



# Asynchronous Programming and Collections in Dart

Module 4 Chapter 1

# Introduction

## Asynchronous Programming

Helps apps stay **responsive** — especially during time-consuming tasks like fetching data or loading files.

Dart provides strong async tools: **Future**, **Stream**, **async**, and **await**.

## Collections in Dart

- Collections are containers for data — used to **store, manage, and access** multiple values.
- Main types: **List**, **Set**, **Map**, and **Queue**.



# Asynchronous Programming in Dart

## Futures

A **Future** represents an operation that **will finish later** — like getting data from the internet.

Think of it as a *promise* to deliver a value in the future.



A screenshot of a code editor showing a Dart file named `futures.dart`. The code defines a `Future<String>` named `fetchUserName()` which returns the string 'John' after a 2-second delay. The `main` function prints 'hii' and then the result of `fetchUserName()`. The output console shows 'hii' followed by 'John' and 'Exited.'

```
in > futures.dart > ...
1 Future<String> fetchUserName() async {
2   await Future.delayed(Duration(seconds: 2));
3   return 'John';
4 }
5
Run | Debug
6 void main() async {
7   print('hii');
8   print(await fetchUserName());
9 }
10
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE ... Filter (e.g. text, lexclude, \escape)

hii  
John  
Exited.

## Streams

A **Stream** gives a **sequence** of **async values** — like messages or events.

You can listen to them continuously using `.listen()` or `await for`.



A screenshot of a code editor showing a Dart file named `streams.dart`. It creates a `Stream<int>` named `counterStream` that emits integers from 0 to infinity at one-second intervals. The `main` function prints the first 5 numbers from this stream. The output console shows the numbers 0 through 4.

```
in > streams.dart > ...
1 Stream<int> counterStream =
2   Stream<int>.periodic(
3     Duration(seconds: 1), (x) => x); // Stream<int>
4
Run | Debug
5 void main() async {
6   await for (int i in counterStream.take(5)) {
7     print(i);
8   }
9 }
10
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE ... Filter (e.g. text, lexclude, \escape)

0  
1  
2  
3  
4  
Exited.

# Async and Await

## async

- Marks a function that runs **asynchronously** (it may take time to complete).
- When called, it returns a **Future** immediately.

```
awiaacy.dart > main
1 // A function that simulates fetching data from a server.
2 Future<String> fetchData() async {
3   print('Fetching data...');
4
5   // simulate network delay using Future.delayed
6   await Future.delayed(Duration(seconds: 3));
7
8   // Return a value after delay
9   return 'Data fetched successfully!';
10 }
11
12 // Main function
Run | Debug
13 void main() async [
14   print('Start of program');
15
16   // Await pauses here until fetchData() completes
17   String result = await fetchData();
18
19   print(result);
20   print('End of program');
21 ]
22
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE  
Start of program  
Fetching data...  
Data fetched successfully!  
End of program  
ited.

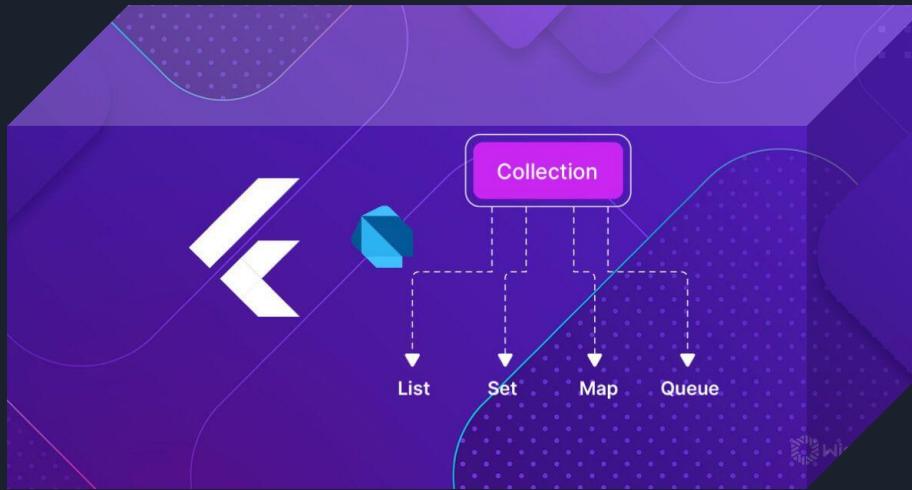
## await

- Temporarily **pauses** the function until the awaited task finishes.
- Keeps the app running smoothly without blocking other code.

