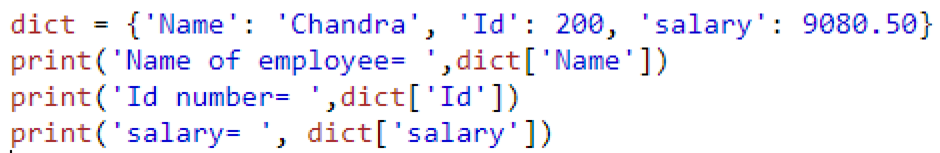
Dictionaries in Python

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 dictionary by the name 'dict' that contains employee details:

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

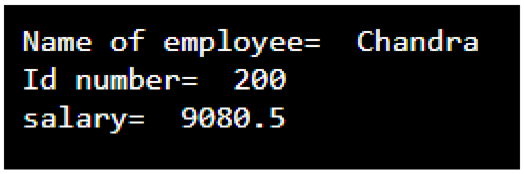
Here, the name of the dictionary is 'dict'. The first element in the dictionary is a string ‘Name'. So, this is called 'key'. The second element is 'Chandra' which is taken as its value. Observe that the key and its value are separated by a colon. Similarly, the next element is 'Id' which becomes 'key' 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.

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.
Ex:



```
dict = {'Name': 'Chandra', 'Id': 200, 'salary': 9080.50}
print('Name of employee= ',dict['Name'])
print('Id number= ',dict['Id'])
print('salary= ', dict['salary'])
```

Output:



```
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 an 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 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 anywhere 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”: 10500.03}

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

The preceding statement will give

True

Now, consider the following statement:
‘Gender’ in dict
The preceding statement will give:

False

Now, if you write
"Gender" not 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 the 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}
  print(emp).

  The output appears as:
  {‘Nag’: 30, ‘vishnu': 20}

- Keys should be an immutable type. For example, we can use numbers, strings 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)
  -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:

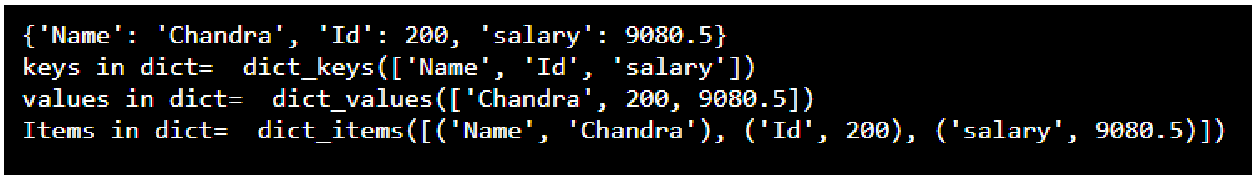
Various methods are provided to process the elements of a dictionary. These methods generally retrieve or manipulate the contents of a dictionary.

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 the 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 the 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])	then KeyError' is raised.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 the program below, we are going to retrieve keys from a dictionary using the keys() method The keys() method returns a dict_keys object that contains only keys. We will also retrieve values from the dictionary using the 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 objects.
Ex:

```
dict={'Name': 'Chandra', 'Id': 200, 'salary': 9080.50}
print(dict)
print('keys in dict= ',dict.keys())
print('values in dict= ', dict.values())
print('Items in dict= ',dict. items())
```

Output:



```
{'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)])
```

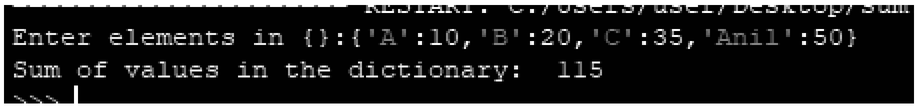
.

In the program below, 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 the eval() function. Once the elements are entered, we want to find the sum of the values using the sum() function on the values of the dictionary.

Ex:

```
dict=eval(input("Enter elements in {}:"))
s=sum(dict.values())
print('Sum of values in the dictionary: ',s)
```

Output:



```
====================== RESTART: C:/Users/user/Desktop/sum
Enter elements in {}:{'A':10,'B':20,'C':35,'Anil':50}
Sum of values in the dictionary:  115
>>> 
```

In the program below, we first 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:

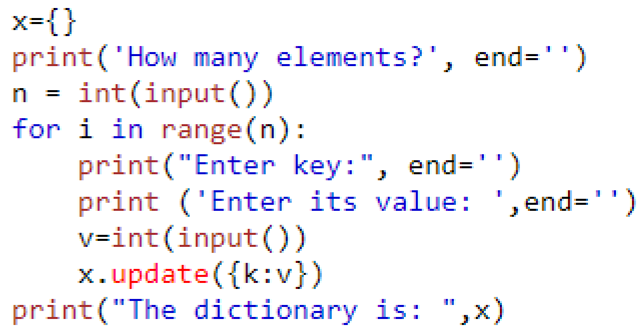
x.update({k:v})

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

Prograin

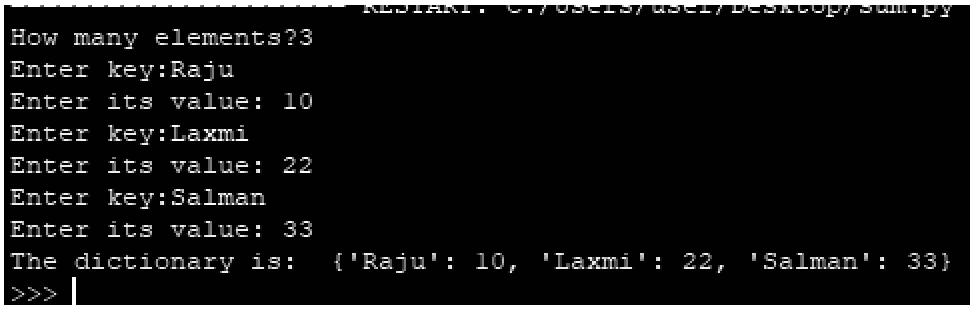
Program 4: A Python program to create a dictionary from keyboard and display the elements.

#creating a dictionary from the keyboard



```
x={}
print('How many elements?', end='')
n = int(input())
for i in range(n):
    print("Enter key:", end='')
    print ('Enter its value: ',end='')
    v=int(input())
    x.update({k:v})
print("The dictionary is: ",x)
```

Output:



```
======================== RESTART: C:/Users/user/Desktop/sum.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:  {'Raju': 10, 'Laxmi': 22, 'Salman': 33}
>>> 
```

Please observe the output of the above program. 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 the following program, we are creating a dictionary with cricket players' names and scores. That means, the 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():
print (pname)

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

runs x.get(name, -1)

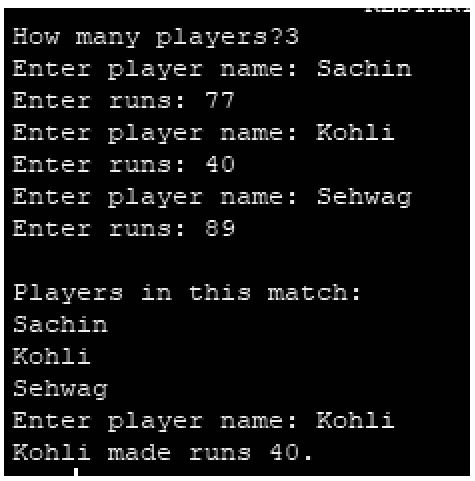
In the get() method, we should provide the key, i.e. player name. If the key is found in the dictionary, this method returns his runs'. If the player is not found in the dictionary, then it returns -1.

```
x={}
print('How many players?', end='')
n = int(input())
for i in range(n):
    print('Enter player name: ',end='')
    k=input()
    print ('Enter runs: ',end='')
    v=int(input())
    x.update({k:v})
print('\nPlayers in this match: ')
for pname in x keys:
    print (pname)

print('Enter player name: ', end='')
name=input()

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

Output:



```
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:
Sachin
Kohli
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)
```

