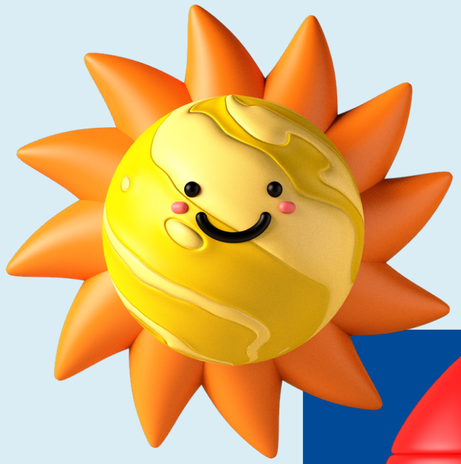
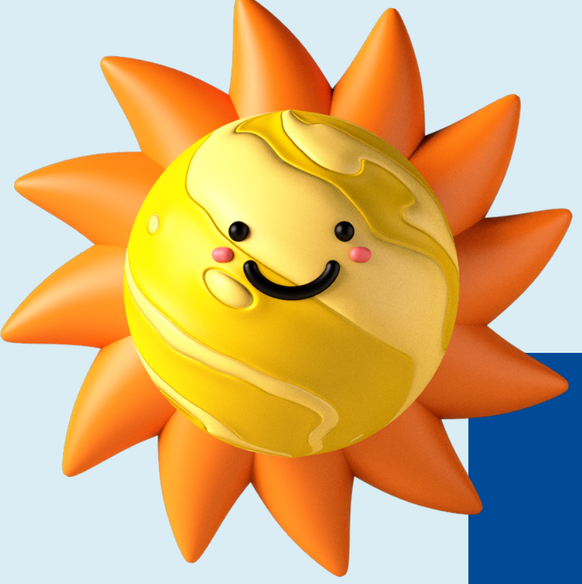


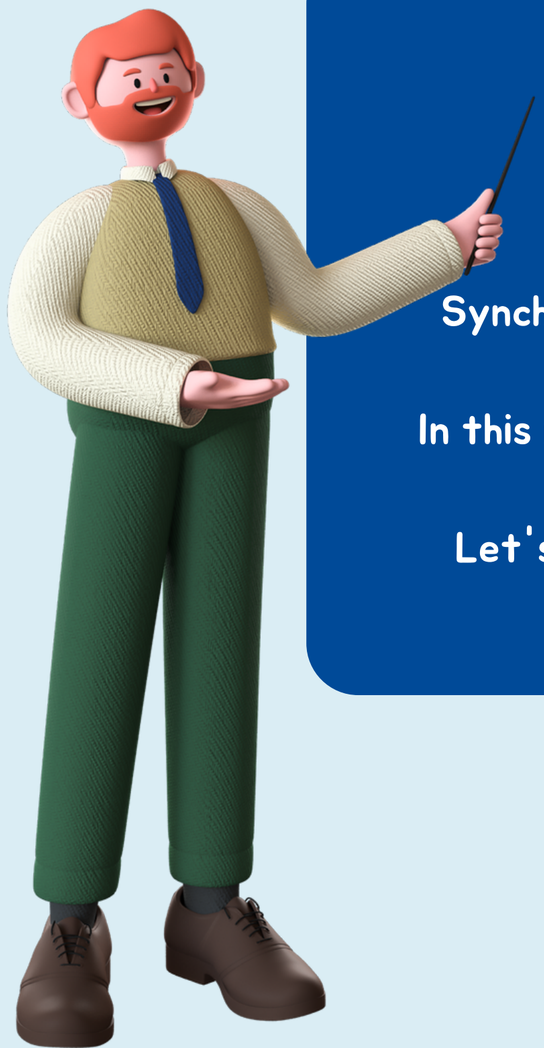
API INTEGRATION

WORKSHOP 6



CONTENTS

Workshop's topics range from types of programming (sync, async), exception handling in Dart, making http requests & Dart futures.
JSON parsing and dynamic data types will be discussed.
We will code a weather app.
Finally, we will take a look at various bonus resources.

