Python Dictionary

A python dictionary is a mutable data type consisting of a collection of key and value pairs separated by a colon (:), enclosed in curly braces {}.

Dictionaries are optimized to retrieve values when the key is known.

Creating Python Dictionary

Creating a dictionary is as simple as placing items inside curly braces {} separated by commas. An item has a key and a corresponding value that is expressed as a pair (**key:value**).

While the **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.

We can create a dictionary using {} brace or using the built-in dict() function.

```
# empty dictionary
>>> y_dict = {}
# dictionary with integer keys
>>> my_dict = {1: 'apple', 2: 'ball'}
# dictionary with mixed keys
>>>my_dict = {'name': 'John', 1: [2, 4, 3]}
# using dict()
>>> my_dict = dict({1:'apple', 2:'ball'})
# from sequence having each item as a pair
>>> my_dict = dict([(1,'apple'), (2,'ball')])
```

Accessing Elements from Dictionary

While indexing is used with other data types to access values, a **dictionary uses keys**. Keys can be used either inside **square brackets []** or with the **get() method**.

If we **use the square brackets []**, **KeyError** is raised in **case a key is not found** in the dictionary. On the other hand, the get() method returns None if the key is not found.

```
# get() vs [] for retrieving elements
     >>> my_dict = {'name': 'Jack', 'age': 26}
     >>> print(my_dict['name'])         # Output: Jack
     >>> print(my_dict.get('age'))      # Output: 26
     # Trying to access keys which doesn't exist throws error
     >>> print(my_dict.get('address'))   # Output: None
     >>> print(my_dict['address'])       # Output:  KeyError
     >>>points={"p1":(10,10), "p2":(20,20)}
     >>>points.get("p2")                # Output: (20,20)
     >>>numbers={1:"one", 2:"Two", 3:"three",4:"four"}
     >>>numbers.get(2)                  # Output: 'Two'
```

We can 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'
```

Access elements using items() method

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()
Output: dict_items([('England', 'Root'), ('Australia',
'Smith'), ('India', 'Virat'), ('Pakistan', 'Sarfraz')])
```

Changing and Adding Dictionary elements

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

If the key is already present, then the existing value gets updated. In case the key is not present, a new (key: value) pair is added to the dictionary.

```
    # Changing and adding Dictionary Elements
    >>> my_dict = {'name': 'Jack', 'age': 26}
    # update value
    >>> my_dict['age'] = 27
    #Output: {'age': 27, 'name': 'Jack'}
    >>> print(my_dict)
    # add item
    >>> my_dict['address'] = 'Downtown'
    # Output: {'address': 'Downtown', 'age': 27, 'name':
'Jack'}
    >>> print(my_dict)
    Output: {'name': 'Jack', 'age': 27}
            {'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

Removing elements from Dictionary

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

The **popitem()** method can be used to **remove and return the last (key, value)** item pair from the dictionary.

The remove() method can be used to remove and return an arbitrary (key, value) item pair from 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.

```
# Removing elements from a dictionary
     # create a dictionary
     >>> squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
     # remove a particular item, returns its value
     >>> print(squares.pop(4))          # Output: 16
     >>> print(squares)       # Output: {1: 1, 2: 4, 3: 9, 5: 25}
     # remove an arbitrary item, return (key,value)
     >>> print(squares.popitem()) # Output: (5, 25)
     >>> print(squares)       # Output: {1: 1, 2: 4, 3: 9}
     >>> squares.clear()      # remove all items
     >>> print(squares)       # Output: {}
     >>> captains={'England': 'Root', 'Australia': Paine',
     'India': 'Virat', 'Srilanka': 'Jayasurya'}
     >>> del captains['Srilanka']  #Syntax:
del(tuplename[value])
     >>> captains
     Output: {'England': 'Root', 'Australia': Paine', 'India': 'Virat'}
     >>> del captains
     >>> captains
     # Output: NameError: name 'captains' is not defined
```

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

Updating a Dictionary

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

To add a new item in the dictionary **use an unique key for index and assign value to it** to update the dictionary. The dictionary will show an additional key-value pair in it.

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

Add items using update() method

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

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

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

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()
Output: dict_keys(['name', 'age', 'marks', 'course'])
>>> d1.values()
Output: dict_values(['Steve', 21, 60, 'Computer Engg'])
```

The result of the keys() and values() methods provide a view which can be stored as a list object. If a new key-value pair is added, the **view object is automatically updated**. Similarly 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 = {'name': 'Steve', 'age': 21, 'marks': 60, 'course': 'Computer Engg'}
>>> keys=d1.keys()
>>> values=d1.values()
>>> keys
Output: dict_keys(['name', 'age', 'marks', 'course'])
>>> values
```

```
Output: dict_values(['Steve', 21, 60, 'Computer Engg'])
>>> d1.update({"college":"IITB"})
>>> keys
Output: dict_keys(['name', 'age', 'marks', 'course',
'college'])
>>> values
Output: 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
Output: {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]
Output: {'name': 'Steve', 'age': 25, 'marks': 60}
```

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

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

Built-in Dictionary Function

len()

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

```
>>> lang={'A':('Ada','ActionScript'),'P':("Python", "Perl",
"PHP")}
>>> len(lang)       # Output: 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)
# Output: 3
```