Init

```
flutter create my_project
```

Specify organisation name

```
flutter create --org com.yourorg your_project
```

Healthcheck

```
flutter doctor
```

Hello World

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


void main() {
    runApp(MyApp());
}


class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Hello world!',
      home: Scaffold(
        body: Center(
          child: Text('Hello world'),
        ),
      ),
```

```
    );
  }
}
```

Stateless Widget

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


class Greeter extends StatelessWidget {
  Greeter({Key key @required this.name}) : super(key: key);


  final String name;


  @override
  Widget build(BuildContext context) {
    return Container(
      child: Text('Hello, $name'),
    );
  }
}
```

Required and default props

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


class SomeComponent extends StatelessWidget {
  SomeComponent({
    @required this.foo,
```

```
    this.bar = 'some string',
  });


  final String foo;
  final String bar;


  @override
  Widget build(BuildContext context) {
    return Container(
      child: Text('$foo $bar'),
    );
  }
}
```

Stateful Widget

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


class WidgetWithState extends StatefulWidget {
  @override
  _WidgetWithStateState createState() =>
_WidgetWithStateState();
}


class _WidgetWithStateState extends State<WidgetWithState> {
  int counter = 0;
```

```
  increment() {
    setState(() {
      counter++;
    });
  }

  decrement() {
    setState(() {
      counter--;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Row(
      children: <Widget>[
        FlatButton(onPressed: increment, child:
Text('Increment')),
        FlatButton(onPressed: decrement, child:
Text('Decrement')),
        Text(counter.toString()),
      ],
    );
  }
```

```
}
```

Combining props and state

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


class SomeWidget extends StatefulWidget {
  SomeWidget({@required this.fruit});


  final String fruit;


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


class _SomeWidgetState extends State<SomeWidget> {
  int count = 0;


  @override
  Widget build(BuildContext context) {
    return Container(
      child: Text('$count ${widget.fruit}'),
    );
  }
}
```

```
class ParentWidget extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Container(

        child: SomeWidget(fruit: 'oranges'),

    );

  }

}
```

Lifecycle hooks

```
class _MyComponentState extends State<MyComponent> {

  @override

  void initState() {

    // this method is called before the first build

    super.initState();

  }


  @override

  void didUpdateWidget(MyComponent oldWidget) {

    // this method IS called when parent widget is rebuilt

    super.didUpdateWidget(oldWidget);

  }


  @override didChangeDependencies() {

    // called when InheritedWidget updates
```

```
    // read more here
https://api.flutter.dev/flutter/widgets/InheritedWidget-
class.html

    super.didChangeDependencies();

  }



  @override

  void dispose() {

    // called after widget was unmounted from widget tree

    super.dispose();

  }

}
```

Android Ink effect

```
InkWell(

  child: Text('Button'),

  onTap: _onTap,

  onLongPress: _onLongPress,

  onDoubleTap: _onDoubleTap,

  onTapCancel: _onTapCancel,

);
```

Detecting Gestures

```
GestureDetector(

  onTap: _onTap,

  onLongPress: _onLongPress,

  child: Text('Button'),
```

```
);
```

Loading indicator

```
class SomeWidget extends StatefulWidget {

  @override

  _SomeWidgetState createState() => _SomeWidgetState();

}



class _SomeWidgetState extends State<SomeWidget> {

  Future future;



  @override

  void initState() {

    future = Future.delayed(Duration(seconds: 1));

    super.initState();

  }



  @override

  Widget build(BuildContext context) {

    return FutureBuilder(

      future: future,

      builder: (context, snapshot) {

        return snapshot.connectionState ==
ConnectionState.done

            ? Text('Loaded')
```

```
            : CircularProgressIndicator();

      },

    );

  }

}
```

Platform specific code

```
import 'dart:io' show Platform;


if (Platform.isIOS) {

  doSmthIOSSpecific();

}


if (Platform.isAndroid) {

  doSmthAndroidSpecific();

}
```

Hide status bar

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


void main() {

    SystemChrome.setEnabledSystemUIOverlays([]);

}
```

Lock orientation

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

```
void main() async {
  await SystemChrome.setPreferredOrientations([
    DeviceOrientation.portraitUp,
  ]);

  runApp(App());
}
```

Show alert

```
showDialog<void>(
  context: context,
  barrierDismissible: false,
  builder: (BuildContext context) {
    return AlertDialog(
      title: Text('Alert Title'),
      content: Text('My Alert Msg'),
      actions: <Widget>[
        FlatButton(
          child: Text('Ask me later'),
          onPressed: () {
            print('Ask me later pressed');
            Navigator.of(context).pop();
          },
        ),
```

```
        FlatButton(
          child: Text('Cancel'),
          onPressed: () {
            print('Cancel pressed');
            Navigator.of(context).pop();
          },
        ),
        FlatButton(
          child: Text('OK'),
          onPressed: () {
            print('OK pressed');
            Navigator.of(context).pop();
          },
        ),
      ],
    );
  },
);
```

