

Programmazione Mobile

Nicola Noviello


nicola.noviello@unimol.it

Corso di Laurea in Informatica
Dipartimento di Bioscienze e Territorio
Università degli Studi del Molise
Anno 2023/2024

Lezione: Flutter e Dart Foundation (parte prima)

- Basi di Flutter
- Sintassi di Dart
- Comprendere il funzionamento del framework
- Primi approcci alla creazione di un progetto
- Widgets





La prima applicazione Flutter: Struttura del progetto

Comandi base

flutter doctor

Utilizzato per diagnosticare e risolvere eventuali problemi nell'ambiente di sviluppo, esamina le dipendenze del sistema e fornisce raccomandazioni per l'installazione o la configurazione corretta.

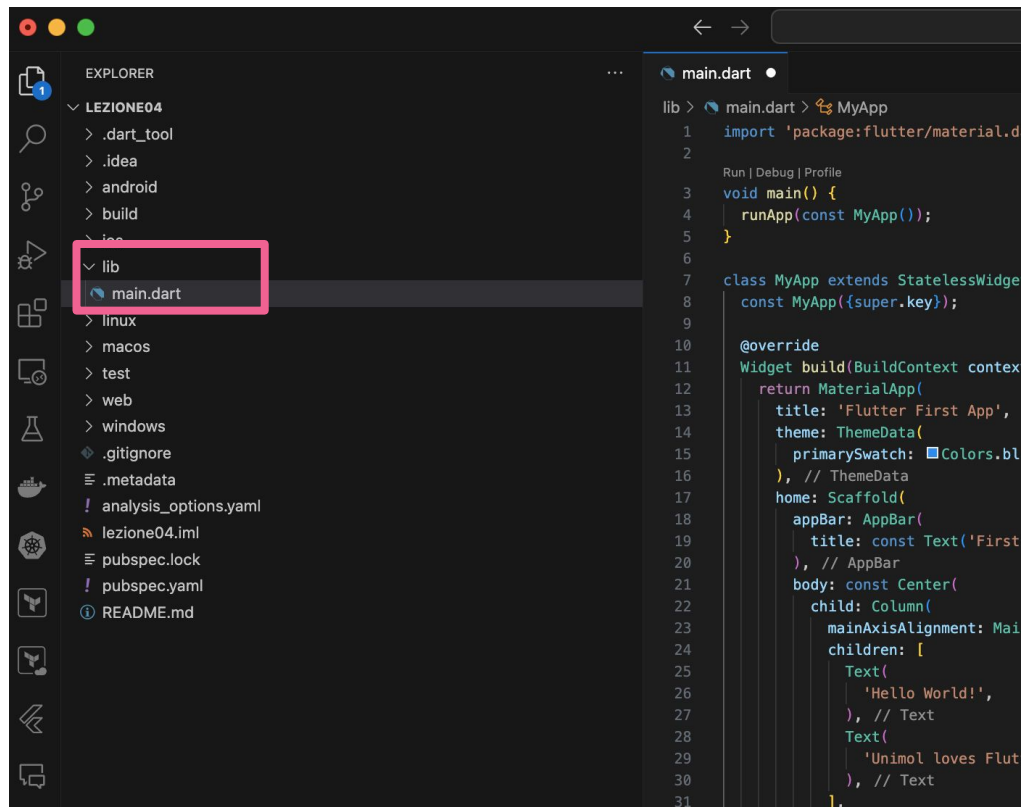
flutter upgrade

Aggiorna l'installazione di Flutter alla versione più recente disponibile. Garantisce l'accesso alle ultime funzionalità, bug fix e miglioramenti di prestazioni.

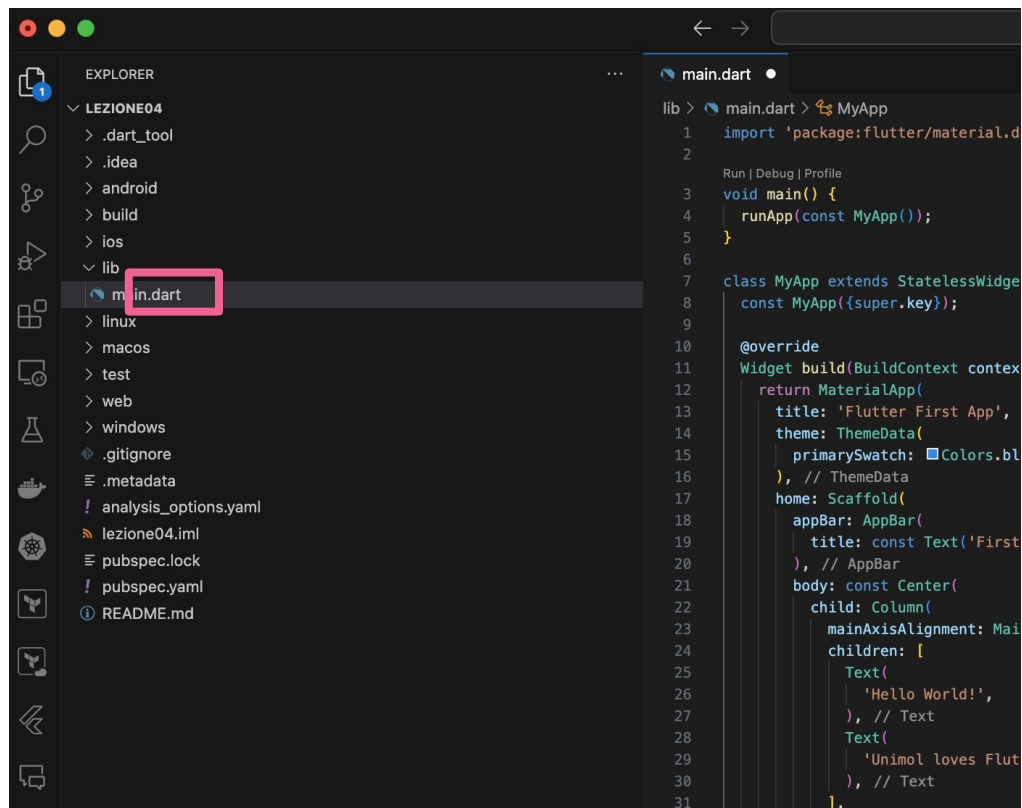
flutter create nome_progetto

Crea un nuovo progetto Flutter. Genera una struttura di base per un'applicazione Flutter, inclusi i file di codice necessari e la configurazione del progetto.