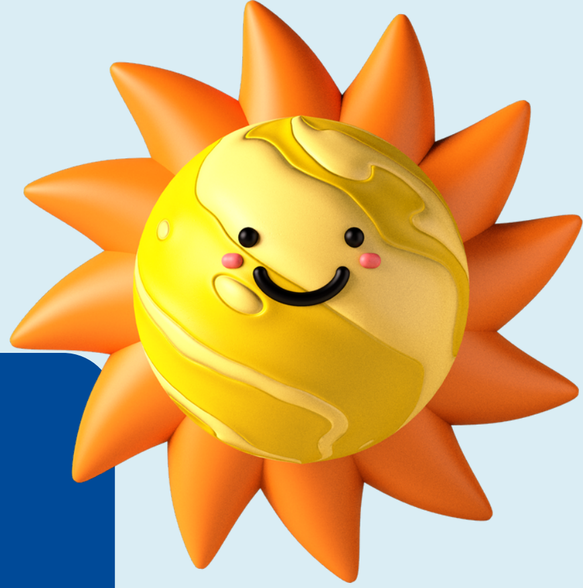


Synchronous Programming

Synchronous programming is a programming model where operations take place sequentially.

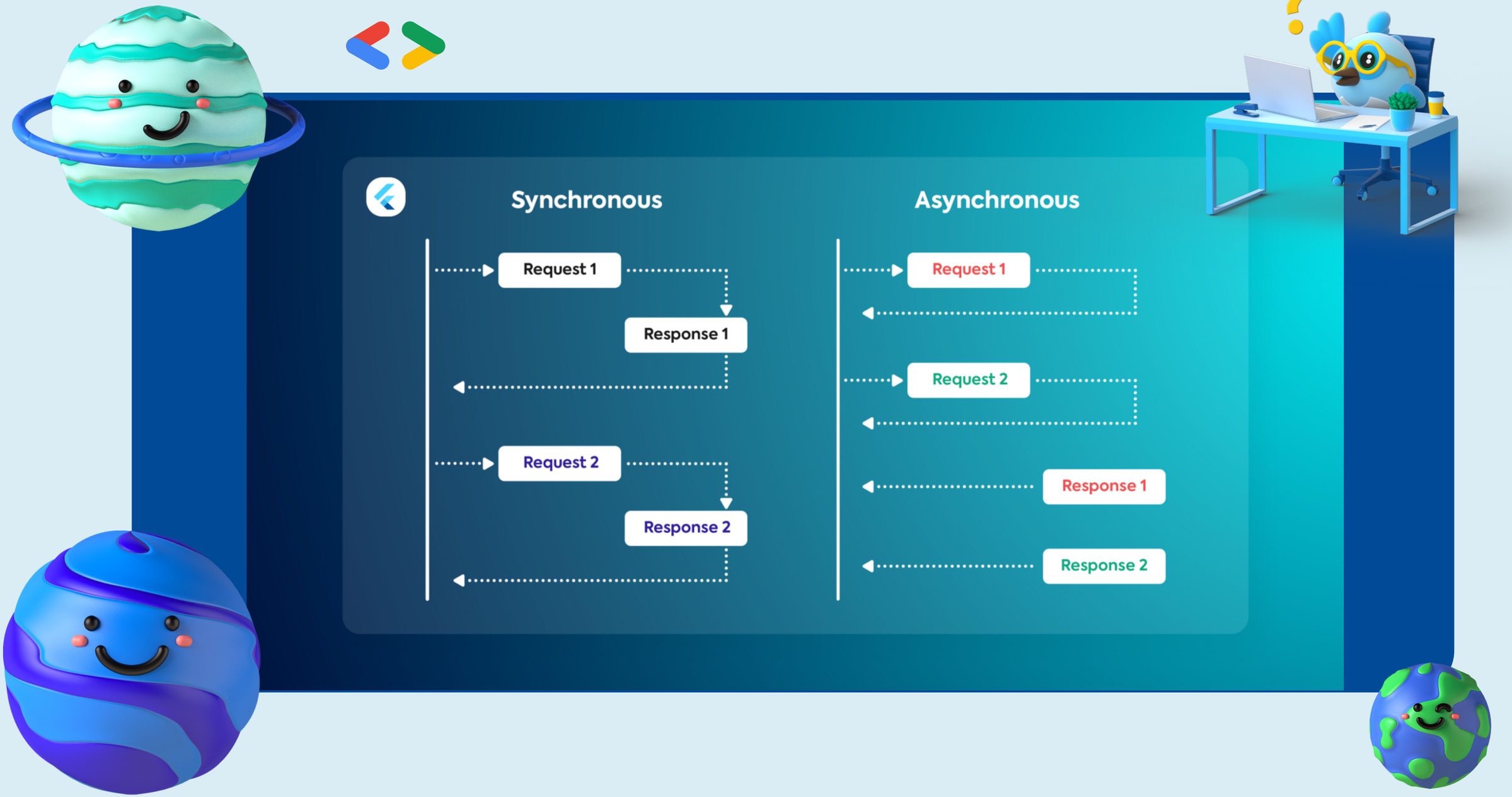
In this model, upcoming code execution will wait for the previous code to run completely.

