Informatics Practices

DICTIONARIES

In

Python

Rajesh Verma, MRA DAV Public School, Solan

Dictionaries

- A dictionary is a data type similar to arrays, but works with keys and values instead of indexes. Each value stored in a dictionary can be accessed using a key, which is any type of object (a string, a number, a list, etc.) instead of using its index to address it.

Characteristics of a Dictionary

- Unordered set
- Not a sequence
- Indexed by keys, not numbers
- Keys must be unique
- Dictionaries are mutable
- Internally stored as mapping

Creating A Dictionary

- It is enclosed in curly braces { }.
- Each item is separated from other item by a comma(,).
- Within each item, key and value are separated by a colon (:).

Syntax :

Dictionary-name = {<key>:<value>, <key>:<value>}

e.g.

dict = {‘Subject': ‘Informatics Practices', 'Class': ‘11'}

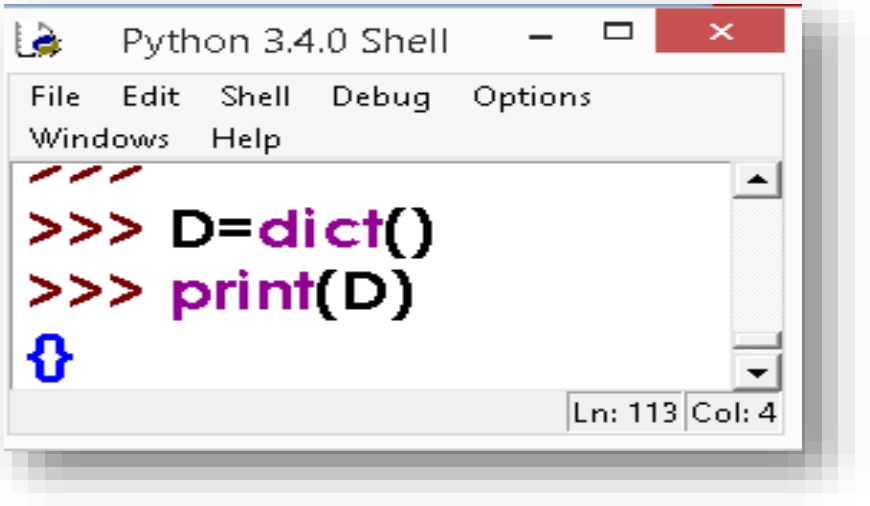
STATE DIAGRAM

```
>>> A={1:"one",2:"two",3:"three"}
```

A =	1	one
	2	two
	3	three
	KEYS	VALUES

CREATING DICTIONARAY – dict()

- The function dict( ) is used to create a new dictionary with no items. This function is called built-in function.



Accessing Elements of a dictionary

In dictionaries, the elements are accessed through the keys defined in the key : value pairs.

Syntax:

```
dictionary-name[<key>]
```

```
>>>dict = {'Subject': 'Informatics Practices', 'Class': 11}
>>>print(dict)
>>>print ("Subject : ", dict['Subject'])
>>>print ("Class : ", dict.get('Class'))
```

Creating and Traversing a Dictionary

- Traversal of a collection means accessing and processing each element of it. Traversing a dictionary is done with python loops. The for loop makes it easy to traverse or loop over the same items in a dictionary.
- Dictionaries can be iterated over, just like a list. However, a dictionary, unlike a list, does not keep the order of the values stored in it.

Syntax:

```
for <item> in <dictionary>
        Process each item
```

e.g.

```
device = {‘four’:’scanner’, ‘three’:’printer’, ‘two’:’mouse’, ‘one’:’keyboard’}
 for i in device:
    print(device[i])
```

Zip(): Joins to items together

- Write a python program to create a dictionary namely **dict** from the following two lists or tuples.

Solution:

```
a = ["John", "Charles", "Mike“]
b = ["Jenny", "Christy", "Monica“]
dict = dict(zip(a, b))
print(dict)
```

Adding elements to dictionary

- You can add new elements (key:value pair) to a dictionary using assignments.

