Creating A Dictionary

Way 1- The Basic Approach

Simply put pairs of key-value within curly brackets. Keys are separated from their corresponding values by a colon. Different key-value pairs are separated from each other by commas. Assign this to a variable.

```
myDict= {"red":"boy", 6: 4, "name":"boy"}
months= {1:"January", 2:"February", 3: "March"}
```

Way 2- Type Casting

This way involves type casting a list into a dictionary. To typecast, there are a few mandates to be followed by the given list.

- The list must contain only tuples.
- These tuples must be of length **2**.
- The first element in these tuples must be the key and the second element is the corresponding value.

This type casting is done using the keyword **`dict`.**

```
>>> myList= [("a",1),("b",2),("c",2)]
>>> myDictionary= dict(myList)
>>> print(myDictionary)
{"a":1,"b":2,"c":2}
```

Way 3- Using inbuilt method `.copy()`

We can copy a dictionary using the method **`.copy()`**. This would create a shallow copy of the given dictionary. Thus the existing dictionary is copied to a new variable. This method is useful when we wish to duplicate an already existing dictionary.

```
>>> myDict= {"a":1,"b":2,"c":2}
>>> myDict2= myDict.copy()
>>> print(myDict2)
{"a":1,"b":2,"c":2}
```

Way 4- Using inbuilt method `.fromkeys()`

This method is particularly useful if we want to create a dictionary with variable keys and all the keys must have the same value. The values corresponding to all the keys is exactly the same. This is done as follows:

```
>>> d1= dict.fromkeys(["abc",1,"two"])
>>> print(d1)
{"abc":None ,1: None, "two": None}
# All the values are initialized to None if we provide only one argument
i.e. a list of all keys.
```

```
# We can initialise all the values to a custom value too. This is done
by providing the second argument as the desired value.
>>> d2= dict.fromkeys(["abc",1,"two"],6)
>>> print(d2)
["abc":6 ,1:6, "two":6]
```

How to access elements in a dictionary?

We already know that the indexing in a dictionary is done with the keys from the various key-value pairs present within. Thus to access any value we need to use its index i.e. it's **key**.

Similar to the list and tuples, this can be done by the square bracket operator **[ ]**.

```
foo= {"a":1,"b":2,"c":3}
print(foo["a"])
--> 1
print(foo["c"])
--> 3
```

If we want the value corresponding to any key , we can even use an inbuilt dictionary method called **`get`.**

```
>>> foo= {"a":1,"b":2,"c":3}
>>> print(foo.get("a"))
1
>>> print(foo.get("c"))
3
```

A very unique feature about this method is that , incase the desired **key** is not present in the dictionary , it won't throw an error or an exception. It would simple return **`None`**.

We can make use of this feature in another way. Say we want the method to do the following action: If the key is present in the dictionary then return the value corresponding to the key. In case the key is not present, return a custom desired value (say 0).

This can be done as follows:

```
>>> foo= {"a":1,"b":2,"c":3}
>>> print(foo.get("a",0))
1
>>> print(foo.get("d",0))
0
```

Accessing all the available keys in the dictionary:

This can be done using the method **`.keys()`**. This method returns all the different keys present in the dictionary in the form of a list.

```
>>> foo= {"a":1,"b":2,"c":3}
>>> foo.keys()
dict_keys(["a","b","c"])
```

Accessing all the available values in the dictionary:

This can be done using the method **`.values()`**. This method returns all the different values present in the dictionary in the form of a list.

```
>>> foo= {"a":1,"b":2,"c":3}
>>> foo.values()
dict_values([1,2,3])
```

Accessing all the available items in the dictionary:

This can be done using the method `.items()`. This method returns all the different items (key-value pairs) present in the dictionary in the form of a list of tuples, with the first element of the tuple as the key and the second element as the value corresponding to this key .

```
>>> foo= {"a":1,"b":2,"c":3}
>>> foo.items()
dict_items([("a",1),("b",2),("c",3)])
```

Checking if the dictionary contains a given key:

The keyword used is `in`. We can easily get a boolean output using this keyword. It returns True if the dictionary contains the given **key**, else the output is False. This checks the presence of the keys and not the presence of the values.

```
>>> foo= {"a":1,"b":2,"c":3}
>>> "a" in foo
True
>>> 1 in foo
False
```

Iterating over a Dictionary:

In order to traverse through the dictionary, we can use a simple for loop. The loop will go through the keys of the dictionary one by one and do the required action.

```
bar= {2:1,3:2,4:3}
for t in bar:
      print(t)
Out[]:
2
3
4
# Here t is the key in the dictionary and hence when we print t in all iterations then all the keys are printed.
```

```
for t in bar:
      print(t, bar[t])
Out[]:
2 1
3 2
4 3
# Here along with the keys, the values which are bar[t] are printed. In this loop, the values are accessed using the keys.
```