

1. Dictionaries in Python:

- A dictionary in Python is a **built-in data type** that **stores data in the form of key–value pairs**.
- Each **key** in a dictionary must be **unique** and **immutable** (such as strings, numbers, or tuples), while the **values can be of any data type** and can even be **duplicated**.
- Dictionaries are extremely useful when we need to establish a **logical connection between two pieces of information**, like storing a person's name as a key and their phone number as a value.
- Unlike **lists or tuples, which are indexed by positions**, dictionaries are **indexed by keys**, making data retrieval faster and more meaningful.
- Python dictionaries are **also dynamic**, meaning we **can add, update, or remove items at any time**.
- Since **Python 3.7**, dictionaries **preserve the order of insertion**.

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. In Python, **dictionaries preserve the insertion order** starting from **Python 3.7**
8. Dictionary **object** having **mutable nature**.
9. Dictionary **objects are dynamic**.
10. **Indexing and slicing** concepts are **not applicable**.

2. How to Create a dictionary in Python?

The syntax for creating dictionaries with **key,value pairs** is:

```
d = {key1:value1, key2:value2, ..., keyN:valueN}
```

Example: creating dictionary (demo1.py)

```
d={1:"Ramesh", 2:"Arjun", 3:"Nireekshan"}  
print(d)
```

Output: **{1: 'Ramesh', 2: 'Arjun', 3: 'Nireekshan'}**

2.1. Creating an Empty dictionary in Python:

- We can create empty dictionaries by using **curly braces**.
- And **later on**, we can add the key-value pairs into it as shown in the below example.

Example: creating an empty dictionary and adding the items (demo2.py)

```
d = {}  
d[1] = "Ramesh"  
d[2] = "Suresh"  
d[3] = "Mahesh"  
print(d)
```

Output: {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}

3. ACCESSING DICTIONARIES IN PYTHON:

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

Example: Accessing dictionary values by using keys (demo3.py)

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
print(d[1])  
print(d[2])  
print(d[3])
```

Output:

```
Ramesh  
Suresh  
Mahesh
```

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

Example: KeyError (demo4.py)

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
print(d[10])
```

Output: **KeyError: 10**

4. How to handle this KeyError?

- We can handle this error **by checking whether a key is already available or not by using “in” operator**.

Example: Handle KeyError in Python Dictionary(demo5.py)

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
if 400 in d:  
    print(d[400])  
else:  
    print("key not found")
```

Output: **key not found**

Example: Employee info program by using a dictionary(demo6.py)

```
d={}
n=int(input("Enter number of employees: "))
i=1
while i <=n:
    name=input("Enter Employee Name: ")
    salary=input("Enter Employee salary: ")
    d[name]=salary
    i=i+1
for x in d:
    print("The name is: ", x, " and his salary is: ", d[x])
```

Output

```
Enter number of employees: 2
Enter Employee Name: Ruhan
Enter Employee salary: 20000
Enter Employee Name: Farhan
Enter Employee salary: 30000
The name is: Ruhan and his salary is: 20000
The name is: Farhan and his salary is: 30000
```

5. Updating a dictionary in Python:

- 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

Case2: If the **key is already existing** in the dictionary, then the **old value will be replaced** with a new value

Example: Updating a dictionary – Case 1(demo7.py)

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}
print("Old dictionary:", d)
d[10]="Hari"
print("Added key-value(10:Hari) pair to dictionary:",d)
```

Output

```
Old dictionary: {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}
Added key-value(10:Hari) pair to dictionary: {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh', 10: 'Hari'}
```

Example: Updating a dictionary – Case 2(demo8.py)

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}
print("Old dictionary:", d)
d[3]="Hari"
print("UPdated value of key 3 pair to dictionary:",d)
```

Output

```
Old dictionary: {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
UPdated value of key 3 pair to dictionary: {1: 'Ramesh', 2: 'Suresh', 3: 'Hari'}
```

6. Removing or deleting elements from the dictionary in Python:

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

6.1.By using the del keyword

Syntax: **del d[key]**

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

Example: By using **del** keyword (demo9.py)

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
print("Before deleting key from dictionary",d)  
del d[1]  
print("After deleting key from dictionary:", d)
```

Output

```
Before deleting key from dictionary {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
After deleting key from dictionary: {2: 'Suresh', 3: 'Mahesh'}
```

Example: By using del with **KeyError** (demo10.py)

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
print("Before deleting key from dictionary",d)  
del d[10]  
print("After deleting key from dictionary:", d)
```

Output: **KeyError: 10**

6.2. By Using clear() method:

- The **clear () method** removes **all entries in the dictionary**.
- After deleting all entries, it just keeps an empty dictionary.

Example: By using clear() (demo11.py)

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
print("Before clearing dictionary: ", d)  
d.clear()  
print("After cleared entries in dictionary: ", d)
```

Output:

```
Before clearing dictionary: {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
After cleared entries in dictionary: {}
```

6.3.Delete total dictionary object in Python:

- 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.

Syntax: **del nameofthedictionary**

Example: Deleting the dictionary (demo12.py)

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}
print("Before clearing dictionary: ", d)
del d
print("After cleared entries in dictionary: ", d)
```

Output

```
Before clearing dictionary: {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}
Traceback (most recent call last):
  File "/home/ruhan/python_course/demo12.py", line 4, in <module>
    print("After cleared entries in dictionary: ", d)
NameError: name 'd' is not defined
```

7. Functions and methods of dictionary in Python

1. dict() function
2. dict({key1:value1, key2:value2}) function
3. dict([tuple1,tuple2])
4. len() function
5. clear() method
6. get() method
7. pop() method
8. popitem() method
9. keys() method
10. Items() method
11. keys() method
12. copy() method

7.1.dict() function:

- This can be **used to create an empty dictionary**. Later, we can add elements to that created dictionary

Example:

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

Output

```
{ }
<class 'dict'>
```

`dict({key1:value1, key2:value2})` function:

- It creates dictionary with specified elements

Example:

```
d = dict({1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'})
print(d)
```

Output: `{1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}`

`dict([tuple1,tuple2]):`

- This function creates a dictionary with the given list of tuple elements

Example:

```
d = dict([(1, "Ramesh"), (2, "Arjun")])
print(d)
```

Output: `{1: 'Ramesh', 2: 'Arjun'}`

`len()` function:

- This function returns the number of items in the dictionary

Example:

```
d = dict([(1, "Ramesh"), (2, "Arjun")])
print("length of dictionary is: ",len(d))
```

Output: `length of dictionary is: 2`

`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.

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)`

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)`

Example:

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}
print(d.get(1))
print(d.get(100))
```

Output

```
Ramesh
None
```

Example:

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
print(d.get(1))  
print(d.get(100,'No key found'))
```

Output

```
Ramesh  
No key found
```

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)`

Example:

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
print("Before pop:", d)  
d.pop(1)  
print("After pop:", d)
```

Output

```
Before pop: {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
After pop: {2: 'Suresh', 3: 'Mahesh'}
```

Example:

```
d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}  
d.pop(23)
```

Output: `KeyError: 23`

8. Dictionary Comprehension in Python:

- Just as we had lists and sets comprehensions, we also have this **concept applicable for dictionaries even.**

Example:

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

Output: `{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}`