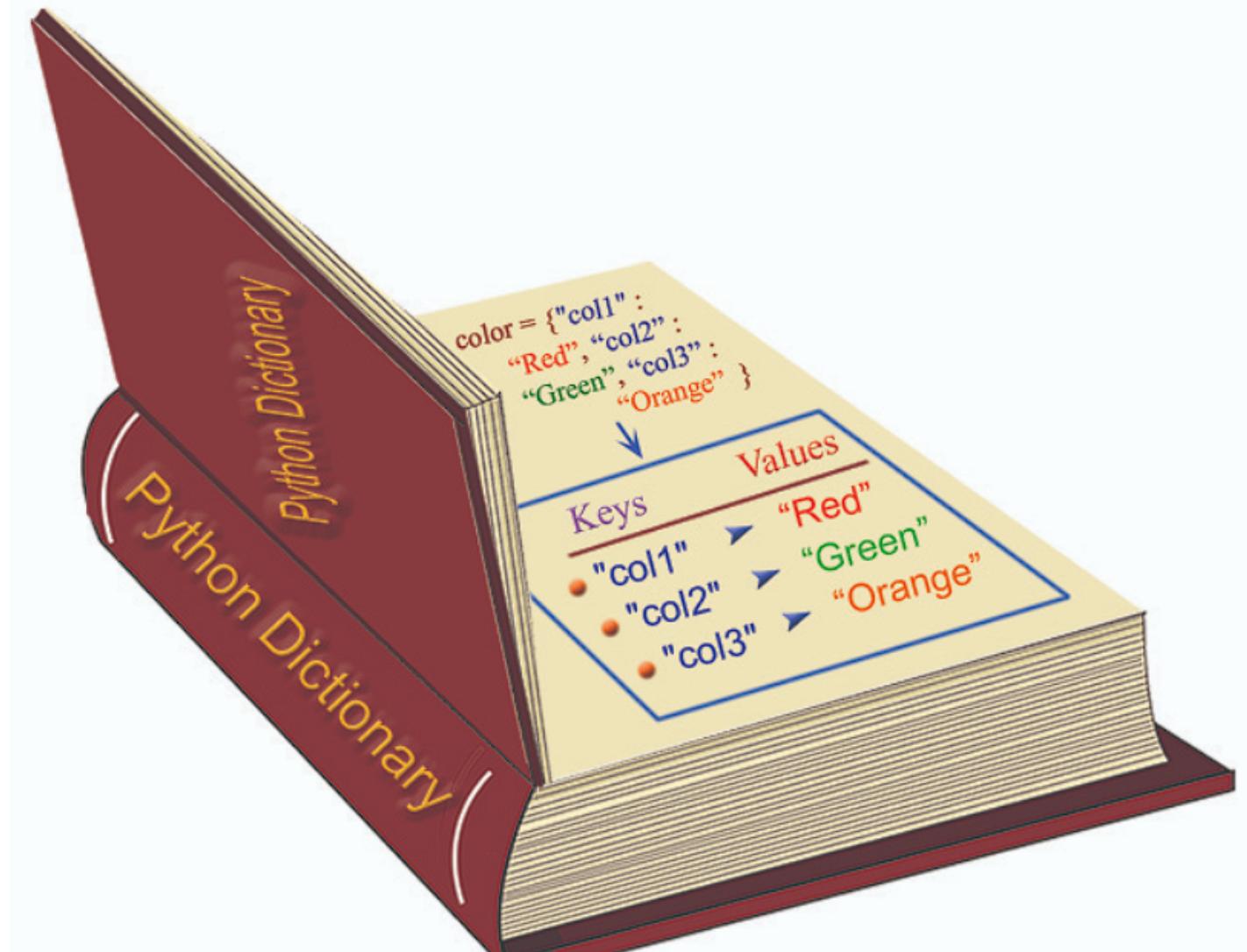


Python dictionary is a container of the unordered set of objects like lists. The objects are surrounded by curly braces { }. The items in a dictionary are a comma-separated list of key:value pairs where keys and values are Python data type.

