nimble

The Flutter Framework

Luong

Growth Session #23 - March 21-22 2019

Goals

- Understand the fundamentals of the Flutter framework
- Incorporate widgets and state into our apps
- Use Flutter's tools to enhance our development process

Why Flutter?

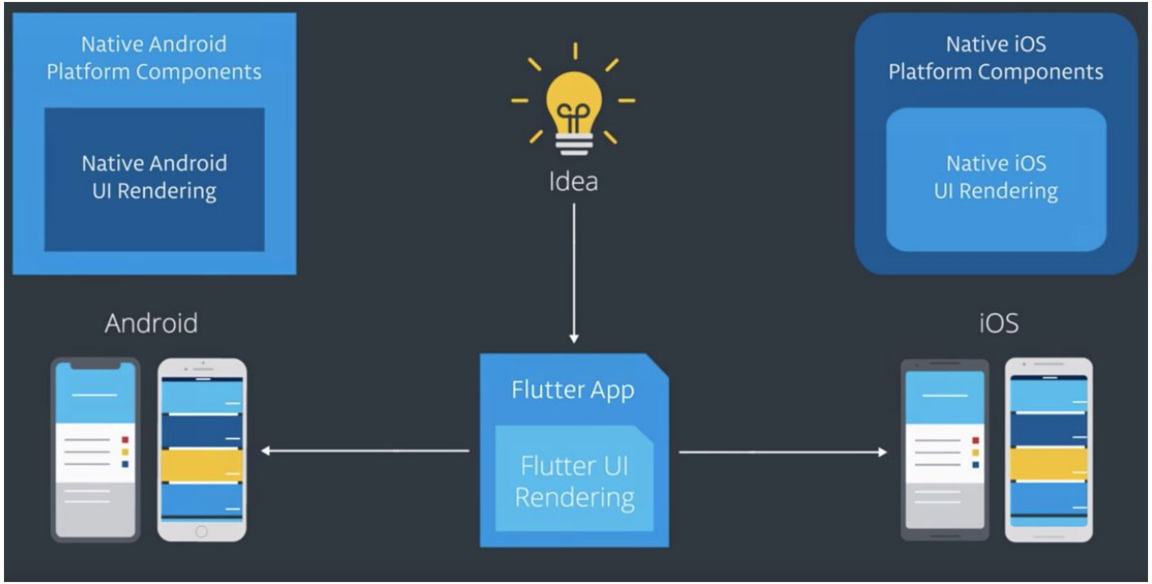
- Fast Development
  - Stateful Hot Reload.
  - Fully-customizable widgets to build native interfaces.
- Expressive and Flexible UI
  - Focus on native end-user experiences.
  - Layered architecture allows for full customization, which results in incredibly fast rendering and expressive and flexible designs.
- Native Performance
  - Flutter’s widgets incorporate all critical platform differences such as scrolling, navigation, icons and fonts to provide full native performance on both iOS and Android.

Why Dart?

