FLUTTER QUICK REFERENCE

ALVIN ALEXANDER

Flutter Quick Reference

Alvin Alexander

My notes on building iOS and Android apps
with Flutter (and Dart)

Copyright

Flutter Quick Reference

Version 0.1, published October 27, 2019

Other books by Alvin Alexander:

- Scala Cookbook
- Functional Programming, Simplified
- How I Sold My Business: A Personal Diary
- A Survival Guide for New Consultants

Contents

1
Introduction

This is an extremely early release of a book about Flutter and Dart. Because it's basically just a very large cheatsheet at this point — and also because I don't know if I'll ever take the time to finish it — I decided to make it free.

1.1 Background

The reason this book exists is because I've been looking for a way to develop iOS and Android applications from one codebase, and when I found Flutter I liked what I saw, so I used it to develop my free *Back To Now* app, and I'm also using it to develop other apps.

If you'd like to see some things you can do with Flutter, here are links to my Back To Now:

- Back To Now on Apple's App Store
- Back To Now on Google Play

2

Why Dart?

Flutter uses the Dart programming language, so to use Flutter you need to know Dart. Therefore, the first section of this book is about Dart.

I saw that one author referred to Dart as "Java light." That's accurate in that it's a language that reminds me of Java, C, and JavaScript, and if you have that background, you can start working with it fairly quickly. That being said, when I first started working with it I thought it had a few quirks, but in the end, those "quirky" features seem to make it a good language for Flutter development.

2.1 Good URLs

The dart.dev documentation is generally very good:

- dart.dev cheatsheet
- dart.dev language-tour
- dart.dev codelabs cheatsheet

3

Dart Overview

Some important things to know about Dart:

- Initially created in 2011; 2.0 was released in 2018
  - The current version in October, 2019 is 2.5.0
- Object-oriented — everything is an object
- Garbage collected
- Will feel familiar to C and Java programmers
- Type annotations are optional
  - You can write typed or untyped code
- When using types there's a special type named `dynamic`
  - Such as `List<dynamic>`, or `Map<String, dynamic>`
- Can have top-level functions, as well as methods in classes
- Can have top-level variables, as well as variables in classes
- Has `static`, like Java
- Doesn't have `public`, `protected`, and `private`
  - If a name begins with an underscore (_), it's private
- Can pass functions around
- Can use lambdas
- Google originally planned to have a Dart VM in Chrome, but they instead created a Dart-to-JavaScript compiler
- Supports Ahead of Time (AOT) compiling, which is important for Flutter

Other:

- The Dart recommendation is to use two spaces for indentation, I use four
- Uses single-quotes for strings (can also use double-quotes)
- Documentation comments use `///`
- Fields and methods are named in camelcase (by convention)

4

Dart REPL?

Dart doesn't have a REPL — a command-line tool for testing Dart code — but it does have DartPad:

- dartpad.dev

5
Dart Tour

To give you a taste of the Dart programming language, here's a quick overview of some of its most common features.

5.1 Variables

Variables can be declared in a variety of ways:

```
var i = 1;
int i = 1;
final i = 1;
final int i = 1;
const i = 1;

var s = 'foo';
String s = 'foo';
final s = 'foo';
final String s = 'foo';
const s = 'foo';
```

`final` is similar to `final` in Java and `val` in Scala and Kotlin, and it should be used when a value won't vary. `var` or the variable's data type can be used when the variable's value will vary over time.

`const` should be used for compile-time constants:

```
const PI = 3.14;
```

As shown in those examples, Dart statements end with the `;` character.

5.2 Strings

Strings are declared with single-quotes:

```
final hello = 'Hello, world';
String hello = 'Hello, world';
```

You can also use double-quotes, but single-quotes are preferred.

String interpolation works similar to other languages:

```
var first = 'Alvin';
var last = 'Alexander';
var name = '$first $last';
print('$last has ${last.length} characters');
```

Dart also supports multiline strings:

```
final address = '''
    P.O. Box 1082
    Talkeetna, AK 99676
''';
```

5.3 Lists

List are created with brackets:

```
final ints = [1,2,3];
final names = ['al', 'dave', 'frank'];
List<int> ints = [1,2,3];
ints.forEach((i) { print(i); });
```

A few more examples:

```
final nums = List<int>();
nums.add(1);
nums.addAll([2,3]);
nums.forEach((i) { print(i); });
```

```
final stuff = [1, 'a', 2.2];
List<dynamic> stuff = [1, 'a', 2.2];
stuff.forEach((x) { print(x); });
```

5.4 Maps

Maps are created with curly braces:

```
final map = {
    1 : 'one',
    2 : 'two'
}

Map<int,String> map = {
    1 : 'one',
    2 : 'two'
}
```

Iterate over map elements like this:

```
map.forEach((k,v) { print('k: $k, v: $v'); });
```

More examples:

```
var map = Map<int, String>();
map[1] = 'one';
map[2] = 'two';
map.forEach((k,v) { print('k: $k, v: $v'); });

var map = Map<int, dynamic>();
map[1] = 'one';
map[2] = 22;
map.forEach((k,v) { print('k: $k, v: $v'); });
```

5.5 Sets

Sets are also created with curly braces:

```
// duplicate values are dropped
final nums = {1,2,3,1,2};

// prints 1,2,3 (on separate lines)
nums.forEach((i) { print(i); });
```

5.6 Classes

Because Dart is an object-oriented language, classes are a common construct:

```
class Person {
    // these are private because they begin with an underscore
    String _firstName;
    String _lastName;

    Person(this._firstName, this._lastName);
    toString() => '$_firstName $_lastName';
}

var p = Person('Alvin', 'Alexander');
print(p);
```

Note that any field, method, or function that begins with an underscore (_) is "private to its library."

As shown, classes have fields and methods. Like Java, they can also have `static` members.

6

if/else

if/else statements look like Java and other C-like languages:

```
var age = 18;
if (age < 18) {
    print('No voting, no drinking');
} else if (age >= 18 && age < 21) {
    print('Yes to vote, no to drink');
} else {
    print('Yes to vote and drink');
}
```

Like Java, Dart has a ternary operator syntax:

```
var s = age >= 18 ? 'vote' : 'no vote';
print(s);
```

6.1 loops

Old-style for-loops look like this:

```
var nums = [1,2,3];
for (var i=0; i<nums.length; i++) {
    print(nums[i]);
}
```

There's also a for/in syntax:

```
for (int n in nums) {
    print(n);
}
```

Iterable things also have a `forEach` method:

```
nums.forEach((i) { print(i); });
```

6.2 Functions

You can define functions and function parameters using types, or not. One-line functions that are expressions (they return a value) can be written like this:

```
add1(i) => i + 1;          // no types
int add2(int i) => i + 2;  // with types
num add3(num i) => i + 3;

print('${add1(1)}');
print('${add2(1)}');
print('${add3(1)}');
```

Ways to define multiline functions with and without types:

```
int add1(int i) {
    return i + 1;
}

add2(i) {
    return i + 2;
}
```

6.3 Comments

Comments are like C and Java, except /// should be used for documentation comments:

```
// one-line comments

/// doc comments
/// use three slashes

/* you can do this too */
```

7

Create a new Dart project

To execute a single Dart file you can:

- Create a file like *foo.dart*
- Compile/run it with `dart foo.dart`

There are also at least two ways to create Dart projects:

- With Stagehand
- By hand/manually

7.1 1) Use Stagehand

- easiest way is to use Stagehand
- Example:
  pub global activate stagehand mkdir playground cd playground stagehand
  stagehand console-full // create a command-line application

7.2 2) Create a project manually

I wrote this article:

- Create a Dart project manually

The short version of that article is:

```
# create directory
mkdir my_app
cd my_app
```

```
# create config file
vi pubspec.yaml
    name: my_app
    description: My app
    version: 0.0.1
    author: alvin

    dependencies:
      test: ^1.6.0

# get dependencies
pub get

# create app
mkdir lib
vi lib/my_app.dart
      void main() {
          print('hello, world');
      }

# run
dart my_app.dart
```

7.2.1 pubspec.yaml and pub

Here are more details on *pubspec.yaml* and pub:

- pubspec.yaml
- "pub" package manager

8
Hello, world

Like C and Java, Dart uses `main` as the app entry point

- Can use these signatures:

```
void main() { ... }
void main(List<String> args) { ... }
```

- Here's a "Hello, world" application:

```
void main() {
   print('Hello, world');
}
```

- Save as *hello_world.dart*
- Compile and run:

  dart hello_world.dart

8.1 Other Dart commands

There are other Dart commands, such as for transpiling Dart to JavaScript, or Ahead-of-Time (AOT) compiling:

```
dart
dart2aot
dart2js
dartanalyzer
dartaotruntime
dartdevc
dartdoc
dartfmt
```

Flutter uses the AOT technology to create native Android and iOS apps.

9
Fields/Variables

There are several ways to declare variables in Dart. In general:

- Use `var` if the variable's contents will vary over time
- Use `final` if the variable will not vary
- Can also use `const` and `dynamic`
- Keywords for declaring variables:

  var generic var (dynamic) final can be set only once const compile-time constant static const at the class level, must be `static const` dynamic type can vary

Some examples:

```
var i = 1;
int i = 1;

final i = 1;
final int i = 1;

var s = 'foo';
String s = 'foo';

final s = 'foo';
final String s = 'foo';

// dynamic
dynamic aName = 'Bob';
aName = 42;
```

`const` should be used for compile-time constants:

```
const PI = 3.14;
const int LIFE_MEANING = 42;
```

A few notes:

- Instance variables can be `final` but not `const`
- `final` fields can be empty initially

```
final listOfInts = <int>[];
final setOfInts = <int>{};
final mapOfIntToDouble = <int, double>{};
final mapOfStringToWhatev = <String, dynamic>{};
```

9.1 Private fields

- If a field, method, or function (any identifier) name starts with an underscore (_) it's "private to its library"
  - It's only available in the *.dart* file in which it's defined

9.2 const vs final

- Both can only be set once
- `const` is a constant at compile-time — its value must be known at compile-time
- Examples:

```
const x = DateTime.now();  //error
final x = DateTime.now();  //works
```

Can also do this:

```
var list = const [1,2,3];
```

TODO: write more about `const`

9.3 Default values

- Uninitialized fields are `null`
- This includes numbers, because all fields are objects

9.4 Testing variable types

- Operators `as` and `is`

```
as   typecast
is   like instanceof
is!  !instanceof  (not `!is`)
```

10

Built-in types

- Built-in data types:

  bool int double String List num
- `num` is a superclass of `int` and `double`
- more:

```
final hexColor = 0xff993333;
final color = const Color(hexColor);
final exponent = 4.2e10;
```

- Note that Color is a Flutter class

11
Strings

- Declare with single-quotes
  - can also use double-quotes, but single-quotes are preferred
  - no difference in how they're interpolated
- Examples:

```
final hello = 'Hello, world';
String hello = 'Hello, world';
```

String interpolation works similar to other languages:

```
var first = 'Alvin';
var last = 'Alexander';
var name = '$first $last';
print('$last has ${last.length} characters');
```

Dart also supports multiline strings:

```
final address = '''
    P.O. Box 1082
    Talkeetna, AK 99676
''';
```

Being able to use both single- and double-quotes lets you do things like this:

```
final x = "'Hello', she said.";
final y = '"Hello", he said.';
```

11.1 Runes

- A Dart string is a sequence of UTF-16 values, so expressing 32-bit Unicode values requires a special syntax
- Similar to other languages, use `\uXXXX`, where XXXX is a 4-digit hexadecimal value
- More information here at dart.dev

11.2 Multiline strings

- Dart has multiline strings:

```
import 'dart:core';
var createString = '''
    create table quotes (
        _id INTEGER PRIMARY KEY AUTOINCREMENT,
        quote TEXT UNIQUE,
        author TEXT
    )
''';
```

- Dart does not have a way to remove whitespace on the left (like Scala's `stripMargin`)
  - I wrote this Dart `stripMargin` function for multiline strings

12

DateTime, DateFormat

- Dart has a DateTime class:

```
import 'package:intl/intl.dart';
var dt = DateTime.parse('2019-09-20 06:00:00');
var dt = DateTime
    .now()
    .add(new Duration(seconds: 10));
```

12.1 DateFormat

- TODO: this is a Flutter class
- https://api.flutter.dev/flutter/intl/DateFormat-class.html
- their examples:

```
Pattern                           Result
----------------                  -------
new DateFormat.yMd()             -> 7/10/1996
new DateFormat("yMd")            -> 7/10/1996
new DateFormat.yMMMMd("en_US")   -> July 10, 1996
new DateFormat.jm()              -> 5:08 PM
new DateFormat.yMd().add_jm()    -> 7/10/1996 5:08 PM
new DateFormat.Hm()              -> 17:08 // force 24 hour time
```

12.2 A function to get “number of seconds since epoch”

A `DateTime` example:

```
/// the current time, in “seconds since the epoch”
static int currentTimeInSeconds() {
    var ms = (new DateTime.now()).millisecondsSinceEpoch;
```

```
    return (ms / 1000).round();
}
```

13

Control-Flow Statements (TODO)

Dart has the usual assortment of control-flow statements:

- if/else
    - a ternary operator
- for loops
- while, do/while
- break, continue
- switch/case
- try/catch/finally

14

if/else

if/else statements look like Java and other C-like languages:

```
var age = 18;
if (age < 18) {
    print('No voting, no drinking');
} else if (age >= 18 && age < 21) {
    print('Yes to vote, no to drink');
} else {
    print('Yes to vote and drink');
}
```

Like Java, Dart has a ternary operator syntax:

```
var s = age >= 18 ? 'vote' : 'no vote';
print(s);

// ternary operator in a function
Color _rowBgColor(int rowNum) {
    return rowNum % 2 == 0 ? Colors.grey[100] : Colors.white;
}
```

14.1 for loops

Old-style for-loops look like this:

```
var nums = [1,2,3];
for (var i=0; i<num.length; i++) {
    print(nums[i]);
}
```

There's also a for/in syntax:

```
// typed
for (int n in nums) {
    print(n);
}

// no type
for (var n in nums) {
    print(n);
}
```

Iterable things also have a `forEach` method:

```
nums.forEach((i) { print(i); });
```

14.2 while, do-while

```
while (!isDone()) {
    doX();
}

do {
    doX();
} while (someCondition());
```

14.3 break, continue

```
while (true) {
    if (doX()) break;
    doY();
}

var nums = [1,2,3];
for (int i = 0; i < nums.length; i++) {
    var x = nums[i];
    if (x < 3) {
        print(x);
        continue;
    }
```

```
    doSomethingElse();
}
```

14.4 switch, case

- See the switch and case docs
- Can use `int`, `String`, and compile-time constants
- Use `break` in each case
  - If you don't use `break`, code will continue with the next case
- Use `default` as a catch-all

```
var num = 1;
switch (num) {
    case 1:
        print('1!');
        break;
    default:
        print('Not a 1!');
}
```

14.5 try/catch/finally, exceptions

- use `on` to catch a type
- use `catch` to catch an instance

```
try {
    doX();
} on FooException {
    handleFooException();
} on Exception catch (e) {
    print('Other exception: $e');
} catch (e) {
    print('Some other exception:$e');
} finally {
    doCleanup();
}
```

- can re-throw an exception:

```
} catch (e) {
    rethrow;
}
```

- `catch` can also get the stack trace as a variable:

```
} catch (e, s) {
    // e is exception
    // s is the stack trace
}
```

See the docs for more:

- exceptions on dart.dev

14.6 assert

Dart has an `assert` statement:

```
myFunction(x) {
    assert(x != null);
    // more code ...
}
```

14.7 Comments

Comments are like C and Java, except `///` should be used for documentation comments:

```
// one-line comments

/// doc comments
/// use three slashes

/* you can do this too */
```

15

Special Operators for null Values

Like Kotlin, Dart has special `null` operators:

- `a ??= b`
- `a = b ?? x`
- `a?.b`
- `a?.b?.doX()`

TODO: add more here

16

Packages, Imports, Libraries

Dart imports and libraries are different than Java.