Each object or value accessed by key and keys are unique in the dictionary. As keys are used for indexing, they must be the immutable type (string, number, or tuple). You can create an empty dictionary using empty curly braces. values can be assigned and accessed using square brackets[].



```
dis={'name':'red','age':10}  
print(dis) #will output all the key-value pairs. {'name':'red','age':10}
```

```
{'name': 'red', 'age': 10}
```

```
print(dis['name']) #will output only value with 'name' key. 'red'
```

```
red
```

```
print(list(dis.values())) #will output list of values in dic. ['red',10]
```

```
['red', 10]
```

```
print(dis.keys())
```

```
dict_keys(['name', 'age'])
```

It is also known as *Mapping* in Python. As they are primarily used for referencing items by key, they are not sorted.

▼ Rules for creating a dictionary:

1. Every key must be unique (otherwise it will be overridden)
2. Every key must be hashable (can use the hash function to hash it; otherwise TypeError will be thrown).
3. There is no particular order for the keys.

Creating a dict

Dictionaries can be initiated in many ways:

1. literal syntax

```
d = {} # empty
d = {'key': 'value'} # dict with initial values
```

2. dict comprehension

```
d = {k:v for k,v in [('key', 'value',)]}
d = { 1: "Mukta" for k,v in[('key', 'value')] }
d
{1: 'Mukta'}
```

3. built-in class: dict()

```
d = dict() # empty dict
d = dict(key='value') # explicit keyword arguments
d = dict([('key', 'value')]) # passing in a list of key/value pairs

di = dict(Name = 'Diksha')
di
```

{'Name': 'Diksha'}

```
d = dict([('Name', 'Diksha'), ('Age', '21')])
d
```

{'Name': 'Diksha', 'Age': '21'}

Modifying a dict

```
#To add items to a dictionary, simply create a new key with a value:
d['H.No'] = 42
#It also possible to add list and dictionary as value:
d['new_list'] = [1, 2, 3]
d['new_dict'] = {'nested_dict': 1}

#To delete an item, delete the key from the dictionary:
del d['newkey']
d
```

{'Name': 'Diksha',
 'Age': '21',
 'H.No': 42,
 'new_list': [1, 2, 3],
 'new_dict': {'nested_dict': 1}}

▼ Built-in Dictionary Methods

In Python Dictionary we have various built-in functions that provide a wide range of operations for working with dictionaries. These techniques enable efficient manipulation, access, and transformation of dictionary data.

1. Dictionary clear() Method

The clear() method in Python is a built-in method that is used to remove all the elements (key-value pairs) from a dictionary. It essentially empties the dictionary, leaving it with no key-value pairs.

```
my_dict = {'1': 'MUKTA', '2': 'DIKSHA', '3': 'NISHTHA'}
my_dict.clear()
print(my_dict)

{}
```

2. Dictionary get() Method

In Python, the `get()` method is a pre-built dictionary function that enables you to obtain the value linked to a particular key in a dictionary. It is a secure method to access dictionary values without causing a `KeyError` if the key isn't present.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
print(d.get('Name'))
print(d.get('Gender'))
print(d.get('Country'))
```

```
Ram
None
India
```

3. Dictionary items() Method

In Python, the `items()` method is a built-in dictionary function that retrieves a view object containing a list of tuples. Each tuple represents a key-value pair from the dictionary. This method is a convenient way to access both the keys and values of a dictionary simultaneously, and it is highly efficient.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
print((d.items()))
print(list(d.items())[1][1])
print(list(d.items())[1])

dict_items([('Name', 'Ram'), ('Age', '19'), ('Country', 'India')])
19
('Age', '19')
```

4. Dictionary keys() Method

The `keys()` method in Python returns a view object with dictionary keys, allowing efficient access and iteration.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
print(d.keys())
print(list(d.keys()))
```

```
dict_keys(['Name', 'Age', 'Country'])
['Name', 'Age', 'Country']
```

5. Dictionary values() Method

The values() method in Python returns a view object containing all dictionary values, which can be accessed and iterated through efficiently.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
print(list(d.values()))
```

```
['Ram', '19', 'India']
```

6. Dictionary update() Method

Python's update() method is a built-in dictionary function that updates the key-value pairs of a dictionary using elements from another dictionary or an iterable of key-value pairs. With this method, you can include new data or merge it with existing dictionary entries.

```
d1 = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
d2 = {'Name': 'Neha', 'Age': '22'}
```

```
d1.update(d2)
print(d1)
```

```
{'Name': 'Neha', 'Age': '22', 'Country': 'India'}
```

