How to create an Ubuntu Desktop Yaru application with Flutter

This is a **beginner** tutorial for those new to the Dart programming language, new to programming languages in general and new to the Yaru design.

Intro

The preinstalled applications on Ubuntu are quite diverse in their programming language and tooling origins. Some examples are the Firefox internet browser and the Thunderbird e-mail client both being C++ and JavaScript applications, the Libre-Office suite being written in C++, XML, and Java and gnome-files (A.K.A. nautilus) which is written in C with gtk.

Another toolkit is Flutter. Flutter is a multi platform toolkit written in C++ and dart. The GUI of the new Ubuntu Desktop installer is made Flutter as well as the next iteration of Ubuntu Software plus there are hundreds of iOS, Android, Windows, Web and MacOS applications created with Flutter.

Over the past years we've designed and developed several dart libraries which make it easy to create Ubuntu Desktop applications with Flutter. This tutorial will make all of this less mystical for people not familiar with neither Flutter nor our dart libraries.

What you will learn

- How to setup your Flutter development environment on Ubuntu
- Learn VsCode basics
- Get to know dart libraries to create an aesthetic and visually consistent desktop application
- Get to know dart libraries to interact with existing Free-Desktop and hardware related APIs on Ubuntu
- Create a starting point for either a multi-page, single-page or wizard-like desktop applications on Ubuntu

Skill requirements

It should be an advantage if you have created an application before, preferable with an object oriented language and if you are not scared to copy and paste commands into your terminal. But since this is a step-by-stand, hands-on tutorial everyone with a bit of technical interest should do fine.

Setup

Install Flutter

If you want to create Android or Web applications with Flutter from your Ubuntu machine, all you need should be the flutter snap (`snap install flutter --classic`). However, this tutorial is about creating apps for the Ubuntu *Desktop*. Some of our dart libraries make use of native libraries which may not behave perfectly with the way the flutter snap interacts with your system.

The following lines will install the dependencies for Flutter Linux apps, create a directory in your home dir, clone the flutter git repository and export the `flutter` and `dart` commands to your path so you can run it from any user shell.

So please open up your terminal on Ubuntu by either pressing the key-combination CTRL + ALT + T or by searching for "Terminal" in your Ubuntu search. Now

either copy & paste the following lines successively into your terminal and press enter after:

```
sudo apt install git curl cmake meson make clang libgtk-3-dev pkg-config
mkdir -p ~/development
cd ~/development
git clone https://github.com/flutter/flutter.git -b stable
echo 'export PATH="$PATH:$HOME/development/flutter/bin"' >> ~/.bashrc
source ~/.bashrc
```

OR use this one-liner to copy and paste everything into your terminal, ⚠️ this does not stop until it is done:

```
sudo apt -y install git curl cmake meson make clang libgtk-3-dev pkg-config
&& mkdir -p ~/development && cd ~/development && git clone
https://github.com/flutter/flutter.git -b stable && echo 'export
PATH="$PATH:$HOME/development/flutter/bin"' >> ~/.bashrc && source
~/.bashrc
```

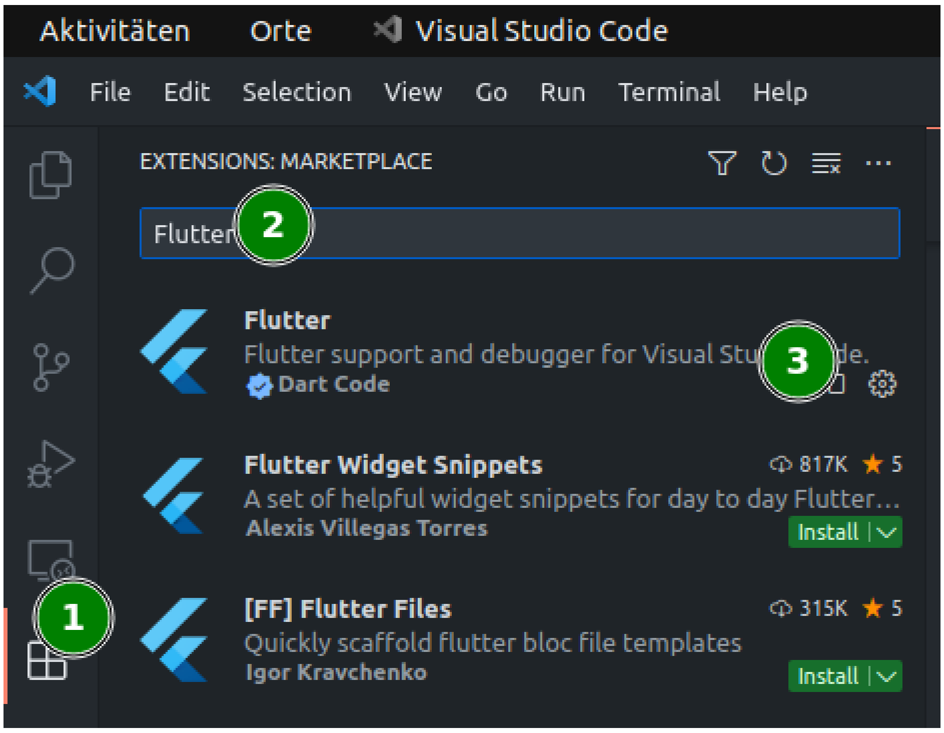
Install VsCode

Run the following command to install VsCode on your Ubuntu machine (or install it from Ubuntu Software):

```
sudo snap install code --classic
```

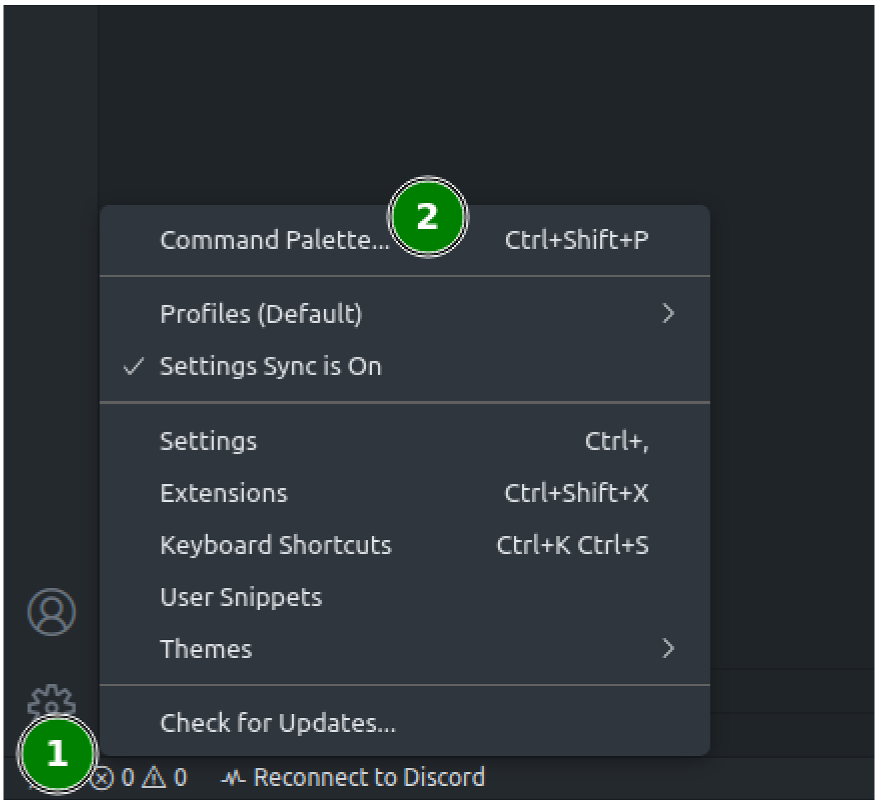
Setup VsCode

Open VsCode, click on the extension icon in the left sidebar (1), type "Flutter" and click "Install" on the first entry (3), this should be the Flutter extension by Dart Code.

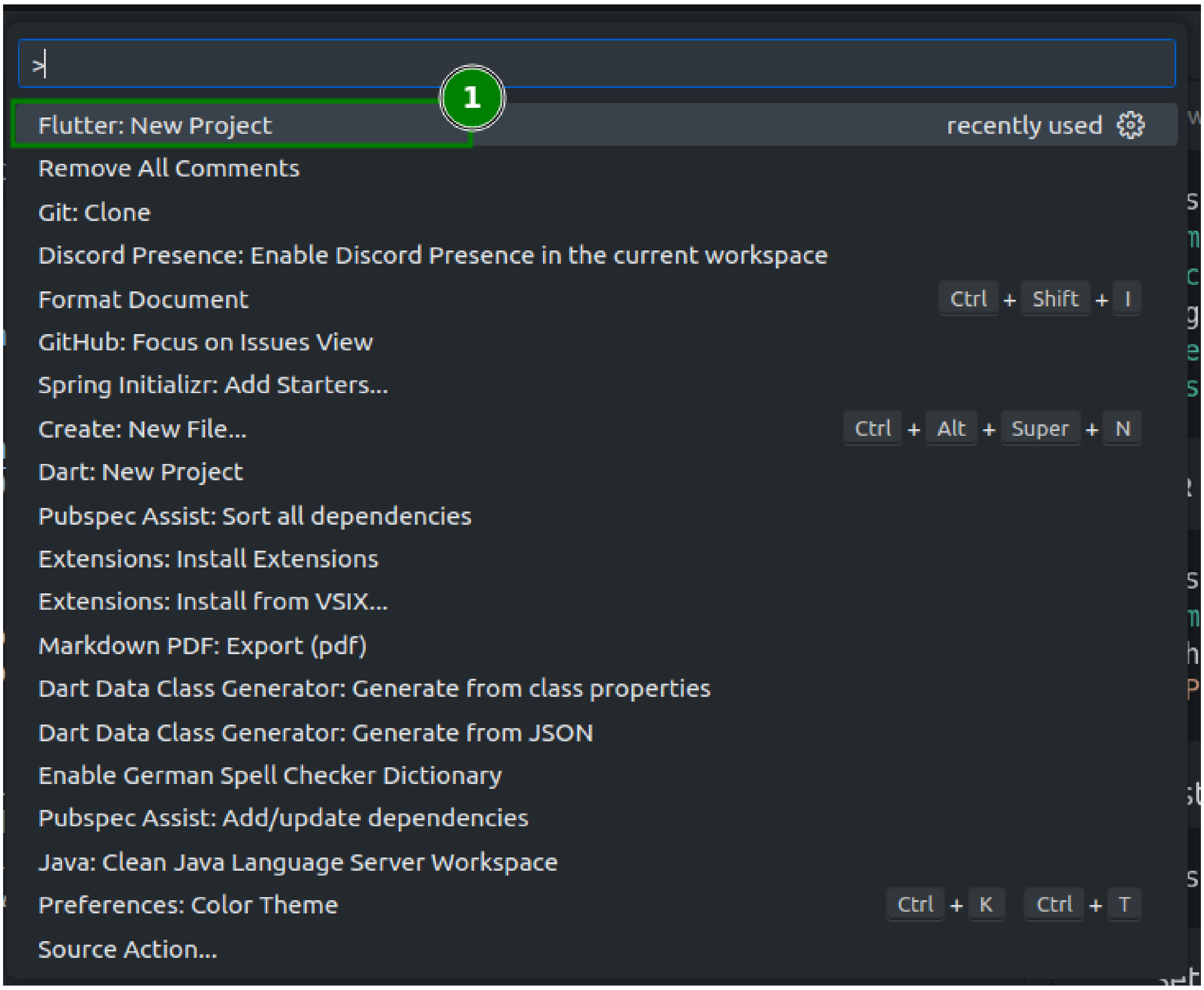


Let's get started: flutter create

VsCode offers a command palette which you can open with either CTRL+SHIFT+P or by clicking on the :gear: icon

