PART 4 OF 8

Python Self-Learning Booklet 4: Dictionaries and More Loops

In this session you will learn how to store data using **dictionaries** (key-value pairs) and how to loop through them in different ways – using `.items()`, `.keys()`, and `.values()`. You will also learn how to remove duplicates with `set()` and sort output with `sorted()`.

📘 **Session** Dictionaries & More Loops	🎯 **Concepts** 12 core concepts
🛠️ **Practice** 5 hands-on tasks	🔍 **Explore** 3 open questions

LEARNING GOALS

By the End of This Session, You Will Be Able To...

1. **Create dictionaries**

   Use the correct `{key: value}` syntax to build a dictionary.

2. **Access, add, modify & delete**

   Read values by key, add new pairs, update existing ones, and remove entries with `del`.

3. **Avoid KeyErrors safely**

   Use `.get()` to look up keys that might not exist without crashing.

4. **Loop through dictionaries**

   Use `.items()`, `.keys()`, and `.values()` to iterate in different ways.

5. **Clean up output**

   Use `set()` to print unique values and `sorted()` for alphabetical order.

CONCEPT 1

Why Dictionaries? Lists vs. Dictionaries

Lists — Access by Position

Lists use a number (index) to find items. You need to remember *where* something is stored.

```
fruits = ["apple", "banana", "cherry"]
print(fruits[0])  # apple
```

Prints the first fruit using index `0`.

Dictionaries — Access by Name

Dictionaries use a **key** (a label) to find items. You look something up by *what it is called*.

```
person = {"name": "Ali", "age": 20}
print(person["name"])  # Ali
```

Prints the value connected to the key `"name"`. No index needed!

💡 Use a **list** when order matters. Use a **dictionary** when you want to look something up by a meaningful label.

CONCEPT 2

Dictionary Syntax: Keys and Values

A dictionary uses curly braces `{ }`, colons `:` between keys and values, and commas `,` to separate pairs.

```
d = {"key 1": "value 1", "key 2": "value 2"}
print(d["key 2"])
```

Curly Braces `{ }`

Wrap the whole dictionary.

Colon `:`

Separates each key from its value.

Comma `,`

Separates each key-value pair.

Unique Keys

You cannot have the same key twice in one dictionary.

CONCEPT 3

Accessing Values by Key (No Index!)

To get a value from a dictionary, write the dictionary name followed by square brackets containing the **key** – not a number.

The Code

```
student = {"name": "Ali", "age": 20}
print(student["age"])
```

What It Does

Prints `20`. You must use `"age"` (the key) – not a number like `student[1]`.

Remember: dictionaries have no position number. Keys are the only way in.

⚠️ If you try `student[0]` on a dictionary, Python will raise a **KeyError** – unless `0` happens to be an actual key.

CONCEPT 4

Adding and Modifying Data

Dictionaries are **dynamic** – you can add new keys or change existing values at any time by assigning to a key.

```
student = {"name": "Ali", "age": 20}

student["major"] = "Software Engineering"  # add new key
student["age"] = 21                        # modify existing key

print(student)
```

Adding a new key

If the key `"major"` does not exist yet, Python creates it and assigns the value.

Modifying an existing key

If the key `"age"` already exists, the old value (`20`) is replaced with the new one (`21`).

