CLASS – XI

Dictionary

STUDY NOTE

Period -01

Introduction

You must have realized that Python offers many different ways to organize collections (i.e., a bunch of values in a single "variable") of data items, such as strings, lists, dictionaries, tuples etc. Of these you have already worked with strings, lists and tuples in previous chapters. Now is the time to work with other collection types. So, in this chapter, we shall be talking about dictionaries. In this chapter, you shall be learning about how data-items are organized in a Python dictionary, how you can access items from a dictionary, various operations that you can perform on a dictionary and various related functions and methods.

Dictionary – Key: Value Pairs

Among the built-in Python data types is a very versatile type called a dictionary. Dictionaries are simply another type of collection in Python, but with a twist. Rather than having an index associated with each data-item (just like in lists or strings), Dictionaries in Python have a "key" and a "value of that key". That is , Python dictionaries are collection of some key-value pairs.

Do not get confused, just read on. Just like in English dictionaries you can search for a word's meaning, because for each word, there is a meaning associated with it. In the same manner, Python dictionaries have some keys (just like English dictionaries have words) and associated values (just like English dictionaries have associated meanings for every word).

Dictionaries are containers that associate keys to values. This is, in a way, similar to lists. In lists, you must remember the index value of an element from the list, but with the case of dictionaries, you'll have to know the key to find the element in the dictionaries.

All this will become clear to you while you go through following sections.

Creating a Dictionary

To create a dictionary, you need to include the key, value pairs in curly braces as per following syntax :

<dictionary-name>={ <key>: <value>, <key> : <value>....}

Following is an example dictionary by the name teachers that stores the names of teachers as keys and the subjects being taught by them as values of respective keys.

Teachers = {'Dimple" : "Computer Science", "Karen" : "Sociology", "Harpreet" : Mathematics", "Sabah" : Legal Studies"}

Notice that

- The curly brackets mark the beginning and end of the dictionary.
- Each entry (Key : Value) consists of a pair separated by a colon – the key and corresponding value is given by writing colon (☺ between them,
- The key-value pairs are separated by commas (,).

As you can see that there are four key : value pairs in above dictionary. Following table illustrates the key-value relationships in above dictionary teachers.

Key-Value pair	Key	Value
"Dimple" : "Computer Science"	"Dimple"	"Computer Science"
"Karen" : "Sociology"	"Karen"	"Sociology"
"Harpreet" : "Mathematics"	"Harpreet"	"Mathematics"
"Sabah" : "Legal Studies"	"Sabah"	"Legal Studies"

Consider some more dictionary declarations:

$dict1 = \{\ \}$ # It is an empty dictionary with no elements

DaysInMonths = { "January" : 31, "February" : 28, "March" : 31, "April" : 30, "May" : 31,
"June" : 30, "July" : 31, "August" : 31, "September" : 30,
"October" : 31, "November" : 30, "December" : 31 }

BirdCount = {"Finch" : 10, "Myna" : 13, "Parakeet" : 16,
"Hornbil1" : 15, "Peacock" : 15 }

Now you can easily identify the keys and corresponding values from above given dictionaries.

One things that you must know is that keys of a dictionary must be of immutable types, such as :

- A Python string
- A number
- A tuple (containing only immutable entries).

If you try to give a mutable type as key, Python will give you an error as : "unhashable type". See below :

$>>> dict3 = \{[2,3]:"abc"\}$

:

TypeError : unhashable type : 'list'

Accessing Elements of a Dictionary

While accessing elements from a dictionary, you need the key. While in lists, the elements are accessed through their index ; in dictionaries, the elements are accessed through the keys defined in the key : value pairs, as per the syntax shown below :

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

Thus to access the value for key defined as "Karen" in above declared teachers dictionary, you will write :

$>>> teachers["Karen"]$

and python will return

sociology

Similarly, following statement

>>> print ("Karen teachers", teachers ['Karen'])

will give output as :

Karen teaches sociology

While giving key inside square brackets gives you assess only to the value corresponding the mentioned, key, mentioning the dictionary name without any square brackets prints/ displays the entire contents of the dictionary.

Consider following example :

```
>>> d = {"Vowel1" : "a", "Vowel2" : "e", "vowel3" : "i", "Vowel4" : "o", "Vowel5" : 'u"}
>>> d
 {'Vowel5' : 'u', 'Vowel4' : 'o', 'Vowel3' : 'i', 'Vowel2' : 'e', 'Vowel1' : 'a'}
 >>>print (d)
{'Vowel5' : 'u', 'Vowel4' : 'o', 'Vowel3' : 'i', 'Vowel2' : 'e', 'Vowel1' : 'a'}
 >>> d["Vowel1"]
 'a'
>>> d["Vowel4"]
 'o'
```


As per above examples, we can say that key order is not guaranteed in Python, This is because in Python dictionaries, the elements (key : value pairs) are unordered ; one cannot access element as per specific order. The only way to access a value is through key. Thus we can say that keys act like indexes to access values from a dictionary.

Also, attempting to access a key that doesn't exist causes an error. Consider the following statement that is trying to access a non-existent key (13) from dictionary d.

```
 >>> d[13]
Trace back (most recent call last) :
File "<pyshell#7>", line 1, in <module>
 d KeyError : 13
 [13]
```

The above error means that before we can access the value of a particular key using expression such as d["13'], we must first ensure the key (13 here) exists in the dictionary.