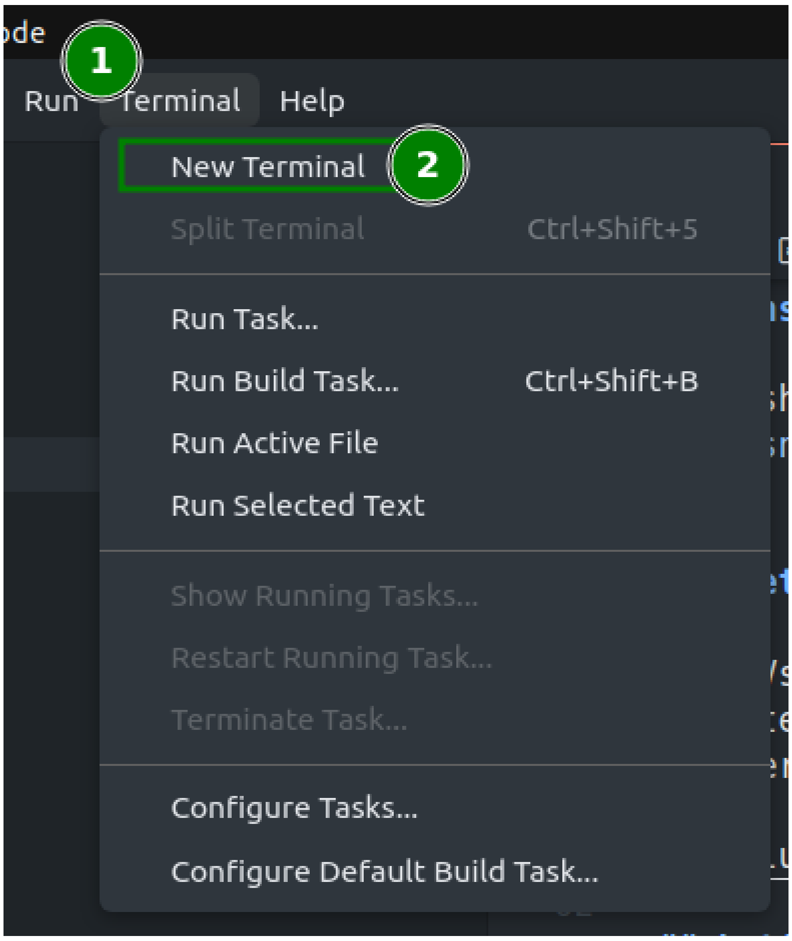


We *could* now type "Flutter new project"


>
1
Flutter: New Project
recently used
Remove All Comments
Git: Clone
Discord Presence: Enable Discord Presence in the current workspace
Format Document
Ctrl + Shift + I
GitHub: Focus on Issues View
Spring Initializr: Add Starters...
Create: New File...
Ctrl + Alt + Super + N
Dart: New Project
Pubspec Assist: Sort all dependencies
Extensions: Install Extensions
Extensions: Install from VSIX...
Markdown PDF: Export (pdf)
Dart Data Class Generator: Generate from class properties
Dart Data Class Generator: Generate from JSON
Enable German Spell Checker Dictionary
Pubspec Assist: Add/update dependencies
Java: Clean Java Language Server Workspace
Preferences: Color Theme
Ctrl + K Ctrl + T
Source Action...

However, since we want to make amount of auto created files as small as possible to make the management as easy as possible, we want to specify the platforms for our new project.

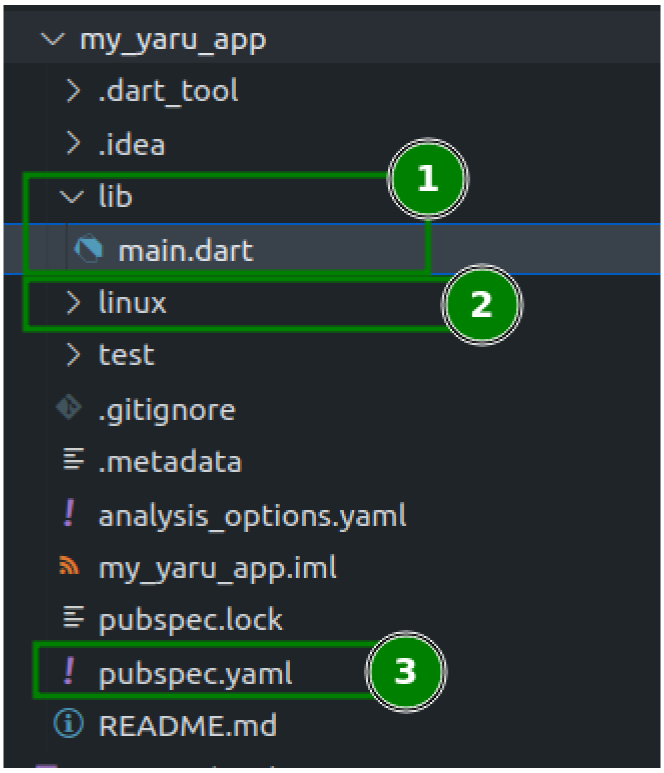
Open the integrated terminal in vscode if it is not already opened



And run the following command to create a new Flutter project for Linux only (you can add more platforms at any point if you want) and specify the name of your organization/company and your appname:

```
flutter create --platforms=linux --org com.test my_yaru_app
```

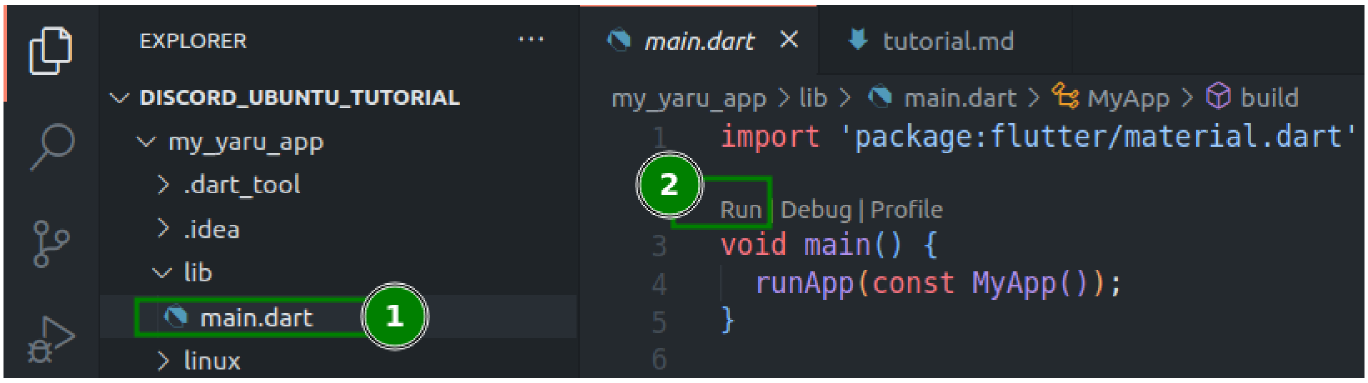
Flutter created a small template app for us. Let's take a look at the three locations we need to visit first:



(1) Is the `lib` directory where all of our dart code lives. For now a single `main.dart` file should be enough. All platforms our app wants to be available for gets its own directory. In our case only the `Linux` directory (2). We will come this back later. To define metadata of our app and the dependencies we want to use we need the `pubspec.yaml` file (3).

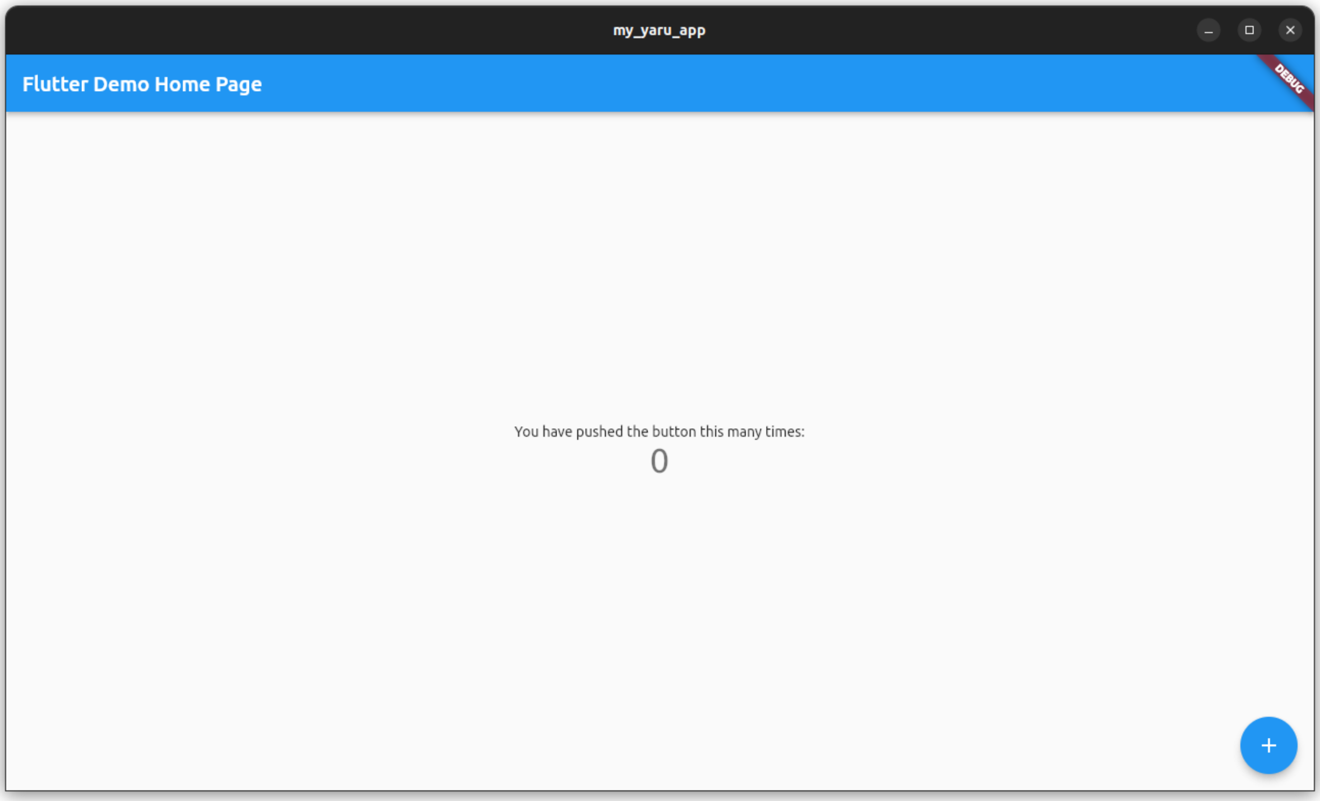
First run

Now click on `main.dart` (1) to open the file in your editor and click on the small `Run` label above the `void main()` declaration (2) to run the app for the first time



