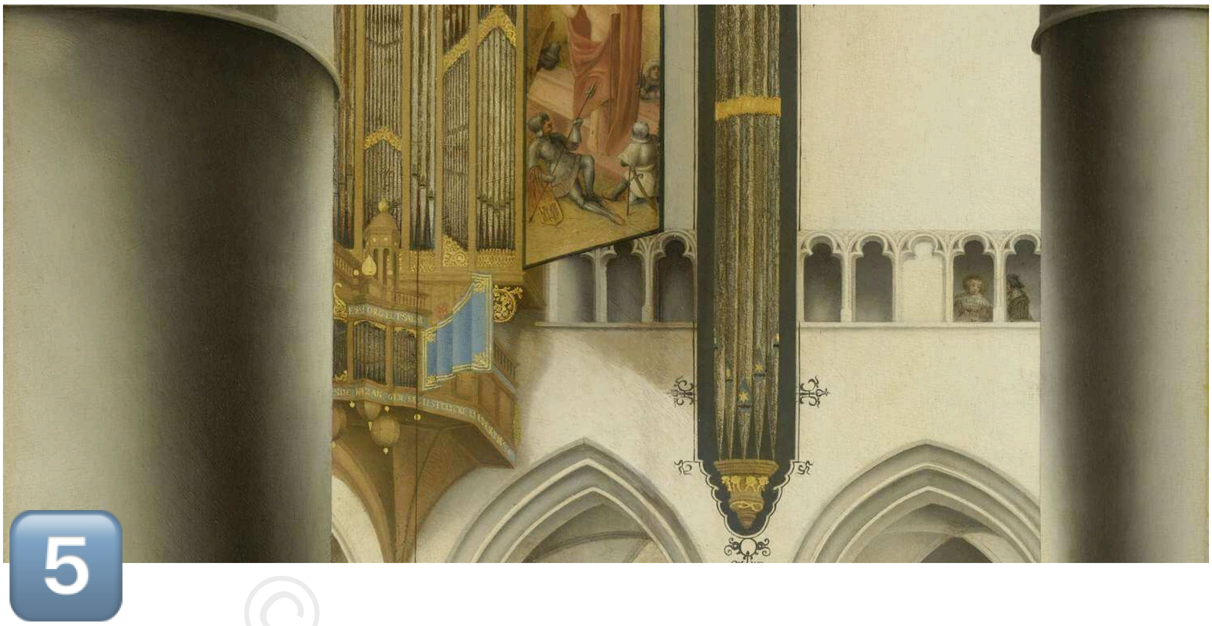

5

Chapter 5

© Prathamesh Nalge

✨ Chapter 5 Notes – Dictionaries & Sets

📘 Dictionary.py

```
marks = {
    "Prathamesh": 100,
    "Harry": 90,
    "Larry": 80
}

marks1 = {}  # This is an empty dictionary

print(marks, type(marks))        # Shows entire dictionary with type
print(marks["Harry"])            # Access value using a key
# print(marks[0])  ❌ Gives KeyError (dictionaries aren't accessed by index)
```

🧠 Key Concepts:

- Dictionary = `key: value` pairs
- Unordered and mutable

- Can’t use indexes like in lists

📘 Dictionary methods.py

```
marks = {
    "Prathamesh": 100,
    "Harry": 90,
    "Larry": 80
}

print(marks.items())      # All key-value pairs as tuples
print(marks.keys())       # Only keys
print(marks.values())     # Only values

marks.update({"Prathamesh": 99, "Lucy": 10})  # Add/update key-value pairs
print(marks)

print(marks.get("Tarry"))     # ✅ Returns None (safe)
print(marks["Harry2"])        # ❌ Gives KeyError if key doesn't exist
```

⚠️ Use `.get()` over direct key access to avoid crashing on missing keys.

© Prathamesh Nalge

📘 sets.py

```
# Set is a collection of **unique** elements
s = {1, 2, 3}
e = set()  # Correct way to create an empty set

# e = {} will create an empty dictionary, NOT a set

# Sets are:
# ✅ Unordered
# ✅ Unindexed
# ✅ Do NOT allow duplicates
```

Example:

```
s = {1, 1, 2, 3}
print(s)  # Output: {1, 2, 3} (removes duplicate 1)
```

📘 `setmethods.py`

```
s = {1, 2, 3, 4, 56, 5, 67, 62}

s.add(29323)       # Add new item
print(s)

s.pop()           # Remove a random item
print(s)

s.copy()          # Returns a copy of the set
```

📘 `setoperations.py`

```
s1 = {21, 342, 0, 28938, 928, 1, 2, 3}
s2 = {82948, 893, 123, 1, 2, 3, 0}

print(s1.union(s2))              # Combines elements from both
print(s1.intersection(s2))        # Common elements
print(s1.difference(s2))          # Elements in s1 but not in s2
print(s1.issubset(s2))            # Checks if s1 is subset of s2
print(s1.symmetric_difference(s2)) # Elements not common in both
```

© Prathamesh Nalge

🧪 **Set Operations Refresher:**

- `union()` – OR operation
- `intersection()` – AND operation
- `difference()` – Subtract one from another
- `symmetric_difference()` – XOR operation
- `issubset()` / `issuperset()` – Relationship checks

📘 Set Theory Essentials (Class 11 + Python 🐍)