- Basic file import:

```
import 'dart:html';
```

- Dart has a `package:` scheme where you can specify libraries that are provided by the `pub` package manager:

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

- Can import a part of a library:

```
// import only foo
import 'package:lib1/lib1.dart' show foo;

// import all names EXCEPT foo
import 'package:lib2/lib2.dart' hide foo;
```

- Can rename a library when you import it:

```
import 'package:lib2/lib2.dart' as lib2;
import 'package:lib2/lib2.dart' as lib2X;  //TODO: not tested
```

See the docs for more:

- Libraries and packages

16.1 Import example

As a real-world example, you need this import to use the Dart `DateTime` class:

```
import 'package:intl/intl.dart';
var dt = DateTime.parse('2019-09-20 06:00:00');
var dt = DateTime
    .now()
    .add(new Duration(seconds: 10));
```

16.2 Libraries

- TODO: Cover the `library` keyword

17
Classes

Introduction:

- Somewhat similar to Java
- Private fields and getters/setters are very different
- TODO: Different things you can do with constructor parameters
- TODO: Can have "factory" constructor methods
- Classes *can't* be declared inside other classes
- `static` is like Java

17.1 Example: My Quote class

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

class Quote {
    var id = 0;
    var quote = '';
    var author = '';
    var dateLastTouched = 0;   // unix epoch time (because of sqlite)

    Quote (
        this.id,
        this.quote,
        [this.author = '', this.dateLastTouched = 0]
    );

    bool operator ==(o) =>
      o != null && o is Quote && o.id == id && o.quote == quote && o.author == author;

    @override
```

```
    String toString() {
        return '''\n
id:     $id
quote:  $quote
author: $author
date:   $dateLastTouched
        ''';
    }

    Map<String, dynamic> toMap() {
        return {
            'id'     : id,
            'quote'  : quote,
         'author' : author ?? '',  //use `author` unless it's null, then use ''
            'date'   : dateLastTouched
        };
    }

    // TODO stop the `author` field from being null here?
    factory Quote.fromMap(Map<String, dynamic> qMap) {
        return Quote(
            qMap['id'],
            qMap['quote'],
            qMap['author'],
            qMap['dateLastTouched']
        );
    }

}
```

17.2 factory keyword

- Use `factory` when implementing "a constructor that doesn't always create a new instance." (TODO)

17.3 Class variables and methods

- Use the `static` keyword to implement class-wide variables and methods

- Static variables (class variables) are useful for class-wide state and constants:
- TODO: examples

18

Constructors

- see https://dart.dev/guides/language/language-tour#constructors

- basic syntax example from Dart website:

```
// a better approach is shown below
class Point {
    num x, y;

    Point(num x, num y) {
        // there's a better way to do this, stay tuned.
        this.x = x;
        this.y = y;
    }
}

// this is a better approach and the preferred Dart style
class Point {
    num x, y;
    Point(this.x, this.y);
}
```

Note: Use `this` only when there is a name conflict.

18.1 Inheritance

- Inheritance looks like this:

```
class Employee extends Person {
    //TODO more here
}
```

18.2 Constructors

- TODO:
  - delegating constructor
  - const constructor
  - initializer list

19

Getters and setters

- Getters and setters are significantly different than Java
- TODO: build on this example:

```
class Rectangle {
    num get area => width * height;
    set height(num value) => height = value;
}
```

20

Static Method and Fields

- Static methods are similar to Java
- Can also have static fields (not shown below)

Static method example:

```
import "dart:math";

class ListUtils {

    /// returns a random element from the list
    static T getRandomElement<T>(List<T> list) {
        final random = new Random();
        var i = random.nextInt(list.length);
        return list[i];
    }

}
```

More examples:

```
import 'list_utils.dart';
import 'database_helper.dart';
import 'data_model.dart';
import 'dart:async';

class Utils {

    static bool isNullOrEmpty(String s) {
        if (s == null) return true;
        if (s.trim() == '') return true;
        return false;
    }
```

```
    static Future<Quote> getRandomQuote(DatabaseHelper dbHelper) async {
        final listOfQuoteIds = await dbHelper.selectAllQuoteIds();
      final randomQuoteId = ListUtils.getRandomElement<int>(listOfQuoteIds);
        return await dbHelper.selectQuote(randomQuoteId);
    }

}
```

21

Implementing equals (==), hashCode

- The need to implement == and `hashCode` is similar to Java
- An == example:

```
class Quote {
    var quote = '';
    var author = '';

    Quote (
        this.quote,
        [this.author = '']
    );

    bool operator ==(o) =>
        o != null && o is Quote && o.quote == quote && o.author == author;

    // more code ...
}
```

22

Abstract classes

- Syntax is like Java
- Dart doesn't have interfaces
- Can use abstract classes to define interfaces (with or without implementation)

Example of abstract classes and inheritance:

```
abstract class UpdateTimeInterface {
    void updateTime(TimeOfDay tod);
}

class UpdateStartTime extends UpdateTimeInterface {
    void updateTime(TimeOfDay tod) {
        SharedPreferencesHelper.setStartTime(tod);
    }
}

class UpdateStopTime extends UpdateTimeInterface {
    void updateTime(TimeOfDay tod) {
        SharedPreferencesHelper.setStopTime(tod);
    }
}
```

23

Named parameters

- Named parameters

```
// caller *must* use the names bold and italics
void openSocket({int timeout, int linger}) {...}

openSocket(timeout: 2000, linger: 2000)
```

- Named parameters with default values:

```
// caller *must* use the names bold and italics
void openSocket({
    int timeout: 2000,
    int linger: 2000
}) {...}
```

24

Positional parameters

- [] means parameters are *positional*
- TODO: provide example

25

Enumerations

- enums can't be declared inside classes

```
enum Color { red, green, blue }
```

- TODO: more

26
Generics

- See https://dart.dev/guides/language/language-tour#generics
- Use or not-use as desired

Example of specifying a list of strings:

```
var names = List<String>();
names.addAll(['Joe', 'Donna']);
names.add('Kristy');
```

Example of a generic function:

```
static T getRandomElement<T>(List<T> list) {
    final random = new Random();
    var i = random.nextInt(list.length);
    return list[i];
}

// example 1 - calling getRandomElement
var list = ['a','b','c'];
var element = getRandomElement(list);
print(element);

// example 2 - calling getRandomElement
futureListOfQuotes.then((quotes) {
    var q = getRandomElement<Quote>(quotes);
    // more code here that uses `q` ...
});
```

In Flutter you'll see generic lists like this:

```
actions: <Widget>[
    // your widgets here ...
```

```
]
```

27

Mixins

- TODO

28

Methods and Functions

- Syntax is like Java, but:
  - Can specify the return type, or not
  - Can specify parameter types, or not
- Can use a special shorthand syntax for one-line expressions

Example syntax:

```
returnType functionName (type param1, type param2) {
    ...
    return returnType;
}
```

A real-world function that takes a string and returns a string:

```
String stripMargin(String s) {
    return s.splitMapJoin(
        RegExp(r'^', multiLine: true),
        onMatch: (_) => '\n',
        onNonMatch: (n) => n.trim(),
    );
}
```

28.1 => syntax

- The `=> expr` syntax is shorthand for `{ return expr; }`.
- The => notation is called *arrow syntax*.
- Can only be used for expressions (not statements)

Real-world examples:

```
static double _timeOfDayToDouble(TimeOfDay tod) =>
    tod.hour + tod.minute/60.0;
```

29

Method cascades

- Use `..` for multiple operations on the members of a single object
- Example:

```
foo(param1, param2)
    ..method1(x)
    ..method2(y);
```

30

Functions, anonymous functions (lambdas)

- In Dart you can pass anonymous functions into other functions

30.1 One-line anonymous function

Create a one-line anonymous function:

```
// create a list of 1 to 10
var ints = List<int>.generate(10, (i) => i + 1);

// filter the list with an anonymous function
// to retain only even numbers
ints.retainWhere((i) => i % 2 == 0);

// print the even numbers (2,4,6,8,10)
ints.forEach((i) => print(i));
```

Note that the `ints` list is modified with `retainWhere`. (This is not functional programming.)

30.2 Multiline anonymous functions

Create multiline anonymous functions with this syntax:

```
ints.retainWhere((i) {
    // do whatever you need ...
    return i % 2 == 0;
});
```

31

Passing Functions Around in Dart

- You can pass a function into other functions
- You can define a function as a function parameter using the Dart `Function`
  - This lets you create a higher-order function

31.1 Pass a function into a function

Pass a simple function into `forEach`:

```
// create a function
void printInt(int i) {
    print(i);
}

// pass it into forEach
var ints = [1, 2, 3];
ints.forEach(printInt);
```

31.2 Define a function that takes a function parameter

Here's an example of a function that takes an integer and a function:

```
// 1 - define a method that takes a function
void exec(int i, Function f) {
    print(f(i));
}

// 2 - define a function to pass in
int plusOne(int i) => i + 1;
int double(int i) => i * i;
```

```
// 3 - pass the functions into the method
exec(10, plusOne);  //11
exec(10, double);   //100

// 4 - you can also pass in lambda expressions
exec(10, (x) => x + 1);  //plusOne
exec(10, (x) => x * x);  //double
```

32
List

- TODO: more/better examples
- Creating lists:

```
final ints = <Int>[1,2,3];
final ints = [1,2,3];
List<int> ints = [1,2,3];

final constList = const [1, 2, 3];

List<String> names = [];
names.add('Alvin');
names.add('Kimberly');

// create a list of 1 to 10
final ints = List<int>.generate(10, (i) => i + 1);
final list = Iterable<int>.generate(10).toList()

// create a list from a list
final newList = List.from(otherList);
```

A few more examples:

```
final nums = List<int>();
nums.add(1);
nums.addAll([2,3]);

nums.forEach((i) { print(i); });

final stuff = [1, 'a', 2.2];
List<dynamic> stuff = [1, 'a', 2.2];
stuff.forEach((x) { print(x); });
```

Sorting and folding:

```
ints.sort((n1, n2) => n1 - n2);
var sum = ints.reduce((a, b) => a + b);
var sum = ints.fold(seed, (a, b) => a + b);
```

32.1 Iterating over lists

Old-style for-loops look like this:

```
var nums = [1,2,3];
for (var i=0; i<nums.length; i++) {
    print(nums[i]);
}
```

There's also a for/in syntax:

```
for (int n in nums) {
    print(n);
}
```

33
Map

- The Dart `Map` class

```
var map = {}
var map = new Map<>()
```

Maps are created with curly braces:

```
final map = {
    1 : 'one',
    2 : 'two'
}

Map<int,String> map = {
    1 : 'one',
    2 : 'two'
}
```

Iterate over map elements like this:

```
map.forEach((k,v) { print('k: $k, v: $v'); });
```

More examples:

```
var map = Map<int, String>();
map[1] = 'one';
map[2] = 'two';
map.forEach((k,v) { print('k: $k, v: $v'); });

var map = Map<int, dynamic>();
map[1] = 'one';
map[2] = 22;
map.forEach((k,v) { print('k: $k, v: $v'); });
```

34

Set

Sets are also created with curly braces:

```
// duplicate values are dropped
final nums = {1,2,3,1,2};

// prints 1,2,3 (on separate lines)
nums.forEach((i) { print(i); });
```

More examples:

```
// Set<String>
var names = {
    'emily',
    'hannah',
    'mercedes'
};

// empty set
var names = <String>{};

// add, addAll
var moreNames = <String>{};
moreNames.add('aleka');
moreNames.add('christina');
moreNames.addAll(names);

// compile-time constant set
final constNames = const {
    'emily',
    'hannah',
    'mercedes'
};
```

- Other Set methods
  - add
  - addAll
  - remove
  - contains
  - containsAll

35

Common Dart collections methods

- TODO: more examples
- Common methods
  - forEach
  - map
  - every
  - where, firstWhere, singleWhere
  - take
  - skip

35.1 forEach

```
final fruits = ['apple', 'banana', 'cherry'];
fruits.forEach((fruit) => print(fruit));
```

35.2 map

```
final fruits = ['apple', 'banana', 'cherry'];
List<String> capFruits = fruits.map((fruit) => capitalize(fruit)).toList();
capFruits.forEach((f) => print(f));

// that code assumes you have this `capitalize` function
String capitalize(String s) => s[0].toUpperCase() + s.substring(1);
```

35.3 contains

```
var ints = [1, 3, 2, 5, 4];
print(ints.contains(3));
```

35.4 sort

Sorting:

```
ints.sort((n1, n2) => n1 - n2);
```

35.5 reduce, fold

reduce and fold:

```
var sum = ints.reduce((a, b) => a + b);
var sum = ints.fold(seed, (a, b) => a + b);
```

36

Get a random element from a list

Here's a Dart function that returns a random element from a list:

```
import "dart:math";

T getRandomListElement<T>(List list) {
    final random = new Random();
    var i = random.nextInt(list.length);
    return list[i];
}

void main() {
    var list = ['a','b','c'];
    var element = getRandomListElement(list);
    print(element);
}
```

Notice that the function returns a generic type, so you can give it lists of different types.

37
SQLite

- I haven't used SQLite with plain Dart, but I have used it with Flutter
- Use the sqflite package for SQLite in Flutter

The rest of this chapter contains notes related to using SQLite with Flutter, and specifically with some date/time issues I had to resolve.

37.1 SQLite "date/time" notes

- https://www.sqlite.org/datatype3.html
- SQLite does not have a storage class set aside for storing dates and/or times. Instead, the built-in Date And Time Functions of SQLite are capable of storing dates and times as TEXT, REAL, or INTEGER values:

  TEXT as ISO8601 strings ("YYYY-MM-DD HH:MM:SS.SSS"). REAL as Julian day numbers, the number of days since noon in Greenwich on November 24, 4714 B.C. according to the proleptic Gregorian calendar. INTEGER as Unix Time, the number of seconds since 1970-01-01 00:00:00 UTC.
- Applications can chose to store dates and times in any of these formats and freely convert between formats using the built-in date and time functions.
- https://www.sqlite.org/lang_datefunc.html

  - SQLite supports five date and time functions as follows:

```
date(timestring, modifier, modifier, ...)
time(timestring, modifier, modifier, ...)
datetime(timestring, modifier, modifier, ...)
julianday(timestring, modifier, modifier, ...)
strftime(format, timestring, modifier, modifier, ...)
```

- looks like you can specify `'now'`, or this:

```
INSERT INTO Date (LastModifiedTime) VALUES(CURRENT_TIMESTAMP)

// CURRENT_DATE, CURRENT_TIME, and CURRENT_TIMESTAMP

mDb.execSQL("INSERT INTO "+DATABASE_TABLE+" VALUES (null, datetime()) ");

created_date date default CURRENT_DATE
created_at DATETIME DEFAULT CURRENT_TIMESTAMP
```

- i ended up adding my database field like this:

```
date_last_touched INTEGER DEFAULT (cast(strftime('%s','now') as int))
```

- and in the Dart code (TODO: this code is a little old now):

```
class Quote {
    var id = 0;
    var quote = '';
    var author = '';
    var dateLastTouched = 0;   // unix epoch time (because of sqlite)

    Quote (
        this.id,
        this.quote,
        [this.author = '', this.dateLastTouched = 0]
    );

    @override
    String toString() {
        return '''\n
id:     $id
quote:  $quote
author: $author
date:   $dateLastTouched
        ''';
    }

    Map<String, dynamic> toMap() {
        return {
```

```
            'id'     : id,
            'quote'  : quote,
         'author' : author ?? '',  //use `author` unless it's null, then use ''
            'date'   : dateLastTouched
        };
    }

    // TODO stop the `author` field from being null here?
    factory Quote.fromMap(Map<String, dynamic> qMap) {
        return Quote(
            qMap['id'],
            qMap['quote'],
            qMap['author'],
            qMap['dateLastTouched']
        );
    }

}
```

38

DateTime (and SQLite)

- `DateTime(2018, 11, 17)`
- You'll note there is no direct means of storing a date or time. However, you could insert an integer representation of the value, perhaps by using DateTime's millisecondsSinceEpoch and fromMillisecondsSinceEpoch methods.

39

Asynchronous and parallel

- https://dart.dev/guides/language/language-tour#asynchrony-support
- The Flutter/Dart people prefer `await`, I prefer the Future/then approach
- TODO: Copy some of my blog content here; better intro; add more examples

Some of my blog posts:

- How to run multiple Dart futures in parallel
- Dart futures are NOT run in a separate thread (they are run in the event loop)
- A Dart Future/then/catchError example
- How to use a Future with a Duration delay in Dart (and Flutter)

40

Futures

- The code looks like Futures in other languages
- Dart/Flutter references seem to prefer using `async` and `await`, but I'm more comfortable with `Future.then`, feels more parallel/concurrent
- Note that Dart is single-threaded, and futures are not run in a separate thread

40.1 Examples

- Here are two examples of how to use a `Future` with a `Duration` delay in Dart (and Flutter). This is something I do when testing possible long-running Flutter methods:

```
// example 1
Future<bool> _getFutureBool() {
    return Future.delayed(Duration(milliseconds: 500))
        .then((onValue) => true);
}

// example 2
print('1');
Future.delayed(const Duration(milliseconds: 500), () {
    print('Hello, world');
});
print('2');
```

If you're comfortable with Dart futures, you know that the second example will (eventually) print this output:

```
1
2
Hello, world
```

40.2 Returning a known value in a Future

Sometimes an API may require you to return a `Future` even if you already have a known value. If/when that happens, you can use this approach:

```
return new Future(() {return 42; });
```

I was reminded of this when I saw the following code, and asked why it was the way it was:

```
return Future.delayed(Duration(milliseconds: 0))
    .then((onValue) => 42);
```

40.3 Future, then, and catchError

Here's an example of getting a value that might be returned slowly. It also shows something you might do to catch an error during this process:

```
var rez = false;
Future<SharedPreferences> fPrefs = SharedPreferences.getInstance();
fPrefs.then((value) {
        rez = value.getBool(KEY_ENABLE_NOTIFICATIONS) ?? false;
    })
    .catchError((e) {
        // log the error
        Log.error("ERROR: ${e.error}");
        // return a default value
        return 60;
    });
return rez;
```

- Here's another Future/then example:

```
var futureListOfQuotes = _dbHelper.selectAllQuotes();

futureListOfQuotes.then((quotes) {
    var q = ListUtils.getRandomElement<Quote>(quotes);
    flutterLocalNotificationsPlugin.schedule(
        0,
```

```
        q.author,
        q.quote,
        scheduledNotificationDateTime,
        platformChannelSpecifics,
        payload: 'Default_Sound',
    );
});
```

40.4 Running multiple futures simultaneously

This code shows how to run multiple Dart futures in parallel. They run in a total time of 3 seconds:

```
import 'dart:async';

Future<int> async1() async {
    await Future<String>.delayed(const Duration(seconds: 1));
    return 10;
}

Future<int> async2() async {
    await Future<String>.delayed(const Duration(seconds: 2));
    return 20;
}

Future<int> async3() async {
    await Future<String>.delayed(const Duration(seconds: 3));
    return 30;
}

void main() async {
  var t1 = DateTime.now();
  Future.wait([async1(), async2(), async3()])
    .then((List<int> nums) {
        var t2 = DateTime.now();
        var sum = nums.reduce((curr, next) => curr + next);
        print('sum = $sum');
        var delta = t2.difference(t1);  // should be 3, not 6
```

```
        print('delta = $delta');
    });
}
```

The code output looks like this:

```
sum = 60
delta = 0:00:03.006000
```

- note that there's a Stopwatch class I might have been able to use here

41

Dart Isolates (actors)

- Dart Isolates are a primitive form of actors
- "Primitive" in that you constantly reference low-level features like ports (compared to Akka, which is at a higher level of abstraction)
- I wrote up the following example in my blog post, A Dart Isolates example:

```
import 'dart:async';
import 'dart:isolate';

/// i created this example based on this example:
/// http://jpryan.me/dartbyexample/examples/isolates/
main() async {

    // a long-lived port for receiving messages
    var ourFirstReceivePort = new ReceivePort();

    // spawn the 'echo' actor,
    await Isolate.spawn(echo, ourFirstReceivePort.sendPort);

    /// the 'echo' isolate sends its SendPort as the first message,
    /// so we can talk to it. we'll always use this port to communicate
    /// with it.
    var echoPort = await ourFirstReceivePort.first;

    /// if you try to use our first receive port, you'll get this error:
    /// "Bad state: Stream has already been listened to."
    /// always need a new port to communicate with an actor.
    var ourSecondReceivePort = ReceivePort();
    echoPort.send(['message 1', ourSecondReceivePort.sendPort]);
    var msg = await ourSecondReceivePort.first;
    print('main received "$msg"');
```

```
    var port3 = ReceivePort();
    echoPort.send(['message 2', port3.sendPort]);
    port3.first.then((msg) {
        print('main received "$msg"');
    });

    var port4 = ReceivePort();
    echoPort.send(['port 4', port4.sendPort]);
    port4.first.then((msg) {
        print('main received "$msg"');
    });

    print('end of main');

    // msg = await sendReceive(sendPort, "bar");
    // print('main received "$msg"');
}

/// `echo` is intended to be the equivalent of an actor.
echo(SendPort sendPort) async {

    /// open our receive port. this is like turning on
    /// our cellphone.
    var ourReceivePort = ReceivePort();

    /// tell whoever created us what port they can reach us on
    /// (like giving them our phone number)
    sendPort.send(ourReceivePort.sendPort);

    /// listen for text messages that are sent to us,
    /// and respond to them with this algorithm
    await for (var msg in ourReceivePort) {
      var data = msg[0];           // the 1st element they send us is their message
        print('echo received "$data"');
      SendPort replyToPort = msg[1];   // the 2nd element they send us is their port

        //replyToPort.send('echo said: ' + data);
        Future.delayed(const Duration(milliseconds: 500), () {
            replyToPort.send('echo said: ' + data);
```

```
        });


        //if (data == "bar") ourReceivePort.close();
    }
}

/// sends a message on a port, receives the response,
/// and returns the message
Future sendReceive(SendPort port, msg) {
    ReceivePort response = new ReceivePort();
    port.send([msg, response.sendPort]);
    return response.first;
}
```

42

Using Timers (Scheduling) in Dart

- Fire a timer periodically

```
// do this every 30 seconds
Timer.periodic(Duration(seconds: 30), (timer) {
    final futureQuote = _getRandomQuote();
    futureQuote.then((q) {
        _currentRandomQuoteQuote = q.quote;
        _currentRandomQuoteAuthor = q.author;
        debugPrint('    NEW QUOTE: ${q.quote}');
    });
});
```

- More: https://fluttermaster.com/tips-to-use-timer-in-dart-and-flutter
- TODO: more examples

43

Flutter

- Flutter lets you write apps that work with both iOS and Android
- At some point you may be able to run desktop apps
- Flutter is also said to be the engine for Fuchsia
- I wrote my Back To Now app using Flutter

44

Creating a Flutter Project

- Can create a new project at the command line or through VS Code or Android Studio

44.1 flutter create

- `flutter create`

```
flutter create navigation_101
Wrote 66 files.

All done!
[ ] Flutter is fully installed. (Channel stable, v1.7.8+hotfix.4, on Mac OS X 10.14.6 18G87, loc
[ ] Android toolchain - develop for Android devices is fully installed. (Android SDK version 28.
[!] Xcode - develop for iOS and macOS is partially installed; more components are available. (Xco
[ ] iOS tools - develop for iOS devices is not installed.
[ ] Android Studio is fully installed. (version 3.4)
[!] IntelliJ IDEA Community Edition is partially installed; more components are available. (vers
[ ] VS Code is fully installed. (version 1.37.1)
[!] Connected device is not available.

Run "flutter doctor" for information about installing additional components.
```

44.2 More command line options

It's been a while since I did this last, but I think a command like this is correct:

```
flutter create my_app --project-name MyApp --org com.valleyprogramming
```

That command should create a project directory named *my_app*. `cd` into that directory to start working on your project.

44.3 pubspec.yaml file

The `flutter create` command creates a *pubspec.yaml* file in your project directory. That file will eventually grow to something like this:

```
name: just_be
description: A new Flutter project.

version: 1.0.0+3

environment:
  sdk: ">=2.1.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  sqflite: ^1.1.6
  path_provider: ^1.2.2
  flutter_local_notifications: ^0.8.3
  datetime_picker_formfield: ^0.4.3
  shared_preferences: ^0.5.3+4
  flutter_html: ^0.11.0
  url_launcher: ^5.1.3
  auto_size_text: ^2.1.0
  simple_animations: ^1.3.3

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^0.1.2

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_launcher_icons: "^0.7.3"

flutter_icons:
  ios: true
  android: true
  image_path_ios: "assets/app_icon/justbe_512px.png"
```

```
  image_path_android: "assets/app_icon/justbe_512px.png"
  #adaptive_icon_background: "assets/app_icon/background.png"
  #adaptive_icon_foreground: "assets/app_icon/foreground.png"

flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  # assets:
  #  - images/a_dot_burr.jpeg
  #  - images/a_dot_ham.jpeg

 # An image asset can refer to one or more resolution-specific "variants", see
  # https://flutter.dev/assets-and-images/#resolution-aware.

  # For details regarding adding assets from package dependencies, see
  # https://flutter.dev/assets-and-images/#from-packages

  # To add custom fonts to your application, add a fonts section here,
  # in this "flutter" section. Each entry in this list should have a
  # "family" key with the font family name, and a "fonts" key with a
  # list giving the asset and other descriptors for the font. For
  # example:
  # fonts:
  #   - family: Schyler
  #     fonts:
  #       - asset: fonts/Schyler-Regular.ttf
  #       - asset: fonts/Schyler-Italic.ttf
  #         style: italic
  #   - family: Trajan Pro
  #     fonts:
  #       - asset: fonts/TrajanPro.ttf
  #       - asset: fonts/TrajanPro_Bold.ttf
  #         weight: 700
  #
```

```
  # For details regarding fonts from package dependencies,
  # see https://flutter.dev/custom-fonts/#from-packages
```

44.4 Don't forget the help

Run `flutter help create` to see the options for creating a new Flutter project.

```
$ flutter help create

Create a new Flutter project.

If run on a project that already exists, this will repair the project, recreating any files that are

Usage: flutter create <output directory>
-h, --help                        Print this usage information.
   --[no-]pub                     Whether to run "flutter pub get" after the project has been created.
                                  (defaults to on)

   --[no-]offline                 When "flutter pub get" is run by the create command, this indicates wheth
                                  or not. In offline mode, it will need to have all dependencies already availab
                                  succeed.