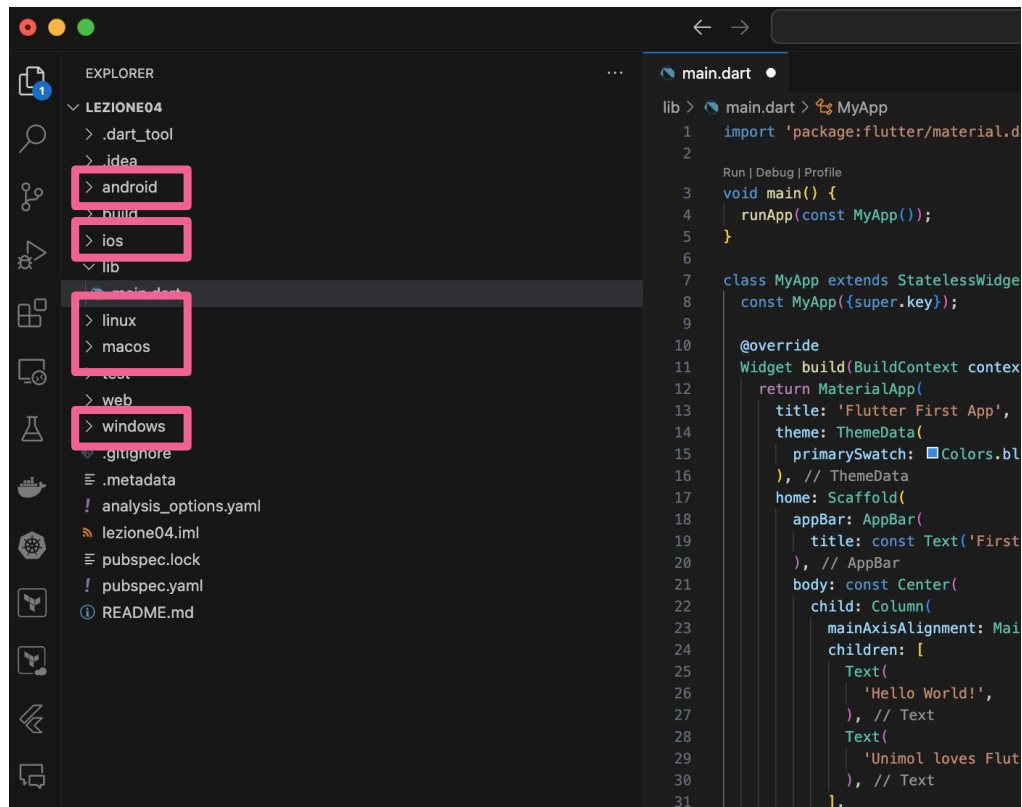
Struttura di un'App Flutter



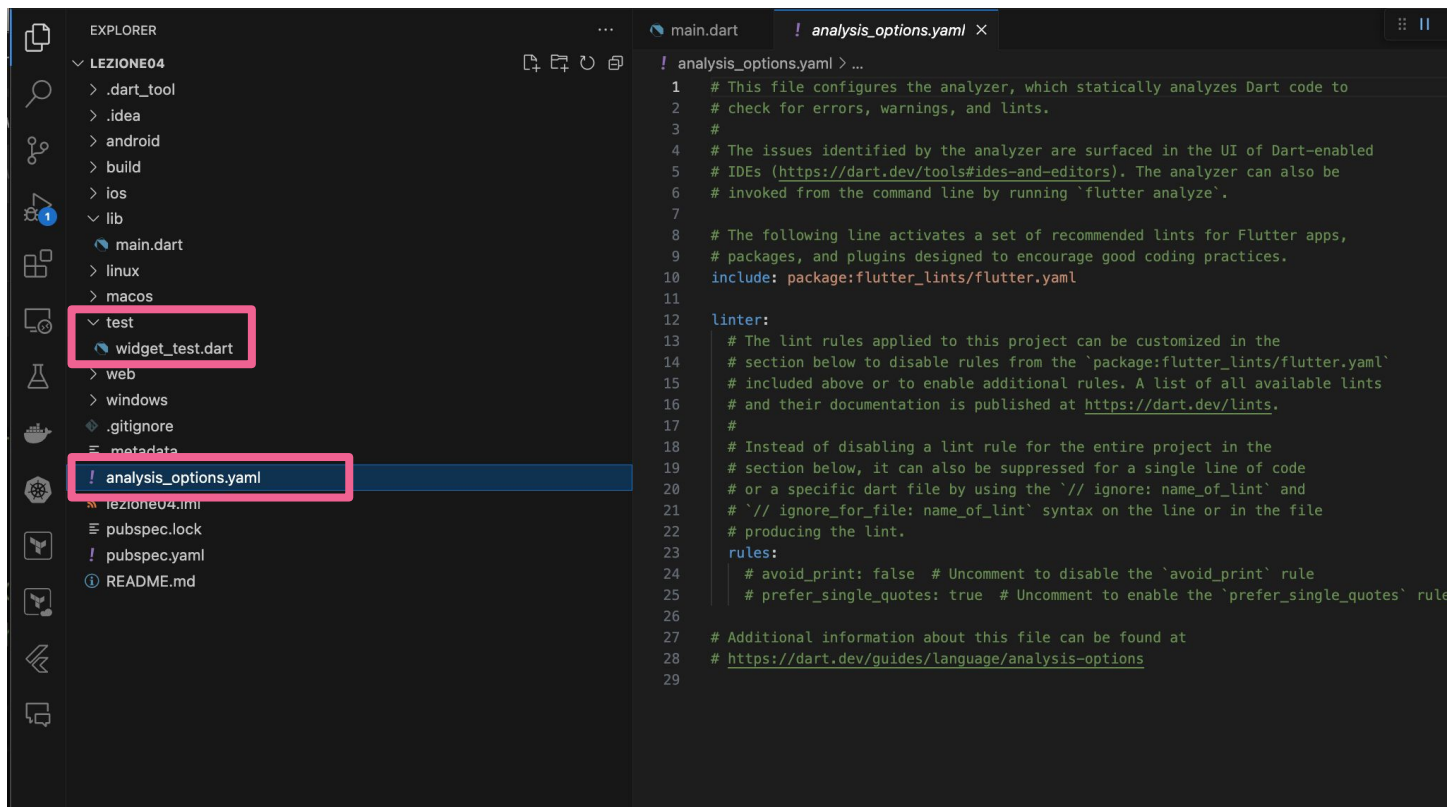
Struttura di un'App Flutter



Struttura di un'App Flutter



Struttura di un'App Flutter



Linter

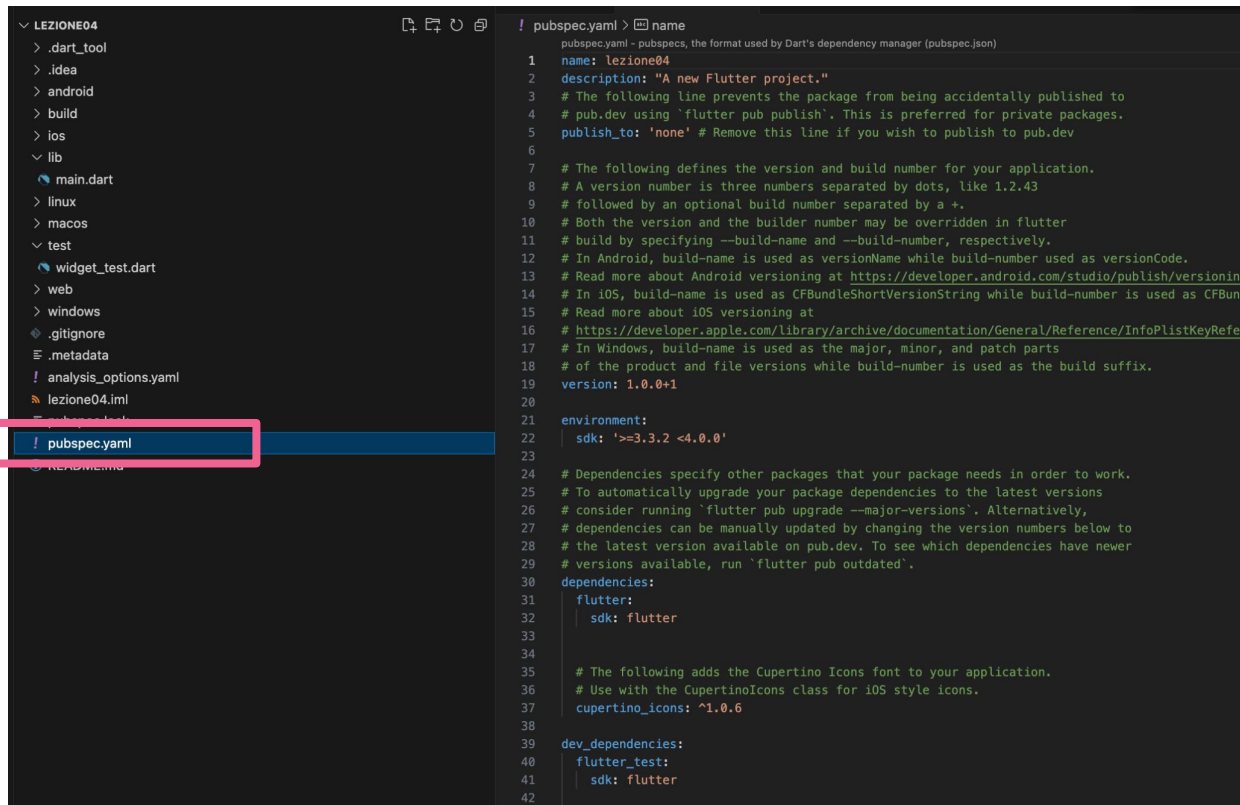
Un **linter** (o **lint**) è uno strumento che analizza il codice sorgente per segnalare errori di programmazione, bug, costrutti stilistici discutibili e costrutti sospetti.

In **Dart**, il **linter** è particolarmente importante perché aiuta a mantenere un codice pulito e coerente, aderendo alle linee guida della community Dart, aiutando a prevenire errori comuni e a migliorare le prestazioni del codice.

<https://dart.dev/tools/linter-rules>



Struttura di un'App Flutter



The image shows a code editor with two panels. The left panel displays the project structure of a Flutter application named 'LEZIONE04'. The right panel shows the content of the 'pubspec.yaml' file.

