---: Dictionary :---

If we want to represent a group of objects as key-value pairs then we should go for dictionaries.

Characteristics of Dictionary

1. Dictionary will contain data in the form of key, value pairs.
2. Key and values are separated by a colon ":" symbol
3. One key-value pair can be represented as an item.
4. Duplicate keys are not allowed.
5. Duplicate values can be allowed.
6. Heterogeneous objects are allowed for both keys and values.
7. Insertion order is not preserved.
8. Dictionary object having mutable nature.
9. Dictionary objects are dynamic.
10. Indexing and slicing concepts are not applicable

syntax for creating dictionaries with key,value pairs is: **d = { key1:value1, key2:value2, …., keyN:valueN }**

Creating an Empty dictionary in Python:

```
d = {}
print(d)
print(type(d))

O/P:--
{}
<class 'dict'>
```

Adding the items in empty dictionary:--

```
d = {}
d[1] = "Neeraj"
d[2] = "Rahul"
d[3] = "Ravi"
print(d)

O/P:--
{1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
```

Accessing dictionary values by using keys:--

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print(d[1])
print(d[2])
print(d[3])

O/P:--
Neeraj
Rahul
Ravi
```

Note:--- While accessing, if the specified key is not available then we will get **KeyError**

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print(d[1])
print(d[2])
print(d[3])
print(d[10])

O/P:--
Neeraj
Rahul
Ravi
Traceback (most recent call last):
  File "E:\DataSciencePythonBatch\dict.py", line 16, in <module>
    print(d[10])
KeyError: 10
```

handle this KeyError by using in operator:

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

if 10 in d:
    print(d[10])
else:
    print('Key Not found')

O/P:--
Key Not found
```

Getting student information’s in the form of dictionaries:--

```
d={}
n=int(input("Enter how many student detail you want: "))
i=1
while i <=n:
   name=input("Enter Employee Name: ")
   email=input("Enter Employee salary: ")
   d[name]=email
   i=i+1
print(d)

O/P:--
Enter how many student detail you want: 3
Enter Employee Name: Neeraj
Enter Employee salary: neeraj@gmail.com
Enter Employee Name: Rahul
Enter Employee salary: rahul@gmail.com
Enter Employee Name: Ravi
Enter Employee salary: ravi@gmail.com
{'Neeraj': 'neeraj@gmail.com', 'Rahul': 'rahul@gmail.com', 'Ravi':
'ravi@gmail.com'}
```

Updating dictionary elements:
We can update the value for a particular key in a dictionary. The syntax is:

d[key] = value

Case1: While updating the key in the dictionary, if the key is not available then a new key will be added at the end of the dictionary with the specified value.

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
print("Old dict data",d)
```

```
d[10]="Arvind"
print("Nwe dict data",d)

O/P:--
Old dict data {1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
Nwe dict data {1: 'Neeraj', 2: 'Rahul', 3: 'Ravi', 10: 'Arvind'}
```

Case2: If the key already exists in the dictionary, then the old value will be replaced with a new value.

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
print("Old dict data",d)
d[2]="Arvind"
print("New dict data",d)

O/P:--
Old dict data {1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
New dict data {1: 'Neeraj', 2: 'Arvind', 3: 'Ravi'}
```

Removing or deleting elements from the dictionary:

1. By using the del keyword, we can remove the keys
2. By using clear() we can clear the objects in the dictionary

By using the del keyword

Syntax: del d[key]

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
del d[3]
print("New dict is",d)

O/P:--
New dict is {1: 'Neeraj', 2: 'Rahul'}
```

By using clear() keyword

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
d.clear()
print("New dict is",d)
O/P:--
```

```
New dict is {}
```

Delete entire dictionary object:- We can also use the del keyword to delete the total dictionary object. Before deleting we just have to note that once it is deleted then we cannot access the dictionary.

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
del d
print("New dict is",d) O/P:--

Traceback (most recent call last):
  File "E:\DataSciencePythonBatch\dict.py", line 51, in <module>
    print("New dict is",d)
NameError: name 'd' is not defined. Did you mean: 'id'?
```