Syntax :

<Dictionary-name>[<key>] = <value>

e.g.

```
>>>Employee = {‘name’ : ‘Mark’, ‘salary’: 10000,’age’: 25}
>>>Employee[‘dept’] = ‘sales’
>>>Employee
```

Output :

```
{'name': 'Mark', 'salary': 10000, 'age': 25, 'dept': 'sales'}
```

Checking for Existence of a key

- Membership operators in and not in work with dictionary.

Syntax:

<key> in <dictionary>

<key> not in <dictionary>

- The in operator will return True if the given key is present in the dictionary.
- The not in operator will return True if the given key is not present in the dictionary, otherwise False.

```
>>> std = {'name': 'John', 'Subject':'Informatics Practices', 'Class':11}
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'Class': 11}
>>> std1 = {'name': 'John', 'Subject':'Informatics Practices', 'Class':11}
>>> std.clear()
>>> std
{}
>>> 'name' in std1
True
>>> 'John' in std1
False
>>> 'john' not in std1
True
>>> 'age' not in std1
True
>>> 'class' not in std1
True
>>> 'Class' not in std1
False
>>> |
```

Del statement: Removing a value

- To delete a dictionary element or a dictionary entry i.e. a key: value pair.

Syntax : del <dictionary>[<key>]

e.g.

```
phonebook = {   "John" : 938477566,
                "Jack" : 938377264,
                "Jill" : 947662781}
del phonebook["John"]
print(phonebook)
```

Dictionary Functions and Methods

Dictionary Method	Meaning
Len()	It returns the length of the dictionary.
dict.get(key)	To get value of the given key
dict.items()	To get all the items(the key:value pairs) of the dictionary.
dict.keys()	To get all the keys of the dictionary
dict.values()	To get all the values of the dictionary.
dict.setdefault()	To insert a new key:value pair only if the key does not already exist.

The len() function

This function returns length of the dictionary i.e. the count of elements (key: value pairs) in the dictionary.

Syntax: len(<dictionary name>)

Python 3.8.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> emp = {'name':'Mark','salary':10000, 'age':23}
>>> emp
{'name': 'Mark', 'salary': 10000, 'age': 23}
>>> len(emp)
3
>>> |
```

The get() Method

To get the value of the given key. If the key is not present , you can specify your own message.
Syntax: <dictionary>.get(default)

Python 3.8.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> emp = {'salary':10000, 'dept':'sales', 'name':'Jack', 'age':23}
>>> emp.get('dept')
'sales'
```

The items() Method

This method returns all of the items in the dictionary as a sequence of(key,value) tuples.

Syntax: <dictionary>.items()

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> std = {'name':'John', 'Subject':'Informatics Practices', 'class': 11}
>>> std.items()
dict_items([('name', 'John'), ('Subject', 'Informatics Practices'), ('class', 11)])
>>> l = std.items()
>>> for x in l:
        print(x)


('name', 'John')
('Subject', 'Informatics Practices')
('class', 11)
>>> for ky,vl in l:
        print(ky,vl)


name John
Subject Informatics Practices
class 11
>>> |
```

The keys ( ) method

This method returns all the keys in the dictionary as a sequence of keys.

Syntax: <dictionary>.keys()

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> std = {'name':'John', 'Subject':'Informatics Practices', 'class': 11}
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'class': 11}
>>> std.keys()
dict_keys(['name', 'Subject', 'class'])
>>>
```

The values ( ) method

This method returns all the values in the dictionary as a sequence of values.

