One key is mapped with one or more than one value.
If it is more than one value, it is represented as list

```
>>> students_dict={1:['naresh','java'],
                   2:['suresh','python'],
                   3:['ramesh','cpp']}
>>> print(students_dict)
{1: ['naresh', 'java'], 2: ['suresh', 'python'], 3: ['ramesh', 'cpp']}
>>>
```

Use the type constructor: dict(), dict([('foo', 100), ('bar', 200)]), dict(foo=100, bar=200)

Type constructor or function is used to convert other iterables into dictionary type.

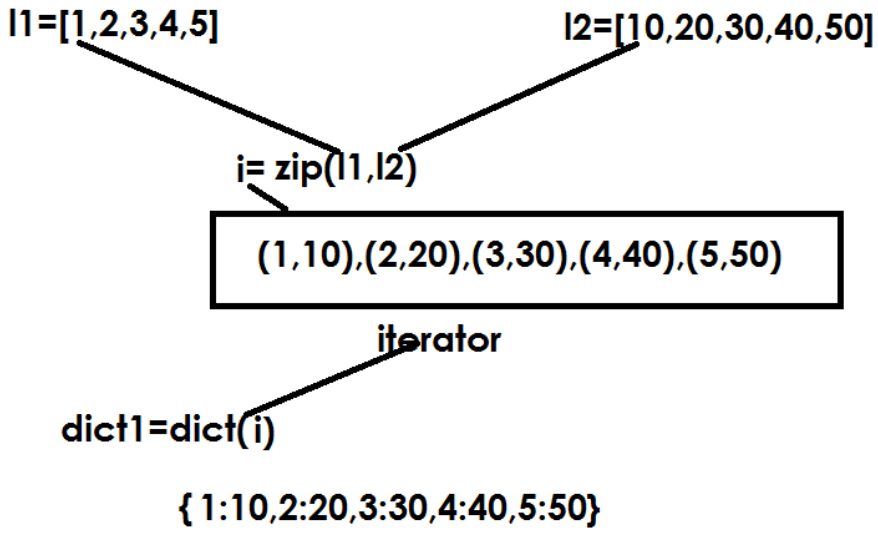
1. dict() ➔ empty dictionary
2. dict(iterable) ➔ this is creating dictionary using existing iterables

Example:

```
>>> dict1=dict()
>>> print(dict1)
{}
>>>
>>> list1=[(1,10),(2,20),(3,30),(4,40),(5,50)]
>>> dict2=dict(list1)
>>> print(dict2)
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>>
>>> list2=[10,20,30,40,50]
>>> e=enumerate(list2)
>>> dict3=dict(e)
>>> print(dict3)
{0: 10, 1: 20, 2: 30, 3: 40, 4: 50}
>>>
```

zip(*iterables)**
Make an iterator that aggregates elements from each of the iterables.

Returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. The iterator stops when the shortest input iterable is exhausted. With a single iterable argument, it returns an iterator of 1-tuples. With no arguments, it returns an empty iterator




```
>>> list1=[1,2,3,4,5]
>>> list2=[10,20,30,40,50]
>>> z=zip(list1,list2)
>>> print(z)
<zip object at 0x00000011A73F9640>
>>> dict4=dict(z)
>>> print(dict4)
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>> numbers_list=[1,2,3,4,5]
>>> sqr_list=[1,4,9,16,25]
>>> z=zip(numbers_list,sqr_list)
>>> dict5=dict(z)
>>> print(dict5)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
>>>
>>> d1={'python':4000,'java':2000,'oracle':1000,'c':2000}
>>> d2=dict(d1)
>>> print(d1)
```

```
{'python': 4000, 'java': 2000, 'oracle': 1000, 'c': 2000}
>>> print(d2)
{'python': 4000, 'java': 2000, 'oracle': 1000, 'c': 2000}
>>>
>>> d1={1:100,2.0:2000}
>>> print(d1)
{1: 100, 2.0: 2000}
>>>
```

How to read content dictionary?

1. Using key
2. Using for loop
3. Using methods of dictionary
    a. getitem()
    b. keys()
    c. values()
    d. items()
4. iterator

Using key

Dictionary is key based collection; we can read value from dictionary using key. If key is not exists it raises KeyError

```
>>> mails_dict={'naresh':'naresh@nit.com',
            'suresh':'s@gmail.com',
            'kiran':'k@yahoo.com'}
>>> print(mails_dict)
{'naresh': 'naresh@nit.com', 'suresh': 's@gmail.com', 'kiran':
'k@yahoo.com'}
>>> mails_dict['naresh']
'naresh@nit.com'
>>> mails_dict['kiran']
'k@yahoo.com'
>>> mails_dict['ramesh']
Traceback (most recent call last):
  File "<pyshell#49>", line 1, in <module>
    mails_dict['ramesh']
KeyError: 'ramesh'
>>>
```

Example:

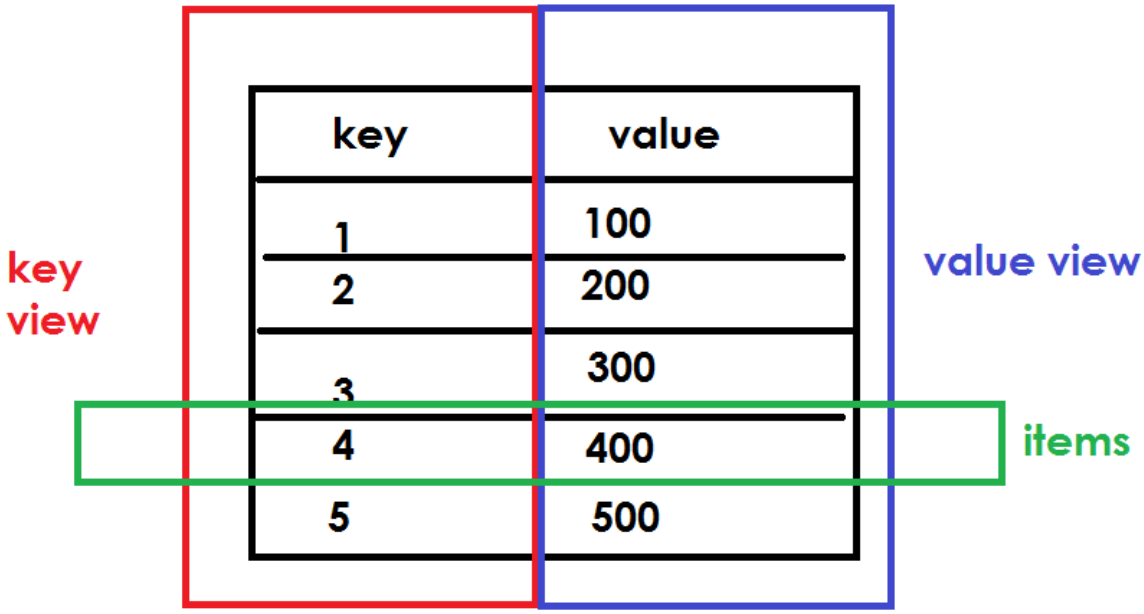
```
users_dict={'naresh':'nit123',
            'suresh':'s321',
            'kiran':'k456'}
print("****Login****")
uname=input("UserName :") # kishore
pwd=input("Password :") # s123
if uname in users_dict:
    p=users_dict[uname] # s321
    if p==pwd:
        print(f'{uname},welcome')
    else:
        print("invalid password")
else:
    print("invalid user name")
```

Output:

```
****Login****
UserName :naresh
Password :nit123
naresh,welcome
>>>
========= RESTART: C:/Users/user/Desktop/python6pm/py96.py
=========
****Login****
UserName :naresh
Password :n123
invalid password
>>>
========= RESTART: C:/Users/user/Desktop/python6pm/py96.py
=========
****Login****
UserName :kishore
Password :k123
invalid user name
>>>
```

Dictionary view objects

The objects returned by dict.keys(), dict.values() and dict.items() are *view objects*. They provide a dynamic view on the dictionary's entries, which means that when the dictionary changes, the view reflects these changes.



keys()
Return a new view of the dictionary's keys.

Example:

```
sales_dict={2015:45000,2016:50000,2017:65000,2018:35000,2019:25000}
years=sales_dict.keys()
print(years)
for year in years:
    print(year)

sales_dict[2020]=37000
print(years)
for year in years:
    print(year)
```

Output:

```
dict_keys([2015, 2016, 2017, 2018, 2019])
2015
2016
2017
2018
```

2019
dict_keys([2015, 2016, 2017, 2018, 2019, 2020])
2015
2016
2017
2018
2019
2020
>>>

values()
Return a new view of the dictionary's values.

```
sales_dict={2015:45000,2016:50000,2017:65000,2018:35000,2019:25000}
sales=sales_dict.values()
total=sum(sales)
for sale in sales:
    print(sale)
print(total)

sales_dict[2020]=50000
total=sum(sales)
print(total)
```

Output:
45000
50000
65000
35000
25000
220000
270000
>>>

items()
Return a new view of the dictionary's items ((key, value) pairs).

Example:
```
sales_dict={2015:45000,2016:50000,2017:65000,2018:35000,2019:25000}
```

```
items_view=sales_dict.items()
print(items_view)
for item in items_view:
    print(item)

for year,sales in items_view:
    print(year,sales)
```

Output:

```
dict_items([(2015, 45000), (2016, 50000), (2017, 65000), (2018, 35000),
(2019, 25000)])
(2015, 45000)
(2016, 50000)
(2017, 65000)
(2018, 35000)
(2019, 25000)
2015 45000
2016 50000
2017 65000
2018 35000
2019 25000
>>>
```

for loop
iter()

```
>>> d1={'a':'apple','b':'ball'}
>>> i=iter(d1)
>>> next(i)
'a'
>>> next(i)
'b'
>>> for k in d1:
        print(k)


a
b
>>>
```

Example:

```
sales_dict={2015:45000,2016:50000,2017:65000,2018:35000,2019:25000}
for year in sales_dict:
    print(f'{year}==>{sales_dict[year]}')
```

Output:

```
2015==>45000
2016==>50000
2017==>65000
2018==>35000
2019==>25000
>>>
```