PYTHON REVIEWER

COLLECTIONS MODULE

The collections module provides **specialized container datatypes** that extend Python's built-in containers.

Counter

- Subclass of dict.
- Counts elements in an iterable.
- Returns unordered dictionary → {element: count}.

OrderedDict

- Like a regular dictionary but **preserves insertion order**.
- From Python 3.7+, normal dict preserves order too, but OrderedDict offers extra features (e.g., moving keys to the end when reinserted).

DefaultDict

- Subclass of dict.
- Provides **default values** for missing keys instead of raising KeyError.

NamedTuple

- Like tuples, but with **named fields**.
- Access data using names (e.g., student.name), improving readability.

Deque

- Double-ended queue: fast append() and pop() from both ends.
- More efficient than lists (O(1) time complexity).

MATH MODULE

- Provides mathematical functions and constants.
- Includes square roots, powers, trigonometric functions, etc.
- Makes computations **faster and more accurate**.

RANDOM MODULE

- Generates random numbers or selects random items.
- Useful for **games, testing, and simulations**.

ZIPFILE MODULE

Used to create, read, write, append, and extract ZIP files.

Zipping Files

- **Without with**: Must manually open/close ZIP file (risk of corruption).
- **With with**: Automatically closes ZIP file safely.

Zipping Multiple Files

- Loop through files to compress many at once.

Zipping a Folder

- Compress entire folder (including subfolders) — ideal for backups.

➕ Appending to a ZIP File

- Use mode 'a' to add new files to an existing ZIP.

📤 Extracting Files

- Use mode 'r' to read ZIP contents.
- Can extract **all files** or **specific files**.

🔢 PYTHON NUMERIC TYPES

Type	Description	Example
int	Whole numbers	5, -10
float	Decimal numbers	3.14, -0.001
complex	Real + imaginary parts	2 + 3j

🔄 TYPE CONVERSION

Changing one data type into another.
Example: float(5) → 5.0, int(3.9) → 3.

➕ MATH OPERATIONS

Basic arithmetic: +, -, *, /, //, %, **

Example:

a = 5

b = 3

print(a + b)

🧠 PRECISION & RATIONAL NUMBERS

- Use decimal for high precision.
- Use fractions.Fraction for rational numbers.

🔢 COMPLEX NUMBER ARITHMETIC

Addition: (a+bi) + (c+di) = (a+c) + (b+d)i
Multiplication: (a+bi)(c+di) = (ac−bd) + (ad+bc)i

➗ FRACTION ARITHMETIC

Addition: a/b + c/d = (ad + cb) / bd
Multiplication: a/b * c/d = (a*c) / (b*d)

🔟 NUMBER SYSTEMS

Hexadecimal

- Base 16 → 0-9 and A-F
- Example: 0x1A

Binary

- Base 2 → 0 and 1
- Example: 0b1010

🧭 OTHER FUNCTIONS

Function	Description	Example
abs(x)	Absolute value	abs(-5) → 5
pow(x, y)	Power	pow(2, 3) → 8
round(x, n)	Rounds to n decimals	round(3.1416, 2) → 3.14

🧵 STRINGS

- A **sequence of characters** enclosed in ' ' or " ".
- **Immutable** — cannot be changed in place.

String Indexing & Slicing

- Indexing: str[0] (first character)
- Slicing: str[start:end:step]

Common String Methods

Method	Description
.strip()	Removes spaces at start/end
.replace(a, b)	Replaces substring
.lower() / .upper()	Converts case
.count(sub)	Counts occurrences
.find(sub)	Finds index of substring

Splitting & Joining Strings

- .split() → breaks string into list
- ' - '.join(list) → joins list into string

String Formatting

Insert variables into strings:

name = "Anna"

print("Hello, {}".format(name))

or

print("Hello, %s" % name)

🧩 SETS

- **Unordered, mutable collection of unique elements.**
- Supports **union, intersection, and difference.**

Creating Sets

s1 = {1, 2, 3}

s2 = set([3, 4, 5])

Advanced Set Operations

Operation	Description	Example
A & B	Intersection	Common elements
A - B	Difference	Unique to A
`A	B`	Union

⚙️ ADVANCED SET FEATURES

Set Comprehension

A concise one-liner for creating sets.
Syntax:

{expression for item in iterable if condition}

Automatically removes duplicates.

Immutable Sets (frozenset)

- Immutable version of a set.
- Cannot add, remove, or modify elements.
  Syntax:

fs = frozenset([1, 2, 3])

Membership Testing

Checks if an element exists in a set.