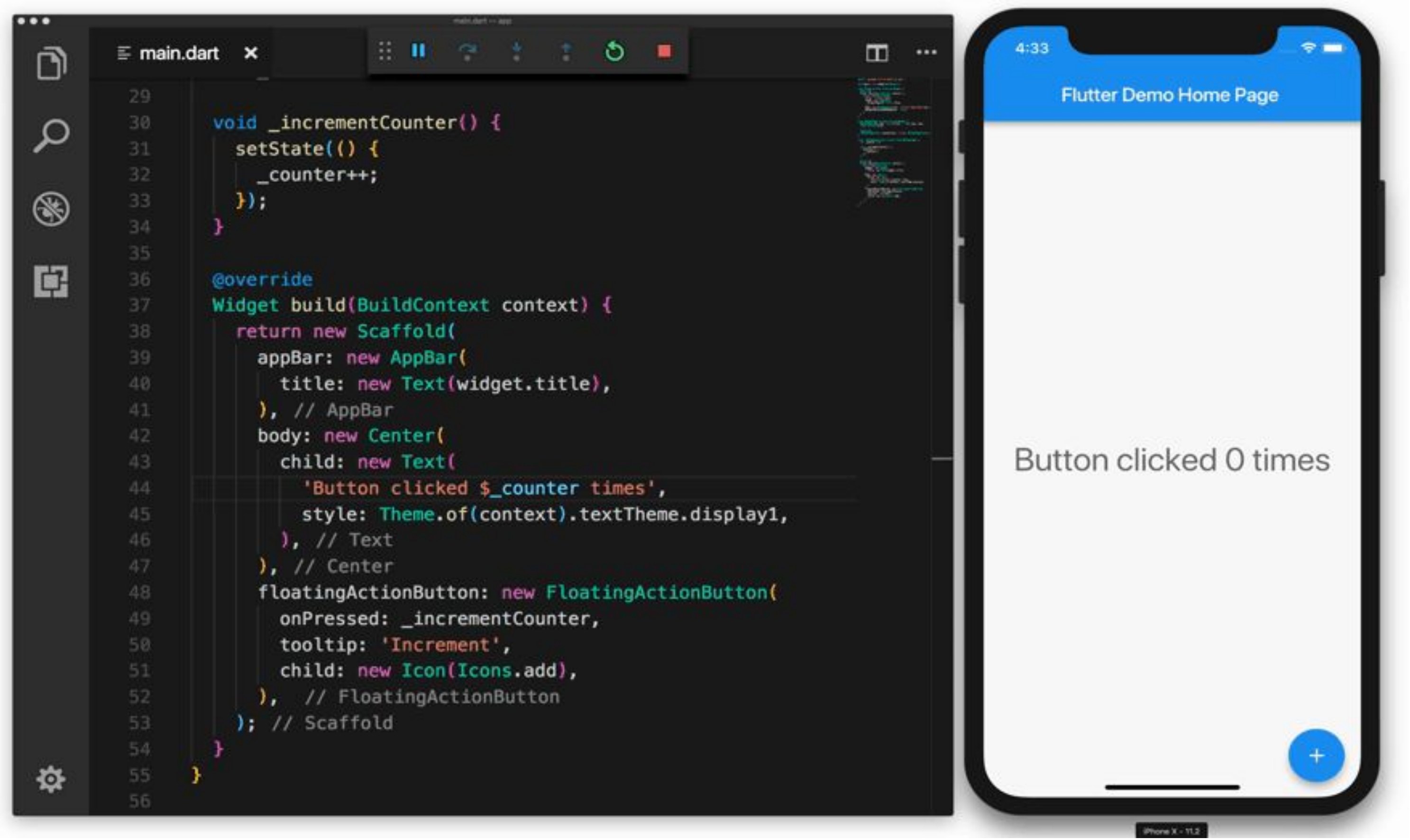
- Flutter is written in Dart
- A terse, strong typed, OOP language
- High performance in development and in production
- Just-in-time
    - Code is continuously recompiled directly on the device, allowing hot reload
- Ahead-of-time
    - The code your app uses is compiled directly to native ARM code, leading to fast startup and predictable performance
- How about Kotlin or Swift?

Flutter Framework

- Modern Reactive Framework
- Fast 2D rendering engine
- Nearly everything is a widget


Native Android Platform Components
Native Android UI Rendering
Idea
Native iOS Platform Components
Native iOS UI Rendering
Android
iOS
Flutter App
Flutter UI Rendering

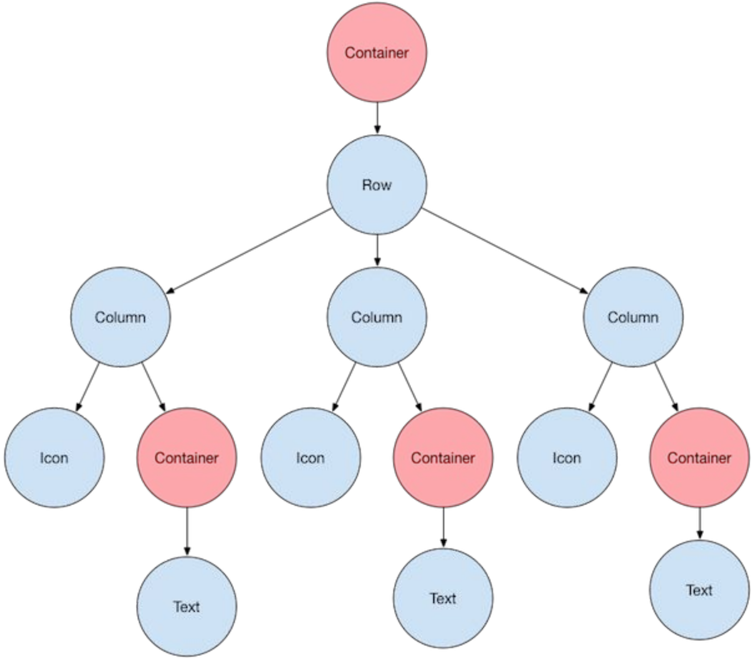
Flutter Framework



```dart
  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text(widget.title),
      ), // AppBar
      body: new Center(
        child: new Text(
          'Button clicked $_counter times',
          style: Theme.of(context).textTheme.display1,
        ), // Text
      ), // Center
      floatingActionButton: new FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: new Icon(Icons.add),
      ),  // FloatingActionButton
    ); // Scaffold
  }
}
```

