LAB_4

Chapter 8: Collections

1. List

A List is an ordered collection of elements. It’s similar to an array in other languages.

Example:

```
void main() {
  List<String> desserts = ['cookies', 'cupcakes', 'donuts', 'pie'];
  print(desserts[1]); // Output: cupcakes
}
```

2. Mutable vs Immutable Lists

- var allows reassigning the entire list.
- final allows modifying elements but not reassigning the list.
- const makes the list and its contents completely unchangeable.

Example:

```
void main() {
  final desserts = ['cake', 'ice cream'];
  desserts.add('brownie'); // Allowed
  // desserts = []; // Error
```

```
  const frozenDesserts = ['popsicle'];
  // frozenDesserts.add('ice'); // Error
}
```

3. List Properties and Methods

- first, last, isEmpty, isNotEmpty, length
- add(), remove(), indexOf()

Example:

```
void main() {
  var drinks = ['water', 'juice', 'soda'];
  print(drinks.first); // water
  print(drinks.length); // 3
}
```

4. Loops on Lists

- For-in:

```
for (var item in desserts) {
  print(item);
}
```

- forEach:

```
desserts.forEach(print);
```

5. Spread Operator (`...`, `...?`)

Used to insert multiple items from one list into another.

Example:

```
void main() {
  const sweets = ['cake', 'candy'];
  const all = ['juice', ...sweets];
  print(all); // [juice, cake, candy]
}
```

6. Collection if / for

Example (if):

```
const peanutAllergy = true;
const candy = ['M&M', if (!peanutAllergy) 'Reeses'];
```

Example (for):

```
const desserts = ['cake', 'pie'];
var uppercase = [for (var d in desserts) d.toUpperCase()];
print(uppercase); // [CAKE, PIE]
```

7. Set

A `Set` is an unordered collection of unique items.

Example:

```
void main() {
  var numbers = {1, 2, 2, 3};
  print(numbers); // {1, 2, 3}
}
```

8. Map

Maps hold key-value pairs like dictionaries.

Example:

```
void main() {
  var calories = {'cake': 300, 'donut': 150};
  print(calories['cake']); // 300
}
```

Mini Exercises:-

1. Create an empty list of type String. Add all 12 months.

Ans:-

```
void main() {
  List<String> months = [];
  months.addAll([
    'January', 'February', 'March', 'April', 'May', 'June',
    'July', 'August', 'September', 'October', 'November', 'December'
  ]);
  print(months);
}
```

2. Create an immutable list with same elements as in Mini-exercise 1.

Ans:-
```
void main() {
  const months = [
    'January', 'February', 'March', 'April', 'May', 'June',
    'July', 'August', 'September', 'October', 'November', 'December'
  ];
  print(months);
}
```

3. Use collection for to create a new list with the month names in all uppercase.

Ans:-
```
void main() {
  const months = [
    'January', 'February', 'March', 'April', 'May', 'June',
    'July', 'August', 'September', 'October', 'November', 'December'
  ];
  final upperMonths = [for (var month in months)
month.toUpperCase()];
  print(upperMonths);
}
```

Mini-Exercies 2

1. Create a map with the following keys: name, profession,country and city. Forthe values, add your own information.

Ans:-

```
void main() {
  Map<String, String> person = {
    'name': 'Smit',
    'profession': 'Student',
    'country': 'India',
    'city': 'Ahmedabad'
  };
  print(person);
}
```

2. You suddenly decide to move to Toronto, Canada. Programmatically update the values for country andcity.

Ans:-

```
void main() {
  Map<String, String> person = {
    'name': 'Smit',
    'profession': 'Student',
    'country': 'India',
    'city': 'Ahmedabad'
  };
  // Updating the values
  person['country'] = 'Canada';
```

```
  person['city'] = 'Toronto';
  print(person);
}
```

3. Iterate over the map and print all the values.

Ans:-

```
void main() {
  Map<String, String> person = {
    'name': 'Smit',
    'profession': 'Student',
    'country': 'Canada',
    'city': 'Toronto'
  };
  // Print all values
  for (var value in person.values) {
    print(value);
  }
}
```

Mini Exercises:3

Given the following exam scores: final scores = [89, 77, 46, 93, 82, 67, 32, 88];

1. Use sort to find the highest and lowest grades.

Ans:-

```
void main() {
  final scores = [89, 77, 46, 93, 82, 67, 32, 88];

  // Make a copy before sorting (to preserve original if needed)
  final sortedScores = [...scores]..sort();

  final lowest = sortedScores.first;
  final highest = sortedScores.last;

  print('Lowest score: $lowest');   // 32
  print('Highest score: $highest'); // 93
}
```

2.Use where to find all B grades (scores between 80 and 90).

