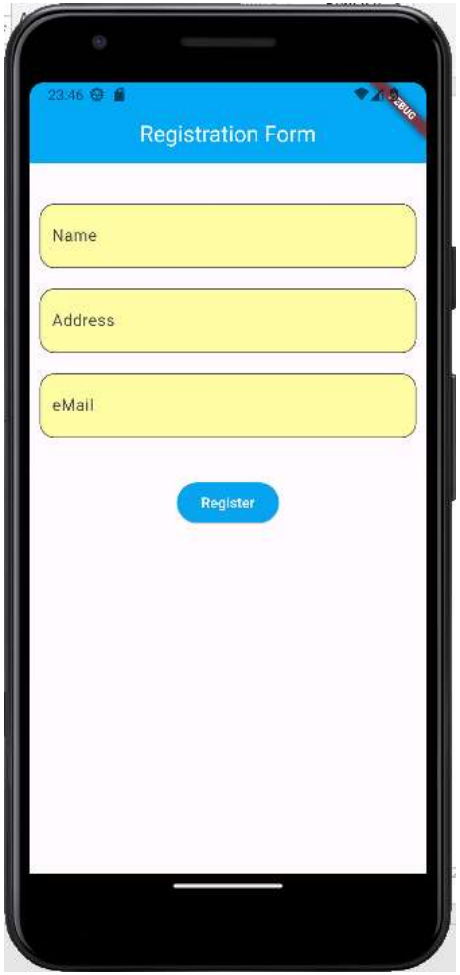
Define the position and size of UI elements

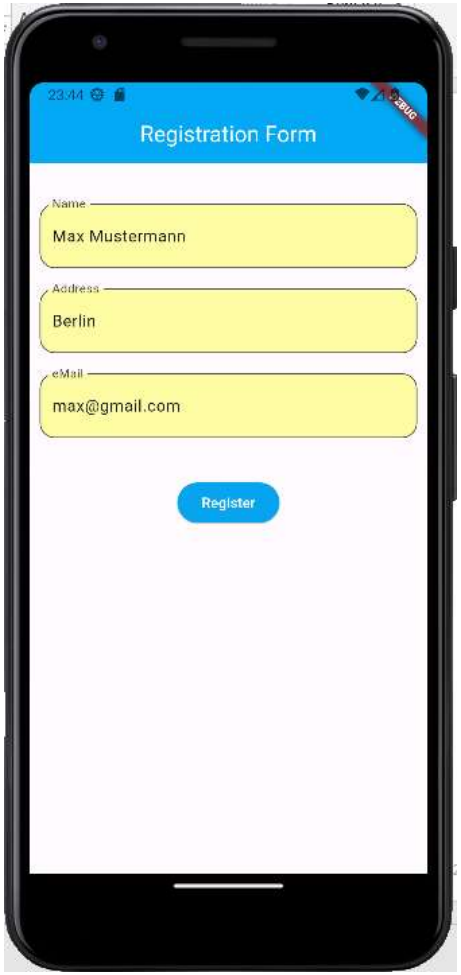
- Implement a simple registration form
- What is the difference between a “SizedBox” and a “Container”
- What is “margin” and “padding” for a container
- How to define the border of a Container
- What are gradient colors

Goal: create the layout of a simple registration form

Without input:

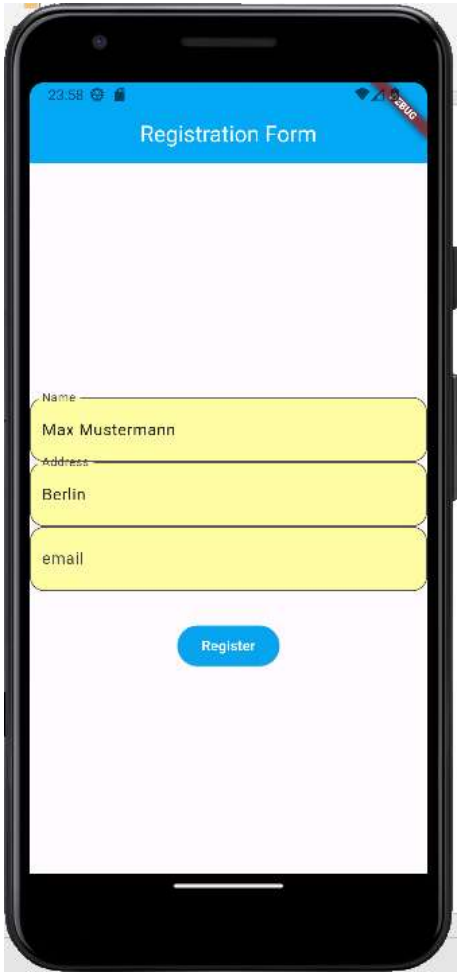


Filled:

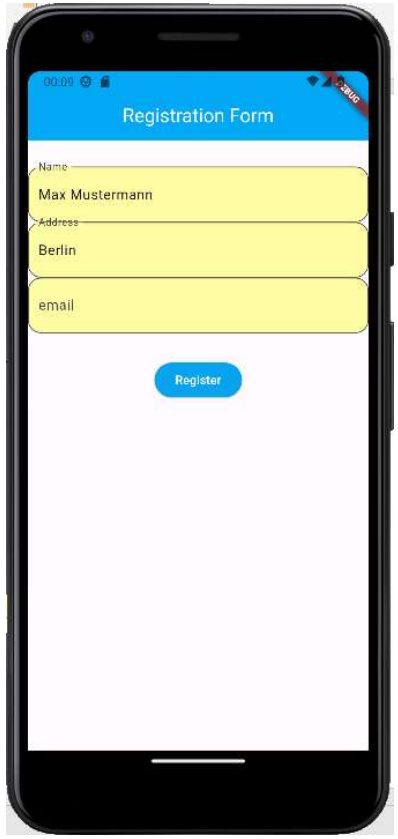


Arranging with Column.MainAxisAlignment

```
body: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const SizedBox(height: 30),
    TextField( // TextField ···
    TextField( // TextField ···
    TextField( // TextField ···
    const SizedBox(height: 30),
    ElevatedButton( // ElevatedButton ···
  ],
```



AxisAlignments do not fully fit our needs here




```
body: Column(
  mainAxisAlignment: MainAxisAlignment.start,
```

```
MainAxisAlignment.spaceEvenly,
```

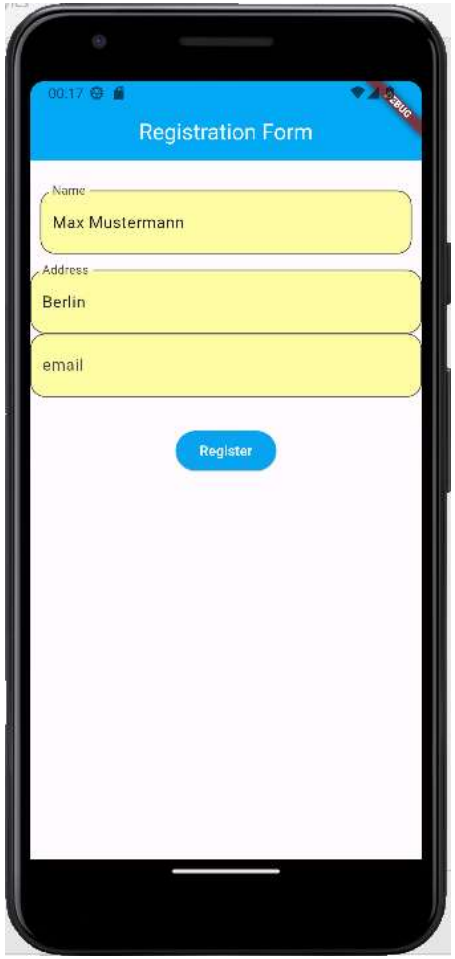

```
MainAxisAlignment.spaceBetween,
```

Issue when wrapping TextField in a row

A TextField has no property width. The first idea to make it “smaller” was to wrap it in a row and put SizedBoxes in front and behind it:

Solution A) Wrapping TextField with a SizedBox

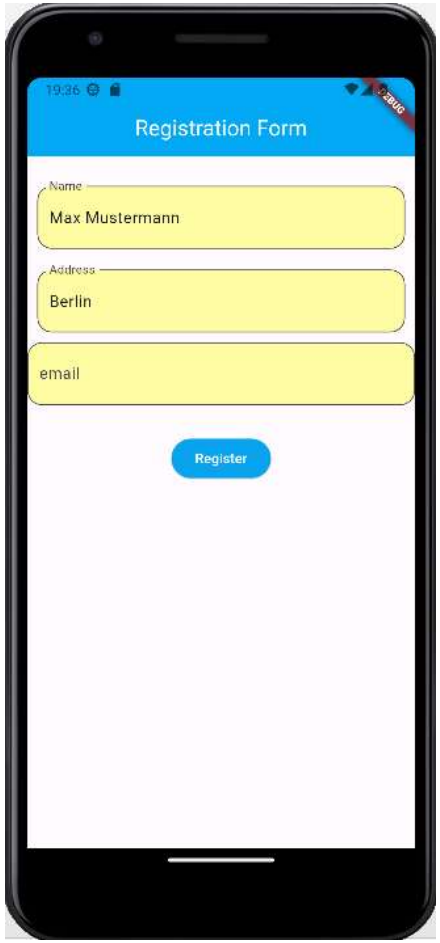
```
body: Column(
  mainAxisAlignment: MainAxisAlignment.start,
  children: [
    const SizedBox(height: 30),
    SizedBox(
      width: MediaQuery.of(context).size.width - 20,
      height: 80,
      child: TextField(
        decoration: InputDecoration(
          filled: true,
          fillColor: myYellow,
          labelText: "Name",
          border: const OutlineInputBorder(
              borderRadius: BorderRadius.all(Radius.circular(15)))),
        ), // InputDecoration
```



BTW: I retrieved “height: 80” by try-and-error.

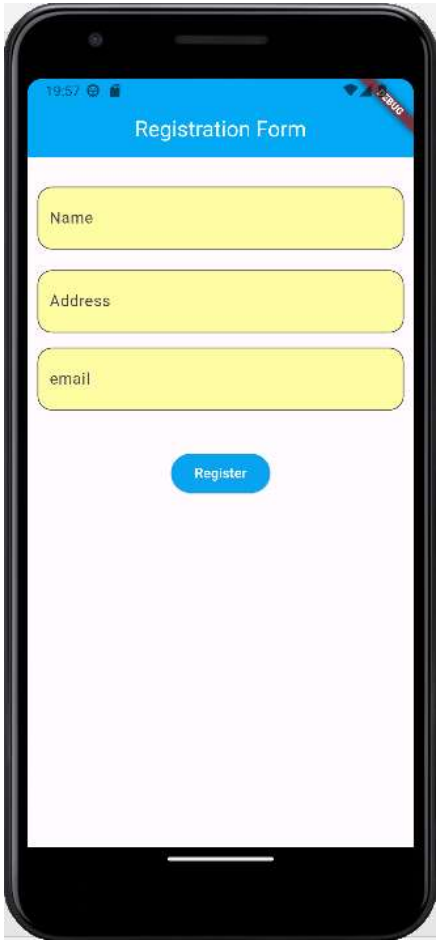
Solution B) Wrapping TextField with Padding

```
Padding(
  padding: const EdgeInsets.fromLTRB(10, 5, 10, 10),
  child: TextField(
    decoration: InputDecoration(
      filled: true,
      fillColor: myYellow,
      labelText: "Address",
      border: const OutlineInputBorder(
          borderRadius: BorderRadius.all(Radius.circular(15))),
    ), // InputDecoration
    onChanged: (value) {
      setState(() {
        address = value;
      });
    },
  ), // TextField
), // Padding
```



Solution C) Wrapping TextField with a Container

```
Container(
  padding: const EdgeInsets.fromLTRB(10, 5, 10, 10),
  child: TextField(
    decoration: InputDecoration(
      filled: true,
      fillColor: myYellow,
      labelText: "email",
      border: const OutlineInputBorder(
          borderRadius: BorderRadius.all(Radius.circular(15)))),
    ), // InputDecoration
    onChanged: (value) {
      setState(() {
        email = value;
      });
    },
  ), // TextField
), // Container
```



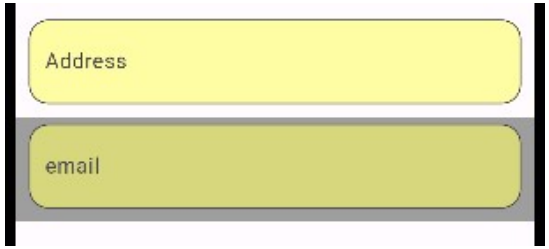
Compare SizedBox with Container

```
(new) SizedBox SizedBox({
  Key? key,
  double? width,
  double? height,
  Widget? child,
})
```