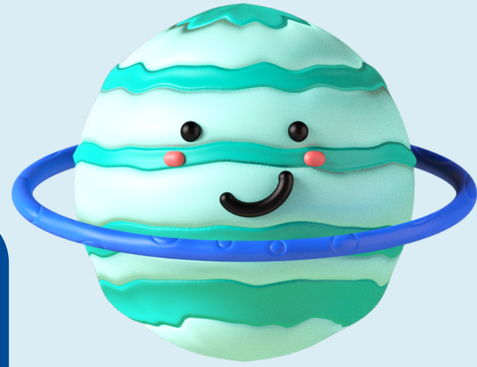
Let's take an example of an app in which a function requires an input or makes a request for data which will take some time to arrive.



Asynchronous Programming

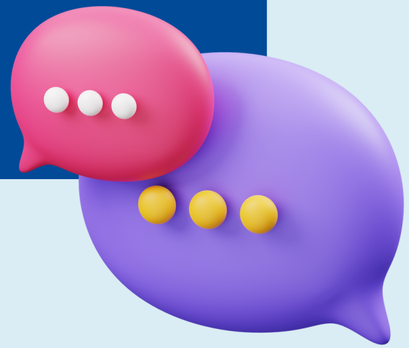
Asynchronous programming is a type of equal programming that permits a unit of work to run independently from the essential application thread. At the point when the work is finished, it tells the main thread.
For an example: we write asynchronous code to run along with main thread, so nothing gets delayed in waiting for any data.

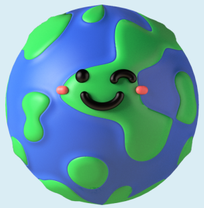

Synchronous
Request 1
Response 1
Request 2
Response 2
Asynchronous
Request 1
Request 2
Response 1
Response 2

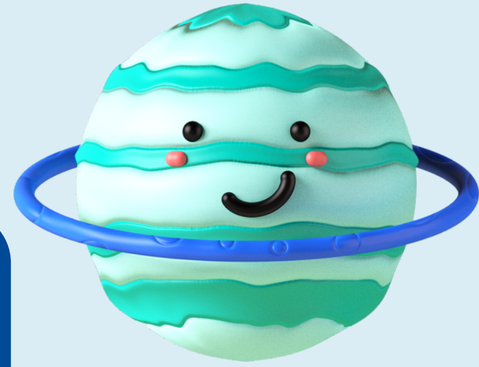


Why we should use Asynchronous Programming?

Although the asynchronous programming may not be the best in all scenarios, yet it can be used in most cases to increase the efficiency of application. Consider a Flutter application in which you want to get the UI built without build method getting stuck in wait for data.







Why we should use Asynchronous Programming??

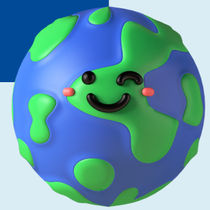
Asynchronous operations let your program complete work while waiting for another operation to finish. Here are some common asynchronous operations:

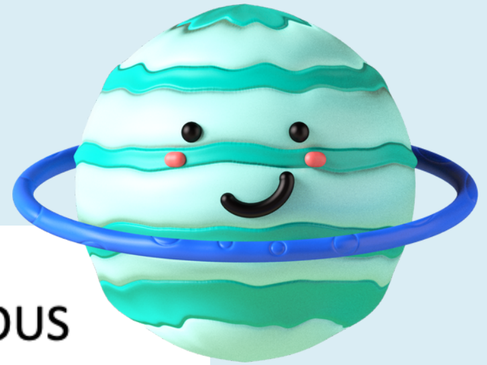
Fetching data over a network.

Writing to a database.