Project Structure (Left Panel):

- LEZIONE04
 - .dart_tool
 - .idea
 - android
 - build
 - ios
 - lib
 - main.dart
 - linux
 - macos
 - test
 - widget_test.dart
 - web
 - windows
 - .gitignore
 - .metadata
 - analysis_options.yaml
 - lezione04.iml
 - pubspec.yaml (highlighted with a red box)
 - README.md

pubspec.yaml File (Right Panel):

```
! pubspec.yaml > name
pubspec.yaml - pubspecs, the format used by Dart's dependency manager (pubspec.json)
1 name: lezione04
2 description: "A new Flutter project."
3 # The following line prevents the package from being accidentally published to
4 # pub.dev using `flutter pub publish`. This is preferred for private packages.
5 publish_to: 'none' # Remove this line if you wish to publish to pub.dev
6
7 # The following defines the version and build number for your application.
8 # A version number is three numbers separated by dots, like 1.2.43
9 # followed by an optional build number separated by a +.
10 # Both the version and the builder number may be overridden in flutter
11 # build by specifying --build-name and --build-number, respectively.
12 # In Android, build-name is used as versionName while build-number used as versionCode.
13 # Read more about Android versioning at https://developer.android.com/studio/publish/versioning
14 # In iOS, build-name is used as CFBundleShortVersionString while build-number is used as CFBundleVersion
15 # Read more about iOS versioning at
16 # https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/LaunchServices.html#Task/InfoPlistKeyReference/Version
17 # In Windows, build-name is used as the major, minor, and patch parts
18 # of the product and file versions while build-number is used as the build suffix.
19 version: 1.0.0+1
20
21 environment:
22   sdk: '>=3.3.2 <4.0.0'
23
24 # Dependencies specify other packages that your package needs in order to work.
25 # To automatically upgrade your package dependencies to the latest versions
26 # consider running `flutter pub upgrade --major-versions`. Alternatively,
27 # dependencies can be manually updated by changing the version numbers below to
28 # the latest version available on pub.dev. To see which dependencies have newer
29 # versions available, run `flutter pub outdated`.
30 dependencies:
31   flutter:
32     sdk: flutter
33
34
35 # The following adds the Cupertino Icons font to your application.
36 # Use with the CupertinoIcons class for iOS style icons.
37 cupertino_icons: ^1.0.6
38
39 dev_dependencies:
40   flutter_test:
41     sdk: flutter
42
```

Run del codice: cosa succede?

Parsing:
Top
to
Bottom

```
main.dart x ! pubspec.yaml
lib > main.dart > MyApp > build
1 import 'package:flutter/material.dart';
2
3 Run | Debug | Profile
4 void main() {
5   runApp(const MyApp());
6 }
7
8 class MyApp extends StatelessWidget {
9   const MyApp({super.key});
10
11   @override
12   Widget build(BuildContext context) {
13     return MaterialApp(
14       title: 'Flutter First App',
15       theme: ThemeData(
16         primarySwatch: Colors.blue,
17       ), // ThemeData
18       home: Scaffold(
19         appBar: AppBar(
20           title: const Text('First App'),
21         ), // AppBar
22         body: const Center(
23           child: Column(
24             mainAxisAlignment: MainAxisAlignment.center,
25             children: [
26               Text(
27                 'Hello World!',
28               ), // Text
29               Text(
30                 'Unimol loves Flutter!',
31               ), // Text
32             ],
33           ), // Column
34         ), // Center
35       ), // Scaffold
36     ); // MaterialApp
37 }
```

Compilato dai tool Flutter e
Dart in codice nativo iOS,
Android, etc.

Eseguito sul(i)
device di
destinazione



Titolo: Il nostro primo
“Hello World”

Sottotitolo: L’ “Hello
World” più lungo del
mondo

Funzioni

- Funzione specifica di Flutter (non è di Dart).
- Necessaria in tutte le App.
- Mostra una user interface (in questo caso vuota) sullo schermo



Ma perché l'IDE ci segnala un errore?

A screenshot of an IDE showing a Dart file named main.dart. The file contains a single line of code: runApp();. The IDE's breadcrumb navigation shows the path: lib > main.dart > runApp. The code is on line 2, and there is a red squiggly line under the closing parenthesis of runApp(), indicating a compiler error. The IDE interface is dark-themed, and the file name 'main.dart 1' is visible in the top tab.

Funzioni

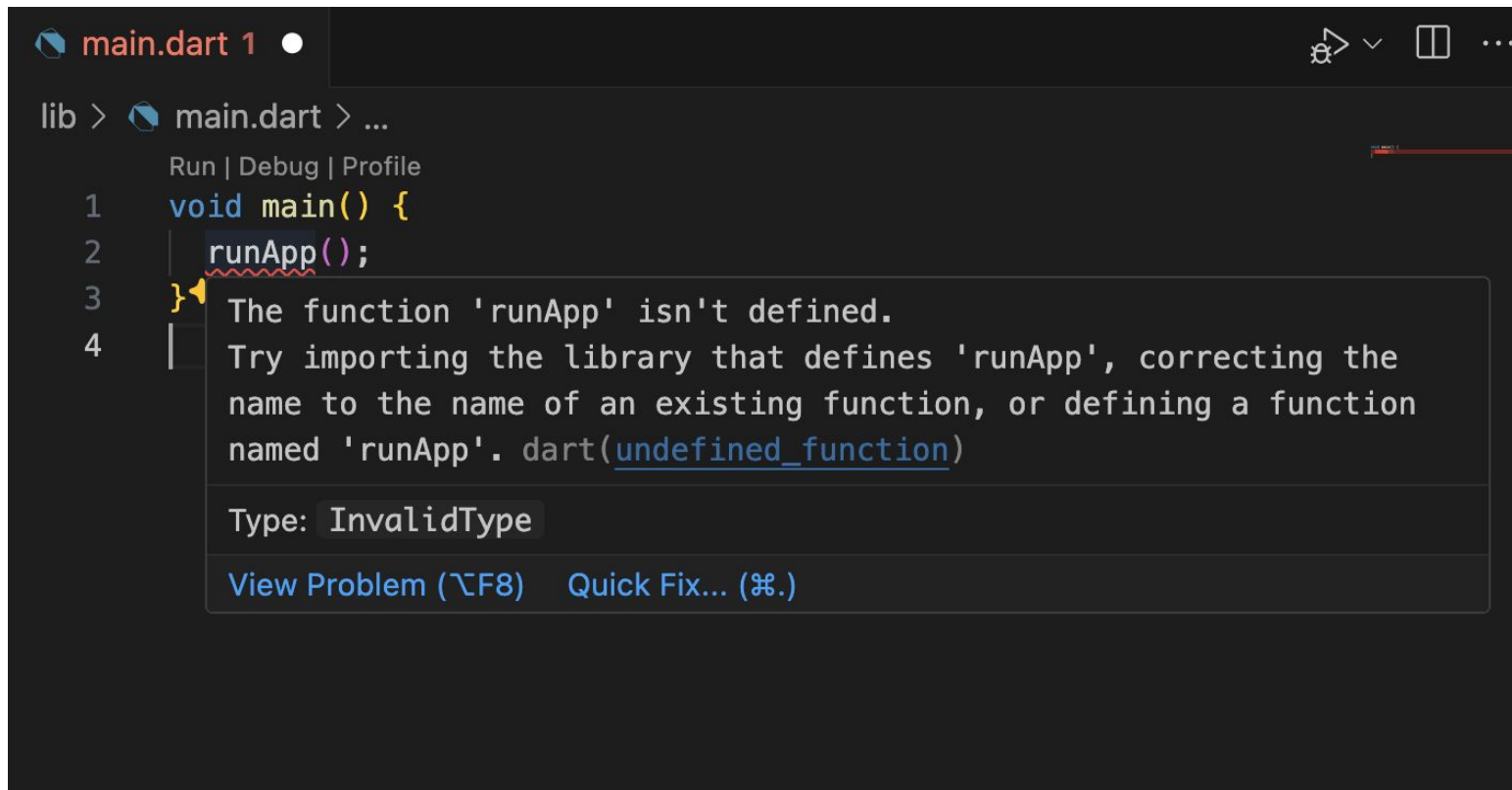
- Funzione specifica di Flutter (non è di Dart).
- Necessaria in tutte le App.
- Mostra una user interface (in questo caso vuota) sullo schermo



Ma perché l'IDE ci segnala un errore?