- Operator: in / not in

- **O(1)** time complexity.
  Example:

5 in {1, 2, 3, 5} # True

Subset, Superset, and Disjoint Testing

Method	Description
A.issubset(B)	True if A ⊆ B
A.issuperset(B)	True if A ⊇ B
A.isdisjoint(B)	True if A and B have no common elements

Mutating Methods

Modify a set in place.

Method	Description
.add(x)	Adds an element
.remove(x)	Removes an element (error if missing)
.discard(x)	Removes element (no error if missing)
.update(iterable)	Adds multiple elements

Tips & Tricks

- Use set operations for fast filtering.
- Use sets to remove duplicates.
- Avoid indexing (sets are unordered).
- Use frozenset for unchangeable sets.
- Use set comprehensions for cleaner code.

Common Mistakes:

- Creating an empty set → use set() not {} (which makes a dict).
- Use discard() instead of remove() when unsure if element exists.

🗂️ DICTIONARIES

Dictionaries are **mutable, unordered** collections of key-value pairs.

Key Characteristics

- Key-Value Pairs
- Mutable
- No Duplicate Keys
- Keys must be hashable

Example

my_dict = {"name": "Alice", "age": 20}

Dictionary Items

- Ordered and changeable.
- Access values using keys:

print(my_dict["name"])

Dictionary Length

Use len(my_dict) to count items.

Data Types

Values can be any data type (string, int, list, boolean, etc.).

Creating Dictionaries

my_dict = dict(name="Alice", age=20)

Accessing Dictionary Items

Method	Description
[]	Access using key
.get(key)	Safe access (returns None if missing)
.keys()	Returns all keys
.values()	Returns all values
.items()	Returns all key-value pairs

Changing or Adding Items

- Direct assignment: my_dict["age"] = 21
- .update() method to add multiple items

Removing Items

Method	Description
del	Removes specific key
.pop(key)	Removes and returns value
.popitem()	Removes last item
.clear()	Clears dictionary

Looping Through Dictionaries

Loop Type	Example
Keys	for k in dict:
Values	for v in dict.values():
Key-Value Pairs	for k,v in dict.items():

Copying Dictionaries

- .copy() → shallow copy
- dict() → creates a new dictionary

Nested Dictionaries

Dictionary inside another dictionary.

```
students = {
 "S1": {"name": "Alice", "age": 20},
 "S2": {"name": "Bob", "age": 22}
}
```

Access with:

```
students["S1"]["name"]
```

Dictionary Methods

Method	Description
.clear()	Removes all elements
.copy()	Returns a copy
.get(key)	Returns value of key
.items()	Returns key-value pairs
.keys()	Returns list of keys
.values()	Returns list of values
.pop(key)	Removes item by key

Method	Description
.popitem()	Removes last inserted item
.update()	Updates dictionary

📜 LIST COMPREHENSIONS

- Concise way to create lists.
  Syntax:

[expression for item in iterable if condition]

- Expression → what to store
- Item → each value
- Condition (optional) → filter

📋 LIST METHODS

sort()

- Arranges items ascending or descending.

mylist.sort(reverse=True)

sorted()

- Returns a new sorted list without modifying original.

sorted_list = sorted(mylist)

---

append()

Adds one item at the end.

fruits.append("mango")

---

extend()

Adds multiple items from another iterable.

fruits.extend(["orange", "melon"])

---

insert()

Adds item at specific position.

fruits.insert(1, "mango")

---

pop()

Removes and returns an item (by index or last by default).

fruits.pop()

---

remove()

Removes an item by value.

fruits.remove("apple")

---

clear()

Removes all items.

fruits.clear()

---

index()

Finds the index of an item.

fruits.index("banana")

---

count()

Counts occurrences of an item.

fruits.count("apple")

---

map()

Applies a function to each item.

list(map(lambda x: x*2, [1, 2, 3]))

---

filter()

Keeps only items that meet a condition.

list(filter(lambda x: x % 2 == 0, [1, 2, 3, 4]))

---

reduce()

Combines list elements into one value (requires functools).

from functools import reduce

reduce(lambda a,b: a+b, [1,2,3,4])

✅ **REVIEW SUMMARY**

- **Collections:** Advanced data containers (Counter, Deque, etc.)
- **Math & Random:** Numerical and random utilities.
- **Zipfile:** File compression and extraction.
- **Numbers:** Int, Float, Complex, conversions, math ops.
- **Strings:** Immutable text manipulation.
- **Sets:** Unique, unordered collections.
- **Dictionaries:** Key-value data mapping.
- **Lists:** Ordered, mutable sequences with many built-in methods.