```
asyawa.dart > main
1 Run | Debug
2 void main() async {
3   print('Start');
4   await Future.delayed(Duration(seconds: 3));
5   print('End');
6 }
7
8 PROBLEMS 1 OUTPUT DEBUG CONSOLE ...
9 Filter (e.g. text, lexclude, \escape)
10 Start
11 End
12 Exited.
```

# Collections in Dart

Collections help manage groups of data.  
Types → **List, Set, Map, Queue**



# Lists

A **List** is an **ordered collection** of items, accessed by index.

## Fixed-length List:

A screenshot of a code editor showing a fixed-length list. The code in `listf.dart` defines a list of three integers, changing the first element to 10. The output in the DEBUG CONSOLE shows the list [10, 0, 0].

```
bin > listf.dart > main
Run | Debug
1 void main(){
2   var numbers = List<int>.filled(3, 0);
3   numbers[0] = 10;
4   print(numbers);
5
6 }
```

PROBLEMS 1    OUTPUT    DEBUG CONSOLE    ...

Filter (e.g. text, !exclude, \escape)

[10, 0, 0]

Exited.

## Growable List:

A screenshot of a code editor showing a growable list. The code in `listg.dart` adds two strings to an initially empty list and prints it. The output in the DEBUG CONSOLE shows the list [Alice, Bob].

```
bin > listg.dart > main
Run | Debug
1 void main(){
2   var names = [];
3   names.add('Alice');
4   names.add('Bob');
5   print(names);
6 }
```

PROBLEMS 1    OUTPUT    DEBUG CONSOLE

Filter (e.g. text, !exclude, \escape)

[Alice, Bob]

Exited.

# Set

A **Set** stores unique items — duplicates are not allowed.

Great for checking membership or removing duplicates.

```
set.dart > main
Run | Debug
void main(){
| var fruits = ['apple', 'banana', 'orange'];
| print(fruits); |
}

PROBLEMS 1 OUTPUT DEBUG CONSOLE ...
Filter (e.g. text, !exclude, \escape)
{apple, banana, orange}
Exited.
```

# Map

A **Map** stores **key-value pairs** — like a dictionary.

Using literal:

```
bin > map.dart > main
Run | Debug
1 void main(){
2     var student = {
3         'name': 'Rahul', 'age': 21
4     };
5     print(student['name']);
6 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE ...

Filter (e.g. text, !exclude, \escape)

Rahul

Exited.

Using constructor:

```
bin > mapcon.dart > main
Run | Debug
1 void main(){
2     var country = Map();
3     country['India'] = 'New Delhi';
4     country['Japan'] = 'Tokyo';
5     print(country);
6 }
7 }
```

PROBLEMS 1 DEBUG CONSOLE ... Dart 2

Filter (e.g. text, !exclude, \escape)

{India: New Delhi, Japan: Tokyo}

Exited.

# Queue

A **Queue** allows insertion and removal from **both ends**.