   --[no-]with-driver-test        Also add a flutter_driver dependency and generate a sample 'flutter d
-t, --template=<type>             Specify the type of project to create.

          [app]                   (default) Generate a Flutter application.
       [package]                  Generate a shareable Flutter project containing modular Dart code.
       [plugin]                   Generate a shareable Flutter project containing an API in Dart code with a
                                  implementation for Android, for iOS code, or for both.

-s, --sample=<id>                 Specifies the Flutter code sample to use as the main.dart for an applica
                                  The value should be the sample ID of the desired sample from the API documenta
                                     (http://docs.flutter.dev). An example can be found at
                                  https://master-api.flutter.dev/flutter/widgets/SingleChildScrollView-class

   --list-samples=<path>          Specifies a JSON output file for a listing of Flutter code samples that
   --[no-]overwrite               When performing operations, overwrite existing files.
   --description                  The description to use for your new Flutter project. This string ends up
```

```
                                (defaults to "A new Flutter project.")

   --org                 The organization responsible for your new Flutter project, in reverse dom
                        string is used in Java package names and as prefix in the iOS bundle identif
                                (defaults to "com.example")

   --project-name          The project name for this new Flutter project. This must be a valid dar
-i, --ios-language                [objc, swift (default)]
-a, --android-language            [java, kotlin (default)]
   --[no-]androidx          Generate a project using the AndroidX support libraries

Run "flutter help" to see global options.
```

45

Simulators/Emulators

I generally open simulators from Visual Studio Code or Android Studio, but you can also do it from the command line.

- iOS:

```
open -a Simulator
```

- Android: TBD

These commands may also work:

```
flutter emulators --launch <emulator id>
flutter emulators --launch apple_ios_simulator
```

46

Run project

- Step 1: Have an emulator running, or a physical device running and connected, and then …
- VS Code
    - right-click main.dart, select “Start ”
- Command line

```
flutter run
```

Can also use this command to build a faster AOT app on a physical device:

```
flutter run --profile
```

47

Flutter performance and Debug and Profile Modes

- Debug mode can be slow on both emulators and physical devices
- Use *Profile mode* — only available on real hardware — to see something that's close to AOT performance:

```
flutter run --profile
```

- Per the docs:

  Flutter's profile mode compiles and launches your application almost identically to release mode, but with just enough additional functionality to allow debugging performance problems.

- note: with `flutter run --profile` you can't use commands like `r` or `R` to rebuild your app
- the Flutter UI performance page
- my page on Flutter performance problems and Profile Mode
- note that when you use `flutter run`, you see this output:

```
Launching lib/main.dart on Android SDK built for x86 in debug mode...
Initializing gradle...                                              7.4s
Resolving dependencies...                                           1.5s
Running Gradle task 'assembleDebug'...
Running Gradle task 'assembleDebug'... Done                        13.2s
Built build/app/outputs/apk/debug/app-debug.apk.
Installing build/app/outputs/apk/app.apk...                         4.3s
D/EGL_emulation(22080): eglMakeCurrent: 0x9b305420: ver 3 0 (tinfo 0x9b303350)
Syncing files to device Android SDK built for x86...
```

- with `flutter run --profile`, you see output like this:

```
Initializing gradle...                                      7.4s
Resolving dependencies...                                   1.7s
Launching lib/main.dart on Nexus 9 in profile mode...
Running Gradle task 'assembleProfile'...
Running Gradle task 'assembleProfile'... Done              54.0s
Built build/app/outputs/apk/profile/app-profile.apk (11.0MB).
Installing build/app/outputs/apk/app.apk...                 3.2s
```

48

Flutter commands

- `flutter create`:

```
$ flutter create navigation_101

Wrote 66 files.

All done!
[ ] Flutter is fully installed. (Channel stable, v1.7.8+hotfix.4, on Mac OS X 10.14.6 18G87, loc
[ ] Android toolchain - develop for Android devices is fully installed. (Android SDK version 28.
[!] Xcode - develop for iOS and macOS is partially installed; more components are available. (Xco
[ ] iOS tools - develop for iOS devices is not installed.
[ ] Android Studio is fully installed. (version 3.4)
[!] IntelliJ IDEA Community Edition is partially installed; more components are available. (vers
[ ] VS Code is fully installed. (version 1.37.1)
[!] Connected device is not available.

Run "flutter doctor" for information about installing additional components.
```

- devices

```
flutter devices
```

- list emulators

```
$ flutter emulators

6 available emulators:

Nexus_5X_API_25_7.1.1_      • Nexus 5X      • Google • Nexus 5X API 25 (7.1.1)
Nexus_9_API_23_6.0_         • Nexus 9       • Google • Nexus 9 API 23 (6.0)
Nexus_9_API_25              • Nexus 9       • Google • Nexus 9 API 25
Pixel_2_API_27_Oreo_8.1_    • pixel_2       • Google • Pixel 2 API 27 (Oreo 8.1)
```

```
Pixel_C_API_25_Android_7.1.1_ • pixel_c     • Google • Pixel C API 25 (Android 7.1.1)
apple_ios_simulator           • iOS Simulator • Apple
```

- start emulator

```
flutter emulators --launch <emulator id>
flutter emulators --launch apple_ios_simulator
```

- run

```
$ flutter run

[your app will start in the emulator once it's compiled/built]

r, R
d - detach
h - more help
q
```

- flutter help:

```
> flutter -h
Manage your Flutter app development.

Common commands:

  flutter create <output directory>
    Create a new Flutter project in the specified directory.

  flutter run [options]
    Run your Flutter application on an attached device or in an emulator.

Usage: flutter <command> [arguments]

Global options:
-h, --help                  Print this usage information.
-v, --verbose           Noisy logging, including all shell commands executed.
                            If used with --help, shows hidden options.
```

```
-d, --device-id             Target device id or name (prefixes allowed).
    --version               Reports the version of this tool.
   --suppress-analytics   Suppress analytics reporting when this command runs.
   --bug-report        Captures a bug report file to submit to the Flutter team.
                   Contains local paths, device identifiers, and log snippets.

Available commands:
  analyze                  Analyze the project's Dart code.
  attach                   Attach to a running application.
  bash-completion         Output command line shell completion setup scripts.
  build                    Flutter build commands.
  channel                  List or switch flutter channels.
  clean                    Delete the build/ and .dart_tool/ directories.
  config                   Configure Flutter settings.
  create                   Create a new Flutter project.
  devices                  List all connected devices.
  doctor                   Show information about the installed tooling.
  drive                   Runs Flutter Driver tests for the current project.
  emulators                List, launch and create emulators.
  format                   Format one or more dart files.
  help                     Display help information for flutter.
  install                  Install a Flutter app on an attached device.
  logs                     Show log output for running Flutter apps.
  make-host-app-editable  Moves host apps from generated directories to non-generated directori
                           developers.
  precache             Populates the Flutter tool's cache of binary artifacts.
  pub                      Commands for managing Flutter packages.
  run                      Run your Flutter app on an attached device.
  screenshot               Take a screenshot from a connected device.
  test                     Run Flutter unit tests for the current project.
  upgrade                  Upgrade your copy of Flutter.
  version                  List or switch flutter versions.
```

49

Debug output

Print debug output in your code with `debugPrint`:

```
debugPrint("quote:  ${quote.quote}");
debugPrint("author: ${quote.author}");
```

50

Flutter Stateless Widgets

- TODO: lots of work needed here

50.1 How to start

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

More:

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

class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Not Sure Where This Is Used',
            home: TodoListScreen(),
        );
    }
}
```

51
StatefulWidget

- TODO: lots of work needed here

51.1 Example 1

This is the wrong way to use `FutureBuilder`, but everything else should be okay. Shows a simple boolean field:

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

class PreferencesEnableNotifications extends StatefulWidget {
    @override
   PreferencesEnableNotificationsState createState() => PreferencesEnableNotificationsState()
}

class PreferencesEnableNotificationsState extends State<PreferencesEnableNotifications> {

    @override
    Widget build(BuildContext context) {

        return Scaffold(
            appBar: AppBar(
                title: Text("Enable Notifications"),
            ),
            body: FutureBuilder<bool>(
                future: SharedPreferencesHelper.getEnableNotifications(),
                builder: (BuildContext context, AsyncSnapshot snapshot) {
                    if (snapshot.connectionState == ConnectionState.done) {
                        return CheckboxListTile(
                            title: const Text('Enable Notifications'),
                         value: snapshot.data,  //the bool in the preferences
```

```
                            onChanged: (val) {
                                setState(() {
                            SharedPreferencesHelper.setEnableNotifications(val);
                                });
                            }
                        );
                    } else {
                        return new CircularProgressIndicator();
                    }
                }
            )
        );
    }
}
```

51.2 Initialization

- example:

```
@override
initState() {
    super.initState();
    Notifications.initializeNotifications();
}
```

- example:

```
@override
void initState() {
    super.initState();
    _quoteController = TextEditingController(text: "Hello, world");
   _authorController = TextEditingController(text: "Kernighan and Ritchie");
}
```

52
AppBar Widget

- Purpose: Show an AppBar at the top of the current screen
- Basic example:

```
return Scaffold(
    appBar: AppBar(title: Text('To-Do List')),
```

52.1 Back button

- Manually add an AppBar Back button:

```
appBar: AppBar(
    title: const Text("Just Be"),
    backgroundColor: Colors.blue,
    leading: IconButton(icon:Icon(Icons.arrow_back),  //<<--- back button
        onPressed:() => Navigator.pop(context, false),
    ),
)
```

52.2 Menu item (in upper-right)

- AppBar with a button as an icon/action/menu-item:

```
return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
          title: const Text("Just Be"),
          backgroundColor: Colors.blue,
          actions: <Widget>[              //<<---
            IconButton(
              icon: Icon(Icons.arrow_back),
```

```
              onPressed: () {
                Navigator.pop(context);
              },
            ),
          ]
      ),
      ...
      ...
```

- Add overflow menu items to the AppBar; i only care about the Cancel action, so it's a little contrived; your `onSelected` algorithm should be a function that handle the `value` field in the `PopupMenuButton` options:

```
actions: <Widget>[
    // overflow menu
    PopupMenuButton<void>(
        onSelected: (value) => Navigator.of(context).pop(),
        itemBuilder: (context) => [
            PopupMenuItem(
                value: 1,
                child: Text("Cancel"),
            ),
            PopupMenuItem(
                value: 2,
                child: Text("Action 2"),
            ),
        ],
    ),
]
```

53

Text, TextField, TextStyle

- Text, TextField
- Text API
  - a TextSpan is shown on that page
- Example:

```
title: Text('New Task'),
content: TextField(
    controller: controller,
    autofocus: true,
),
```

- TextStyle
  - see TextStyle API

```
Text(
    'Notification Time Interval',
    style: TextStyle(
        fontSize: 13,
        color: Colors.black54
    ),
)
```

Larger example:

```
return Container(
    child: Column(
        children: <Widget>[
            Text(
                customer.name,
```

```
              style: TextStyle(fontSize: 30.0, fontWeight: FontWeight.bold),
            ),
            Text(
                customer.location,
              style: TextStyle(fontSize: 24.0, fontWeight: FontWeight.bold),
            )
        ],
    ),
    padding: EdgeInsets.all(20.0)
);

Text(_customer.name,
    style: TextStyle(
        fontSize: 30.0,
        fontWeight: FontWeight.bold,
    ),
    textAlign: TextAlign.center
),
```

54

FlatButton, RaisedButton

- TODO: need some discussion here

54.1 FlatButton

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

FlatButton(
    child: Text("Cancel"),
    onPressed: () => Navigator.of(context).pop(false),
    textColor: Colors.black,
),
```

54.2 RaisedButton

- I think these examples come from flutter.dev/docs/cookbook/navigation/navigation-basics

```
child: RaisedButton(
  child: Text('Open route'),
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => SecondRoute()),
    );
  },
```

```
),

body: Center(
  child: RaisedButton(
    onPressed: () {
      Navigator.pop(context);
    },
    child: Text('Go back!'),
  ),
),

PaddedRaisedButton(
    buttonText: 'Show plain notification with payload',
    onPressed: () async {
        await _showNotification();
    },
),
```

55
IconButton

- Set the color of the arrow_back icon to white (it's black by default):

```
child: IconButton(
    icon: Icon(Icons.arrow_back, color: Colors.white),
    onPressed: () {
        Navigator.of(context).pop();
    }
)
```

56

FloatingActionButton widget

- This is the Android FAB button, i.e., the big "+" button at the bottom-right of a screen that's used to add a new item

```
return Scaffold(
    appBar: AppBar(title: Text('My App')),
    body: ...
    floatingActionButton: FloatingActionButton(
        child: Icon(Icons.add),
        tooltip: 'Add',
        onPressed: _addNewFoo,
    ),
);
```

- Example:

```
floatingActionButton: FloatingActionButton(
    onPressed: () {
        Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => EditQuoteForm(null)),
        );
    },
    ...
    ...
```

- Example:

```
floatingActionButton: FloatingActionButton(
    backgroundColor: Globals.BLUE_BG_COLOR,
    onPressed: () {
        Navigator.push(
            context,
```

```
            MaterialPageRoute(builder: (context) => EditQuoteForm(null)),
        );
    },

    // // NOTIFICATIONS
    // onPressed: () {
    //     debugPrint('*** SENDING NOTIFICATION ***');
    //     _showNotificationWithDefaultSound();
    //     //TODO delete this when done
    //     _scaffoldKey.currentState.showSnackBar(
    //         SnackBar(
    //             content: Text('Sending notification'),
    //             duration: Duration(seconds: 3),
    //     ));
    // },

    tooltip: 'Add new',
    child: const Icon(Icons.add)
),
```

57

Layout Containers

- Flutter containers can container multiple children or just one child.
  - They will have a `children:` or `child:` property
- There are more containers than what I cover here

57.1 Multi-Child Layout Containers

- There are many containers, but Row and Column can probably be used for 80% of needs (outside of lists and grids)
- See flutter.dev/docs/development/ui/widgets/layout… for more
- Row
  - Lay widgets out horizontally
  - No scrolling
    - Will get an error if it's too big
  - Use MainAxisAlignment for horizontal alignment
  - Use CrossAxisAlignment for vertical alignment
  - "If you only have one child, consider using Align or Center to position the child."
- Column
  - Children are shown at their natural size
    - To make a child fit all the space, wrap it with Expanded
  - No scrolling
    - Will get an error if it's too big
    - Use ListView if you need scrolling

  - Use MainAxisAlignment for vertical alignment
  - Use CrossAxisAlignment for horizontal alignment
  - "If you only have one child, consider using Align or Center to position the child."