CONCEPT 5

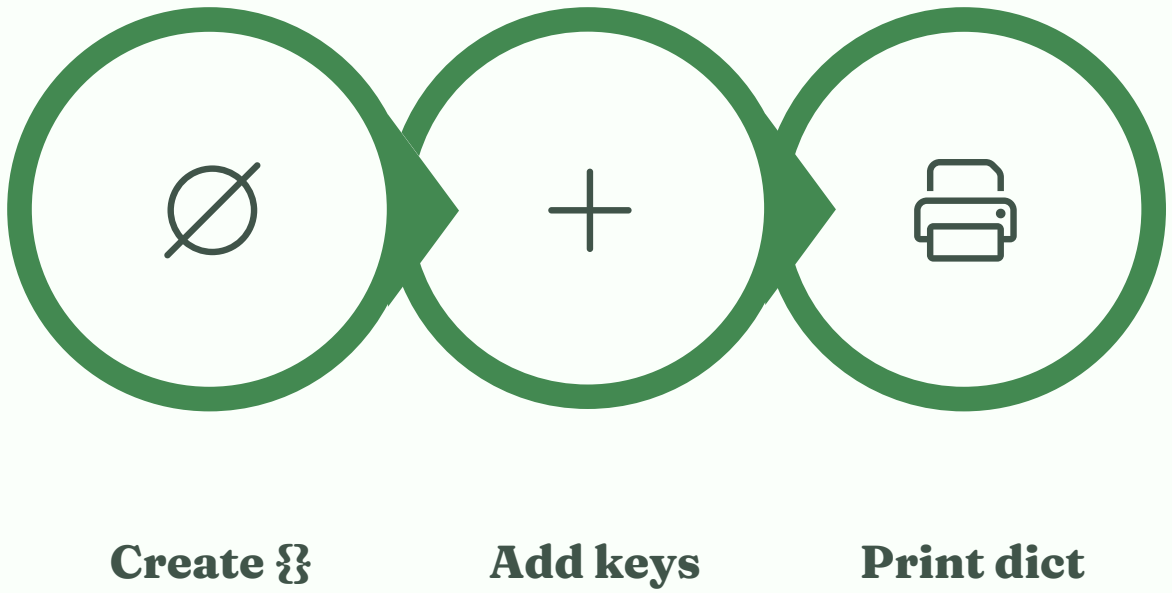
Starting With an Empty Dictionary

You don't need to fill a dictionary all at once. Start empty with `{}` and add entries as you go – great for building data step by step.

```
person = {}

person["first_name"] = "Lina"
person["last_name"] = "Chen"
person["age"] = 19

print(person)
```



Each assignment adds a new key - value pair until the dictionary is complete.

CONCEPT 6

Deleting Key–Value Pairs with `del`

Use the `del` keyword to **permanently remove** a key and its value from a dictionary.

The Code

```
alien = {"color": "green",
         "points": 5}

del alien["points"]

print(alien)
```

What It Does

The key `"points"` and its value `5` are removed. The dictionary now only contains `{'color': 'green'}`.

Important: `del` is permanent – there is no undo. Make sure you really want to remove it!

CONCEPT 7

KeyError and Safe Access with `.get()`

Accessing a key that doesn't exist causes a **KeyError** and crashes your program. The `.get()` method is a safe alternative.

❌ Unsafe — causes a crash

```
alien = {"color": "green",
         "points": 5}
print(alien["planet"])  # KeyError!
```

✅ Safe — use `.get()`

```
alien = {"color": "green",
         "points": 5}

planet = alien.get("planet",
    "No planet assigned.")
print(planet)
```

Since `"planet"` is missing, `.get()` returns the default message instead of crashing.

💡 **Syntax:** `dictionary.get("key", "default message")` – the second argument is optional but recommended.

CONCEPT 8

Looping Through Key–Value Pairs with .items()

When you need **both** the key and the value in your loop, use `.items()`. It returns each pair as a tuple of two variables.

```
student = {"name": "Ali", "age": 21, "major": "Software Engineering"}

for key, value in student.items():
    print(key, value)
```

Output

```
name Ali
age 21
major Software Engineering
```

How It Works

Each loop iteration unpacks one pair: `key` gets the key name, `value` gets the associated value.

Best Used When

You want to display or process both sides of each pair – for example, printing a summary of a student's profile.

CONCEPT 9

Looping Through Keys Only with `.keys()`

If you only need the **key names**, loop with `.keys()`. This is clear and explicit for beginners.

The Code

```
student = {"name": "Ali",
           "age": 21,
           "major": "Software Engineering"}

for key in student.keys():
    print(key)
```

Output & Notes

Prints: `name` `age` `major` – one key per line.

You could also write `for key in student:` and get the same result – but `.keys()` makes your intention clearer.

CONCEPT 10

Looping Through Values Only with `.values()`

Use `.values()` when you only care about the **data stored**, not the labels (keys).

```
student = {"name": "Ali", "age": 21, "major": "Software Engineering"}

for value in student.values():
    print(value)
```

`.items()`

Key **and** value together

`.keys()`

Key names **only**

`.values()`

Values **only**

The code above prints: `Ali`, `21`, `Software Engineering` – one value per line.

CONCEPT 11

Avoid Duplicates with `set()`

Sometimes multiple keys share the same value. Wrapping `.values()` in `set()` removes duplicates so each value prints only once.

```
favorite_languages = {
    "jen": "python",
    "sarah": "c",
    "edward": "rust",
    "phil": "python",
}

for language in set(favorite_languages.values()):
    print(language.title())
```

Without `set()`

Python, C, Rust, Python – `"python"` appears **twice**.

With `set()`

Python, C, Rust – each language appears only **once**. Order is not guaranteed.

CONCEPT 12

Sort Keys Alphabetically with `sorted()`

Dictionaries do not automatically loop in alphabetical order. Wrap `.keys()` (or `.values()`) in `sorted()` to get ordered output.

```
student = {"name": "Ali", "age": 21, "major": "Software Engineering"}

for key in sorted(student.keys()):
    print(key)
```

1

Original order

`name` → `age` → `major`

2

After `sorted()`

`age` → `major` → `name`

💡 `sorted()` does not change the dictionary itself – it only affects the order used *during that loop*.

