Dictionary Basics

Table of contents

01

Introduction

Introduction

- A **dictionary** in Python is a versatile built-in data structure that holds **key-value pairs**, allowing fast retrieval of values based on keys.
- Think of a dictionary book – it maps **words (keys)** to their **meanings (values)**.

Characteristics:

Unordered/Ordered	Items do not follow a fixed order (until Python 3.6). Python 3.7+ retains **insertion order**.
Mutable	You can **modify** the contents: add, delete, or change elements.
Indexed	**Keys** act like indexes to retrieve **values** directly.
Unique Keys	No two keys can be the same. Duplicate keys will overwrite earlier ones.
Heterogeneous	Keys must be immutable (like str, int, tuple), but values can be of any type.

02

Creating Dictionary

Creating Dictionaries – Using braces {}

Dictionary can be created using **curly braces {}** with **comma-separated key-value pairs**, where each **key** is followed by a **colon ‘ : ’** and mapped to its corresponding **value**.

Syntax

```
{ key1: value1, key2: value2, ... }
```

Python

```
employee = {"name": "Ravi", "age": 28, "department": "HR"}
print(employee)
print(type(employee))
```

Output

```
{'name': 'Ravi', 'age': 28, 'department': 'HR'}
<class 'dict'>
```

Creating Dictionaries – Using dict()

You can use the built-in **dict()** function, especially when keys are valid identifiers (no spaces or special characters)

Python

```
employee = dict(name="Ravi", age=28, department="HR")
print(employee)
print(type(employee))
```

Output

```
{'name': 'Ravi', 'age': 28, 'department': 'HR'}
<class 'dict'>
```

Creating Dictionaries – Using zip()

- **Combines two lists**: one as keys, one as values.
- **Length** of both lists should ideally be **equal**.

Python

```python
keys = ["id", "name", "score"]
values = [101, "Arjun", 89]
record = dict(zip(keys, values))
print(record)
```

Output

```
{'id': 101, 'name': 'Arjun', 'score': 89}
```

03

Accessing Elements

Accessing Elements

In Python, dictionary elements are accessed using **keys**, not numerical indexes like lists or tuples. Each key in a dictionary maps directly to its associated value, and there are **two primary methods** to access these values:

1. Using Square Brackets [ ]
2. Using **get()** Method

Python

```
student = {"name": “Amar", "age": 20, "course": “Python"}
print(student["name"])
```

Output

```
Amar
```

⚠️

Important: If the key does not exist, this method raises a KeyError.

Accessing Elements

In Python, dictionary elements are accessed using **keys**, not numerical indexes like lists or tuples. Each key in a dictionary maps directly to its associated value, and there are **two primary methods** to access these values:

1. Using **Square Brackets [ ]**
2. Using get() Method

The **get()** method is a **safer** way to access dictionary elements. It returns **None** (or a **default value** if specified) when the key is not found, thus avoiding a crash.

Python	Output

`student = {"name": “Amar", "age": 20, "course": “Python"}`
`print(student.get("age"))`
`print(student.get("grade"))`
`print(student.get("grade", "N/A"))`
	20
	None
	N/A

Accessing Elements

You can also access **all keys** using **.keys()**, **all values** using **.values()**, and **both** using **.items()** for iteration.

Python

```
student = {"name": “Amar", "age": 20, "course": “Python"}
for key in student:
    print(f"{key} → {student[key]}")  # Access via key

for value in student.values():
    print(value)  # Access all values

for key, value in student.items():
    print(f"{key}: {value}")  # Access both key and value
```

Output

```
name → Amar
age → 20
course → Python

Amar
20
Python

name: Amar
age: 20
course: Python
```

Important Points

- Keys are **case-sensitive**: **"Name"** and **"name"** are different.
- Always use **get()** when you're unsure if the key exists to prevent runtime errors.
- You can also access all **keys** using **.keys()**, all **values** using **.values()**, and **both** using **.items()** for iteration.