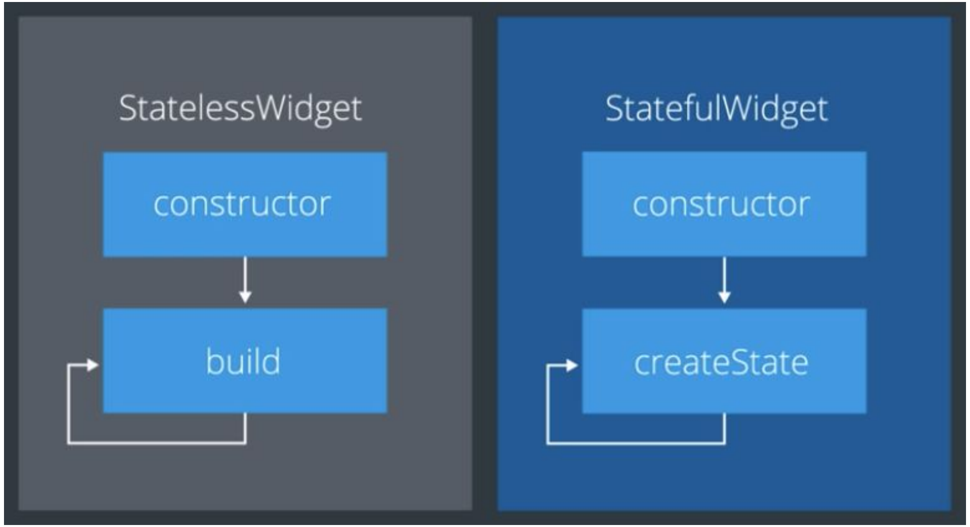
Widget

- Part of User Interface like View (Android), UIView (iOS)
- Container, Row/Column, Text, Icon, IconButton, ListView, GridView...

