

18. PYTHON - DICTIONARY DATA STRUCTURE

Table of Contents

1. Dictionary data structure	2
2. When should we go for dictionary?.....	3
3. Creating dictionary	4
4. Empty dictionary	4
5. We can add key-value pairs to empty dictionary.....	5
6. Access values by using keys from dictionary.....	6
7. Update dictionary.....	7
7.1.Case 1.....	7
7.2.Case 2.....	7
8. Removing or deleting elements from dictionary	9
9. len(p) function	11
10. Methods in dict class data structure	12
10.1. clear() method	14
10.2. keys() method	15
10.3. values().....	15
10.4. items()	16
11. Dictionary Comprehension:	17

18. PYTHON - DICTIONARY DATA STRUCTURE

1. Dictionary data structure

- ✓ If we want to represent group of **individual objects** as a single entity then we should go for below data structures,
 - list
 - set
 - tuple
- ✓ If we want to represent a group of objects as **key-value pairs** then we should go for,
 - dict or dictionary
- ✓ We can create dict by using,
 - **Curly braces** {} symbols
 - **dict()** predefined function.
- ✓ Dictionary data structure contains **key, value pairs**.
- ✓ key-value
 - In dictionary key, value pairs are **separated** by colon : symbol
 - One key-value pair is called as **item**.
 - In dictionary every item is separated by **comma symbol**.
 - In dictionary **duplicate keys** are **not allowed**.
 - In dictionary **duplicate values** can be **allowed**.
 - A dictionary keys and values can store **same** (Homogeneous) type of elements.
 - A dictionary keys and values can store **different** (Heterogeneous) type of elements.
- ✓ In dictionary insertion order is not preserved means **not fixed**.
- ✓ Dictionary **size** will **increase** dynamically.
- ✓ Dictionary object having **mutable** nature.
- ✓ Dictionary data structure **cannot** store the elements in **index** order.
 - Indexing and slicing concepts are not applicable

Note:

- ✓ dict is a predefined class in python.
- ✓ Once if we create dictionary object means internally object is creating for dict class.

2. When should we go for dictionary?

- ✓ If we want to represent a group of objects as **key-value** pairs then we should go for,
 - dict or dictionary

Symbols to create data structure for specific data structure?

- ✓ To create **list**, we need to use **square bracket** symbols : `[]`
- ✓ To create **tuple**, we need to use **parenthesis** symbols : `()`
- ✓ To create **set** we need to use **curly braces** with values : `{}`
- ✓ So, to create **dict** we need to use **curly braces** : `{}`

Create dictionary

Syntax

```
d = { key1 : value1, key2 : value2 }
```

3. Creating dictionary

- ✓ We can create dictionary with key, value pairs.

Program Name	creating dictionary demo1.py
	<pre>d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"} print(d)</pre>
output	<pre>{10: "Ramesh", 20: "Arjun", 30: "Daniel"}</pre>

4. Empty dictionary

- ✓ We can create empty dictionary.

Program Name	creating empty dictionary demo2.py
	<pre>d = {} print(d) print(type(d))</pre>
output	<pre>{} <class 'dict'></pre>

5. We can add key-value pairs to empty dictionary

- ✓ As we know we can create an empty dictionary.
- ✓ For that empty dictionary we can add key, value pairs.

Program Name creating empty dictionary and adding elements
demo3.py

```
d = {}  
d[10] = "Ramesh"  
d[20] = "Arjun"  
d[30] = "Daniel"  
print(d)
```

output

```
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}
```

6. Access values by using keys from dictionary

- ✓ We can access dictionary values by using keys
- ✓ Keys play main role to access the data.

Program Name Accessing dictionary values by using keys
demo4.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
  
print(d[10])  
print(d[20])  
print(d[30])
```

output

```
Ramesh  
Arjun  
Daniel
```

Program Name Accessing key and value from dictionary using for loop
demo5.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
  
for k in d:  
    print(k, d[k])
```

output

```
10 Ramesh  
20 Arjun  
30 Daniel
```

7. Update dictionary

- ✓ We can update the key in dictionary.

Syntax

```
d[key] = value
```

7.1.Case 1

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

7.2.Case 2

- ✓ While updating the key in dictionary, if key is already existing then old value will be replaced with new value.