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

void main() {
  runApp(const MyApp());
}
```

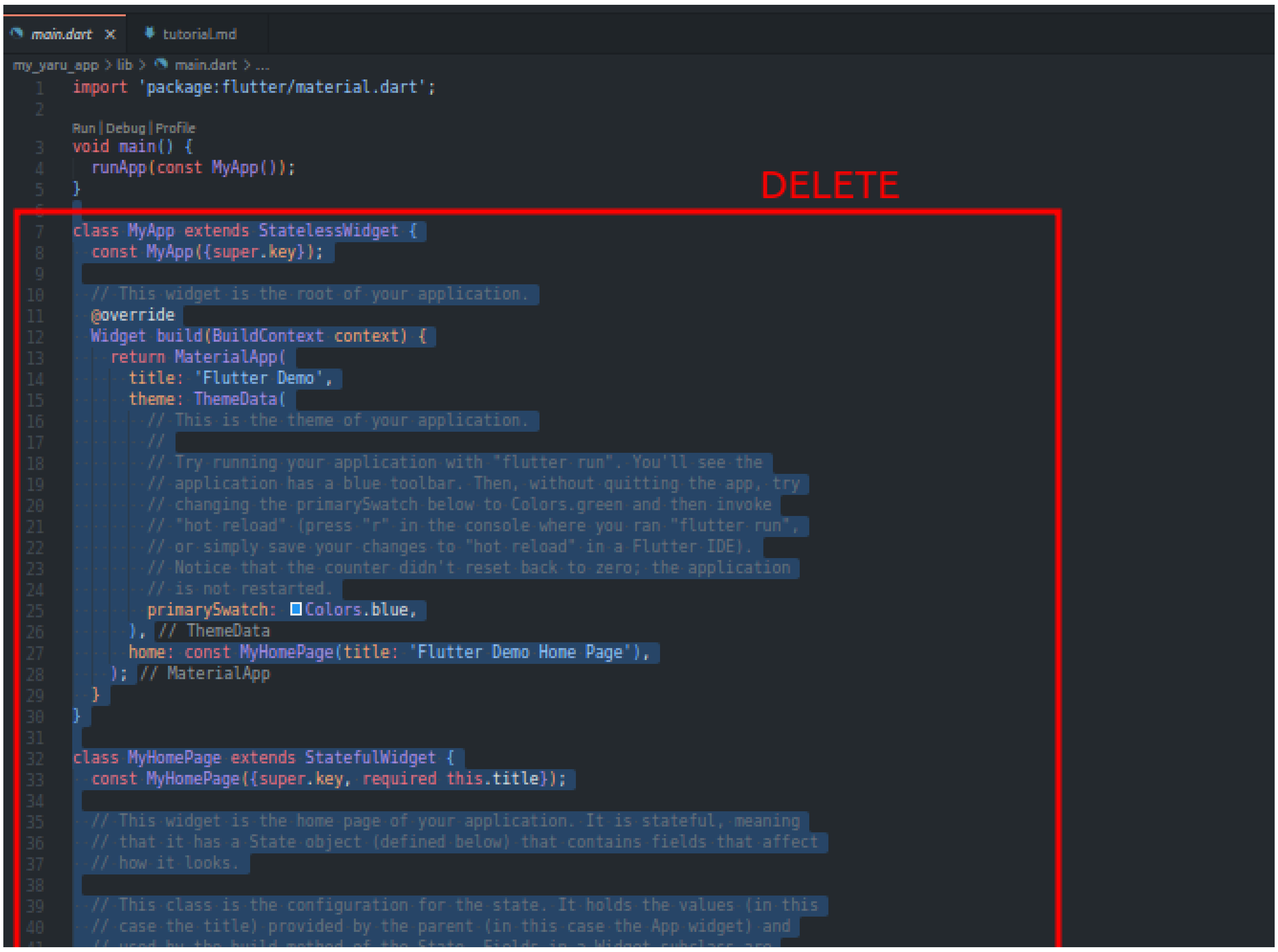
Caution, it is not pretty yet:



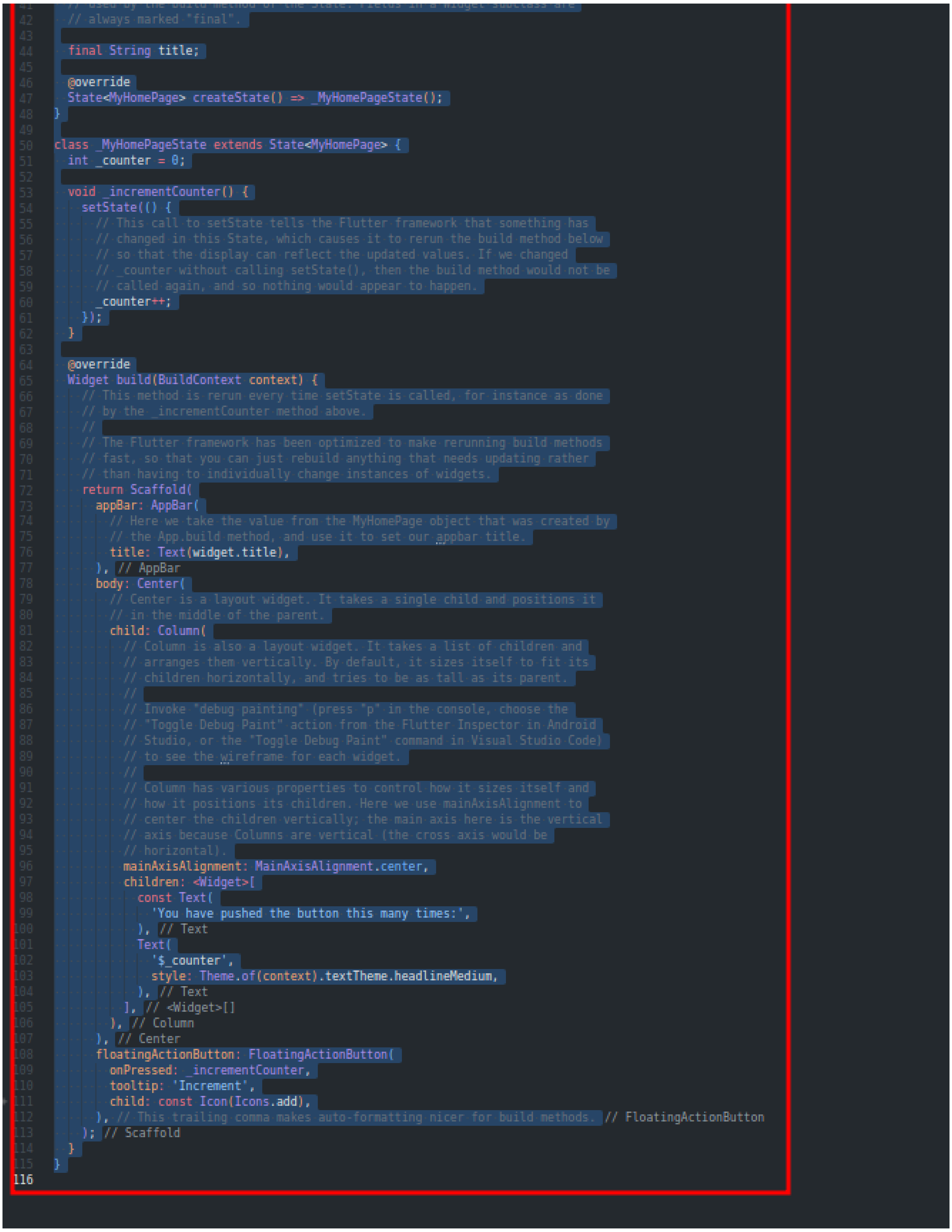
You have pushed the button this many times:
0

Clean up

The Flutter template app is quite verbose explaining what it contains but we don't need most of the things in here for now. Delete everything in your main.dart file below line 5

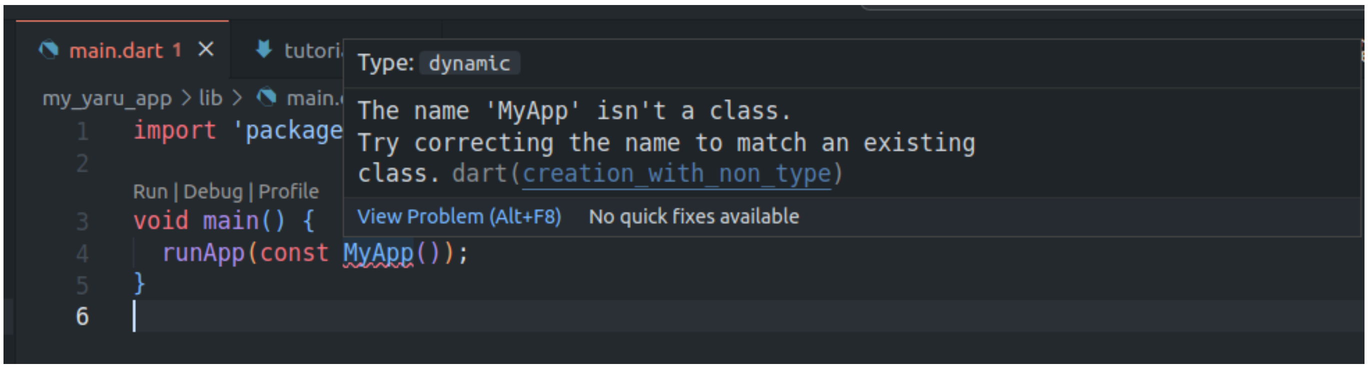


```
1   import 'package:flutter/material.dart';
2
3   void main() {
4     runApp(const MyApp());
5   }
6
7   class MyApp extends StatelessWidget {
8     const MyApp({super.key});
9
10    // This widget is the root of your application.
11    @override
12    Widget build(BuildContext context) {
13      return MaterialApp(
14        title: 'Flutter Demo',
15        theme: ThemeData(
16          // This is the theme of your application.
17          //
18          // Try running your application with "flutter run". You'll see the
19          // application has a blue toolbar. Then, without quitting the app, try
20          // changing the primarySwatch below to Colors.green and then invoke
21          // "hot reload" (press "r" in the console where you ran "flutter run",
22          // or simply save your changes to "hot reload" in a Flutter IDE).
23          // Notice that the counter didn't reset back to zero; the application
24          // is not restarted.
25          primarySwatch: Colors.blue,
26        ), // ThemeData
27        home: const MyHomePage(title: 'Flutter Demo Home Page'),
28      ); // MaterialApp
29    }
30  }
31
32  class MyHomePage extends StatefulWidget {
33    const MyHomePage({super.key, required this.title});
34
35    // This widget is the home page of your application. It is stateful, meaning
36    // that it has a State object (defined below) that contains fields that affect
37    // how it looks.
38
39    // This class is the configuration for the state. It holds the values (in this
40    // case the title) provided by the parent (in this case the App widget) and
```



```dart
  // used by the build method of the State. Fields in a Widget subclass are
  // always marked "final".

