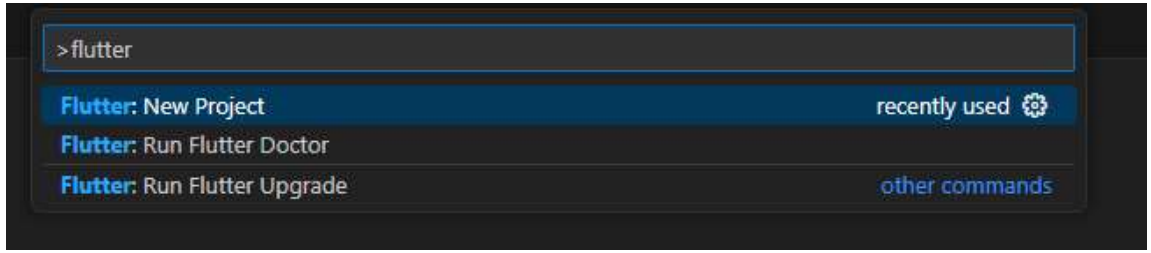
Your first experiences with Flutter

- 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

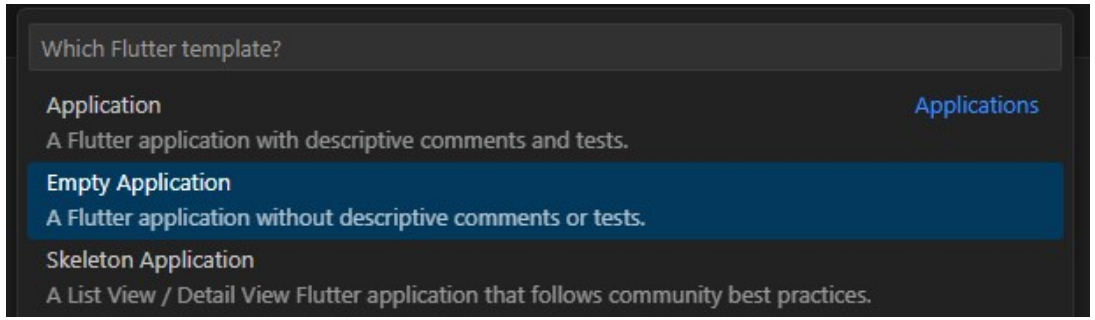
Create a HelloWorld Flutter app in VS Code (part 1)

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

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

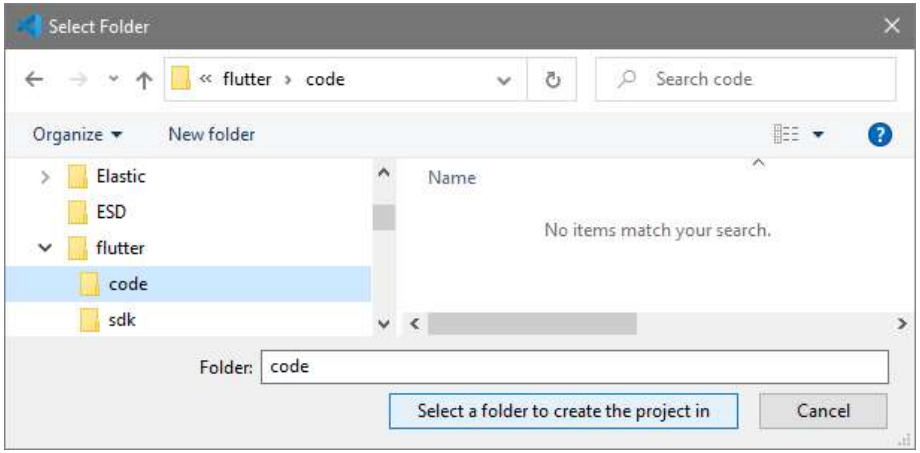


In the next drop-down, select “Empty Application”:

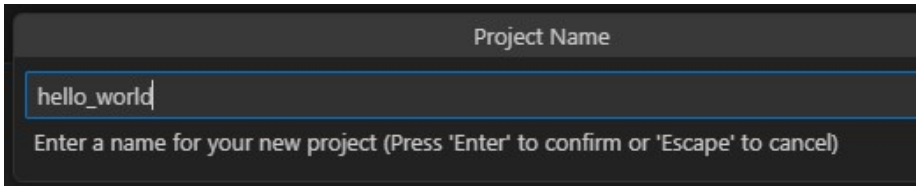


Create a HelloWorld Flutter app in VS Code (part 2)

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

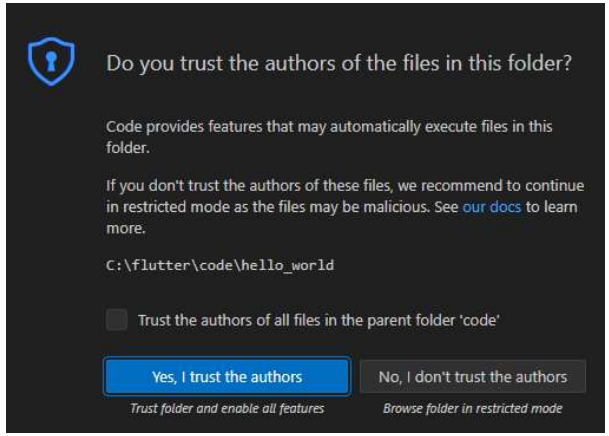


Enter the project name (no blanks or capital letters are allowed):