A screenshot of an IDE showing a Dart file named 'main.dart'. The code contains a single line: `runApp();`. There is a red squiggly line under the opening parenthesis of `runApp()`. A tooltip error message is displayed below the code, stating: 'A function body must be provided. Try adding a function body. dart(missing_function_body)'. Below the error message are two links: 'View Problem (\F8)' and 'Quick Fix... (\#.)'. The IDE interface includes a top bar with icons for running, saving, and other actions, and a breadcrumb trail showing 'lib > main.dart > runApp'.

Funzioni



The screenshot shows an IDE window titled "main.dart 1". The editor displays the following code:

```
lib > main.dart > ...  
Run | Debug | Profile  
1 void main() {  
2   runApp();  
3 }  
4
```

A red squiggly line is under the `runApp()` call on line 2. A tooltip is displayed over this line, containing the following text:

The function 'runApp' isn't defined.
Try importing the library that defines 'runApp', correcting the name to the name of an existing function, or defining a function named 'runApp'. dart([undefined_function](#))

Below the tooltip, the text "Type: InvalidType" is shown. At the bottom of the tooltip, there are two links: "View Problem (\F8)" and "Quick Fix... (\#.)".

Funzioni

lib >  main.dart > ...

```
1  import 'package:flutter/material.dart';
```

Run | Debug | Profile

```
2  void main() {
```

```
3    | runApp();
```

```
4  }
```

```
5
```

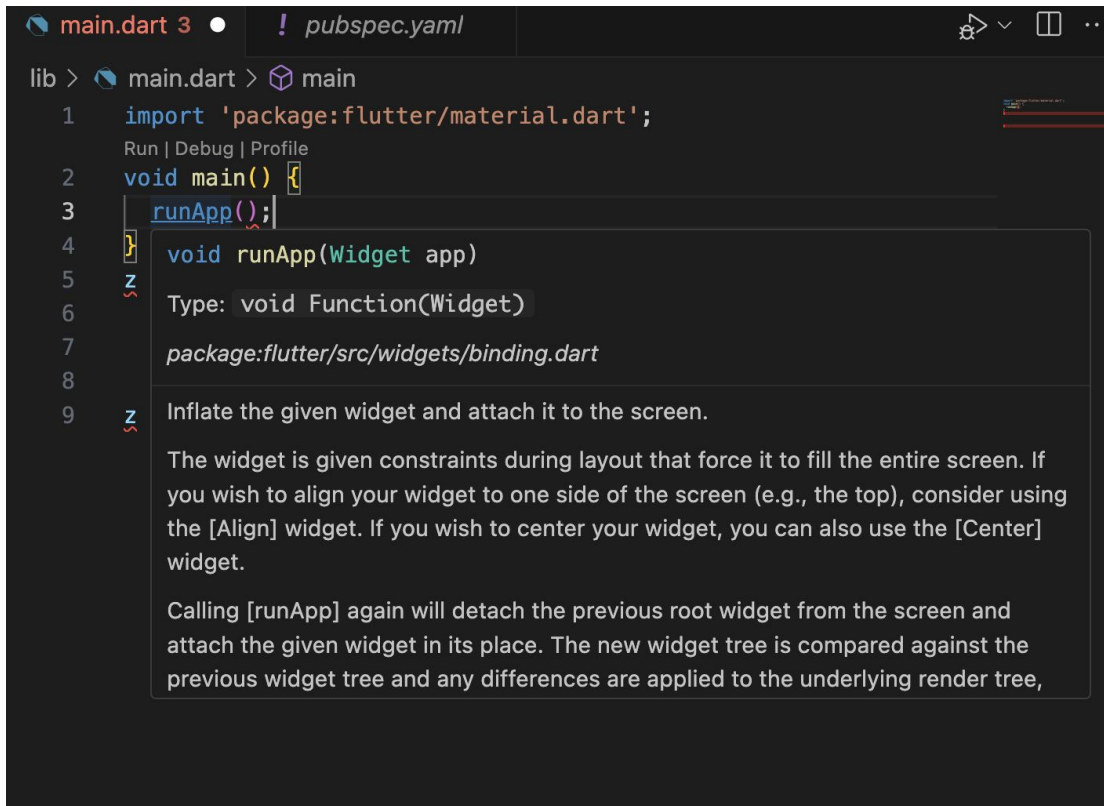
```
6
```

```
7
```

lib > ... > Flutter/material.dart >
1 import 'package:flutter/material.dart';
2 runApp();
3 }

Funzioni

Perché abbiamo ancora un errore?



```
main.dart 3 • ! pubspec.yaml
lib > main.dart > main
1 import 'package:flutter/material.dart';
  Run | Debug | Profile
2 void main() {
3   runApp();
4 }
5
6
7
8
9
```

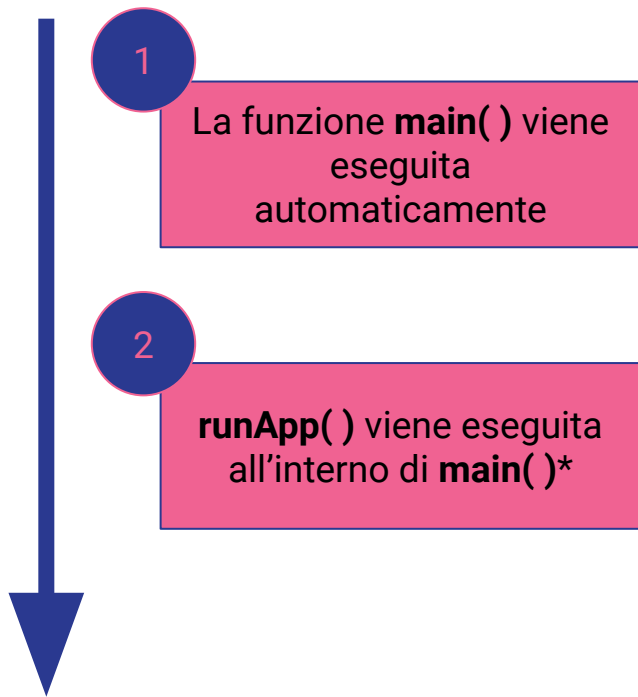
void runApp(Widget app)
Type: void Function(Widget)
package:flutter/src/widgets/binding.dart

Inflate the given widget and attach it to the screen.

The widget is given constraints during layout that force it to fill the entire screen. If you wish to align your widget to one side of the screen (e.g., the top), consider using the [Align] widget. If you wish to center your widget, you can also use the [Center] widget.

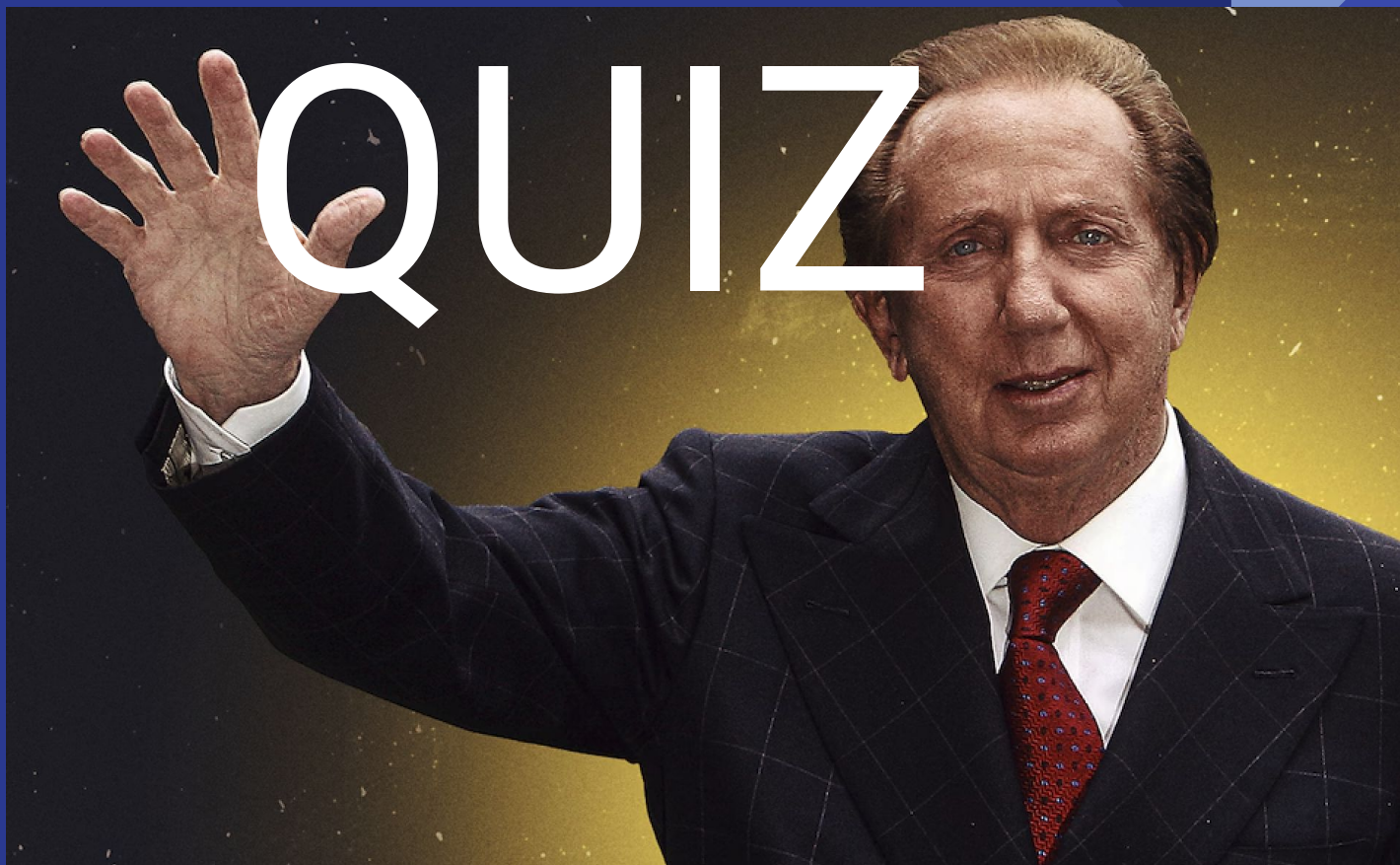
Calling [runApp] again will detach the previous root widget from the screen and attach the given widget in its place. The new widget tree is compared against the previous widget tree and any differences are applied to the underlying render tree,

