Asynchronous Programming and Collections in Dart

Module 4 Chapter 1

Introduction

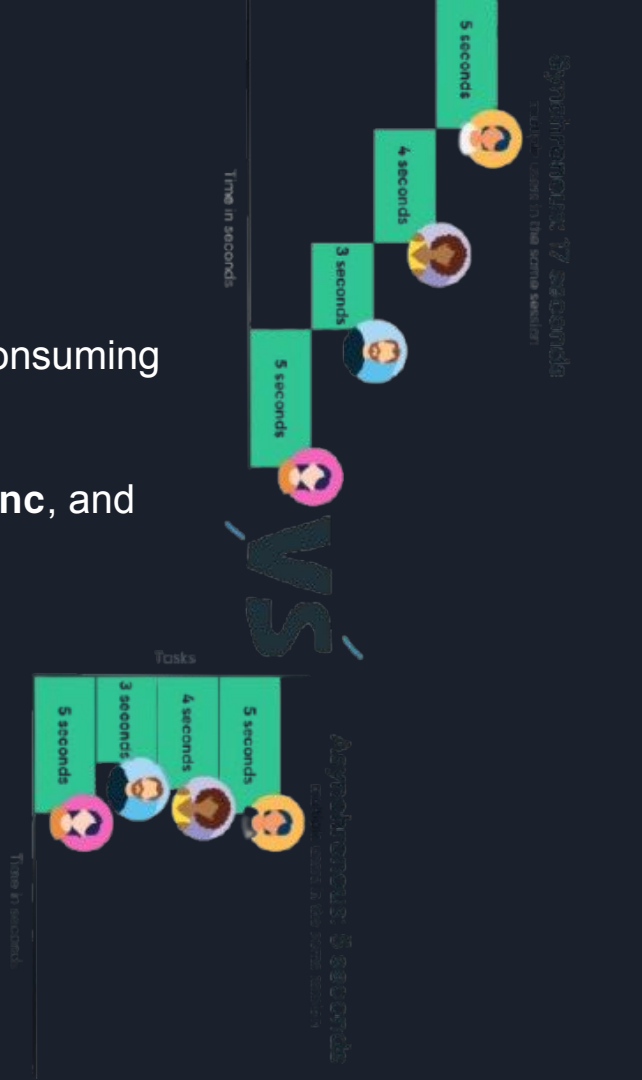
Asynchronous Programming

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.

Streams

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

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

Async and Await

async

- Marks a function that runs **asynchronously**

  (it may take time to complete).

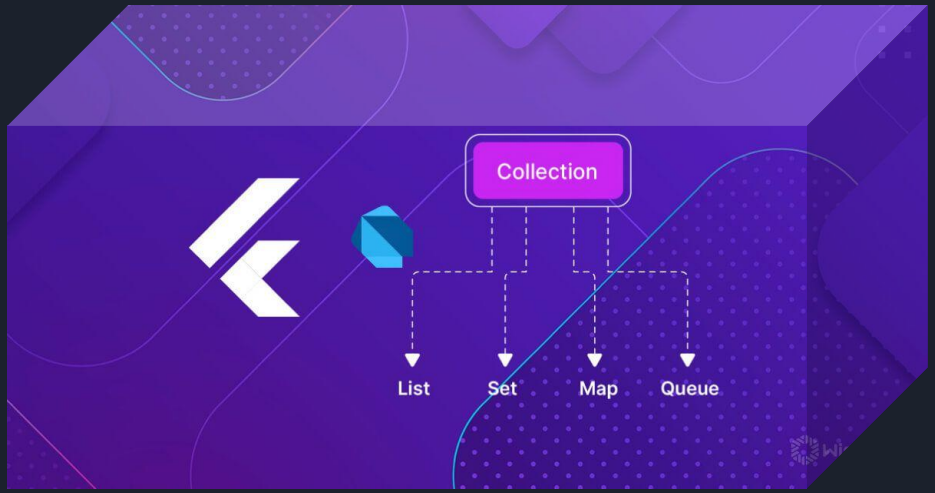
- When called, it returns a **Future** immediately.