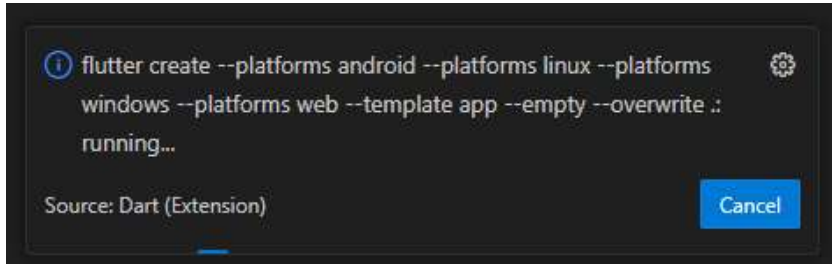


Create a HelloWorld Flutter app in VS Code (part 3)

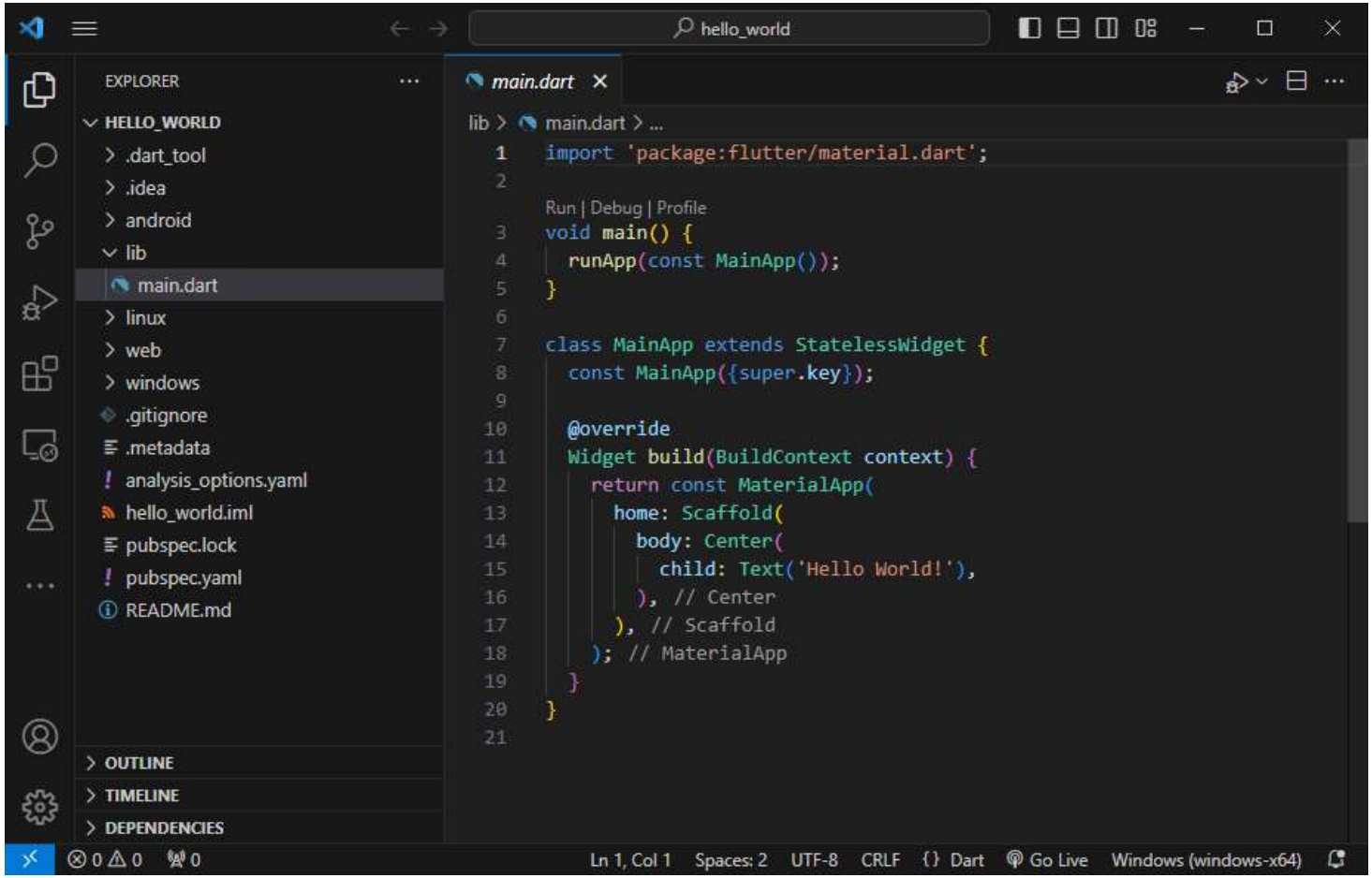
Allow VS Code to open the new created folder:



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

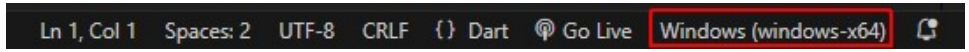


Your first created Flutter app in VS Code



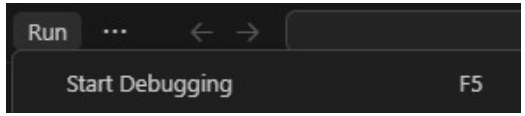
Test your app on Chrome or Edge (part 1)

