Dictionaries and Sets

A dictionary stores pairs of keys and values, like a word and its meaning — and a set stores unique values only, with duplicates removed automatically.

Introduction

A list finds items by position: first, second, third. But sometimes position is the wrong way to look something up. You do not find a word in a real dictionary by its position — you find it by the word itself. Python's **dictionary** works the same way: every value has a **key**, and you look the value up by its key.

For example, a student record could be `{"name": "Ali", "age": 20, "marks": 85}`. Here `"name"` is a key and `"Ali"` is its value. To get the age, you ask for the key `"age"`, and Python returns 20. Keys are usually strings, and each key appears only once.

A **set** is a different kind of group. It holds values with no order and no duplicates. If you put the same value in twice, the set keeps it once. That makes sets perfect for questions like "which items are unique?" or "what appears in both groups?"

(Roman Urdu: Dictionary mein har value ki aik key hoti hai — jaisay lafz aur us ka matlab. Set mein har value sirf aik dafa hoti hai, duplicate khud hi khatam ho jatay hain.)

Two things trip beginners here. First, a dictionary lookup that fails crashes the program — so you must decide, every time, whether the key is guaranteed to exist (use square brackets) or might be missing (use `.get()`). Second, a set looks like a dictionary with no values, but it behaves completely differently: no order, no indexing, no duplicates. Treat them as different tools, not cousins.

Key points

- **A dictionary uses curly brackets with key-value pairs.** Format: `{"key": value, "key": value}`.
- **Look up a value by its key.** `student["name"]` gives `"Ali"`. Missing keys give an error, so use `.get()` for safety.
- **`.get()` returns `None` (or your default) for a missing key.** `student.get("city", "Unknown")` never crashes on a missing key.
- **Adding or changing is the same step.** `student["city"] = "Lahore"` adds the key if it is new, or replaces the value if it already exists.
- **`.keys()` and `.values()` list all keys and all values.** `.items()` gives both together as pairs, which is handy in loops.
- **Dictionary keys must be unique.** Writing the same key twice keeps only the last value.
- **A set uses curly brackets with values only.** Example: `{"red", "blue", "green"}`. An empty set needs `set()`, because `{}` makes an empty dictionary.

- **Sets remove duplicates automatically.** `{"a", "b", "a"}` becomes `{"a", "b"}`.
- **Sets have no order and no indexing.** You cannot write `my_set[0]`; loop over a set instead, or test membership with `in`.
- **`.update()` merges another dictionary in.** `student.update({"city": "Lahore"})` adds or replaces several keys at once.
- **Set operations answer group questions.** `&` is the overlap (in both), `|` is everything combined, and `-` is what is only in the first set.

Examples

Example 1 — A student dictionary

```
student = {"name": "Ali", "age": 20, "marks": 85}
print(student["name"])
print(student["marks"])
student["city"] = "Lahore"
print(student)
print(student.get("phone"))
print(student.get("phone", "Not given"))
```

What you will get:

```
Ali
85
{'name': 'Ali', 'age': 20, 'marks': 85, 'city': 'Lahore'}
None
Not given
```

Direct lookup with `student["phone"]` would crash here, because the key does not exist. `.get()` is the safe version.

Example 2 — Looping over keys, values and pairs

```
student = {"name": "Ayesha", "age": 19, "marks": 92}
print(list(student.keys()))
print(list(student.values()))

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

What you will get:

```
['name', 'age', 'marks']
['Ayesha', 19, 92]
name: Ayesha
age: 19
marks: 92
```

`list()` is used here only to show the keys and values neatly. In a normal loop you can use `.items()` directly, as in the second part.

Example 3 — A set removes duplicates

```
colours = {"red", "blue", "red", "green", "blue"}
print(colours)
print(len(colours))
print("red" in colours)
print("yellow" in colours)

numbers = [1, 2, 2, 3, 3, 3, 4]
unique = set(numbers)
print(unique)
print(len(unique))
```

What you will get (set order may vary):

```
{'red', 'blue', 'green'}
3
True
False
{1, 2, 3, 4}
4
```

The set keeps each colour once, even though red and blue were added twice. Converting a list to a set is the fastest way to find its unique values. Note that set order is not guaranteed, so do not rely on it.

Example 4 — Updating values and finding the top mark

Dictionaries are living records: marks change, new keys arrive, and you often need the biggest value rather than one student's mark:

```
marks = {"Ali": 85, "Ayesha": 92, "Bilal": 78}
marks["Bilal"] = 88
print(marks)
print("Top mark:", max(marks.values()))
for name in marks:
    print(name, marks[name])
```

What you will get:

```
{'Ali': 85, 'Ayesha': 92, 'Bilal': 88}
Top mark: 92
Ali 85
Ayesha 92
Bilal 88
```

Looping directly over a dictionary visits its keys, in the order they were added. `max(marks.values())` looks only at the values, ignoring the names.

Example 5 — Set operations: who plays both games?

Two clubs keep sets of members. Set operations answer the natural questions — who is in both, who is in either, who is only in one — in a single expression:

```
cricket = {"Ali", "Bilal", "Sara"}
football = {"Bilal", "Sara", "Usman"}
print(cricket & football)
print(cricket | football)
print(cricket - football)
```

What you will get (set order may vary):

```
{'Bilal', 'Sara'}
{'Ali', 'Bilal', 'Sara', 'Usman'}
{'Ali'}
```

Bilal and Sara play both games; all four names play at least one; only Ali plays cricket and not football. Because set order is not guaranteed, your printed order may differ — the membership is what matters.

Example 6 — Real-world case: shop stock and unique words

A shopkeeper tracks stock in a dictionary: each item name maps to how many are left. Selling 5 packs of sugar is one subtraction on the right key. And a writer checking a draft counts unique words by turning the word list into a set:

```
stock = {"tea": 20, "sugar": 15, "rice": 10}
stock["sugar"] = stock["sugar"] - 5
print(stock)
print("Sugar left:", stock.get("sugar"))
words = ["apple", "banana", "apple", "mango", "banana"]
print("Unique words:", len(set(words)))
```

What you will get:

```
{'tea': 20, 'sugar': 10, 'rice': 10}
Sugar left: 10
Unique words: 3
```

Dictionaries for "look this up by name", sets for "how many different ones" — that split covers most real uses of both.

Quick recap

- A dictionary maps keys to values: `{"name": "Ali"}`.
- Look up with `dict[key]`; use `.get()` to avoid crashes on missing keys.
- `.keys()`, `.values()` and `.items()` let you loop over a dictionary.
- A set holds unique values only, with no order and no indexing.
- `set(list)` removes duplicates from a list.
- `.update()` merges keys into a dictionary.
- Set `&`, `|` and `-` give the overlap, the combination, and the difference.

Common mistakes

1. **Using a missing key with square brackets** — `student["phone"]` crashes if the key is absent. Fix: use `student.get("phone", "Not given")`.
2. **Writing `{}` when you want an empty set** — `{}` is an empty dictionary, not a set. Fix: write `set()` for an empty set.
3. **Trying to index a set** — `my_set[0]` gives an error because sets have no order. Fix: loop over the set, or convert it to a list first with `list(my_set)`.
4. **Forgetting quotes around string keys** — `{name: "Ali"}` treats `name` as a variable, not as text. Fix: write `{"name": "Ali"}`.
5. **Expecting a set to keep order** — sets do not promise any order. Fix: if order matters, keep the data in a list.

1. **Using a list as a dictionary key** — `{["a"]: 1}` fails because keys must be unchangeable (hashable), and lists can change. Fix: use a tuple key like `{("a",): 1}`, or a string.

2. **Writing a duplicate key and losing data silently** — `{"a": 1, "a": 2}` keeps only 2, with no error. Fix: check for the key first, or build the dictionary step by step so duplicates are visible.

Practice

1. Make a dictionary for yourself: name, age, city and favourite subject. Print your city using its key, then add a new key for your hobby and print the whole dictionary.
2. Try looking up a key that does not exist, first with square brackets (read the error), then with `.get()` and a default message. Write down the difference.
3. Make a list with repeated numbers, for example `[5, 5, 10, 10, 15]`. Convert it to a set and print both the set and its length.
4. Loop over your dictionary from task 1 with `.items()` and print each pair as "key: value".
5. Make a phonebook dictionary with three names and numbers. Add a fourth person with `.update()`, change one number, and print the whole phonebook.
6. Two classes have student sets: class A `{"Ali", "Sara", "Bilal"}` and class B `{"Sara", "Usman"}`. Print the students in both classes and the students only in class A.
7. Make a dictionary of three subjects and your marks. Loop over it with `.items()` and print each subject with "Pass" if the mark is 50 or more, otherwise "Fail".

Practice quiz

1. What is a dictionary in Python?

- A list of words
- Pairs of keys and values, so you look things up by name
- A set of numbers
- A file

Answer: Pairs of keys and values, so you look things up by name

2. What does ages.get("Ali", "not found") do if "Ali" is missing?

- Crashes the program
- Returns "not found" instead of crashing
- Adds Ali to the dictionary
- Returns 0

Answer: Returns "not found" instead of crashing

3. What do sets automatically remove?

- Numbers
- Duplicate values
- Text

- Empty strings only

Answer: Duplicate values

Watch the video lesson: https://youtu.be/TlJrcr3z12o

Read online: https://scopetreasure.com/notes/py-dicts-sets · More free tools and notes at https://scopetreasure.com