await

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

Collections in Dart

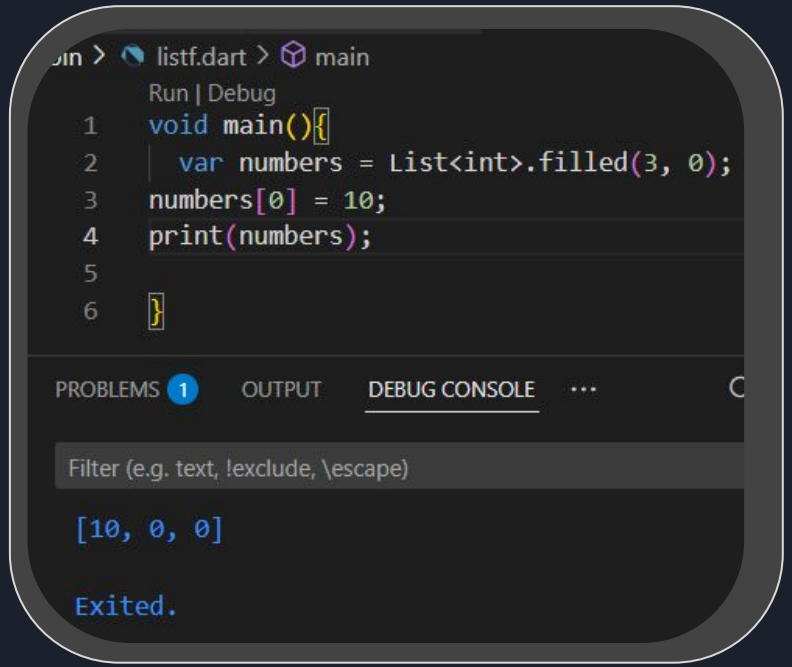
Collections help manage groups of data.
Types → **List**, **Set**, **Map**, **Queue**



Lists

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

Fixed-length List:



Growable List:

Set

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

Great for checking membership or removing duplicates.

Map

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

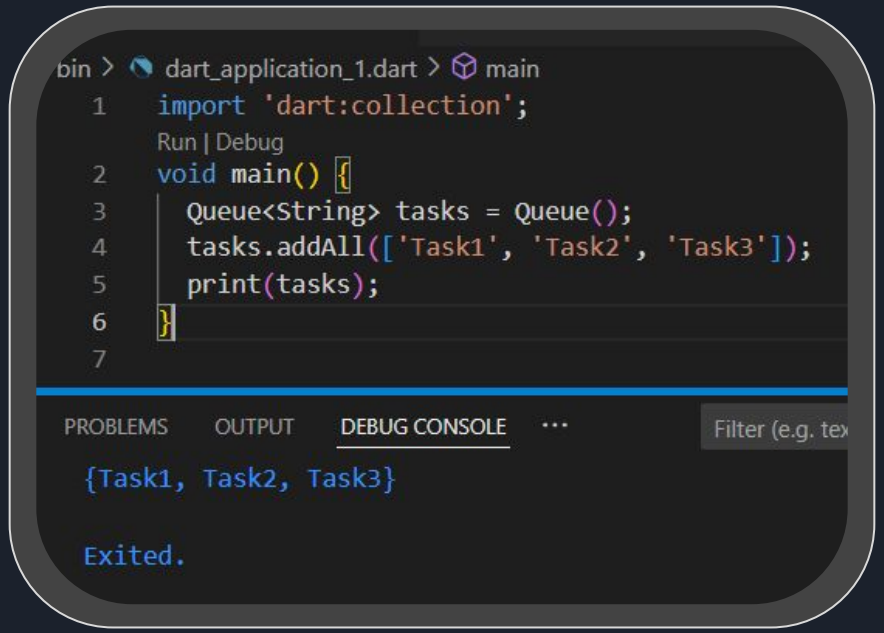
Using literal:

Using constructor:

Queue

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

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



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

```dart
void main(){
  var petPrices ={'Bella': 1.5,'Lucy':0.8,'Loki':1.2,'Leo':2.0,'oggy':3.5};
  print('Pet Prices: $petPrices');

  var price= petPrices.values;
  double cost=price.reduce((current,nextval)=> current+nextval);
  print('Total Cost : $cost');

}
```

