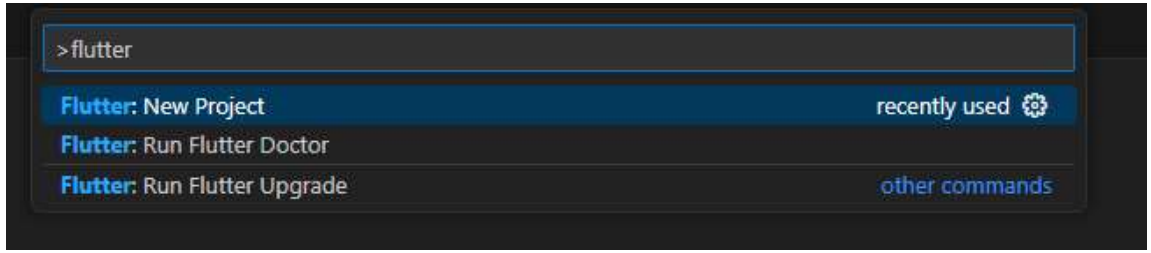
Your first experiences with Flutter

- Create a HelloWorld app in Flutter
- Run the app on Chrome and on Android emulator
- Change color and font of text widgets
- Observe your code changes directly with “Hot Reload” (no Rebuild needed)
- Learn about axis alignments of rows and columns
- Use buttons and images in your UI

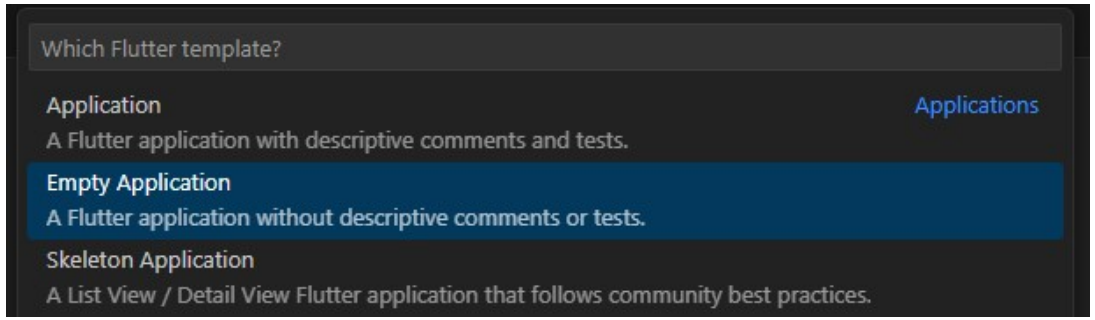
Create a HelloWorld Flutter app in VS Code (part 1)

Open VS Code and select menu “View / Command Palette …” (or simply press F1).

In the search field enter “flutter” and select “flutter: New Project”

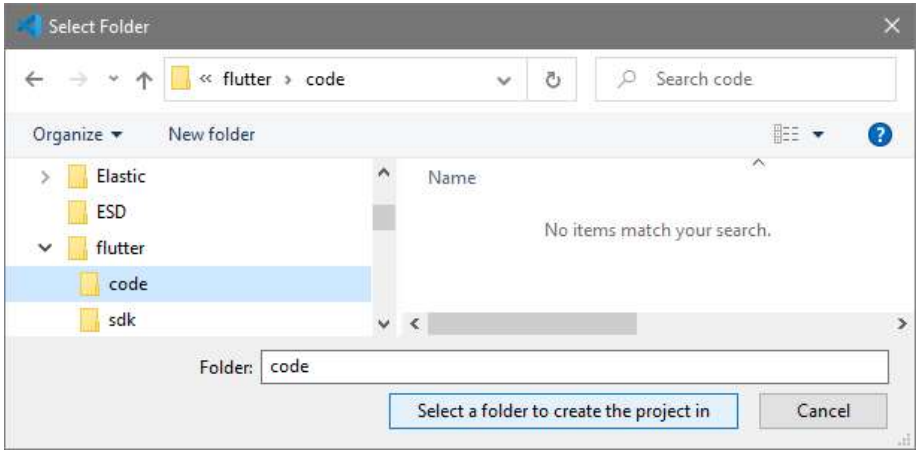


In the next drop-down, select “Empty Application”:

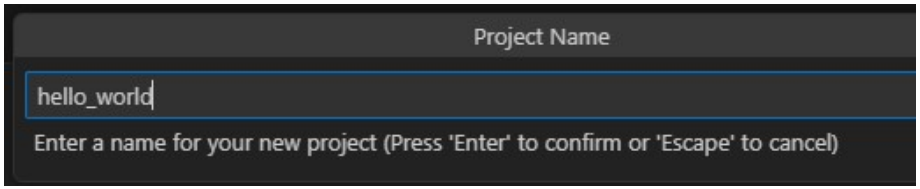


Create a HelloWorld Flutter app in VS Code (part 2)

Select the folder where the new project should be created, e.g. “C:\flutter\code”

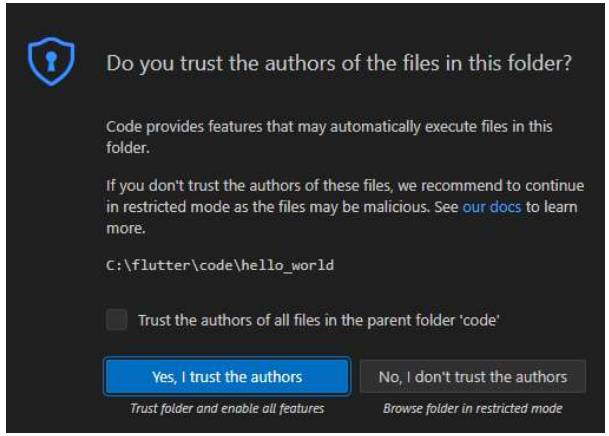


Enter the project name (no blanks or capital letters are allowed):

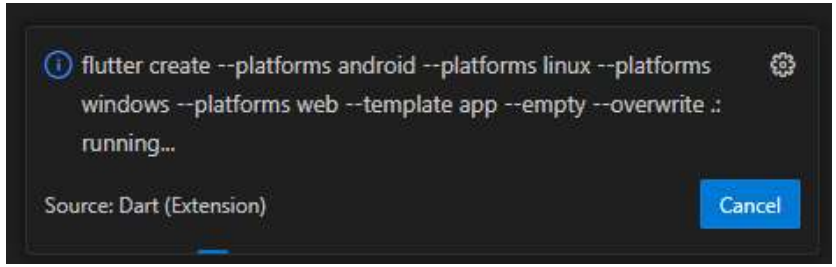


Create a HelloWorld Flutter app in VS Code (part 3)