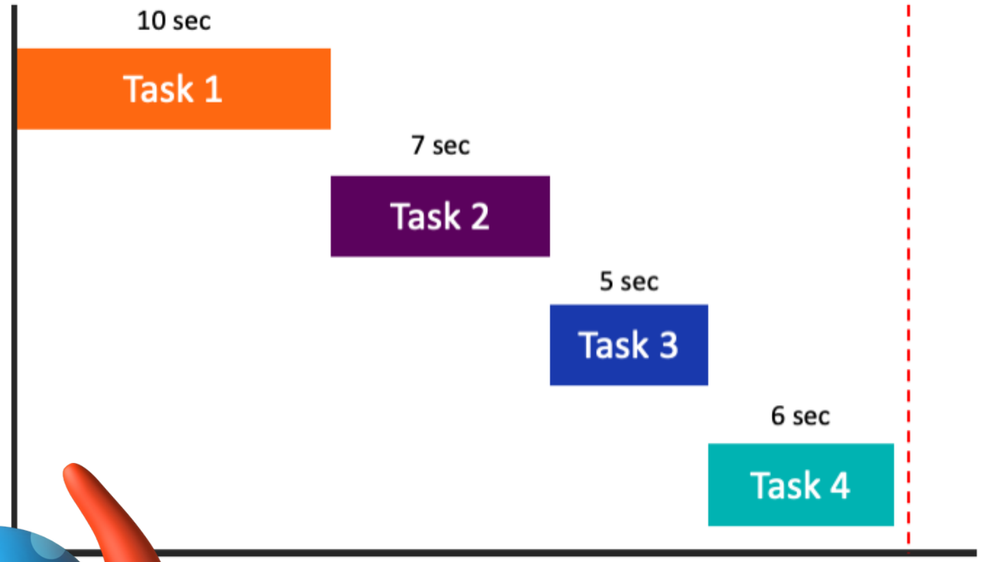
Reading data from a file.

To perform asynchronous operations in Dart, you can use the Future class and the async and await keywords.



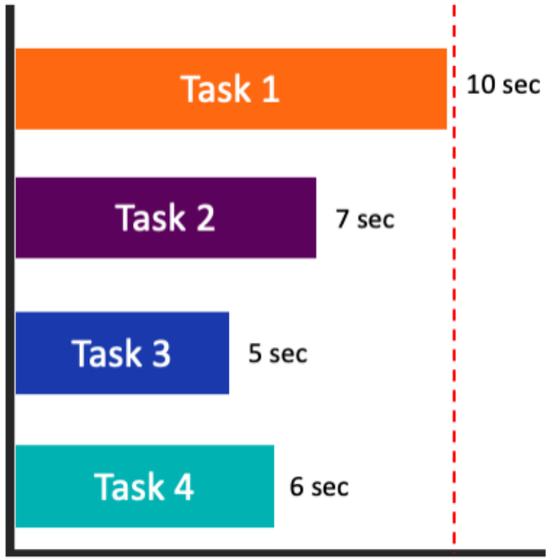


SYNCHRONOUS

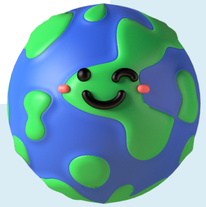


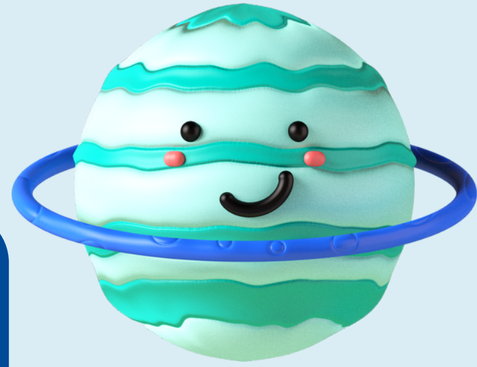
Time taken (28 sec)

ASYNCHRONOUS



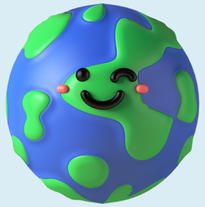
Time taken (10 sec)

