CHAPTER

11

DICTIONARIES

A dictionary represents a group of elements arranged in the form of key-value pairs. In the dictionary, the first element is considered as 'key' and the immediate next element is taken as its 'value'. The key and its value are separated by a colon (:). All the key-value pairs in a dictionary are inserted in curly braces { }. Let's take a dictionary by the name 'dict' that contains employee details:

```
dict = {'Name': 'Chandra', 'Id': 200, 'Salary': 9080.50}
```

Here, the name of the dictionary is 'dict'. The first element in the dictionary is a string 'Name'. So, this is called 'key'. The second element is 'Chandra' which is taken as its 'value'. Observe that the key and its value are separated by a colon. Similarly, the next element is 'Id' which becomes 'key' and the next element '200' becomes its value. Finally, 'Salary' becomes key and '9080.50' becomes its value. So, we have 3 pairs of keys and values in this dictionary. This is shown in Figure 11.1:

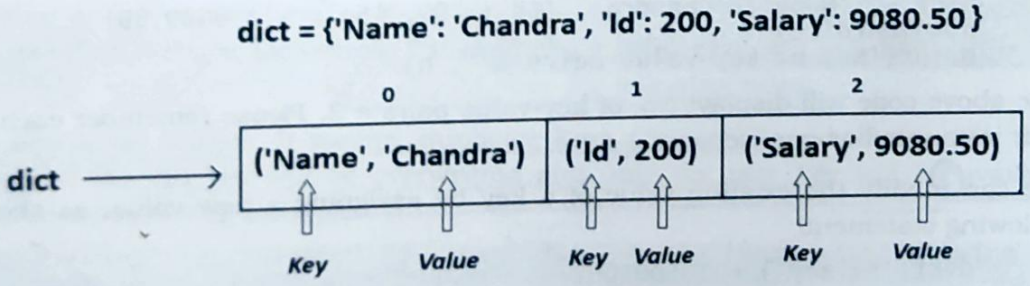

Figure 11.1: A Dictionary with 3 Key-Value Pairs

When the 'key' is provided, we can get back its 'value'. This is how we search for the values in a dictionary. For example, 'Name' is the key. To get its value, i.e. 'Chandra', we should mention the key as an index to the dictionary, as: dict['Name']. This will return the value 'Chandra'. Similarly, dict['Id'] returns its value, i.e. 200. See Program 1.

Program

Program 1: A Python program to create a dictionary with employee details and retrieve the values upon giving the keys.

```
# creating dictionary with key- value pairs
"""
Create a dictionary with employee details.
Here 'Name' is key and 'Chandra' is its value.
'Id' is key and 200 is its value.
'Salary' is key and 9080.50 is its value.
"""

dict = {'Name': 'Chandra', 'Id': 200, 'Salary': 9080.50}

# access value by giving key
print('Name of employee= ', dict['Name'])
print('Id number= ', dict['Id'])
print('Salary= ', dict['Salary'])
```

Output:

```
C:\>python dict.py
Name of employee= Chandra
Id number= 200
Salary= 9080.5
```

Operations on Dictionaries

To access the elements of a dictionary, we should not use indexing or slicing. For example, dict[0] or dict[1:3] etc. expressions will give error. To access the value associated with a key, we can mention the key name inside the square braces, as: dict['Name']. This will return the value associated with 'Name'. This is nothing but 'Chandra'.

If we want to know how many key-value pairs are there in a dictionary, we can use the len() function, as shown in the following statements:

```
dict = {'Name': 'Chandra', 'Id': 200, 'Salary': 9080.50}
n = len(dict)
print('No. of key-value pairs = ', n)
```

The above code will display: No. of key-value pairs = 3. Please remember each key-value pair is counted as one element.

We can modify the existing value of a key by assigning a new value, as shown in the following statement:

```
dict['Salary'] = 10500.00
```

Here, the 'Salary' value is modified as '10500.00'. The previous value of 'Salary', i.e. 9080.50 is replaced by the new value, i.e. 10500.00.