Ans:-

```
void main() {
  final scores = [89, 77, 46, 93, 82, 67, 32, 88];
```

```
  final bGrades = scores.where((score) => score >= 80 && score
<= 90);


  print('B grades: $bGrades'); // (89, 82, 88)
}
```

Challenges:

Challenge 1: A unique request

Write a function that takes a paragraph of text and returns a collection of unique String characters that the text Contains.

Ans:-

```
Set<String> uniqueCharacters(String text) {
  return text.split('').toSet(); // Convert to set to remove duplicates
}


void main() {
  const paragraph = 'Dart is awesome!';
  final uniqueChars = uniqueCharacters(paragraph);


  print('Unique characters: $uniqueChars');
```

```
}
```

Challenge 2: Counting on you

Repeat Challenge 1, but this time have the function return a collection that contains the frequency, or count, of every unique character.

Ans:-

```
Map<String, int> characterFrequency(String text) {
  final Map<String, int> frequencies = {};

  for (var char in text.split('')) {
    if (frequencies.containsKey(char)) {
      frequencies[char] = frequencies[char]! + 1;
    } else {
      frequencies[char] = 1;
    }
  }

  return frequencies;
```

```
}

void main() {
  const paragraph = 'hello world';
  final freqMap = characterFrequency(paragraph);

  print('Character frequencies: $freqMap');
}
```

Challenge 3: Mapping users

Create a class called User with properties for id and name. Make a List with three users, specifying any appropriate names and IDs you like. Then write a function that converts your user list to a list of maps whose keys are id and name.

Ans:-

```
class User {
  final int id;
  final String name;
```

```
  User({required this.id, required this.name});
}

// Function to convert List<User> to List<Map<String, Object>>
List<Map<String, Object>> convertToMap(List<User> users) {
  return users.map((user) => {
    'id': user.id,
    'name': user.name,
  }).toList();
}
void main() {
  final users = [
    User(id: 1, name: 'Alice'),
    User(id: 2, name: 'Bob'),
    User(id: 3, name: 'Charlie'),
  ];
  final mappedUsers = convertToMap(users);
  print(mappedUsers);
}
```

Chapter 9: Advanced Classes

1. Inheritance

Child classes can inherit from parent classes using `extends`.

Example:

```
class Person {
  String name;
  Person(this.name);
}

class Student extends Person {
  Student(String name) : super(name);
}
```

2. Superclass and Subclass

- Superclass: The parent class (`Person`)
- Subclass: The derived class (`Student`)

3. Method Overriding

Use `@override` to redefine methods in the child class.

Example:

```
class Person {
  String fullName() => 'Anonymous';
}


class Student extends Person {
  @override
  String fullName() => 'Student Name';
}
```

4. Calling Superclass Constructor

Use `super()` in subclass constructor to call parent constructor.

```
class Student extends Person {
  Student(String name) : super(name);
}
```

5. Multi-level Inheritance

You can extend a subclass further.

```
class Student {}
class SchoolBandMember extends Student {}
class StudentAthlete extends Student {}
```

6. Polymorphism and Type Checking

You can check object types at runtime using `is` and `is!`.

```
print(obj is Person); // true
```

7. Composition Over Inheritance

Instead of deep inheritance, prefer adding objects/roles.

```
class Student {
  List<Role>? roles;
}
```

8. Abstract Classes

Cannot be instantiated. Define shared behavior.

```
abstract class Animal {
  void eat();
```

```
}
```

9. Concrete Subclass

Must implement all abstract methods.

```
class Platypus extends Animal {
  @override
  void eat() => print('Munch');
}
```

10. Interfaces

Interfaces describe how classes should behave.

```
abstract class Database {
  void save();
}
```

Mini-Exercises

1. Create a class named Fruit with a String field named color and a method named describeColor, which uses colorto print a message.

