Dictionary

A Python dictionary is a data structure that stores the value in **key: value** pairs. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be *immutable*.

Declaration/Creation of Dictionary -

```python
# A Python dictionary is a data structure that stores the value in key: value pairs.
# Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable.
# Dictionary is mutable
# empty dictionary
d={}
d=dict()

d=dict(a="apple", b="ball", c="cat")
print(d)
# key must be unique
# d={key:value}
d={200:"great", 300:"great"}
print(d)
# d[key]= value # we are storing this key value into dictionary d
d[100]="durga"
print(d)
# We are replacing the value of key
d[100]="veer" # Modifying key 100 of dict d, replacing value durga with veer
print(d)
```

Remember Points:

- **Dictionary keys are case sensitive:** the same name but different cases of Key will be treated distinctly.
- **Keys must be immutable:** This means keys can be strings, numbers, or tuples but not lists.
- **Keys must be unique:** Duplicate keys are not allowed and any duplicate key will overwrite the previous value.

Adding and Updating Dictionary Items

We can add new key-value pairs or update existing keys by using different ways.

Update Method -updates the dictionary with the elements from another dictionary object or from an iterable of key/value pairs.

[Note: d = {'a': 10, 'b': 20, 'c': 30, 'bet':'ball', 9:'popi'}
 # Update the value of key 9 using keyword arguments
 d.update(9='boom') # invalid coz digit id not identifier
keyword arguments must be valid **Python identifiers** (variable-like names).Identifiers must start with a letter (`a-z`, `A-Z`) or underscore (`_`) and can contain digits (`0-9`), but **cannot start with a digit**.
d.update(a='new_value') # ✅ Valid because 'a' is a valid identifier]

Setdefault Method – It returns the value of a key (if the key is in dictionary). Else, it inserts a key with the default value to the dictionary.

```
d = {1: 10, 2: 20, 3: 30}
# Adding using update method
d.update({4: 40, 5: 50})  # Add multiple key-value pairs to dictionary d
print(d)
# Adding using key
# d[key]= value # we are storing this key value into dictionary d, Adding this
data to dictionary d
d[100]="durga"
# Adding using setdefault function
d.setdefault(400, "geet") # if key not exist it will add that it into
dictionary , if key exist its value would remain unchanged.
print(d)
d1={400:'Jaipur'}
d.update(d1) # updating dictionary d with dictionary d1 data
print(d)
d2={1:'Beep'}
d.update(d2) # here it modifying existing key 1 of d with new value 'Beep'
print(d)
# We are replacing the value of key, Updaying the value of key
d[100]="veer" # Modifying key 100 of dict d, replacing value durga with veer
print(d)
```

Accessing Dictionary Items
We can access a value from a dictionary by using the key within square brackets or get() method.

```
d={200:"great", 300:"great"}
print(d)
# d[key]= value # we are storing this key value into dictionary d, Adding this
data to dictionary d
d[100]="durga"
print(d)
# We are replacing the value of key, Updaying the value of key
d[100]="veer" # Modifying key 100 of dict d, replacing value durga with veer
print(d)
# we can acess the data of dict using key not using index position
print(d[100]) # Accessing value through key
print(d.values()) # Fetching all the values present in dict
# d.get(key) # fetching the value of that key, which is passes to the get()
method
print(d.get(100))
print(d.items()) # fetching key value both

#  Iterating over the dictionary item
# Fetching data of dictionary
for key in d:
    print(d[key])
for value in d.values():
    print(value)
for key, value in d.items():
    print(key, value)
for key in d:
    print(key)
for i in d.keys():
    print(i)

# Nested dictionary - Dictionary inside dictionary
d = {"person": {"name": "Alice", "age": 25},"location": {"city": "New
York","country": "USA"}}

# Accessing nested dictionary values
print(d["person"]["name"])
print(d["location"]["city"])
```

Removing Dictionary Items

We can remove items from dictionary using the following methods:

- del: Removes an item by key. We can delete complete dictionary object.

