# Lab 10

Instructor: Sidra Khatoon E-mail: skhatoon[@uit.edu](mailto:ad@uit.edu)

# Objective

# The objective of lab is exploring inkwell widgets and its usages.

**Student Information**

|  |  |
| --- | --- |
| **Student Name** |  |
| **Student ID** |  |
| **Date** |  |

**Assessment**

|  |  |
| --- | --- |
| **Marks Obtained** |  |
| **Remarks** |  |
| **Signature** |  |

# Objective

# The objective of lab is exploring custom text style , widget style and theme.

# Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

# How to Submit?

Submit lab work using Teams.

**Styling Text**

There are two topics regarding the appearance of Text: TextStyle and Custom Fonts.

**Text Style:**

Text widgets have a style property which takes a TextStyle object. You’ll simply set the *style* property to an

instance of a TextStyle widget and set properties. TextStyle supports about 20 properties. Here are the most

useful:

* color – Any of the valid 16+ million colors
* decoration – TextDecoration.underline, overline,
* strikethrough, none
* fontSize – A double. The number of pixels tall to make
* the characters. Default size 14.0 pixels
* fontStyle – FontStyle.italic or normal
* fontWeight – FontWeight.w100-w900. Or bold (which is
* w700) or normal (which is w400)
* fontFamily – A string

**A screen shot of a computer code

Description automatically generated**

**Custom Font**

Certain designers call for custom fonts when they design scenes. In Flutter, using custom fonts is easy to

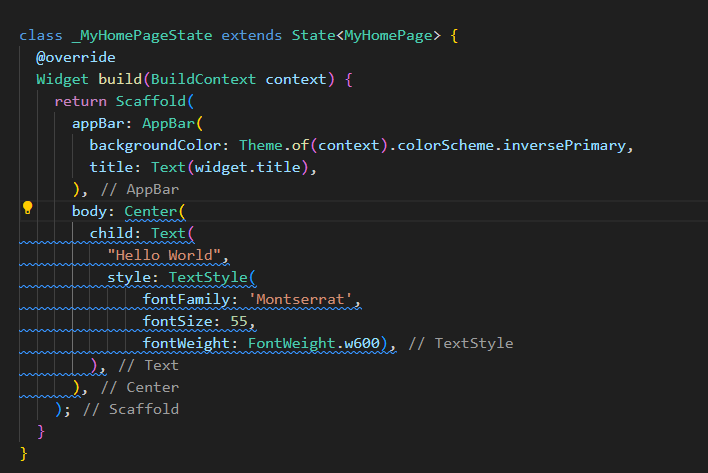
implement, and they work cross-platform. To style our text, let's use a custom font. Create a new directory

called assets/fonts. Copy the font files located at assets/fonts in the step/step03 branch of this repo and copy

it to our new directory. Update our pubspec.yaml file so we can use our fonts.

**A screen shot of a computer code

Description automatically generated**

****

**Understanding Flutter Material Design Themes**

A theme in Flutter is a collection of property-value pairs that dictate the appearance of the app’s widgets. [ThemeData](https://api.flutter.dev/flutter/material/ThemeData-class.html) is the class responsible for holding these properties. Let’s first understand the significance of ThemeData and how it helps in theming.

[**ThemeData**](https://api.flutter.dev/flutter/material/ThemeData-class.html)

The ThemeData class encapsulates a Material Design theme’s colors, typography, and shape properties. We typically use it

as an argument for the [MaterialApp](https://api.flutter.dev/flutter/material/MaterialApp-class.html) widget, which in turn applies the theme to all descendant widgets.

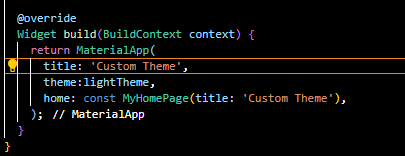
**Creating a Custom Theme**

Create a ThemeData instance and assign values to the properties you wish to customize. Let’s create a custom theme and apply it to our Flutter app. Just modify the existing default app there.



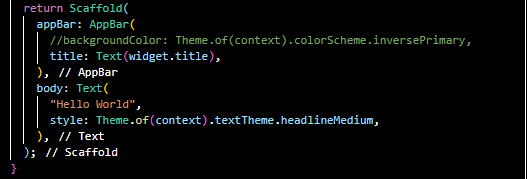
**Applying the ThemeData Instance**

Pass the custom theme to the theme property of the MaterialApp widget.



