Dictionary Data Type

- We can use List, Tuple and Set to represent a group of individual objects as a single entity.
- If we want to represent a group of objects as **key-value pairs** then we should go for Dictionary.

  Eg:
  roll no----name
  phone number--address
  ipaddress---domain name
- Duplicate keys are not allowed but values can be duplicated.
- Heterogeneous objects are allowed for both key and values.
- insertion order is preserved
- Dictionaries are mutable
- Dictionaries are **dynamic**
- indexing and slicing concepts are not applicable

Note: In C++ and Java Dictionaries are known as "Map" where as in Perl and Ruby it is known as "Hash"

How to create Dictionary?

d={} or d=dict()

we are creating empty dictionary. We can add entries as follows:

```
d[100]="durga"
d[200]="ravi"
d[300]="shiva"
print(d) #{100: 'durga', 200: 'ravi', 300: 'shiva'}
```

If we know data in advance then we can create dictionary as follows:

```
d={100:'durga' ,200:'ravi', 300:'shiva'}
d={key:value, key:value}
```

How to access data from the dictionary?

We can access data by using keys.

```
d={100:'durga' ,200:'ravi', 300:'shiva'}
print(d[100]) #durga
print(d[300]) #shiva
```

If the specified key is not available then we will get KeyError

```
print(d[400]) # KeyError: 400
```

How to update dictionaries?

d[key]=value

- If the key is not available then a new entry will be added to the dictionary with the specified key-value pair

- If the key is already available then old value will be replaced with new value.

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d)
d[400]="pavan"
print(d)
d[100]="sunny"
print(d)
```

Output:

```
{100: 'durga', 200: 'ravi', 300: 'shiva'}
{100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}
{100: 'sunny', 200: 'ravi', 300: 'shiva', 400: 'pavan'}
```

How to delete elements from dictionary?

a)del d[key]

It deletes entry associated with the specified key.

If the key is not available then we will get KeyError

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d)
del d[100]
print(d)
del d[400]
```

Output

```
{100: 'durga', 200: 'ravi', 300: 'shiva'}
{200: 'ravi', 300: 'shiva'}
KeyError: 400
```

b)d.clear()

To remove all entries from the dictionary

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d)
d.clear()
print(d)
```

Output

```
{100: 'durga', 200: 'ravi', 300: 'shiva'}
{}
```

c)del d

To delete total dictionary.Now we cannot access d

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d)
del d
print(d)

Output
{100: 'durga', 200: 'ravi', 300: 'shiva'}
NameError: name 'd' is not defined
```

Important functions of dictionary:

1. dict():

To create a dictionary

d=dict() ===>It creates empty dictionary

d=dict({100:"durga",200:"ravi"}) ==>It creates dictionary with specified elements

d=dict([(100,"durga"),(200,"shiva"),(300,"ravi")])==>It creates dictionary with the given list of tuple element.

2. len()

Returns the number of items in the dictionary

3. clear():

To remove all elements from the dictionary

4. get():

To get the value associated with the key

d.get(key)

If the key is available then returns the corresponding value otherwise returns None.It won't raise any error.

d.get(key,defaultvalue)

If the key is available then returns the corresponding value otherwise returns default value.

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d[100]) ==>durga
print(d[400]) ==>KeyError:400
print(d.get(100)) ==durga
print(d.get(400)) ==>None
print(d.get(100,"Guest")) ==durga
print(d.get(400,"Guest")) ==>Guest
```

3. pop():

d.pop(key)

It 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

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d.pop(100))
print(d)
print(d.pop(400))
```

Output:

```
durga
{200: 'ravi', 300: 'shiva'}
KeyError: 400
```

4. popitem():

It removes an last item(key-value) from the dictionary and returns it.

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d)
print(d.popitem())
print(d)
```

Output:

```
{100: 'durga', 200: 'ravi', 300: 'shiva'}
(300, 'shiva')
{100: 'durga', 200: 'ravi'}
```

If the dictionary is empty then we will get KeyError

```
d={}
print(d.popitem()) ==>KeyError: 'popitem(): dictionary is empty'
```

5. keys():

It returns all keys associated with dictionary

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d.keys())
for k in d.keys():
        print(k)
```

Output:

```
dict_keys([100, 200, 300])
100
200
300
```

6. values():

It returns all values associated with the dictionary

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d.values())
for v in d.values():
print(v)
```

Output:

```
dict_values(['durga', 'ravi', 'shiva'])
durga
ravi
shiva
```

7. items():

It returns **list of tuples** representing key-value pairs.

[(k,v),(k,v),(k,v)]

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
for k,v in d.items():
print(k,"--",v)
```

Output:

```
100 -- durga
200 -- ravi
300 -- shiva
```

8. copy():

To create exactly duplicate dictionary(cloned copy)

d1=d.copy()

9. setdefault():

d.setdefault(k,v)

If the key is already available then this function returns the corresponding value.

If the key is not available then the specified key-value will be added as new item to the dictionary

Eg:

```
d={100:"durga",200:"ravi",300:"shiva"}
print(d.setdefault(400,"pavan"))
print(d)
print(d.setdefault(100,"sachin"))
print(d)
```

Output

```
pavan
{100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}
durga
{100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}
```

Dictionary Comprehension:

Comprehension concept applicable for dictionaries also.

```
squares={x:x*x for x in range(1,6)} #1,2,3,4,5
```

```
print(squares)
doubles={x:2*x for x in range(1,6)}
print(doubles)
```

Output

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
{1: 2, 2: 4, 3: 6, 4: 8, 5: 10}
```