Like list elements, the keys and values of a dictionary are stored through their references. Figure illustrates the same for a dictionary

{"goose" : 3, "tern" : 3, "hawk" : 1}

Traversing a Dictionary:-

Traversal of a collection means accessing and processing each element of it. Thus traversing a dictionary also means the same and same and same is the tool for it, i.e, the python loops.

The for loop makes it easy to traverse or loop over the items in a dictionary, as per following syntax:

```
for <item> in <Dictionary> :
process each item here
```

The loop variable <item> will be assigned the keys of <Dictionary> one by one, (just like, they are assigned indexes of strings or lists while traversing them), which you can use inside the body of the for loop.

Consider following example that will illustrate this process. A dictionary namely d1 is defined with three keys - a number, a string, a tuple of integers.

```
d1 = {t : "number", /
      "a" : "string", /
       (1,2) : "tuple"}
```

To traverse the above dictionary, you can write for loop as :

```
for key in d1 :
 print (key, ":", d1 [key])
```

The above loop will produce the output as shown here

```
a : string
(1, 2) : tuple
5 : number
```

How it works:

The loop variable key in above loop will be assigned the keys of the Dictionary d1, one at a time. As dictionary elements are unordered, the order the assignment of keys may be different from what you stored.

Using the loop variable, which has been assigned one key at a time, the corresponding value is printed along with the key inside the loop-body using through statement:

```
print (key, ":", d1 [key])
```

As per above output, loop-variable key will be assigned 'a' in first iteration and hence the key "a" and its value "string" will be printed ; in second iteration, key will get element (1, 2) and its value 'tuple' will be printed along with it; and so on.

So, you can say that traversing all collections in similar. You can use a for loop to get hold of indexes or keys and then print or access corresponding values inside the loop-body.

Write a program to create a phone dictionary for all your friends and then print it.

```
PhoneDict = {"Madhav" : 1234567, "Steven" : 7654321, "Dilpreet" : 6734521, "Rabiya" : 4563217, "Murughan" : 3241567, "Sampree" : 4673215}
    for name in PhoneDict:
            print (name, ":", PhoneDict[name])
```

The output produced by above program will be

```
Rabiya :        4563217
Murughan :      3241567
Madhav :        1234567
Dilpreet :      6734521
Steven :        7654321
Sampree :       4673215
```

As you can make out from the output, that the loop variable name got assigned the keys of dictionary PhoneDict, one at a time. For the first iteration, it got the key "Rabiya", for the second iteration, it was assigned "Murughan" and so on. Since the dictionaries are unordered set of elements, the printed order of elements is not same as the order you stored the elements in.

Accessing Keys or Values Simultaneously:

To see all the keys in a dictionary in one go, you may write <dictionary>.keys() and to see all values in one go, you may write <dictionary>.values(), as shown below (for the same dictionary d created above) :

```
>>> d. keys ()
dict_keys ( ['Vowel5', 'Vowel4', 'Vowel3', 'Vowl2', 'Vowel1'])
 >>> d . values ()
dict_values (['u', 'o', 'i', 'e', 'a'])
```


As you can see that the keys () function returns all the keys defined in a dictionary in the form of a sequence and also the values () function returns all the values defined in that dictionary in the form of a sequence.

You can convert the sequence returned by keys () and values () functions by using list () as shown below?

```
 >>> list (d.keys ())
['Vowel5', 'Vowel4', 'Vowel3', 'Vowel2', 'Vowel1' ]
 >>> list (d.values ( ) )
['u', 'o', 'i', e', 'a']
```

Quick Interesting Facts:

A dictionary is an **unordered** and **mutable** Python **container** that stores mappings of unique **keys to values**. Dictionaries are written with curly brackets ({}), including **key-value** pairs separated by commas (,). A colon (:) separates each **key** from its **value**.

PERIOD-02.

Creating and adding elements to Dictionary

Characteristics of a Dictionary:-

Dictionaries like lists are mutable and that is the only similarity they have with lists. Otherwise, dictionaries are different type of data structures with following characteristics:

1. Unordered Set : A dictionary is a unordered set of key : value pairs. Its values can contain references to any type of object.

2. Not a sequence : Unlike a string, list and tuple, a dictionary is not a sequence because it is unordered set of elements. The sequences are indexed by a range of ordinal numbers. Hence, they are ordered, but a dictionary is an unordered collection.

3. Indexed by Keys, Not Numbers : Dictionaries are indexed by keys. According to Python, a key can be "any non-mutable type" Since strings and numbers are not mutable, you can use them as a key in a dictionary. And if a tuple contains immutable objects such as integers or strings etc, then only it can also be used as a key. But the values in a dictionary can be of any type, and types can be mixed within only dictionary. Following dictionary dict1 has keys of different immutable types.

```
>>> dict1 = {
        θ : "Value for key θ", 1 : "value for key 1",
        "3" : "Value for a string -as-a-key",
        (4, 5) : "Value for a tuple-as-a-key",
         "and for fun" : 7
 }
>>> dict1 [θ]
'Value for key θ'
>>> dict1[(4, 5)]
 'Value for a tuple-as-a-key'
>>> dict1 [4, 5]
 'Value for a tuple-as-a-key'
  >>> dict1 ["3"]
'Value for a string-as-a-key'
```

4. Keys must be Unique : Each of the keys within a dictionary must be unique. Since keys are used to identify values in a dictionary, there cannot be duplicate keys in a dictionary.