04

Modifying Elements

Modifying Elements - direct

Dictionaries are **mutable**, which means their contents—i.e., key-value pairs—can be **changed, added, or removed** after creation. Modification can happen in two main ways:

1. **Updating an existing value**
2. Inserting a new key-value pair.

To update the **value** associated with a **particular key**, simply assign a new value using the existing key

Python	Output

`book = {"title": "1984", "author": "Orwell", "price": 350}`
`book["price"] = 299`
`print(book)`
	`{'title': '1984', 'author': 'Orwell', 'price': 299}`

Modifying Elements – update()

Dictionaries are **mutable**, which means their contents—i.e., key-value pairs—can be **changed, added, or removed** after creation. Modification can happen in two main ways:

1. **Updating an existing value**
2. Inserting a new key-value pair.

You can update **multiple keys** at once using the **update()** method. If a **key exists**, it **updates** the value; if it **doesn't**, it adds the **new key-value pair**.

Python	Output

`book = {"title": "1984", "author": "Orwell", "price": 350}`
`book.update({"price": 320, "pages": 328})`
`print(book)`
	`{'title': '1984', 'author': 'Orwell', 'price': 320, 'pages': 328}`

Modifying Elements - Inserting

Dictionaries are **mutable**, which means their contents—i.e., key-value pairs—can be **changed, added, or removed** after creation. Modification can happen in two main ways:

1. Updating an existing value
2. **Inserting a new key-value pair.**

If the **key** does **not already exist** in the dictionary, assigning a value to it will automatically add a new entry.

Python	Output

`book = {"title": "1984", "author": "Orwell", "price": 350}`
`book["publisher"] = "Penguin"`
`print(book)`
	`{'title': '1984', 'author': 'Orwell', 'price': 350, 'publisher': 'Penguin'}`

Important Points

- Modifying a **key** is **not possible** directly; you must **delete** the **key** and **re-add** it.
- **Adding** and **updating** are performed the same way: **dict[key] = value**
- Use **update()** for **batch changes** or **merging** another dictionary.

05

Removing Elements

Removing Elements

- Python provides **multiple ways** to remove elements from a dictionary.
- Since **dictionaries** are **mutable**, you can delete **specific key-value pairs**, remove the **last inserted item**, or even **clear the entire dictionary**.

Removing Elements – pop()

Removes the item with the **specified key** and **returns its value**. If the key does not exist, it raises a **KeyError**, unless a default value is provided.

Python	Output

`car = {"brand": "Toyota", "model": "Corolla", "year": 2020}`
`year_removed = car.pop("year")`
`print(year_removed)`
`print(car)`
	`2020`
	`{'brand': 'Toyota', 'model': 'Corolla'}`

Removing Elements – popitem()

- Removes and returns the **last inserted key-value pair** as a tuple. It's useful when you want to treat the dictionary like a stack (**LIFO**).
- Raises **KeyError** if the dictionary is empty.

Python	Output

`car = {"brand": "Toyota", "model": "Corolla", "year": 2020}`
`last_item = car.popitem()`
`print(last_item)`
`print(car)`
	`('year', 2020)`
	`{'brand': 'Toyota', 'model': 'Corolla'}`

Removing Elements – clear()

Removes **all items** from the dictionary, making it empty.

Python	Output

`car = {"brand": "Toyota", "model": "Corolla", "year": 2020}`
`car.clear()`
`print(car)`
	`{}`

Removing Elements – del

Removes a **specific key-value pair** using the key name. If the key doesn't exist, it raises a **KeyError**.