Esecuzione di un'App Flutter



Quando? Quando Dart esegue l'App compilata sul device target

runApp() dice a Flutter cosa mostrare sullo schermo



Domanda 1

Qual è la directory più importante in un progetto Flutter nella quale lavorerete la maggior parte del vostro tempo?

- La directory “web”
- La directory “android”
- <https://chat.openai.com>
- La directory “lib”
- La directory “pubspec”



Domanda 1

Qual è la directory più importante in un progetto Flutter nella quale lavorerete la maggior parte del vostro tempo?

- La directory “web”
- La directory “android”
- <https://chat.openai.com>
- **La directory “lib”**
- La directory “pubspec”



Domanda 2

Quale file è l'entry-point di un'applicazione Flutter?

- android/app/build.gradle
- lib/app.dart
- pubspec.yaml
- lib/main.dart
- lib/index.html



Domanda 2

Quale file è l'entry-point di un'applicazione Flutter?

- android/app/build.gradle
- lib/app.dart
- pubspec.yaml
- **lib/main.dart**
- lib/index.html



Domanda 3

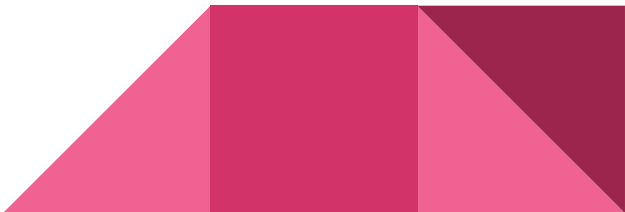
Cosa fa un compilatore Dart?

- genera la user interface per le applicazioni Flutter
- importa automaticamente i Package in un progetto dart
- converte il codice Dart in codice nativo che può essere eseguito su diverse piattaforme
- converte il codice Dart nel codice usato per le diverse piattaforme, i diversi codici poi vengono compilati ed eseguiti sui vari dispositivi
- si occupa del debug del codice Dart



Domanda 3

Cosa fa un compilatore Dart?

- genera la user interface per le applicazioni Flutter
 - importa automaticamente i Package in un progetto dart
 - **converte il codice Dart in codice nativo che può essere eseguito su diverse piattaforme**
 - converte il codice Dart nel codice usato per le diverse piattaforme, i diversi codici poi vengono compilati ed eseguiti sui vari dispositivi
 - si occupa del debug del codice Dart
- 

Domanda 4

Cosa sono le funzioni nella programmazione?

- una sequenza di istruzioni scritte in codice macchina (machine code)
- dei “contenitori” di dati
- una sequenza di istruzioni per eseguire un task specifico
- dei mini tool per compilare il codice
- dei metodi per creare delle User Interface (UI)



Domanda 4

Cosa sono le funzioni nella programmazione?

- una sequenza di istruzioni scritte in codice macchina (machine code)
- dei “contenitori” di dati
- **una sequenza di istruzioni per eseguire un task specifico**
- dei mini tool per compilare il codice
- dei metodi per creare delle User Interface (UI)



Domanda 5

Come si importa un package in un file Dart?

- use "package_name";
- include "package_name";
- require "package_name";
- import "package_name";
- include "package_name"



Domanda 5

Come si importa un package in un file Dart?

- use "package_name";
- include "package_name";
- require "package_name";
- **import "package_name";**
- include "package_name"



Domanda 6

Quali elementi fondamentali sono coinvolti nel processo di creazione di una UI sullo schermo di un device?

- la directory main e la funzione runApp()
- la funzione runApp() e il widget Main
- la funzione main() e il widget App
- la funzione main() e la funzione runApp()
- la funzione runApp() e il file app.dart



Domanda 6

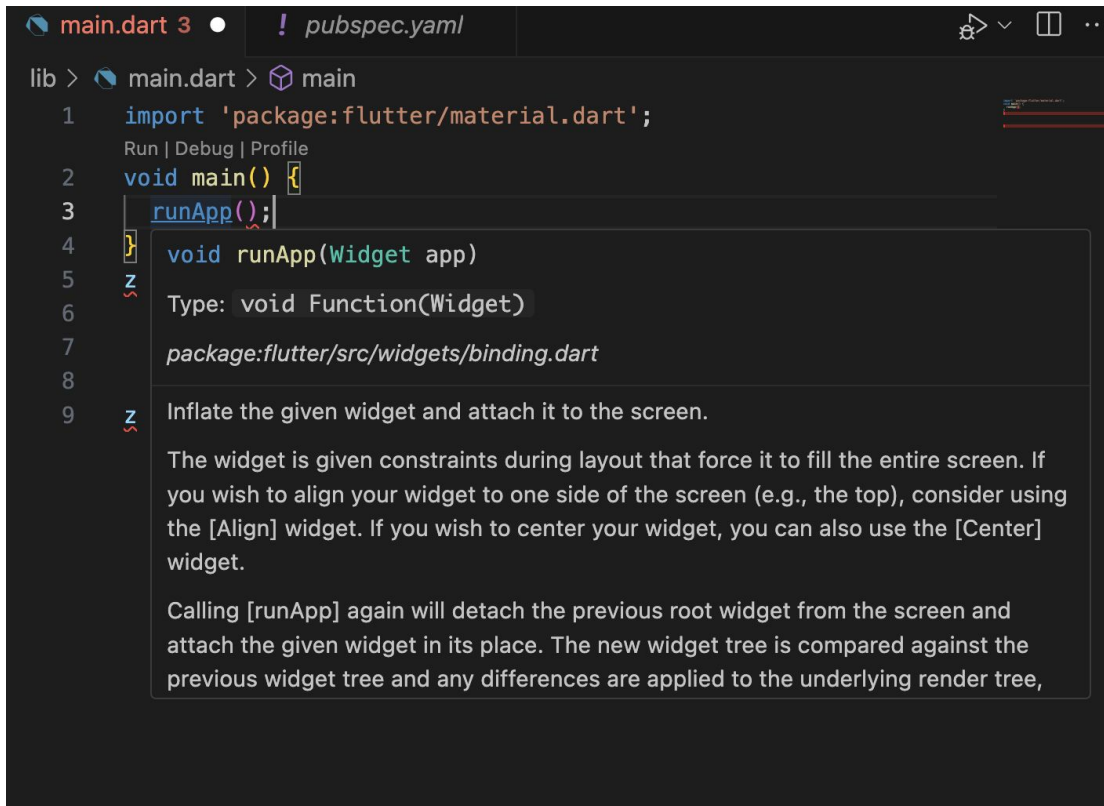
Quali elementi fondamentali sono coinvolti nel processo di creazione di una UI sullo schermo di un device?