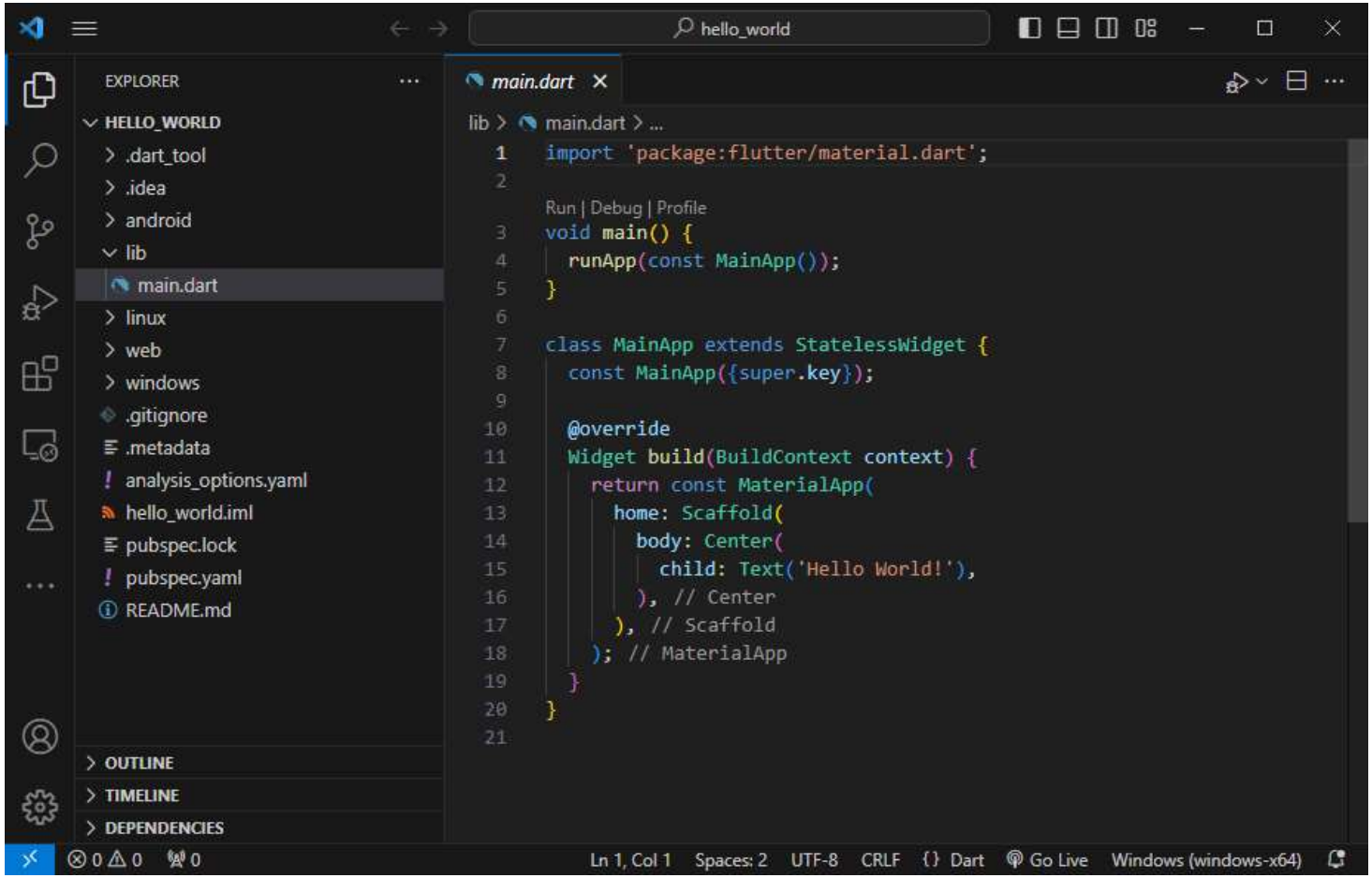
Allow VS Code to open the new created folder:



Wait until the project is created (you need an Internet connection during this step):

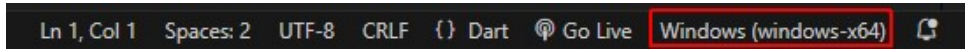


Your first created Flutter app in VS Code



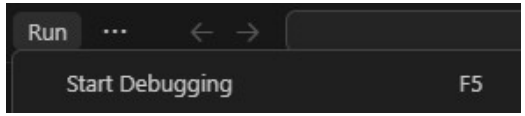
Test your app on Chrome or Edge (part 1)

In the bottom line of VS Code, tap the red marked area:



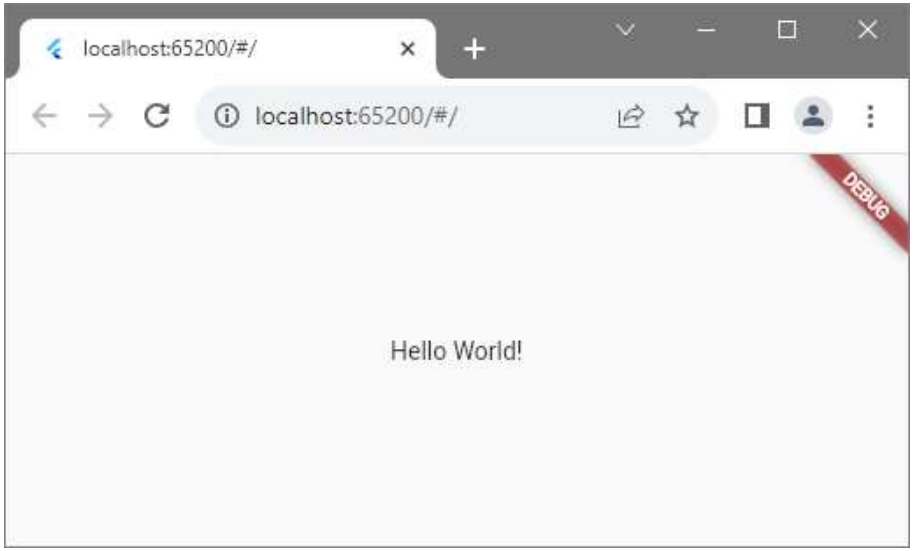
Select Chrome or Edge:

Select menu “Run / Start Debugging” or press F5:



Test your app on Chrome or Edge (part 2)

After some seconds, a Chrome or Edge window should come up showing your app:

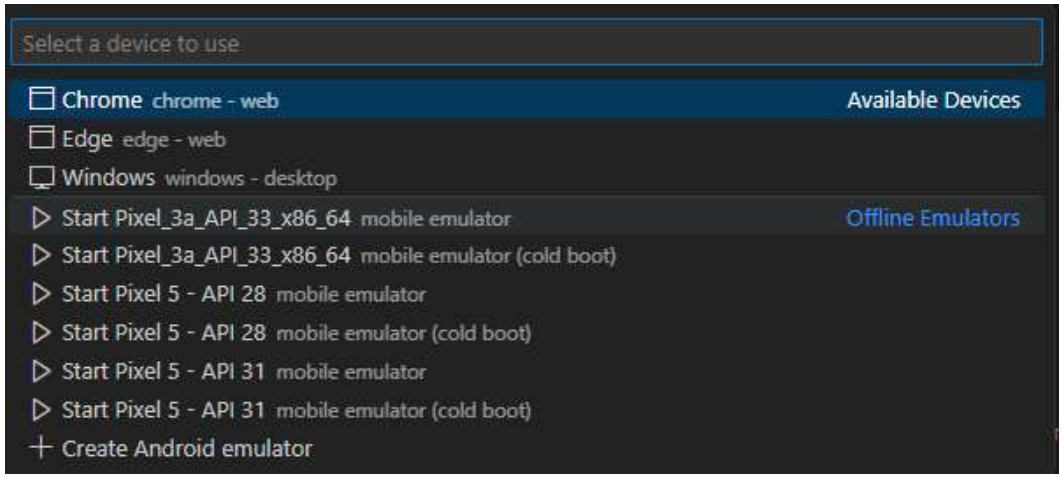


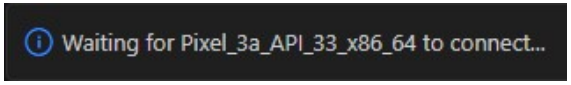
and in VS Code you see a “Debug Console”:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
Launching lib\main.dart on Chrome in debug mode...
This app is linked to the debug service: ws://127.0.0.1:65247/D3R7DtAdEmM=/ws
Debug service listening on ws://127.0.0.1:65247/D3R7DtAdEmM=/ws
Connecting to VM Service at ws://127.0.0.1:65247/D3R7DtAdEmM=/ws
```

Test your app on Pixel Emulator (part 1)