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



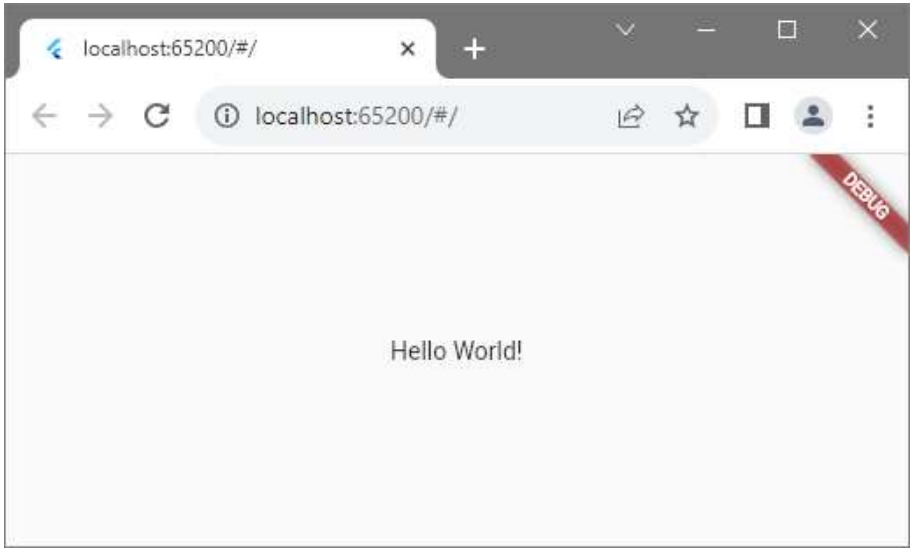
Select Chrome or Edge:

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:

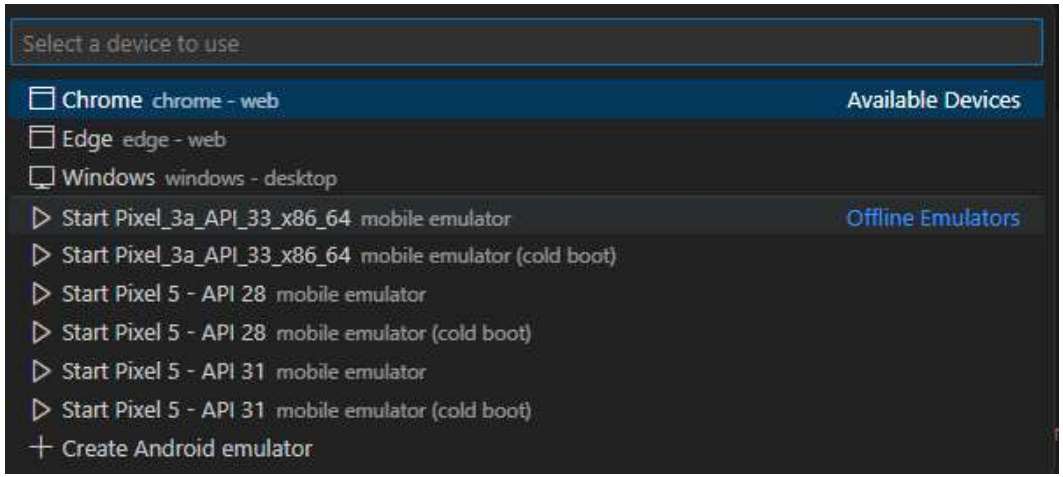


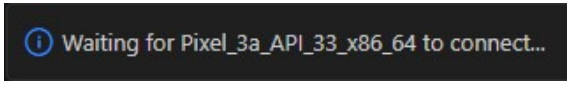
and in VS Code you see a “Debug Console”:

```
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:

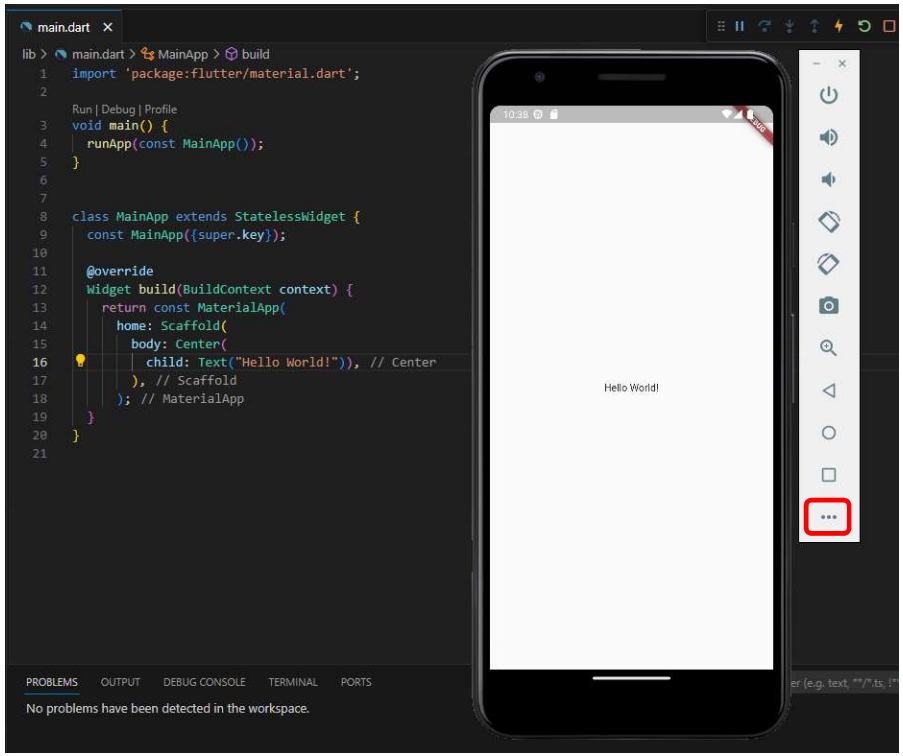


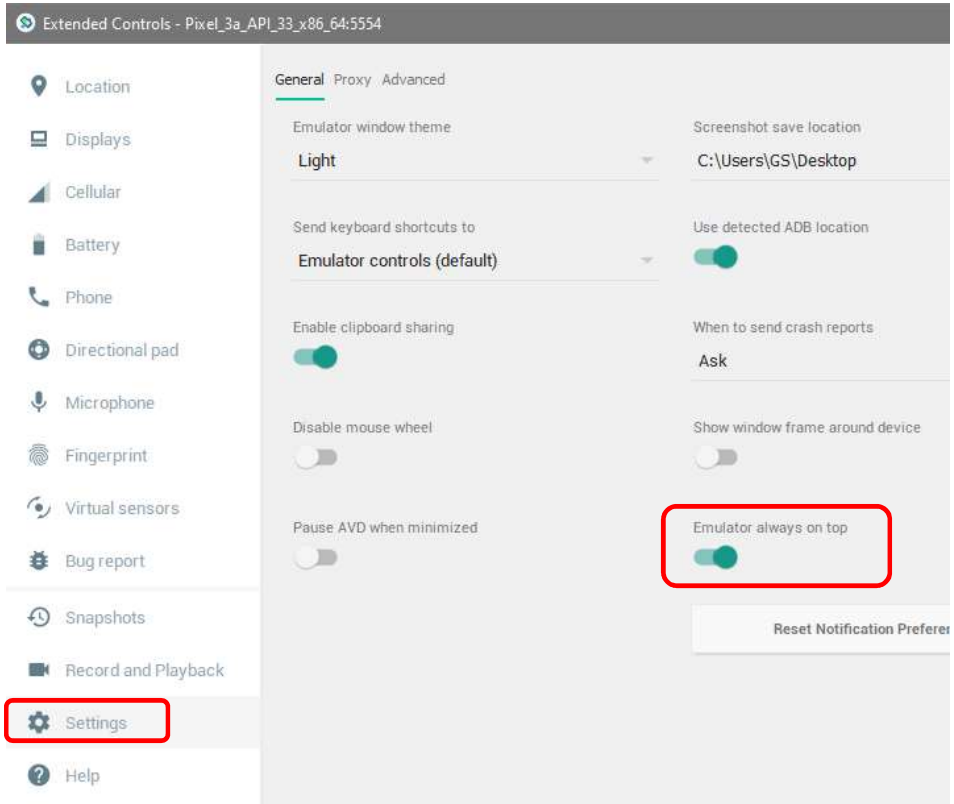
It takes some time to start: 

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:



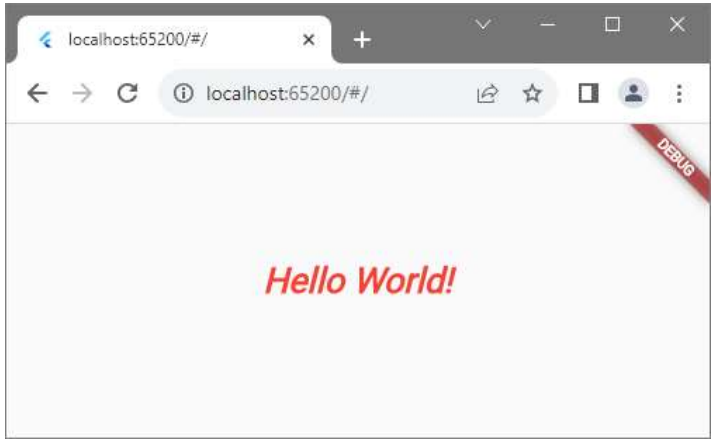


Change some Text properties

Define a style for your Text widget:

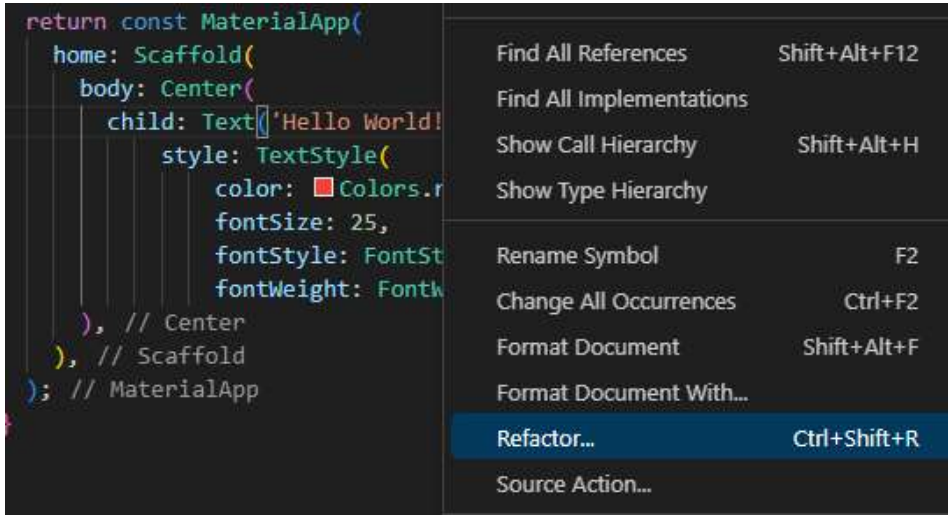
```
@override
Widget build(BuildContext context) {
  return const MaterialApp(
    home: Scaffold(
      body: Center(
        child: Text('Hello World!',
          style: TextStyle(
            color: Colors.red,
            fontSize: 25,
            fontStyle: FontStyle.italic,
            fontWeight: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

