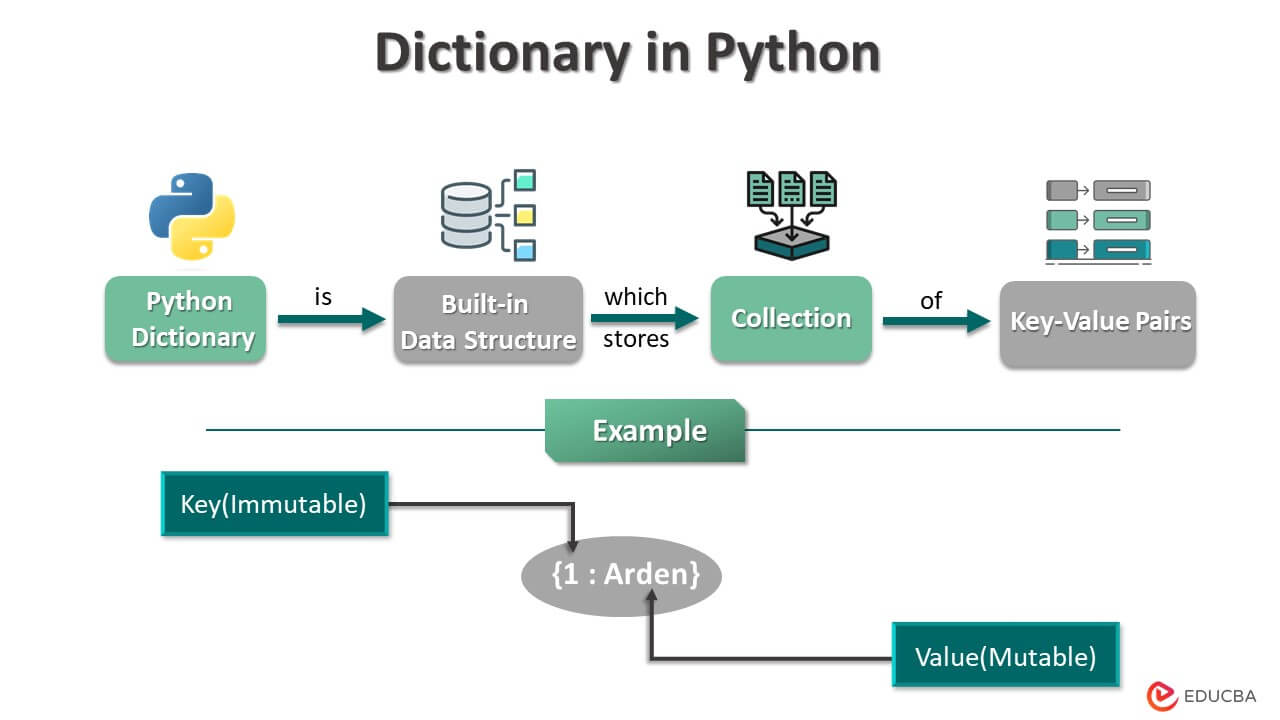
**Dictionary**

In Python, a dictionary is a mutable, unordered collection of key-value pairs. Each key is unique and immutable, and it maps to a corresponding value. Dictionaries are enclosed in curly braces **{}**, and each key-value pair is separated by a colon :.



### **Key Characteristics:**

* **Mutable**: Dictionaries can be modified after creation. You can add, update, or remove key-value pairs.
* **Unordered**: Unlike lists, dictionaries are unordered collections, meaning that the order of elements is not guaranteed.
* **Keys are Unique:** Each key in a dictionary must be unique. If you attempt to add a duplicate key, the existing value will be overwritten.
* **Keys are Immutable:** Dictionary keys must be immutable objects, such as strings, numbers, or tuples.
* **Values Can Be Mutable:** Dictionary values can be of any data type and can be mutable or immutable.
* **1. Creating Dictionaries:**

Dictionaries in Python are created using curly braces `{}` and consist of key-value pairs separated by commas.

```python

**# Creating dictionaries**

empty\_dict = {}

student = {'name': 'John', 'age': 25, 'grade': 'A'}

```

**2. Accessing Elements:**

Dictionary elements are accessed using keys, similar to how elements in a list are accessed using indices.

```python

student = {'name': 'John', 'age': 25, 'grade': 'A'}

# Accessing elements using keys

print(student['name']) # Output: John

print(student['age']) # Output: 25

```

**3. Dictionary Operations:**

- **Adding or Updating Elements:**

New elements can be added to a dictionary, or existing elements can be updated by assigning values to their keys.

```python

student = {'name': 'John', 'age': 25}

# Adding a new key-value pair

student['grade'] = 'A'

# Updating an existing value

student['age'] = 26

```

**- Removing Elements:**

Elements can be removed from a dictionary using the `del` keyword or the `pop()` method.

```python

student = {'name': 'John', 'age': 25, 'grade': 'A'}

# Removing a key-value pair

del student['grade']

# Removing and returning the value of a specific key

age = student.pop('age')

```

**- Membership Testing:**

You can check if a key exists in a dictionary using membership testing.

```python

student = {'name': 'John', 'age': 25}

print('age' in student) # Output: True

print('grade' in student) # Output: False

```

**4. Dictionary Methods:**

**`keys()`, `values()`, and `items()`:**

These methods return views of the dictionary's keys, values, and key-value pairs, respectively.

```python

student = {'name': 'John', 'age': 25}

# Getting keys

print(student.keys()) # Output: dict\_keys(['name', 'age'])

# Getting values

print(student.values()) # Output: dict\_values(['John', 25])

# Getting key-value pairs

print(student.items()) # Output: dict\_items([('name', 'John'), ('age', 25)])

```

**get():**

This method returns the value associated with a specified key. If the key does not exist, it returns a default value (or `None`).

```python

student = {'name': 'John', 'age': 25}

# Getting the value for the 'grade' key

grade = student.get('grade', 'Not available')

```

**Use Cases:**

1. **Data Storage**: Dictionaries are commonly used to store and retrieve data based on key-value pairs.

2. **Configuration Settings**: Dictionaries can be used to store configuration settings for applications, where each setting is identified by a unique key.

3. **Mapping Relationships**: Dictionaries are ideal for representing mappings between entities, such as mapping student names to their grades.

4. **JSON Serialization**: Dictionaries closely resemble JSON objects and are often used to represent data that needs to be serialized/deserialized.

5. **Counting and Frequency Analysis**: Dictionaries are useful for counting occurrences of elements in a dataset and performing frequency analysis.

Understanding dictionaries and their methods will empower you to efficiently manage and manipulate key-value data structures in Python.