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      // This call to setState tells the Flutter framework that something has
      // changed in this State, which causes it to rerun the build method below
      // so that the display can reflect the updated values. If we changed
      // _counter without calling setState(), then the build method would not be
      // called again, and so nothing would appear to happen.
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    // This method is rerun every time setState is called, for instance as done
    // by the _incrementCounter method above.
    //
    // The Flutter framework has been optimized to make rerunning build methods
    // fast, so that you can just rebuild anything that needs updating rather
    // than having to individually change instances of widgets.
    return Scaffold(
      appBar: AppBar(
        // Here we take the value from the MyHomePage object that was created by
        // the App.build method, and use it to set our appbar title.
        title: Text(widget.title),
      ), // AppBar
      body: Center(
        // Center is a layout widget. It takes a single child and positions it
        // in the middle of the parent.
        child: Column(
          // Column is also a layout widget. It takes a list of children and
          // arranges them vertically. By default, it sizes itself to fit its
          // children horizontally, and tries to be as tall as its parent.
          //
          // Invoke "debug painting" (press "p" in the console, choose the
          // "Toggle Debug Paint" action from the Flutter Inspector in Android
          // Studio, or the "Toggle Debug Paint" command in Visual Studio Code)
          // to see the wireframe for each widget.
          //
          // Column has various properties to control how it sizes itself and
          // how it positions its children. Here we use mainAxisAlignment to
          // center the children vertically; the main axis here is the vertical
          // axis because Columns are vertical (the cross axis would be
          // horizontal).
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'You have pushed the button this many times:',
            ), // Text
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headlineMedium,
            ), // Text
          ], // <Widget>[]
        ), // Column
      ), // Center
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build methods. // FloatingActionButton
    ); // Scaffold
  }
}
```

Dart will now complain that the class MyApp does not exist any longer. Because we've just deleted it on purpose.



```
import 'package
void main() {
  runApp(const MyApp());
}
```

Type: dynamic

The name 'MyApp' isn't a class.
Try correcting the name to match an existing class. dart(creation_with_non_type)

View Problem (Alt+F8) No quick fixes available

First snipped: stle

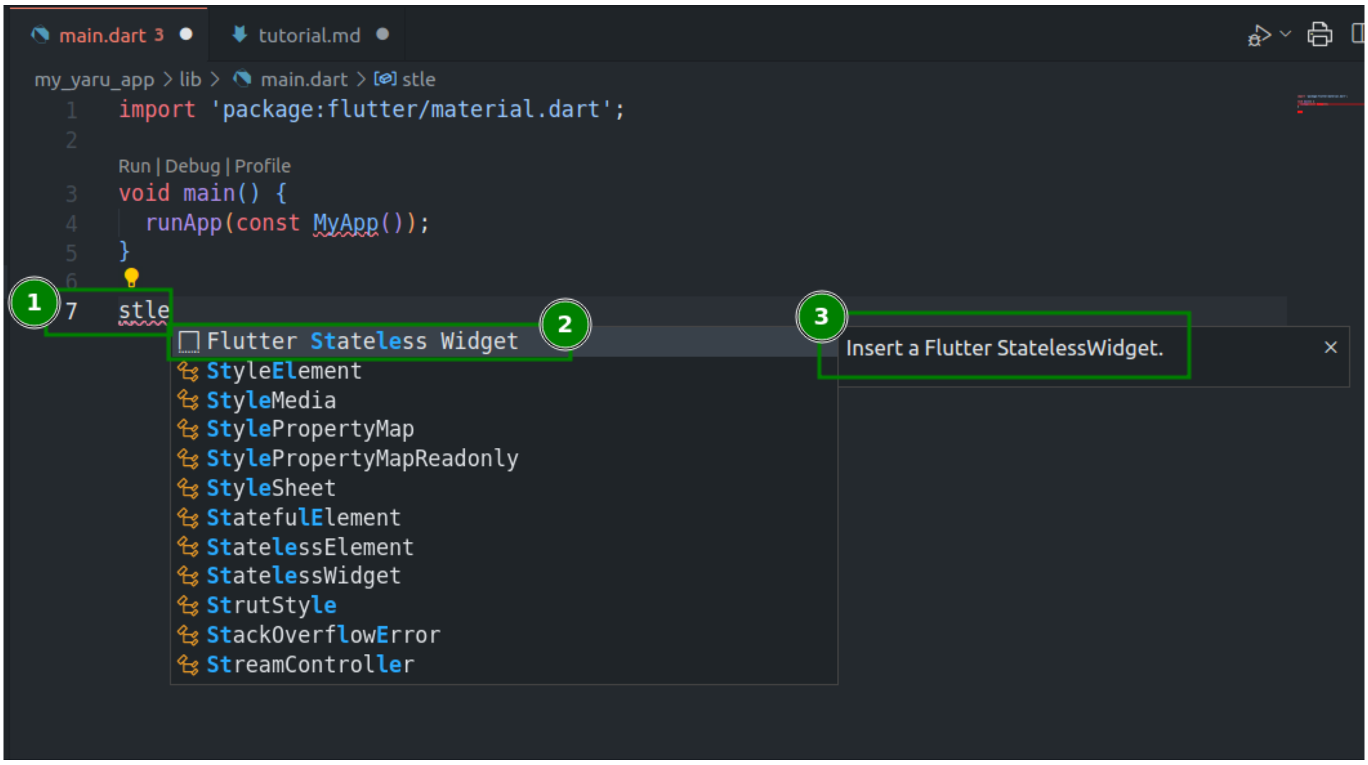
The Flutter VsCode extensions is extremely helpful for almost any task and saves us a lot of lines to write. There are quick commands, snippets, auto-complete and auto fix features available which we will use in this tutorial. The first help we will use is the snippet `stle` which is short for `StatelessWidget`.

Move below line 5 and write

```
stle
```

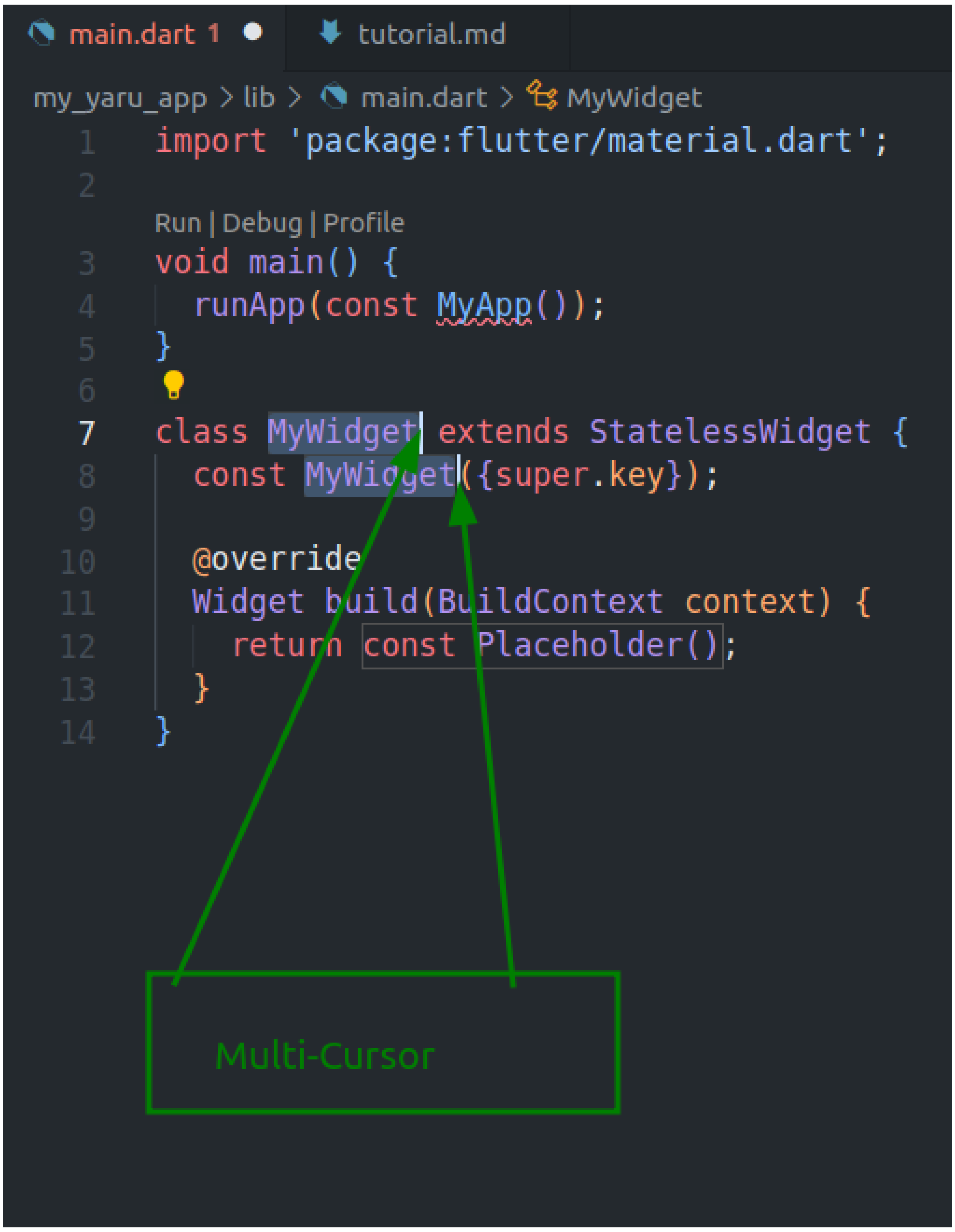
Now a popup should ... pop-up. (if not press CTRL+ENTER, if this does not help either, there is something wrong with your setup of vscode, flutter and the Flutter VsCode extension).

(1) is your text and the cursor, (2) is the detected snippet `Flutter Stateless Widget` and (3) is a little explanation what will happen if you press ENTER now, which you please do now:



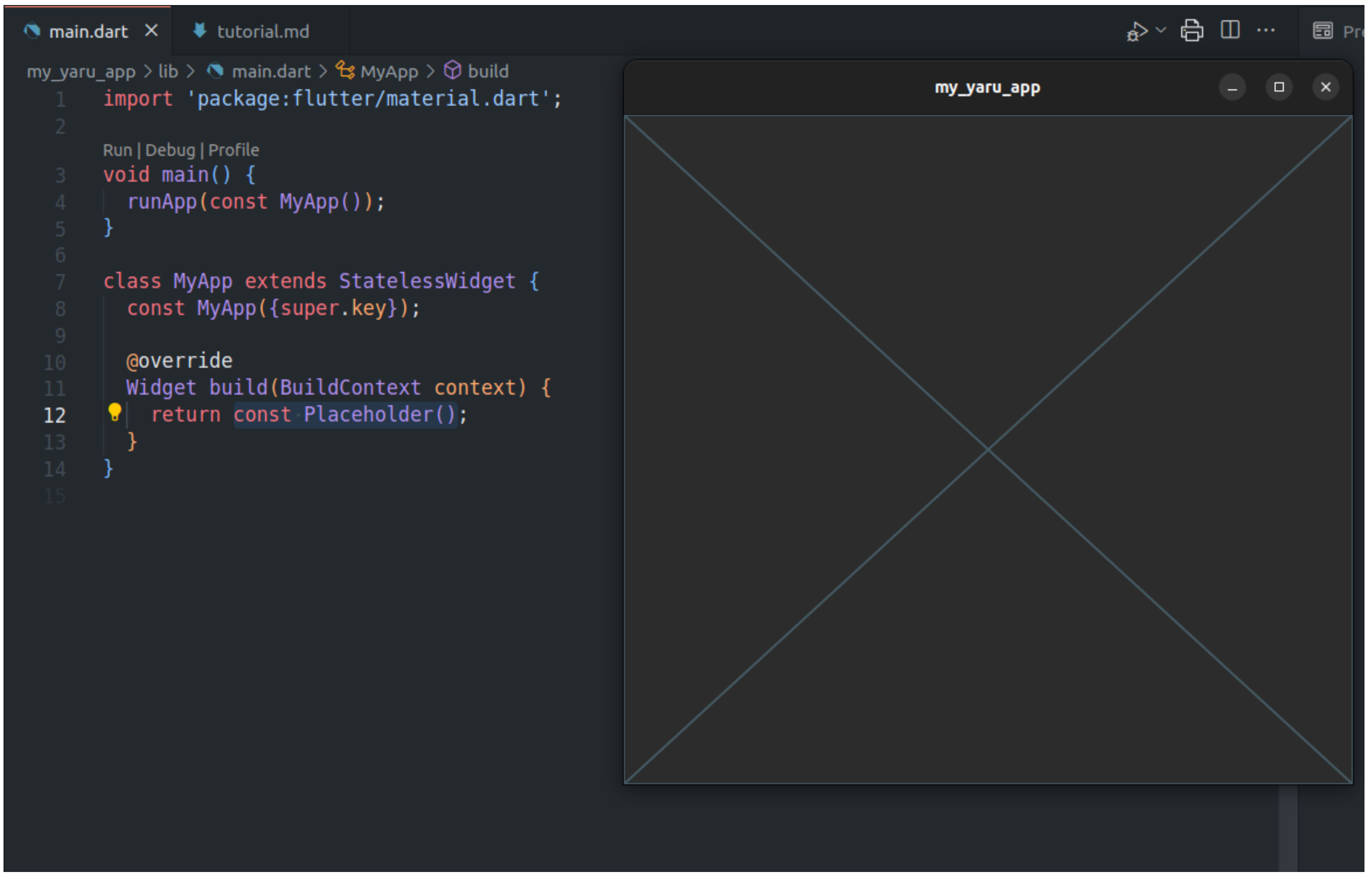
```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