Check if dev

```
bool isDev = false;
assert(isDev == true);

if (isDev) {
    doSmth();
```

```
}
```

Navigation

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



class FirstScreen extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Center(

      child: RaisedButton(

        child: Text('Go to SecondScreen'),

        onPressed: () => Navigator.pushNamed(context,
'/second'),

      ),

    );

  }

}



class SecondScreen extends StatelessWidget {

  void _pushSecondScreen(context) {

    Navigator.push(context, MaterialPageRoute(builder:
(context) => SecondScreen()));

  }



  @override

  Widget build(BuildContext context) {
```

```
    return Column(
      children: <Widget>[
        RaisedButton(
          child: Text('Go back!'),
          onPressed: () => Navigator.pop(context),
        ),
        RaisedButton(
          child: Text('Go to SecondScreen... again!'),
          onPressed: () => _pushSecondScreen(context),
        ),
      ],
    );
  }
}


void main() {
  runApp(MaterialApp(
    initialRoute: '/',
    routes: {
      '/': (context) => FirstScreen(),
      '/second': (context) => SecondScreen(),
    },
  ));
}
```

Arrays

```dart
final length = items.length;


final newItems = items..addAll(otherItems);


final allEven = items.every((item) => item % 2 == 0);


final filled = List<int>.filled(3, 42);


final even = items.where((n) => n % 2 == 0).toList();


final found = items.firstWhere((item) => item.id == 42);


final index = items.indexWhere((item) => item.id == 42);


final flat = items.expand((_) => _).toList();


final mapped = items.expand((item) => [item + 1]).toList();


items.forEach((item) => print(item));


items.asMap().forEach((index, item) => print('$item,
$index'));
```

```
final includes = items.contains(42);


final indexOf = items.indexOf(42);


final joined = items.join(',');


final newItems = items.map((item) => item + 1).toList();


final item = items.removeLast();


items.add(42);


final reduced = items.fold({}, (acc, item) {
  acc[item.id] = item;
  return acc;
});


final reversed = items.reversed;


items.removeAt(0);


final slice = items.sublist(15, 42);


final hasOdd = items.any((item) => item % 2 == 0);
```

```
items.sort((a, b) => a - b);


items.replaceRange(15, 42, [1, 2, 3]);


items.insert(0, 42);
```

Make http request

```
dependencies:
  http: ^0.12.0
```

```
import 'dart:convert' show json;
import 'package:http/http.dart' as http;


http.get(API_URL).then((http.Response res) {
    final data = json.decode(res.body);
    print(data);
});
```

Async Await

```
Future<int> doSmthAsync() async {
  final result = await Future.value(42);
  return result;
}


class SomeClass {
  method() async {
```

```
    final result = await Future.value(42);

    return result;

  }

}
```

JSON

```
import 'dart:convert' show json;


json.decode(someString);

json.encode(encodableObject);
```

`json.decode` returns a `dynamic` type, which is probably not very useful

You should describe each entity as a Dart class with `fromJson` and `toJson` methods

```
class User {

    String displayName;

    String photoUrl;


    User({this.displayName, this.photoUrl});


    User.fromJson(Map<String, dynamic> json)

      : displayName = json['displayName'],

        photoUrl = json['photoUrl'];


    Map<String, dynamic> toJson() {

      return {

        'displayName': displayName,
```

```
        'photoUrl': photoUrl,
      };
    }
}


final user = User.fromJson(json.decode(jsonString));
json.encode(user.toJson());
```

However this approach is error-prone (e.g. you can forget to update map key after class field was renamed), so you can use `json_serializable` as an alternative

Add `json_annotation`, `build_runner` and `json_serializable` to dependencies

```
dependencies:
  json_annotation: ^2.0.0

dev_dependencies:
  build_runner: ^1.0.0
  json_serializable: ^2.0.0
```

Update your code

```
import 'package:json_annotation/json_annotation.dart';

part 'user.g.dart';

@JsonSerializable()
class User {
  String displayName;
```

```
  String photoUrl;


  User({this.displayName this.photoUrl});


  // _$UserFromJson is generated and available in
user.g.dart
  factory User.fromJson(Map<String, dynamic> json) {
    return _$UserFromJson(json);
  }


  // _$UserToJson is generated and available in user.g.dart
  Map<String, dynamic> toJson() => _$UserToJson(this);
}


final user = User.fromJson(json.decode(jsonString));
json.encode(user); // toJson is called by encode
```

Run `flutter packages pub run build_runner build` to generate serialization/deserialization code

To watch for changes run `flutter packages pub run build_runner watch`

Singleton

```
class Singleton {
  static Singleton _instance;


  final int prop;
```

```
  factory Singleton() =>
    _instance ??= new Singleton._internal();


  Singleton._internal()
    : prop = 42;
}
```

Debounce

```
Timer _debounce;


if (_debounce?.isActive ?? false) _debounce.cancel();
_debounce = Timer(const Duration(milliseconds: 500), () {
    someFN();
});
```