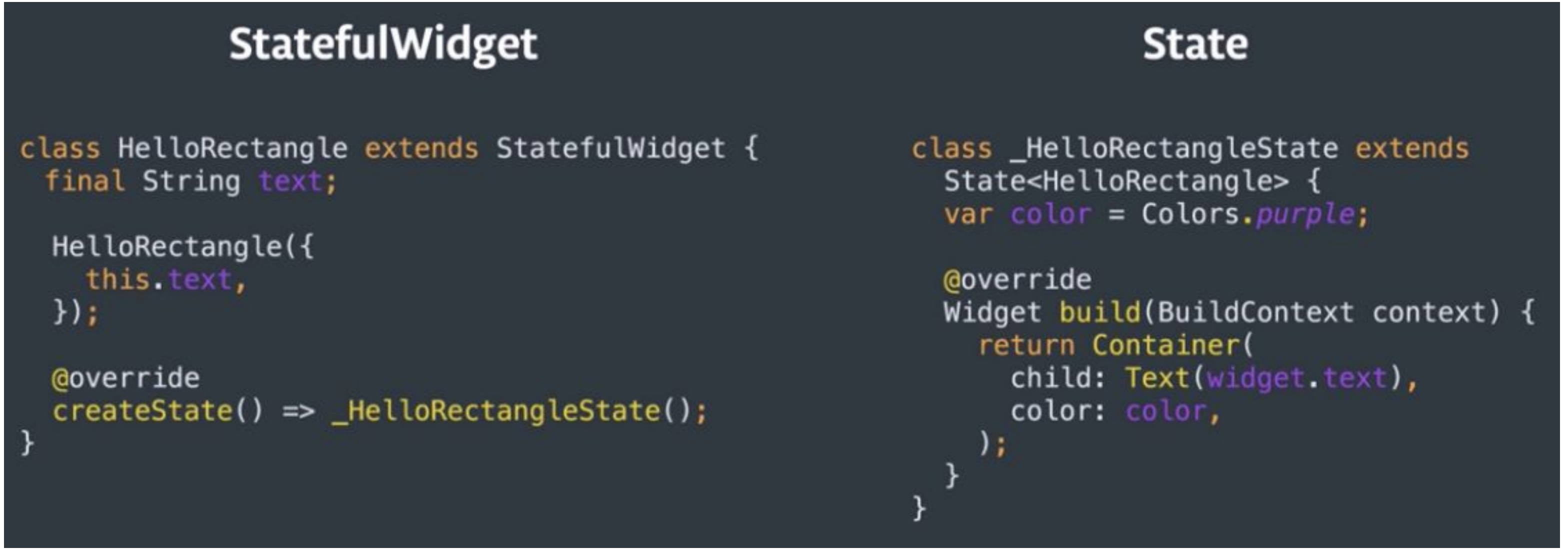


Widget

StatelessWidget & StatefulWidget



Widget



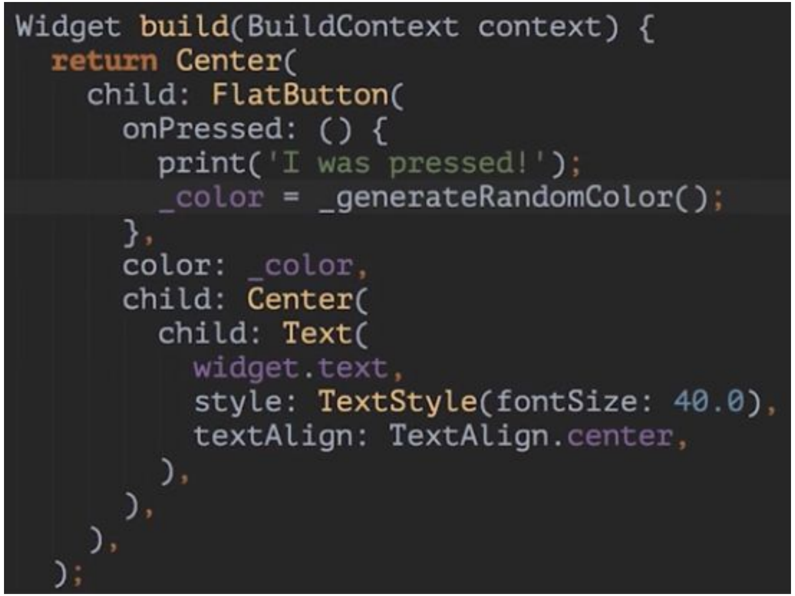
StatefulWidget

```
class HelloRectangle extends StatefulWidget {
  final String text;

  HelloRectangle({
    this.text,
  });

  @override
  createState() => _HelloRectangleState();
}
```

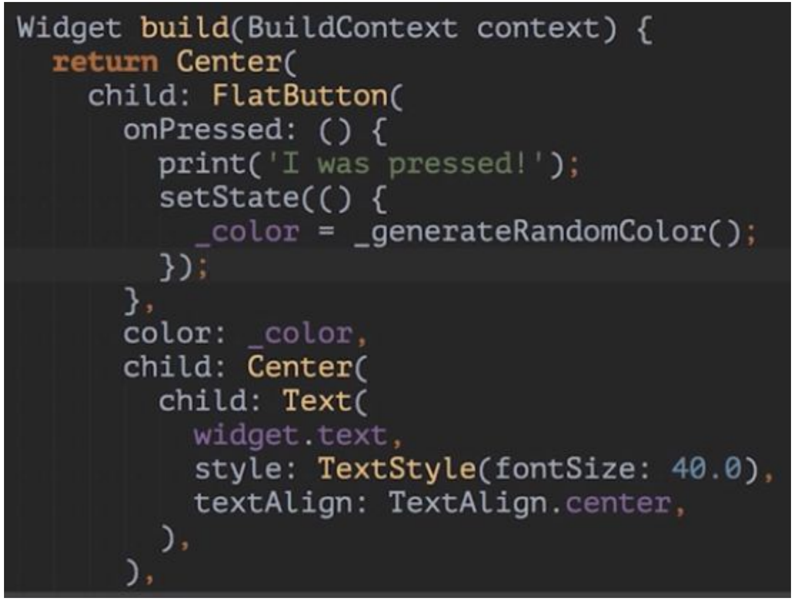
State

```
class _HelloRectangleState extends
  State<HelloRectangle> {
  var color = Colors.purple;

  @override
  Widget build(BuildContext context) {
    return Container(
      child: Text(widget.text),
      color: color,
    );
  }
}
```

