Flutter: Data Persistent:

What?

It’s really hard to use an app if you have to login every time you use it. Also, it is difficult if you have to reenter your data every time you use it. Thus you have to really be determined, or rather *persistent*, in using this particular app that always forgets you and your data. :]

In order to avoid frustrating users, your app needs to save data that survives an app restart. Saving app data to some type of storage that survives app restarts is called **data persistence**.

Why?

Most real-world apps will need to save some data that is referenced again and again — every time the app is used. Mobile OS, such as iOS and Android, manage the lifecycle of each running App — when an App goes into the background it may be forced to quit any time before the user next opens it. Regardless of whether the App was quit in between times, the next time the user opens the app it should behave and function the same (apart from some session-based apps, e.g. mobile-banking) as where they left off. An app may still be running from the last time it was opened, or it may need to open-from-scratch and any “state” may have to be reloaded. The App-code (yes, the code that we as developers write) needs to take care of this without the users having to concern themselves with whether that data has been appropriately stored for later access.

How?

Flutter offers us several ways we can achieve persistent data storage for our mobile Apps, including:

- **Key-Value Store(Shared-Preferences)**
- **File storage**
- **Local database using SQLite**
- **Remote datastore**

Data Persistence with Shared Preferences:

There's multiple ways to store data in disk. For instance, one of them is using a key-value store. Accordingly, iOS and Android has native solutions for doing key-value storage in disk. Specifically, iOS has UserDefaults and Android has SharedPreferences. Instead of manually using each of them, you can use an existing Flutter package called shared_preferences that uses the appropriate one depending on the platform on which you are running.

The most important thing you should remember while using shared_preferences is **SMALL DATA**. Do not use it for storing large data or complicated data, it will make your code super complicated because of reading and writing this data from shared_preferences.

I can list down here some use cases that shared_preferences is useful:

- User changes view mode (night mode).
- Changing the app's language.
- Select the app theme.

This recipe uses the following steps:

1. Add the dependency.
2. Save data.
3. Read data.
4. Remove data.

1. Add the dependency

Before starting, add the `shared_preferences` plugin to the `pubspec.yaml` file:

```
dependencies:
  flutter:
    sdk: flutter
  shared_preferences: "<newest version>"
```

2. Save data

To persist data, use the setter methods provided by the `SharedPreferences` class. Setter methods are available for various primitive types, such as `setInt`, `setBool`, and `setString`.

Setter methods do two things: First, synchronously update the key-value pair in-memory. Then, persist the data to disk.

```
// obtain shared preferences
final prefs = await SharedPreferences.getInstance();

// set value
prefs.setInt('counter', counter);
```

3. Read data

To read data, use the appropriate getter method provided by the `SharedPreferences` class. For each setter there is a corresponding getter. For example, you can use the `getInt`, `getBool`, and `getString` methods.

```
final prefs = await SharedPreferences.getInstance();

// Try reading data from the counter key. If it doesn't exist, return 0.
final counter = prefs.getInt('counter') ?? 0;
```

4. Remove data

To delete data, use the `remove()` method.

```
final prefs = await SharedPreferences.getInstance();

prefs.remove('counter');
```

Supported types

Although key-value storage is easy and convenient to use, it has limitations:

- Only primitive types can be used: `int`, `double`, `bool`, `string`, and `stringList`.
- It's not designed to store a lot of data.

For more information about shared preferences on Android, see the shared preferences documentation on the Android developers website.

Complete example

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';

void main() => runApp(const MyApp());
```

```dart
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  // This widget is the root of the application.
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Shared preferences demo',
      home: MyHomePage(title: 'Shared preferences demo'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  @override
  void initState() {
    super.initState();
    _loadCounter();
  }

  //Loading counter value on start
  void _loadCounter() async {
    final prefs = await SharedPreferences.getInstance();
    setState(() {
      _counter = (prefs.getInt('counter') ?? 0);
    });
  }

  //Incrementing counter after click
  void _incrementCounter() async {
    final prefs = await SharedPreferences.getInstance();
    setState(() {
      _counter = (prefs.getInt('counter') ?? 0) + 1;
      prefs.setInt('counter', _counter);
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headline4,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build
methods.
    );
  }
}
```