Python Programming
Unit 02 – Lecture 03 Notes
Sets and Dictionaries

Tofik Ali

February 14, 2026

Contents

1 Lecture Overview

This lecture introduces two extremely practical collection types:

- **sets** for uniqueness and fast membership checking, and
- **dictionaries** for key-value modeling and frequency counting.

Students should leave with a clear mental model: *use a set when you care about "is it present?" and uniqueness; use a dictionary when you care about "for this key, what value should I store?"*

2 Sets

2.1 What Is a Set?

A set is an **unordered** collection of **unique** elements. When you add duplicates, they are automatically removed.

2.2 Creating Sets and Testing Membership

```
colors = {"red", "blue", "red"}
print(colors) # duplicates removed
print("red" in colors) # membership check

s = set([1, 2, 2, 3])
empty_set = set() # NOT {}
```

Important: `{}` creates an empty dictionary, not an empty set.

2.3 Set Operations (Set Algebra)

Given two sets `A` and `B`:

- Union: `A | B` (elements in either)
- Intersection: `A & B` (elements common to both)
- Difference: `A - B` (in A but not in B)
- Symmetric difference: `A ^ B` (in exactly one of them)

Example:

```
A = {1, 2, 3}
B = {2, 3, 4}
A | B # {1, 2, 3, 4}
A & B # {2, 3}
A - B # {1}
A ^ B # {1, 4}
```

2.4 When Sets Are the Right Tool

Sets shine in problems like:

- removing duplicates (de-duplication),
- checking if something has been seen before, and
- computing overlap between two groups.

Membership tests in sets are typically fast (average-case constant time), which is why sets are preferred for repeated `in` checks.

2.5 Pitfalls with Sets

- **No indexing:** sets are unordered, so `s[0]` is invalid.
- **Elements must be hashable:** you cannot put a list inside a set.

If order is needed, convert to a sorted list:

```
s = {3, 1, 2}
ordered = sorted(s) # [1, 2, 3]
```

3 Dictionaries

3.1 What Is a Dictionary?

A dictionary maps **keys** to **values**. Keys are unique and must be immutable (hashable). Values can be any type.

3.2 Creation and Basic Operations

```
student = {"name": "Asha", "marks": 88}
student["name"] # access
student["marks"] = 90 # update

for k, v in student.items():
    print(k, v)
```

3.3 Access: `d[key]` vs `d.get(key)`

`d[key]` raises a `KeyError` if the key does not exist. `d.get(key)` returns `None` by default (or a custom default).

```
d = {"a": 1}
# d["b"] # KeyError

d.get("b") # None
d.get("b", 0) # 0
```

Rule of thumb:

- Use `d[key]` when missing keys are a real error.
- Use `get` when missing keys are expected.

3.4 Worked Pattern: Frequency Counting

Counting is one of the most common dictionary use cases.

```
text = "to be or not to be"
freq = {}
for w in text.split():
    freq[w] = freq.get(w, 0) + 1
```

The key idea is `get(w, 0)` which uses 0 for unseen words.

3.5 Sorting Dictionary Items

Dictionaries themselves are mappings; sorting creates a *list of pairs*:

```
counts = {"a": 3, "b": 1, "c": 2}
by_value = sorted(counts.items(), key=lambda kv: kv[1])
by_value_desc = sorted(counts.items(), key=lambda kv: kv[1], reverse=True)
```

The result is a list like `[("b", 1), ("c", 2), ("a", 3)]`.

4 Demo Walkthrough: Word Frequency and Unique Words

Script: `demo/word_frequency.py`

4.1 What the Demo Is Teaching

- Use a set to get the unique vocabulary.
- Use a dictionary to count word occurrences.
- (Extension) Sort counts to find top frequent words.

4.2 Suggested Run Steps

```
python demo/word_frequency.py
```

Suggested student activity: change the sentence and predict which word becomes most frequent.

5 Quiz and Solutions

5.1 Checkpoint 1 (Set Removes Duplicates)

Question. What is the result of `set([1,1,2,2])`?
Solution. `{1, 2}` (order does not matter).

5.2 Checkpoint 2 (`d[key]` vs `get`)

Question. What is the difference between `d[key]` and `d.get(key)`?
Solution. `d[key]` raises `KeyError` if the key is missing. `d.get(key)` returns `None` (or a provided default) and avoids the exception.

5.3 Think-Pair-Share (Set vs List for Membership)

Prompt. Choose set vs list for membership checks and justify.

Sample response. Use a set when you repeatedly check `x in data` for many `x`. A set is designed for fast membership; a list must scan values one by one, which becomes slow as the list grows.

5.4 Exit Question (Sort by Value Descending)

Question. Write one line to sort dictionary items by value in descending order.

Solution.

```
sorted(d.items(), key=lambda kv: kv[1], reverse=True)
```

6 Practice Exercises (With Solutions)

6.1 Exercise 1: Unique Words (Case-Insensitive)

Task. Given a sentence, print the number of unique words ignoring case.

Solution.

```
text = "Python python is FUN fun"
words = [w.lower() for w in text.split()]
unique = set(words)
print(len(unique))
```

6.2 Exercise 2: Top-3 Frequent Words

Task. From a text, print the top-3 words by frequency.

Solution.

```
text = "to be or not to be"
freq = {}
for w in text.split():
    freq[w] = freq.get(w, 0) + 1

top3 = sorted(freq.items(), key=lambda kv: kv[1], reverse=True)[:3]
print(top3)
```

6.3 Exercise 3: Jaccard Similarity

Task. Given two sets `A` and `B`, compute $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

Solution.

```
def jaccard(A, B):
    inter = A & B
    uni = A | B
    return len(inter) / len(uni) if uni else 0.0
```

7 Further Reading

- https://docs.python.org/3/tutorial/datastructures.html#sets
- https://docs.python.org/3/tutorial/datastructures.html#dictionaries