Dictionary in Python

Dipesh Thapa

October 16, 2024

1 Introduction

A dictionary in Python is an unordered collection of key-value pairs, where each key must be unique and immutable. Dictionaries are mutable, meaning that their contents can change.

1.1 Creating a Dictionary

A dictionary can be created using curly braces or the `dict()` constructor.

```
# Using curly braces
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

# Using dict() constructor
my_dict = dict(name='Alice', age=30, city='New York')
```

1.2 Properties of Dictionaries

- Keys must be unique and immutable (e.g., strings, numbers, tuples).
- Values can be of any type and can be duplicated.
- Dictionaries are mutable, allowing for dynamic changes.

2 Common Operations on Dictionaries

2.1 Adding a Key-Value Pair

You can add a new key-value pair to a dictionary using square brackets.

```
my_dict['occupation'] = 'Engineer'
```

2.2 Accessing Values

Values can be accessed using their corresponding keys.

```
name = my_dict['name']  # Returns 'Alice'
```

2.3 Updating Values

To update the value associated with a specific key:

```
my_dict['age'] = 31  # Update age to 31
```

2.4 Removing Key-Value Pairs

Use `del` or the `pop()` method to remove items.

```
del my_dict['city']  # Removes the key 'city'
age = my_dict.pop('age')  # Removes 'age' and returns its value
```

2.5 Checking if a Key Exists

You can check if a key exists in a dictionary using the `in` keyword.

```
if 'name' in my_dict:
    print("Name␣exists.")
```

2.6 Getting the Number of Key-Value Pairs

Use the `len()` function to get the number of items.

```
num_items = len(my_dict)  # Returns the number of key-value pairs
```

2.7 Getting All Keys and Values

You can retrieve all keys or values using the `keys()` and `values()` methods.

```
keys = my_dict.keys()    # Returns a view of all keys
values = my_dict.values() # Returns a view of all values
```

2.8 Iterating Over a Dictionary

You can iterate over keys, values, or key-value pairs using a for loop.

```
for key in my_dict:
    print(key, my_dict[key])  # Iterating over keys

for value in my_dict.values():
    print(value)               # Iterating over values

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

2.9 Merging Dictionaries

You can merge two dictionaries using the `update()` method or the `**` unpacking method (Python 3.5+).

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
dict1.update(dict2)  # Merges dict2 into dict1
# or
merged_dict = {**dict1, **dict2}  # Merges both into a new dictionary
```

2.10 Finding the Intersection of Two Dictionaries

To find keys that exist in both dictionaries, you can use set operations.

```
intersection_keys = set(dict1.keys()) & set(dict2.keys())
```

2.11 Nested Dictionaries

Dictionaries can contain other dictionaries, allowing for complex data structures.

```
nested_dict = {
    'person': {
        'name': 'Alice',
        'age': 30,
    },
    'job': {
        'title': 'Engineer',
        'company': 'Tech␣Corp'
    }
}
# Accessing nested values
name = nested_dict['person']['name']
```

3 Handling Mutable Objects as Keys

Mutable objects (like lists) cannot be used as dictionary keys because they are unhashable. Always use immutable types (strings, numbers, tuples) as keys.