# CSI-160 Python Programming
# Collection Data types

Vikas Thammanna Gowda

07/22/2025

## Contents

## 1 Introduction

In Python, a *collection* is a data type that can hold multiple values under a single name. Collections help us organize and work with groups of related items—like a list of student names, a tuple of coordinates, a set of unique vocabulary words, or a dictionary mapping student IDs to grades. Python provides four built-in collection types:

- **List**: An ordered, *mutable* sequence of items.
- **Tuple**: An ordered, *immutable* sequence of items.
- **Set**: An *unordered* collection of *unique* items.
- **Dictionary**: A mapping of *keys* to *values* (an unordered collection of key-value pairs).

Together, these four **data structures** let you handle almost any grouping of data you might need in your programs.

## 2 List

**Definition:** A list is an ordered sequence of elements, enclosed in square brackets `[ ]`. (Ordering starts at 0 and goes until length - 1)
**Mutability:** You can change (add, remove, or modify) elements after the list is created.
**Typical Uses:** Storing a sequence of items where order matters or duplicates are allowed.

### 2.1 Key Operations

- `append(item)`: add to end
- `insert(i, item)`: insert at index `i`
- `remove(item)`: delete first matching
- `pop()` or `pop(i)`: remove and return
- `len(list)`: number of items

### 2.2 Illustration

Example:

```
students = ["Alice", "Bob", "Charlie", "Bob"]
print(students)


# Accessing elements by index (0-based)
first = students[0]
last = students[-1]

# Adding items
students.append("Diana")     # adds to end

# Removing items
students.remove("Bob")       # removes first "Bob"

print(students)
```

Output:

```
['Alice', 'Bob', 'Charlie', 'Bob']
['Alice', 'Charlie', 'Bob', 'Diana']
```

## 3 Tuple

**Definition:** A tuple is like a list but immutable, enclosed in parentheses ().
**Immutability:** Once created, you cannot change its elements.
**Typical Uses:** Fixed collections—such as coordinates (x, y), or days of the week—that should not change.

### 3.1 Key Properties

- Ordered (indexing and slicing possible)
- Immutable (elements cannot be changed once set)
- Faster for fixed data

### 3.2 Illustration

Example:

```
# Creating a tuple of coordinates
point = (3, 7)
print(point)

# Accessing elements
x, y = point            # tuple unpacking
print("x =", x, "y =", y)

# Single-element tuple needs a trailing comma
singleton = ("only one",)
print(type(singleton))
```

Output:

```
(3, 7)
x = 3 y = 7
<class 'tuple'>
```

## 4 Set

**Definition:** A set is an unordered collection of unique items, enclosed in braces {}.
**Uniqueness:** Duplicate entries are automatically removed.
**Typical Uses:** When you need to test membership quickly or want only one of each item. For example, unique words in a text.

### 4.1 Key Operations

- `add(item)`: add element
- `remove(item)`: remove element (error if missing)
- `discard(item)`: remove if present
- `union()/intersection()/difference()`

### 4.2 Illustration

Example:

```
# Creating a set of vocabulary words
words = {"apple", "banana", "apple", "cherry"}
print(words)


# Adding and removing items
words.add("date")
words.remove("banana")

# Membership test
if "apple" in words:
    print("We have an apple!")

# Set operations
even = {2, 4, 6, 8}
odd  = {1, 3, 5, 7}
all_numbers = even.union(odd)          # union
common = even.intersection({4,5}) # intersection
diff = even.difference({6,8})    # difference

print("All numbers:", all_numbers)
print("Common numbers:", common)
print("Difference:", diff)
```

Output:

```
{'banana', 'cherry', 'apple'}
We have an apple!
All numbers: {1, 2, 3, 4, 5, 6, 7, 8}
Common numbers: {4}
Difference: {2, 4}
```

## 5 Dictionary

**Definition:** A dictionary stores key-value pairs, enclosed in braces with colons (`:`)

Syntax:

```
{key: value}
```

**Uniqueness:** Since Python 3.7, insertion-order is preserved, but keys are still looked up by hash, not position.
**Typical Uses:** When you need to look up information by a unique key—such as student IDs, product SKUs, or word - definition pairs.

### 5.1 Key Methods

- `dict.keys()/values()/items()`
- `get(key, default)`: safe lookup
- `pop(key)`: remove and return value
- `clear()`: remove all entries

### 5.2 Illustration

Example:

```
# Creating a dictionary mapping student IDs to grades
grades = {"S001": 85, "S002": 92, "S003": 78}
print(grades)

# Accessing values by key
grade_s002 = grades["S002"]    # 92

# Adding or updating entries
grades["S004"] = 88            # add new key
grades["S001"] = 90            # update existing

# Removing entries
del grades["S003"]             # removes key "S003"


# Getting all keys or all values
all_ids    = grades.keys()
all_scores = grades.values()

print("All keys:", all_ids)
print("All values:", all_scores)
```

Output:

```
{'S001': 85, 'S002': 92, 'S003': 78}
All keys: dict_keys(['S001', 'S002', 'S004'])
All values: dict_values([90, 92, 88])
```

## 6 Comparison of Collections

| **Collection** | **Ordered?** | **Mutable?** | **Duplicates?** | **Best For** |
|---|---|---|---|---|
| List | Yes | Yes | Yes | Sequences you need to change (e.g., playlists) |
| Tuple | Yes | No | Yes | Fixed data (e.g., coordinates) |
| Set | No | Yes | No | Unique items & fast membership tests |
| Dictionary | Yes | Yes | Keys unique | Mapping keys to values (e.g., lookups by ID) |