Mutable Operations of dictionary

Adding an item in dictionary

1. **dictionary-name[key]=value**

This syntax perform two operations
1. adding new item
2. modifying value of existing item

adding new item is done if key is not exists
modifying value of key is done, if key exists

```
>>> d1={}
>>> print(d1)
{}
>>> type(d1)
<class 'dict'>
>>> d1[1]=10
>>> print(d1)
{1: 10}
>>> d1[2]=20
>>> print(d1)
{1: 10, 2: 20}
>>> d1[3]=30
>>> print(d1)
{1: 10, 2: 20, 3: 30}
>>> d1[4]=[100,200,300]
>>> print(d1)
{1: 10, 2: 20, 3: 30, 4: [100, 200, 300]}
```

Example:

```
# Write a program to read details of n students
# each student is having name and grade

n=int(input("Enter how many students?"))
stud={}
for i in range(n):
    name=input("Enter Name :")
    grade=input("Enter Grade :")
```

```
    stud[name]=grade


for name,grade in stud.items():
    print(name,grade)
```

Output:

```
Enter how many students?2
Enter Name :naresh
Enter Grade :A
Enter Name :suresh
Enter Grade :B
naresh A
suresh B
```

Replacing value or Update value

```
>>> names={'naresh':40,'suresh':50,'kishore':25}
>>> print(names)
{'naresh': 40, 'suresh': 50, 'kishore': 25}
>>> names['naresh']=55
>>> print(names)
{'naresh': 55, 'suresh': 50, 'kishore': 25}
>>> names['ramesh']=20
>>> print(names)
{'naresh': 55, 'suresh': 50, 'kishore': 25, 'ramesh': 20}
>>> names['ramesh']=50
>>> print(names)
{'naresh': 55, 'suresh': 50, 'kishore': 25, 'ramesh': 50}
```

Deleting items from dictionary

del d[key]

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

Example:

```
>>> sales={2001:50000,2002:65000,2003:67000,2004:76000}
>>> print(sales)
{2001: 50000, 2002: 65000, 2003: 67000, 2004: 76000}
>>> del sales[2002]
>>> print(sales)
```

```
{2001: 50000, 2003: 67000, 2004: 76000}
>>> del sales[2004]
>>> print(sales)
{2001: 50000, 2003: 67000}
>>> del sales[2002]
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    del sales[2002]
KeyError: 2002
```

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.

```
>>> courses={'python':5000,'java':4000,'oracle':2000,'.net':7000}
>>> print(courses)
{'python': 5000, 'java': 4000, 'oracle': 2000, '.net': 7000}
>>> fee=courses.pop(".net")
>>>print(fee)
7000
>>> print(courses)
{'python': 5000, 'java': 4000, 'oracle': 2000}
>>> fee=courses.pop("oracle")
>>> print(fee)
2000
>>> print(courses)
{'python': 5000, 'java': 4000}
>>> fee=courses.pop("oracle")
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    fee=courses.pop("oracle")
KeyError: 'oracle'
>>> fee=courses.pop("oracle",2000)
>>> print(fee)
2000
```

popitem()
Remove and return a (key, value) pair from the dictionary. Pairs are returned in LIFO order.

```
>>> students={'naresh':'python','suresh':'java','kishore':'oracle'}
>>> print(students)
{'naresh': 'python', 'suresh': 'java', 'kishore': 'oracle'}
>>> stud1=students.popitem()
>>> print(stud1)
('kishore', 'oracle')
>>> print(students)
{'naresh': 'python', 'suresh': 'java'}
>>> stud2=students.popitem()
>>> print(stud2)
('suresh', 'java')
>>> print(students)
{'naresh': 'python'}
```

clear()
Remove all items from the dictionary.

```
>>> d1=dict(zip(range(1,6),range(10,60,10)))
>>> print(d1)
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>> d1.clear()
>>> print(d1)
{}
```

Example:

```
# shoping cart

cart={}
while True:
    print("1.Add Product")
    print("2.Update Product")
    print("3.Remove Product")
    print("4.View Cart")
    print("5.Exit")
    opt=int(input("Enter Your Option "))
    if opt==1:
        pname=input("ProductName: ")
```

```
        if pname in cart:
            print(f'{pname} exists within cart')
        else:
            qty=int(input("Qty :"))
            cart[pname]=qty
            print("Product Added...")
    elif opt==2:
        pname=input("ProductName: ")
        if pname in cart:
            qty=int(input("Update Qty:"))
            cart[pname]=qty
            print("Qty Updated...")
        else:
            print("Invalid Product Name")
    elif opt==3:
        pname=input("ProductName: ")
        if pname in cart:
            del cart[pname]
            print("Product Deleted...")
        else:
            print("Product Not Found")
    elif opt==4:
        for pname,qty in cart.items():
            print(f'{pname}-->{qty}')
    elif opt==5:
        break
    else:
        print("Invalid option...")
```

Output:

```
1.Add Product
2.Update Product
3.Remove Product
4.View Cart
5.Exit
Enter Your Option 1
ProductName: Keyboard
Qty :3
Product Added...
```

fromkeys(*iterable*[, *value*])
Create a new dictionary with keys from *iterable* and values set to *value*.

```
>>> dict1=dict.fromkeys(range(1,11))
>>> print(dict1)
{1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None,
9: None, 10: None}
>>> stud_dict=dict.fromkeys(range(1,6))
>>> print(stud_dict)
{1: None, 2: None, 3: None, 4: None, 5: None}
>>> dict2=dict.fromkeys(['k.l.rahul','rohit','virat','surya'])
>>> print(dict2)
{'k.l.rahul': None, 'rohit': None, 'virat': None, 'surya': None}
```

reversed(d)
Return a reverse iterator over the keys of the dictionary. This is a shortcut for reversed(d.keys()).

```
>>> courses={'python':5000,'java':4000,'oracle':2000,'.net':7000}
>>> for cname in courses.keys():
...     print(cname)
...
...
python
java
oracle
.net
>>> for cname in reversed(courses):
    print(cname)


.net
oracle
java
python
>>> for cname in reversed(courses):
...     print(f'{cname}-->{courses[cname]}')
...
...
.net-->7000
```

oracle-->2000
java-->4000
python-->5000

setdefault(*key*[, *default*])
If *key* is in the dictionary, return its value. If not, insert *key* with a value of *default* and return *default*. *default* defaults to None.

```
>>> d1={}
>>> value1=d1.setdefault(1)
>>> print(d1)
{1: None}
>>> print(value1)
None
>>> value2=d1.setdefault(2,20)
>>> print(value2)
20
>>> print(d1)
{1: None, 2: 20}
```

update([*other*])
Update the dictionary with the key/value pairs from *other*, overwriting existing keys. Return None.

```
>>> d1={1:10,2:20,3:30}
>>> d2={1:99,4:88,5:77,6:66,2:55}
>>> print(d1)
{1: 10, 2: 20, 3: 30}
>>> print(d2)
{1: 99, 4: 88, 5: 77, 6: 66, 2: 55}
>>> d1.update(d2)
>>> print(d1)
{1: 99, 2: 55, 3: 30, 4: 88, 5: 77, 6: 66}
```

d | other
Create a new dictionary with the merged keys and values of *d* and *other*, which must both be dictionaries. The values of *other* take priority when *d* and *other* share keys.

```
>>> d1={1:10,2:20}
```

```
>>> d2={3:30,4:40}
>>> d3=d1|d2
>>> print(d1)
{1: 10, 2: 20}
>>> print(d2)
{3: 30, 4: 40}
>>> print(d3)
{1: 10, 2: 20, 3: 30, 4: 40}
```