Ans:-

```
class Fruit {
  String color;
  Fruit(this.color);
  void describeColor() {
    print("This fruit is $color");
  }
}
```

2. Create a subclass of Fruit named Melon and then create two Melon subclasses named Watermelon and Cantaloupe.

Ans:-
```
class Melon extends Fruit {
  Melon(String color) : super(color);
}
class Watermelon extends Melon {
  Watermelon() : super('green');
}
```

```
class Cantaloupe extends Melon {
  Cantaloupe() : super('orange');
}
```

3. Override describeColorin the Watermelon class to vary the output.

Ans:-

```
class Watermelon extends Melon {
  Watermelon() : super('green');

  @override
  void describeColor() {
    print("Watermelons are typically $color outside and red inside.");
  }
}
```

Mini Exercises:2

1. Create an interface called Bottle and add a method to it called open.

Ans:-

```
abstract class Bottle {
```

```
  void open();
}
```

2. Create a concrete class called SodaBottle that implements Bottle and prints “Fizz fizz” when open is called.

Ans:-

```
class SodaBottle implements Bottle {
  @override
  void open() {
    print('Fizz fizz');
  }
}
```

3. Add a factory constructor to Bottle that returns a SodaBottle instance.

Ans:-

```
abstract class Bottle {
  void open();

  factory Bottle() => SodaBottle(); // Factory constructor
}
```

4. Instantiate SodaBottle by using the Bottle factory constructor and call open on the object.

Ans:-

```
void main() {
  Bottle bottle = Bottle(); // Using factory constructor
  bottle.open();            // Output: Fizz fizz
}
```

Mini-Exercies:3

1. Create a class called Calculator with a method called sum that prints the sum of any two integers you give it.

Ans:-

```
class Calculator {
  void sum(int a, int b) {
    print('Sum is: ${a + b}');
  }
}
```

2. Extract the logic in sum to a mixin called Adder.

Ans:-

```
mixin Adder {
  void sum(int a, int b) {
    print('Sum is: ${a + b}');
  }
}
```

3. Use the mixin in Calculator.

Ans:-

```
class Calculator with Adder {}

void main() {
  Calculator calc = Calculator();
  calc.sum(5, 7); // Output: Sum is: 12
}
```

Challenges:-

Challenge 1: Heavy monotremes

Dart has a class named Comparable, which is used by the the sort method of List to sort its elements. Add a weight field to the Platypus class you made in this lesson. Then make

Platypus implement Comparable so that when you have a list of Platypus objects, calling sort on the list will sort them by weight.

Ans:

```
class Platypus implements Comparable<Platypus> {
  final String name;
  final double weight; // in kg

  Platypus({required this.name, required this.weight});

  @override
  int compareTo(Platypus other) {
    return weight.compareTo(other.weight); // Ascending order
  }

  @override
  String toString() => '$name: ${weight}kg';
}
```

```
void main() {
  final platypuses = [
    Platypus(name: 'Perry', weight: 4.2),
    Platypus(name: 'Polly', weight: 3.8),
    Platypus(name: 'Patty', weight: 5.1),
  ];

  platypuses.sort();
  print('Sorted by weight: $platypuses');
}
```

Challenge 2: Fake notes

Design an interface to sit between the business logic of your note-taking app and a SQL database. After that, implement a fake database class that will return mock data.

Ans:-

```
// Interface
abstract class Database {
  Future<List<String>> getNotes();
```

```
}

// Fake implementation for testing
class FakeDatabase implements Database {
  @override
  Future<List<String>> getNotes() async {
    return Future.delayed(
      Duration(seconds: 1),
      () => ['Buy groceries', 'Study Dart', 'Call Mom'],
    );
  }
}

void main() async {
  final db = FakeDatabase();
  final notes = await db.getNotes();
  print('Mock notes: $notes');
}
```

Challenge 3: Time to code

Dart has a Duration class for expressing lengths of time.

