Python Dictionary

Difficulty Level : Easy • Last Updated : 14 Jun, 2020

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds `key:value` pair. Key value is provided in the dictionary to make it more optimized.

Note – Keys in a dictionary doesn't allows Polymorphism.

Creating a Dictionary

In Python, a Dictionary can be created by placing sequence of elements within curly **{}** braces, separated by 'comma'. Dictionary holds a pair of values, one

being the Key and the other corresponding pair element being its **`Key:value`**. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be *immutable*.

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

```
# Creating a Dictionary
# with Integer Keys
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

# Creating a Dictionary
# with Mixed keys
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)
```

Output:

```
Dictionary with the use of Integer Keys:
{1: 'Geeks', 2: 'For', 3: 'Geeks'}

Dictionary with the use of Mixed Keys:
{1: [1, 2, 3, 4], 'Name': 'Geeks'}
```

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing to curly braces{}.

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Creating a Dictionary
# with dict() method
Dict = dict({1: 'Geeks', 2: 'For', 3:'Geeks'})
print("\nDictionary with the use of dict(): ")
print(Dict)

# Creating a Dictionary
# with each item as a Pair
Dict = dict([(1, 'Geeks'), (2, 'For')])
print("\nDictionary with each item as a pair: ")
print(Dict)
```

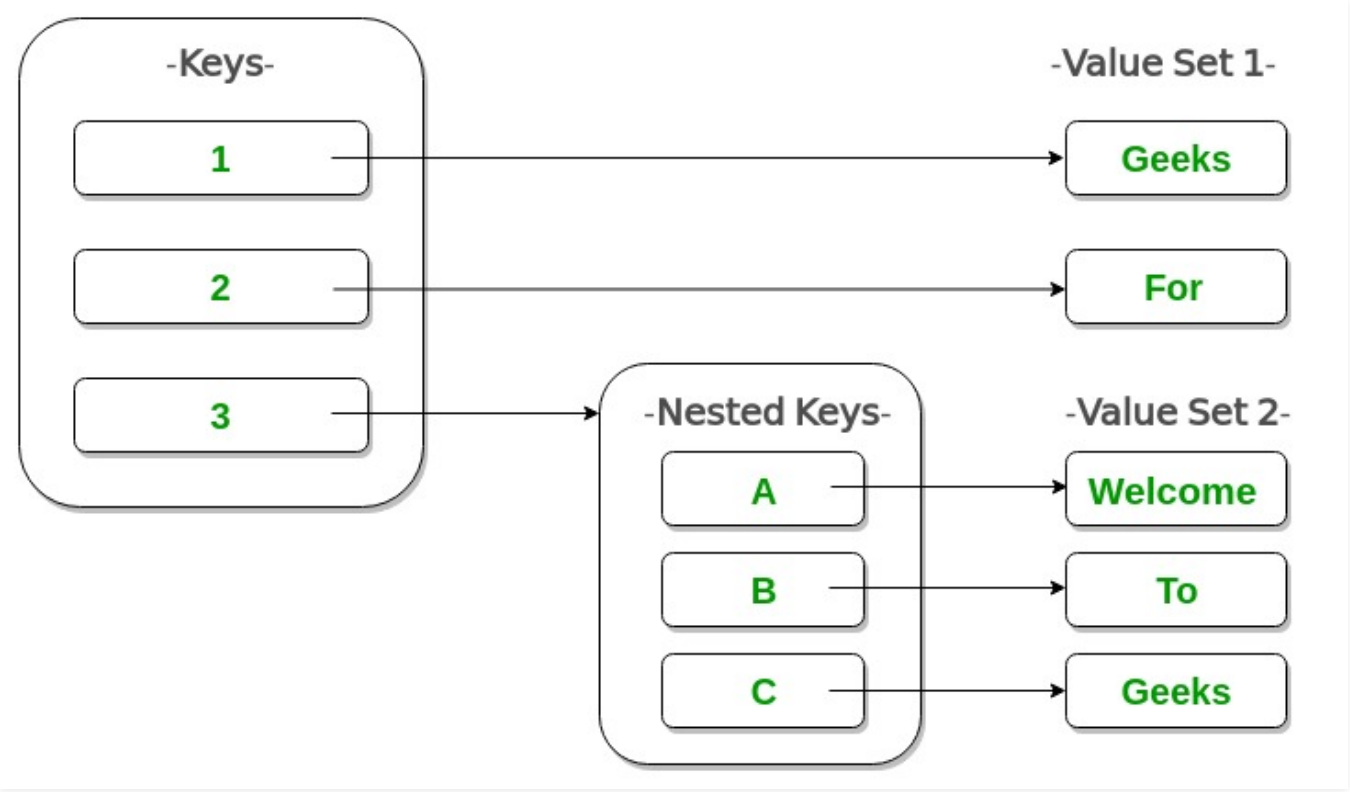
Output:

```
Empty Dictionary:
{}

Dictionary with the use of dict():
{1: 'Geeks', 2: 'For', 3: 'Geeks'}

Dictionary with each item as a pair:
{1: 'Geeks', 2: 'For'}
```

Nested Dictionary:




```
# Creating a Nested Dictionary
# as shown in the below image
```

```
Dict = {1: 'Geeks', 2: 'For',
        3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Geeks'}}

print(Dict)
```

Output:

```
{1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}
```

Adding elements to a Dictionary

In Python Dictionary, Addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = 'Value'. Updating an existing value in a Dictionary can be done by using the built-in `update()` method. Nested key values can also be added to an existing Dictionary.

Note- While adding a value, if the key value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

```
# Creating an empty Dictionary
Dict = {}
```

```
print("Empty Dictionary: ")
print(Dict)

# Adding elements one at a time
Dict[0] = 'Geeks'
Dict[2] = 'For'
Dict[3] = 1
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Adding set of values
# to a single Key
Dict['Value_set'] = 2, 3, 4
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[2] = 'Welcome'
print("\nUpdated key value: ")
print(Dict)

# Adding Nested Key value to Dictionary
Dict[5] = {'Nested' :{'1' : 'Life', '2' : 'Geeks'}}
print("\nAdding a Nested Key: ")
print(Dict)
```

Output:

```
Empty Dictionary:
{}

Dictionary after adding 3 elements:
{0: 'Geeks', 2: 'For', 3: 1}

Dictionary after adding 3 elements:
{0: 'Geeks', 2: 'For', 3: 1, 'Value_set': (2, 3, 4)}

Updated key value:
{0: 'Geeks', 2: 'Welcome', 3: 1, 'Value_set': (2, 3, 4)}

Adding a Nested Key:
{0: 'Geeks', 2: 'Welcome', 3: 1, 5: {'Nested': {'1': 'Life', '2': 'Geeks'}}
```

Accessing elements from a Dictionary

In order to access the items of a dictionary refer to its key name.Key can be used inside square brackets.

```
# Python program to demonstrate
# accessing a element from a Dictionary

# Creating a Dictionary
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# accessing a element using key
print("Accessing a element using key:")
print(Dict['name'])

# accessing a element using key
print("Accessing a element using key:")
print(Dict[1])
```

Output:

```
Accessing a element using key:
For
```

```
Accessing a element using key:
Geeks
```

There is also a method called get() that will also help in acessing the element from a dictionary.

```
# Creating a Dictionary
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# accessing a element using get()
# method
print("Accessing a element using get:")
print(Dict.get(3))
```

Output:

```
Accessing a element using get:
Geeks
```

Accessing element of a nested dictionary

In order to access the value of any key in nested dictionary, use indexing [] syntax.