We can also insert a new key-value pair into an existing dictionary. This is done by mentioning the key and assigning a value to it, as shown in the following statement:

```
dict['Dept'] = 'Finance'
```

Here, we are giving a new key 'Dept' and its value 'Finance'. This pair is stored into the dictionary 'dict'. Now, if we display the dictionary using print(dict), it will display:

```
{'Name': 'Chandra', 'Dept': 'Finance', 'Id': 200, 'Salary': 10500.0}
```

Observe the new pair 'Dept': 'Finance' is added to the dictionary. Also, observe that this pair is not added at the end of existing pairs. It may be added at any place in the dictionary.

Suppose, we want to delete a key-value pair from the dictionary, we can use del statement as:

```
del dict['Id']
```

This will delete the key 'Id' and its corresponding value from the dictionary. Now, the dictionary looks like this:

```
{'Name': 'Chandra', 'Dept': 'Finance', 'Salary': 10500.0}
```

To test whether a 'key' is available in a dictionary or not, we can use 'in' and 'not in' operators. These operators return either True or False. Consider the following statement:

```
'Dept' in dict   # check if 'Dept is a key in dict
```

The preceding statement will give:

```
True
```

Now, consider the following statement:

```
'Gender' in dict   # check if 'Gender' is a key in dict
```

The preceding statement will give:

```
False
```

Now, if you write:

```
'Gender' not in dict   # check if 'Gender' is not a key in dict
```

Then the following output appears:

```
True
```

We can use any datatypes for values. For example, a value can be a number, string, list, tuple or another dictionary. But keys should obey the following rules:

- Keys should be unique. It means, duplicate keys are not allowed. If we enter same key again, the old key will be overwritten and only the new key will be available. Consider the following example:

```
emp = {'Nag':10, 'Vishnu':20, 'Nag':30}   # Key 'Nag' entered twice
print(emp)
```

The output appears as:

```
{'Nag': 30, 'Vishnu': 20} # first 'Nag' is replaced by new key and its
# value
```

- Keys should be immutable type. For example, we can use a number, string or tuples as keys since they are immutable. We cannot use lists or dictionaries as keys. If they are used as keys, we will get 'TypeError'. Consider the following example:

```
emp = {['Nag']:10, 'Vishnu':20, 'Raj':30}   # ['Nag'] is a list element
    - so error
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    emp = {['Nag']:10, 'Vishnu':20, 'Raj':30}
TypeError: unhashable type: 'list'
```

Dictionary Methods

Home Work...

Various methods are provided to process the elements of a dictionary. These methods generally retrieve or manipulate the contents of a dictionary. They are summarized in Table 11.1:

Table 11.1: Methods to Process Dictionaries

Method	Example	Description
clear()	d.clear()	Removes all key-value pairs from dictionary 'd'.
copy()	d1= d.copy()	Copies all elements from 'd' into a new dictionary 'd1'.
fromkeys()	d.fromkeys(s [,v])	Create a new dictionary with keys from sequence 's' and values all set to 'v'.
get()	d.get(k [,v])	Returns the value associated with key 'k'. If key is not found, it returns 'v'.
items()	d.items()	Returns an object that contains key-value pairs of 'd'. The pairs are stored as tuples in the object.
keys()	d.keys()	Returns a sequence of keys from the dictionary 'd'.
values()	d.values()	Returns a sequence of values from the dictionary 'd'.
update()	d.update(x)	Adds all elements from dictionary 'x' to 'd'.
pop()	d.pop(k [,v])	Removes the key 'k' and its value from 'd' and returns the value. If key is not found, then the value 'v' is returned. If key is not found and 'v' is not mentioned then 'KeyError' is raised.
setdefault()	d.setdefault(k [,v])	If key 'k' is found, its value is returned. If key is not found, then the k, v pair is stored into the dictionary 'd'.

In Program 2, we are going to retrieve keys from a dictionary using the keys() method. The keys() method returns dict_keys object that contains only keys. We will also retrieve values from the dictionary using values() method. This method returns all values in the form of dict_values object. Similarly, the items() method can be used to retrieve all key-value pairs into dict_items object.