Make an extension on int so that you can express a duration like so:

Ans:

```
extension TimeExtension on int {
  Duration get seconds => Duration(seconds: this);
  Duration get minutes => Duration(minutes: this);
  Duration get hours => Duration(hours: this);
}

void main() {
  final timeRemaining = 3.minutes + 2.seconds;
  print('Time remaining: $timeRemaining'); // 0:03:02.000000
}
```

Chapter 10: Asynchronous Programming

1. Synchronous vs Asynchronous

- Synchronous: Code executes line by line.
- Asynchronous: Code can be delayed and resume later without blocking the main thread.

Example:

```
void main() {
  print('First');
  print('Second');
  print('Third');
}
// Output: First Second Third
```

2. Future

Represents a value that might be available later.

Example:

```
Future<int> calculate() async {
  return 42;
}
```

3. Future.delayed

Simulates a delayed operation.

```
void main() {
  final future = Future.delayed(Duration(seconds: 2), () => 42);
  future.then((value) => print('Value: $value'));
  print('Done');
}
```

4. Callbacks

`then`, `catchError`, and `whenComplete` handle Future results.

```
Future.delayed(Duration(seconds: 1), () => 42)
  .then((value) => print('Value: $value'))
  .catchError((e) => print('Error: $e'))
  .whenComplete(() => print('Done'));
```

5. async and await

Makes async code look synchronous.

```
Future<void> main() async {
```

```
  print('Before');
  final value = await Future.delayed(Duration(seconds: 1), () =>
42);
  print('Value: $value');
  print('After');
}
```

6. try-catch-finally

Used for error handling in async code.

```
Future<void> main() async {
  try {
    final value = await Future.delayed(Duration(seconds: 1), () =>
42);
    print(value);
  } catch (e) {
    print('Error: $e');
  } finally {
    print('Done');
  }
```

```
}
```

7. Event Loop

Dart uses an event loop with microtask and event queues to manage tasks. This is how Dart remains responsive.

8. Isolates

Dart's way of achieving true parallelism. Each isolate has its own memory and thread.

9. HTTP Request (Network Call using Future & async-await)

- **Add dependency to `pubspec.yaml`:**

```
dependencies:
  http: ^0.13.1
```

- **Code:-**

```
import 'dart:convert';
import 'package:http/http.dart' as http;

Future<void> main() async {
```

```
  final url =
Uri.parse('https://jsonplaceholder.typicode.com/todos/1');
  final response = await http.get(url);


  if (response.statusCode == 200) {
    final jsonMap = jsonDecode(response.body);
    print(jsonMap['title']);
  } else {
    throw Exception('Failed to load');
  }
}
```

Mini Exercise: 1

1. Use the Future.delayed constructor to provide a string after two seconds that says “I am from the future.”

2. Create a String variable named message that awaits the future to complete with a value.

3. Surround the code above with a try-catch block.

Ans:-

```
Future<void> main() async {
  try {
    // Wait for 2 seconds and then return a string
    final message = await Future.delayed(
      Duration(seconds: 2),
      () => 'I am from the future.',
    );

    print(message); // Output: I am from the future.
  } catch (e) {
    print('An error occurred: $e');
  }
}
```

Mini-Exercises: 2

1. The following code produces a stream that outputs an integer every second and then stops after the tenth time.

Stream<int>.periodic(

Duration(seconds: 1),
(value) => value,
).take(10);

1. Set the stream above to a variable named myStream.
2. Use await for to print the value of the integer on each data event coming from the stream.

Ans:-

```
Future<void> main() async {
  final myStream = Stream<int>.periodic(
    Duration(seconds: 1),
    (value) => value,
  ).take(10);

  await for (final number in myStream) {
    print('Value from stream: $number');
  }
}
```

Challenges:

Challenge 1: Whose turn is it?
This is a fun one and will test how well you understand how Dart handles asynchronous tasks. In what order will Dart print the text with the following print statements? Why?

```
void main() {
print('1 synchronous');
Future(() => print('2 event queue')).then(
(value) => print('3 synchronous'),
);
```

```
Future.microtask(() => print('4 microtask queu
e'));
Future.microtask(() => print('5 microtask queu
e'));
Future.delayed(
Duration(seconds: 1),
() => print('6 event queue'),
);
Future(() => print('7 event queue')).then(
(value) => Future(() => print('8 event queu
e')),
);
Future(() => print('9 event queue')).then(
(value) => Future.microtask(
() => print('10 microtask queue'),
),
);
print('11 synchronous');
}
```

Try to answer before checking. If you're right, give yourself a well-deserved pat on the back!