stle
```

Something happened! Now please stay calm and look what you got. The created snippet left a multi-cursor in the places which change if you change the name of your `StatelessWidget`.



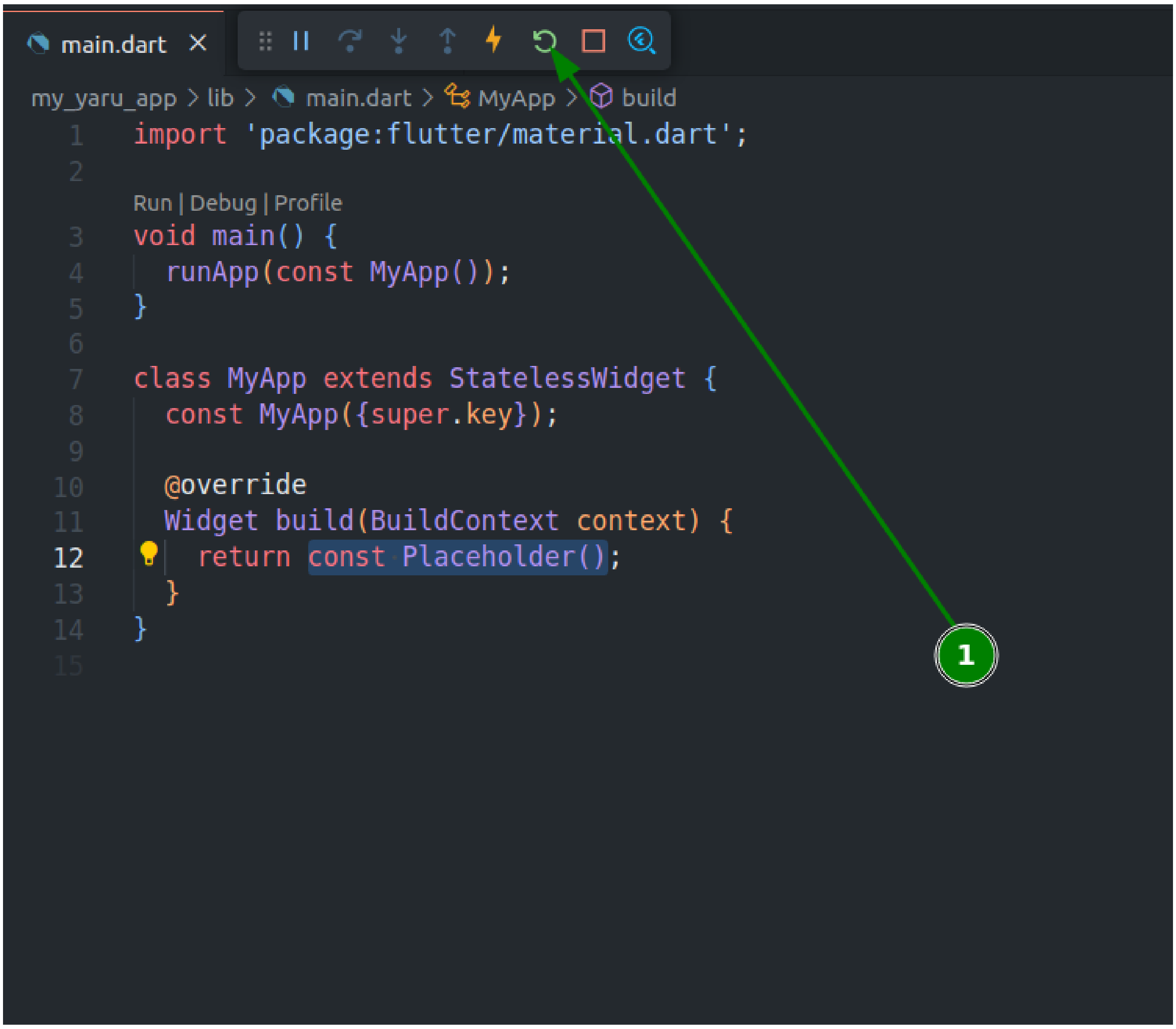
```dart
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyWidget extends StatelessWidget {
  const MyWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

Just start writing now! Write `MyApp` and the text will be written into both places at once. When you are done press the ESC key on your keyboard to stop the multi-cursor. Pressing CTRL+S will save your code and the changes will be hot-reloaded immediately into your app:



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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

Every time you save your code by either pressing CTRL+S or by the menu entry File->Save, Flutter will Hot-Reload your changes right into your dart process. This means that you do not need to re-run your app every time you change something in your code. However if you exchange bigger parts you might need to click on Restart