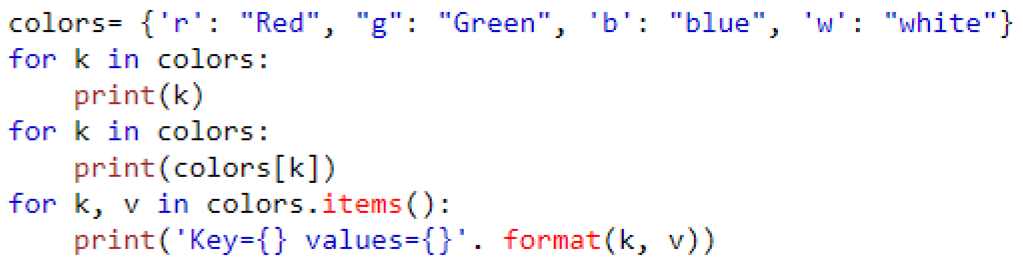
In the above loop, ‘k' stores each element of the colors dictionary. Here, 'k’ assumes only keys and hence this loop displays only keys. Suppose, we want to retrieve values, then we can obtain them by passing the key to the colors dictionary, as: color[k]. The following for loop retrieves all the values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

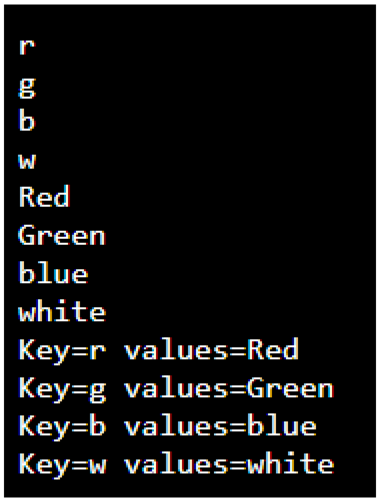
```
for k,v in colors.items:
print('Key={} value={}’ . format(k, v))
```

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.



```
colors= {'r': "Red", "g": "Green", 'b': "blue", 'w': "white"}
for k in colors:
    print(k)
for k in colors:
    print(colors[k])
for k, v in colors.items():
    print('Key={} values={}'. format(k, v))
```

Output:



```
r
g
b
w
Red
Green
blue
white
Key=r values=Red
Key=g values=Green
Key=b values=blue
Key=w values=white
```

We will write another program to count the number of times each letter has occurred in a string For example, "Book" is the string and we are supposed to find the number of occurrences of each letter in this string. It means the letter 'B' has occurred for 1 time, the letter ‘o' occurred for 2 times and the letter ‘k' occurred for 1 time. In this program, we use the get() method very effectively. Please recollect that the get() method is used on a dictionary to retrieve the value by giving the key. If the key is not found in the dictionary. then it returns some default value. The format of the get() method is:

dict.get(x, 0)

This statement says that if the key 'x' is found in the dictionary 'dict', then return its ‘value' from the dictionary, else return 0. Now, consider the following code:

```
dict={}
str=”Book”
for x in str:
dict[x] = dict.get(x,0)+1
```

In the preceding code, the last statement is very important.

dict [x] dict.get(x, 0)+1
Observe the right hand side expression with the get() method. It says if 'x' (this is the letter of the string) is found in the dictionary 'dict', then return its value, else return 0, But we added '1' to the value returned by get() method and hence, if 'x' is not found, it returns 1. If ‘x' is found then it returns the value of ‘x' plus 1.

Observe the left side expression, i.e. dict[x].This represents ‘x' is stored as a key in the dictionary. Whatever the value returned by the right side expression will be stored into that dictionary as a value for the key 'x'. That means:

dict[x]=value returned by get()+1

Let's take the first letter of the string, i.e. 'B'. Since the dictionary is initially empty, there are no elements in it and hence 'B' is not found in the dictionary. So, dict.get(x, 0)+1 returns 1. On the left side, we have dict(x). It represents dict['B’]. Here, 'B' is taken as key. So, the statement becomes:
dict [‘B’]=1

It means B' is stored as a key and 1 is stored as its value into the dictionary 'dict'. So the dictionary contains a pair of elements as: {‘B’:1}.In the next step, 'o' is the letter for which the get() method searches in the dictionary. It is not found in the 'dict' and hence 1 is returned. So,

dict['o']=1

This will store the new key-value pair, i.e. 'o' and 1 into the dict' and hence the dictionary contains:

{"B": 1, ‘o’: 1}.

