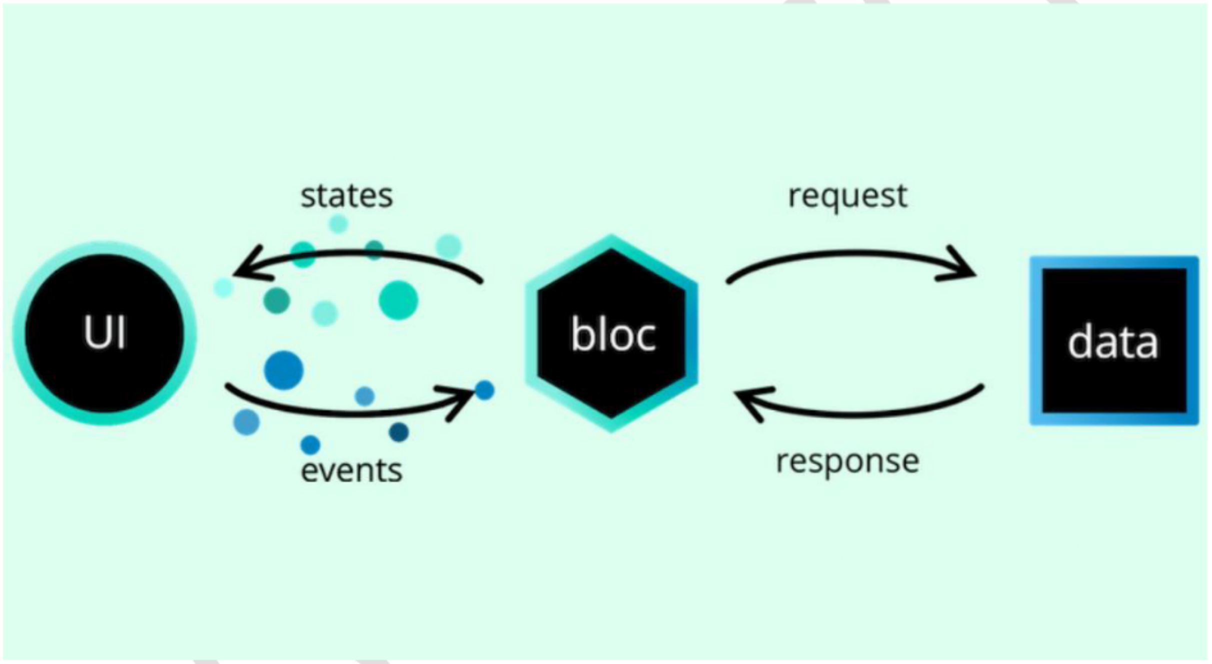
Bloc:-

Bloc is a design pattern created by Google to help separate business logic from the presentation layer and enable a developer to reuse code more efficiently.

A state management library called Bloc was created and maintained by Felix Angelo. It helps developers implement the Bloc design pattern in their Flutter application. It means that a developer must know the state of an app at any time. There should be something displayed on the screen for every interaction with the app to let users know what is happening.

Technically, for every interaction inside the application, there should be a state emerging from it. For example, when the data is fetching, the app should be in a loading state displaying a loading animation on the screen. When the internet is off, the application should display a pop-up to let the user know there is no internet connection.



Example:-

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

import 'package:flutter_bloc/flutter_bloc.dart';


void main() {
  Bloc.observer = AppBlocObserver();
  runApp(const App());
}

/// Custom [BlocObserver] that observes all bloc and cubit state changes.
class AppBlocObserver extends BlocObserver {
  @override
  void onChange(BlocBase bloc, Change change) {
    super.onChange(bloc, change);
    if (bloc is Cubit) print(change);
  }

  @override
  void onTransition(Bloc bloc, Transition transition) {
    super.onTransition(bloc, transition);
    print(transition);
  }
}