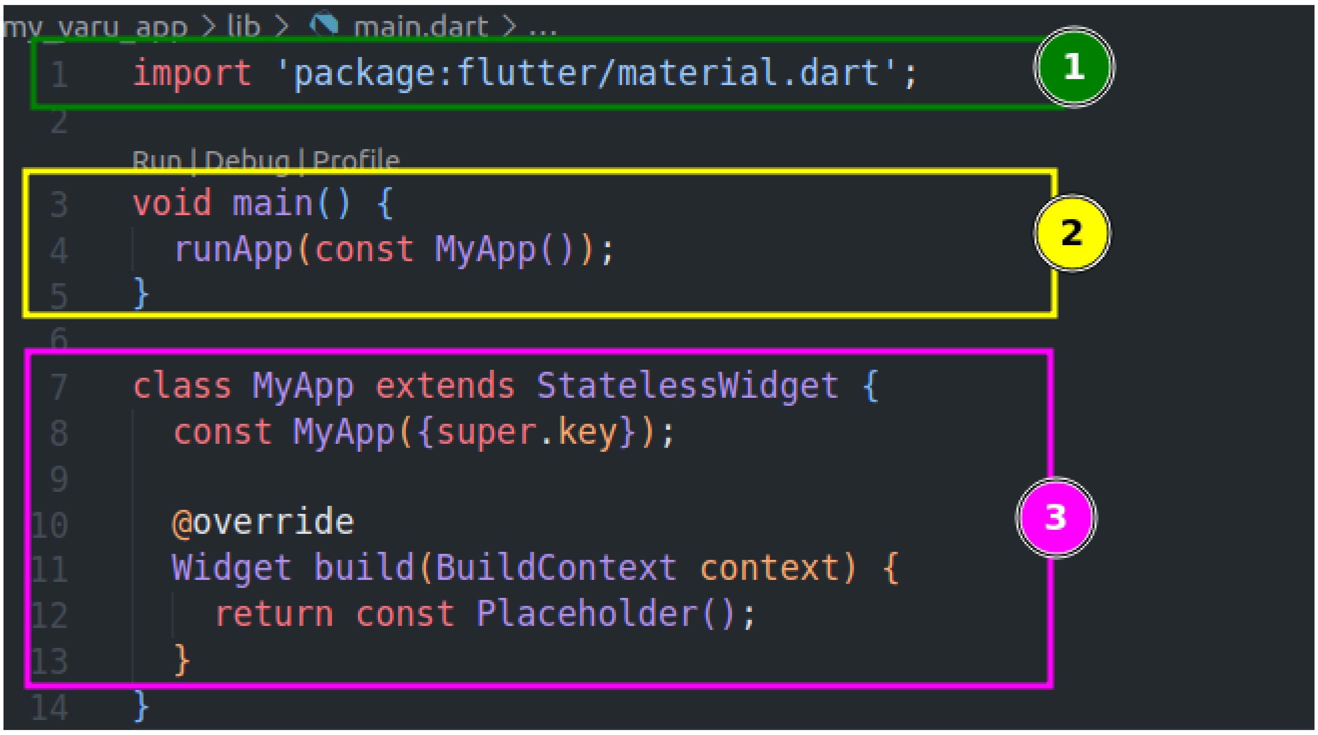
```dart
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

First recap

(1) Imports the package `material.dart`

(2) Is the main application with the `runApp` function call.

(3) Is your class `MyApp` which extends the class `StatelessWidget`. Extending this class forces your app to implement the `Widget build(BuildContext context)` method, which you do by returning the Widget `PlaceHolder`.



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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

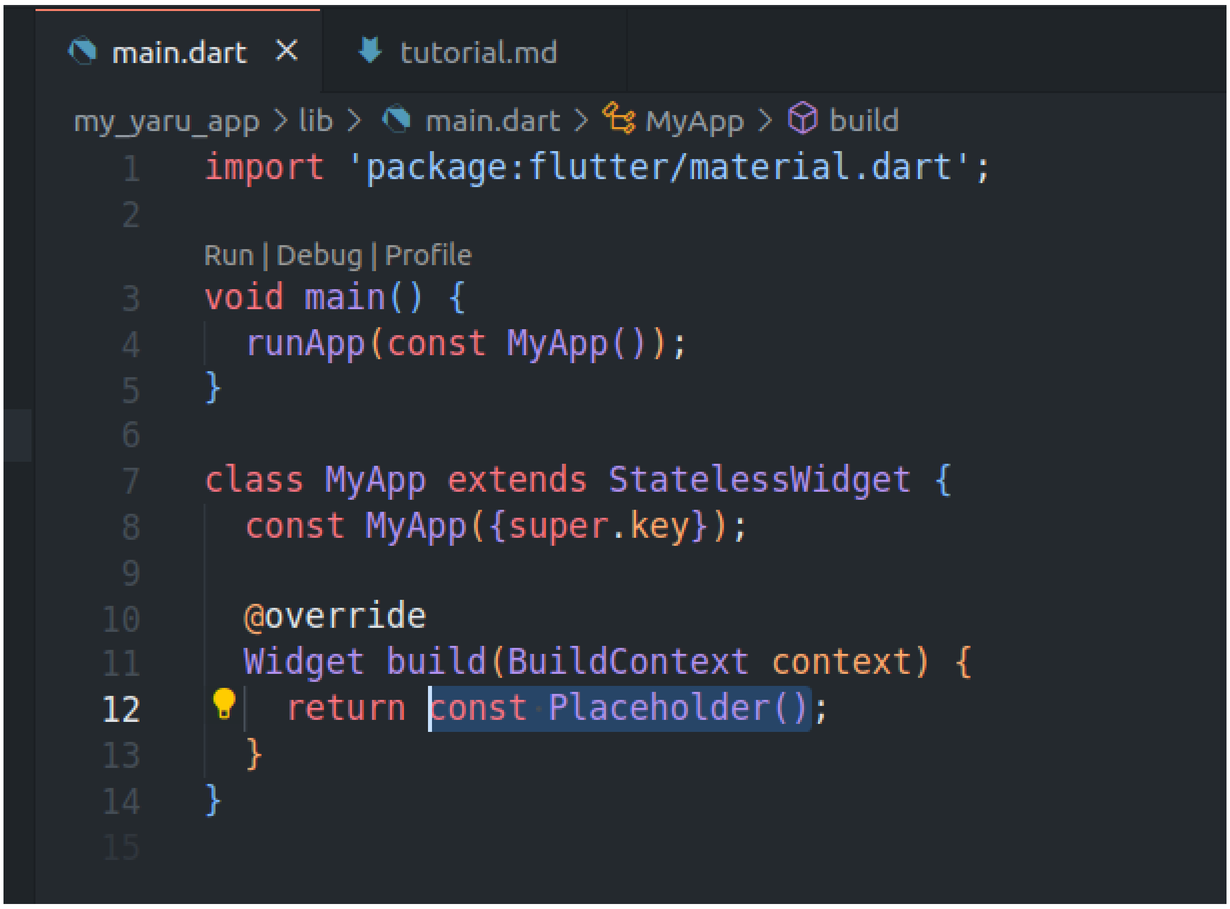
dart keywords used

- import
- void
- const
- class
- extends
- super
- return

Creating the app skeleton

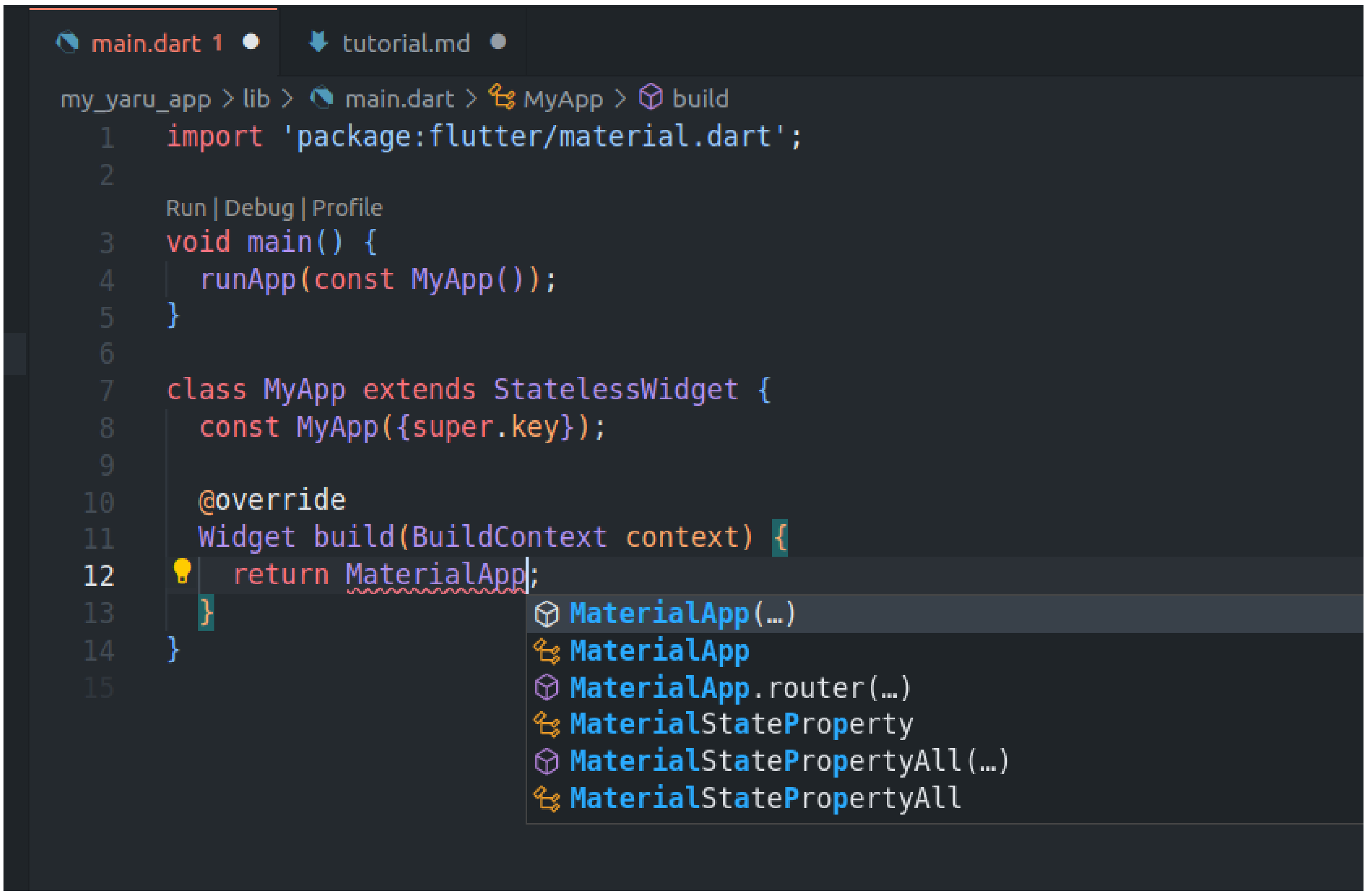
MaterialApp

Mark `const Placeholder`



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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

and write `MaterialApp` which opens a popup with a suggested class, press ENTER to replace `PlaceHolder` with `MaterialApp()`



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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp;
  }
}
```

```
MaterialApp(…)
MaterialApp
MaterialApp.router(…)
MaterialStateProperty
MaterialStatePropertyAll(…)
MaterialStatePropertyAll
```

Quick look into named parameters in dart

Don't code now, just read.

Functions in dart, as in any other modern programming language, can either have no or any kind and amount of parameters (also called arguments or input variables). *(In mathematics this is different. All functions must have at least one argument and a return value.)*

To make reading function calls easier dart has the optional feature of named parameters. Where a function, if defined with (a) named parameter(s), must be called by naming the parameter, followed by a `:` and the value that should be set.

Example definition without a named parameter:

```
int incrementByOne(int myParameter) {
    return myParameter + 1;
}
```

Calling the function:

```
incrementByOne(3);
```

Example definition with a named parameter:

```
int incrementByOne({required int myParameter}) {
    return myParameter + 1;
}
```

Calling the function:

```
incrementByOne(myParameter: 3);
```

To create an instance of a class one needs to call the constructor "function" (called method if part of a class).

Flutter widget classes almost always use named parameters, which is increasingly useful the more parameters a Widget has when you call its constructor method.