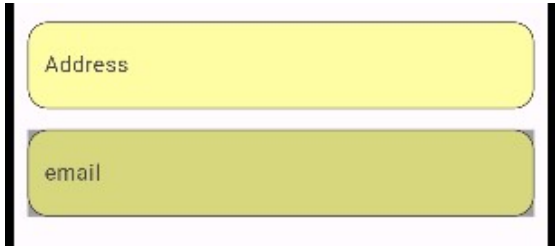
```
(new) Container Container({
  Key? key,
  AlignmentGeometry? alignment,
  EdgeInsetsGeometry? padding,
  Color? color,
  Decoration? decoration,
  Decoration? foregroundDecoration,
  double? width,
  double? height,
  BoxConstraints? constraints,
  EdgeInsetsGeometry? margin,
  Matrix4? transform,
  AlignmentGeometry? transformAlignment,
  Widget? child,
  Clip clipBehavior = Clip.none,
})
```

Margin as alternative to padding in a Container

```
Container(
  color: Colors.grey,
  padding: const EdgeInsets.fromLTRB(10, 5, 10, 10),
  child: TextField(
    decoration: InputDecoration(
      filled: true,
      fillColor: myYellow,
      labelText: "email",
```




```
Container(
  color: Colors.grey,
  //padding: const EdgeInsets.fromLTRB(10, 5, 10, 10),
  margin: const EdgeInsets.fromLTRB(10, 5, 10, 10),
  child: TextField(
    decoration: InputDecoration(
      filled: true,
      fillColor: myYellow,
      labelText: "email",
```



Margin vs. Padding

```
margin: const EdgeInsets.fromLTRB(10, 0, 10, 20),
```

Margin is the space around the widget. For example, from the edge of the container to the edge of the phone screen.

Padding is the space within the widget. For example, from the edge of the container to the text in it.

```
padding: const EdgeInsets.fromLTRB(10, 0, 10, 20),
```

Text was copied from
https://stackoverflow.com/questions/67351333/what-the-difference-between-margin-and-padding-in-container-widget-using-flutter

Finalizing registration form (part I)

To avoid code duplication, we create one line of the form in a method:

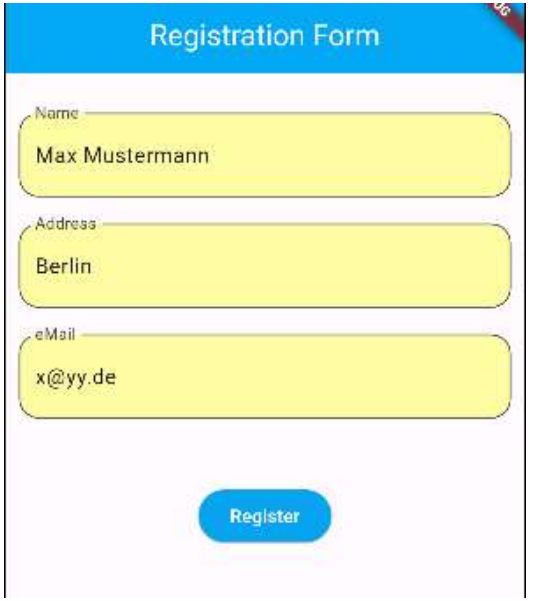
```
Widget getRegistrationLine(String label, void Function(String) callback) {
  return Padding(
    padding: const EdgeInsets.fromLTRB(10, 5, 10, 10),
    child: TextField(
      decoration: InputDecoration(
        filled: true,
        fillColor: myYellow,
        labelText: label,
        border: const OutlineInputBorder(
            borderRadius: BorderRadius.all(Radius.circular(15))), // OutlineInputBorder
      ), // InputDecoration
      onChanged: callback,
    ), // TextField
  ); // Padding
}
```

Finalizing registration form (part II)

When creating the column of our form, we call the method for each line:

```
body: Column(
  mainAxisAlignment: MainAxisAlignment.start,
  children: [
    const SizedBox(height: 30),
    getRegistrationLine("Name", setName),
    getRegistrationLine("Address", setAddress),
    getRegistrationLine("email", setMail),
    const SizedBox(height: 30),
    ElevatedButton(
```

```
void setName(String p1) {
  setState(() {
    name = p1;
  });
}
```