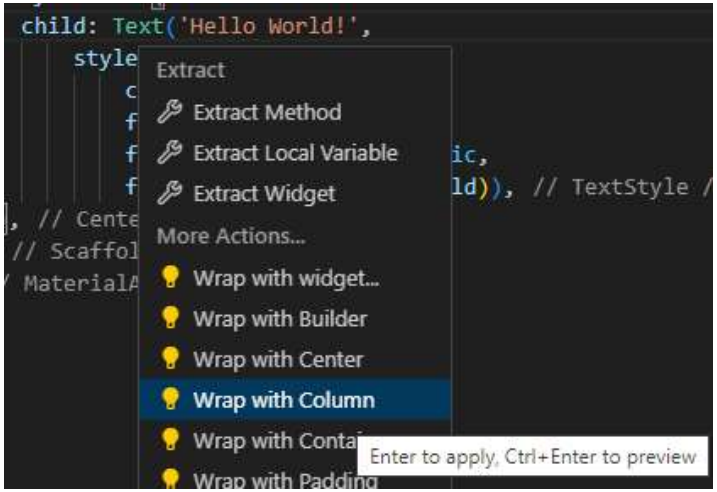
After saving your code, a “Hot Reload” is performed automatically and you can see the changes in Chrome:



Allow more widgets by introducing a Column

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

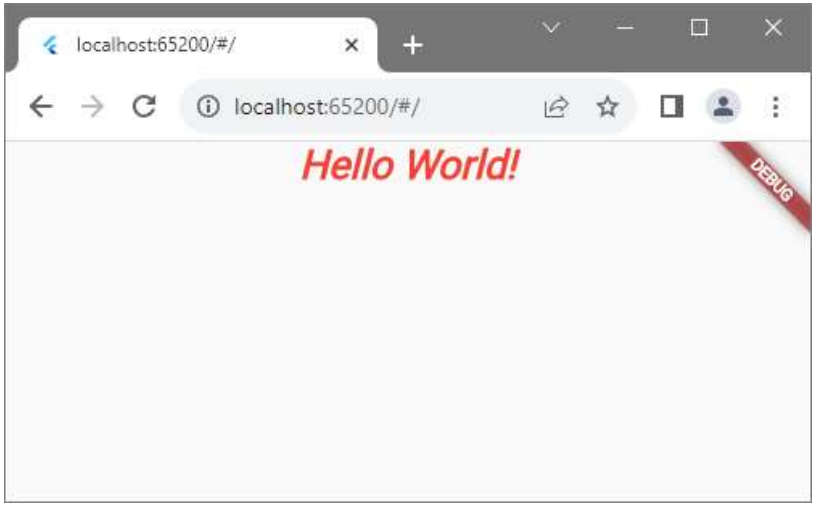




This will add a “Column” widget around your Text. Column widgets can have several children:

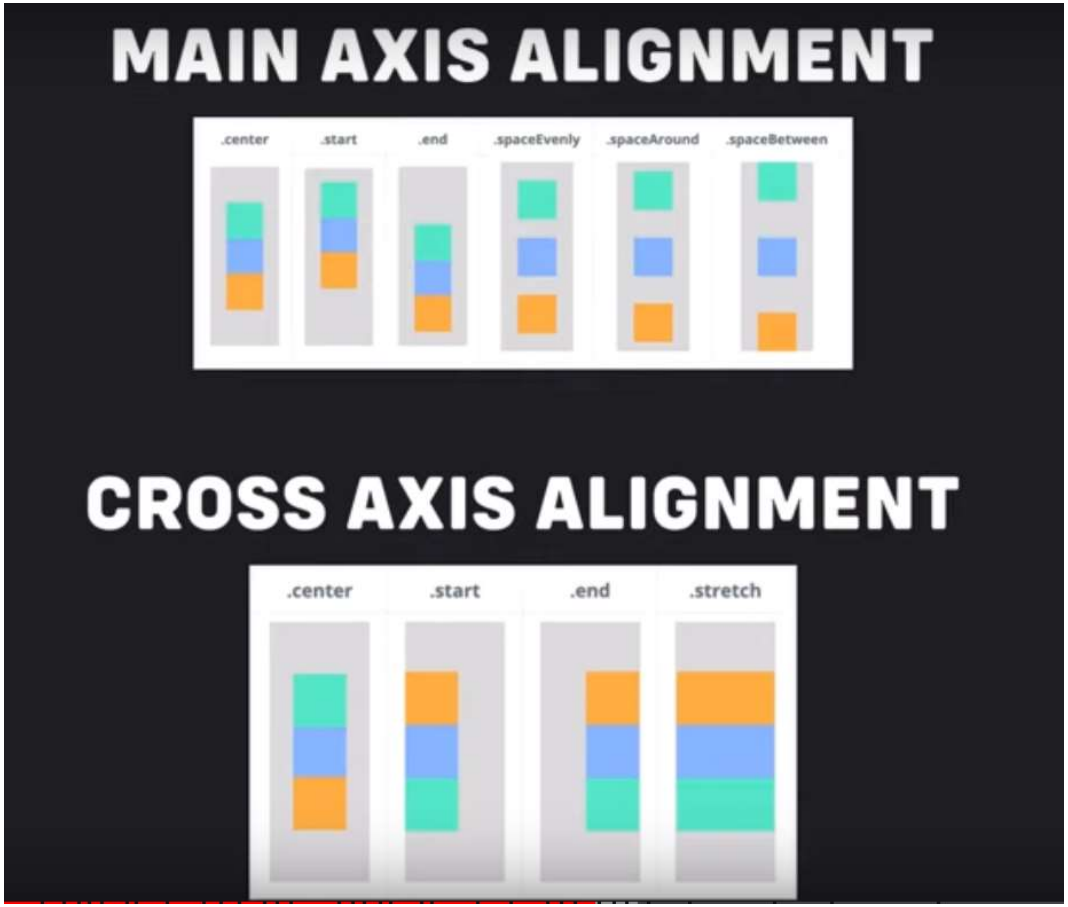
Allow more widgets by introducing a Column