Widget



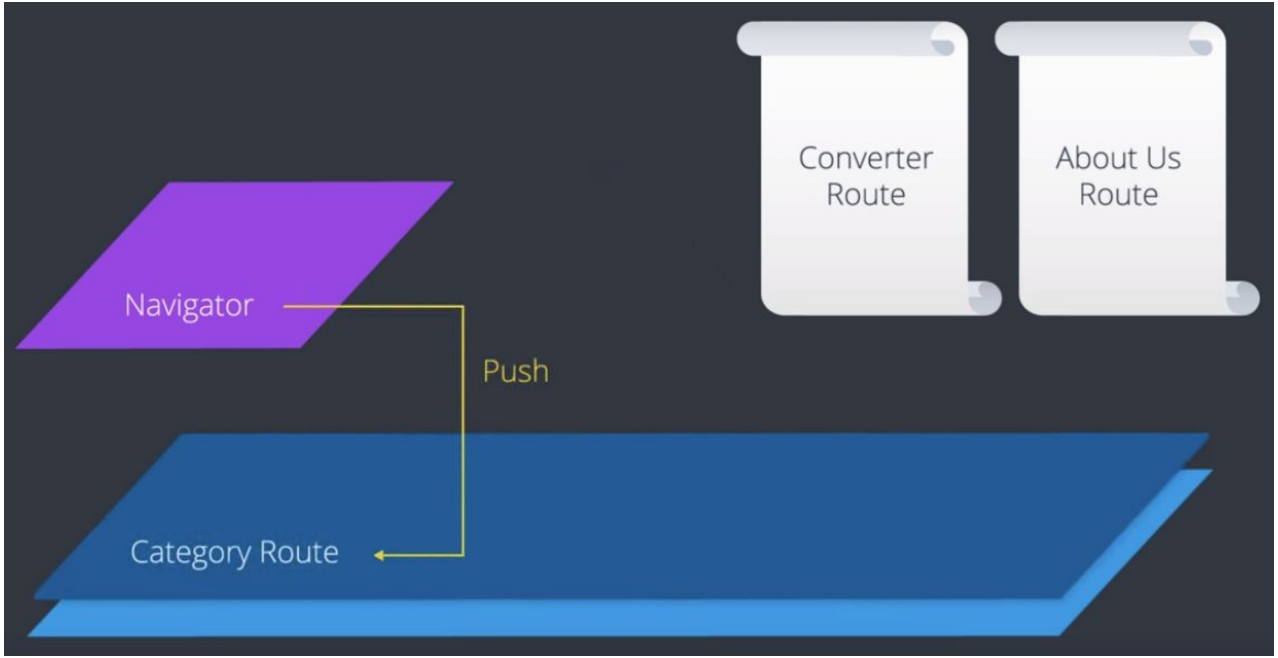
```
Widget build(BuildContext context) {
  return Center(
    child: FlatButton(
      onPressed: () {
        print('I was pressed!');
        _color = _generateRandomColor();
      },
      color: _color,
      child: Center(
        child: Text(
          widget.text,
          style: TextStyle(fontSize: 40.0),
          textAlign: TextAlign.center,
        ),
      ),
    ),
  );
```



```
Widget build(BuildContext context) {
  return Center(
    child: FlatButton(
      onPressed: () {
        print('I was pressed!');
        setState(() {
          _color = _generateRandomColor();
        });
      },
      color: _color,
      child: Center(
        child: Text(
          widget.text,
          style: TextStyle(fontSize: 40.0),
          textAlign: TextAlign.center,
        ),
      ),
```