However, two unique keys can have same values, e.g, consider the same BirdCount dictionary declared above:

BirdCount = { "Finch" : 10, "Myna" : 13, "Parakeet" : 16, "Hornbill" : 15, "Peacock" : 15}

Notice that two different keys "Hornbill" and "Peacock" have same value 15.

5. Mutable: Like lists, dictionaries are also mutable. We can change the value of a certain key "in place" using the assignment statement as per syntax:

<dictionary> [,key.] = <value>

For example, consider the dictionary dict1 defined above:

```
>>> dict1["3]
'Value for a string-as-a-key'
>>> dict1 ["3"] = "Changed to new string"
>>> dict1 ["3"]
'Changed to new string'
```

You can even add a new key : value pair to a dictionary using a simple assignment statement. But thekey being added should be unique. If the key already exists, then value is simply changed as in the assignment statement above.

```
>>> dict1 ["new"] = "a new pair is added"
>>> dict1
{θ : 'Value for key θ', 1 : 'Value for key 1', (4, 5) : 'Value for a tuple-as-a-key', '3' : 'Value for a string -as-a-key', 'and for fun' : 7, 'new' : 'a new pair is added'}
```

6. Internally stared as Mappings : Internally, the key : value pairs of a dictionary are associated with one another with some internal function (called hash-function [1]). This way of linking is called mapping.

Working with dictionaries:

After basics of dictionaries, let us discuss about various operations that you can perform on dictionaries, such as adding elements to dictionaries, updating and deleting elements of dictionaries. This section is dedicated to the same.

Multiple Ways of Creating Dictionaries:-

You have learnt to create dictionaries by enclosing key:value pairs in curly braces. There are other ways of creating dictionaries too that we going to discuss here.

1. **Initializing a Dictionary**: In this method all the key:value pairs of a dictionary are written collectively, separated by commas and enclosed in curly braces. You have already worked with this method. All the dictionaries created so far have been created through this method, e.g following dictionary Employee is also being created the same way :

```
Employee = {'name' : 'John', 'salary' " 10000, 'age' : 24}
```

2. **Adding key : value pairs to an Empty Dictionary** : In this method, firstly an empty dictionary is created and then keys and values are added to it one pair at a time.

To create and empty dictionary, there are two ways :

(i) By giving dictionary contents in empty curly braces as shown below:

```
Employee = { }
```

(ii) By using dictionary constructor dict ( ) as shown below:

```
Employee = dict ( )
```

Next step is to add key : value pairs, one at a time as per syntax given below.

```
<dictionary> [<key>] = <value>
```

For example, in the adjoining statements we are first adding a key : value pair of 'name' : 'John', next statements adds value 10000 for key 'salary' and the third and last statement adds value 24 for key 'age'. This method of creating dictionaries is used to create dictionaries dynamically, at runtime.

3. **Creating a Dictionary from name and value pairs**: Using the dict ( ) constructor of dictionary, you can also create a new dictionary initialized from specified set of keys and values. There are multiple ways to provide keys and values to dict ( ) constructor.

(i) Specify Key : Value pairs as keyword arguments to dict ( ) function, keys as arguments and values as their values. here the argument names are taken as keys of string type.

```
Employee = dict (name = 'John', salary = 10000, age = 24)
>>> Employee
{ 'salary' : 10000, 'age' : 24, 'name' : 'John'}
```

(ii) Specify comma-seprarted key : value pairs. To give key : value pairs in this format, you need to enclose them in curly braces as shown below :

```
>>> Employee = dict ( {'name' : 'Johm', 'salary' : 10000, 'age' : 24} )
>>> Employee
{'salary' : 10000, 'age' : 24, 'name' : 'John'}
```

(iii) Specify keys separately and corresponding values separately. In this method, the keys and values are enclosed separately in parentheses (i.e, as tuples) and are given as arguments to the zip ( ) function, which is then given as argument of dict ( )

```
>>> Employee = dict (zip (  ('name', 'salary', 'age'), ('John', 10000, 24) ) )
>>> Employee
{'salary' : 10000, 'age' : 24, 'name' : 'John'}
```

The zip function clubs first value from first set with the first value of second set, second value from first set with the second value of second set, and so on, e.g, in above example, 'name' is clubbed constructor method creates key : value pairs for the dictionary.

(iv) Specify key : value pairs separately in form of sequences. In this method, one list or tuple argument is passed to dict ( ). This argument (passed list or tuple) contains lists/ tuples of individual key : value pairs.

That is, a key : value pair is specified in the form of alist and there are as many lists as items of the outer list as there are key : value pairs in the dictionary.