Program

Program 2: A Python program to retrieve keys, values and key-value pairs from a dictionary.

```
# dictionary methods
# create a dictionary with employee details.
dict = {'Name': 'Chandra', 'Id': 200, 'Salary': 9080.50}

# print entire dictionary
print(dict)

# display only keys
print('Keys in dict= ', dict.keys())

# display only values
print('Values in dict= ', dict.values())

# display both key and value pairs as tuples
print('Items in dict= ', dict.items())
```

Output:

```
C:\>python dict.py
{'Name': 'Chandra', 'Id': 200, 'Salary': 9080.5}
Keys in dict=  dict_keys(['Name', 'Id', 'Salary'])
Values in dict=  dict_values(['Chandra', 200, 9080.5])
Items in dict=  dict_items([('Name', 'Chandra'), ('Id', 200),
    ('Salary', 9080.5)])
```

In Program 3, we are going to create a dictionary by entering the elements from the keyboard. When we enter the elements from the keyboard inside curly braces, then they are treated as key – value pairs of a dictionary by eval() function. Once the elements are entered, we want to find sum of the values using sum() function on the values of the dictionary.

Program

Program 3: A Python program to create a dictionary and find the sum of values.

```
# program to find sum of values in a dictionary
# enter the dictionary entries from keyboard
dict = eval(input("Enter elements in { }: "))

# find the sum of values
s = sum(dict.values())
print('Sum of values in the dictionary: ', s)  # display sum
```

Output:

```
C:\>python dict.py
Enter elements in { }: {'A':10, 'B':20, 'C':35, 'Anil': 50}
Sum of values in the dictionary: 115
```

In Program 4, first we create an empty dictionary 'x'. We enter the key into 'k' and value into 'v' and then using the update() method, we will store these key-value pairs into the dictionary 'x', as shown in the following statement:

```
x.update({k:v})
```

Here, the update() method stores the 'k' and 'v' pair into the dictionary 'x'.

Program

Program 4: A Python program to create a dictionary from keyboard and display its elements.

```
# creating a dictionary from the keyboard
x = {}  # take an empty dictionary

print('How many elements? ', end='')
n = int(input()) # n indicates no. of key-value pairs

for i in range(n):  # repeat for n times
    print('Enter key: ', end='')
    k = input()  #  key is string
    print('Enter its value: ', end='')
    v = int(input())  # value is integer
    x.update({k:v})  # store the key-value pair in dictionary x

# display the dictionary
print('The dictionary is: ', x)
```

Output:

```
C:\>python dict.py
How many elements? 3
Enter key: Raju
Enter its value: 10
Enter key: Laxmi
Enter its value: 22
Enter key: Salman
Enter its value: 33
The dictionary is: {'Laxmi': 22, 'Raju': 10, 'Salman': 33}
```

Please observe the output of Program 4. The key-value pairs which are entered by us from the keyboard are not displayed in the same order. Dictionaries will not maintain orderliness of pairs.

In Program 5, we are creating a dictionary with cricket players' names and scores. That means, player name becomes key and the score becomes its value. Once the dictionary 'x' is created, we can display the players' names by displaying the keys as:

```
for pname in x.keys():  # keys() will give players names
    print(pname)
```

To find the score of a player, we can use get() method, as:

```
runs = x.get(name, -1)
```

In the get() method, we should provide the key, i.e. player name. If the key is found in the dictionary, this method returns his 'runs'. If the player is not found in the dictionary, then it returns -1.

Program

Program 5: A Python program to create a dictionary with cricket players names and scores in a match. Also we are retrieving runs by entering the player's name.

```
# creating a dictionary with cricket players names and scores
x = {}  # take an empty dictionary

print('How many players? ', end='')
n = int(input()) # n indicates no. of key-value pairs

for i in range(n):  # repeat for n times
    print('Enter player name: ', end='')
    k = input()  #  key is string
    print('Enter runs: ', end='')
    v = int(input())  # value is integer
    x.update({k:v})  # store the key-value pair in dictionary x

# display only players names
print('\nPlayers in this match: ')
for pname in x.keys():  # keys() will give only keys
    print(pname)

# accept a player name from keyboard
print('Enter player name: ', end='')
name = input()

# find the runs done by the player
runs = x.get(name, -1)
if(runs == -1):
    print('Player not found')
else:
    print('{} made runs {}.'.format(name, runs))
```