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

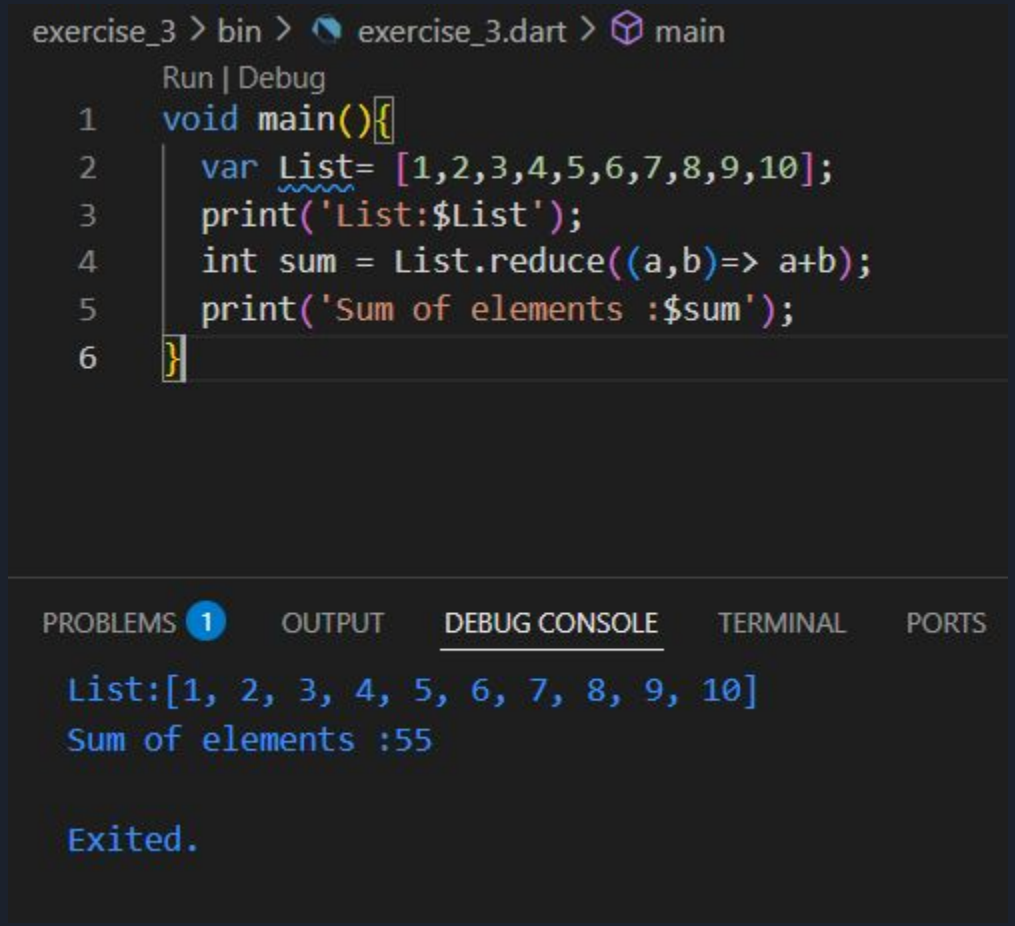
Run | Debug

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

}
```

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

```dart
Future<String> fetchData()
 async{
  await Future.delayed(Duration(seconds: 3 ));
  return"Some  Data from the server";
}
Run | Debug
void main()
async{

  print('program started');
  print('fetching data...');
  String data= await fetchData();
  print('Data recived : $data');
  print('Program ended');

}
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
program started
fetching data...
Data recived : 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

```dart
import 'dart:collection';

Run | Debug
void main(){
  final queue =Queue<int>();
  //print(queue.runtimeType);
  queue.addAll([10,20,30,40,50]);
  print('Initial Queue :$queue');

  int firstqueue =queue.first;
  print('First element of the Queue :$firstqueue');

  print('Last element of the Queue :${queue.last}');

  int remove1 = queue.removeFirst();
  print('Removed element :$remove1');

  int remove2 = queue.removeFirst();
  print('Removed element :$remove2');

  int remove3 = queue.removeFirst();
  print('Removed element :$remove3');

  int remove4 = queue.removeFirst();
  print('Removed element :$remove4');

  int remove5 = queue.removeFirst();
  print('Removed element :$remove5');

  print('Queue After Removal :$queue');
}
```