/// {@template app}
/// A [StatelessWidget] that:
/// * uses [bloc](https://pub.dev/packages/bloc) and
/// [flutter_bloc](https://pub.dev/packages/flutter_bloc)
/// to manage the state of a counter and the app theme.
/// {@endtemplate}
class App extends StatelessWidget {
  /// {@macro app}
  const App({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (_) => ThemeCubit(),
      child: const AppView(),
    );
  }
}

/// {@template app_view}
/// A [StatelessWidget] that:
/// * reacts to state changes in the [ThemeCubit]
/// and updates the theme of the [MaterialApp].
/// * renders the [CounterPage].
/// {@endtemplate}
class AppView extends StatelessWidget {
  /// {@macro app_view}
  const AppView({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<ThemeCubit, ThemeData>(
      builder: (_, theme) {
        return MaterialApp(
          theme: theme,
          home: const CounterPage(),
        );
      },
    );
  }
}
```

```
/// {@template counter_page}
/// A [StatelessWidget] that:
/// * provides a [CounterBloc] to the [CounterView].
/// {@endtemplate}
class CounterPage extends StatelessWidget {
 /// {@macro counter_page}
 const CounterPage({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
  return BlocProvider(
   create: (_) => CounterBloc(),
   child: const CounterView(),
  );
 }
}

/// {@template counter_view}
/// A [StatelessWidget] that:
/// * demonstrates how to consume and interact with a [CounterBloc].
/// {@endtemplate}
class CounterView extends StatelessWidget {
 /// {@macro counter_view}
 const CounterView({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: const Text('Counter')),
body: Center(
  child: BlocBuilder<CounterBloc, int>(
    builder: (context, count) {
      return Text('$count', style: Theme.of(context).textTheme.headline1);
    },
  ),
),
floatingActionButton: Column(
  crossAxisAlignment: CrossAxisAlignment.end,
  mainAxisAlignment: MainAxisAlignment.end,
  children: <Widget>[
    FloatingActionButton(
      child: const Icon(Icons.add),
      onPressed: () {
        context.read<CounterBloc>().add(CounterIncrementPressed());
      },
    ),
    const SizedBox(height: 4),
    FloatingActionButton(
      child: const Icon(Icons.remove),
      onPressed: () {
        context.read<CounterBloc>().add(CounterDecrementPressed());
      },
    ),
    const SizedBox(height: 4),
    FloatingActionButton(
      child: const Icon(Icons.brightness_6),
      onPressed: () {
        context.read<ThemeCubit>().toggleTheme();
      },
    ),
      ],
    ),
  );
 }
}

/// Event being processed by [CounterBloc].
abstract class CounterEvent {}

/// Notifies bloc to increment state.
class CounterIncrementPressed extends CounterEvent {}

/// Notifies bloc to decrement state.
class CounterDecrementPressed extends CounterEvent {}

/// {@template counter_bloc}
/// A simple [Bloc] that manages an `int` as its state.
/// {@endtemplate}
class CounterBloc extends Bloc<CounterEvent, int> {
 /// {@macro counter_bloc}
 CounterBloc() : super(0) {
  on<CounterIncrementPressed>((event, emit) => emit(state + 1));
  on<CounterDecrementPressed>((event, emit) => emit(state - 1));
 }
}

/// {@template brightness_cubit}
/// A simple [Cubit] that manages the [ThemeData] as its state.
/// {@endtemplate}
class ThemeCubit extends Cubit<ThemeData> {
  /// {@macro brightness_cubit}
  ThemeCubit() : super(_lightTheme);

  static final _lightTheme = ThemeData(
    floatingActionButtonTheme: const FloatingActionButtonThemeData(
      foregroundColor: Colors.white,
    ),
    brightness: Brightness.light,
  );

  static final _darkTheme = ThemeData(
    floatingActionButtonTheme: const FloatingActionButtonThemeData(
      foregroundColor: Colors.black,
    ),
    brightness: Brightness.dark,
  );

  /// Toggles the current brightness between light and dark.
  void toggleTheme() {
    emit(state.brightness == Brightness.dark ? _lightTheme : _darkTheme);
  }
}
```