W5-S2 *PRACTICE*

LISTS, PASS FUNCTIONS, LIFT STATES UP

Learning objectives

- ✓ Generating Widgets with **'for' Loops**
- ✓ Render screens **conditionally**
- ✓ Accepting and passing functions as values
- ✓ **Lift** states up

No AI tools allowed to solve this practice

How to submit?

- ✓ **Push** your final code on **your GitHub repository**
- ✓ Then **attach the GitHub path** to the MS Team assignment and **turn it in**

Before practice, to be prepared!

Read the following documentation to be ready for this practice:

Render widgets conditionally
Lifting state up
Work with outlined buttons

EX 1 – 3 ways to create widgets with loops

Let's explore 3 ways to **create Widgets with Loops** in Flutter!

1. **Integrating the Loop Directly into the List**
   In this approach, we integrate the loop within the array itself.
   It's done using the for-loop syntax inside square bracket!

```
Column(
  children: [
    Text("Start"),
    for (var i = 0; i < 10; i++) Text('Item $i'),
    Text("End"),
  ],
)
```

2. **Using the map() Method**
   The map() method applies a function to each element of a list, transforming it into another form

```
List<String> numbers = ["MON", "TUE", "WED", "THU"]

Column (
  children: [
    Text("Start"),
    ...numbers.map((item) => Text(item)).toList(),
    Text("End"),
  ],
)
```

3. **Using a Dedicated Function**
   We separate the logic of generating widgets into a dedicated function. This helps improve readability, especially when the widget creation logic is more complex.

```
List<String> numbers = ["MON", "TUE", "WED", "THU"]

List<Widget> getLabels() {
  return numbers.map((item) => Text(item)).toList();
}

Column (
  children: [
    Text("Start"),
    getLabels(),
    Text("End"),
  ],
)
```

Q1 - In what scenarios might one approach **be more advantageous** than another?
(in terms of readability, maintainability, performance, etc.)?

Q2 – Let s code a bit!

We have a list of colors

```
List<String> colors = ["red", "blue", "green"];
```

Create a Flutter app with a ListView that **displays the colors** using three different methods:

- Using a **direct for loop** within the widget list.
- Using the **map**() method on a list.
- Using a **dedicated function** to return a list of widgets.

Each method should display the numbers in a different section of the screen

<u>Note</u>: You have a start code with an already defined customer Label widget to render the text in bold.

*EX 2 – Refactor the code to **bind** with data*

In this exercise, you are provided with a **start code**.

The widgets are already created in the START CODE. But everything is static!

You need to **bind the widget** ProfileApp **with the data,** defined in the folder `/data`
The ProfileData model is defined as follows:

1. The **ProfileApp** widget must take as argument a **ProfileData** object.

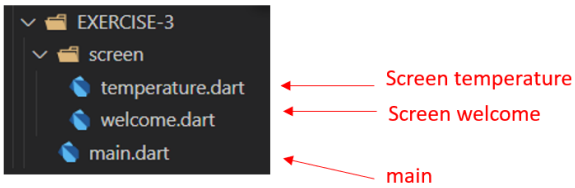
2. Each tile of the given profile data must be rendered in the profile app, as a card.
   TIP: you will need to loop to render each tile.

3. You also need to **manage the scrolling,** when too many tiles need to be displayed.

4. For fun: customize this view! *Update the layout and the data as you wish.*

*EX 3 – Switching from screen using a **state***

In this exercise, you are provided with a start code with 3 files:



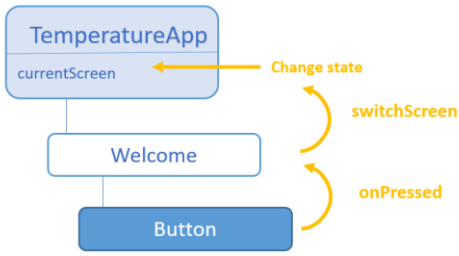
The objective is to be able to **start with the welcome screen** and to **switch to the converter screen** upon click on START:

Q1 – First create a stateful widget in the main.dart (`TemperatureApp`)

- This widget shall manage the **active screen state (**whether we are in the welcome screen or the temperature screen)
- Render conditionally the 2-screen depending on the state.

TIP: you can [read this resource]() to render widgets conditionally

Q2– Next step is to link the welcome button with a **callback to change the TemperatureApp** state



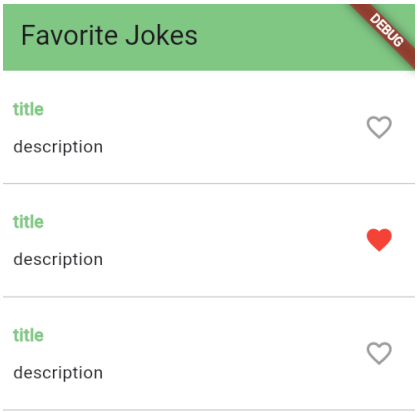
- **Create a callback** TemperatureApp to change the state
- **Pass this callback** to the Welcome view
- **Bind this callback** the button press event

*EX 4 – Manage a **list of jokes***

We **start** from the code we used to work on (W4-S3).

<u>OBJECTIVES</u>

- ✓ First you need to create a database (`jokes.dart`**)** of jokes to **populate the list of jokes**
- ✓ Then you should display at least 20 jokes, so the list **should be scrollable**
- ✓ Last but not least: **only 1 joke can be set a best joke** (heart icon)



*Only 1 joke can be **THE** favorite!*

You will need to **re-think** about your **data structure** and **widget interaction** to perform the changes:

What **type of data** will store the jokes? What **type of data** will store the favorite joke?	
Which widget should be in charge **of storing the favorite joke?** Which widget should be **stateful**?	
How will your widget interact? Do you need to **pass callback function** between widgets?	

BONUS-1 – Handle the TextField input

For this bonus exercise, you can continue on exercise 2 code and make the temperature converter working properly.

TIP

- *How to get the value entered in the input field?*
- *We suggest (for now) to use a state to handle this input field value!*