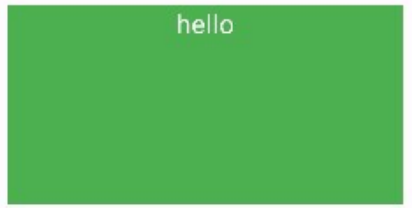


Some more info on Container Widget

```
(new) Container Container({
  Key? key,
  AlignmentGeometry? alignment,
  EdgeInsetsGeometry? padding,
  Color? color,
  Decoration? decoration,
  Decoration? foregroundDecoration,
  double? width,
  double? height,
  BoxConstraints? constraints,
  EdgeInsetsGeometry? margin,
  Matrix4? transform,
  AlignmentGeometry? transformAlignment,
  Widget? child,
  Clip clipBehavior = Clip.none,
})
```

Alignment in Container

```
Container(
    alignment: Alignment.topCenter,
    width: 300,
    height: 100,
    color: Colors.green,
    child: const Text("hello",
        style: TextStyle(color: Colors.white, fontSize: 20))),
```

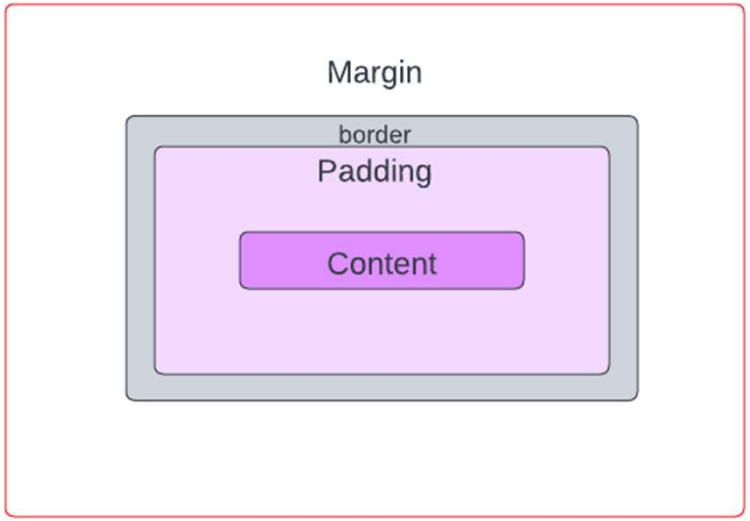



```
bottomCenter Alignment
bottomLeft Alignment
bottomRight Alignment
center Alignment
centerLeft Alignment
centerRight Alignment
topCenter Alignment
topLeft Alignment
topRight Alignment
```

Container with border

```
Container(
    //width: 300,
    //height: 100,
    margin: const EdgeInsets.fromLTRB(70, 20, 70, 20),
    padding: const EdgeInsets.fromLTRB(50, 20, 50, 20),
    // you cannot define both color and decoration !
    //color: Colors.green,
    decoration: BoxDecoration(
        color: Colors.green,
        border: Border.all(
            color: Colors.red, width: 10, style: BorderStyle.solid), // Border.all
        borderRadius: const BorderRadius.all(Radius.circular(20))), // BoxDecoration
    child: Container(height: 50, color: Colors.orange)), // Container
```

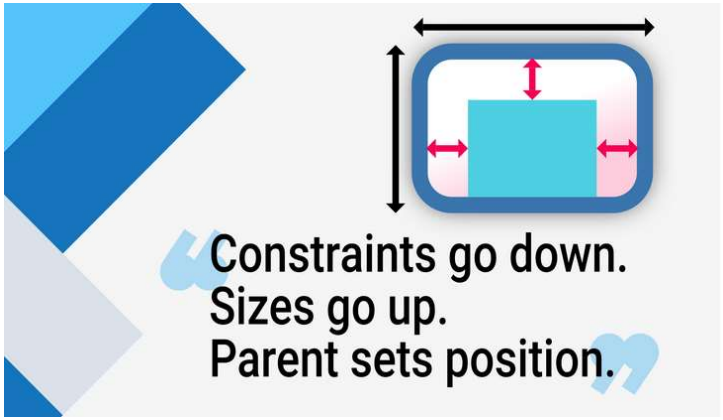
Container: box model




```dart
Container(
    // width: 300,
    // height: 200,
    color: Colors.green,
    child: Container(
      margin: EdgeInsets.all(30),
      padding: EdgeInsets.fromLTRB(50, 30, 50, 20),
      //alignment: Alignment.center,
      color: Colors.blue,
      child: Text("hello world",
          style: TextStyle(backgroundColor: Colors.red)), // Text
    )) // Container // Container
```

The picture of “box-model” was taken from https://blog.logrocket.com/flutter-layouts-guide-margins-padding/

Understanding Flutter layout algorithm



Flutter layout can't really be understood without knowing this rule, so Flutter developers should learn it early on.

