



Asynchronous Programming and Collections in Dart

Module 4 Chapter 1

Introduction

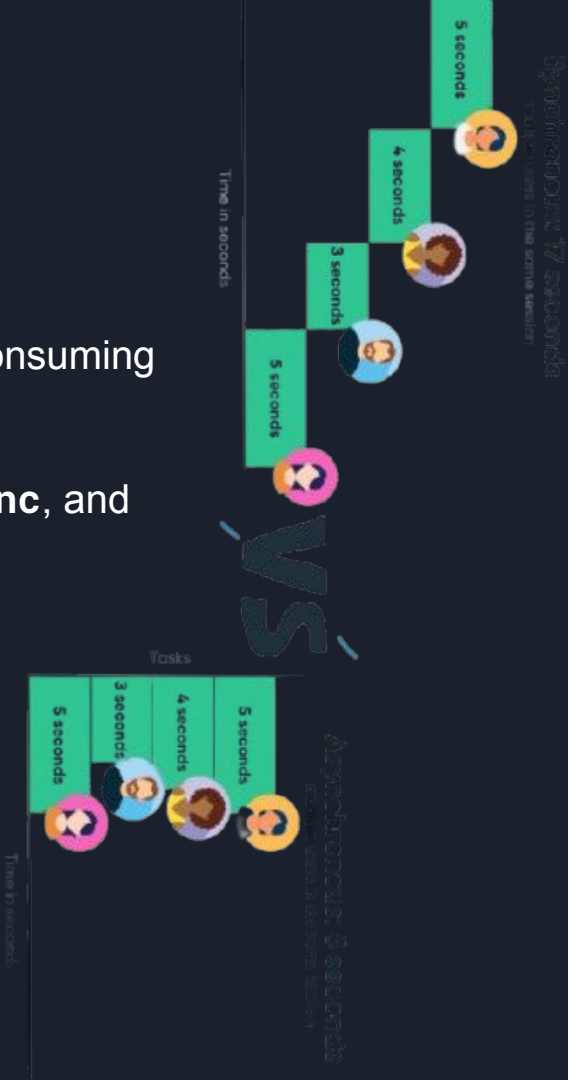
Asynchronous Programming

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.



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

Run | Debug

DEBUG CONSOLE

Filter (e.g. text, exclude, \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`.



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

Run | Debug

DEBUG CONSOLE

Filter (e.g. text, exclude, \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.

await

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

```
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
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
Exited.
```

asylawa.dart > main

```
Run | Debug
1 void main() async {
2   print('Start');
3   await Future.delayed(Duration(seconds: 3));
4   print('End');
5 }
6
```

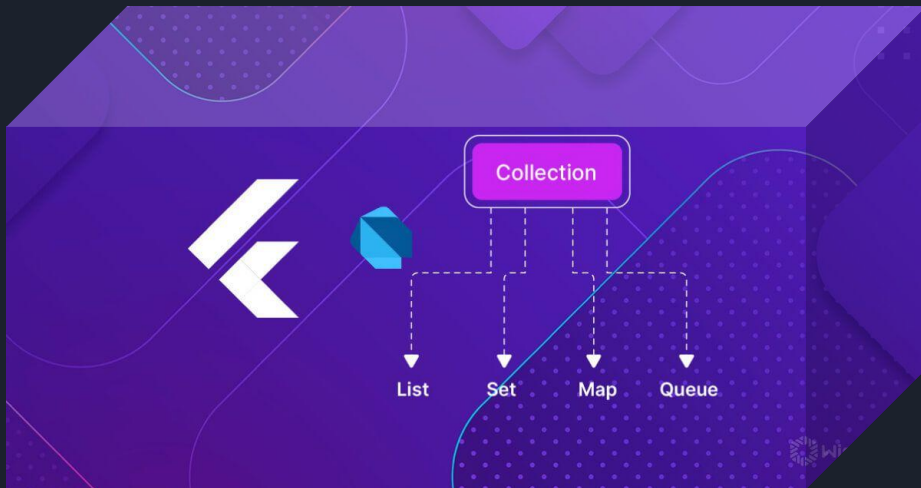
PROBLEMS 1 OUTPUT DEBUG CONSOLE ...

Filter (e.g. text, lexclude, \escape)

```
Start
End
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:

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:

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)

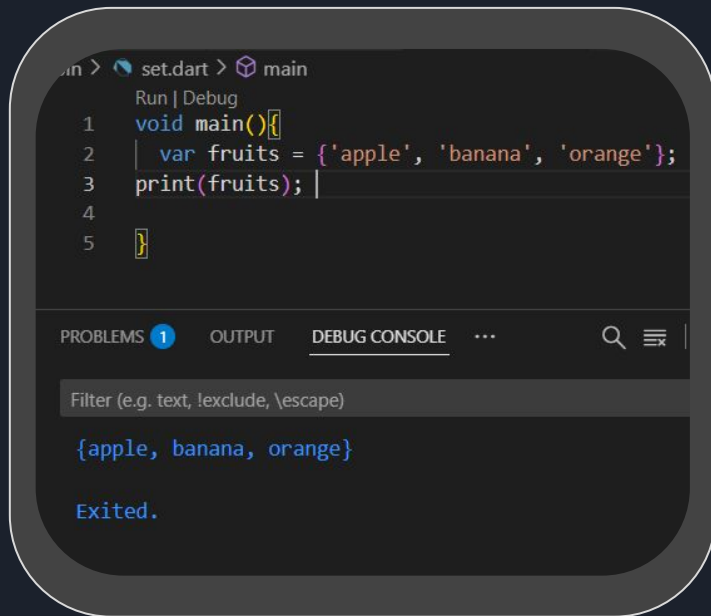
[Alice, Bob]

Exited.

Set

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

Great for checking membership or removing duplicates.



The screenshot shows a Dart IDE with a file named `set.dart` and a `main` function. The code defines a `Set` of fruits and prints it. The output in the debug console shows the set `{apple, banana, orange}` and the message `Exited.`

```
in > set.dart > main
Run | Debug
1 void main(){
2   var fruits = {'apple', 'banana', 'orange'};
3   print(fruits);
4
5 }
```

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. text)

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