DICTIONARIES IN PYTHON

- Python dictionary represents a *mapping between a key and a value*. In simple terms, a Python dictionary can store pairs of keys and values. *Each key is linked to a specific value*.

- Once stored in a dictionary, you can later obtain the value using just the key.

- For example, consider the Phone lookup, where it is very easy and fast to find the phone number(value) when we know the name(Key) associated with it.

Creating a dictionary

There are following three ways to create a dictionary.

- **Using curly brackets**: The dictionaries are created by enclosing the comma-separated Key: Value pairs inside the `{}` curly brackets. The colon '`:`' is used to separate the key and value in a pair.
- **Using dict() constructor**: Create a dictionary by passing the comma-separated key: value pairs inside the `dict()`.
- **Using sequence** having each item as a pair (key-value)

Example:

```
# create a dictionary using {}
person = {"name": "Jessa", "country": "USA", "telephone": 1178}
print(person)
# output {'name': 'Jessa', 'country': 'USA', 'telephone': 1178}

# create a dictionary using dict()
person = dict({"name": "Jessa", "country": "USA", "telephone": 1178})
print(person)
# output {'name': 'Jessa', 'country': 'USA', 'telephone': 1178}

# create a dictionary from sequence having each item as a pair
person = dict([("name", "Mark"), ("country", "USA"), ("telephone", 1178)])
print(person)

# create dictionary with mixed keys
# first key is string and second is an integer
sample_dict = {"name": "Jessa", 10: "Mobile"}
print(sample_dict)
```

```
# output {'name': 'Jessa', 10: 'Mobile'}

# create dictionary with value as a list
person = {"name": "Jessa", "telephones": [1178, 2563, 4569]}
print(person)
# output {'name': 'Jessa', 'telephones': [1178, 2563, 4569]}
```

Empty Dictionary

When we create a dictionary without any elements inside the curly brackets then it will be an empty dictionary.

```
emptydict = {}
print(type(emptydict))
# Output class 'dict'
```

- A dictionary value can be of any type, and duplicates are allowed in that.
- Keys in the dictionary must be unique and of immutable types like string, numbers, or tuples.

Accessing elements of a dictionary

There are two different ways to access the elements of a dictionary.

1. Retrieve value using the key name inside the [] square brackets
2. Retrieve value by passing key name as a parameter to the get() method of a dictionary.

Example

```
# create a dictionary named person
person = {"name": "Jessa", "country": "USA", "telephone": 1178}

# access value using key name in []
print(person['name'])
# Output 'Jessa'

#  get key value using key name in get()
print(person.get('telephone'))
# Output 1178
```

Get all keys and values

Use the following dictionary methods to retrieve all key and values at once

Method	Description
keys()	Returns the list of all keys present in the dictionary.
values()	Returns the list of all values present in the dictionary
items()	Returns all the items present in the dictionary. Each item will be inside a tuple as a key-value pair.

Example

```
person = {"name": "Jessa", "country": "USA", "telephone": 1178}
# Get all keys
print(person.keys())
# output dict_keys(['name', 'country', 'telephone'])

print(type(person.keys()))
# Output class 'dict_keys'

# Get all values
print(person.values())
# output dict_values(['Jessa', 'USA', 1178])


print(type(person.values()))
# Output class 'dict_values'

# Get all key-value pair
print(person.items())
```

```
# output dict_items([('name', 'Jessa'), ('country', 'USA'), ('telephone', 1178)])
print(type(person.items()))
# Output class 'dict_items'
```

Iterating a dictionary

We can iterate through a dictionary using a for-loop and access the individual keys and their corresponding values.

```
person = {"name": "Jessa", "country": "USA", "telephone": 1178}

# Iterating the dictionary using for-loop
print('key', ':', 'value')
for key in person:
    print(key, ':', person[key])

# using items() method
print('key', ':', 'value')
for key_value in person.items():
    # first is key, and second is value
    print(key_value[0], key_value[1])
```

key : value
name : Jessa
country : USA
telephone : 1178

key : value
name Jessa
country USA
telephone 1178

Find a length of a dictionary

In order to find the number of items in a dictionary, we can use the len() function.

```
person = {"name": "Jessa", "country": "USA", "telephone": 1178}
```

```
# count number of keys present in  a dictionary
print(len(person))
# output 3
```

Adding items to the dictionary

We can add new items to the dictionary using the following two ways.

- **Using key-value assignment:** Using a simple assignment statement where value can be assigned directly to the new key.
- **Using update() Method:** In this method, the item passed inside the update() method will be inserted into the dictionary. The item can be another dictionary or any iterable like a tuple of key-value pairs.

Example

```
person = {"name": "Jessa", 'country': "USA", "telephone": 1178}

# update dictionary by adding 2 new keys
person["weight"] = 50
person.update({"height": 6})

# print the updated dictionary
print(person)
# output {'name': 'Jessa', 'country': 'USA', 'telephone': 1178, 'weight': 50, 'height': 6}
```

Removing items from the dictionary

There are several methods to remove items from the dictionary. Whether we want to remove the single item or the last inserted item or delete the entire dictionary, we can choose the method to be used.