```
# Creating a Dictionary
Dict = {'Dict1': {1: 'Geeks'},
        'Dict2': {'Name': 'For'}}

# Accessing element using key
print(Dict['Dict1'])
print(Dict['Dict1'][1])
print(Dict['Dict2']['Name'])
```

Output:

```
{1: 'Geeks'}
Geeks
For
```

Removing Elements from Dictionary

Using `del` keyword

In Python Dictionary, deletion of keys can be done by using the `del` keyword. Using del keyword, specific values from a dictionary as well as whole dictionary can be deleted. Items in a Nested dictionary can also be deleted by using del keyword and providing specific nested key and particular key to be deleted from that nested Dictionary.

Note- `del Dict` will delete the entire dictionary and hence printing it after deletion will raise an Error.

```
# Initial Dictionary
Dict = { 5 : 'Welcome', 6 : 'To', 7 : 'Geeks',
        'A' : {1 : 'Geeks', 2 : 'For', 3 : 'Geeks'},
        'B' : {1 : 'Geeks', 2 : 'Life'}}
print("Initial Dictionary: ")
print(Dict)

# Deleting a Key value
del Dict[6]
print("\nDeleting a specific key: ")
print(Dict)

# Deleting a Key from
# Nested Dictionary
```

```
del Dict['A'][2]
print("\nDeleting a key from Nested Dictionary: ")
print(Dict)
```

Output:

```
Initial Dictionary:
{'A': {1: 'Geeks', 2: 'For', 3: 'Geeks'}, 'B': {1: 'Geeks', 2: 'Life'}, 5:

Deleting a specific key:
{'A': {1: 'Geeks', 2: 'For', 3: 'Geeks'}, 'B': {1: 'Geeks', 2: 'Life'}, 5:

Deleting a key from Nested Dictionary:
{'A': {1: 'Geeks', 3: 'Geeks'}, 'B': {1: 'Geeks', 2: 'Life'}, 5: 'Welcome'
```

Using `pop()` method

Pop() method is used to return and delete the value of the key specified.

```
# Creating a Dictionary
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}
```

```
# Deleting a key
# using pop() method
pop_ele = Dict.pop(1)
print('\nDictionary after deletion: ' + str(Dict))
print('Value associated to poped key is: ' + str(pop_ele))
```

Output:

```
Dictionary after deletion: {3: 'Geeks', 'name': 'For'}
Value associated to poped key is: Geeks
```

Using `popitem()` method

The popitem() returns and removes an arbitrary element (key, value) pair from the dictionary.

```
# Creating Dictionary
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# Deleting an arbitrary key
# using popitem() function
```

```
pop_ele = Dict.popitem()
print("\nDictionary after deletion: " + str(Dict))
print("The arbitrary pair returned is: " + str(pop_ele))
```

Output:

```
Dictionary after deletion: {3: 'Geeks', 'name': 'For'}
The arbitrary pair returned is: (1, 'Geeks')
```

Using `clear()` method

All the items from a dictionary can be deleted at once by using **`clear()`** method.

```
# Creating a Dictionary
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}


# Deleting entire Dictionary
Dict.clear()
print("\nDeleting Entire Dictionary: ")
print(Dict)
```

Output:

```
Deleting Entire Dictionary:
{}
```

Dictionary Methods

Methods	Description
copy()	They copy() method returns a shallow copy of the dictionary.
clear()	The clear() method removes all items from the dictionary.
pop()	Removes and returns an element from a dictionary having the given key.
popitem()	Removes the arbitrary key-value pair from the dictionary and returns it as tuple.

get()	It is a conventional method to access a value for a key.
dictionary_name.values()	returns a list of all the values available in a given dictionary.
str()	Produces a printable string representation of a dictionary.
update()	Adds dictionary dict2's key-values pairs to dict
setdefault()	Set dict[key]=default if key is not already in dict
keys()	Returns list of dictionary dict's keys
items()	Returns a list of dict's (key, value) tuple pairs
has_key()	Returns true if key in dictionary dict, false otherwise
fromkeys()	Create a new dictionary with keys from seq and values set to value.