🧩 Section 2: Collections

# 🎯 Objectives

By the end of this section, students will be able to:

* Create and use **Lists**, **Sets**, and **Maps**.
* Perform **CRUD operations** on collections.
* Iterate through collections using for, for-in, and forEach.
* Use spread (...) and null-aware (...?, ??) operators.

# 🔹 1. Lists

## 🧠 Concepts

* **List** = ordered collection (like arrays in other languages).
* You can access elements by **index**.
* Lists can be fixed-length or growable.

### 💡 Example: Create & CRUD Operations

void main() {

// Create

List<String> fruits = ['Apple', 'Banana', 'Orange'];

// Read

print('First fruit: ${fruits[0]}');

// Update

fruits[1] = 'Mango';

// Add new

fruits.add('Grapes');

// Delete

fruits.remove('Orange');

print('All fruits: $fruits');

print('Total: ${fruits.length}');

}

Explanation:

* .add() → append
* .remove(value) → delete by value
* You can also use .removeAt(index) or .clear().

### 🧩 Exercise 2.1 — List Practice

Create a list called students with three names.  
Then:

1. Add a new student.
2. Change the second student’s name.
3. Remove the first student.
4. Print the final list and its length.

# 🔹 2. Sets

## 🧠 Concepts

* **Set** = unordered collection of **unique** items.
* Automatically removes duplicates.
* Fast membership tests using .contains().

### 💡 Example: Create & CRUD

void main() {

Set<String> cities = {'Paris', 'London', 'Tokyo'};

cities.add('Rome'); // Add

cities.add('Paris'); // Ignored (duplicate)

cities.remove('Tokyo'); // Delete

print('Cities: $cities');

print('Contains Rome? ${cities.contains('Rome')}');

}

Explanation:

* Use {} for Set literals.
* .add() won’t insert duplicates.
* .contains() is useful for lookups.

### 🧩 Exercise 2.2 — Set Practice

Create a Set<int> named numbers with {1, 2, 3, 3, 4}.  
Then:

1. Add 5.
2. Remove 2.
3. Check if it contains 4.
4. Print all numbers.

💡 Observe how duplicates behave automatically.

# 🔹 3. Maps

## 🧠 Concepts

* **Map** = key–value pairs (like dictionaries or hash maps).
* Keys must be unique; values can repeat.

### 💡 Example: Create & CRUD

void main() {

Map<String, int> ages = {

'Alice': 25,

'Bob': 30,

'Charlie': 28

};

// Read

print('Alice is ${ages['Alice']} years old.');

// Create / Update

ages['David'] = 22; // Add new

ages['Bob'] = 31; // Update existing

// Delete

ages.remove('Charlie');

print('All ages: $ages');

print('Keys: ${ages.keys}');

print('Values: ${ages.values}');

}

Explanation:

* Access using map[key].
* .keys and .values return collections.
* .remove(key) deletes an entry.

### 🧩 Exercise 2.3 — Map Practice

Create a map products where keys are product names and values are prices.  
Then:

1. Add two products.
2. Update one product’s price.
3. Remove one product.
4. Print all key–value pairs.

# 🔹 4. Iteration (Looping over Collections)

### 💡 Example:

void main() {

List<String> fruits = ['Apple', 'Banana', 'Grapes'];

// for loop

for (int i = 0; i < fruits.length; i++) {

print('Fruit $i: ${fruits[i]}');

}

// for-in

for (var fruit in fruits) {

print('Fruit: $fruit');

}

// forEach

fruits.forEach((f) => print('Fruit name: $f'));

}

**Explanation:**  
All three approaches achieve the same goal.  
forEach is ideal for concise function-based iterations.

### 🧩 Exercise 2.4 — Loop Practice

Create a list of integers {2, 4, 6, 8, 10}.  
Use:

1. A traditional for loop to print each number doubled.
2. A for-in loop to print their sum.
3. A forEach to print all numbers in one line separated by commas.

# 🔹 5. Spread & Null-aware Operators

## 🧠 Concepts

* **Spread operator (...)** → merge collections.
* **Null-aware spread (...?)** → safely merge nullable collections.
* **Null-coalescing (??)** → use a default if value is null.

### 💡 Example:

void main() {

List<int> numbers = [1, 2, 3];

List<int>? more = [4, 5];

// Combine lists safely

List<int> combined = [...numbers, ...?more];

print(combined);

// Null-aware operator

int? value;

print(value ?? 10); // prints 10 if value is null

}

Explanation:

* ...? prevents an error if more is null.
* ?? provides a fallback when a value is null.

### 🧩 Exercise 2.5 — Spread & Null-Aware Practice

Create:

* list1 = [1, 2, 3]
* list2 = null

Then:

1. Combine both using ...? safely.
2. Print the combined list.
3. Use ?? to print a default message if a variable is null.

# 🏁 Summary

In this section, you learned how to:  
✅ Create and manipulate **Lists, Sets, Maps**  
✅ Perform **add, update, delete** operations  
✅ Iterate over collections efficiently  
✅ Use **spread** and **null-aware** operators