Example Widget definition:

```
class _MyNumberWidget extends StatelessWidget {
  // This is the constructor definition
  const _MyNumberWidget({required this.number});
  // This is your parameter of the type integer.
  final int number;
  @override
  Widget build(BuildContext context) {
    // using the parameter to be shown inside the UI
    return Text(number.toString());
  }
}
```

Somewhere else (where calling functions is allowed):

```
final Widget myNumberWidget = MyNumberWidget(number: 3)
```

New keywords learned

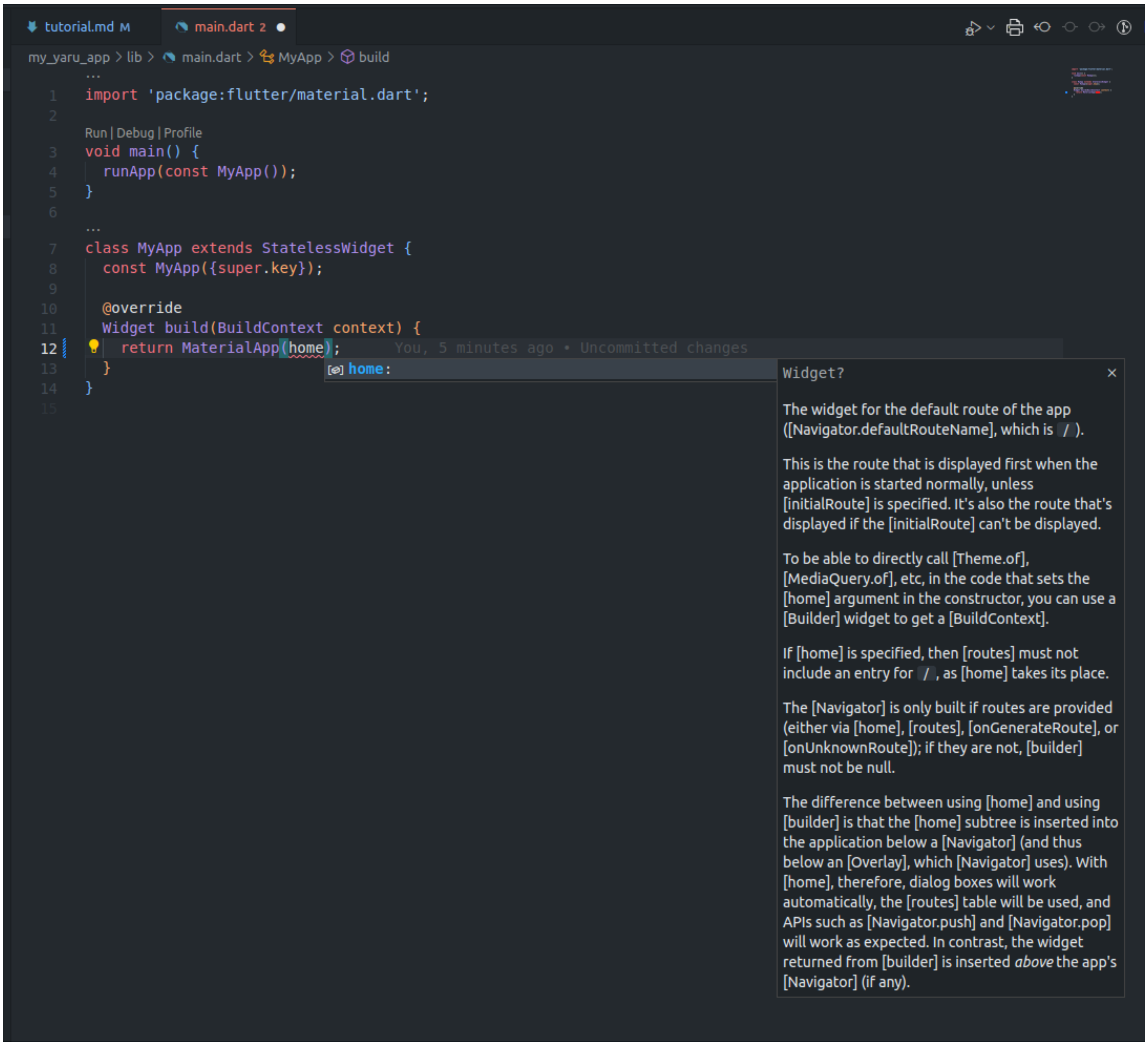
- final
- required

Back to coding: Scaffold

Move your cursor inside the brackets of the `MaterialApp()` constructor call and insert

```
home:
```

VsCode then suggests:



tutorial.md M main.dart 2

my_yaru_app > lib > main.dart > MyApp > build

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

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(home);      You, 5 minutes ago • Uncommitted changes
  }
}
```

home:

Widget?

The widget for the default route of the app ([Navigator.defaultRouteName], which is `/`).

This is the route that is displayed first when the application is started normally, unless [initialRoute] is specified. It's also the route that's displayed if the [initialRoute] can't be displayed.

To be able to directly call [Theme.of], [MediaQuery.of], etc, in the code that sets the [home] argument in the constructor, you can use a [Builder] widget to get a [BuildContext].

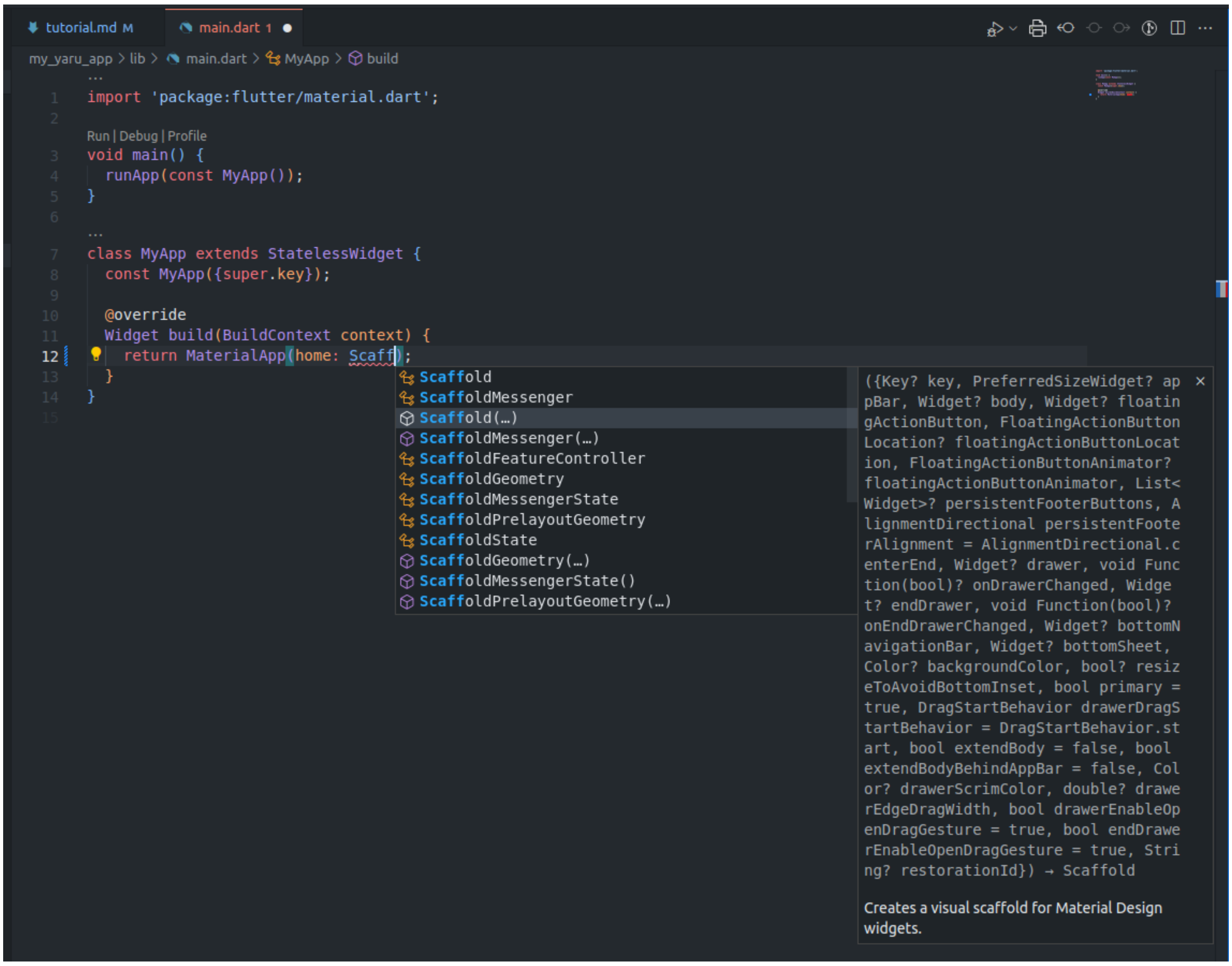
If [home] is specified, then [routes] must not include an entry for `/`, as [home] takes its place.

The [Navigator] is only built if routes are provided (either via [home], [routes], [onGenerateRoute], or [onUnknownRoute]); if they are not, [builder] must not be null.

The difference between using [home] and using [builder] is that the [home] subtree is inserted into the application below a [Navigator] (and thus below an [Overlay], which [Navigator] uses). With [home], therefore, dialog boxes will work automatically, the [routes] table will be used, and APIs such as [Navigator.push] and [Navigator.pop] will work as expected. In contrast, the widget returned from [builder] is inserted *above* the app's [Navigator] (if any).

Press enter, and write `Scaffold()`

VsCode then suggests:



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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(home: Scaff);
  }
}
```

```
({Key? key, PreferredSizeWidget? appBar, Widget? body, Widget? floatingActionButton, FloatingActionButtonLocation? floatingActionButtonLocation, FloatingActionButtonAnimator? floatingActionButtonAnimator, List<Widget>? persistentFooterButtons, AlignmentDirectional persistentFooterAlignment = AlignmentDirectional.centerEnd, Widget? drawer, void Function(bool)? onDrawerChanged, Widget? endDrawer, void Function(bool)? onEndDrawerChanged, Widget? bottomNavigationBar, Widget? bottomSheet, Color? backgroundColor, bool? resizeToAvoidBottomInset, bool primary = true, DragStartBehavior drawerDragStartBehavior = DragStartBehavior.start, bool extendBody = false, bool extendBodyBehindAppBar = false, Color? drawerScrimColor, double? drawerEdgeDragWidth, bool drawerEnableOpenDragGesture = true, bool endDrawerEnableOpenDragGesture = true, String? restorationId}) → Scaffold
```

Creates a visual scaffold for Material Design widgets.

Move the selection to `Scaffold()` by pressing your arrow-down key on your keyboard. Press enter when `Scaffold()` is selected.