In case your PC has an Intel CPU supporting VT-x, you can select a Pixel emulator for tests:

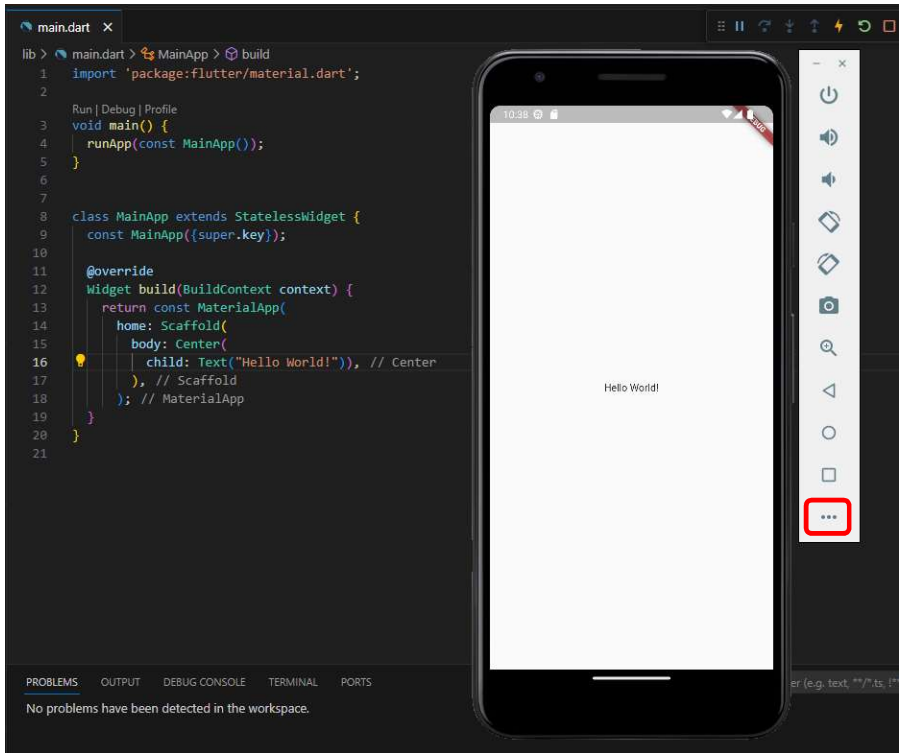


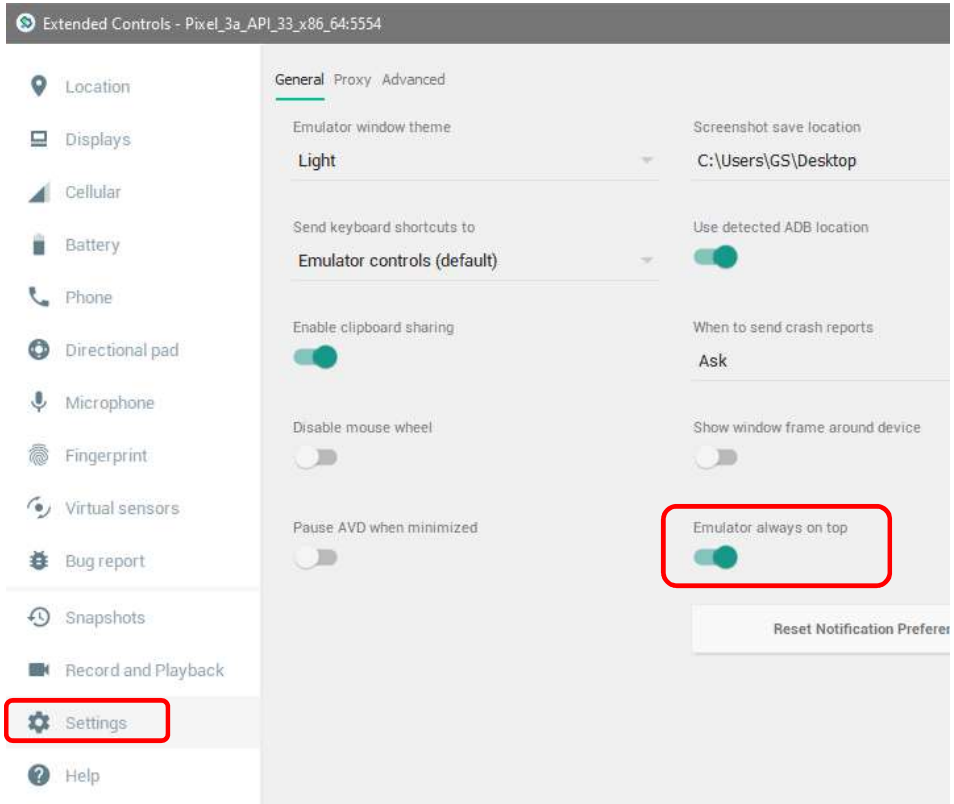
It takes some time to start: 

Then press F5 to start debugging. Take care: first build may take more than a minute.

Test your app on Pixel Emulator (part 2)

Emulator appears on top of VS Code and stays on-top with setting:



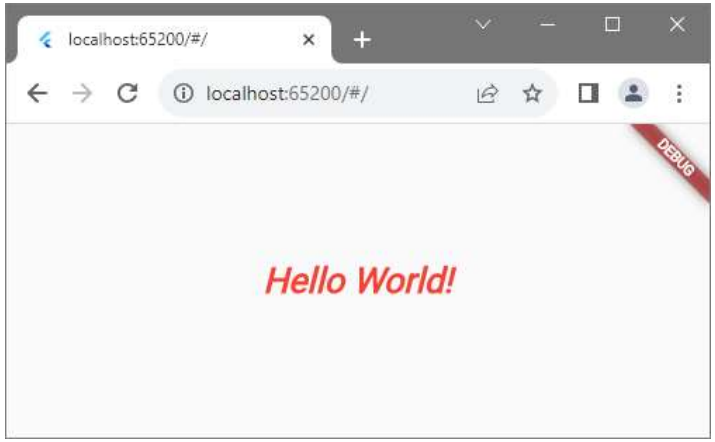


Change some Text properties

Define a style for your Text widget:

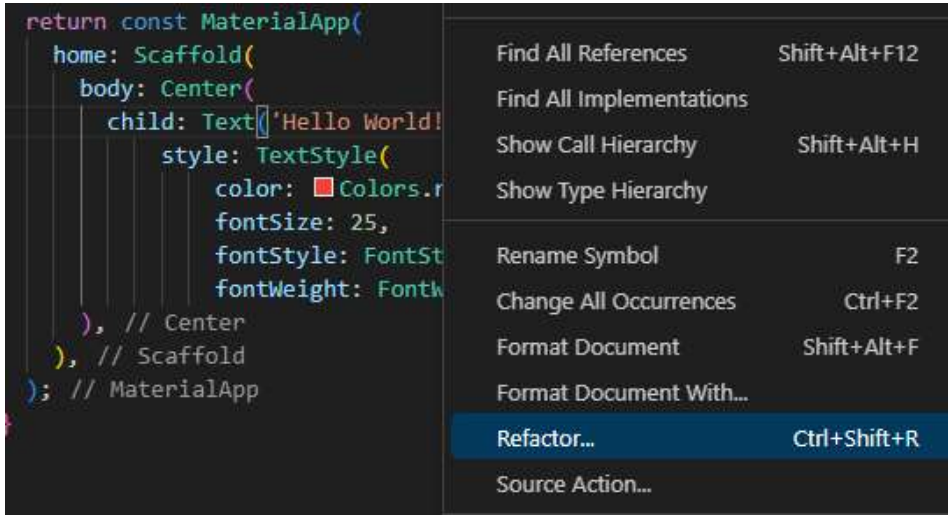
```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    home: Scaffold(