7. Dictionary pop() Method

In Python, the pop() method is a pre-existing dictionary method that removes and retrieves the value linked with a given key from a dictionary. If the key is not present in the dictionary, you can set an optional default value to be returned.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
d.pop('Age')
print(d)
```

```
{'Name': 'Ram', 'Country': 'India'}
```

8. Dictionary popitem() Method

In Python, the `popitem()` method is a dictionary function that eliminates and returns a random (key, value) pair from the dictionary.

As opposed to the `pop()` method which gets rid of a particular key-value pair based on a given key, `popitem()` takes out and gives back a pair without requiring a key to be specified.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India', 'Gender':'Male'}  
d.popitem()  
print(d)
```

```
{'Name': 'Ram', 'Age': '19', 'Country': 'India'}
```

```
d.popitem()  
print(d)
```

```
d.popitem()  
print(d)
```

```
{}
```

```
-----  
KeyError Traceback (most recent call last)  
<ipython-input-59-583be89c38c7> in <cell line: 4>()  
      2 print(d)  
      3  
----> 4 d.popitem()  
      5 print(d)
```

```
KeyError: 'popitem(): dictionary is empty'
```

[EXPLAIN ERROR](#)

✓ PRACTICE QUESTIONS:

1. Basic Dictionary Operations:

- Create a dictionary named `student` with keys "name," "age," and "grade," and assign appropriate values.
- Print the value associated with the "age" key.

- Update the "grade" to a new value.
- Add a new key-value pair for "subject" and its corresponding value.

2. Dictionary Manipulation:

- Create two dictionaries, `dict1` and `dict2`, with at least three key-value pairs each.
- Merge these dictionaries into a new dictionary called `merged_dict`.
- Remove a key from `merged_dict`.
- Check if a specific key exists in `dict1`.

3. Iterating Through a Dictionary:

- Use a loop to print all keys in the `student` dictionary.
- Use another loop to print all values in the `student` dictionary.
- Write a loop to print each key-value pair in the `student` dictionary.

4. Dictionary Comprehension:

- Create a dictionary comprehension to generate a dictionary of squares from 1 to 10.
- Filter the above dictionary to include only even squares.

5. Nested Dictionaries:

- Create a dictionary named `school` with multiple students (nested dictionaries).
- Access and print information about a specific student within the `school` dictionary.

6. Dictionary Functions:

- Use the `len()` function to find the number of items in a dictionary.
- Use the `keys()`, `values()`, and `items()` methods on a dictionary and print the results.

7. Sorting a Dictionary:

- Create a dictionary with unsorted keys.
- Use the `sorted()` function to print the keys in alphabetical order.

8. Dictionary Methods:

- Use the `get()` method to retrieve a value with a default if the key doesn't exist.
- Use the `pop()` method to remove and return a specific key-value pair.

Sets:

1. Set Operations:

- Create two sets, `set1` and `set2`, with some common elements.
- Find and print the union of the two sets.
- Determine the intersection of the sets.
- Check if one set is a subset of the other.

2. Set Manipulation:

- Add an element to `set1`.
- Remove an element from `set2`.
- Clear all elements from one of the sets.

3. Set Comprehension:

- Create a set comprehension that generates a set of squares from 1 to 10.
- Filter the above set to include only even squares.

Dictionaries:

4. Dictionary Operations:

- Create a dictionary named `book` with keys "title," "author," and "year," and assign appropriate values.
- Print the author of the book.
- Update the year of the book to a new value.
- Remove the "title" key from the dictionary.

5. Nested Dictionaries:

- Create a dictionary named `library` with multiple books (nested dictionaries).
- Access and print information about a specific book within the `library` dictionary.

Lists:

6. List Operations:

- Create a list named `fruits` with at least five different fruit names.
- Add a new fruit to the list.
- Remove a specific fruit from the list.
- Reverse the order of the elements in the list.

7. List Comprehension:

- Create a list comprehension that generates a list of cubes from 1 to 10.
- Filter the above list to include only even cubes.

Tuples:

8. Tuple Operations:

- Create a tuple named `colors` with at least three different color names.
- Access and print the second element of the tuple.
- Try to change the value of one element in the tuple and explain the result.

9. Mixed Sequences:

- Create a list that contains a mix of sets, dictionaries, lists, and tuples.
- Access and print an element from each of these sequences within the list.