3 Everything is widgets

Now that we know what widgets are and why we're using them, let's create the ones that will form the basis for our Dinner-And-A-Movie app.

1. Before we get down to the business of creating our widgets, briefly review the requirements:
    - import 'package:flutter/material.dart'; at the top of each widget file.
    - Both types of classes inherit from their supertypes; StatelessWidget and StatefulWidget
    - StatefulWidgets have two classes, a State and the widget itself.

Alright, with these things in mind, let's create some widgets.

2. Create a stateful widget called Landing in lib/landing.dart. Don't forget to create both classes that make up a stateful widget.

This widget will serve as a "page"; it will take up the entire screen, support navigation, and include sub-widgets. These types of widgets really need a Scaffold.

3. Give it a Scaffold() and a Text() with the name of the widget. Something like this should work:

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Text("Landing"),
  );
}
```

    We'll see later in the course that a Scaffold renders an AppBar at the top, making it a really nice top-level scene widget.

4. Let's display it by putting it in main.dart as the *home* property of MyApp's MaterialApp widget.

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Dinner and a Movie',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
      useMaterial3: true,
    ),
    home: const Landing(),   // <-- Change this
  );
}
```

5. Run and test, making sure you can see Landing.

Composing widgets

All the widgets we create will be made up of smaller widgets. Said differently, we will *compose* our widgets. Let's put some widgets inside of other, larger widgets.

6. Create two widgets as stateless widgets, FilmBrief and DatePicker.

7. Create another stateful widget called ShowingTimes.

8. Edit your Landing widget. Change its build method to this:

```
@override
Widget build(BuildContext context) {
  return Column(children: [
    Text("Landing"),
    DatePicker(),
    FilmBrief(),
  ]);
}
```

   Hint:
   - That Column() allows us to have more than one widget inside another. (More about Columns later).

9. Now do a composition with fewer details. Put an instance of ShowingTimes inside FilmBrief.

10. Run and test. Make sure you can see all of your widgets.

Working with widget state

A stateful widget has state, which is merely displayed data that will potentially change. When it changes, we have to re-render the widget with the new data. We'll do this with a call to setState().

11. In the _LandingState class, create a List<Film> called films.

12. Create an initState() method:

```
@override
void initState() {
  fetchFilms().then((f) => setState(() => films = f));
  super.initState();
}
```

13. In the build() method, go ahead and print(films).

14. Run and test. Make sure that you can see the printed films. If you're fast enough, notice that the widget draws itself with an empty list of films and then a split-second later it re-draws itself, but now it has six films. This is the setState() running.

Iterating to create multiple widgets

Our landing page is intended to show the user all the films they can buy tickets for. Your widget has that list of films. Let's iterate/enumerate those films and display each to the user in a FilmBrief widget. A great way of doing that is to use List.map() to convert the list of film objects into a list of FilmBrief widgets. Ready?

15. Find where you're displaying one FilmBrief. Instead of displaying just one, do this:

```
body: Column(
  children: [
    Text("Landing"),
    DatePicker(),
    Container(
      padding: EdgeInsets.all(10),
      child: Column(
        children: films.map((f) => FilmBrief()).toList(),
      ),
    ),
  ],
),
```

16. Run and test. Do you see six copies of FilmBrief()? You should.

Problem is they all look the same. How do we know which is for which film? Let's fix that by having each FilmBrief widget display the film's title. We're going to want to pass each film object into the FilmBrief widget so that it is aware of its one film.

17. First, get FilmBrief ready to receive a Film object. Change it's constructor like so:

```
class FilmBrief extends StatelessWidget {
  final Film film;                                  // <-- add this
  const FilmBrief({required this.film, super.key});  // <-- change this
```

18. Next display the film title instead of the word "FilmBrief".

```
@override
Widget build(BuildContext context) {
  return Column(
    children: [
      Text(film.title),
      ShowingTimes(),
    ],
  );
}
```

19. Lastly, pass the film down. Edit Landing and change this:

```
child: Column(
  children: films.map((f) => FilmBrief()).toList(),
),
```

to this:

```
child: Column(
  children: films.map((f) => FilmBrief(film: f,)).toList(),
),
```

20. Run and test. You should now see six film titles.

21. Bonus! If you want, you can delete the MyHomePage widget in main.dart that was created initially. We won't need that any more.