      body: Center(
        child: Text('Hello World!',
          style: TextStyle(
            color: Colors.red,
            fontSize: 25,
            fontStyle: FontStyle.italic,
            fontWeight: FontWeight.bold)), // TextStyle // Text
      ), // Center
    ), // Scaffold
  ); // MaterialApp
}
```

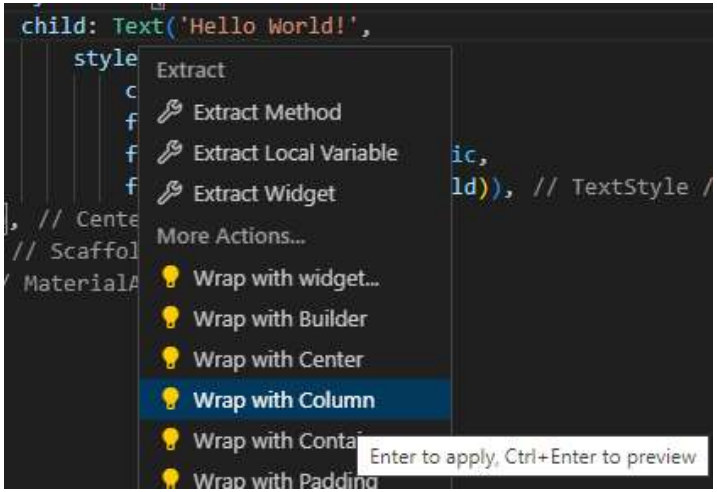
After saving your code, a “Hot Reload” is performed automatically and you can see the changes in Chrome:



Allow more widgets by introducing a Column

Right-Click on your Text widget and select “Refactor”, then select “Wrap with Column”:

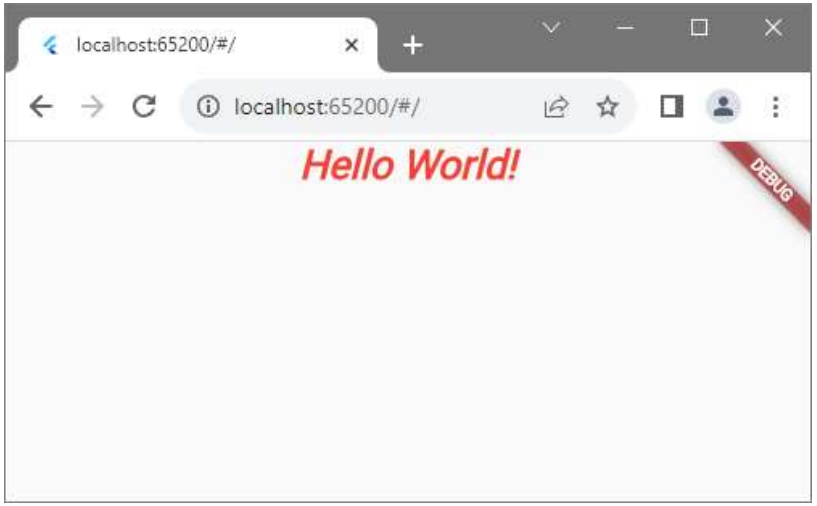