- la directory main e la funzione runApp()
- la funzione runApp() e il widget Main
- la funzione main() e il widget App
- **la funzione main() e la funzione runApp()**
- la funzione runApp() e il file app.dart



Funzioni

Perché abbiamo ancora un errore?



```
main.dart 3 • ! pubspec.yaml
lib > main.dart > main
1 import 'package:flutter/material.dart';
  Run | Debug | Profile
2 void main() {
3   runApp();
4 }
5
6
7
8
9
```

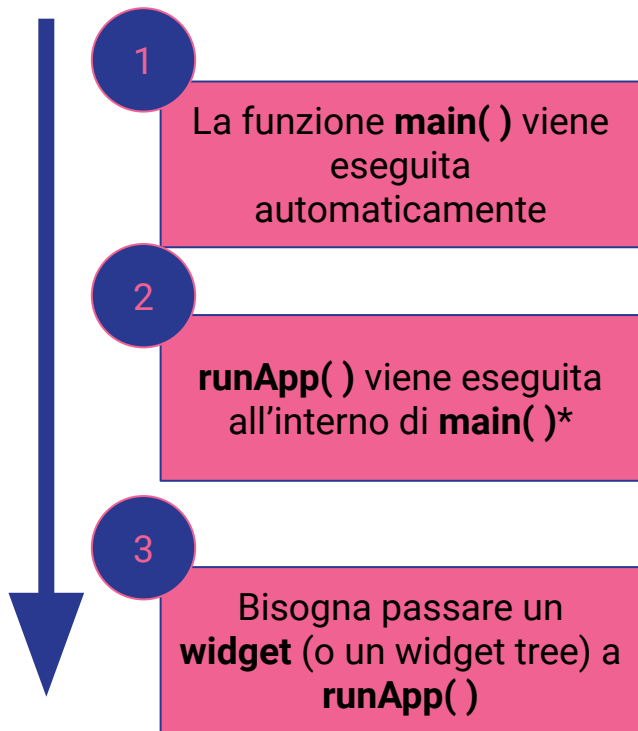
void runApp(Widget app)
Type: void Function(Widget)
package:flutter/src/widgets/binding.dart

Inflate the given widget and attach it to the screen.

The widget is given constraints during layout that force it to fill the entire screen. If you wish to align your widget to one side of the screen (e.g., the top), consider using the [Align] widget. If you wish to center your widget, you can also use the [Center] widget.

Calling [runApp] again will detach the previous root widget from the screen and attach the given widget in its place. The new widget tree is compared against the previous widget tree and any differences are applied to the underlying render tree,

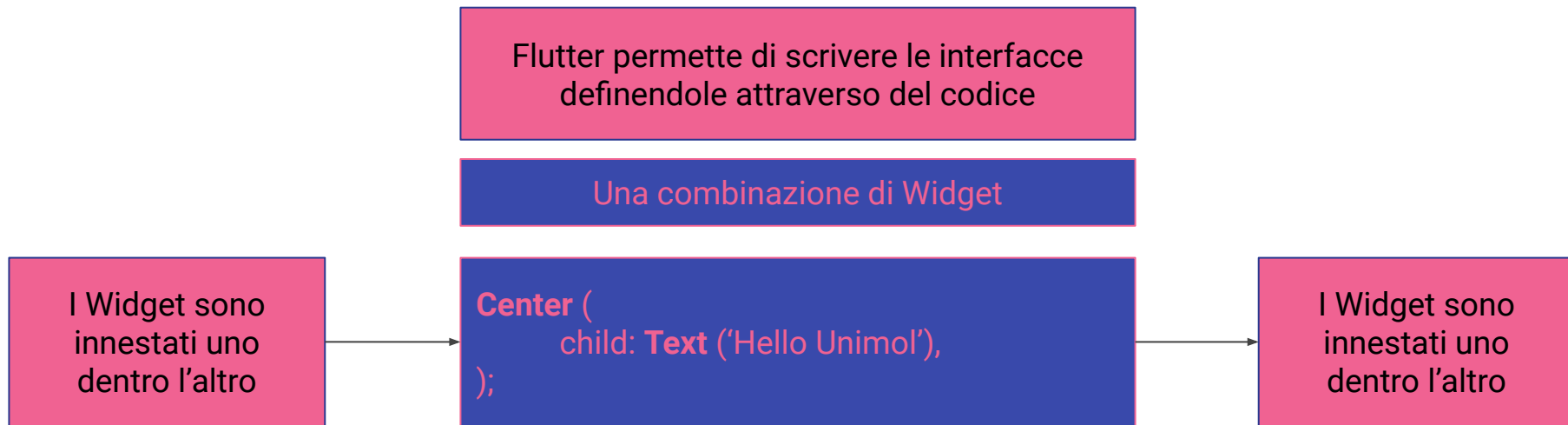
Esecuzione di un'App Flutter



Quando? Quando Dart esegue l'App compilata sul device target

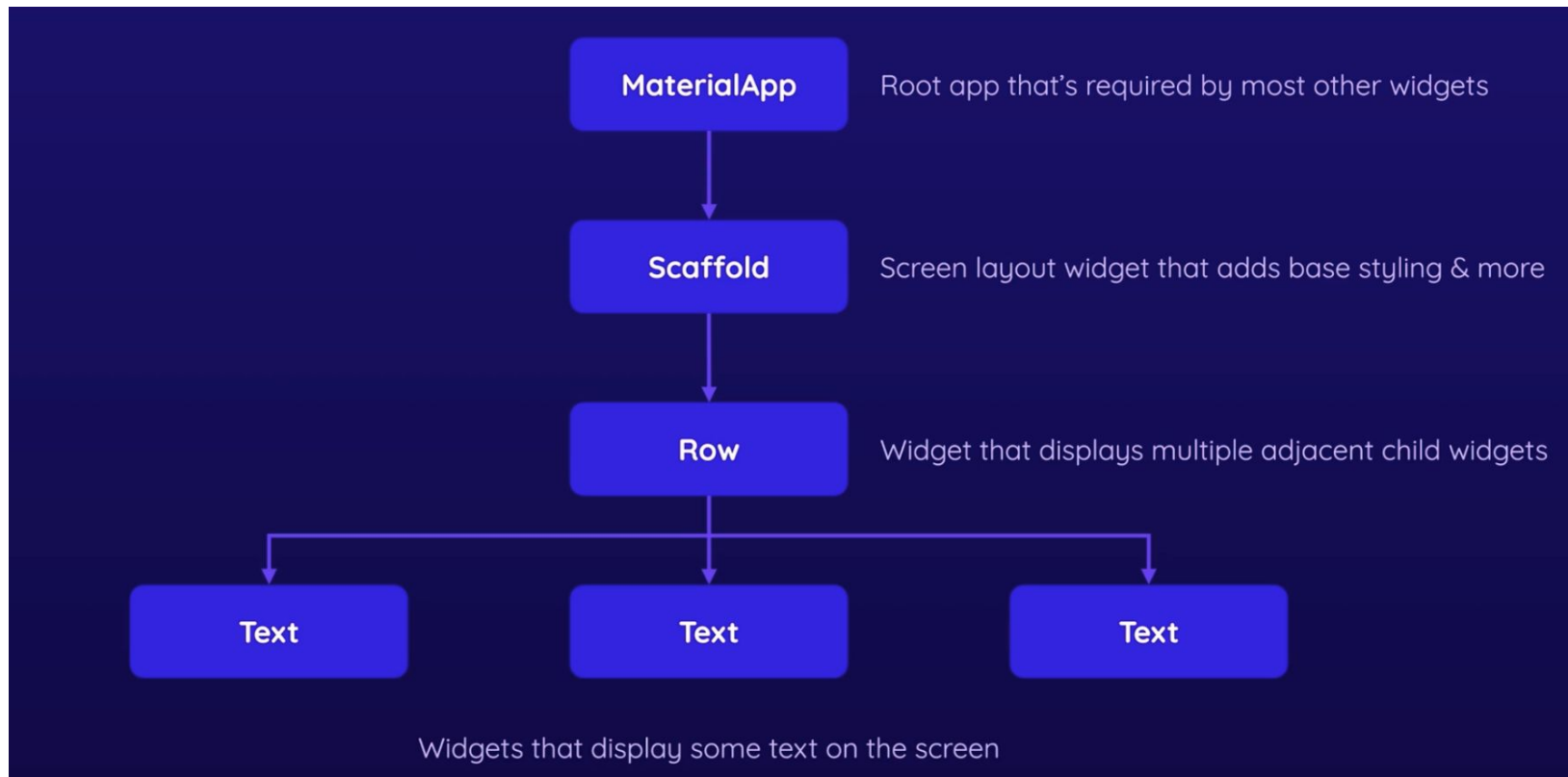
runApp() dice a Flutter cosa mostrare sullo schermo