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:

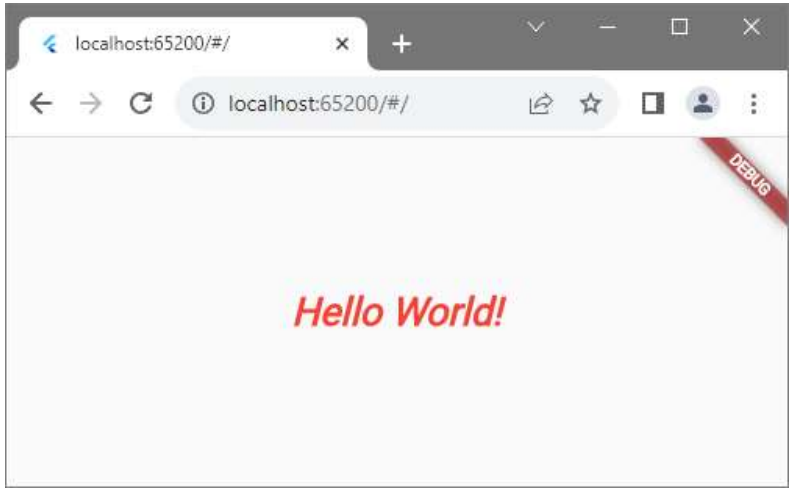
Axis alignments of a Column



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

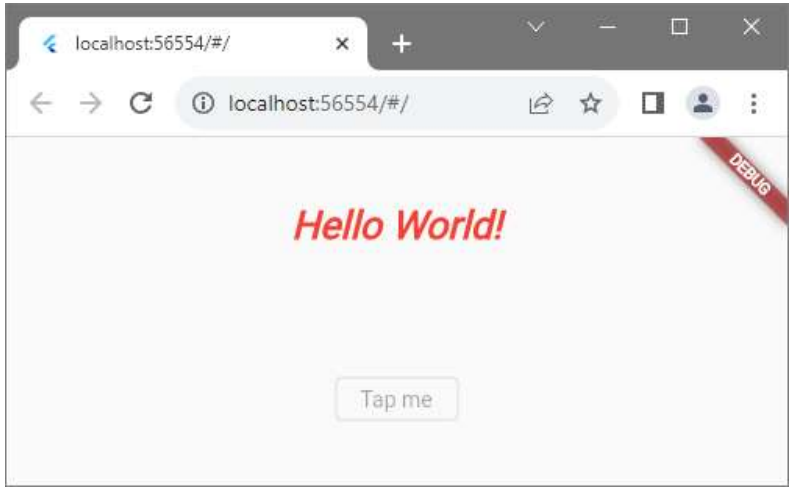
Center the text again with MainAxisAlignment

```
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

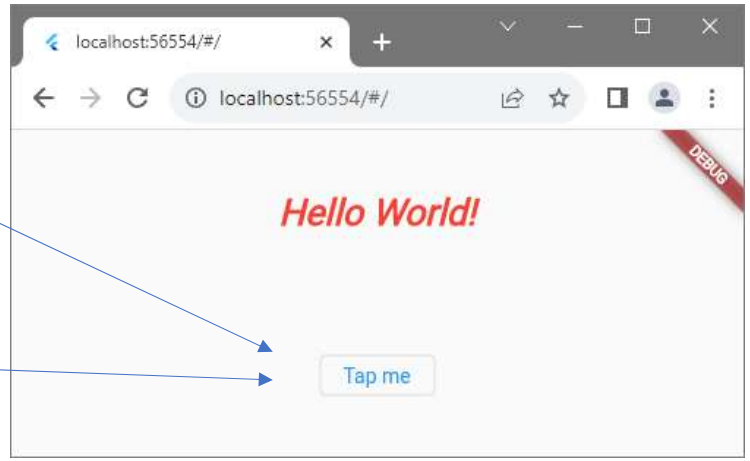
Either with a new function (can be inside or outside the class, normally inside):

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

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

Or use an anonymous function:

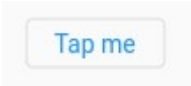
```
OutlinedButton(
    onPressed: () {
      print("OutlinedButton was pressed");
    },
    child: Text("Tap me")) // OutlinedButton
```



Style the button

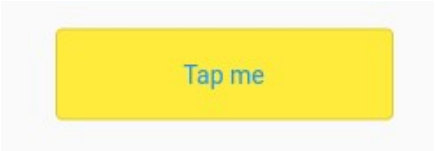
Without style:

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



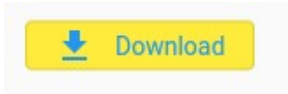
With style:

```
OutlinedButton(
    style: OutlinedButton.styleFrom(
      minimumSize: Size(200, 60),
      backgroundColor: Colors.yellow,
    ),
    onPressed: () {
      print("OutlinedButton was pressed");
    },
    child: Text("Tap me")) // OutlinedButton
```

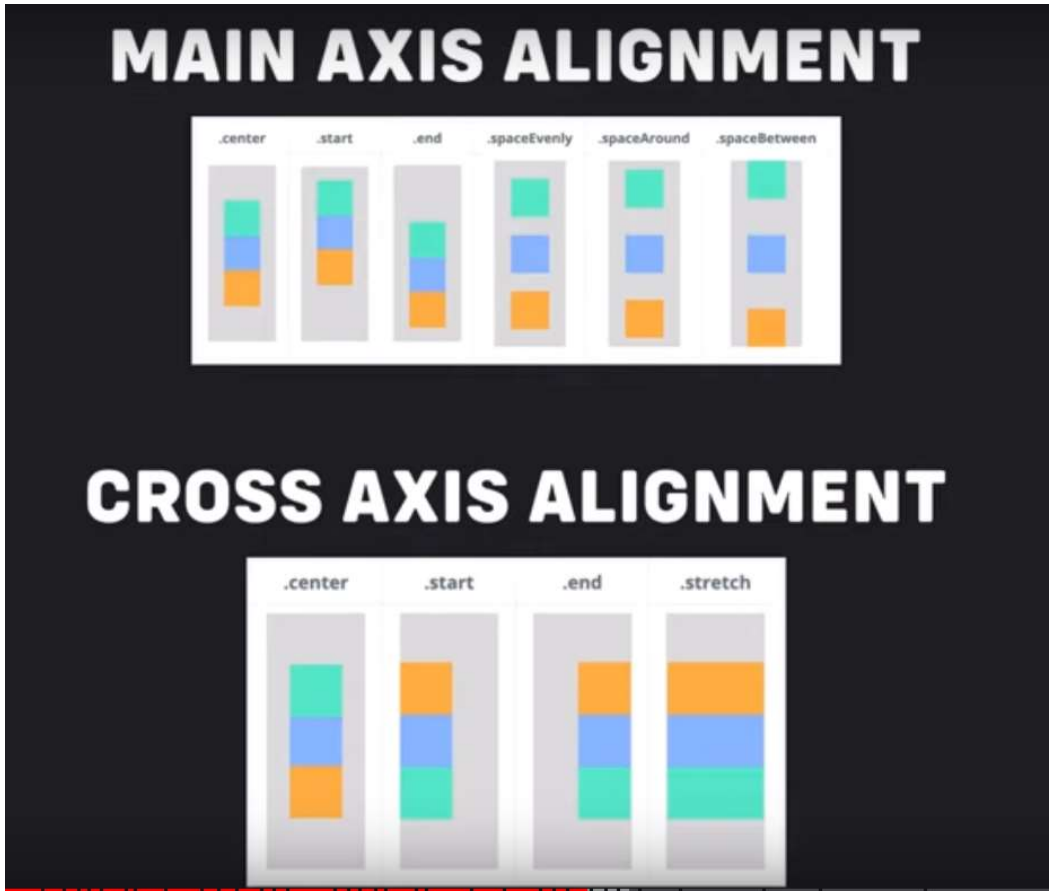


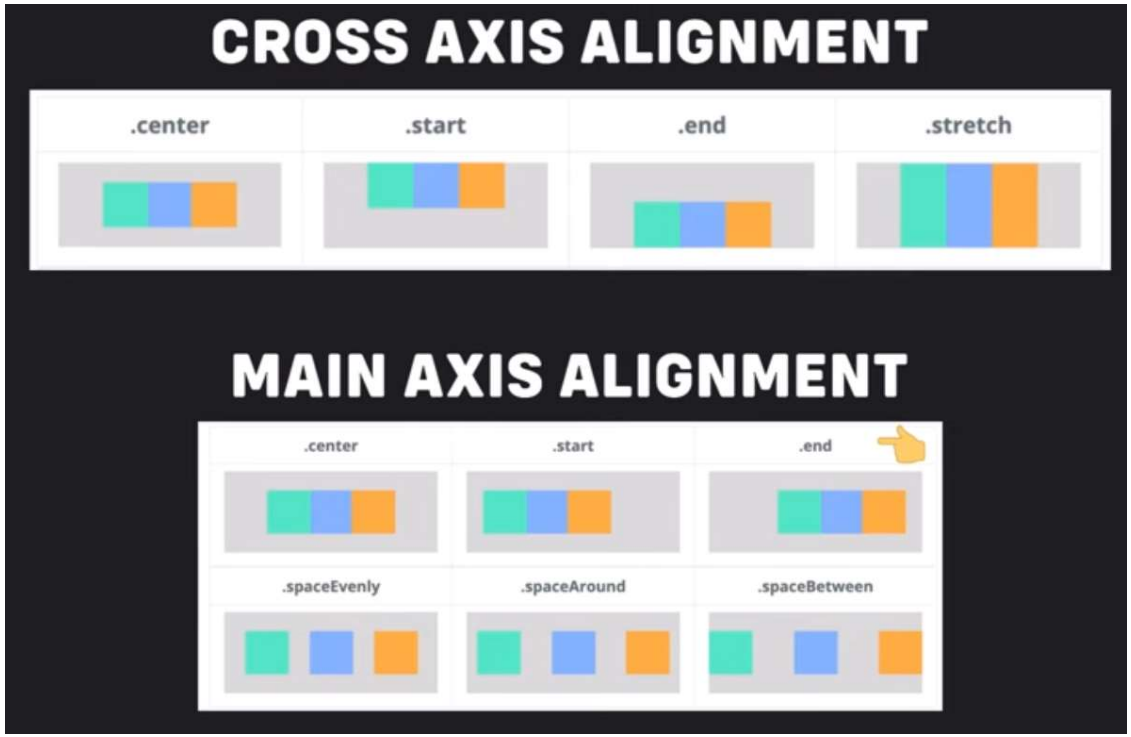
Add an icon inside the button