Operation	Symbol	Python Method	Meaning / Formula
Union	$A \cup B$	`A.union(B)`	Elements in A **or** B
Intersection	$A \cap B$	`A.intersection(B)`	Elements **common** to both A and B
Difference	$A - B$	`A.difference(B)`	Elements in A but **not in B**
Symmetric Difference	$A \Delta B$	`A.symmetric_difference(B)`	Elements in A or B but **not both** (XOR)
Subset	$A \subseteq B$	`A.issubset(B)`	Every element of A is also in B
Superset	$A \supseteq B$	`A.issuperset(B)`	Every element of B is also in A
Complement	A'	No direct Python method	Elements **not in A** (use universal set)
Disjoint Sets	–	`A.isdisjoint(B)`	Returns `True` if sets **have no common elements**

© Prathamesh Nalge

📐 De Morgan's Theorems (Logic + Sets)

Set Theory Formula	Description
$(A \cup B)' = A' \cap B'$	Complement of Union is Intersection of Complements
$(A \cap B)' = A' \cup B'$	Complement of Intersection is Union of Complements

🧠 **In Python**, these require manually defining the **Universal Set**, then applying difference logic.

Example in Python:

```
U = {1, 2, 3, 4, 5, 6, 7, 8, 9}  # Universal Set
A = {1, 2, 3}
B = {3, 4, 5}

# Complement of A
A_comp = U.difference(A)

# (A ∪ B)' = A' ∩ B'
```

```
de_morgan1 = U.difference(A.union(B)) == A_comp.intersection(U.difference(B))

# (A ∩ B)' = A' ∪ B'
de_morgan2 = U.difference(A.intersection(B)) == A_comp.union(U.difference(B))
```

🧮 Quick Formulas

Formula	Meaning
n(A∪B)=n(A)+n(B)−n(A∩B)	Inclusion-Exclusion Principle
n(A∩B)′=n(U)−n(A∩B)	Complement of intersection
n(A′)=n(U)−n(A)n(A')	Complement of a set

> n(X) = number of elements in set X
>
> U = Universal Set

© Prathamesh Nalge

✅ Problem 1: Hindi-English Dictionary

```
a = {
    "pustak": "book",
    "khaana": "food",
    "pyaar": "love",
    "dost": "friend",
    "ghar": "home",
    "paani": "water",
    "khushi": "happiness",
    "suraj": "sun",
    "chand": "moon",
    "aasmaan": "sky"
}
b = input("Enter the hindi word: ")
print(a[b])
```

🔍 **Explanation**:

- You're building a dictionary with Hindi → English mappings.

- But there's a **problem**: If the user enters a word that’s not in the dictionary, it throws a **KeyError**.

✅ **Improvement**:

```
print(a.get(b, "Word not found in dictionary."))
```

`dict.get(key, default)` is **safe**. If the key isn't found, it won't crash your program.

✅ Problem 2: Unique Numbers in Set

```
set = {a,b,c,d,e,f,g,h}
```

🧠 Sets automatically **remove duplicates**. If the user enters the same number twice, it will still only appear once.

✅ Problem 3: `s = {18, '18'}`

🔍 **Explanation**:

- `18` → `int`
- `'18'` → `str`

  👉 They are **different data types**, so both will coexist.

✅ Output:

```
{18, '18'}  # Set length is 2
```

✅ Problem 4: `20` , `20.0` , `'20'`

```
s = set()
s.add(20)       # int
s.add(20.0)     # float
s.add('20')     # str
print(len(s))  # Output: 2
```

🧠 **Important Concept**:

- Python considers `20 == 20.0` → `True`

© Prathamesh Nalge

- So `set` only keeps one of them (since both are **numerically equal** even if types are different).
- `'20'` is a **string**, so it stays.

💡 This is because sets use **hashing**, and both `int(20)` and `float(20.0)` share the same hash if they're numerically equal.

✅ Problem 5: `s = {}`

```
s = {}
print(type(s))  # Output: <class 'dict'>
```

🧠 `{}` by default creates an **empty dictionary**, not a set.

✅ Correct way to make an empty set:

```
s = set()
```

✅ Problem 6: Favourite Language Dictionary

```
lang = {
    a:e,
    b:f,
    c:g,
    d:h
}
```

⚠️ **Warning**: If two users enter the **same name**, the earlier value will be **overwritten** because dictionary keys must be **unique**.

✅ Safer version (uses `.update()`):

```
lang = {}
lang.update({a:e})
lang.update({b:f})
lang.update({c:g})
lang.update({d:h})
```

© Prathamesh Nalge

📘 CHAPTER 5 SUMMARY – Dictionary and Sets

Concept	Explanation
Dictionary	Key-value pairs. Keys must be **unique** and **immutable**.
Dictionary Methods	`.keys()` , `.values()` , `.items()` , `.update()` , `.get()`
Accessing Dictionary	`dict[key]` → crashes if key doesn't exist. Use `.get(key, default)` to be safe
Dictionary Mutability	✅ Mutable (you can change/update items)
Set	Unordered, unindexed, and stores only **unique** items
Empty Set Syntax	Use `set()` instead of `{}`
Set Methods	`.add()` , `.remove()` , `.pop()` , `.clear()` , `.union()` , `.intersection()`
Set Behavior	Ignores duplicates. `{20, 20.0}` becomes `{20}` due to numerical equality
Hashing in Sets	Based on value equality (`20 == 20.0` is True, so one value is dropped)

© Prathamesh Nalge