This will add a “Column” widget around your Text. Column widgets can have several children:

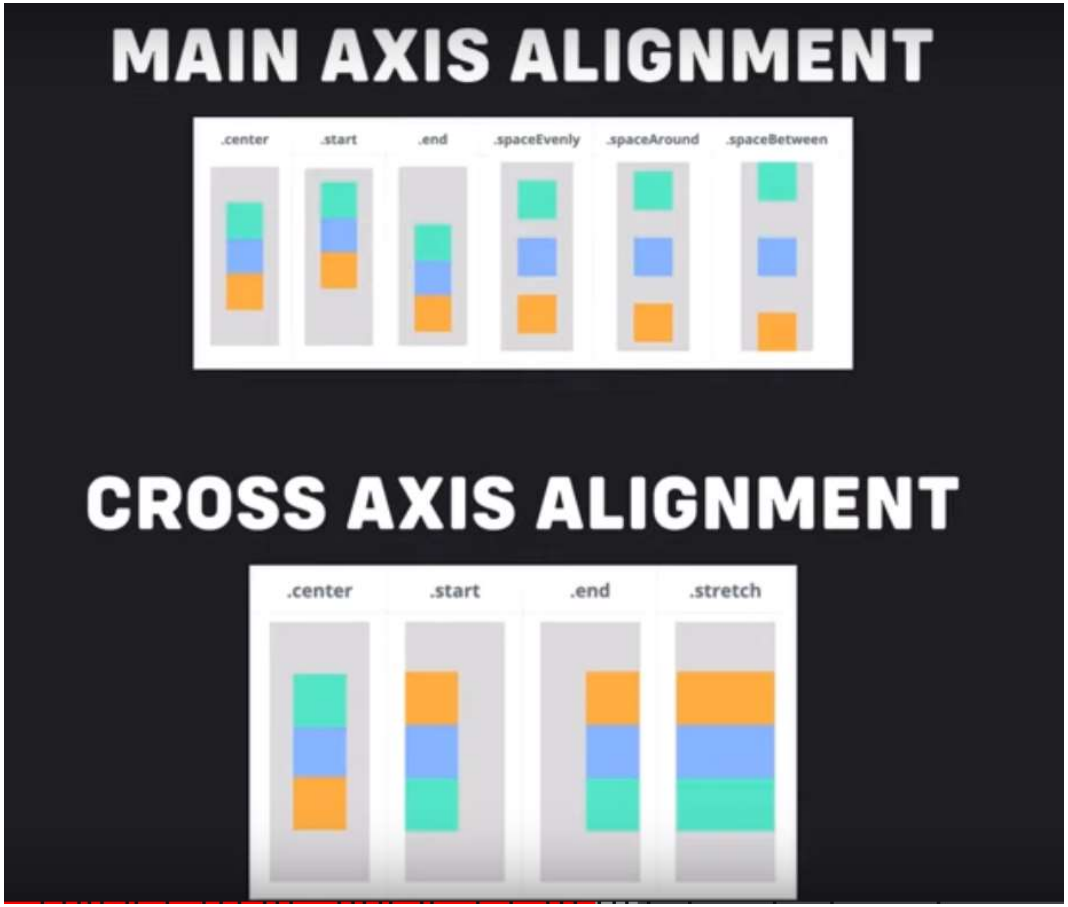
Allow more widgets by introducing a Column

After saving your code, the “Hello World” text will move up, because Columns take the whole space and put their children by default on the top of the column:



You can change this by setting the MainAxisAlignment property of the Column:

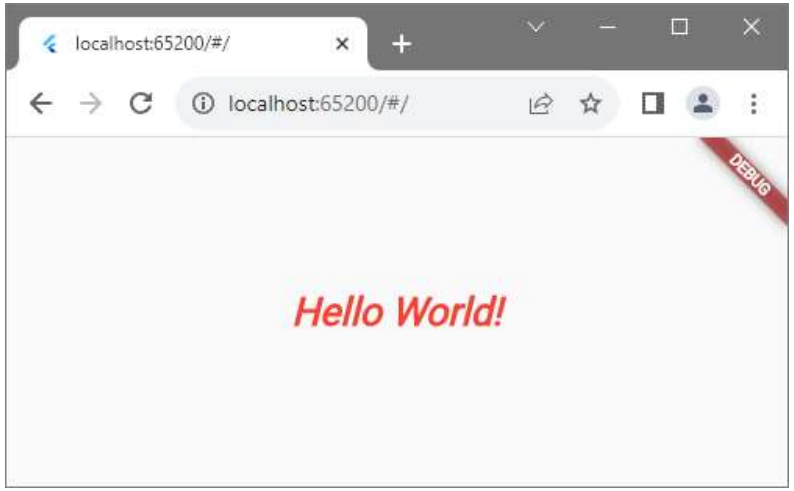
Axis alignments of a Column



Picture taken from https://www.youtube.com/watch?v=D4nhaszNW4o

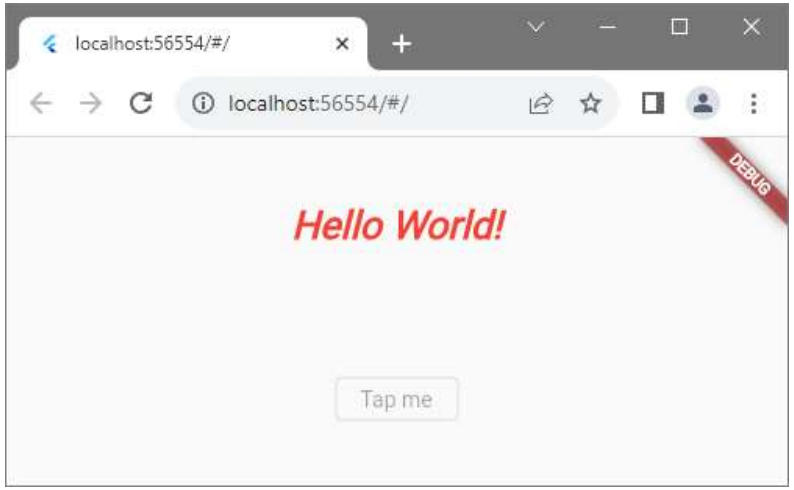
Center the text again with MainAxisAlignment

