue", "Red"]}
```

The same key cannot appear more than once in a collection. If the key appears more than once, only the last will be retained. The value can be of any data type. One value can be assigned to more than one key.

```
>>>staff={"Krishna":"Officer", "Steve":"Manager", "John":"officer",
"Anil":"Clerk", "John":"Manager"}
>>>staff
{"Krishna":"Officer", "Steve":"Manager", "Anil":"Clerk",
"John":"Manager"}
```

In above example the same value parameter is used more than once. However, when key 'John' is assigned two different values, only the latest is retained, overwriting the previous value.

Accessing a Dictionary

Dictionary is not an ordered collection, so a value cannot be accessed using an index in square brackets. A value in a dictionary can be fetched using the associated key, using the get() method. Specify the key in the get() method to retrieve its value.

```
>>>capitals={"USA":"New York", "France":"Paris", "Japan":"Tokyo",
"India":"New Delhi"}
>>>capitals.get("France")
'Paris'
>>>points={"p1":(10,10), "p2":(20,20)}
>>>points.get("p2")
(20,20)
>>>numbers={1:"one", 2:"Two", 3:"three",4:"four"}
>>>numbers.get(2)
'Two'
```

Use the for loop to iterate a dictionary in the Python script.

```
capitals={"USA":"Washington, D.C.", "France":"Paris", "Japan":"Tokyo", "India":"New Delhi"}

for key in capitals:
   print("Key = " + key + ", Value = " + capitals[key])
```

Output

```
Key = 'USA', Value = 'Washington, D.C.'
Key = 'France', Value = 'Paris'
Key = 'Japan', Value = 'Tokyo'
Key = 'India', Value = 'New Delhi'
```

Updating a Dictionary

As mentioned earlier, the key cannot appear more than once. Use the same key and assign a new value to it to update the dictionary object.

```
>>> captains={"England":"Root", "Australia":"Smith", "India":"Dhoni"}
>>> captains['India']='Virat'
>>> captains['Australia']='Paine'
>>> captains
{'England': 'Root', 'Australia': 'Paine', 'India': 'Virat'}
```

Use a new key and assign a value to it. The dictionary will show an additional key-value pair in it.

```
>>> captains['SouthAfrica']='Plessis'
>>> captains
{'England': 'Root', 'Australia': 'Paine', 'India': 'Virat', 'SouthAfrica':
'Plessis'}
```

Deleting Values from a Dictionary

Use the **del** keyword to delete a pair from a dictionary or the dictionary object itself. To delete a pair, use its key as parameter. To delete a dictionary object, use its name.

```
>>> captains={'England': 'Root', 'Australia': Paine', 'India': 'Virat', 'Srilanka': 'Jayasurya'}
>>> del captains['Srilanka']
>>> captains
{'England': 'Root', 'Australia': Paine', 'India': 'Virat'}
>>> del captains
>>> captains
NameError: name 'captains' is not defined
```

NameError indicates that the dictionary object has been removed from memory.

View Keys and Values

The keys() and values() methods of Python dictionary class return a view object consisting of keys and values respectively, used in the dictionary.

```
>>> d1 = {'name': 'Steve', 'age': 21, 'marks': 60, 'course': 'Computer Engg'}
>>>d1.keys()
dict_keys(['name', 'age', 'marks', 'course'])
```

The result of the keys() method is a view which can be stored as a list object. If a new key-value pair is added, the view object is automatically updated.

```
>>> keys=d1.keys()
>>> keys
dict_keys(['name', 'age', 'marks', 'course'])
>>>d1.update({"college":"IITB"})
```

```
>>> keys
dict_keys(['name', 'age', 'marks', 'course', 'college'])
```

This is similar for the values() method.

```
>>> d1= {'name': 'Steve', 'age': 21, 'marks': 60, 'course': 'Computer Engg'}
>>>values=d1.values()
dict_values(['Steve', 21, 60, 'Computer Engg'])<
```

The result of the values() method is a view which can be stored as a list object. If a new key-value pair is added, the view is dynamically updated.

