Deleting items from dictionary

del d[key]
Remove d[key] from *d*. Raises a KeyError if *key* is not in the map.

```
>>> dict1={1:10,2:20,3:30,4:40,5:50}
>>> print(dict1)
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>> del dict1[1]
>>> print(dict1)
{2: 20, 3: 30, 4: 40, 5: 50}
>>> del dict1[4]
>>> print(dict1)
{2: 20, 3: 30, 5: 50}
>>> del dict1[4]
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    del dict1[4]
KeyError: 4
```

clear()
Remove all items from the dictionary.

```
>>> dict2=dict(zip(range(1,6),range(10,60,10)))
>>> print(dict2)
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>> dict2.clear()
>>> print(dict2)
{}
```

pop(*key*[, *default*])
If *key* is in the dictionary, remove it and return its value, else return *default*. If *default* is not given and *key* is not in the dictionary, a KeyError is raised.

```
>>> dict3=dict(zip(range(1,6),range(10,60,10)))
>>> print(dict3)
```

```
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>> value=dict3.pop(1)
>>> print(value)
10
>>> print(dict3)
{2: 20, 3: 30, 4: 40, 5: 50}
>>> value=dict3.pop(4)
>>> print(dict3)
{2: 20, 3: 30, 5: 50}
>>> print(value)
40
>>> value=dict3.pop(4,None)
>>> print(value)
None
>>> value=dict3.pop(4)
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    value=dict3.pop(4)
KeyError: 4
```

popitem()

Remove and return a (key, value) pair from the dictionary. Pairs are returned in LIFO order. Dictionary can be used as stack. Stack follows LIFO (Last In First Out). The item inserted last is removed first.

```
>>> dict1=dict(zip(range(1,6),range(10,60,10)))
>>> print(dict1)
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>> item1=dict1.popitem()
>>> print(item1)
(5, 50)
>>> item2=dict1.popitem()
>>> print(item2)
(4, 40)
>>> item3=dict1.popitem()
>>> print(item3)
(3, 30)
```

Example:

```
# Contacts (dictionary)

contacts={}
while True:
    print("1. Add Contact")
    print("2. Update Contact")
    print("3. Remove Contact")
    print("4. Search Contact")
    print("5. List Contacts")
    print("6. Exit")
    opt=int(input("Enter Your Option :"))
    if opt==1:
        contact_name=input("Contact Name :")
        if contact_name in contacts:
            print(f'{contact_name} is exists')
        else:
            contact_no=input("Contact No :")
            contacts[contact_name]=contact_no
            print("Contact Added....")
    elif opt==2:
        contact_name=input("Contact Name :")
        if contact_name in contacts:
            contact_no=input("New Contact No :")
            contacts[contact_name]=contact_no
            print("Contact Updated...")
        else:
            print("Contact not exists")
    elif opt==3:
        contact_name=input("Contact Name to Remove :")
        if contact_name in contacts:
            del contacts[contact_name]
            print("Contact Removed....")
        else:
            print("contact not exists")
    elif opt==4:
        contact_name=input("Contact Name to Search :")
```

```
        if contact_name in contacts:
            print(f'{contact_name}-------->{contacts[contact_name]}')
        else:
            print("contact not exists")
    elif opt==5:
        if len(contacts)==0:
            print("contact empty")
        else:
            for contact_name,contact_no in contacts.items():
                print(f'{contact_name}------------------------>{contact_no}')
    elif opt==6:
        break
    else:
        print("invalid option try again....")
```

Output

```
1. Add Contact
2. Update Contact
3. Remove Contact
4. Search Contact
5. List Contacts
6. Exit
Enter Your Option :5
contact empty
```

Dictionary Comprehension

Dictionary is mutable collection it allows to create dictionary using comprehension.

Syntax1: {key:value for variable in iterable}
Syntax2: {key:value for variable in iterable if test}

Example:

```
# Create dictionary with number as key and sqr of number of value

# without comprehension
```

```
dict1={}
for n in range(1,11):
    dict1[n]=n**2

print(dict1)

# with comprehension
dict1={n:n**2 for n in range(1,11)}
print(dict1)
```

Output

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Example:

```
# Write program to create dictionary to store sales data
# read the sales n persons, each person having name,sales


n=int(input("Enter how many sales persons?"))

# without comprehension
sales={}
for i in range(n):
    name=input("Enter Name ")
    s=float(input("Enter Sales "))
    sales[name]=s

print(sales)

# with comprehension
sales={input("Enter Name :"):float(input("Enter Sales :")) for i in
range(n)}
print(sales)
```