Output:

```
C:\>python dict.py
How many players? 3
Enter player name: Sachin
Enter runs: 77
Enter player name: Kohli
Enter runs: 40
Enter player name: Sehwag
Enter runs: 89
Players in this match:
Kohli
Sachin
Sehwag
Enter player name: Kohli
Kohli made runs 40.
```

Using for Loop with Dictionaries

For loop is very convenient to retrieve the elements of a dictionary. Let's take a simple dictionary that contains color code and its name as:

```
colors = {'r': "Red", 'g': "Green", 'b': "Blue", 'w': "White"}
```

Here, 'r', 'g', 'w' represent keys and "Red", "Green", "White" indicate values. Suppose, we want to retrieve only keys from 'colors' dictionary, we can use a for loop as:

```
for k in colors:
    print (k)
```

In the above loop, 'k' stores each element of the colors dictionary. Here, 'k' assumes only keys and hence this loop displays only keys. Suppose, we want to retrieve values, then we can obtain them by passing the key to colors dictionary, as: colors[k]. The following for loop retrieves all the values from the colors dictionary:

```
for k in colors:
    print (colors[k])
```

Since values are associated with keys, we can retrieve them only when we mention the keys. Suppose, we want to retrieve both the keys and values, we can use the items() method in for loop as:

```
for k, v in colors.items():
    print('Key= {} Value= {}'. format(k, v))
```

In the preceding code, the colors.items() method returns an object by the name 'dict_items' that contains key and value pairs. Each of these pairs is stored into 'k', 'v' and then displayed. Consider Program 6.

Program

Program 6: A Python program to show the usage of for loop to retrieve elements of dictionaries.

```
# Using for loop with dictionaries
# take a dictionary
colors = {'r': "Red", 'g': "Green", 'b': "Blue", 'w': "White"}

# display only keys
for k in colors:
    print (k)

# pass keys to dictionary and display the values
for k in colors:
    print (colors[k])

# items() method returns key and value pair into k, v
for k, v in colors.items():
    print('Key= {} Value= {}'. format(k, v))
```

Output:

```
C:\>python dict.py
b
w
r
g
Blue
White
Red
Green
Key= b Value= Blue
```

```
# sort the dictionary by keys, i.e. 0th element
c1 = sorted(colors.items(), key = lambda t: t[0])
print(c1)

# sort the dictionary by values, i.e. 1st element
c2 = sorted(colors.items(), key = lambda t: t[1])
print(c2)
```

Output:

```
C:\>python dict.py
[(10, 'Red'), (15, 'Blue'), (25, 'White'), (35, 'Green')]
[(15, 'Blue'), (35, 'Green'), (10, 'Red'), (25, 'White')]
```

Converting Lists into Dictionary

When we have two lists, it is possible to convert them into a dictionary. For example, we have two lists containing names of countries and names of their capital cities.

```
countries = ["USA", "India", "Germany", "France"]
cities = ['Washington', 'New Delhi', 'Berlin', 'Paris']
```

We want to create a dictionary out of these two lists by taking the elements of 'countries' list as keys and of 'cities' list as values. The dictionary should look something like this:

```
d = {"USA" : 'Washington', "India" : 'New Delhi' , "Germany" :
     'Berlin', "France" : 'Paris'}
```

There are two steps involved to convert the lists into a dictionary. The first step is to create a 'zip' class object by passing the two lists to zip() function as:

```
z = zip(countries, cities)
```

The zip() function is useful to convert the sequences into a zip class object. There may be 1 or more sequences that can be passed to zip() function. Of course, we passed only 2 lists to zip() function in the above statement. The resultant zip object is 'z'.