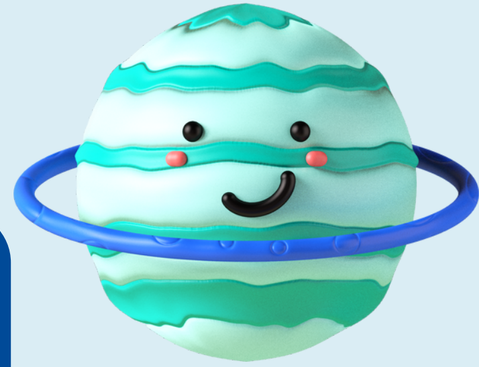




Dart Futures

A future represents the result of an asynchronous operation, and can have two states: uncompleted or completed.
When you call an asynchronous function, it returns an uncompleted future.
That future is waiting for the function's asynchronous operation to finish or to throw an error.
If the asynchronous operation succeeds, the future completes with a value.
Otherwise it completes with an error.

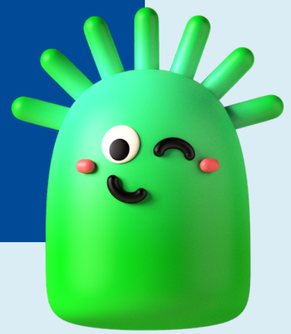




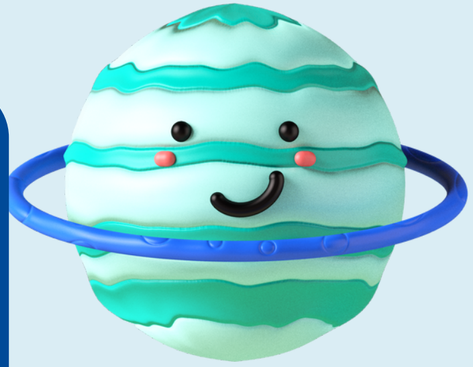
About 'async' & 'await'

The async and await keywords provide a declarative way to define asynchronous functions and use their results. Remember these two basic guidelines when using async and await:

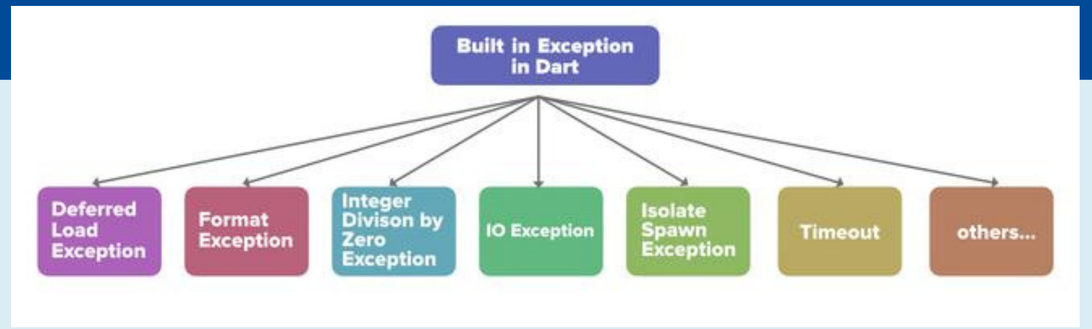
- To define an async function, add async before the function body:
- The await keyword works only in async functions.

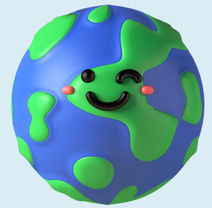


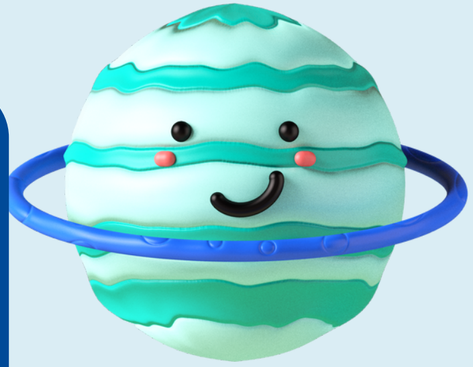
Exception Handling



An exception is an error that takes place inside the program. When an exception occurs inside a program the normal flow of the program is disrupted and it terminates abnormally, displaying the error and exception stack as output. So, an exception must be taken care to prevent the application from termination.



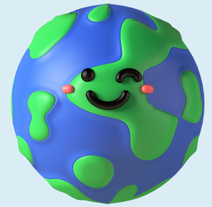




Try, On, & Catch

The try block embeds code that might possibly result in an exception. The on block is used when the exception type needs to be specified. The catch block is used when the handler needs the exception object.