Widget tree



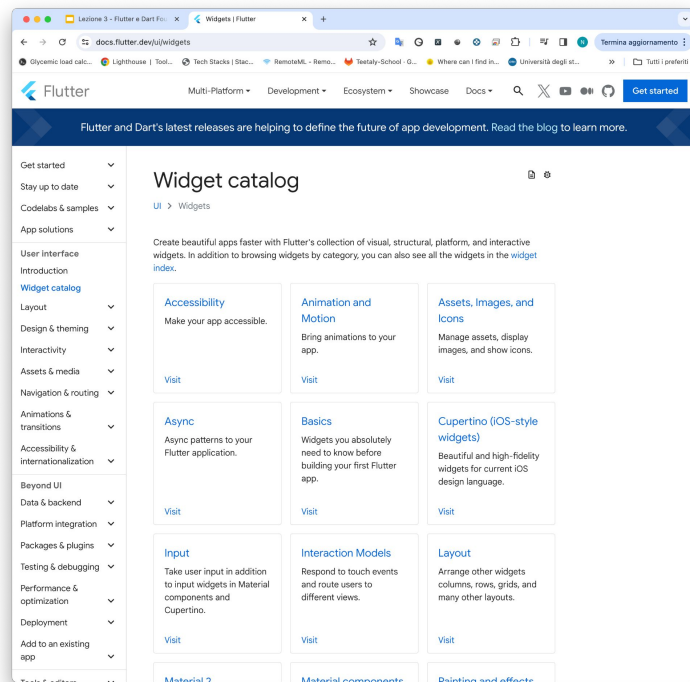
- Cosa fa **Center**? Cosa fa **Text**?
- I Widget possono essere presi da un catalogo oppure scritti in maniera custom

Widget tree



Widgets Catalog

<https://docs.flutter.dev/ui/widgets>

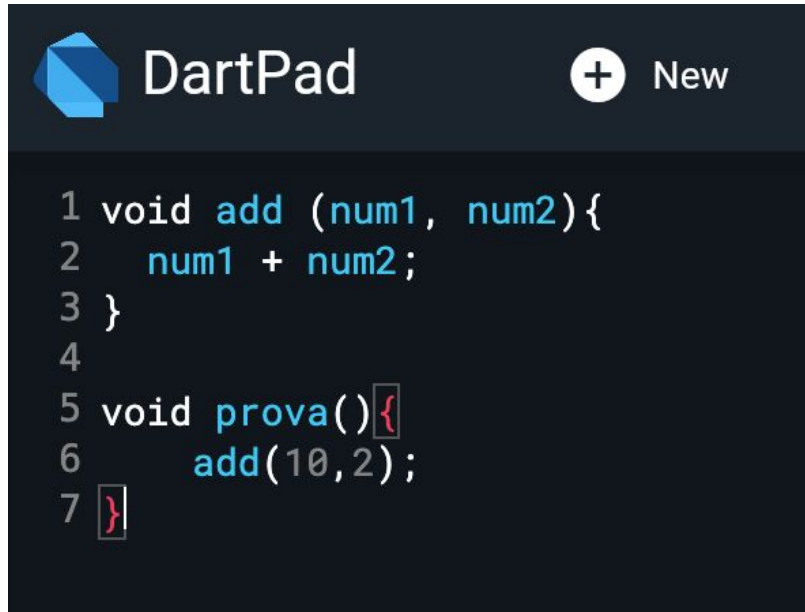


MaterialApp ()

```
import 'package:flutter/material.dart';  
Run | Debug | Profile  
void main() {  
  runApp(MaterialApp());  
}
```



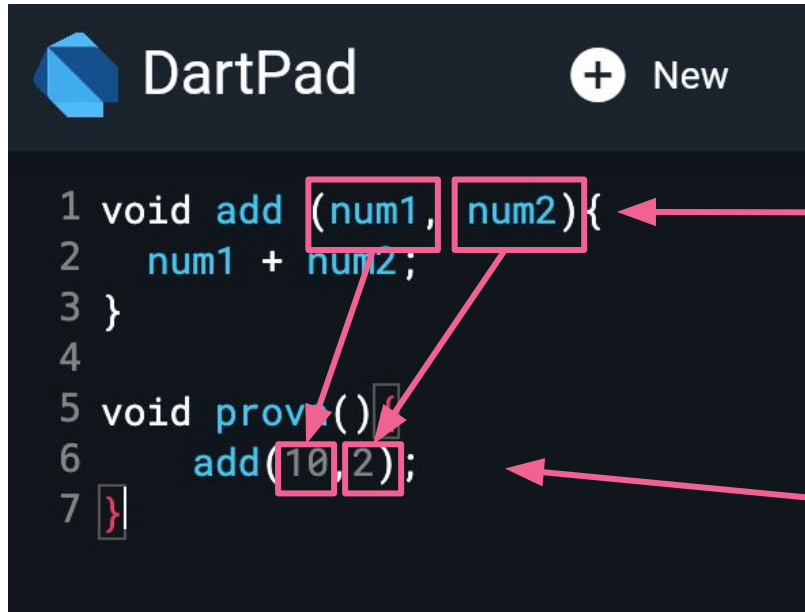
Uso di una funzione



The image shows a screenshot of the DartPad web editor interface. At the top left is the DartPad logo, a blue hexagon with a white 'D'. To its right is the text 'DartPad' in white. Further right is a circular button with a white plus sign and the word 'New' in white. Below the header is a dark blue code editor area containing the following Dart code:

```
1 void add (num1, num2){  
2   num1 + num2;  
3 }  
4  
5 void prova(){  
6   add(10,2);  
7 }
```

Uso di una funzione



```
1 void add (num1, num2){  
2   num1 + num2;  
3 }  
4  
5 void prova()  
6   add(10, 2);  
7 }
```

The image shows a screenshot of the DartPad web-based code editor. The interface has a dark theme. At the top left is the Dart logo and the text 'DartPad'. At the top right is a button with a plus sign and the text 'New'. The code is written in a monospaced font. Line 1: 'void add (num1, num2){'. Line 2: ' num1 + num2;'. Line 3: '}'. Line 4: an empty line. Line 5: 'void prova()'. Line 6: ' add(10, 2);'. Line 7: '}'. There are red annotations: a red box around the function signature '(num1, num2)' on line 1, with a red arrow pointing to it from the text 'Definizione di una funzione'. Another red box around the arguments '10, 2' on line 6, with a red arrow pointing to it from the text 'Esecuzione (o chiamata) della funzione con gli argomenti richiesti (valori di input)'. There is also a red box around the opening parenthesis of the function call 'add(' on line 6, with a red arrow pointing to it from the same text.

Definizione di una funzione

Esecuzione (o chiamata) della
funzione con gli argomenti
richiesti (valori di input)

MaterialApp ()

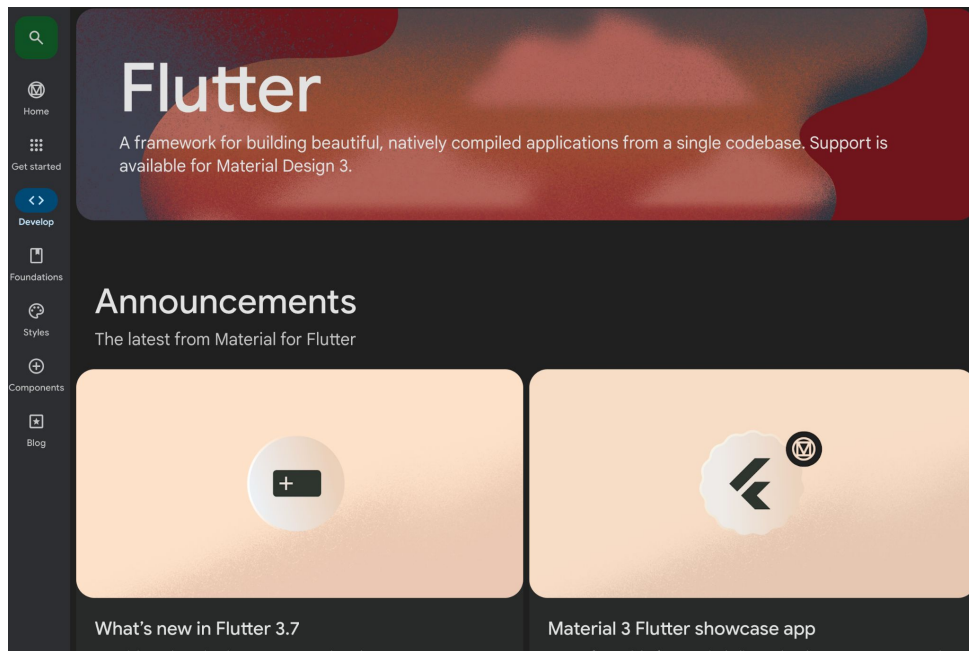
```
import 'package:flutter/material.dart';  
Run | Debug | Profile  
void main() {  
  runApp(MaterialApp());  
}
```



material.io

<https://m3.material.io/develop/flutter>

<https://m3.material.io/>



Funzioni e parametri

```
import 'package:flutter/material.dart';  
Run | Debug | Profile  
void main() {  
  runApp(MaterialApp());  
}
```