Program Name **Case 1:** Adding key-value pair to dictionary
demo6.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
print("Old dictionary: ",d)
```

```
d[99] = "John"  
print("Added key-value 99:John pair to dictionary: ", d)
```

output

```
Old dictionary: {10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}
```

```
Added key-value 99:John pair to dictionary:  
{10: 'Ramesh', 20: 'Arjun', 30: ' Daniel', 99: 'John'}
```

Program Name **Case 2:** Updating key-value pair in dictionary
demo7.py

```
d = {10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}  
print("Old dictionary:", d)
```

```
d[30] = 'Chandhu'  
print("Updated dictionary 3:Chandhu :", d)
```

output

```
Old dictionary: {10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}  
Updated dictionary 3:Chandhu : {10: 'Ramesh', 20: 'Arjun', 30:  
'Chandhu'}
```


8. Removing or deleting elements from dictionary

- ✓ By using **del** keyword, we can remove the keys
- ✓ By using **clear()** method we can clear the objects in dictionary

8.1. By using del keyword

Syntax

```
del d[key]
```

- ✓ As per the syntax, it deletes entry associated with the specified key.
- ✓ If the key is not available, then we will get KeyError

Program Name Deleting key in dictionary
demo8.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
print("Before deleting key from dictionary: ", d)  
del d[10]  
print("After deleting key from dictionary: ", d)
```

output

```
Before deleting key from dictionary:  
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}
```

```
After deleting key from dictionary:  
{20: 'Arjun', 30: 'Daniel'}
```

We can delete total dictionary object

Syntax

```
del nameofthedictionary
```

- ✓ It can delete the total dictionary object.
- ✓ Once it deletes then we cannot access the dictionary.

Program Name Delete key in dictionary
demo9.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
print(d)  
  
del d  
print(d)
```

output

```
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}  
NameError: name 'd' is not defined
```

9. len(p) function

- ✓ This function returns the number of items in the dictionary

Program Finding length of dictionary
Name demo10.py

```
d = {100: "Ramesh", 200: "Arjun"}  
print("length of dictionary is: ",len(d))
```

output
length of dictionary is:2

10. Methods in dict class data structure

- ✓ As discussed dict is a predefined class.
- ✓ So, dict class can contain methods because methods can be created inside of class only.
- ✓ We can check these method by using `dir(parameter1)` predefined function.
- ✓ So, internally dict class contains two types of methods,
 - With underscore symbol methods.
 - We no need to focus
 - Without underscore symbol methods.
 - We need to focus much on these

Program Name Printing dict data structure methods by using `dir(dict)` function
demo11.py

```
print(dir(dict))
```

output

```
[  
  
    '__class__', ..... '__subclasshook__',  

```

Important methods

```
'clear', 'items', 'keys', 'values'
```

```
]
```

Important point

- ✓ As per object-oriented principle,
 - If we want to access instance methods, then we should access by using object name.
- ✓ So, all dict methods we can access by using dict object.

Important methods

- ✓ `clear()` method
- ✓ `keys()` method
- ✓ `values()` method
- ✓ `items()` method

10.1. clear() method

- ✓ clear() is a method in dict class, we should access this method by using dictionary object.
- ✓ This method removes all entries in dictionary.
- ✓ After deleting all entries, it just keeps empty dictionary

Program removing dictionary object by using clear() method
Name demo12.py

```
d = {10: "Ramesh", 20: "Arjun", 30:"Daniel"}  
print(d)  
d.clear()  
print("After cleared entries in dictionary: ", d)
```

output

```
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}  
After cleared entries in dictionary: {}
```

10.2. keys() method

- ✓ keys() is a method in dict class, we should access this method by using dict object.
- ✓ This method returns all keys associated with dictionary

Program Name keys() method
demo13.py

```
d = {100: "Ramesh", 200: "Daniel", 300: "Mohan"}  
print(d)  
print(d.keys())
```

Output

```
{100: 'Ramesh', 200: 'Daniel', 300: 'Mohan'}  
dict_keys([100, 200, 300])
```

10.3. values()

- ✓ values() is a method in dict class, we should access this method by using dict object.
- ✓ This method returns all values associated with the dictionary

Program Name values() method
demo14.py

```
d = {100: "Ramesh", 200: "Daniel", 300: "Mohan"}  
print(d)  
print(d.values())
```

Output

```
{100: 'Ramesh', 200: 'Daniel', 300: 'Mohan'}  
dict_values(['Ramesh', 'Daniel', 'Mohan'])
```

10.4. items()

- ✓ items() is a method in dict class, we should access this method by using dict object.
- ✓ By using this method we can access keys and values from the dictionary.

Program Name Accessing key and value from dictionary using items() method
demo15.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}
```

```
for k, v in d.items():  
    print(k, v)
```

output

```
10 Ramesh  
20 Arjun  
30 Daniel
```


11. Dictionary Comprehension:

- ✓ A dictionary comprehension represents creating new dictionary from Iterable object like a list, set, tuple, dictionary and range.
- ✓ Dictionary comprehensions code is very concise way.

Program Name	Dictionary comprehension demo15.py
	<pre>squares = {a: a*a for a in range(1,6)} print(squares)</pre>
Output	{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}