Python	Output

`car = {"brand": "Toyota", "model": "Corolla", "year": 2020}`
`del car["model"]`
`print(car)`
	`{'brand': 'Toyota', 'year': 2020}`

Important Points

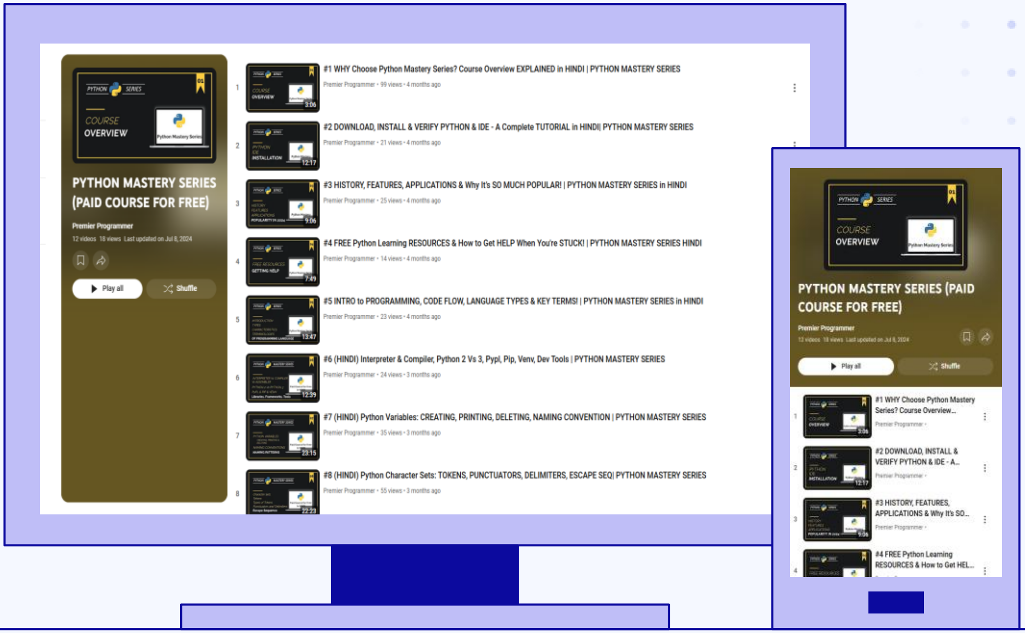
- Keys must be immutable types like strings, numbers, or tuples. Using lists or other mutable types as keys raises an error
- When defining a dictionary with duplicate keys, only the last assignment survives

Practice Set - 1

Download Link in **Description** and **Pinned Comment**

WATCH

Level up your coding with each episode in this focused Python series.



PYTHON MASTERY SERIES (PAID COURSE FOR FREE)

Premier Programmer

12 videos 18 views Last updated on Jul 8, 2024

1 #1 WHY Choose Python Mastery Series? Course Overview EXPLAINED in HINDI | PYTHON MASTERY SERIES
Premier Programmer • 99 views • 4 months ago

2 #2 DOWNLOAD, INSTALL & VERIFY PYTHON & IDE - A Complete TUTORIAL in HINDI| PYTHON MASTERY SERIES
Premier Programmer • 21 views • 4 months ago

3 #3 HISTORY, FEATURES, APPLICATIONS & Why It's SO MUCH POPULAR! | PYTHON MASTERY SERIES in HINDI
Premier Programmer • 25 views • 4 months ago

4 #4 FREE Python Learning RESOURCES & How to Get HELP When You're STUCK! | PYTHON MASTERY SERIES HINDI
Premier Programmer • 14 views • 4 months ago

5 #5 INTRO to PROGRAMMING, CODE FLOW, LANGUAGE TYPES & KEY TERMS! | PYTHON MASTERY SERIES in HINDI
Premier Programmer • 23 views • 4 months ago

6 #6 (HINDI) Interpreter & Compiler, Python 2 Vs 3, Pypl, Pip, Venv, Dev Tools | PYTHON MASTERY SERIES
Premier Programmer • 24 views • 3 months ago

7 #7 (HINDI) Python Variables: CREATING, PRINTING, DELETING, NAMING CONVENTION | PYTHON MASTERY SERIES
Premier Programmer • 35 views • 3 months ago

8 #8 (HINDI) Python Character Sets: TOKENS, PUNCTUATORS, DELIMITERS, ESCAPE SEQ| PYTHON MASTERY SERIES
Premier Programmer • 55 views • 3 months ago

Next Video!

Practice Set - 1
Solution