Consider the following example:

```
>>> Employee = dict ([['named, 'Johrn'], ['salary', 10000], ['age', 24]] )
>>> Employee
{'salary' : 10000, 'age' : 24, 'name' : 'John'}
```

You can also pass a tuple containing key : value pairs as list -element or tuples as elements. See following examples.

```
>>> Empl2 = dict ( ( ['name', 'John'}, ['salary', 10000], ['age', 24] ) )
>>> Empl2
{'salary' : 10000, 'age' : 24, 'name' : 'John'}
>>> Empl3 = dict ( ( ('name', 'John'), ('salary', 10000), ('age', 24) ) )
>>> Empl3
{'salary' : 10000, 'age' : 24, 'name' : 'John}
>>>
```

Adding Elements to Dictionary:-

You can add new elements (key : value pair) to a dictionary using assignment as per following syntax. BUT the key being added must not exist in dictionary and must be unique. If the key already exists, then this statement will change the value of existing key and no new entry will be added to dictionary.

```
<dictionary> [<key>] = <value>
```

Consider the following example :

```
>>> Employee = {'name' : 'John', 'salary' : 10000, 'age' : 24}
>>> Employee ['dept'] = 'sales'
>>> Employee
{'salary' : 10000, 'dept' : 'sales', 'age' : 24, 'name' : 'John'}
```

Nesting Dictionaries:-

You can even add dictionaries as values inside a dictionary. Program uses such a dictionary. Storing a dictionary inside another dictionary is called nesting of dictionaries. But one thing you must know about nesting of dictionaries is that, you can store a dictionary as a value only, inside a dictionary. You cannot have a key of dictionary type; you know the reason - only immutable types can form the keys of a dictionary.

A dictionary contains details fo two workers with their name as keys and other details in the form of dictionary as value.

Write a program to print the workers' information in records format.

```
Employees = {'John' : 25, 'salary : 20000'}, 'Diya' : {'age' : 35, 'salary' : 50000} } for key in Employees :
        print ("Employee", key, ':')
         print ('Age : ', str (Employees [key] ['age'] ) )
        print ('salary :', str (Employees [key] ['salary'] ) )
 Employee John :
          Age : 25
Employee Diya :
          Age : 35
        Salary : 50000
```

Quick Interesting Facts:

Dictionaries can also be created with the built-in function **dict(**kwarg)**. This function takes an arbitrary number of **keywords arguments** (arguments proceeded by an identifier **kwarg=value**) as input, returning **None**.

PERIOD-03

Updating elements of Dictionary

Updating Existing Elements in a Dictionary:-

Updating an element is similar to what we did just now. That is, you can change value of an existing key using assignment as per following syntax:

dictionary > [<key>] = <value>

Consider following example:

```
>>> Employee = {'name' : 'John', 'salary' : 10000, 'age' : 24}
>>> Employee ['salary'] = 20000
>>> Employee
 {'salary' : 20000, 'age' : 24, 'name' : 'John'}
```

But make sure that the key must exist in the dictionary otherwise new entry will be added to the dictionary. Using this technique of adding key:value pairs, you can create dictionaries interactively at runtime by accepting input from user. Following program illustrates this.

Program : Write a program to create a dictionary containing names of competition winner students as keys and number of their wins as values.

```
    n = int (input ("How many students ?"))
```

```
Compwinners = { }
for a in range (n)
 key = input ("Name of the student :")
 value = int (input("Number of competitions won :") )
 Compwinners [key] = value
print ("The dictionary now is :")
print (CompWinners)
```

The sample run of above program is:

How many students : 5
Name of the student : Rohan
Number of competitions won : 5
Name of the student : Zeba
Number of competitions won : 3
Name of the student : Nihar
Number of competitions won : 3
Nam of the student : Roshan
Number of competitions won : 1
Name of the student : James
Number of competitions won : 5
The dictionary now is :
{'Nihar' : 3, 'Rohan' : 5, 'Zeba' : 3, 'Roshan' : 1, 'James' : 5}

Deleting elements from a Dictionary:

There are two methods for deleting elements from a dictionary.

(i) To delete a dictionary element or a dictionary entry, i.e, a key:value pair, you can use del command. The syntax for doing so is as given below.

```
del<dictionary>[<key>]
```

Consider the following example:

```
>>>empl3
{'salary' : 10000, 'age' : 24, 'name' : 'John'}
>>>del empl3 ['age']
>>> empl3
 {'salary' : 10000, 'name' : 'John'}
```

But with del statement, the key that you are giving to delete must exist in the dictionary, otherwise Python raises exception (KeyError). See below:

```
>>> del empl3['new']
:
KeyError : 'new'
```

(ii) Another method to delete elements from a dictionary is by using pop ( ) method as per following syntax:

<dictionary>.pop(<key>)

The pop ( ) method will not only delete the key:value pair for mentioned key but also return the corresponding value. Consider following code example

```
>>> employee
{'salary' : 10000, 'age' : 24, 'name' : 'John'}
>>> employee. pop('age')
24
>>> employee
{'salary' : 10000, 'name' : 'John'}
```

If you try to delete a key which does not exist, the Python returns error. See below :

```
>>> employee. pop('new')
:
KeyError : 'new'
```

However, pop ( ) method allows you to specify what to display when the given key does not exist, rather than the default error message. This can be done as per following syntax:

<dictionary> . pop(<key>, <in-case-of-error-show-me)

For example :

```
>>> employee. pop ('new', "Not Found")
'Not Fund'
```

See, now python returned the specified message in place of error-message.

Checking for Existence of a Key:

Usual membership operators in and not in work with dictionaries as well. But they can check for the existence of keys only. To check for whether a value is present in a dictionary (called reverse lookup), you need to write proper code for that. (Solved Problem 8 is based on reverse lookup). To use a membership operator for a key's presence in a dictionary, you may write statement as per syntax given below.

<key> in <dictionary>

<key> not in <dictionary>

The in operator will return True if the given key is present in the dictionary, otherwise false.

The not in operator will return true if the given key is not present in the dictionary, otherwise false.