- Wrap
  - like Row and Column, but when it runs out of space it wraps to the next line
  - direction, alignment, spacing, children

- Flex
  - "displays its children in a one-dimensional array"
  - can act as both Row and Column
  - if you know the main axis in advance, use Row or Column instead

- Flow
  - "sizes and positions children efficiently, according to the logic in a FlowDelegate"

- ListView
  - Most commonly used *scrollable* widget
  - Can be a horizontal or vertical list

- ListViewBuilder
  - use for large lists, optimized for scrolling

- ListTile (doesn't belong in this section, but it's used with ListView and ListViewBuilder)
  - 1 to 3 lines of text
  - optional icons and other widgets, such as checkboxes
  - different subtypes
    * CheckboxListTile

        * RadioListTile

- GridView
    - A *scrollable* 2D array of widgets
    - Use for things like a grid of photos
    - GridTile, GridTileBar
- GridViewBuilder
    - use for large lists, optimized for scrolling
- Stack
    - lay widgets on top of each other
    - might see the lower cards depending on the sizes
    - differs from Card
- StreamBuilder
    - listens for stream updates, rebuilds the widgets when the data changes
- Table
    - A grid that doesn't scroll
    - Similar to HTML
      tag
- More:
    - CustomMultiChildLayout
    - IndexedStack
    - LayoutBuilder
    - ListBody
- TODO: TabBar, Tabs, TabBarView

57.2 Single-Child Layout Containers

- These have a `child:` property
- More details: flutter.dev/docs/development/ui/widgets/layout…
- Container
  - color, padding, decoration, margin, transform, width, height, alignment
  - `decoration: BoxDecoration` is common
- Padding
  - Control padding with `EdgeInsets`
- Center
  - Centers its child
- Align
  - Place widgets in a defined area of their parent widget
- AspectRatio
- Baseline
- ConstrainedBox
  - impose constraints on its child widget
  - ex: `constraints: const BoxConstraints(minHeight: 20.0),`
- CustomSingleChildLayout
- FittedBox
  - "Scales and positions its child within itself according to `fit:`."
- Expanded
  - expands its child widget per its `flex:` field
- Flexible

  - makes its child flexible using `flex:` and `fit:` (`FlexFit.loose`, `FlexFit.tight`)

- SafeArea
  - "Insets its child by sufficient padding to avoid intrusions by the operating system"
  - Think about the "notch" on top of newer phones
  - I had problems with my Nexus 9 and the iOS simulator

- More:
  - Gesture
  - Card
  - FractionallySizedBox
  - Offstage
  - OverflowBox
  - SizedBox
  - Transform

58
Container

- `Container` is useful for specifying a number of things:
  - one child
  - color
  - width, height
  - padding (with `EdgeInsets`)
  - alignment
  - constraints
  - decoration
  - margin
  - transform
- padding and margin examples:

```
padding: EdgeInsets.all(20.0)
padding: EdgeInsets.only(top: 15.0)
padding: EdgeInsets.fromLTRB(15, 15, 15, 40)
margin: EdgeInsets.symmetric(vertical: 16.0)
```

- example:

```
body: Container(
  child: Column(
    children: <Widget>[
      Expanded(
        child: Container(),
        flex: 1,
      ),
      Expanded(
        child: TaskView(),
```

```
        flex: 3
      )
    ]
  ),
```

- example:

```
return Dismissible(
    key: UniqueKey(),
    //key: ValueKey(cat.hashCode.toString()),
    background: Container(
        color: Colors.red,
        child: Icon(Icons.delete_forever)
    ),
    direction: DismissDirection.endToStart,
    onDismissed: (direction) {
        _handleDelete(currentQuote.id);
    },
    child: Container(
        color: _rowBgColor(index),
        child: _buildListTile(padding, currentQuote)
    )
); //Dismissible
```

- example:

```
Container(
    color: _darkBlue,
    width: _width,
    height: _topHeight,
    padding: EdgeInsets.all(40.0),
    child: Row(
        children: <Widget>[
            Expanded( // Constrains AutoSizeText to the width of the Row
                // this does all the work of fading-in the text
                child: FadeIn(2, _getSizedText(_quoteText))
            ),
        ],
    )
```

```
),
```

59
Column

- Column
  - Children are shown at their natural size
    - To make a child fit all the space, wrap it with Expanded
  - No scrolling
    - Will get an error if it's too big
    - Use ListView if you need scrolling
  - Use MainAxisAlignment for vertical alignment
  - Use CrossAxisAlignment for horizontal alignment
  - "If you only have one child, consider using Align or Center to position the child."
- example:

```
body: Container(
  child: Column(
    children: <Widget>[
      Expanded(
        child: Container(),
        flex: 1,
      ),
      Expanded(
        child: TaskView(),
        flex: 3
      )
    ]
  ),
```

- Column + Container + Row example:

```
child: Column(
  children: <Widget>[
    Container(
      color: _darkBlue,
      ...
      child: Row(
        children: <Widget>[
          Expanded(
            child: FadeIn(2, _getSizedText(_quoteText))
          ),
        ],
      )
    ),

    Row(
      children: [
        Container(
          color: _darkBlue,
          ...
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              _homeButton(context),
              _refreshButton(context)
            ]
          ),
        )
      ],
    )
  ],
)
```

60
Row

- Row
  - Lay widgets out horizontally
  - No scrolling
    * Will get an error if it's too big
  - Use MainAxisAlignment for horizontal alignment
  - Use CrossAxisAlignment for vertical alignment
  - "If you only have one child, consider using Align or Center to position the child."
- a Row with three children, so it's on row like "X X X" (not my code):

```
Row(
    children: [
        Container(
            color: Colors.red,
            child: const Icon(Icons.add)
        ),
        Container(
            color: Colors.green,
            child: const Icon(Icons.add)
        ),
        Container(
            color: Colors.blue,
            child: const Icon(Icons.add)
        ),
    ],
)
```

- Column + Container + Row example:

```
child: Column(
  children: <Widget>[
    Container(
      color: _darkBlue,
      ...
      child: Row(
        children: <Widget>[
          Expanded(
            child: FadeIn(2, _getSizedText(_quoteText))
          ),
        ],
      )
    ),

    Row(
      children: [
        Container(
          color: _darkBlue,
          ...
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              _homeButton(context),
              _refreshButton(context)
            ]
          ),
        )
      ],
    )
  ],
)
```

- Good resource: medium.com/jlouage/flutter-row-column-cheat-sheet…

61

FutureBuilder (and Future)

- Pretty much need this any time you're retrieving data: from a database, web service, anything that might take a while — any time you use a Future to retrieve data (which is always)
- A correct FutureBuilder example
- Good summary in this medium.com article: "FutureBuilder is a widget that returns another widget based on the Future's execution result. It serves as a bridge between Futures and the widget's UI."
- See the last example in this chapter for a 100%-correct example

Here's a 90%-correct `FutureBuilder` example/solution with a `CheckboxListTile`:

```
var _enableNotificationsCheckbox = FutureBuilder<bool>(
    /// this is not 100% correct because it references a function;
    /// should never reference a function, always a variable
    future: SharedPreferencesHelper.getEnableNotifications(),
    builder: (BuildContext context, AsyncSnapshot snapshot) {
        return CheckboxListTile(
            title: const Text('Enable Notifications'),
         subtitle: const Text('Select this checkbox to enable notifications'),
            value: snapshot.data,  //the bool in the preferences
            onChanged: (val) {
                setState(() {
                    // save the new value
                    SharedPreferencesHelper.setEnableNotifications(val);
                });
            }
        );
    }
);
```

61.1 Building up to a FutureBuilder

As a little bit of process/explanation, I started with this:

```
return CheckboxListTile(
    title: const Text('Enable Notifications'),
    subtitle: const Text('Select this checkbox to enable notifications'),
    value: ???
    onChanged: (val) {
        setState(() {
            ???
        });
    }
);
```

- i knew i needed to replace those question marks, but the data comes from a function that returns a Future, so you have to use a FutureBuilder.
- so the next code started to look like this:

```
return FutureBuilder(
    future: SharedPreferencesHelper.getEnableNotifications(),
    builder: (BuildContext context, AsyncSnapshot snapshot) {
        // return the contained widget here
    }
);
```

- key points about FutureBuilder:
  - it's a widget
  - wrap your other widget with it so you can send it the "future" data
  - it has smarts built into it that update the wrapped widget when it eventually receives the data
  - the `future:` parameter connects to your data source, in this case the `SharedPreferencesHelper.getEnableNotifications` function, which returns a `bool`
    * `future: SharedPreferencesHelper.getEnableNotifications()`
  - i find that it helps to show that data type in this code:
    * `var _enableNotificationsCheckbox = FutureBuilder<bool>(`

  - this is boilerplate:

```
* builder: (BuildContext context, AsyncSnapshot snapshot) {
```

61.2 Future

- I didn't need this code, but if you ever need a Future.then:

```
// using with FutureBuilder, simulating a slow response
Future<bool> _getFutureBool() {
    return Future.delayed(Duration(milliseconds: 500))
        .then((onValue) => true);
}
```

- Another `Future.then`:

```
bool rez = false;
Future<SharedPreferences> fPrefs = SharedPreferences.getInstance();
fPrefs.then((value) {rez = value.getBool(KEY_ENABLE_NOTIFICATIONS) ?? false; })
   .catchError((e) {
       debugPrint("===== ERROR: ${e.error}");
       return 60;
   });
return rez;
```

- Here's a `Future.delayed`:

```
// prints 1, 2, Hello
print('1');
Future.delayed(const Duration(milliseconds: 500), () {
    print('Hello, world');
});
print('2');
```

61.3 A correct example

I need to trim this down to show only what's needed, but I think this is a 100% correct example:

```
import 'package:flutter/material.dart';
import 'package:auto_size_text/auto_size_text.dart';
import 'utils.dart';
import 'database_helper.dart';
import 'data_model.dart';
import 'main.dart';
import 'fade_in.dart';
import 'dart:async';

class CurrentQuoteWidget extends StatefulWidget {
    @override
    CurrentQuoteWidgetState createState() {
        return new CurrentQuoteWidgetState();
    }
}

class CurrentQuoteWidgetState extends State<CurrentQuoteWidget> with
SingleTickerProviderStateMixin {

  final _dbHelper = DatabaseHelper.instance;
  static const _textColor = Color(0xbbffffff);
  static const _buttonColor = Color(0xaaffffff);

  static const _lightBlue = Color(0xFF90CAF9);
  static const _darkBlue = Color(0xFF1565C0);
  static const _animationDurationMs = 1500;

  @override
  void initState() {
      super.initState();
      _futureQuote = _getRandomQuote();
  }

  IconButton _homeButton(BuildContext context) {
    return IconButton(
      icon: Icon(Icons.home, color: _buttonColor),
      onPressed: () {
        Navigator.pushNamed(context, '/home');
      }
```

```
    );
  }

  IconButton _refreshButton(BuildContext context) {
    return IconButton(
      icon: Icon(Icons.refresh, color: _buttonColor),
      onPressed: () {
        // var _navState = Navigator.of(context);
        // debugPrint('Navigator-can-pop: ${_navState.canPop()}');
        // debugPrint('Navigator-context: ${_navState.toString()}');

        // this is supposed to clear all routes/history. see:
      // https://stackoverflow.com/questions/45889341/flutter-remove-all-routes
        Navigator.of(context)
          .pushNamedAndRemoveUntil('/here', (Route<dynamic> route) => false);
      }
    );
  }

  AutoSizeText _getSizedText(String text) {
    return AutoSizeText(
      text,
      maxLines: 10,
      softWrap: true,
      textAlign: _getTextAlignment(text),
      style: TextStyle(
        color: _textColor,
        fontSize: 40,
        height: 1.4
      ),
    );
  }

  TextAlign _getTextAlignment(String s) {
    if (s.length < 30)
      return TextAlign.center;
    else
      return TextAlign.left;
  }
```

```
String _convertQuoteAndAuthorToString(Quote q) {
  if (q.author == null || q.author == '') {
    return q.quote;
  } else {
    return q.quote + '\n\n' + q.author;
  }
}

Future<Quote> _futureQuote;
Future<Quote> _getRandomQuote() async {
  return Utils.getRandomQuote(_dbHelper);
}

@override
Widget build(BuildContext context) {

  final _height = MediaQuery.of(context).size.height;
  final _width = MediaQuery.of(context).size.width;

 /// NOTE: SafeArea was causing all of my problems related to too many border
  /// pixels and seeing the yellow/orange “caution” bars, so i removed it.
  var _topHeight = 0.8 * _height;
  var _bottomHeight = _height - _topHeight;

  return MaterialApp(

    routes: {
        '/home': (context) => MyApp(),
        '/here': (context) => CurrentQuoteWidget(),
    },

    home: Scaffold(

        body: FutureBuilder<Quote>(
          // it’s important that this is a variable, and not a function
          future: _futureQuote,
        builder: (BuildContext context, AsyncSnapshot<Quote> asyncQuote) {
            if (asyncQuote.hasData) {
```

```
var _quoteText = _convertQuoteAndAuthorToString(asyncQuote.data);
       return Center(
           child: Column(
             children: <Widget>[
               Container(
                 color: _darkBlue,
                 width: _width,
                 height: _topHeight,
                 padding: EdgeInsets.all(40.0),
                 child: Row(
                   children: <Widget>[
              Expanded( // Constrains AutoSizeText to the width of the Row
                      // this does all the work of fading-in the text
                         child: FadeIn(2, _getSizedText(_quoteText))
                       ),
                   ],
                 )
               ),

               Row(
                 children: [
                   Container(
                     color: _darkBlue,
                     height: _bottomHeight,
                     width: _width,
                     child: Row(
                     mainAxisAlignment: MainAxisAlignment.center,
                       children: [
                         _homeButton(context),
                         _refreshButton(context)
                       ]
                     ),
                   )
                 ],
               )
             ],
           )
     );
```

```
              } // `builder:`
              else {
                return Center(child: CircularProgressIndicator());
              }
            },
          )
      )
    );

  } //build
}  //class
```