Functions of dictionary in Python

1. dict()
2. len() – total object length
3. max() – on the basis of key
4. min() – on the basis of key
5. id()
6. type()

Methods of dictionary in Python

1. setdefault() # x.setdefault('name','Neeraj')

```
x = {'age': 25}
x.setdefault('name', 'Neeraj')
print(x)
```

- Since 'name' is not in x, setdefault adds 'name': 'Neeraj' to the dictionary.
- If 'name' were already a key in the dictionary, setdefault would leave it unchanged and just return the existing value.

2. fromkeys() # dict.fromkeys(keys, value) **Initializing multiple keys with the same value.**

```
keys = ['a', 'b', 'c']
new_dict = dict.fromkeys(keys, 0)
```

print(new_dict) # **{'a': 0, 'b': 0, 'c': 0}**

4. update() # x.update(collection) **Updating a dictionary with another dictionary**
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
dict1.update(dict2)
print(dict1)   # **{'a': 1, 'b': 3, 'c': 4}**

5. copy() # x.copy()
6. get() # x.get('key')
7. clear() # x.clear()
8. pop() # x.pop('key')
9. popitem() # x.popitem()
10. key() # x.keys()
11. values() # x.values()
12. items() # x.items()

dict() function:

This can be used to create an empty dictionary.

```
d=dict()
print(d)
print(type(d))

O/P:--
{}
<class 'dict'>
```

len() function: This function returns the number of items in the dictionary.

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
print(len(d))

O/P:--
3
```

clear() method: This method can remove all elements from the dictionary.

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
print(d.clear())O/P:--

O/P:--
```

```
None
```

get() method:

This method used to get the value associated with the key. This is another way to get the values of the dictionary based on the key. The biggest advantage it gives over the normal way of accessing a dictionary is, this doesn’t give any error if the key is not present. Let’s see through some examples:

Case1: If the key is available, then it returns the corresponding value otherwise returns None. It won’t raise any errors.
Syntax: d.get(key)

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
print(d.get(1))
print(d.get(2))
print(d.get(3))

O/P:--
Neeraj
Rahul
Ravi
```

Case 2: If the key is available, then returns the corresponding value otherwise returns the default value that we give.
Syntax: d.get(key, defaultvalue)

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
print(d.get(7,"Neeraj"))
print(d.get(6,"Neeraj"))
print(d.get(5,"Neeraj"))

O/P:--
Neeraj
Neeraj
Neeraj
```

pop() method: This method removes the entry associated with the specified key and returns the corresponding value. If the specified key is not available, then we will get KeyError.
Syntax: **d.pop(key)**

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
d.pop(3)
print(d)

O/P:
{1: 'Neeraj', 2: 'Rahul'}
```

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}
print(d.pop(3)) O/P:-- Ravi
```

popitem() method: This method removes an arbitrary item(key-value) from the dictionary and returns it.

```
d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi',4:'Jai',5:'Santosh'}
print(d.popitem())
print(d)

O/P:--
(5, 'Santosh')
{1: 'Neeraj', 2: 'Rahul', 3: 'Ravi', 4: 'Jai'}
```

keys() method: This method returns all keys associated with the dictionary

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}
print(d)
for k in d.keys():
  print(k)

O/P:--
1
2
3
```

values() method: This method returns all values associated with the dictionary

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}
print(d)
for k in d.values():
   print(k)


O/P:--
Ramesh
Suresh
Mahesh
```

items() method: A key-value pair in a dictionary is called an item. items() method returns the list of tuples representing key-value pairs.

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}
for k, v in d.items():
   print(k, "---", v)

O/P:--
1 --- Ramesh
2 --- Suresh
3 --- Mahesh
```