Consider the following example:

```
>>> empl = {'salary' : 10000, 'age' :24, 'name' :'John'}
>>> 'age' in empl
True
>>> 'John' in empl
False
>>> 'John' not in empl
True
>>> 'age' not in empl
False
```

Please remember that the operators in and not in do not apply on value of a dictionary, if you use them with dictionary name directly. That is, if you search for something within or not in operator with a dictionary, it will only look for in the keys of the dictionary and not value, e.g.

```
>>> dictl
{'age' : 25, 'name' : 'xyz', 'salary' : 23480.5}
>>>'name' indictl
True
>>> 'xyz' indictl
False
```

However, if you need to search for a value in a dictionary, then you can use the in operator with >dictionary _ name>.values ( ), i.e

```
>>>'xyz' in dictl. values ( )
True
```

Pretty Printing a dictionary:

You generally use print function to print a dictionary in Python, e.g, if you have a dictionary as Winners = {'Nihar' : 3, 'Rohan' : 5, 'Zeba' : 3, 'Roshan' : 1, 'James' : 5} and if you give a print ( ) to print, it, Python will give result as:

```
>>> print (Winners)
{'Nihar' : 3, 'Rohan' : 5, 'Zeba' : 3, . 'Roshan' : 1, 'James' : 5}
```

From above style of printing dictionary, you can still make out the keys and values, but if the dictionary is large, then shouldn't there be any way of printing dictionary in a way that makes it more readable and presentable?

Well there certainly is a way. For this, you need to import ison module by giving statement import json (recall that in a program the import statement should be given at the top). And then you can use the function dumps ( ) of json module as per following syntax inside print ( ).

json.dumps (<dictionary name>, indent = <n>)

For example, consider following result

```
>>> print (json, dumps (Winners, indent = 2) )
{
 "Nihar" : 3
  "Rohan" : 5,
  "Zeba" : 3,
  "Roshan" : 1,
   "James" : 5
 }
```

Counting Frequency of Elements in a list using Dictionary:

Since a dictionary uses key: value pairs to store its elements, you can use this feature effectively if you trying to calculate frequency of elements in sequence such as lists. You can do this as:

Create an empty dictionary

Take up an element from the list list (first element 1st time, second element 2nd time and so on)

Check if this element exists as a key in the dictionary: If not then add {key:value) to dictionary in the form {List-element : count of List-element in List}

Following program illustrates this. But before we move on to the program code, it will be good if we talk about an interesting and useful function split ( ) that you can use with text to split it in words. (it will come handy especially if try to find frequency of words in a sentence or in a text file). The split ( ) works on string type data and breaks up a string into words and creates a list out of it, e.g.

```
>>> text = "This is sample string"
>>> words = text.split ( )
>>> words
['This', 'is', 'sample', 'string']
```

```
>>> text = "one, two, three"
>>> words = text.split ("  ,  ")
>>> words
['one', 'two', 'three']
```

By default split () breaks up a text based on white spaces, but you can give a separator character also e.g, see sample code on the right above. Now consider the following program.

Program :- Program to count the frequency of a list-elements using a dictionary.

```
import json
sentence = "This is a super idea this/
idea will change the idea of learning"
words = sentence.split ( )
d = { }
```

```
Counting frequencies is list
['This', 'is', 'a', 'super'
'idea', 'This', 'idea', 'will', 'change', 'the', 'idea', 'of',
'learning']
{
```

```
for one in words :
  key = one
  if key not in d :
        count = words.count(key)
        d[key] = count
print ("Counting frequencies in list \n", words)
print (json.dumps(d, indent = 1))
```

```
 "This" : 2,
 "is" : 1,
 "a" : 1,
"Super" : 1,
"idea" : 3,
"will" : 1,
"change" : 1,
"the" : 1,
"of" : 1,
"learning" : 1
}
```

Quick Interesting Facts:

We can change the **value** of an item by accessing the **key** using square brackets ([]). To modify multiple values at once, we can use the **.update()** method, since this function overwrites existing **keys**.

PERIOD-04

Dictionary Functions and Methods

Dictionary Functions and Methods:-

Let us now talk about various built-in functions and methods provided by Python to manipulate Python dictionaries.

1. The len ( ) method

The method returns length of the dictionary, i.e, the count of elements (key : value pairs) in the dictionary. The syntax to use this method is given below:

len (<dictionary>)

- Takes dictionary name as argument and returns an integer.

Example:-

```
>>> employee = {'name':'John', 'salary' : 10000, 'age' : 24)
 >>> len (employee)
 3
 >>> employee ['dept'] = 'sales'
```

```
>>> len (employee)
4
```

2. The clear ( ) method

This method removes all items from the dictionary and the dictionary becomes empty dictionary post this method. the syntax to use this method is given below:

<dictionary>.clear()

- Takes no argument, returns no value

Example:-

```
>>> Employee = {'name' : 'John', 'salary' : 10000, 'age' : 24}
>>> Employee . clear ()
>>> Employee
{ }
```

The clear method does not delete the dictionary; it just deletes all the elements inside the dictionary. If, however, you want to delete the dictionary itself so that dictionary is also removed and no object remains, then you can use del statement along with dictionary name.

```
>>> Employee = {'name' : 'John', 'salary' : 10000, 'age' : 24}
>>> del Employee
>>> Employee
:
NameError : name 'Employee' is not defined
```

3. The get ( ) method

With this method, you can get the item with the given key, similar to dictionary [key]. If the key is not present, Python by default gives error, but you can specify your own message through default argument as per following syntax:

<dictionary> . get (key . [default])

Take key as essential argument and returns corresponding value if the key exists or when key does not exist, it returns Python's Error if default argument is missing otherwise returns default argument.

