Python Dictionary

A Python dictionary is a collection of items, similar to lists and tuples. However, unlike lists and tuples, each item in a dictionary is a **key-value** pair (consisting of a key and a value).

Create a Dictionary

We create a dictionary by placing `key: value` pairs inside curly brackets `{}`, separated by commas. For example,

```
# creating a dictionary
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
  "England": "London"
}

# printing the dictionary
print(country_capitals)
```

Output

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London'}
```

The `country_capitals` dictionary has three elements (key-value pairs), where `'Germany'` is the key and `'Berlin'` is the value assigned to it and so on.

Python Dictionary

Notes:

- Dictionary keys must be immutable, such as tuples, strings, integers, etc. We cannot use mutable (changeable) objects such as lists as keys.
- We can also create a dictionary using a Python built-in function `dict()`. To learn more, visit Python dict().

Valid and Invalid Dictionaries

Keys of a dictionary must be immutable

Immutable objects can't be changed once created. Some immutable objects in Python are integer, tuple and string.

```
# valid dictionary
# integer as a key
my_dict = {1: "one", 2: "two", 3: "three"}

# valid dictionary
# tuple as a key
my_dict = {(1, 2): "one two", 3: "three"}

# invalid dictionary
# Error: using a list as a key is not allowed
my_dict = {1: "Hello", [1, 2]: "Hello Hi"}

# valid dictionary
# string as a key, list as a value
my_dict = {"USA": ["Chicago", "California", "New York"]}
```

In this example, we have used integers, tuples, and strings as keys for the dictionaries. When we used a list as a key, an error message occurred due to the list's mutable nature.

Note: Dictionary values can be of any data type, including mutable types like lists.

Keys of a dictionary must be unique

The keys of a dictionary must be unique. If there are duplicate keys, the later value of the key overwrites the previous value.

```
hogwarts_houses = {
    "Harry Potter": "Gryffindor",
    "Hermione Granger": "Gryffindor",
    "Ron Weasley": "Gryffindor",
    # duplicate key with a different house
    "Harry Potter": "Slytherin"
```

```
}

print(hogwarts_houses)
```

Output

```
{'Harry Potter': 'Slytherin', 'Hermione Granger': 'Gryffindor', 'Ron Weasley': 'Gryffindor'}
```

Here, the key `Harry Potter` is first assigned to `Gryffindor`. However, there is a second entry where `Harry Potter` is assigned to `Slytherin`.

As duplicate keys are not allowed in a dictionary, the last entry `Slytherin` overwrites the previous value `Gryffindor`.

Access Dictionary Items

We can access the value of a dictionary item by placing the key inside square brackets.

```
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
  "England": "London"
}

# access the value of keys
print(country_capitals["Germany"])    # Output: Berlin
print(country_capitals["England"])    # Output: London
```

Note: We can also use the get() method to access dictionary items.

Add Items to a Dictionary

We can add an item to a dictionary by assigning a value to a new key. For example,

```
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
}

# add an item with "Italy" as key and "Rome" as its value
country_capitals["Italy"] = "Rome"


print(country_capitals)
```

Output

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'Italy': 'Rome'}
```

Remove Dictionary Items

We can use the del statement to remove an element from a dictionary. For example,

```
country_capitals = {
```

```
  "Germany": "Berlin",
  "Canada": "Ottawa",
}

# delete item having "Germany" key
del country_capitals["Germany"]


print(country_capitals)
```

Output

```
{'Canada': 'Ottawa'}
```

Note: We can also use the pop() method to remove an item from a dictionary.

If we need to remove all items from a dictionary at once, we can use the clear() method.

```
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
}

# clear the dictionary
country_capitals.clear()


print(country_capitals)
```

Output

```
{}
```

Change Dictionary Items

Python dictionaries are mutable (changeable). We can change the value of a dictionary element by referring to its key. For example,

```
country_capitals = {
  "Germany": "Berlin",
  "Italy": "Naples",
  "England": "London"
}

# change the value of "Italy" key to "Rome"
country_capitals["Italy"] = "Rome"


print(country_capitals)
```

Output

```
{'Germany': 'Berlin', 'Italy': 'Rome', 'England': 'London'}
```

Note: We can also use the update() method to add or change dictionary items.

Iterate Through a Dictionary

A dictionary is an ordered collection of items (starting from Python 3.7), therefore it maintains the order of its items.

We can iterate through dictionary keys one by one using a for loop.

```
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome"
}

# print dictionary keys one by one
for country in country_capitals:
    print(country)

print()

# print dictionary values one by one
for country in country_capitals:
    capital = country_capitals[country]
```

```
    print(capital)
```

Output

```
United States
Italy

Washington D.C.
Rome
```

Find Dictionary Length

We can find the length of a dictionary by using the len() function.

```
country_capitals = {"England": "London", "Italy": "Rome"}

# get dictionary's length
print(len(country_capitals))   # Output: 2


numbers = {10: "ten", 20: "twenty", 30: "thirty"}

# get dictionary's length
print(len(numbers))            # Output: 3


countries = {}

# get dictionary's length
print(len(countries))          # Output: 0
```

Python Dictionary Methods

Here are some of the commonly used dictionary methods.

Function	Description
pop()	Removes the item with the specified key.
update()	Adds or changes dictionary items.
clear()	Remove all the items from the dictionary.
keys()	Returns all the dictionary's keys.
values()	Returns all the dictionary's values.
get()	Returns the value of the specified key.
popitem()	Returns the last inserted key and value as a tuple.
copy()	Returns a copy of the dictionary.

Dictionary Membership Test

We can check whether a key exists in a dictionary by using the `in` and `not in` operators.

```
file_types = {
    ".txt": "Text File",
    ".pdf": "PDF Document",
    ".jpg": "JPEG Image",
}

# use of in and not in operators
print(".pdf" in file_types)       # Output: True
print(".mp3" in file_types)       # Output: False
print(".mp3" not in file_types)   # Output: True
```

Note: The `in` operator checks whether a key exists; it doesn't check whether a value exists or not.

Table of Contents

- Add Items to a Dictionary
- Remove Dictionary Items
- Change Dictionary Items
- Iterate Through a Dictionary
- Find Dictionary Length
- Python Dictionary Methods
- Dictionary Membership Test