STEP-BY-STEP EXAMPLE

Putting It All Together: Favorite Foods

This example combines creating a dictionary, modifying it, using `.get()`, looping with `sorted()`, and removing duplicates with `set()`.

```
favorite_foods = {
    "Sara": "pizza",
    "Ali": "noodles",
    "Lina": "tacos",
}

# Add one more friend (with a repeated food)
favorite_foods["Mina"] = "pizza"

# Change an existing value
favorite_foods["Ali"] = "dumplings"

# Safely access a key that might not exist
unknown = favorite_foods.get("Sam", "No food saved for this person.")
print(unknown)

print("\nPeople and their favorite foods (sorted):")
for name in sorted(favorite_foods.keys()):
    print(f"{name} likes {favorite_foods[name]}.")

print("\nUnique foods:")
for food in set(favorite_foods.values()):
    print(food)
```

STEP-BY-STEP LOGIC

How the Example Works — Step by Step

01

Create the dictionary

Each **key** is a person's name; each **value** is their favorite food.

02

Add a new entry

`favorite_foods["Mina"] = "pizza"` adds a new key-value pair. Mina's food is also pizza – a deliberate duplicate.

03

Modify an existing entry

`favorite_foods["Ali"] = "dumplings"` replaces `"noodles"` with `"dumplings"`.

04

Safe lookup with `.get()`

Since `"Sam"` is not in the dictionary, `.get()` returns the default message without crashing.

05

Sorted loop

`sorted(favorite_foods.keys())` prints names in alphabetical order: Ali, Lina, Mina, Sara.

06

Unique foods with `set()`

`set(favorite_foods.values())` ensures `"pizza"` only appears once in the output.

COMMON MISTAKES

Watch Out: 5 Common Dictionary Mistakes

1 — Using an index instead of a key

❌ `person[0]` ✅ `person["name"]`

Dictionaries don't have numbered positions.

2 — Forgetting quotes around string keys

❌ `person[name]` ✅ `person["name"]`

Without quotes, Python looks for a variable called `name`.

3 — Crashing with a missing key

❌ `person["nickname"]` → KeyError

✅ `person.get("nickname", "No nickname")`

4 — Mixing up loop methods

`.items()` → key + value | `.keys()` → keys only | `.values()` → values only

5 — Expecting alphabetical order

Dictionaries do not loop alphabetically by default.

Use `sorted(d.keys())` when you need ordered output.

PRACTICE TASKS

Practice Tasks

Work through these tasks in order. Each one builds on the previous. Try to write the code yourself before checking any notes.

1

Student Dictionary

Create a dictionary `student` with keys `"name"` and `"age"`. Print: *My name is ... and I am ... years old.*

2

Add and Modify

Add `"major"`, `"city"`, and `"hobby"` to `student`. Change the value of one existing key. Print the full dictionary.

3

Delete

Remove one key from `student` using `del`. Print the dictionary again.

4

Favorite Numbers

Make a dictionary with 3 people and their favorite number. Use a loop to print each person and their number.

5

Nested Dictionary Mini-Project

Create `people = {"Ali": {"country": "...", "major": "..."}, ...}`. Loop through it and print: *Ali is from X and studies Y.*

EXPLORATION QUESTIONS

Go Further: Exploration Questions

These questions don't have a single right answer. Try them out in Python and observe what happens – this is how programmers learn!

🔀 Question 1 — Order of set()

Run `set(favorite_foods.values())` several times. Does the order of foods always print the same? What do you notice?

📋 Question 2 — Value as a List

Add a person whose favorite food is a **list**, e.g. `["pizza", "salad"]`. How would you print that nicely inside your loop?

🔐 Question 3 — .get() With and Without a Default

Try `.get("missing_key")` with no second argument. What does Python return? How is it different from providing a default message?

SUMMARY

Session Summary

Here's everything you learned in Session 4 at a glance. Use this as a quick reference while practising.

Concept	Quick Reminder
Create a dictionary	`{"name": "Ali", "age": 20}` – curly braces, colon, comma
Access a value	`d["key"]` – use the key, not an index number
Add or modify	`d["key"] = value` – works for both new and existing keys
Delete an entry	`del d["key"]` – permanently removes the pair
Safe access	`d.get("key", "default")` – no crash if key is missing
Loop: key + value	`for k, v in d.items():`
Loop: keys only	`for k in d.keys():`
Loop: values only	`for v in d.values():`
Remove duplicates	`set(d.values())` – prints each unique value once
Alphabetical order	`sorted(d.keys())` – orders output A→Z

🎉 **Well done!** You've completed Session 4 of 8. Up next: more advanced data structures and loops. Keep practising!