Example:

```
>>> empl1
{'salary' : 10000, 'dept' : 'sales', 'age' : 24, 'name' : 'John'}
>>> empl1.get ('dept')
'sales'
```

See it returned value for given key if key exists in the dictionary. If you only specify the key as argument, which does not exist in the dictionary, then Python will return error:

```
>>> empl1.get ('desig')
:
NameError: name'desig' is not defined
```

In place of error, you can also specify your own custom error message as second argument:

```
 >>> empl1. get ('desig', "Error ! ! key not found")
 'Error ! ! key not found'
```

4. The items ( ) method

This method returns all of the items in the dictionary as a sequence of (key, value) tuples. Note that these are returned in no particular order.

<dictionary> . items ( )

Takes no argument; Returns a sequence of (key, value) pairs.

Example :

```
 employee = {'name' : 'John', 'salary' : 10000, 'age' : 24}
 myList = employee . items (  )
 for x in myList :
   print (x)
```

The above code gives output as :

```
(  'salary', 10000)
('age',  24)
(  'name' , 'John')
```

As the items () function returns a sequence of (key value) pairs, you can write a loop having two variables to access key value pairs.

For example,,

```
empl = {'age' : 25, 'name' : 'Bhuvan', 'salary' : 20000}
 seq = empl.items (  )
 for ky, vl in seq :
    print (val, key)
```

The loop has two iteration variables ky, vI because the function items () returns a list of tuples and ky, vI is a tuple assignment that successively iterates through each of the key-value pairs in the dictionary i.e, each key-value pair is assigned to loop variables ky and vI at a time.

5. The keys () method

You have already worked with this method. This method returns all of the keys in the dictionary as a sequence of keys (in form of a list). Note that these are returned in no particular order.

<dictionary>. keys ()

Takes no argument; Returns a list sequence of keys.

Example:- >>> employee

{'salary' : 10000, 'dept' : 'sales', 'age' : 24, 'name' : 'John'}

>>> employee.keys ( )

['salary', 'dept', 'age', 'name'

6. The values () method

This method returns all the values from the dictionary as a sequence (a list). Note that these are returned in no particular order. The syntax to use this method is given below:

<dictionary>.values ()

Takes no argument; Returns a list sequence of values

Example:-

>>> employee

{'salary' : 10000, 'dept' : 'sales', 'age' : 24, 'name' : 'John'}

>>>employee.values ()

{10000, 'sales', 24, 'John']

7. The update () method

This 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. The syntax to use this method is given below:

<dictionary>. update (<other-dictionary>)

Example:

>>> employee1={'name' : 'John', 'salary' : 10000, 'age' : 24}

>>> employee2= {'name' : 'Diya', 'salary' : 54000, 'dept' : 'sales'}

>>> employee1. update (employee2)

>>> employee 1

{'salary' : 54000, 'dept' : 'sales', 'name' : 'Diya', 'age' : 24}

>>> employee2

{'salary' : 54000, 'dept' : 'sales', 'name' : 'Diya'}

This method is equivalent to the following Python statement:

```
for key in newDict. keys () :
    #newDict is employee2 here
   dictionary [k] = newDict[k]
```

07. Consider the following code fragments. What outputs will they produce?

(i) aDict = {'Bhavan' : 1, "Richard" : 2, "Firoza" : 10, "Arshnoor" : 20}

```
    temp = 0
     for value in aDict.values()  :
     temp = temp + value
     print (temp)
```

(ii)
```
aDict = {  'Bhavna' : 1, "Richard" : 2, "Firoza" : 10, "Arshnoor" : 20}
     temp = "  "
     for key in aDict:
          if temp < key :
            temp = key
        print(temp)
```

(iii)
```
aDict = {  'Bhavna' : 1,  "Richard" : 2, "Firoza" : 10, "Arshnoor" : 20}
     k = "Bhavna"
      v = -1
     if k in aDict :
       aDict [k] = v
      print (aDict)
```

Solution:-

(i) 33 (ii) Richard

(iii) { 'Bhavna' : -1, 'Richard' : 2, 'Firoza' = 10, 'Arshnoor' : 20}

08. The membership operators only work with keys of a dictionary but to check for the presence of a value in a dictionary, you need to write a code.

Write a Python program that takes a value and checks whether the given value is part of given dictionary or not. If it is, it should print the corresponding key otherwise print an error message.

Solution:-

```
dict1 = {0 : "Zero", 1 : "one", 2 : "Two", 3 : "Three", 4 : "Four", 5 : "Five"}
  ans = "y"
  while ans = = 'y' or ans == 'Y' :
  val = input ("enter value :")
  print ("Value", val, end = " ")
 for k in dict1 :
  if dict1[k] = = val :
  print ("exists at ", k)
else :
  print ("not found")
```

ans = input ("Want to check more values? (y/n) : ")

Marks of three students "Sunita", "Ryna" and "Zeba" in three subjects are available in following three dictionaries respectively:

d1 = {1 : 40, 2 : 70, 3 : 70}

d2 = {1 : 40, 2 : 50, 3 : 60}

d3 = {1 : 70, 2 : 80, 3 : 90}

9. Create a nested dictionary that stores the marks details along with student names and then prints the output as shown below:

Name

Ryna

subject (key)	Marks (value)
1	40
2	50
3	60

Name

Zeba

Subject (key)	Marks (value)
1	70
2	80
3	90

Name

Suiti

Subject (key)	Marks (value)
1	40
2	70
3.	70

Solution:-

```
d1 = {1 : 40, 2 : 70, 3 : 70}
d2 = {1 : 40, 2 : 50, 3 : 60}
d3 = {1 : 70, 2 : 80, 3 : 90}
d4 = {"Suniti" : d1, "ryna" : d2, "Zeba" : d3}
for x in d4. keys () :
    print ("Name")
    print (x)
```

```
print ("Subject(key)", '\t\, "Marks(value)")
for y in d4 [x].keys() :
    print (" ", y, '\t\t', d4[x][y])
print ()
```

10. Consider a dictionary my_points with single-letter keys, each followed by a 2-element tuple representing the coordinates of a point in an x-y coordinate plane.

    my_points = { 'a' : (4, 3), 'b' : (1, 2), 'c' : (5, 1) }

    Write a program to print the maximum value from within all of the values tuples at same index.

    For example, maximum for 0th index will be computed from values 4, 1 and 5, i.e all the entries at 0th index in the value-tuple. Print the result in following format:

    Maximum value at index (my _points, 0) = 5

    Maximum value at index (my_points, 1) = 3

Solution:-

```
my_points = {'a' : (4, 3), 'b' : (1, 2), 'c' :(5, 1)  }
highest = [0, 0]
init = 0
 for a in range (2) :
    init = 0
  for b in my_points.keys () :
     val = my_points [b] [a]
    if init = = 0 :
  highest [a] = val
  init + = 1
  if val > highest [a]
 highest [a] = val
  print ("Maximum value at index (my_points, ", a, ") = ", highest [a])
```

Glossary:-

Dictionary : A mutable, unordered collection with elements in the form of a key:value pairs that associate keys to value.

Mapping: Linking of a key with a value through some internal function (hash function).

Nesting : Having an element of similar type inside another element.

Lookup : A dictionary operation that takes a key and finds the corresponding value.

Assignments:-

Type A : Short Answer Questions/ Conceptual Questions:-

01. Why is a dictionary termed as an unordered collection of objects?
02. What type of objects can be used as keys in dictionaries?
03. Though tuples are immutable type, yet they cannot always be used as keys in a dictionary. What is a condition to use tuples as a key in a dictionary?
04. What all types of values can you store in : (a) dictionary - values? (b) dictionary-keys?
05. Can you change the order of dictionary's contents, i.e, can you sort the contents of a dictionary?
06. In no more than one sentence, explain the following Python error and how it could arise:
    TypeError : unhashable type ; 'list'
07. Can you check for a value inside a dictionary using in operator? How will you check for a value inside a dictionary using in operator?
08. Dictionary is a mutable type, which means you can modify its contents? What all is modifiable in a dictionary? Can you modify the keys of a dictionary?
09. How is del D and del D[<key>] different from one another if D is a dictionary?
10. How is clear () function different from del <del> statement?

Type B : Application Based Questions:

01. Which of the following will result in an error for a given valid dictionary D?
    (i) D + 3 (ii) D * 3 (iii) D + {3 : "3"}
    (iv) D, update ({3 : "3"}) (v) D. update { {"3" : 3}} (vi) D. update ("3" : 3
02. The following code is giving some error. Find out the error and correct it.
    d1 = {"a" : 1, 1 : "a", [1, "a"] : "two"}
03. The following code has two dictionaries with tuples as keys. While one of these dictionaries being successfully created, the other is giving some error. Find out which dictionary will be created successfully and which one will give error and correct it:
    dict1 = { (1, 2) : [1, 2], (3, 4) : [3, 4]}
    dict2 = { ([1], [2]) : [1, 2], ([3], [4] : [3, 4]
04. Nesting of dictionary allows you to store a dictionary inside another dictionary. Then why is following code raising error? What can you do to correct it?
    d1 = {1 : 10, 2 : 20, 3 : 30}
    d2 = {1 : 40, 2 : 50, 3 : 30}
    d3 = {1 : 70, 2 :80, 3 : 90}
    d4 = {d1 : "a", d2 : "b", d3 : "c"}
05. Why is following code not giving correct output even when 25 is a member of the dictionary?
    dic1 = {'age' : 25, 'name' : 'xyz', 'salary' : 23450.5}
    val = dic1 ['age']

```
if val in dic1 :
  print ("This is member of the dictionary")
else :
  print ("This is not a member of the dictionary")
```

06. What is the output produced by above code:

```
d1 = {5 : [6,  7,  8], "a" : (1, 2, 3) }
print (d1.keys () )
print (d1 . values() )
```

07. Consider the following code and then answer the questions that follow:

```
myDict = {'a': 27, 'b' : 43, 'c' : 25, 'd' : 30}
valA = ' '
for i in myDict :
  if i > valA
    valA = i
    valB = myDict[i]
print (valA)              #Line1
print (valB)              #Line2
print (30 in myDict)    #Line3
myLst = list (myDict.items())
myLst. sort ()           # Line4
print (myLst[-1])        # Lines5
```

(i) What output does Line 1 produce? (ii) What output does Line 2 produce?

(iii) What output does Line 3 produce? (iv) What output does Line 5 produce?