CMD+CLICK (mac)

CMD+CLICK (win)

Funzioni e parametri

```
///   element for the element hierarchy.
/// * [WidgetsBinding.handleBeginFrame], which pumps the widget pipe
///   ensure the widget, element, and render trees are all built.
void runApp(Widget app) {
  final WidgetsBinding binding = WidgetsFlutterBinding.ensureInitiali
  assert(binding.debugCheckZone('runApp'));
  binding
    ..scheduleAttachRootWidget(binding.wrapWithDefaultView(app))
    ..scheduleWarmUpFrame();
}

String _debugDumpAppString() {
  const String mode = kDebugMode ? 'DEBUG MODE' : kReleaseMode ? 'REL
```

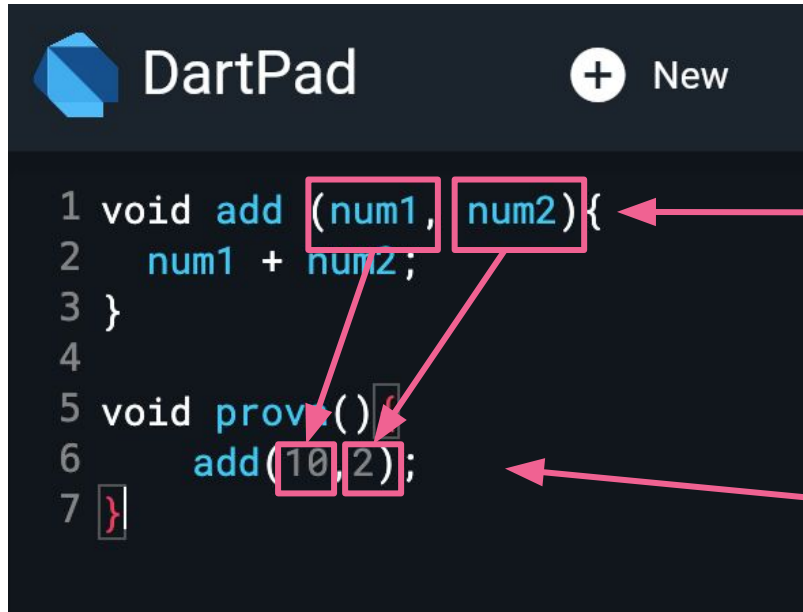
L'input richiesto da
runApp è tra
parentesi tonde

Funzioni e parametri

```
///  
/// This class creates an instance of [WidgetsApp].  
const MaterialApp({  
  super.key,  
  this.navigatorKey,  
  this.scaffoldMessengerKey,  
  this.home,  
  Map<String, WidgetBuilder> this.routes = const <String, WidgetBuilder>{},  
  this.initialRoute,  
  this.onGenerateRoute,  
  this.onGenerateInitialRoutes,  
  this.onUnknownRoute,  
  this.onNavigationNotification,  
  List<NavigatorObserver> this.navigatorObservers = const <NavigatorObserver>[],  
  this.builder,  
  this.title = '',  
  this.onGenerateTitle,  
  this.color,  
  this.theme,  
  this.darkTheme,  
  this.highContrastTheme,  
  this.highContrastDarkTheme,  
  this.themeMode = ThemeMode.system,  
  this.themeAnimationDuration = kThemeAnimationDuration,  
  this.themeAnimationCurve = Curves.linear,  
})
```

L'input richiesto da MaterialApp è tra parentesi tonde e graffe ({})

Uso di una funzione



```
1 void add (num1, num2){  
2   num1 + num2;  
3 }  
4  
5 void prova()  
6   add(10, 2);  
7 }
```

The image shows a screenshot of the DartPad web interface. The code editor has a dark background. The Dart logo is in the top left, and a '+ New' button is in the top right. The code is as follows:

```
1 void add (num1, num2){  
2   num1 + num2;  
3 }  
4  
5 void prova()  
6   add(10, 2);  
7 }
```

Red boxes highlight the function definition `(num1, num2)` on line 1 and the function call `add(10, 2)` on line 6. Red arrows point from these boxes to explanatory text on the right.

Definizione di una funzione

Esecuzione (o chiamata) della
funzione con gli argomenti
richiesti (valori di input)

Funzioni e parametri

```
1 void add ({num1, num2}){  
2     num1 + num2;  
3 }  
4  
5 void prova(){  
6     add(10, 2);  
7 }
```

Funzioni e parametri

```
1 void add ({num1, num2}) {  
2   num1 + num2;  
3 }  
4  
5 void prova() {  
6   add(num2: 10, num1: 2);  
7 }
```

The diagram illustrates argument passing by name in the C programming language. It shows two functions: `add` and `prova`. The `add` function has two parameters, `num1` and `num2`, which are highlighted with red boxes. The `prova` function calls `add` with two arguments: `num2: 10` and `num1: 2`, which are also highlighted with red boxes. Red arrows indicate the mapping from the arguments in the function call to the parameters in the function definition: one arrow points from `num2: 10` to `num1`, and another points from `num1: 2` to `num2`. This demonstrates how the arguments are substituted directly into the function body.

MaterialApp ()

Al nostro codice
manca ancora
qualcosa...

```
import 'package:flutter/material.dart';  
Run | Debug | Profile  
void main() {  
  runApp(MaterialApp());  
}
```


MaterialApp ()

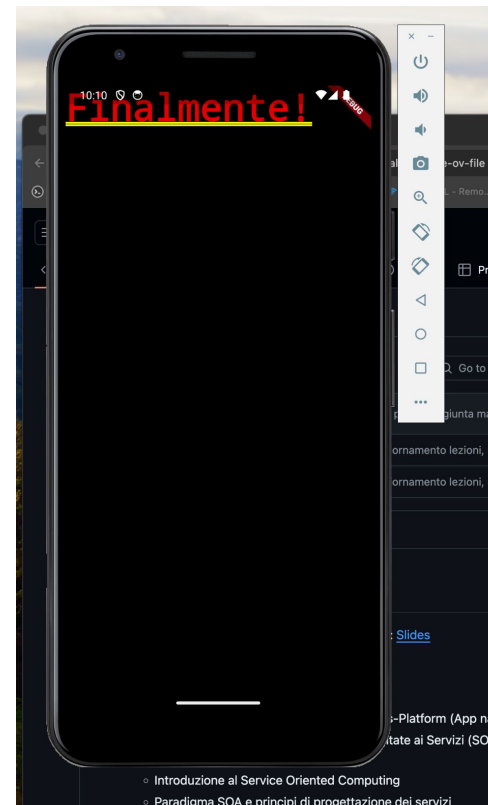
Scegliamo un widget...

```
main.dart 1 •
lib > main.dart > main
1 import 'package:flutter/material.dart';
  Run | Debug | Profile
2 void main() {
3   runApp(MaterialApp(home: ));
4 }
5
6
7
8
9
```

- AboutDialog
- AboutListTile
- AbsorbPointer
- ActionChip
- ActionIconTheme
- ActionListener
- Actions
- AdaptiveTextSelectionToolbar
- AlertDialog
- Align
- AlignTransition
- AndroidView

e finalmente...

```
main.dart ●  
lib > main.dart > ...  
1 import 'package:flutter/material.dart';  
   Run | Debug | Profile  
2 void main() {  
3   runApp(MaterialApp(home: Text('Finalmente!')));  
4 }  
5  
6
```



Esercizio

Integriamo il codice appena visto spostando però il testo al centro