In the next repetition of the for loop, we get 'o' into x. Since it is already available in the dict', its value 1 is returned by the get() method for which I will be added. So, we get: dict['0'] = 2 It means, the old value of 'o' is now updated to 2 in the dictionary and 'dict' contains the elements: {‘B’: 1, ‘o’: 2}. In this way, 'dict' stores each letter as a key and its number of occurrences as value.

x	Found in dict	dict.get(x, 0)+1	dict[x]= dict.get(x, 0)+1	dict
B	No	1	dict[‘B’]=1	{‘B’:1}
o	No	1	dict[‘o’]=2	{‘B’:1,‘o’:1}
o	Yes	2	dict[‘o’]=2	{‘B’:1,‘o’:2}
k	No	1	dict[‘k’]=1	{‘B’:1,‘o’:1,’k’:1}

```
str="Book"
dict = {}
for x in str:
    dict [x] = dict.get(x, 0)+1
for k,v in dict.items():
    print('Key={}\t Its occurrences= {}'.format(k,v))
```

Output:

```
Key=B    Its occurrences= 1
Key=o    Its occurrences= 2
Key=k    Its occurrences= 1
```

The output of the above program may not show the letter occurrences in an orderly manner. This is because the dictionary does not store the elements in the same order as they were entered.

Sorting the Elements of a Dictionary using Lambdas

A lambda is a function that does not have a name. Lambda functions are written using a single statement and hence look like expressions. Lambda functions are written without using the def keyword. They are useful to perform calculations or processing easily.
For example,

f=lambda x, y: x+y

The above expression is a lambda function with 2 arguments, x and y. After colon (:), we wrote the body, L.e. x+y. This is the value returned by the lambda function. At the time of calling this function, we are supposed to pass 2 values for x and y as: f(10, 15). This will return 25 as result.

Let's take an example dictionary as:

colors={10: "Red", 35: "Green", 15: "Blue", 25: "white"}

Here, color code and its name are given as key-value pairs. Suppose we want to sort this dictionary into ascending order of keys, i.e on color codes, we can use sorted() function in the following format:

sorted(elements, key=color code)

Here, elements of the dictionary can be accessed using the colors.items() method. A key can be prescribed using a lambda function as:

key=lambda t: t[0]

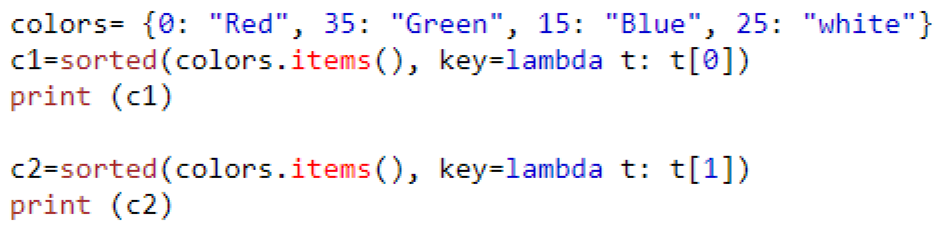
Here, 't' is the argument for the lambda function and t[0] is the value returned by the function. Since we are supposed to sort the dictionary, we should pass the entire dictionary to the lambda function. So, 't' represents the dictionary that is passed to the function and t[0] represents the 0th element in the dictionary, i.e. color code. So, the sorted() function can be written as:

sorted(colors.items(), key=lambda t: t[0])

This will sort all the elements of the dictionary by taking color code (indicated by t[0]) as the key. If we want to sort the dictionary based on color name, then we can write:

sorted(colors.items(), key=lambda t: t[1])

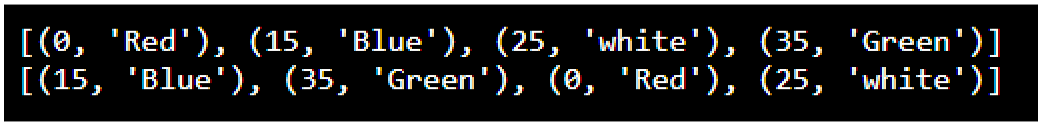
This will sort the elements of the dictionary by taking the color name (indicated by t[1]) as the key.



```
colors= {0: "Red", 35: "Green", 15: "Blue", 25: "white"}
c1=sorted(colors.items(), key=lambda t: t[0])
print (c1)

c2=sorted(colors.items(), key=lambda t: t[1])
print (c2)
```