Use the following dictionary methods to remove keys from a dictionary.

Method	Description
pop (key [, d])	Return and removes the item with the key and return its value. If the key is not found, it raises KeyError.

Method	Description
popitem()	Return and removes the last inserted item from the dictionary. If the dictionary is empty, it raises KeyError.
del key	The del keyword will delete the item with the key that is passed
clear()	Removes all items from the dictionary. Empty the dictionary
del dict_name	Delete the entire dictionary

Example

```
person = {'name': 'Jessa', 'country': 'USA', 'telephone': 1178, 'weight': 50, 'height': 6}

# Remove last inserted item from the dictionary
deleted_item = person.popitem()
print(deleted_item)                    # output ('height', 6)

# display updated dictionary
print(person)
                # Output {'name': 'Jessa', 'country': 'USA', 'telephone': 1178, 'weight': 50}

# Remove key 'telephone' from the dictionary
deleted_item = person.pop('telephone')
print(deleted_item)                    # output 1178

# display updated dictionary
print(person)
                        # Output {'name': 'Jessa', 'country': 'USA', 'weight': 50}

# delete key 'weight'
del person['weight']
```

```
# display updated dictionary
print(person)
                                    # Output {'name': 'Jessa', 'country': 'USA'}

# remove all item (key-values) from dict
person.clear()
# display updated dictionary
print(person)                       # {}

# Delete the entire dictionary
del person
# display updated dictionary
print(person)
```

Checking if a key exists

In order to *check whether a particular key exists in a dictionary*, we can use the keys() method and in operator. We can use the in operator to check whether the key is present in the list of keys returned by the keys() method.

In this method, we can just check whether our key is present in the list of keys that will be returned from the keys() method.

Let's check whether the 'country' key exists and prints its value if found.

```
person = {'name': 'Jessa', 'country': 'USA', 'telephone': 1178}

# Get the list of keys and check if 'country' key is present
key_name = 'country'
if key_name in person.keys():
    print("country name is", person[key_name])
else:
    print("Key not found")
# Output country name is USA
```

Copy a Dictionary

We can create a copy of a dictionary using the following two ways

- Using copy() method.

- Using the `dict()` constructor

```
dict1 = {'Jessa': 70, 'Emma': 55}

# Copy dictionary using copy() method
dict2 = dict1.copy()
# Printing the new dictionary
print(dict2)
                                        # output {'Jessa': 70, 'Emma': 55}

# Copy dictionary using dict() constructor
dict3 = dict(dict1)
print(dict3)
                                        # output {'Jessa': 70, 'Emma': 55}

# Copy dictionary using the output of items() methods
dict4 = dict(dict1.items())
print(dict4)
                                        # output {'Jessa': 70, 'Emma': 55}
```

Copy using the assignment operator

- We can simply use the `'='` operator to create a copy.

Note: When you set `dict2 = dict1`, you are making them refer to the same dict object, so when you modify one of them, all references associated with that object reflect the current state of the object. So don't use the assignment operator to copy the dictionary instead use the `copy()` method.

Example

```
dict1 = {'Jessa': 70, 'Emma': 55}

# Copy dictionary using assignment = operator
dict2 = dict1
# modify dict2
dict2.update({'Jessa': 90})
print(dict2)
```

```
                                    # Output {'Jessa': 90, 'Emma': 55}

print(dict1)
                                    # Output {'Jessa': 90, 'Emma': 55}
```

Nested dictionary

Nested dictionaries are *dictionaries that have one or more dictionaries as their members*.

It is a collection of many dictionaries in one dictionary.

Example

```
# address dictionary to store person address
address = {"state": "Texas", 'city': 'Houston'}

# dictionary to store person details with address as a nested dictionary
person = {'name': 'Jessa', 'company': 'Google', 'address': address}

# Display dictionary
print("person:", person)

# Get nested dictionary key 'city'
print("City:", person['address']['city'])

# Iterating outer dictionary
print("Person details")
for key, value in person.items():
    if key == 'address':
        # Iterating through nested dictionary
        print("Person Address")
        for nested_key, nested_value in value.items():
            print(nested_key, ':', nested_value)
    else:
        print(key, ':', value)
```

Output

```
person: {'name': 'Jessa', 'company': 'Google', 'address': {'state': 'Texas', 'city': 'Houston'}}

City: Houston
```

```
Person details
name: Jessa
company: Google

Person Address
state: Texas
city : Houston
```

Sort dictionary

The built-in method sorted() will sort the keys in the dictionary and returns a sorted list. In case we want to sort the values we can first get the values using the `values()` and then sort them.

Example

```
dict1 = {'c': 45, 'b': 95, 'a': 35}

# sorting dictionary by keys
print(sorted(dict1.items()))
# Output [('a', 35), ('b', 95), ('c', 45)]

# sort dict eys
print(sorted(dict1))
# output ['a', 'b', 'c']

# sort dictionary values
print(sorted(dict1.values()))
# output [35, 45, 95]
```