**Using the Theme Properties**

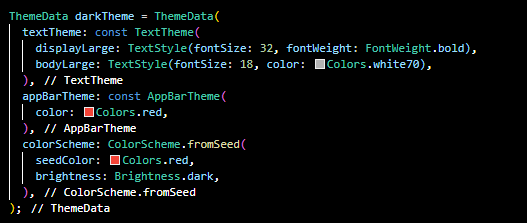
Access the ThemeData properties with the [Theme.of(context)](https://api.flutter.dev/flutter/material/Theme/of.html) method. Here’s an example of how to use the primary color and text theme in a Text widget:



**Dark and Light Themes**

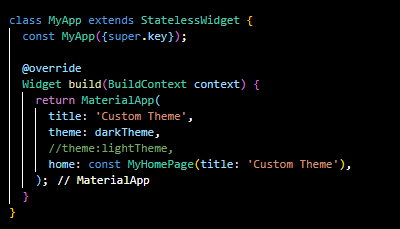
Flutter also allows you to define separate themes for dark and light modes. You can set the darkTheme

Property of the MaterialApp widget. Make sure you set the ColorScheme’s brightness property to Brightness.dark to indicate it’s a dark theme.



**Applying Dark ThemeData Instance**

This tells Flutter that there are two themes: light and dark. Flutter automatically switches between the two themes based on the device’s brightness settings.

****

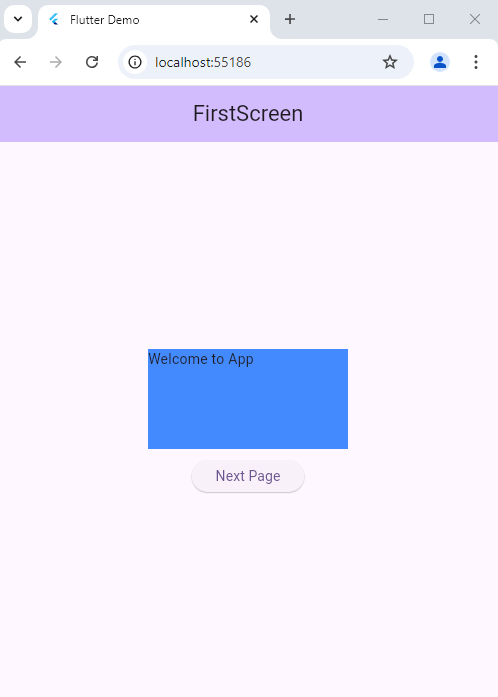
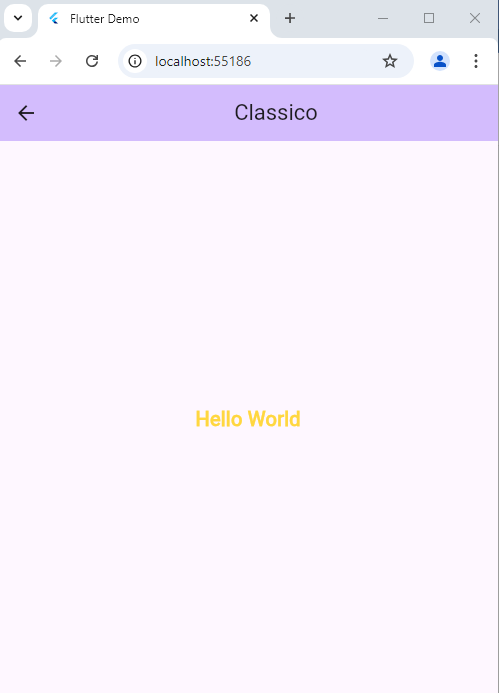
**Flutter – Navigate from One Screen to Another**

Flutter apps may have multiple screens or pages. Pages are groups of functionalities. The user navigates between different pages to use different functionalities. Concepts like pages are called routes in Flutter. We can use Navigator.push() to navigate to a new route and Navigator.pop() to navigate to the previous route. Routes are managed by the Navigator widget. The navigator manages a stack of routes. Routes can be pushed on the stack using push() method and popped off the stack using pop() method. The top element in the stack is the currently active route. Navigator is a stateful widget with NavigatorState as its state. In this article, we will see how to navigate from one screen to another screen in Flutter.

**How to use:**

Navigator class has a push method to Navigate to the next screen.



**Assessment**

**Task:** You are required to build a simple Flutter app with **two screens**:

* **Home Screen**: The main screen of the app with buttons to navigate to Student List Screen. In this screen you have display your Batch name and Semester by using new font. Import font manually.
* **Student List Screen**: Displays a list of students by using ListView in this Screen. List contain Name in title and registration number in subtitle.