Output:



```
[(0, 'Red'), (15, 'Blue'), (25, 'white'), (35, 'Green')]
[(15, 'Blue'), (35, 'Green'), (0, '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'}

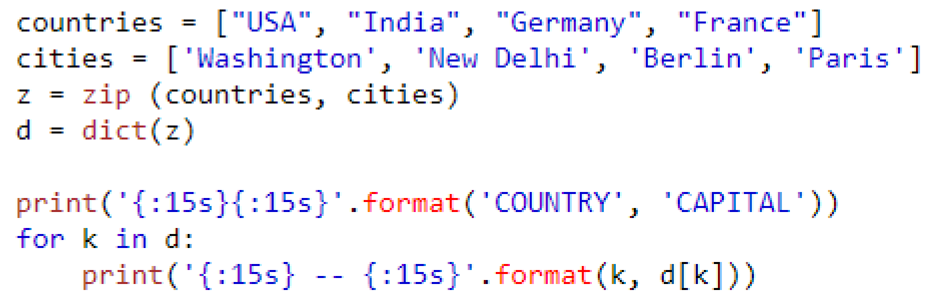
There are two steps involved to convert the lists into a dictionary. The first step is to create a ‘zip' class object by passing the two lists to zip() function as:
z = zip(countries, cities)
The zip() function is useful to convert the sequences into a zip class object. There may be 1 or more sequences that can be passed to the zip() function. Of course, we passed only 2 lists to the 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 the 1st element is converted into its value. Similarly, the 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:

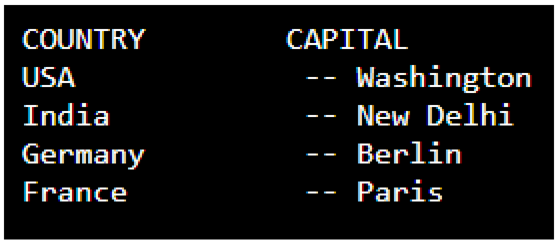
{"India”: ‘New Delhi’, “USA”: ‘washington’, “Germany”: 'Berlin', "France': 'Paris’}



```
countries = ["USA", "India", "Germany", "France"]
cities = ['Washington', 'New Delhi', 'Berlin', 'Paris']
z = zip (countries, cities)
d = dict(z)

print('{:15s}{:15s}'.format('COUNTRY', 'CAPITAL'))
for k in d:
    print('{:15s} -- {:15s}'.format(k, d[k]))
```

Output:



```
COUNTRY        CAPITAL
USA             -- Washington
India           -- New Delhi
Germany         -- Berlin
France          -- Paris
```

Converting Strings into Dictionary

When a string is given with key and value pairs separated by some delimiter (or separatori like a comma (,) we can convert the string into a dictionary and use it as a 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 equal (=) 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 break 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()
```

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

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

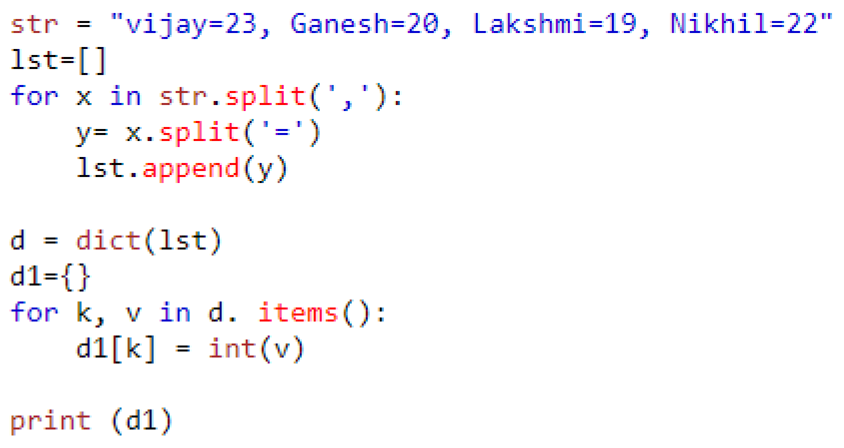
Now, this dictionary 'd' contains the elements as:

```
['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 a string and his age '23' is also stored as a string. If we want we can convert this '23' into an integer using the 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)
```

Here, k represents the key and int(v) represents the converted value being stored into d1.

