Python

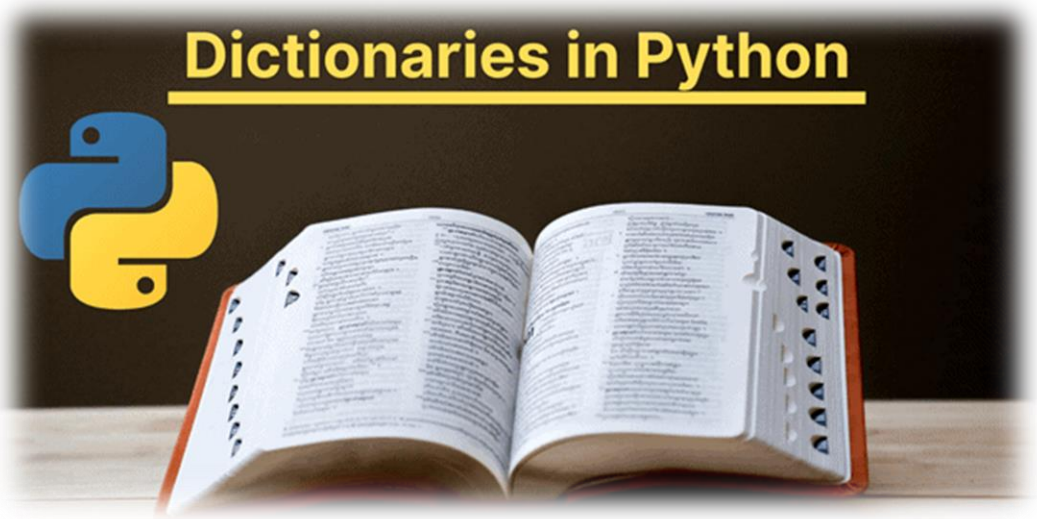
Full stack Skills Bootcamp

Introducing Python Dictionaries

What are Dictionaries?

A dictionary in Python is like a real-life dictionary. Instead of words and definitions, it stores keys and values.

- Keys: Unique identifiers (like words).
- Values: Information associated with each key (like definitions).
- Think of a contact list. Each name (key) has a phone number (value) associated with it.



Why use Dictionaries?

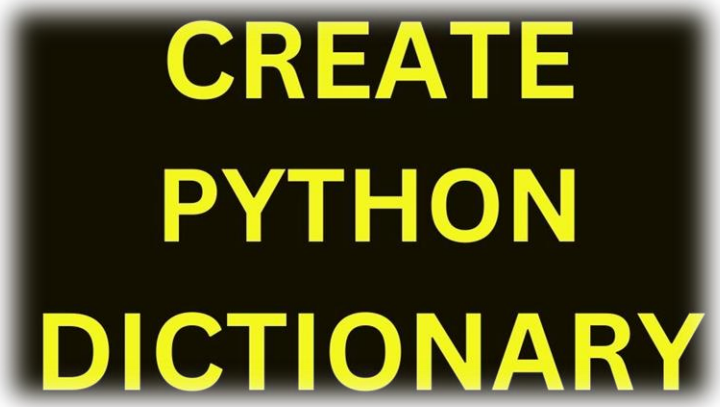
- Unordered: No fixed order to the items.
- Mutable: Can change, add, or remove key-value pairs.
- No duplicate keys: Each key can appear only once.

Creating a Dictionary

Code Example

```python
my_dict = {
    "name": "Alice",  # "name" is the key, "Alice" is the value
    "age": 25,
    "city": "New York"
}
```

- “name”, “age”, and “city” are the keys.
- “Alice”, 25, and “New York” are the values associated with the keys.



Analogy

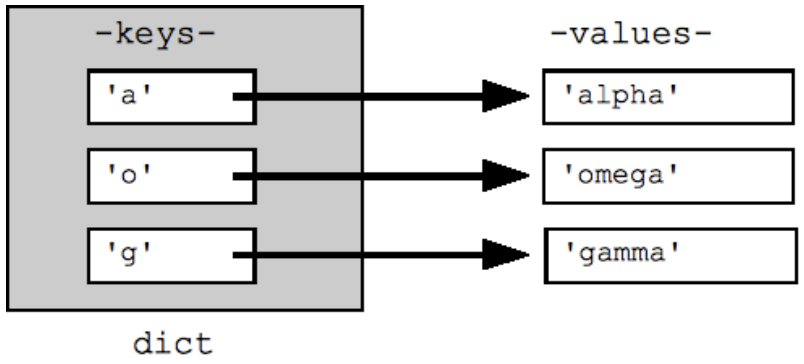
- This is like a profile for Alice, with her name, age, and city stored in a dictionary.

Accessing Dictionary Values

How to Access Values from a Dictionary?

To get a value, you use the key inside square brackets [ ].

```python
print(my_dict["name"])  # Output: Alice
print(my_dict["age"])   # Output: 25
```



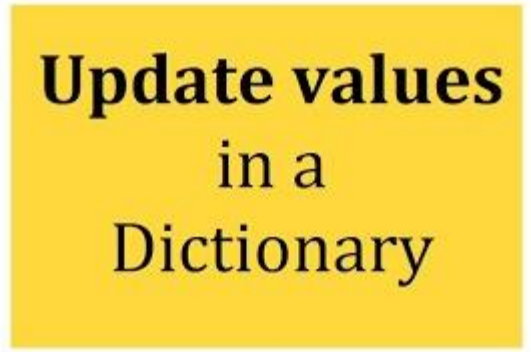
- You can use the key “name” to get the value “Alice”.
- It’s like looking up a word in a dictionary to get its definition.
- Accessing a key that doesn’t exist will cause an error (Key Error).

Updating Values in a Dictionary

How to Update Values in a Dictionary ?

You can change the value associated with a key by assigning a new value.

```python
my_dict["age"] = 26  # Changes the value of "age" to 26
print(my_dict)
```



- The key “age” is updated from 25 to 26.
- Like updating a person’s phone number in your contact list.
- You can change values, but the keys remain the same unless you delete or add a new key.

Adding New Key-Value Pairs

How to Add New Information

You can add new key-value pairs by assigning a value to a new key.

```python
my_dict["email"] = "alice@example.com"
print(my_dict)
```

```
dictionary = {
        key : value,
        key : value,
        key : value
}
```

- A new key “email” with the value alice@example.com is added to the dictionary.
- Imagine adding a new field like "email" to someone’s profile in your contact list.

.

Removing Key-Value Pairs

- How to Remove Information:

  Use the del keyword to delete a key-value pair from the dictionary.

```python
del my_dict["city"]
print(my_dict)
```

The key “city” and its value “New York” are removed from the dictionary. Like deleting a person's address from their contact profile.

:

Iterating Through a Dictionary

- How to Loop Through Keys, Values, and Items:

  You can iterate through keys, values, or key-value pairs in a dictionary using loops.

```python
# Looping through keys
for key in my_dict.keys():
    print(key)

# Looping through values
for value in my_dict.values():
    print(value)

# Looping through key-value pairs
for key, value in my_dict.items():
    print(key, value)
```

:

- .keys() gives you all the keys, .values() give you all the values, and .items() gives both keys and values.
- Like going through a contact list to see all the names, phone numbers, or both.

.

Conclusion

- Key Points
  - Dictionaries store related data as key-value pairs.
  - They are mutable, meaning you can change them after they are created.
  - You can add, update, and remove key-value pairs easily.
  - Iterating through dictionaries lets you work with keys and values in various ways