Syntax: <dictionary>.values()

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> std = {'name':'John', 'Subject':'Informatics Practices', 'class': 11}
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'class': 11}
>>> std.values()
dict_values(['John', 'Informatics Practices', 11])
>>>
```

The setdefault( ) method

The setdefault( ) method insert a new key : value pair ONLY IF the key doesn't already exist. If the key already exists, it returns the current value of the key.

Syntax: <dictionary>.setdefault(<key>, value)

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> std = {'name':'John', 'Subject':'Informatics Practices', 'class': 11}
>>> std.setdefault('Roll Number', 20)
20
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'class': 11, 'Roll Number': 20}
>>> std.setdefault('name', 'Rajesh')
'John'
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'class': 11, 'Roll Number': 20}
>>> |
```

The update( ) method

The update( ) method merges key:value pairs from the new dictionary into the original dictionary, adding or replacing as needed. The items in the new dictionary are added to the old one and override any items already there with the same keys.
Syntax: <dictionary>.update(<other-dictionary>)

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> std = {'name':'John', 'Subject':'Informatics Practices', 'class': 11}
>>> std1 = {'name':'Rajesh', 'RollNo':11, 'Mobt':90}
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'class': 11}
>>> std1
{'name': 'Rajesh', 'RollNo': 11, 'Mobt': 90}
>>> std.update(std1)
>>> std
{'name': 'Rajesh', 'Subject': 'Informatics Practices', 'class': 11, 'RollNo': 11, 'Mobt': 90}
>>> std1
{'name': 'Rajesh', 'RollNo': 11, 'Mobt': 90}
>>> |
```

Dictionary Functions and Methods

Dictionary Method	Meaning
dict.update()	This method merges key: value pairs from the new dictionary into the original dictionary, adding or replacing as needed.
dict.copy()	Create a copy of a dictionary using copy(). A copy of keys is created with the new name and the values referenced are shared by the two copies
dict.pop()	It removes and returns the dictionary element associated to passed key.
dict.popitem()	It remove and returns the items which was the last item entered in the dictionary.
dict.clear()	This method removes all items from the dictionary and the dictionary becomes empty dictionary .

The copy( ) method

The copy( ) method creates a copy of a dictionary.

Syntax: <dictionary>.copy( )

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> std = {'name':'John', 'Subject':'Informatics Practices', 'class': 11}
>>> std1 = {'name':'Rajesh', 'RollNo':11, 'Mobt':90}
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'class': 11}
>>> std1
{'name': 'Rajesh', 'RollNo': 11, 'Mobt': 90}
>>> std.update(std1)
>>> std
{'name': 'Rajesh', 'Subject': 'Informatics Practices', 'class': 11, 'RollNo': 11, 'Mobt': 90}
>>> std1
{'name': 'Rajesh', 'RollNo': 11, 'Mobt': 90}
>>> |
```

The pop( ) method

The pop( ) method removes and returns the dictionary element associated to passed key

Syntax: <dictionary>.pop(key, <value>)

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> std = {'name':'John', 'Subject':'Informatics Practices', 'class': 11}
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'class': 11}
>>> std.pop('class')
11
>>> std
{'name': 'John', 'Subject': 'Informatics Practices'}
>>>
```

The popitem( ) method

The popitem( ) method removes and returns a (key, value) pair from the dictionary. It returns the items which was the last item in the dictionary. The items will removed from the dictionary in the LIFO (Last In First Out) order.
It returns the deleted key:value pair in the form of a tuple. If the dictionary is empty, calling popitem raise a keyerror.
Syntax: <dictionary>.popitem()

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> std = {'name':'John', 'Subject':'Informatics Practices', 'class': 11}
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'class': 11}
>>> std.pop('class')
11
>>> std
{'name': 'John', 'Subject': 'Informatics Practices'}
>>> std.popitem()
('Subject', 'Informatics Practices')
>>> std
{'name': 'John'}
>>> |
```

The clear( ) method

This method removes all items from the dictionary and the dictionary becomes empty dictionary post this method

Syntax: <dictionary>.clear()

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> std = {'name': 'John', 'Subject':'Informatics Practices', 'Class':11}
>>> std
{'name': 'John', 'Subject': 'Informatics Practices', 'Class': 11}
>>> std1 = {'name': 'John', 'Subject':'Informatics Practices', 'Class':11}
>>> std.clear()
>>> std
{}
>>>
```

Any Questions Please