```
OutlinedButton(
    style: OutlinedButton.styleFrom(
      //minimumSize: Size(200, 60),
      maximumSize: Size(130, 50),
      backgroundColor: Colors.yellow,
    ),
    onPressed: () {
      print("OutlinedButton was pressed");
    },
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Icon(Icons.download),
        Text("Download"),
      ],
    )) // Row // OutlinedButton
```



Axis alignments of a Column … compared to a Row





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

More Button widgets in Flutter


```
TextButton(onPressed:() { }, child: Text("TextButton")),
OutlinedButton(onPressed: () { }, child: Text("OutlinedButton")),
ElevatedButton(onPressed: () { }, child: Text("ElevatedButton")),
IconButton(onPressed: () { }, icon: Icon(Icons.pedal_bike)),
```

Show Images from the network

```
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Text('Hello World!', style: TextStyle(color: Colors.red, fontSize: 24)),
    OutlinedButton(
        style: OutlinedButton.styleFrom(backgroundColor: Colors.yellow,
        maximumSize: Size(130, 50)),
        onPressed: () {},
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Icon(Icons.download),
            Text("Download"),
          ],
        )), // Row // OutlinedButton
    Image.network(
      "https://fdg-ab.de/wp-content/uploads/2021/03/logo_fdg_neu_freigestellt.png",
    ), // Image.network
```

Reduce the size of the image by setting width or height or both:

```
Image.network(
  "https://fdg-ab.de/wp-content/uploads/2021/03/logo_fdg_neu_freigestellt.png",
  width: 120
), // Image.network
```

Link to the used FDG image for copy/paste: https://fdg-ab.de/wp-content/uploads/2021/03/logo_fdg_neu_freigestellt.png

Transparent background

PNG format allows images to have a transparent background.

Our used FDG logo has such a transparent background.

To test this you can set e.g. the scaffold's background color:

```
return MaterialApp(
  home: Scaffold(
    backgroundColor: Color.fromARGB(255, 166, 189, 231),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text('Hello World!', style: TextStyle(color: Colors.red, fontSize: 24)),
          OutlinedButton(
```

BTW: JPG format does not allow transparent background !

API Doc for Color.fromARGB

Color.fromARGB constructor

```
const Color.fromARGB(
  int a,
  int r,
  int g,
  int b
)
```

Construct a color from the lower 8 bits of four integers.

- a is the alpha value, with 0 being transparent and 255 being fully opaque.
- r is red, from 0 to 255.
- g is green, from 0 to 255.
- b is blue, from 0 to 255.

Out of range values are brought into range using modulo 255.

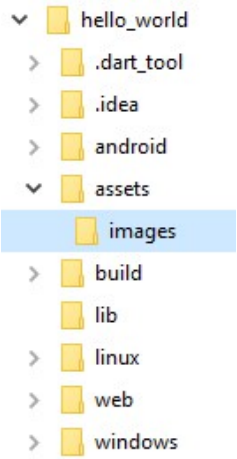
See also fromRGBO, which takes the alpha value as a floating point value.

Copied from https://api.flutter.dev/flutter/dart-ui/Color/Color.fromARGB.html

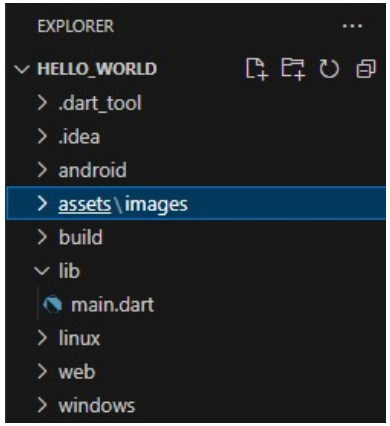
Show your own images (part 1)

1) In your project's directory, create a new directory assets and therein a directory images
(you can name these directories as you want and if you like create only one directory instead of 2 nested ones):

In Windows Explorer:



In VS Code:



2) Put the image you want to show as .jpg or .png file in this directory.

BTW: you can use Windows Paint or "Irfan View" to cut out your images e.g. from screenshots and save them.
But keep in mind: **images might be licensed** !

Show your own images (part 2)

3) Define an **Image.asset** in your code referencing your image file:

```
Image.asset("assets/images/snoopy_laptop.jpg",
  width: 140), // Image.asset
```

Take care: use “forward slashes” / in the expression, no “backward slashes” \ as in Windows!

4) To make this work, add the following 2 lines at the end of your “**pubspec.yaml”** file (this is described e.g. in https://docs.flutter.dev/ui/assets/assets-and-images):

```
flutter:
  uses-material-design: true
  assets:
    - assets/images/
```

Images with rounded corners

Surround your image with a ClipRRect widget (stands for “Clip on Rounded Rectangle”) :

```
ClipRRect(
  borderRadius: BorderRadius.circular(20.0),
  child: Image.asset("assets/images/snoopy_laptop.jpg",
  width: 140)), // Image.asset // ClipRRect
```


```
ClipRRect(
    borderRadius: BorderRadius.only(
        topLeft: Radius.circular(30), bottomRight: Radius.circular(30)),
    child: Image.asset("assets/images/snoopy_laptop.jpg", width: 140)),
```

```
ClipOval(child: Image.asset("assets/images/snoopy_laptop.jpg", width: 140)),
```