Let's see some code



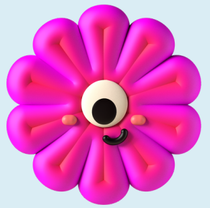
DYNAMIC DATA TYPE

Dart is an optionally typed language. If the type of a variable is not explicitly specified, the variable's type is dynamic. Using dynamic data type can change TYPE of the variable, & can change VALUE of the variable later in code.

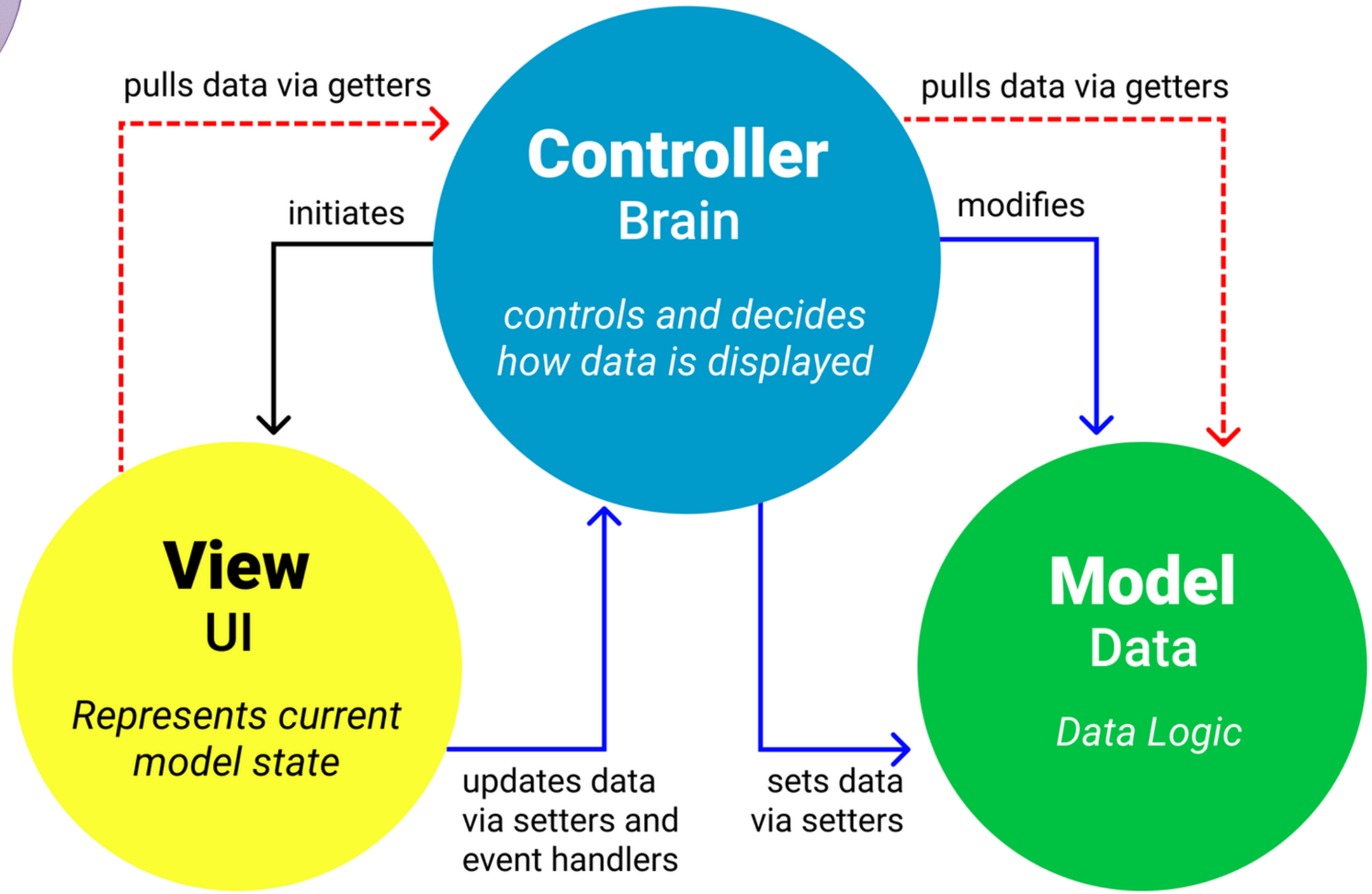
```dart
void main() {
  dynamic value;

  value = 'Rashid';
  print(value.runtimeType);
  // output: String

  value = 3.0;
  print(value.runtimeType);
  // output: double
}
```