```
>>> d1.update({"college":"IITB"})
>>> values
dict_values(['Steve', 21, 60, 'Computer Engg', 'IITB'])<
```

Multi-dimensional Dictionary

Let's assume there are three dictionary objects, as below:

```
>>> d1={"name":"Steve","age":25, "marks":60}
>>> d2={"name":"Anil","age":23, "marks":75}
>>> d3={"name":"Asha", "age":20, "marks":70}<
```

Let us assign roll numbers to these students and create a multi-dimensional dictionary with roll number as key and the above dictionaries at their value.

```
>>> students={1:d1,2:d2,3:d3}
>>> students
{1: {'name': 'Steve', 'age': 25, 'marks': 60}, 2: {'name': 'Anil', 'age': 23, 'marks': 75}, 3: {'name': 'Asha', 'age': 20, 'marks': 70}}<
```

The students object is a two-dimensional dictionary. Here d1, d2 and d3 are assigned as values to keys 1,2, and 3, respectively. students [1] returns d1.

```
>>> students[1]
{'name': 'Steve', 'age': 25, 'marks': 60}
```

The value of a key inside d1 can be obtained as below:

```
>>> students[1]['age']
25
```

Built-in Dictionary Methods

len()

Returns the number of key:value pairs in the dictionary.

```
>>> lang={'A':('Ada','ActionScript'), 'P':("Python", "Perl","PHP")}
>>> len(lang)
2
```

max()

If all keys in the dictionary are numbers, the heighest number will be returned. If all keys in the dictionary are strings, the one that comes last in alphabetical order will be returned.

```
>>> lang={'J':'Java', 'A': 'ActionScript', 'P':'Python'}
>>> max(lang)
'P'
>>> num={5:"five", 100:"hundred",3:"three"}
>>> max(num)
100
```

min()

If all keys in the dictionary are numbers, the lowest number will be returned. If all keys in the dictionary are strings, the one that comes first in alphabetical order will be returned.

```
>>> lang={'J':'Java', 'A': 'ActionScript', 'P':'Python'}
>>> min(lang)
'A'
>>> num={5:"five", 100:"hundred",3:"three"}
>>> min(num)
3
```

pop()

Returns the value associated with the key and the corresponding key-value pair is removed.

```
>>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat',
'Pakistan': 'Sarfraz'}
>>> captains.pop('India')
'Virat'
>>> captains
{'England': 'Root', 'Australia': 'Smith', 'Pakistan': 'Sarfraz'}
```

clear()

Returns empty object by deleting all the key-value pairs.

```
>>> captains
{'England': 'Root', 'Australia': 'Smith', 'Pakistan': 'Sarfraz'}
>>> captains.clear()
>>> captains
{}
```

items() :

Returns a list of tuples, each tuple containing the key and value of each pair.

```
>>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}
>>> captains.items()
dict_items([('England', 'Root'), ('Australia', 'Smith'), ('India', 'Virat'), ('Pakistan', 'Sarfraz')])
```

keys()

Returns a list object comprising of the keys in the dictionary.

```
>>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}
>>> captains.keys()
dict_keys(['England', 'Australia', 'India', 'Pakistan'])
```

values():

Returns a list object comprising of the values in the dictionary.

```
>>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}
>>> captains.values()
dict_values(['Root', 'Smith', 'Virat', 'Sarfraz'])
```

update()

Adds key-value pairs from the second dictionary object to the first. If the second dictionary contains a key already used in first dictionary object, its value is updated.

```
>>> mark1={"Sagar":67, "Amrita":88, "Bhaskar":91, "Kiran":49}
>>> mark2={"Arun":55, "Bhaskar":95, "Geeta":78}
>>> mark1.update(mark2)
>>> mark1
{'Sagar': 67, 'Amrita': 88, 'Bhaskar': 95, 'Kiran': 49, 'Arun': 55, 'Geeta':
78}
```

mark1 dictionary now has new items from mark2, with the value of one key updated.