The second step is to convert the zip object into a dictionary by using dict() function.

```
d = dict(z)
```

Here, the 0th element of z is taken as 'key' and 1st element is converted into its 'value'. Similarly, 2nd element becomes 'key' and 3rd one becomes its 'value', etc. They are stored into the dictionary 'd'. If we display 'd', we can see the following dictionary:

```
{'India': 'New Delhi', 'USA': 'Washington', 'Germany': 'Berlin',
     'France': 'Paris'}
```

Program

Program 9: A Python program to convert the elements of two lists into key-value pairs of a dictionary.

```
# converting lists into a dictionary
# take two separate lists with elements
countries = ["USA", "India", "Germany", "France"]
cities = ['Washington', 'New Delhi', 'Berlin', 'Paris']
```

```
# make a dictionary
z = zip(countries, cities)
d = dict(z)

# display key - value pairs from dictionary d
print('{:15s} -- {:15s}'.format('COUNTRY', 'CAPITAL'))
for k in d:
    print('{:15s} -- {:15s}'.format(k, d[k]))
```

Output:

```
C:\>python dict.py
COUNTRY         -- CAPITAL
India           -- New Delhi
USA             -- Washington
Germany         -- Berlin
France          -- Paris
```

Converting Strings into Dictionary

When a string is given with key and value pairs separated by some delimiter (or separator) like a comma (,) we can convert the string into a dictionary and use it as dictionary. Let's take an example string:

```
str = "Vijay=23,Ganesh=20,Lakshmi=19,Nikhil=22"
```

This string 'str' contains names and their ages. Each pair is separated by a comma (,). Also, each name and age are separated by equals (=) symbol. To convert such a string into a dictionary, we have to follow 3 steps. First, we should split the string into pieces where a comma is found using split() method and then brake the string at equals (=) symbol. This can be done using a for loop as:

```
for x in str.split(','):
    y= x.split('=')
```

Each piece of the string is available in 'y'. The second step is to store these pieces into a list 'lst' using append() method as:

```
lst.append(y)
```

The third step is to convert the list into a dictionary 'd' using dict() function as:

```
d = dict(lst)
```

Now, this dictionary 'd' contains the elements as:

```
{'Vijay': '23', 'Ganesh': '20', 'Lakshmi': '19', 'Nikhil': '22'}
```

Please observe that this dictionary contains all elements as strings only. See first pair: 'Vijay': '23'. Here, 'Vijay' is string and his age '23' is also stored as string. If we want we can convert this '23' into an integer using int() function. Then we can store the name and age into another dictionary 'd1' as:

```
for k, v in d.items():
    d1[k] = int(v)   # store k and int(v) as key-value pair into d1.
```

Here, k represents the key and int(v) represents the converted value being stored into d1. This logic is shown in Program 10.

Program

Program 10: A Python program to convert a string into key-value pairs and store them into a dictionary.

```
# converting a string into a dictionary
# take a string
str = "Vijay=23,Ganesh=20,Lakshmi=19,Nikhil=22"

# brake the string at ',' and then at '='
# store the pieces into a list lst
lst=[]
for x in str.split(','):
    y= x.split('=')
    lst.append(y)

# convert the list into dictionary 'd'
# but this 'd' will have both name and age as strings
d = dict(lst)

# create a new dictionary 'd1' with name as string
# and age as integer
d1={}
for k, v in d.items():
    d1[k] = int(v)

# display the final dictionary
print(d1)
```

Output:

```
C:\>python dict.py
{'Ganesh': 20, 'Vijay': 23, 'Lakshmi': 19, 'Nikhil': 22}
```

Passing Dictionaries to Functions

We can pass a dictionary to a function by passing the name of the dictionary. Let's define a function that accepts a dictionary as a parameter.

```
def fun(dictionary):
    for i, j in dictionary.items():
        print(i, '--', j)
```

This function fun() is taking 'dictionary' object as parameter. Using for loop, we are displaying the key – value pairs of the dictionary. To call this function and pass a dictionary 'd', we can simply write:

```
fun(d)
```

Program

Program 11: A Python function to accept a dictionary and display its elements.

```
# A function that takes a dictionary as parameter
def fun(dictionary):
    for i, j in dictionary.items():
        print(i, '--', j)
```

```
# take a dictionary
d = {'a':'Apple', 'b': 'Book', 'c': 'Cook'}

# call the function and pass the dictionary
fun(d)
```

Output:

```
C:\>python dict.py
b -- Book
a -- Apple
c -- Cook
```