- pop(): Removes an item by key and returns its value.
- clear(): Empties the dictionary.
- popitem(): Removes and returns the last key-value pair.

```
d={100: 'veer', 200:'great', 300:'great'}
# d.clear()# remove all elements
del d[100] # deleting data of key 100
# del d # Total delete: it will delete the object from the memory location
print(d)
# pop(key)
print(d.pop(400)) #  pop method will return the value of that key
print(d.popitem()) # It will  delete the last elements from the dictionary
print(d)
```

Dictionary Comprehension: Dictionary comprehension is a concise way to create and manipulate dictionaries using a single line of code.

```
# Dictionary Comprehension
myDict = {x: x**2 for x in [1,2,3,4,5]}
print (myDict)
# Creating a dictionary with squares of numbers
squares = {x: x**2 for x in range(5)}
print(squares)  # Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
sDict = {x.upper(): x*3 for x in 'coding '}
print (sDict) # Output: {'O': 'ooo', 'N': 'nnn', 'I': 'iii', 'C': 'ccc', 'D':
'ddd', 'G': 'ggg'}
```

Using the zip() Function

zip() function can combine two lists (keys and lists of values) into a dictionary of lists. This method is efficient when the data is already structured as two separate lists.

```
# Zip function
k = ["Fruits", "Vegetables", "Drinks"]
val = [["Apple", "Banana"], ["Carrot", "Spinach"], ["Water", "Juice"]]
# Create a dictionary of lists using zip
d = dict(zip(k, val))
print(d)
# zip(keys, values) combines the keys and values lists into dictionary
```

Using setdefault()

Setdefault() method simplifies handling missing keys by initializing a default list if the key doesn't exist.

```
# Using setdefault()
# setdefault() method simplifies handling missing keys by initializing a default
list if the key doesn't exist.
li = [("Fruits", "Apple"), ("Fruits", "Banana"), ("Vegetables", "Carrot")]
# Initialize an empty dictionary
d = {}
# Use setdefault to populate the dictionary
for k, item in li:
    d.setdefault(k, []).append(item)
print(d)
```

Dictionary of lists

A dictionary of lists is a type of dictionary where each value is a list. These dictionaries are commonly used when we need to associate multiple values with a single key.

```
# # Dictionary of list
# d = {} # empty dictionary
# # Adding list as value
# d["1"] = [1, 2]
# d["2"] = ["Geeks", "For", "Geeks"]
# print(d) # {'1': [1, 2], '2': ['Geeks', 'For', 'Geeks']}
```

Dictionary of dictionary

A dictionary of lists is a type of dictionary where each value is a dictionary.

```
# Dictionary of Dictionary
# Nested dictionary - Dictionary inside dictionary
d = {"person": {"name": "Alice", "age": 25},"location": {"city": "New
York","country": "USA"}}
# Accessing nested dictionary values
print(d["person"]["name"])
print(d["location"]["city"])
```

Use defaultdict from collections

defaultdict automatically creates a default value for keys that don't exist, making it ideal for building a dictionary of lists dynamically.

```
# Initialize a defaultdict with list as the default type and not defined
d = defaultdict(list)
# Add values to the dictionary
d[1].append("Apple")
d[2].append("Banana")
d[3].append("Carrot")
d[4].append(1)
print(d) # defaultdict(<class 'list'>, {1: ['Apple'], 2: ['Banana'], 3:
['Carrot']})
```

Dictionary Methods

```
# Dictionary Methods
d_m={10:"ten", 20:"Twenty", 30:"Thiry"}
d_c=d_m # Both sharing the same location, id is same
print("Copying data of d_m to d_c", id(d_c), id(d_m))
d_s=d_m.copy() # copy() creates a shallow copy of my_dict, meaning new_dict is a
new dictionary with the same key-value pairs
#  but a different memory address.
# id(new_dict) != id(my_dict), confirming that they are distinct objects in memory.

# fromkeys() method creates a new dictionary with specified keys and an optional
default value
# dict.fromkeys(keys, value)
# keys: A sequence (list, tuple, or set) of keys for the new dictionary.
# value: The value assigned to all keys (default is None if not provided)
seq = {'a', 'b', 'c', 'd', 'e'}
# creating dict with default values as None
res_dict = dict.fromkeys(seq)
print("The newly created dict with None values : " + str(res_dict))
# creating dict with default values as 1
res_dict2 = dict.fromkeys(seq, 1)
print("The newly created dict with 1 as value : " + str(res_dict2))
```

```
# Dictionary unpacking allows us to merge dictionaries into a new one.
d1 = {'x': 1, 'y': 2}
d2 = {'y': 3, 'z': 4}
```

```python
d3 = {**d1, **d2}
print(d3)
```