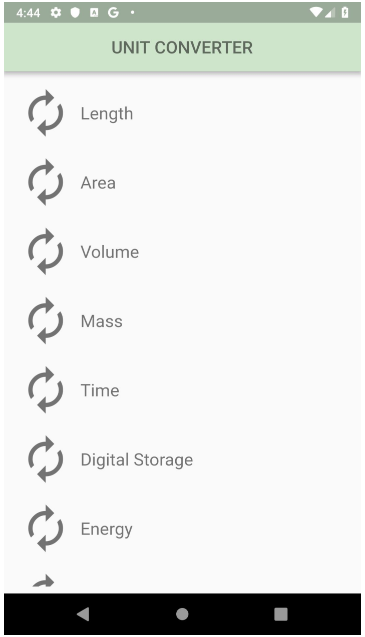
Navigator & Route

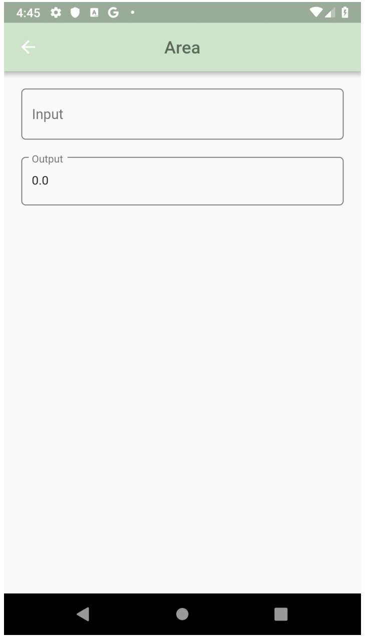
- A route takes you to a page, or screen
- Supports **Push** & **Pop**



Achievement and Progress

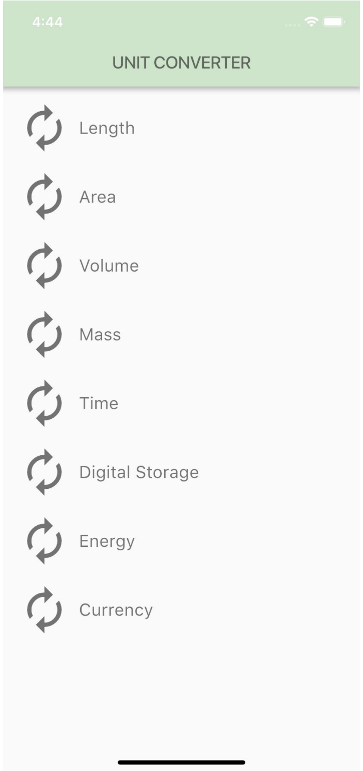
Small demonstration with a Unit Converter app

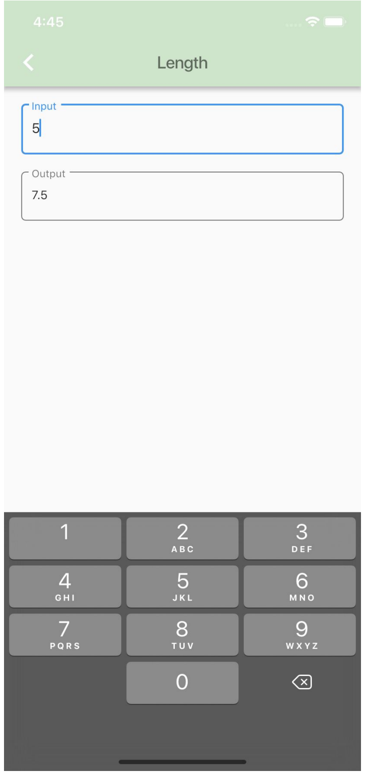




Input

Output
0.0





Input
5

Output
7.5

Achievement and Progress

- Gain better understanding about the Flutter framework
- Start building apps with Flutter
- Spread into iOS development

Next Steps

- Building interactive apps
- Customize with Material or Cupertino: themes, assets
- Make app interactive, works with local or remote (API) data sources
- Implement complex widget and UI
- Localization
- Multiple build environments
- External SDK implementation
- Recall code from Kotlin or Swift

Flutter is quick, expressive and performant

Thanks!

Contact Nimble

nimblehq.co
hello@nimblehq.co

Bangkok

399 Interchange 21 Sukhumvit Road, Unit
#2402-03, Klong Toei, Wattana, Bangkok
10110, Thailand

Singapore

28C Stanley St, Singapore 068737

Hong Kong

20th Floor, Central Tower
28 Queen's Road, Central, Hong Kong

nimble