(v) What is the return value from the list sort () function (Line 4)?

08. What will be the output produced by following code?

```
 d1 = {5 : "number", "a" : "string", (1, 2) : "tuple" }
  print ("Dictionary contents")
for x in d1.keys () :            # Iterate on a key list
  print (x, ':', d1 [x], end = ' ')
print (d1[x]*3
print ( )
```

09. Predict the output

(i)
```
d = dict ()
   d['left'] = '<'
```

```
    d['right'] = '>'
   print ("{left} and {right} or {right} and {left}')
(ii) d = dict()
    d['left'] = '<'
   d['right'] = '>'
 d['end'] = ' '
  print (d['left'] and d['right'] or d['right'] and d['left'])
   print (d['left'] and d['right'] or d['right'] and d['left'] and d['end'])
  print (  (d['left'] and d['right'] or ['right'] and d['left']) and d['end'])
  print ("end")
(iii) text = "abracadabraaabbccrr"
     counts = {  }
   ct = 0
   1st = [ ]
  for word in text:
        if word not in 1st :
           1st.append (word)
           counts [word] = 0
         ct = ct = 1
          counts [word] = counts[word] +1
          print (counts)
           print (1st)
(iv) list 1 = [2, 3, 3, 2, 5, 3, 2, 5, 1, 1]
      counts = { }
       1st = [ ]
   for num in list1 :
 if num not in 1st :
   1st. append (num)
   counts[num] = 0
   ct = ct + 1
   counts [num] = counts [num] +1
  print (counts)
  for key in counts.keys () :
    counts [key] = key *counts[key]
```

```
print (counts)
```

10. Find the errors

(a)
```
text = "abracadbra"
counts = { }
for word in text :
counts [word] = counts [word] + 1
```

(b)
```
my_dict = {}
my_dict [(1, 2, 4] = 8
my_dict[  [4, 2, 1] ] = 10
print (my_dict)
```

11. Predict the output

(a)
```
fruit = { }
L1 = ['Apple', 'banana', 'apple']
for index in L1 :
if index in fruit :
fruit [index] + = 1
else :
fruit [index] =1
print (len(fruit))
print (fruit)
```

(b)
```
arr = { }
arr[1] = 1
arr['1'] = 2
arr[1] + = 1
sum = 0
for k in arr:
sum + = arr [k]
print (sum)
```

12. Predict the output

(a)
```
a = {(1, 2) : 1, (2, 3) : 2}
print (a[1, 2])
```

(b)
```
a = {'a' : 1, 'b' : 2, 'c' : 3}
print (a['a', 'b'])
```

13. Find the error/ output. Consider below given two sets of codes. Which one will produce an error? Also, predict the output produced by the correct code.

(a) box = { }
jars = {'Jam' : 4}
crates = { }
box['biscuit'] = 1
box ['cake'] = 3
crates ['box'] = box
crates ['jars'] = jars
print (lan(crates[box]))

(b) box = { }
jars = {'jam' : 4}
crates = { }
box ['biscuit'] = 1
box['cake']=3
crates['box']=box
crates['jars'] = jars
print (len(crates['box']))

Type C : Programming Practice/ Knowledge based Questions:-

01. Repeatedly ask the user to enter a team name and how many games the team has won and how many they lost. Store this information in a dictionary where the keys are the team names and the values are lists of the form (wins, losses).

(a) Using the dictionary created above, allow the user to enter a team name and print out the team's winning percentage.

(b) Using the dictionary, create a list whose entries are the number of wins of each team.

(c) Using the dictionary, create a list of all those teams that have winning records.

02. Write a program that repeatedly asks the user to enter product names and prices. Store all of these in a dictionary whose keys are the product names and whose values are the prices. When the user is done entering products and prices, allow them to repeatedly enter a product name and print the corresponding price or a message if the product is not in the dictionary.

03. Create a dictionary whose keys are month names and whose values are the number of days in the corresponding months.

(a) Ask the user to enter a month name and use the dictionary to tell how many days are in the month.

(b) Print out all of the keys in alphabetical order.

(c) Print out all of the months with 31 days

(d) Print out the (key-value) pairs sorted by the number of days in each month.

04. Can your store the details of 10 students in a dictionary at the same time? Details include -rollno, name, marks, grade etc. Give example to support your answer.
05. Give the dictionary x = {'k1' :'v1', 'k2' :'v2', 'k3' : 'v3'}, create a dictionary with the opposite mapping, i.e write a program to create the dictionary as: inverted _x= {'v1' : 'k1', 'v2' : 'k2', 'v3' :'k3'}
06. Given two dictionaries say D1 and D2. Write a program that lists the overlapping keys of the two dictionaries, i.e if a key of D1 is also a key ofD2, the list it.
07. Write a program that checks if two same values in a dictionary have different keys. That is, for dictionary D1={'a' : 10, 'b' : 20, 'c' :'10'}, the program should print "2 keys have same values" and for dictionary D2 = {'a' : 10, 'b' : 20, 'c' : 30} the program should print, "No keys have same values".

Quick Interesting Facts:

Python **for-loops** are very handy in dealing with repetitive programming tasks; however, there is another alternative to achieve the same results in a more efficient way: **dictionary comprehensions.**

Dictionary comprehensions allow the creation of a dictionary using an elegant and simple syntax: **{key: value for vars in iterable}.** In addition, they are faster than traditional **for-loops**.

****