```
Widget build(BuildContext context) {
  return const MaterialApp(
    home: Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('Hello World!',
                style: TextStyle(
                    color: Colors.red,
                    fontSize: 25,
                    fontStyle: FontStyle.italic,
                    fontWeight: FontWeight.bold)), // TextStyle // Text
          ],
        ), // Column
      ), // Center
    ), // Scaffold
  ); // MaterialApp
}
```



Add an OutlinedButton to the UI

```
child: Column(
  mainAxisAlignment: MainAxisAlignment.spaceAround,
  children: [
    Text('Hello World!',
        style: TextStyle(
            color: Colors.red,
            fontSize: 25,
            fontStyle: FontStyle.italic,
            fontWeight: FontWeight.bold)), // TextStyle // Text
    OutlinedButton(onPressed: null, child: Text("Tap me"))
  ],
```



The button is disabled as long as “onPressed” is null.

Define an “onPressed” handler

Either with a new function (can be inside or outside the class, normally inside):

```
OutlinedButton(onPressed: handlePressed, child: Text("Tap me"))
```

```
void handlePressed() {
  print ("in handlePressed");
}
```

Or use an anonymous function:

```
OutlinedButton(
    onPressed: () {
      print("OutlinedButton was pressed");
    },
    child: Text("Tap me")) // OutlinedButton
```