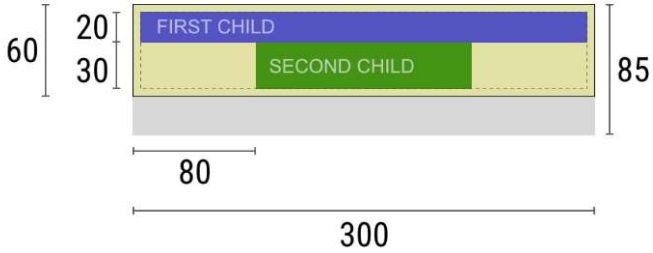
In more detail:

- A widget gets its own **constraints** from its **parent**. A *constraint* is just a set of 4 doubles: a minimum and maximum width, and a minimum and maximum height.
- Then the widget goes through its own list of **children**. One by one, the widget tells its children what their **constraints** are (which can be different for each child), and then asks each child what size it wants to be.
- Then, the widget positions its **children** (horizontally in the `x` axis, and vertically in the `y` axis), one by one.
- And, finally, the widget tells its parent about its own **size** (within the original constraints, of course).

This was copied from https://docs.flutter.dev/ui/layout/constraints

Example for the Flutter layout algorithm

For example, if a composed widget contains a column with some padding, and wants to lay out its two children as follows:



The negotiation goes something like this:

Widget: "Hey parent, what are my constraints?"

Parent: "You must be from `0` to `300` pixels wide, and `0` to `85` tall."

Widget: "Hmmm, since I want to have `5` pixels of padding, then my children can have at most `290` pixels of width and `75` pixels of height."

Widget: "Hey first child, You must be from `0` to `290` pixels wide, and `0` to `75` tall."

First child: "OK, then I wish to be `290` pixels wide, and `20` pixels tall."

Widget: "Hmmm, since I want to put my second child below the first one, this leaves only `55` pixels of height for my second child."

Widget: "Hey second child, You must be from `0` to `290` wide, and `0` to `55` tall."

Second child: "OK, I wish to be `140` pixels wide, and `30` pixels tall."

Widget: "Very well. My first child has position `x: 5` and `y: 5`, and my second child has `x: 80` and `y: 25`."

Widget: "Hey parent, I've decided that my size is going to be `300` pixels wide, and `60` pixels tall."

Also copied from https://docs.flutter.dev/ui/layout/constraints

Gradient colors

```
Container(
    alignment: Alignment.topCenter,
    width: 300,
    height: 200,
    decoration: const BoxDecoration(
        //color: Colors.green,
        gradient: LinearGradient(colors: [Colors.yellow, Colors.blue]),
    ), // BoxDecoration
    child: null), // Container
```

```
gradient: LinearGradient(
    colors: [Colors.yellow, Colors.blue], stops: [0.3, 0.7]),
```

```
gradient: LinearGradient(
    colors: [Colors.yellow, Colors.blue], stops: [0.5, 0.5]),
```

More Gradients

```
gradient: LinearGradient(
    colors: [Colors.yellow, Colors.blue], stops: [0.3, 0.7],
    begin: Alignment.topLeft, end: Alignment.bottomRight), // LinearGradient
```

```
gradient: RadialGradient(
  colors: [Colors.yellow, Colors.blue], stops: [0.3, 0.7]),
```

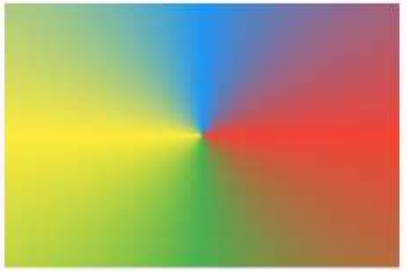
```
gradient: RadialGradient(
    center: Alignment.topRight,
    colors: [Colors.yellow, Colors.blue], stops: [0.3, 0.7]),
```

Sweep Gradients

```
gradient: SweepGradient(
  colors: [Colors.yellow, Colors.blue], stops: [0.3, 0.7]),
```

```
gradient: SweepGradient(
    startAngle: pi * 0.5,
    endAngle: pi,
    colors: [Colors.yellow, Colors.blue], stops: [0.3, 0.7]),
```

```
gradient: SweepGradient(
    colors: [Colors.red, Colors.green,
             Colors.yellow, Colors.blue, Colors.red],
    stops: [0.0, 0.25, 0.5, 0.75, 1]) // SweepGradient
```



Gradient colors in Powerpoint