MVC Architecture Pattern



GetX - Flutter Made Easier

get 4.6.1

Published 47 days ago• getx.site Null safety • Latest: 4.6.1 / Prerelease: 5.0.0-beta.15

FLUTTER | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS

7.81K

Readme | Changelog | Example | Installing | Versions | Scores

WHY GETX?

1: Reduced boilerplate code.
2: Fast
3: Takes very less space
4: State Management
5: Route Management
5: Validation & Storage Support

INTRO TO DIO PACKAGE

A powerful Http client for Dart, which supports Interceptors, Global configuration, FormData, Request Cancellation, File downloading, Timeout etc.

dio 4.0.4

Published 2 months ago• flutterchina.club Null safety • Latest: 4.0.

DART | NATIVE | JS | FLUTTER | ANDROID | IOS | LINUX | MACOS | WEB | W

Readme | Changelog | Example | Installing | Versions | S

Language: English | 中文简体

OPENWEATHERMAP API

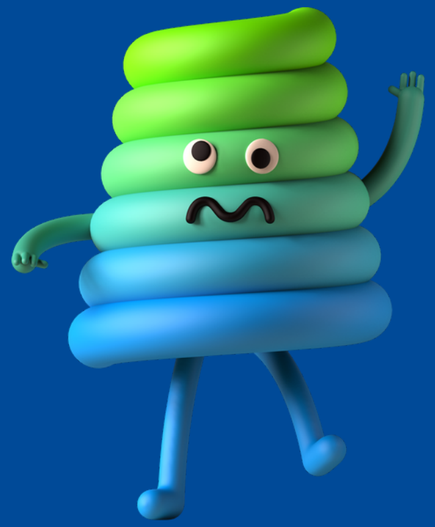
LET'S TAKE A TOUR OF THE WEBSITE


Let's get started with code...

Useful Resources

GENERAL

- jsonplaceholder
- openweathermap
- reqres.in

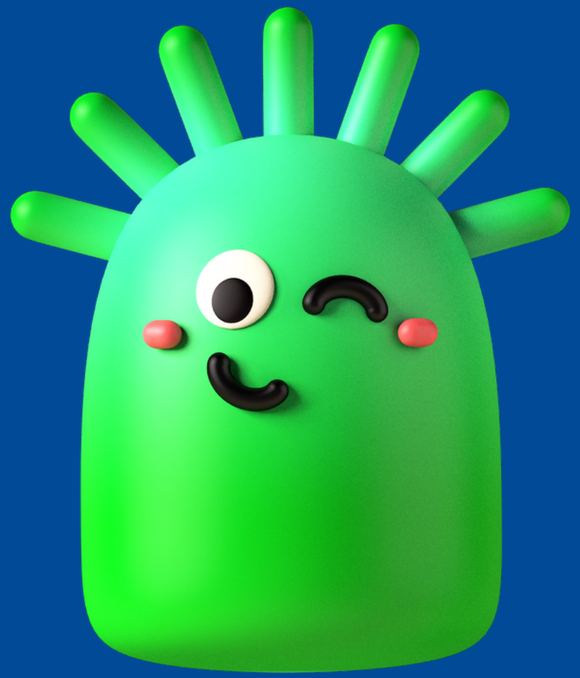


FOR DESIGNS

- Appbrainy's Instagram page.
- Dribble.
- free3dicon.com

LEARNING Resources

1. Flutter Docs
2. The complete Flutter Bootcamp By Angela Yu
3. Flutter Course by Academind (Comprehensive)
4. Google Developers
5. Flutter (widget of the week)
6. Codepur
7. Reso Coder
8. Filled Stacks
9. Fireship
10. Robert Brunhage



Tips to avoid loosing interest in learning

1. Focus on project based learning.
2. Create fun projects or fun elements in projects.
3. Don't worry about syntax, learn conceptually.
4. Subscribe YouTube channels for Flutter and enable notifications.
5. Explore different features of framework.