62
Padding

- Add padding around a Container or other widgets

```
@override
Widget build(BuildContext context) {
    return Padding(
           -------
        padding: EdgeInsets.all(10.0),
        child: Container(
            color: Colors.yellow[100],
            decoration: BoxDecoration(border: Border.all()),
            padding: EdgeInsets.all(20.0),
            child: Center(
                child: Column(children: <Widget>[
                    Text(
                        'yada yada yada',
                        style: TextStyle(fontSize: 24.0)
                    ),
                    Padding(
                    -------
                        padding: EdgeInsets.only(top: 20.0),
                        child: ...
                ])
            )
        )
    );
}
```

- EdgeInsets methods:

```
EdgeInsets.all(10.0)
EdgeInsets.fromLTRB(20, 20, 20, 0)
EdgeInsets.symmetric(vertical: 0, horizontal: 10)
```

```
EdgeInsets.only(bottom : 8)
```

63

ListView, ListView.builder, ListTile

- `ListView`
  - A scrollable list of widgets arranged linearly
  - ListView is the most commonly used scrolling widget
- `ListTile`
  - A single fixed-height row that typically contains some text as well as a leading or trailing icon
  - A list tile contains one to three lines of text optionally flanked by icons or other widgets, such as check boxes
- Example:

```
@override
Widget build(BuildContext context) {
    ListView builder = ListView.builder(
                       ----------------
        itemCount: _people.length,
        itemBuilder: (context, index) {
            var person = _people[index];
            return ListTile(
                title: Text('${person['person']}'),
                subtitle: Text(
                    '${person['fname']}, ${person['lname']}'
                    trailing: Icon(Icons.arrow_right)
                );
            });
        return new Scaffold(
            appBar: new AppBar(
                title: new Text("People"),
            ),
```

```
            body: new Center(child: builder));
                                    -------
}
```

- ListTile example:

```
ListTile createPersonWidget(BuildContext context, Person person) {
    return new ListTile(
        title: Text('${person.fname}, ${person.lname}'),
        trailing: Icon(Icons.arrow_right),
        onTap: () => navigateToPerson(context, person)
    );
}
```

- [Good @suragch ListTile Medium post](https://medium.com/@suragch/a-complete-guide-to-flutters-listtile-597a20a3d449)
- My own Drawer + ListView + ListTile code:

```
drawer: Drawer(
      child: ListView(
          padding: EdgeInsets.fromLTRB(10, 20, 10, 10),
          children: <Widget>[

              ListTile(
                  title: const Text('Quotes'),
                  onTap: () {
                      Navigator.push(
                          context,
                      MaterialPageRoute(builder: (context) => MyHomePage()),
                      );
                  },
              ),

              ListTile(
                  title: const Text('Preferences'),
                  onTap: () {
                    Navigator.push(
                        context,
                   MaterialPageRoute(builder: (context) => PreferencesMain()),
```

```
                  );
                },
              ),
      ]
    )
  ), // drawer: Drawer
```

- My *prefs_time_interval.dart* class:

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

class TimeValue {
    final int _key;
    final String _value;
    TimeValue(this._key, this._value);
}

class PreferencesTimeInterval extends StatefulWidget {
    @override
   PreferencesTimeIntervalState createState() => PreferencesTimeIntervalState();
}

class PreferencesTimeIntervalState extends State<PreferencesTimeInterval> {

    final List<TimeValue> _buttonOptions = [
        TimeValue(60,  "1 hour"),
        TimeValue(120, "2 hours"),
        TimeValue(240, "4 hours"),
        TimeValue(480, "8 hours"),
        TimeValue(720, "12 hours"),
    ];

    List<Widget> _radioButtonOptions(int selectedValue) {        // LIST
      return _buttonOptions.map((timeValue) => RadioListTile(  // RADIO LIST TILE
           groupValue: selectedValue,
           title: Text(timeValue._value),
           value: timeValue._key,
           onChanged: (val) {
```

```
            setState(() {
                SharedPreferencesHelper.setNotificationTimeInterval(val);
            });
        },
      )).toList();
  }

  @override
  Widget build(BuildContext context) {
      return Scaffold(
          appBar: AppBar(
              title: Text("Notification Time Interval"),
          ),
          body: FutureBuilder<int>(
            future: SharedPreferencesHelper.getNotificationTimeInterval(),
              builder: (BuildContext context, AsyncSnapshot snapshot) {
                  if (snapshot.connectionState == ConnectionState.done) {
                      return ListView(
                          padding: EdgeInsets.all(8.0),
                          children: _radioButtonOptions(snapshot.data)
                      );
                  } else {
                      return new CircularProgressIndicator();
                  }
              }
          )
      );
  }
}
```

64
ListTile

- Use with ListView and ListViewBuilder.
- Example:

```
ListTile(
    title: const Text('Quotes'),
    onTap: () {
        Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => MyHomePage()),
        );
    },
),
```

- Example:

```
ListTile createPersonWidget(BuildContext context, Person person) {
    return new ListTile(
        title: Text('${person.fname}, ${person.lname}'),
        trailing: Icon(Icons.arrow_right),
        onTap: () => navigateToPerson(context, person)
    );
}
```

65

Dismissible: Swipe Left to Delete on a ListView

- Use the `Dismissible` widget to enable "swipe left to delete"
- You still have to add the custom code to do the delete

```
return Dismissible(
    key: UniqueKey(),
    background: Container(color: Colors.red),
    onDismissed: (direction) {
        debugPrint('\nWOULD HAVE DONE DELETE');
        //DBProvider.db.deleteClient(item.id);
    },
    child: Container(
        color: _rowBgColor(index),
        child: _buildListTile(padding, item)
    )
);
```

66

GestureDetector

- Used to detect many (all?) possible gestures
- Wrap it around a child to add gestures to that child
- GestureDetector docs
- Example:

```
return GestureDetector (
   onTap: () =>
       Navigator.push(
           context,
           MaterialPageRoute(builder: (context) => SecondRoute())
       ),
   child: Container(
       color: bgColor,
       child: ListTile(
           contentPadding: padding,
           title: Text(
               '${_quote.quote}'
           ),
           //trailing: Icon(Icons.arrow_right)
       ),
    )
);
```

67
SwitchListTile

- A ListTile with a Switch on the right
- ListTile docs
- UI looks like:

```
[icon]  [your text here]  [on/off switch]
```

- Example:

```
SwitchListTile(
    title: const Text('SOME TEXT'),
    value: _object.attribute,
    onChanged: (bool val) => setState(() => YOUR_LOGIC_HERE)
)
```

68

CheckboxListTile

- A ListTile with your text and a Checkbox
- CheckboxListTile API
- UI looks like:

```
[icon]  [your text here]  [checkbox]
```

```
var _enableNotificationsCheckbox = CheckboxListTile(
    title: const Text('Enable Notifications'),
    value: _notificationsAreEnabled,
    //secondary: const Icon(Icons.hourglass_empty),
    onChanged: (val) {
        setState(() { _notificationsAreEnabled = val; });
    }
);
```

- use this inside `CheckboxListTile` to put the checkbox *before* the text:

```
controlAffinity: ListTileControlAffinity.leading,
```

69

SingleChildScrollView

- Use this to scroll the widget it contains
- SingleChildScrollView class
- Example:

```
return showDialog<DialogAction>(
    context: context,
    barrierDismissible: false,  // must tap a button
    builder: (BuildContext buildContext) {
      return AlertDialog(
          title: Text('Quote field is empty'),
          content: SingleChildScrollView(     //<-- HERE
            child: ListBody(
              children: <Widget>[
                Text(_helpText)
              ],
            ),
          ),
          actions: [
            _returnToQuoteButton,
            _returnToQuoteList
          ]
      );
    },
);
```

- example:

```
return new Scaffold(
    appBar: new AppBar(
        title: new Text(title),
    ),
```

```
    body:
      Center(
        child: SingleChildScrollView(
               ---------------------
          padding: EdgeInsets.fromLTRB(15, 15, 15, 40),
          child: Html(
              data: _aboutText,
              customTextAlign: (dom.Node node) {
                  if (node is dom.Element) {
                      switch (node.localName) {
                          case "p": return TextAlign.justify;
                      }
                  }
                return null;
              },
              onLinkTap: (url) {
                  debugPrint("Opening $url...");
                  _launchURL(url);
              },
              customTextStyle: (dom.Node node, TextStyle baseStyle) {
                  if (node is dom.Element) {
                      switch (node.localName) {
                          case "br": return baseStyle.merge(brTextStyle);
                          case "p": return baseStyle.merge(pTextStyle);
                          case "h1": return baseStyle.merge(h1TextStyle);
                      }
                  }
                return baseStyle;
              },
          )


        )
    )
);
```

70

RefreshIndicator

- A widget that enables a “swipe to refresh” gesture
- Docs

71

Icons

- Notes about using stock icons in your apps
- use `Icons.*`
- Example:

```
Icons.add
Icons.arrow_right
Icons.delete_forever

child: Icon(Icons.delete_forever)
child: const Icon(Icons.add)
```

- Example:

```
appBar: AppBar(
    title: Text(_title),
    leading: IconButton(
        icon: Icon(Icons.check),
        onPressed: () => _onBackPressed()
    ),
```

- Example:

```
IconButton _homeButton(BuildContext context) {
    return IconButton(
        icon: Icon(Icons.home, color: _buttonColor),
        onPressed: () {
            Navigator.pushNamed(context, '/home');
        }
    );
}
```

72

FadeInImage

- Per the docs, “An image that shows a placeholder image while the target image is loading, then fades in the new image when it loads.”
- Docs - cookbook
- Example:

```
body: new Center(
    child: FadeInImage.assetNetwork(
        image: 'https://...',
        fadeInDuration: Duration(seconds: 2),
        placeholder: 'assets/loading.gif',
))
```

73

BoxDecoration, DecorationImage, AssetImage

- From the docs, "an immutable description of how to paint a box"
- Examples:

```
child: Container(
    decoration: BoxDecoration(
        color: Colors.blue,
        border: Border.all()),
    padding: EdgeInsets.all(16.0),
```

```
decoration: BoxDecoration(
    image: DecorationImage(
        image: AssetImage('assets/images/...'),
        fit: BoxFit.cover
    ),
```

```
decoration: BoxDecoration(
    color: Colors.orange,
    borderRadius: BorderRadius.only(
        topLeft: Radius.circular(20.0),
        topRight: Radius.circular(20.0)
    )
),
```

74
Colors

- How to specify colors in Flutter
- Flutter Color class
- TODO: Cover Theme colors
- Basics:

```
color: Colors.yellow,
color: Colors.yellow[100],  //alpha-ish
textColor: Colors.white,
color: Theme.of(context).primaryColor,
```

- Custom RGB colors (hexadecimal color strings)
  - must be specified as `0x` number
  - opacity must always be set (leading `ff`):
- Examples:

```
final _appBarColor = const Color(0xff33cc66);  // green
final _appBarColor = const Color(0xff3366ff);  // purple-ish
final _appBarColor = const Color(0x993366ff); // same, but transparent (`99`)
color: const Color(0xbbffffff)  // white with transparency
```

- Example: Definition of blue colors in *colors.dart*:

```
static const MaterialColor blue = MaterialColor(
    _bluePrimaryValue,
    <int, Color>{
        50: Color(0xFFE3F2FD),
        100: Color(0xFFBBDEFB),
        200: Color(0xFF90CAF9),
        300: Color(0xFF64B5F6),
```

```
        400: Color(0xFF42A5F5),
        500: Color(_bluePrimaryValue),
        600: Color(0xFF1E88E5),
        700: Color(0xFF1976D2),
        800: Color(0xFF1565C0),
        900: Color(0xFF0D47A1),
    },
);
static const int _bluePrimaryValue = 0xFF2196F3;
```

75

Themes

- TODO

76

Vertical Padding/Spacing/Spacer

- Use the Spacer class
- Creates an empty spacer whose size you can control
- Use in Flex widgets like Row and Column

```
Spacer()  // default to flex=1
Spacer(flex: 2)
```

- TODO: I used a SizedBox once, but I don't remember why:

```
SizedBox(height: 16.0),
```

77

Duration

- You'll use this all the time in timers and animation
- Docs
- Examples:

```
Duration d = new Duration(
    hours: 1,
    minutes: 2,
    seconds: 3
);
```

```
SnackBar(
    content: Text('Sending notification'),
    duration: Duration(seconds: 3),
)
```

```
final tween = MultiTrackTween([
    Track("opacity")
        .add(Duration(milliseconds: 500), Tween(begin: 0.0, end: 1.0)),
    Track("translateX").add(
        Duration(milliseconds: 500), Tween(begin: 130.0, end: 0.0),
        curve: Curves.easeOut)
]);
```

```
_notificationTimer = Timer.periodic(Duration(minutes: _notificationTimeInterval), (timer) {
    var now = _timeOfDayToDouble(TimeOfDay.now());
    var startTime = _timeOfDayToDouble(_startTime);
    var stopTime = _timeOfDayToDouble(_stopTime);
    if (now > startTime && now < stopTime) {
```

```
        Utils.getRandomQuote(_dbHelper).then((q) {
            _currentRandomQuoteQuote = q.quote;
            _currentRandomQuoteAuthor = q.author;
            _fireNotification();
        });
    } else {
        //debugPrint('**** outside the schedule, not doing any work!');
    }
});
```

78

Using Timers (Scheduling) in Flutter

- Dart: https://fluttermaster.com/tips-to-use-timer-in-dart-and-flutter
- Background tasks in Flutter (Android and iOS):
  - https://stackoverflow.com/questions/51706265/how-to-schedule-background-tasks-in-flutter
  - https://medium.com/flutter/executing-dart-in-the-background-with-flutter-plugins-and-geofencing-2b3e40a1a124
  - https://medium.com/vrt-digital-studio/flutter-workmanager-81e0cfbd6f6e
- An (alpha) package that comes from those Medium articles:
  - https://pub.dev/packages/workmanager
- PROBLEM: flutter_local_notifications does not support custom, periodic notifications:
  - https://github.com/MaikuB/flutter_local_notifications/issues/92
- starting to use Dart::Timer because flutter_local_notifications.periodicallyShow has limitations

79

Forms

- Flutter.dev, Build a form with validation
- `Form` example:

```
body: Container(
  child: Builder(
    builder: (context) => Form(
                          ----
      key: _formKey,
      child: Column(
        //crossAxisAlignment: CrossAxisAlignment.start,
        children: <Widget>[
          TextFormField(
          -------------
            controller: _quoteController,
            keyboardType: TextInputType.multiline,
            minLines: 5,
            maxLines: 12,
            textCapitalization: TextCapitalization.sentences,
            decoration: InputDecoration(
              labelText: 'Quote'
            ),
            validator: (value) {
              if (value.isEmpty) {
                debugPrint('VALIDATOR IS CALLED value isEmpty');
              }
              return null;
            },
            // onSaved: (val) => setState(() => _quote.quote = val)
            // onSaved is an optional block that can be used to run
            // code when the user saves the form.
          ),
```

```
          TextFormField(
          -------------
            controller: _authorController,
            keyboardType: TextInputType.text,
            textCapitalization: TextCapitalization.words,
            decoration: InputDecoration(
              labelText: 'Author',
              hintText: 'not required'
            ),
            // onSaved: (val) => setState(() => _quote.author = val),
          ),
        ],
      ),
    )
  )
)
```

80

Form validation (a two-step process)

- Set up your form field validators:

```
TextFormField(
    validator: (value) {
     if (value.isEmpty) {
       return 'Please enter some text';
     }
     return null;
    },
),
```

- Right before you save the form to a database or web service, call `validate`:

```
if (form.validate()) {
    form.save();
    _user.save();
    _showDialog(context);
}
```

- In my case that call looks like this:

```
class EditQuoteFormState extends State<EditQuoteForm> {

    final _formKey = GlobalKey<FormState>();

    @override
    Widget build(BuildContext context) {

    Future<void> _onBackPressed() async {

        if (_formKey.currentState.validate()) {
            debugPrint('VALIDATION succeeded');
```

```
        } else {
            debugPrint('VALIDATION failed');
        }

        // more code ...
```

More information:

- Flutter.dev — Flutter form validation
- Medium.com — Form validation in Flutter

That second link includes this `TextFormField` example:

```
new TextFormField(
    decoration: const InputDecoration(labelText: 'Name'),
    keyboardType: TextInputType.text,
    validator: (String arg) {
    if (arg.length < 3)
        return 'Name must be more than 2 character';
    else
        return null;
    },
    onSaved: (String val) {
        _name = val;
    },
),
```

81

How to set TextFormField initial value

- How to set TextFormField initial value when using a TextEditingController:

```
class EditQuoteFormState extends State<EditQuoteForm> {

  // STEP 1
  TextEditingController _quoteController;
  TextEditingController _authorController;

  Quote _quote;

  EditQuoteFormState(this._quote) {
    /// STEP 2
    /// this code populates the `text` field, which is used later
    /// by the controllers
    _quoteController = TextEditingController(text: _quote.quote);
    _authorController = TextEditingController(text: _quote.author);
  }

  // later ...

  @override
  Widget build(BuildContext context) {

      // STEP 3
      TextFormField(
        controller: _quoteController,
        ...
      ),
      TextFormField(
        controller: _authorController,
        ...
```

The key here is to pass the initial text field value to the controller, like this:

```
_myController = TextEditingController(text: 'The initial value');
```

Also, note that while I populate the TextEditingController objects in my constructor, they can also be populated in the initState method:

```
@override
void initState() {
    super.initState();
    _quoteController = TextEditingController(text: "Hello, world");
   _authorController = TextEditingController(text: "Kernighan and Ritchie");
}
```

- Can do this if you're not using a controller:

```
TextFormField(
    initialValue: 'Hello, world',
    ...
),
```

- See my example, How to supply an initial value to a TextFormField

81.1 TextInputType, capitalization

- Capitalize the first word of each sentence (`textCapitalization`):

```
TextFormField(
    controller: _authorController,
    keyboardType: TextInputType.text,
    textCapitalization: TextCapitalization.sentences,
    decoration: InputDecoration(
        labelText: 'Author',
        hintText: 'not required'
    ),
    // onSaved: (val) => setState(() => _quote.author = val),
),
```

More:

```
textCapitalization: TextCapitalization.sentences,  //Four score and seven
textCapitalization: TextCapitalization.words,      //Alvin Alexander
```

- Good TextField resource (keyboard types, sentences, borders, max length):
  - https://medium.com/flutter-community/a-deep-dive-into-flutter-textfields-f0e676aaab7a

82

TextFormField keyboard types (keyboardType: TextInputType)

- TextFormField
- TextInputType
  - datetime
  - emailAddress
  - multiline
  - number
  - phone
  - text
  - url
- TextFormField::textCapitalization is also important:
  - textCapitalization: TextCapitalization.sentences (capitalize first word in sentence)
  - textCapitalization: TextCapitalization.words (capitalize each word)
  - TextCapitalization.words (capitalize all characters)
- Examples:

```
new TextFormField(
    decoration: const InputDecoration(labelText: 'Name'),
    keyboardType: TextInputType.text,
    textCapitalization: TextCapitalization.words
),
new TextFormField(
    decoration: const InputDecoration(labelText: 'Phone'),
    keyboardType: TextInputType.phone,
```

```
),
new TextFormField(
    decoration: const InputDecoration(labelText: 'E-Mail'),
    keyboardType: TextInputType.emailAddress,
),
```

- my "Quote" TextFormField in Just Be:

```
TextFormField(
    controller: _quoteController,
    keyboardType: TextInputType.multiline,
    minLines: 5,
    maxLines: 12,
    textCapitalization: TextCapitalization.sentences,
    decoration: InputDecoration(
        labelText: 'Quote'
    ),
    validator: (value) {
        if (value.isEmpty) {
            debugPrint('VALIDATOR IS CALLED value isEmpty');
        }
        return null;
    },
    // onSaved: (val) => setState(() => _quote.quote = val)
    // onSaved is an optional block that can be used to run
    // code when the user saves the form.
),
```

- my "Author" TextFormField in Just Be (not required):

```
TextFormField(
    controller: _authorController,
    keyboardType: TextInputType.text,
    textCapitalization: TextCapitalization.words,
    decoration: InputDecoration(
        labelText: 'Author',
        hintText: 'not required'
    ),
    // onSaved: (val) => setState(() => _quote.author = val),
```

```
),
```

83

My own Drawer + ListView + ListTile code:

- A drawer that slides in from the edge (typically left side) of a Scaffold
- Docs
- Example:

```
drawer: Drawer(
    child: ListView(
        padding: EdgeInsets.fromLTRB(10, 20, 10, 10),
        children: <Widget>[
            ListTile(
                title: const Text('Quotes'),
                onTap: () {
                    Navigator.push(
                        context,
                      MaterialPageRoute(builder: (context) => MyHomePage()),
                    );
                },
            ),
            ListTile(
                title: const Text('Preferences'),
                onTap: () {
                    Navigator.push(
                        context,
                 MaterialPageRoute(builder: (context) => PreferencesMain()),
                    );
                },
            ),
        ]
    )
), // drawer: Drawer
```

84

SharedPreferences

- Save user preferences
- Data is saved in an XML file
- Implemented as a plugin: shared_preferences
- Example of a "helper" class:

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

class SharedPreferencesHelper {

    // preferences keys
    static const KEY_ENABLE_NOTIFICATIONS = 'PREFS_ENABLE_NOTIFICATIONS';

    // the time interval here is minutes, so 60 == 1 hour
   static const KEY_NOTIF_TIME_INTERVAL = 'PREFS_NOTIFICATION_TIME_INTERVAL';
    static const _KEY_START_TIME_HOUR = 'PREFS_START_TIME_HOUR';
    static const _KEY_START_TIME_MIN  = 'PREFS_START_TIME_MIN';
    static const _KEY_STOP_TIME_HOUR = 'PREFS_STOP_TIME_HOUR';
    static const _KEY_STOP_TIME_MIN  = 'PREFS_STOP_TIME_MIN';
    static const KEY_START_TIME  = 'PREFS_START_TIME';
    static const KEY_STOP_TIME = 'PREFS_STOP_TIME';

    var prefsMap = <String, dynamic>{};

    static Future<Map<String, dynamic>> getAllPrefs() async {
      final SharedPreferences prefs = await SharedPreferences.getInstance();

        int startTimeHour = prefs.getInt(_KEY_START_TIME_HOUR) ?? 6;
        int startTimeMin = prefs.getInt(_KEY_START_TIME_MIN) ?? 0;
        int stopTimeHour = prefs.getInt(_KEY_STOP_TIME_HOUR) ?? 22;
```

```
    int stopTimeMin = prefs.getInt(_KEY_STOP_TIME_MIN) ?? 0;

    return {
    KEY_ENABLE_NOTIFICATIONS : prefs.getBool(KEY_ENABLE_NOTIFICATIONS) ?? false,
    KEY_NOTIF_TIME_INTERVAL  : prefs.getInt(KEY_NOTIF_TIME_INTERVAL) ?? 60,
    KEY_START_TIME        : TimeOfDay(hour: startTimeHour, minute: startTimeMin),
    KEY_STOP_TIME         : TimeOfDay(hour: stopTimeHour, minute: stopTimeMin)
    };
}


static Future<bool> getEnableNotifications() async {
  final SharedPreferences prefs = await SharedPreferences.getInstance();
    return prefs.getBool(KEY_ENABLE_NOTIFICATIONS) ?? false;
}

/// in seconds
static Future<int> getNotificationTimeInterval() async {
  final SharedPreferences prefs = await SharedPreferences.getInstance();
    return prefs.getInt(KEY_NOTIF_TIME_INTERVAL) ?? 60;
}

static Future<TimeOfDay> getStartTime() async {
  final SharedPreferences prefs = await SharedPreferences.getInstance();
    int startTimeHour = prefs.getInt(_KEY_START_TIME_HOUR) ?? 6;
    int startTimeMin = prefs.getInt(_KEY_START_TIME_MIN) ?? 0;
    return TimeOfDay(hour: startTimeHour, minute: startTimeMin);
}

static Future<TimeOfDay> getStopTime() async {
  final SharedPreferences prefs = await SharedPreferences.getInstance();
    int stopTimeHour = prefs.getInt(_KEY_STOP_TIME_HOUR) ?? 22;
    int stopTimeMin = prefs.getInt(_KEY_STOP_TIME_MIN) ?? 0;
    return TimeOfDay(hour: stopTimeHour, minute: stopTimeMin);
}


/// SETTERS
```

```
  static Future<void> setEnableNotifications(bool enableNotifications) async {
      final SharedPreferences prefs = await SharedPreferences.getInstance();
       await prefs.setBool(KEY_ENABLE_NOTIFICATIONS, enableNotifications);
   }

   static Future<void> setNotificationTimeInterval(int timeInterval) async {
      final SharedPreferences prefs = await SharedPreferences.getInstance();
       await prefs.setInt(KEY_NOTIF_TIME_INTERVAL, timeInterval);
   }

   static Future<void> setStartTime(TimeOfDay startTime) async {
      final SharedPreferences prefs = await SharedPreferences.getInstance();
       await prefs.setInt(_KEY_START_TIME_HOUR, startTime.hour);
       await prefs.setInt(_KEY_START_TIME_MIN, startTime.minute);
   }

   static Future<void> setStopTime(TimeOfDay stopTime) async {
      final SharedPreferences prefs = await SharedPreferences.getInstance();
       await prefs.setInt(_KEY_STOP_TIME_HOUR, stopTime.hour);
       await prefs.setInt(_KEY_STOP_TIME_MIN, stopTime.minute);
   }

}
```

85

CupertinoDatePickerMode

- CupertinoDatePicker class
- My *prefs_select_time.dart* class:

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

class PreferencesSelectTime extends StatefulWidget {
    String _title;
    TimeOfDay _timeOfDay;
    Function _updateTimeFunction;

   PreferencesSelectTime(this._title, this._timeOfDay, this._updateTimeFunction);

    @override
   PreferencesSelectTimeState createState() => PreferencesSelectTimeState(_title, _timeOfDay,
}

class PreferencesSelectTimeState extends State<PreferencesSelectTime> {

    String _title;
    TimeOfDay _timeOfDay;
    Function _updateTimeFunction;
   PreferencesSelectTimeState(this._title, this._timeOfDay, this._updateTimeFunction);

    @override
    Widget build(BuildContext context) {

        return Scaffold(
            appBar: AppBar(
                title: Text(_title),
```

```
            ),
            body: FutureBuilder<TimeOfDay>(
                future: SharedPreferencesHelper.getStartTime(),
                builder: (BuildContext context, AsyncSnapshot snapshot) {
                    if (snapshot.connectionState == ConnectionState.done) {
                        return Container(
                            height: MediaQuery.of(context).size.height / 4,
                            child: CupertinoDatePicker(
                                mode: CupertinoDatePickerMode.time,
                      initialDateTime: DateTime(1969, 1, 1, _timeOfDay.hour, _timeOfDay.minute),
                                onDateTimeChanged: (DateTime newDateTime) {
                                var newTod = TimeOfDay.fromDateTime(newDateTime);
                                    _updateTimeFunction(newTod);
                                },
                                use24hFormat: false,
                                minuteInterval: 1,
                            )
                        );
                    } else {
                        return new CircularProgressIndicator();
                    }
                }
            )
        );
    }
}
```

- My final “Time picker” code:

```
return Scaffold(
    appBar: AppBar(
        title: Text(_title),
    ),
    body: FutureBuilder<TimeOfDay>(
        future: SharedPreferencesHelper.getStartTime(),
        builder: (BuildContext context, AsyncSnapshot snapshot) {
            if (snapshot.connectionState == ConnectionState.done) {
                return Container(
                  height: MediaQuery.of(context).copyWith().size.height / 4,
```

```
                    child: CupertinoDatePicker(
                        mode: CupertinoDatePickerMode.time,
                initialDateTime: DateTime(1969, 1, 1, _timeOfDay.hour, _timeOfDay.minute),
                        onDateTimeChanged: (DateTime newDateTime) {
                           var newTod = TimeOfDay.fromDateTime(newDateTime);
                            _updateTimeFunction(newTod);
                        },
                        use24hFormat: false,
                        minuteInterval: 1,
                    )
                );
            } else {
                return new CircularProgressIndicator();
            }
        }
    )
);
```

- More MediaQuery info: How to set the size of a Flutter Container (fit a percentage of the device screen)
  - "`copyWith`: Creates a copy of this media query data but with the given fields replaced with the new values."