Works on **FIFO** (First In, First Out).

```
bin > ⚙ dart_application_1.dart > ⚙ main
1 import 'dart:collection';
Run | Debug
2 void main() {
3   Queue<String> tasks = Queue();
4   tasks.addAll(['Task1', 'Task2', 'Task3']);
5   print(tasks);
6 }
7
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    ...    Filter (e.g. tex  
{Task1, Task2, Task3}

Exited.

# data structures in Dart

## TREE (Hierarchical Structure)

A **Tree** is a **non-linear** data structure where:

- The **topmost node** is called the **root**.
- Each node can have **child nodes**.
- Nodes connected by **edges**.

## GRAPH (Network Structure)

A **Graph** is a collection of **nodes (vertices)** and **edges** (connections between nodes).

It can be:

- **Directed** (one-way edges)
- **Undirected** (two-way edges)

## Exercise 1

```
exercise_1 > bin > ⚙ exercise_1.dart > ⚭ main
    Run | Debug
1 void main(){
2     var petPrices ={'Bella': 1.5,'Lucy':0.8,'Loki':1.2,'Leo':2.0,'oggy':3.5};
3     print('Pet Prices: $petPrices');
4
5     var price= petPrices.values;
6     double cost=price.reduce((current,nextval)=> current+nextval);
7     print(['Total Cost : $cost']);
8
9 }
```

PROBLEMS 1    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Pet Prices: {Bella: 1.5, Lucy: 0.8, Loki: 1.2, Leo: 2.0, oggy: 3.5}

Total Cost : 9.0

Exited.

## Exercise 2

```
exercise_2 > bin > exercise_2.dart > main
Run | Debug
1 void main(){
2     var numList= [1,2,3,4,3,2,5,6,7,8,7,9,10];
3     print('Original List: $numList');
4     //Set<int> uniqueelements=numList.toSet();
5     Set<int> uniqueelements=Set.from(numList);
6     print('Unique Elements : $uniqueelements');
7
8 }
```

PROBLEMS 1

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
Original List: [1, 2, 3, 4, 3, 2, 5, 6, 7, 8, 7, 9, 10]
Unique Elements : {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Exited.

## Exercise 3

```
exercise_3 > bin > exercise_3.dart > main
          Run | Debug
1 void main(){
2     var List= [1,2,3,4,5,6,7,8,9,10];
3     print('List:$List');
4     int sum = List.reduce((a,b)=> a+b);
5     print('Sum of elements :$sum');
6 }
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

List:[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Sum of elements :55

Exited.

## Exercise 4

```
exercise_4 > bin > exercise_4.dart > main
1 Future<String> fetchData()
2 {
3     await Future.delayed(Duration(seconds: 3));
4     return "Some Data from the server";
5 }
Run | Debug
6 void main()
7 {
8
9     print('program started');
10    print('fetching data...');
11    String data= await fetchData();
12    print('Data received : $data');
13    print('Program ended');
14
15 }
```

PROBLEMS 1    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
program started
fetching data...
Data received : Some Data from the server
Program ended
```

Exited.

## Exercise 5

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Initial Queue :{10, 20, 30, 40, 50}

First element of the Queue :10

Last element of the Queue :50

Removed element :10

Removed element :20

Removed element :30

Removed element :40

Removed element :50

Queue After Removal :{}

Exited.

```
exercise_5 > bin > exercise_5.dart > main
1 import 'dart:collection';
2
3 Run | Debug
4 void main(){
5   final queue =Queue<int>();
6   //print(queue.runtimeType);
7   queue.addAll([10,20,30,40,50]);
8   print('Initial Queue :$queue');
9
10  int firstqueue =queue.first;
11  print('First element of the Queue :$firstqueue');
12
13  print('Last element of the Queue :${queue.last}');
14
15  int remove1 = queue.removeFirst();
16  print('Removed element :$remove1');
17
18  int remove2 = queue.removeFirst();
19  print('Removed element :$remove2');
20
21  int remove3 = queue.removeFirst();
22  print('Removed element :$remove3');
23
24  int remove4 = queue.removeFirst();
25  print('Removed element :$remove4');
26
27  int remove5 = queue.removeFirst();
28  print('Removed element :$remove5');
29
30  print('Queue After Removal :$queue');
```