Output

```
Enter how many sales persons?2
```

```
Enter Name naresh
Enter Sales 50000
Enter Name suresh
Enter Sales 70000
{'naresh': 50000.0, 'suresh': 70000.0}
Enter Name :naresh
Enter Sales :90000
Enter Name :suresh
Enter Sales :89999
{'naresh': 90000.0, 'suresh': 89999.0}
```

Example:

```
dict1={n:chr(n) for n in range(65,91)}
print(dict1)
print(dict1[65])
dict2={n:chr(n) for n in range(97,123)}
print(dict2)
dict3={n:[value for value in range(1,n+1)] for n in range(1,6)}
print(dict3)
dict4={num:[num*i for i in range(1,11)] for num in range(1,4)}
print(dict4)
```

Output

```
{65: 'A', 66: 'B', 67: 'C', 68: 'D', 69: 'E', 70: 'F', 71: 'G', 72: 'H', 73: 'I', 74: 'J',
75: 'K', 76: 'L', 77: 'M', 78: 'N', 79: 'O', 80: 'P', 81: 'Q', 82: 'R', 83: 'S', 84: 'T',
85: 'U', 86: 'V', 87: 'W', 88: 'X', 89: 'Y', 90: 'Z'}
A
{97: 'a', 98: 'b', 99: 'c', 100: 'd', 101: 'e', 102: 'f', 103: 'g', 104: 'h', 105: 'i',
106: 'j', 107: 'k', 108: 'l', 109: 'm', 110: 'n', 111: 'o', 112: 'p', 113: 'q', 114: 'r',
115: 's', 116: 't', 117: 'u', 118: 'v', 119: 'w', 120: 'x', 121: 'y', 122: 'z'}
{1: [1], 2: [1, 2], 3: [1, 2, 3], 4: [1, 2, 3, 4], 5: [1, 2, 3, 4, 5]}
{1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 2: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20], 3: [3, 6,
9, 12, 15, 18, 21, 24, 27, 30]}
```

Syntax2: {key:value for variable in iterable if test}
This syntax add key and value based on condition.

Example

```
grade_dict={'naresh':'A',
        'suresh':'B',
        'ramesh':'C',
        'kishore':'A',
        'kiran':'A',
        'rajesh':'C'}

print(grade_dict)

grade_dictA={name:grade for name,grade in grade_dict.items() if
grade=='A'}
grade_dictB={name:grade for name,grade in grade_dict.items() if
grade=='B'}
grade_dictC={name:grade for name,grade in grade_dict.items() if
grade=='C'}

print(grade_dictA)
print(grade_dictB)
print(grade_dictC)
```

Output

```
{'naresh': 'A', 'suresh': 'B', 'ramesh': 'C', 'kishore': 'A', 'kiran': 'A', 'rajesh':
'C'}
{'naresh': 'A', 'kishore': 'A', 'kiran': 'A'}
{'suresh': 'B'}
{'ramesh': 'C', 'rajesh': 'C'}
```

Example:

```
data={1:10,2:-20,3:30,4:0,5:-50,6:60,7:70,8:0,9:0,10:-10,11:11}
print(data)
data_pos={k:v for k,v in data.items() if v>0}
data_neg={k:v for k,v in data.items() if v<0}
```

```
data_zero={k:v for k,v in data.items() if v==0}

print(data_pos)
print(data_neg)
print(data_zero)
```

Output

```
{1: 10, 2: -20, 3: 30, 4: 0, 5: -50, 6: 60, 7: 70, 8: 0, 9: 0, 10: -10, 11: 11}
{1: 10, 3: 30, 6: 60, 7: 70, 11: 11}
{2: -20, 5: -50, 10: -10}
{4: 0, 8: 0, 9: 0}
```

Example:

```
>>> dict1={[1,2,3]:100}
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    dict1={[1,2,3]:100}
TypeError: unhashable type: 'list'
>>> dict1={(1,2,3):[10,20,30]}
>>> print(dict1)
{(1, 2, 3): [10, 20, 30]}
```

Nested dictionaries

Nested dictionary is nothing but, defining dictionary inside dictionary as value.

Syntax:

```
{key:{key:value,key:value},key:{key:value,key:value},….}
```

```
>>> dict1={(1,2,3):[10,20,30]}
>>> print(dict1)
{(1, 2, 3): [10, 20, 30]}
>>> sales={2010:{'jan':40000,'feb':45000},
...        2011:{'jan':50000,'feb':65000}}
>>> print(sales[2010]['jan'])
40000
>>> print(sales[2010]['feb'])
```

```
45000
>>> print(sales[2011]['jan'])
50000
```