Your code should now look like this:

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(home: Scaffold());
  }
}
```

Note: it is always better to let VsCode do the work by only typing until the code-completion (they call it "intellisense") popup shows up with suggestions. Pressing enter while one of the suggestions is selected is *always safer because you will avoid typing errors and because VsCode will often also make the necessary import for you, too. However, this was the last time we've written down the auto-complete-workflow, to not make this tutorial unnecessarily long.*

pub.dev

Dart: add dependencies

Yaru.dart

yaru_widgets.dart

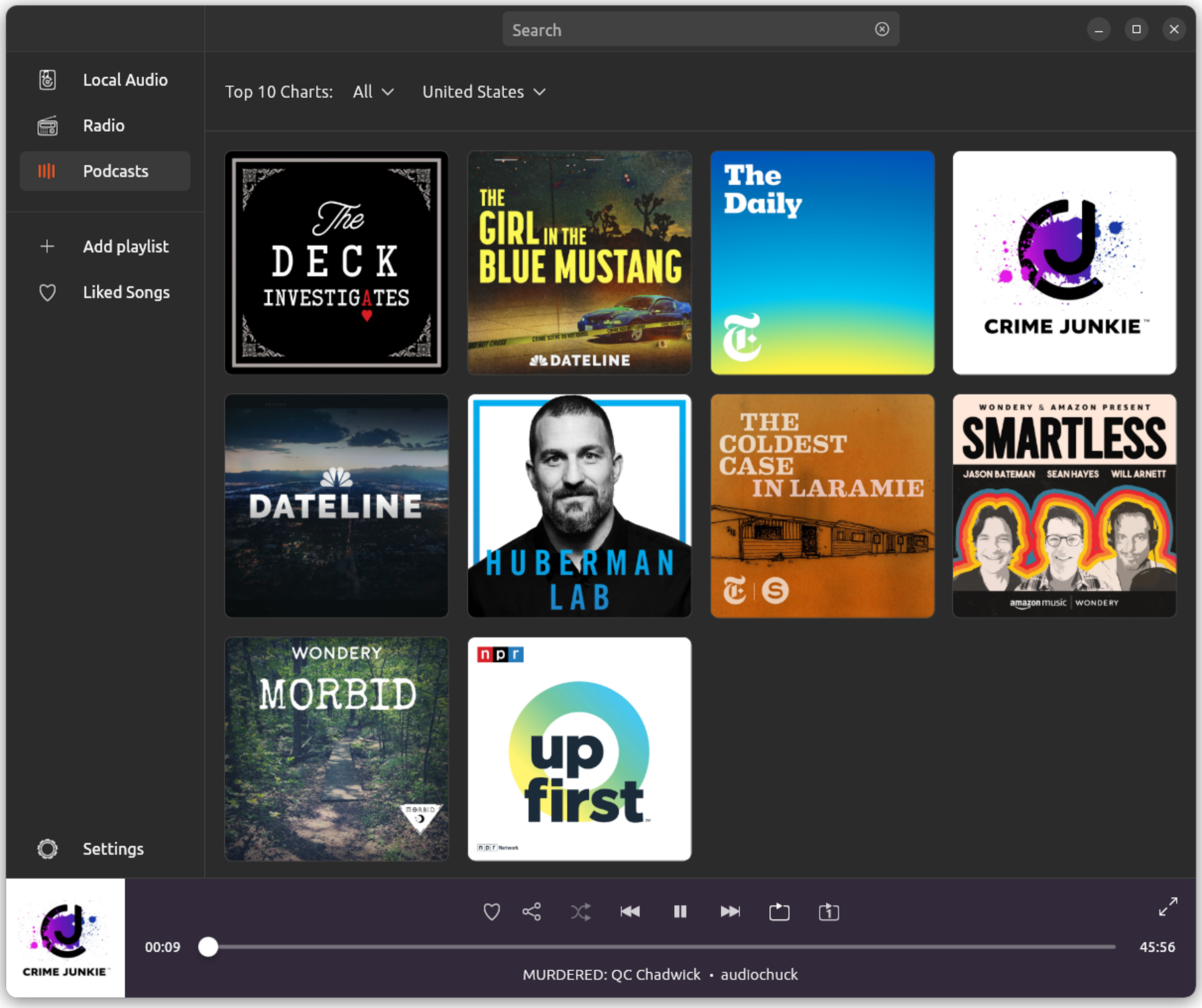
yaru_icons.dart

hand_window.dart

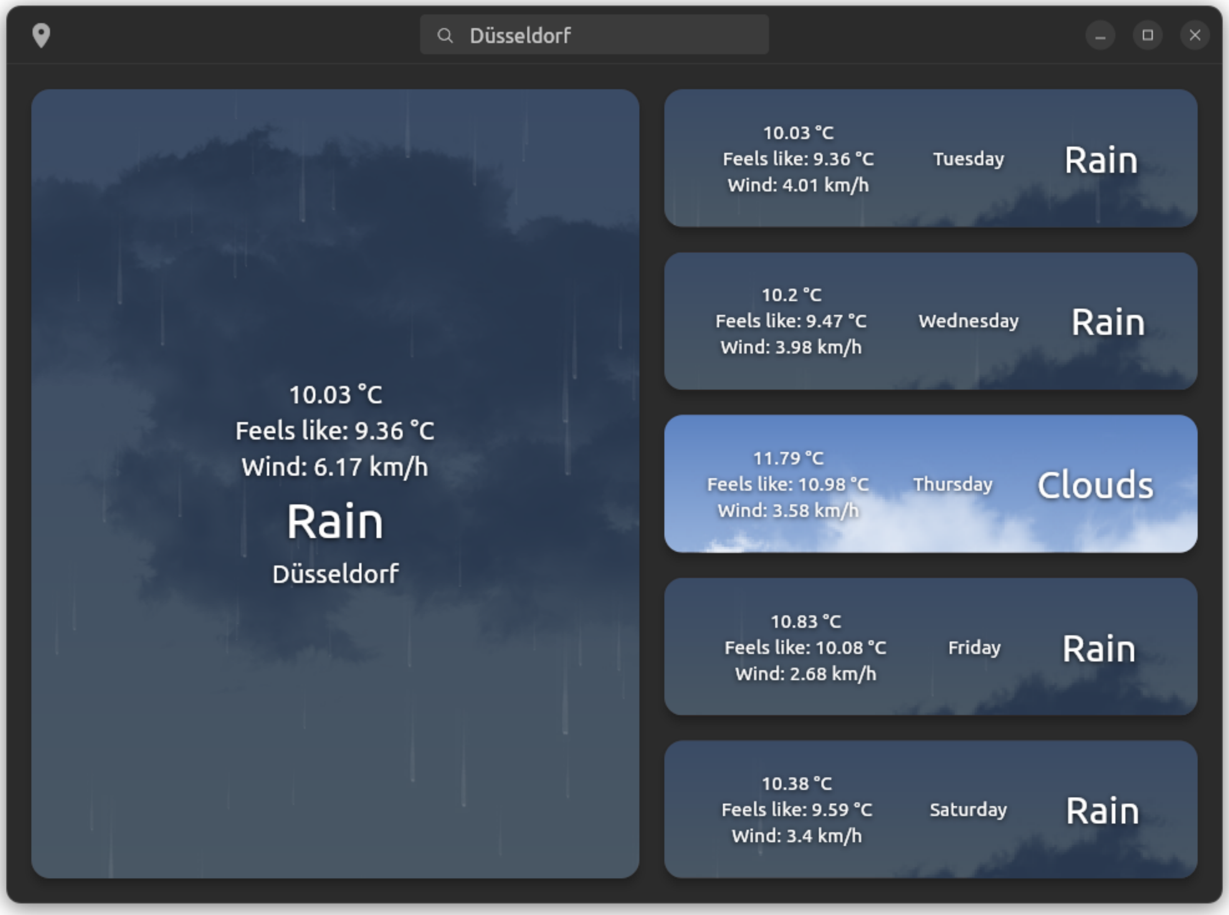
Types of apps + your ideas

Most of the desktop apps we've encountered could be classified into one of the following "concepts":

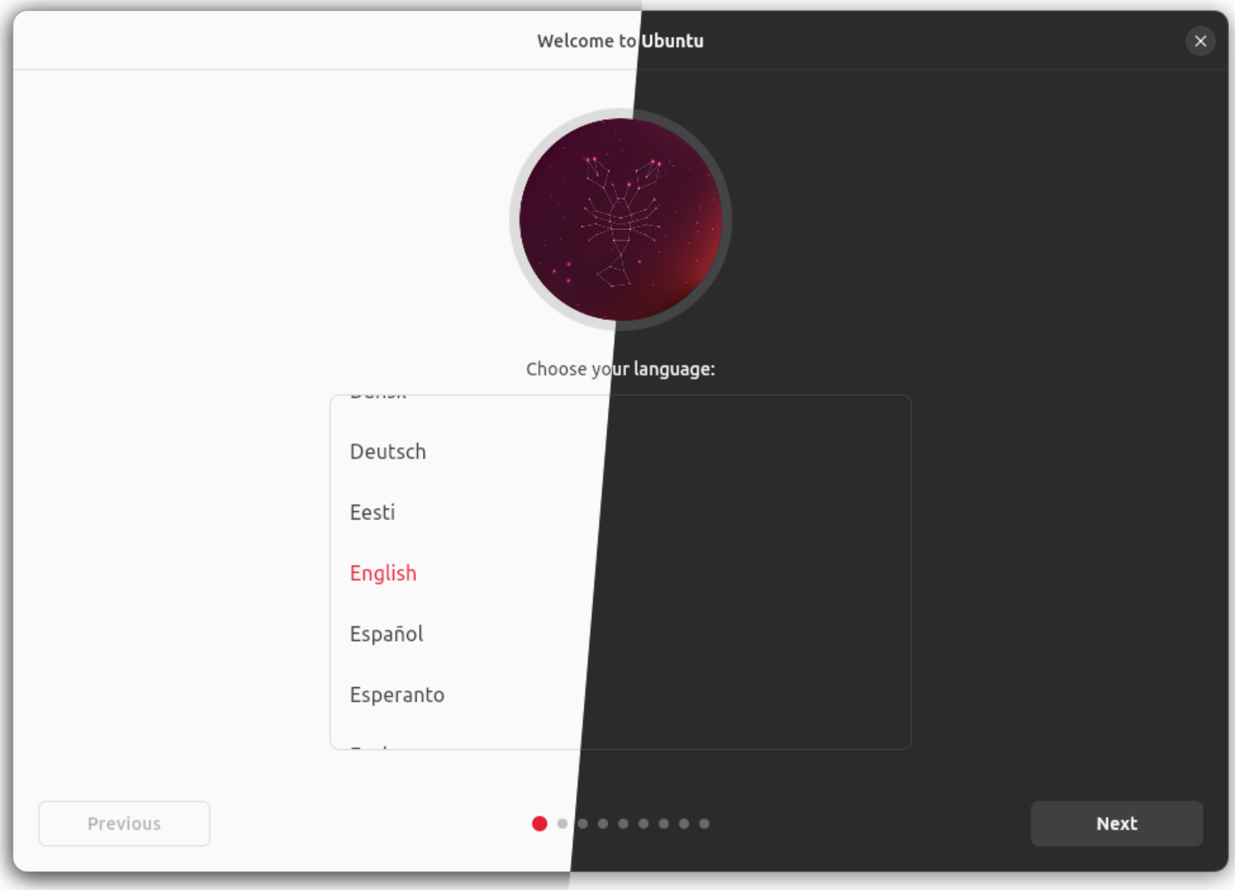
- Master/details apps



- single page apps *(the weather in Düsseldorf is kinda depressing atm)*



- wizard apps



That does not mean there aren't more types of apps and most importantly this should not limit your ideas and creativity in any way.