**Section 48 — Data Structures in Kotlin**

**1. Introduction to Kotlin Collections**

* **Collections** = groups of related data elements.
* Kotlin provides a **rich standard library** for:
  + **Lists** → Ordered elements, can have duplicates.
  + **Sets** → Unordered, unique elements only.
  + **Maps** → Key-value pairs, keys unique.
* **Two main types**:
  + **Immutable collections** → Read-only, cannot be changed after creation.
  + **Mutable collections** → Can be modified (add, remove, update).
* Functional operations available: map, filter, reduce, etc., using **lambda expressions**.

**2. Immutable List**

* **Definition**: Ordered collection, read-only.
* **Create**:
* // Immutable List
* val fruits = listOf("Apple", "Banana", "Cherry")
* // Iterate and print
* for (item in fruits) {
* println(item)
* }

**Notes:**

* + Created with listOf() or listOfNotNull().
  + Can access by **index** (fruits[0] → Apple).
  + No add, remove, or set methods.

**3. Mutable List**

* **Definition**: Ordered collection, modifiable.
* **Create & Modify**:
* val vegetables = mutableListOf("Potato", "Broccoli")
* // Add element
* vegetables.add("Carrot")
* // Remove element by index
* vegetables.removeAt(0) // removes "Potato"
* // Update element
* vegetables[0] = "Garlic"
* // Print all
* for (item in vegetables) println(item)

**Notes:**

* + Created with mutableListOf() or arrayListOf().
  + Can add, remove, and update freely.
  + arrayListOf() is backed by Java’s ArrayList.

**4. Immutable Set**

* **Definition**: Unordered, unique elements, read-only.
* **Create**:
* val colors = setOf("Red", "Green", "Blue")
* for (color in colors) println(color)

**Notes:**

* + No duplicates allowed (setOf("Red", "Red") → one "Red").
  + No modification methods.

**5. Mutable Set**

* **Definition**: Unordered, unique elements, modifiable.
* **Create & Modify**:
* val fruits = mutableSetOf("Apple", "Banana", "Cherry")
* fruits.add("Melon") // Add new
* fruits.add("Apple") // Duplicate ignored
* fruits.remove("Banana") // Remove element
* // Updating means remove old + add new
* fruits.remove("Cherry")
* fruits.add("Orange")
* for (item in fruits) println(item)

**Notes:**

* + Cannot update in-place by index (no ordering).
  + Best when order doesn’t matter but uniqueness does.

**6. Immutable Map**

* **Definition**: Key-value pairs, read-only.
* **Create & Access**:
* val fruits = mapOf("Apple" to 5, "Banana" to 8, "Cherry" to 12)
* // Access value by key
* val qty = fruits["Banana"] // 8
* println(qty)

**Notes:**

* + Keys are unique, values can repeat.
  + Created with mapOf() or mapOfNotNull().

**7. Mutable Map**

* **Definition**: Key-value pairs, modifiable.
* **Create & Modify**:
* val vegetablePrices = mutableMapOf(
* "Potato" to 1.5,
* "Tomato" to 3.5,
* "Broccoli" to 6.0
* )
* // Update value
* vegetablePrices["Tomato"] = 4.0
* // Add new pair
* vegetablePrices.put("Garlic", 2.0)
* // Remove by key
* vegetablePrices.remove("Potato")
* // Iterate over entries
* for ((veg, price) in vegetablePrices) {
* println("$veg -> $price")
* }

**Notes:**

* + Use put() or bracket notation (map[key] = value) to add/update.
  + Removing a key deletes its pair.

**8. Functional Operations**

* **map** → Transform each element.
* **filter** → Keep only elements matching condition.
* **reduce** → Combine all elements into a single value.
* val nums = listOf(1, 2, 3, 4)
* val doubled = nums.map { it \* 2 }
* val evens = nums.filter { it % 2 == 0 }
* val sum = nums.reduce { acc, n -> acc + n }

**9. Tools, Libraries, APIs Used**

* **Kotlin Standard Library Collections API** (listOf, mutableListOf, setOf, mutableSetOf, mapOf, mutableMapOf).
* **Java interoperability**: ArrayList, HashSet, HashMap.

**10. Best Practices**

* Prefer **immutable collections** unless modification is required (safer, thread-friendly).
* Use **val** for collection references unless you plan to reassign the variable.
* Avoid modifying collections during iteration (can cause exceptions).
* Use **functional operations** (map, filter, forEach) for readability.
* For large data, consider **sequences** instead of lists to avoid intermediate copies.

**Part B — Missing but Important**

1. **LinkedList and performance trade-offs**
   * Kotlin doesn’t provide its own; use Java’s LinkedList when insertion/removal in the middle is frequent.
2. **Sorted collections**
   * Use sortedSetOf() or TreeMap for sorted order.
3. **associate and groupBy**
   * Create maps from lists or group data.
4. val names = listOf("Alice", "Bob", "Adam")
5. val grouped = names.groupBy { it.first() } // {A=[Alice, Adam], B=[Bob]}
6. **Destructuring in loops**
   * For maps:
   * for ((key, value) in myMap) { println("$key -> $value") }
7. **contains, containsKey, containsValue**
   * Quick existence checks in collections.
8. **Thread-safe collections**
   * Use Collections.synchronizedList() from Java if multi-threaded.
9. **Null-safe collections**
   * Use listOfNotNull() to skip nulls at creation.
10. **Immutable wrappers**
    * Use toList(), toSet() to create a snapshot that won’t change.