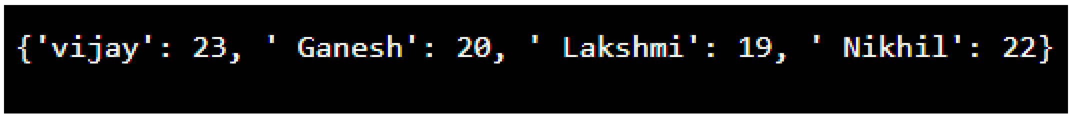


```python
str = "vijay=23, Ganesh=20, Lakshmi=19, Nikhil=22"
lst=[]
for x in str.split(','):
    y= x.split('=')
    lst.append(y)

d = dict(lst)
d1={}
for k, v in d. items():
    d1[k] = int(v)

print (d1)
```

Output:



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

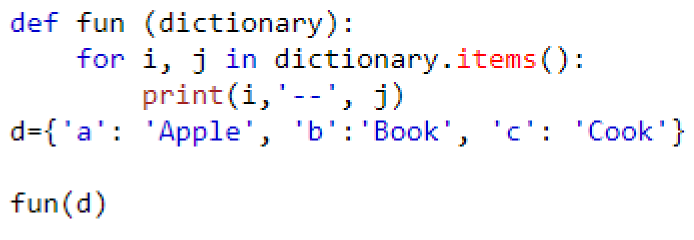
Passing Dictionaries to Functions

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 a 'dictionary' object as a parameter. Using a 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:

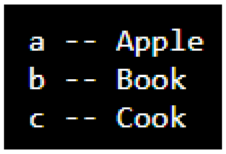
fun (d)



```python
def fun (dictionary):
    for i, j in dictionary.items():
        print(i,'--', j)
d={'a': 'Apple', 'b':'Book', 'c': 'Cook'}

fun(d)
```

Output:



```
a -- Apple
b -- Book
c -- Cook
```

Ordered Dictionaries

We already discussed that the elements of a dictionary are not ordered. It means the elements are not stored into the same order as they were entered into the dictionary. Sometimes this becomes a problem. For example, take an employee database in a company which stores employees details depending on their seniority. i.e senior most employee's data may be in the beginning of the database. If the employees' details are stored in a dictionary, this database will not show the employees details in the same order. When the employees' details are changed, the seniority is disturbed and the data of the employee who joined the company first may not appear in the beginning of the dictionary. In such a case, the solution is to use ordered dictionaries.

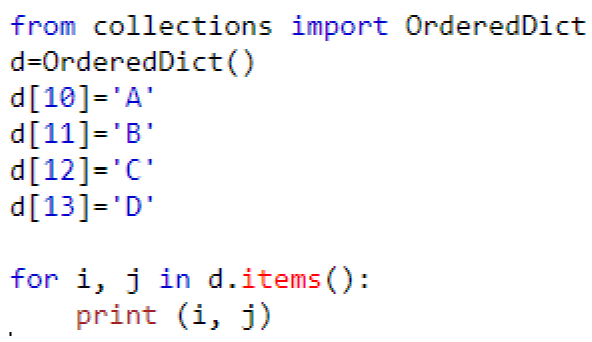
An ordered dictionary is a dictionary but it will keep the order of the elements. The elements are stored and maintained in the same order as they were entered into the ordered dictionary. We can create an ordered dictionary using the Ordered Dict() method of collections' module. So, first we should import this method from collections module, as:

from collections import orderedDict
Once this is done, we can create an ordered dictionary with the name ‘d' as:
d.OrderedDict()

We can store the key and values into 'd', as:

d[10] = 'A'
d[11] = 'B'
d[12] =’C’
d[13] =’D’

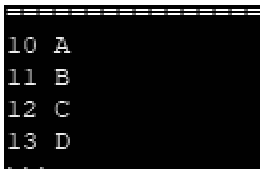
Here, 10 is the key and 'A' is its value and so on. This order is not disturbed as 'd' is an ordered dictionary. When we display the key-value pairs from the dictionary 'd', we can see the same order.



```python
from collections import OrderedDict
d=OrderedDict()
d[10]='A'
d[11]='B'
d[12]='C'
d[13]='D'

for i, j in d.items():
    print (i, j)
```

Output:



```
================
10 A
11 B
12 C
13 D
```