Ans:

```
void main() {
  print('1 synchronous');
  Future(() => print('2 event queue')).then((_) => print('3
synchronous'));
  Future.microtask(() => print('4 microtask queue'));
  Future.microtask(() => print('5 microtask queue'));

  Future.delayed(Duration(seconds: 1), () => print('6 event
queue'));
```

```
  Future(() => print('7 event queue')).then((_) => Future(() =>
print('8 event queue')));

  Future(() => print('9 event queue')).then(
    (_) => Future.microtask(() => print('10 microtask queue')),
  );

  print('11 synchronous');
}
```

Output:-
1 synchronous
11 synchronous
4 microtask queue
5 microtask queue
2 event queue
3 synchronous
7 event queue
9 event queue
10 microtask queue
8 event queue
6 event queue

Challenge 2: Care to make a comment?
The following link returns a JSON list of comments:

https://jsonplaceholder.typicode.com/comments

Create a Commentdata class and convert the raw JSON to a Dart list of type List<Comment>.

Ans:

```
import 'dart:convert';
import 'package:http/http.dart' as http;

class Comment {
  final int id;
  final String name;
  final String email;
  final String body;

  Comment({required this.id, required this.name, required
this.email, required this.body});

  factory Comment.fromJson(Map<String, dynamic> json) {
    return Comment(
      id: json['id'],
      name: json['name'],
      email: json['email'],
      body: json['body'],
    );
  }
}

Future<void> main() async {
  final url =
Uri.parse('https://jsonplaceholder.typicode.com/comments');
  final response = await http.get(url);
```

```
  if (response.statusCode == 200) {
    final List<dynamic> data = jsonDecode(response.body);
    final comments = data.map((json) =>
Comment.fromJson(json)).toList();
    print('Total comments: ${comments.length}');
    print('First comment: ${comments[0].name}');
  } else {
    print('Failed to load comments');
  }
}
```

Challenge 3: Data stream

The following code allows you to stream content from the given URL:

```
final url = Uri.parse('https://raywenderlich.com'
);
final client = http.Client();
final request = http.Request('GET', url);
final response = await client.send(request);
final stream = response.stream;
```

Your challenge is to transform the stream from bytes to strings and see how many bytes each data chunk is. Add error handling, and when the stream is finished, close the client.

Ans:

```
import 'dart:convert';
import 'package:http/http.dart' as http;
```

```
Future<void> main() async {
  final url = Uri.parse('https://raywenderlich.com');
  final client = http.Client();

  try {
    final request = http.Request('GET', url);
    final response = await client.send(request);

    final stream = response.stream;
    await for (final chunk in stream) {
      print('Received chunk: ${chunk.length} bytes');
      print(utf8.decode(chunk)); // Convert bytes to string
    }
  } catch (e) {
    print('Error occurred: $e');
  } finally {
    client.close();
    print('HTTP client closed.');
  }
}
```

Challenge 4: Fibonacci from afar

In Challenge 4 of Chapter 4, you wrote some code to calculate the nth Fibonacci number. Repeat that challenge, but run the code in a separate isolate. Pass the value of n to the new isolate as an argument, and send the result back to the main isolate.

Ans:

```
import 'dart:isolate';

int fibonacci(int n) {
```

```
  if (n <= 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
}

void calculateFibonacci(SendPort sendPort) {
  final ReceivePort isolateReceivePort = ReceivePort();

  sendPort.send(isolateReceivePort.sendPort);

  isolateReceivePort.listen((message) {
    final int n = message[0];
    final SendPort replyPort = message[1];

    final result = fibonacci(n);
    replyPort.send(result);
  });
}

Future<void> main() async {
  final receivePort = ReceivePort();
  await Isolate.spawn(calculateFibonacci, receivePort.sendPort);

  final sendPort = await receivePort.first as SendPort;

  final answerPort = ReceivePort();
  sendPort.send([10, answerPort.sendPort]);

  final result = await answerPort.first;
  print('Fibonacci(10) = $result'); // Output: 55
}
```