```
height: MediaQuery.of(context).size.height / 3
```

86

BottomSheet

- Docs
- Two types:
  - Persistent
  - Modal
- Example:

```
showModalBottomSheet(
    context: context,
    builder: (BuildContext builder) {
        return Container(
            height: MediaQuery.of(context).copyWith().size.height / 4,
        );
    }
)
```

87

BottomSheet, MediaQuery (phone height, width)

- Use it to get the size of the current media (using the current `context`)
- Docs
- Example:

```
Container(
    height: MediaQuery.of(context).size.height / 3,
);
```

- Example:

```
returnContainer(
    height: MediaQuery.of(context).size.height / 4,
    child: CupertinoDatePicker(
        mode: CupertinoDatePickerMode.time,
     initialDateTime: DateTime(1969, 1, 1, _timeOfDay.hour, _timeOfDay.minute),
        onDateTimeChanged: (DateTime newDateTime) {
            var newTod = TimeOfDay.fromDateTime(newDateTime);
            _updateTimeFunction(newTod);
        },
        use24hFormat: false,
        minuteInterval: 1,
    )
);
```

- My article, How to set the size of a Flutter Container (fit a percentage of the device screen)

88

AlertDialog

- Show an alert dialog
- Docs
- Example:

```
Future<void> _showQuoteIsEmptyDialog() async {

  Widget okButton = FlatButton(
      child: Text("OK"),
      onPressed: () {
          Navigator.of(context).pop();
      }
  );

  final _helpText = 'The Quote field is empty. '
    + 'If you want to delete the quote, use the Delete menu item in the upper-right menu.';

  return showDialog<void>(
      context: context,
      barrierDismissible: false,  // must tap a button
      builder: (BuildContext buildContext) {
          return AlertDialog(
              title: Text('Quote field is empty'),
              content: SingleChildScrollView(
                child: ListBody(
                  children: <Widget>[
                    Text(_helpText)
                  ],
                ),
              ),
              actions: [
                okButton
```

```
            ]
          );
      },
  );
}
```

Another example:

```
return showDialog<void>(
  context: context,
  builder: (BuildContext context) {
    return AlertDialog(
      content: Text(
       '${pendingNotificationRequests.length} pending notification requests'),
      actions: [
        FlatButton(
          child: Text('OK'),
          onPressed: () {
            Navigator.of(context).pop();
          },
        ),
      ],
    );
  },
);
```

When you want to call this method:

```
await _showQuoteIsEmptyDialog();
```

More information:

- https://androidkt.com/flutter-alertdialog-example
- https://medium.com/@suragch/making-an-alertdialog-in-flutter-544bc703f758

88.1 How to return a value from an AlertDialog

Create an enum or other data type to handle the possible return values:

```
enum DialogAction { ReturnToThisQuote, ReturnToQuoteList }
```

When you call pop, pass the return value into pop:

```
// note the `<DialogAction` references here also
Future<DialogAction> _showQuoteIsEmptyDialog() async {

    Widget _returnToQuoteButton = FlatButton(
        child: Text("Return to This Quote"),
        onPressed: () {
          Navigator.of(context).pop(DialogAction.ReturnToThisQuote);  //HERE
        }
    );
    Widget _returnToQuoteList = FlatButton(
        child: Text("Return to Quote List"),
        onPressed: () {
          Navigator.of(context).pop(DialogAction.ReturnToQuoteList);  //HERE
        }
    );

    // ...

    return showDialog<DialogAction>(
        context: context,
        barrierDismissible: false,  // must tap a button
        builder: (BuildContext buildContext) {
          return AlertDialog(
              title: Text('Quote field is empty'),
              content: SingleChildScrollView(
                child: ListBody(
                  children: <Widget>[
                    Text(_helpText)
                  ],
                ),
              ),
              actions: [
                _returnToQuoteButton,
                _returnToQuoteList
              ]
```

```
        );
      },
    );
```

89

Snackbar notifications

- Flutter.dev, Display a Snackbar
- The code is relatively simple, but there are tricks/techniques involved in getting a reference to the current Scaffold
- You'll need to read documentation for this one:
  - https://medium.com/@ksheremet/flutter-showing-snackbar-within-the-widget-that-builds-a-scaffold-3a817635aeb2
  - https://api.flutter.dev/flutter/material/Scaffold/of.html
  - https://flutter.dev/docs/cookbook/design/snackbars
- Although this probably won't work as shown (see those articles), the basic code looks like this:

```
// OPTION 1
Scaffold.of(context).showSnackBar(SnackBar(
   content: Text('Sending notification'),
   duration: Duration(seconds: 3),
));


// OPTION 2
final snackBar = SnackBar(
    content: Text('Notification initiated'),
    action: SnackBarAction(
        label: 'Sent!',
        onPressed: () {
            // if you want to handle this
        },
    ),
);
Scaffold.of(context).showSnackBar(snackBar);
```

```
// OPTION 3 - works with the proper setup
_scaffoldKey.currentState.showSnackBar(
  SnackBar(
      content: Text('Sending notification'),
      duration: Duration(seconds: 3),
));
```

90

Navigation/Navigator

- Moving the user from screen to screen is usually easy, just involves:
  - Navigator.push
  - Navigator.pop
- Can also use named paths
- Basic examples:

```
// PUSH
Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => CurrentQuoteWidget()),
);


// PUSH
ListTile(
    title: const Text(
        'About',
        style: prefsTextStyle
    ),
    onTap: () {
        Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => AboutWidget()),
        );
    },
),


// POP
new FlatButton(
```

```
    child: new Text("Close"),
    onPressed: () {
        Navigator.of(context).pop();
    },
),
```

90.1 Using named routes/paths

- Set up named routes:

```
return MaterialApp(

    routes: {
        '/home': (context) => MyApp(),
        '/here': (context) => CurrentQuoteWidget(),
    },

    home: Scaffold(
```

- use named routes:

```
IconButton _homeButton(BuildContext context) {
    return IconButton(
        icon: Icon(Icons.home, color: _buttonColor),
        onPressed: () {
            Navigator.pushNamed(context, '/home');
        }
    );
}
```

90.2 A way to clear the Navigator push/pop stack

- I can confirm that this works as a way to clear the Navigator stack:

```
Navigator.of(context)
    .pushNamedAndRemoveUntil('/here', (Route<dynamic> route) => false);
```

- Can also check to see if you can pop:

```
var _navState = Navigator.of(context);
debugPrint('Navigator-can-pop: ${_navState.canPop()}');
```

91
SQLite

- Use the Flutter sqflite package

91.1 My database_helper.dart class

```
import 'dart:io';
import 'dart:core';
import 'package:flutter/material.dart';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
import 'package:path_provider/path_provider.dart';
import 'data_model.dart';

/// cobbled together from these resources:
/// https://medium.com/@suragch/simple-sqflite-database-example-in-flutter-e56a5aaa3f91 (most
/// https://medium.com/flutter-community/using-sqlite-in-flutter-187c1a82e8b
/// https://www.techiediaries.com/flutter-sqlite-crud-tutorial/
/// https://flutter.dev/docs/cookbook/persistence/sqlite
class DatabaseHelper {

    static final _databaseName = "JustBe.db";
    static final _databaseVersion = 1;

    // make this a singleton class
    DatabaseHelper._privateConstructor();
   static final DatabaseHelper instance = DatabaseHelper._privateConstructor();

    // only have a single app-wide reference (getter) to the database
    static Database _database;
    Future<Database> get database async {  //GETTER
        if (_database != null) return _database;
        // lazily instantiate the db the first time it is accessed
```

```
    _database = await _initDatabase();
    return _database;
}

// this opens the database (and creates it if it doesn't exist)
_initDatabase() async {
 Directory documentsDirectory = await getApplicationDocumentsDirectory();
    String path = join(documentsDirectory.path, _databaseName);
    Database foo = await openDatabase(
        path,
        version: _databaseVersion,
        onOpen: (instance) {},
        onCreate: _onCreate
    );
    return foo;
}

// SQL code to create the database table
Future _onCreate(Database db, int version) async {
    // note that the date/time field is stored as an integer because
    // that's all you can do with SQLite. so this field is the Unix
    // epoch time.
    var createString = '''
        CREATE TABLE quotes (
            id INTEGER PRIMARY KEY,
            quote TEXT UNIQUE,
            author TEXT,
       date_last_touched INTEGER DEFAULT (cast(strftime('%s','now') as int))
        )
    ''';
    await db.execute(createString);

    //TODO do i need `await` before each of these?
 db.execute("insert into quotes (quote, author) values ('Be happy in the moment, that's enough
 db.execute("insert into quotes (quote, author) values ('Be here now.', 'Ram Dass')");
 db.execute("insert into quotes (quote, author) values ('Do every act of your life as though i

    // make the author field == ''
 db.execute("insert into quotes (quote, author) values ('Breath in, relax. Breathe out, relea
```

```
  db.execute("insert into quotes (quote, author) values ('Go through your day as if you were

    // skip the 'author' field, making it null
    db.execute("insert into quotes (quote) values ('Just breathe')");
  db.execute("insert into quotes (quote) values ('Walk mindfully. Be aware of your steps, and
}


// --------------
// Helper methods
// --------------


// use this INSERT approach so you don't have to worry about
// characters like ' and " in the Quote
Future<int> insertQuote(Quote quote) async {
    Database db = await instance.database;
    var row = {
        'quote'  : quote.quote,
        'author' : quote.author
    };
    int id = await db.insert('quotes', row);
    return id;
}

// OLD/ORIGINAL (BAD) APPROACH
Future<int> insertQuote(Quote quote) async {
    Database db = await instance.database;
    // var res = await db.insert("Client", newClient.toMap());
  // skip the `id` field to let it auto-increment (same with the date/time field)
    var result = await db.rawInsert(
        '''
        INSERT INTO quotes (quote, author)
        VALUES ('${quote.quote}', '${quote.author}')
        '''
    );
    return result;
}
```

```
Future<List<Quote>> selectAllQuotes() async {
    Database db = await instance.database;
  var result = await db.rawQuery('SELECT * FROM quotes ORDER BY date_last_touched DESC');
    var quotes = result.map((qAsMap) => Quote.fromMap(qAsMap));
    return quotes.toList();
}

Future<List<int>> selectAllQuoteIds() async {
    Database db = await instance.database;
    var result = await db.rawQuery('SELECT id FROM quotes');
    var ids = result.map((qAsMap) => _getIdFromMap(qAsMap));
    return ids.toList();
}

int _getIdFromMap(Map<String, dynamic> qMap) {
    return qMap['id'];
}

/// Can return `null`.
Future<Quote> selectQuote(int id) async {
    Database db = await instance.database;
   var results = await db.rawQuery('SELECT * FROM quotes WHERE id = $id');
    if (results.length > 0) {
        return Quote.fromMap(results.first);
    }
    return null;
}

/// returns the number of rows update.
/// new, correct approach.
Future<int> updateQuote(Quote quote) async {
    Database db = await instance.database;

    var row = {
        'id'     : quote.id,
        'quote'  : quote.quote,
        'author' : quote.author
    };
```

```
        // returns num rows updated
        return await db.update(
            'quotes',
            row,
            where: 'id = ?',
            whereArgs: [quote.id]
        );
    }

    /// returns the number of rows update.
    /// OLD/BAD approach that doesn't work with ' and ".
    Future<int> updateQuote(Quote quote) async {
        Database db = await instance.database;
        return await db.rawUpdate(
            '''
            UPDATE quotes
            SET quote = '${quote.quote}',
                author = '${quote.author}',
                date_last_touched = (cast(strftime('%s','now') as int))
            WHERE id = ${quote.id}
            '''
        );
    }

    Future<int> deleteQuote(int id) async {
        Database db = await instance.database;
        return await db.rawDelete('DELETE FROM quotes WHERE id = $id');
    }

    // TODO needed???
    // await database.close();

    Future<int> rowCount() async {
        debugPrint('ROW COUNT CALLED');
        Database db = await instance.database;
      return Sqflite.firstIntValue(await db.rawQuery('SELECT COUNT(1) FROM quotes'));
    }

    // Future<int> insert(Map<String, dynamic> row) async {
```

```
    //     Database db = await instance.database;
    //     return await db.insert(table, row);
    // }

    // Future<List<Map<String, dynamic>>> queryAllRows() async {
    //     Database db = await instance.database;
    //     return await db.query(table);
    // }

    // Future<int> queryRowCount() async {
    //     Database db = await instance.database;
   //    return Sqflite.firstIntValue(await db.rawQuery('SELECT COUNT(*) FROM $table'));
    // }

    // Future<int> update(Map<String, dynamic> row) async {
    //     Database db = await instance.database;
    //     int id = row[columnId];
   //    return await db.update(table, row, where: '$columnId = ?', whereArgs: [id]);
    // }

    // Future<int> delete(int id) async {
    //     Database db = await instance.database;
   //    return await db.delete(table, where: '$columnId = ?', whereArgs: [id]);
    // }

}
```

- More details:
  - https://alvinalexander.com/flutter/sqflite-escaping-quotes-sql-insert-update-statements

91.2 Finding and working with the SQLite files on Android

```
adb shell ls data/data
adb shell ls data/data/com.example.just_be
    app_flutter
    cache
    code_cache
```

```
    files
    lib
```

More:

```
> adb shell ls data/data/com.example.just_be/app_flutter
JustBe.db
JustBe.db-journal
flutter_assets
```

Once I found my app directory, this command worked to copy the SQLite database to my computer, from the Android emulator:

```
> adb pull data/data/com.example.just_be/app_flutter/JustBe.db .
data/data/com.example.just_be/app_flutter/JustBe.db: 1 file pulled. 6.6 MB/s (20480 bytes in 0.
```

91.2.1 SQLite commands

- The SQLite filename is *JustBe.db*
- I used these commands to look at my database and its one table:

```
# connect/open the sqlite database
sqlite3 JustBe.db

# list tables (mine is named quotes)
sqlite> .tables
android_metadata  quotes

# look at the quotes schema
sqlite> .schema quotes
CREATE TABLE quotes (
    _id INTEGER PRIMARY KEY,
    quote TEXT UNIQUE,
    author TEXT
);

# show the database in the quotes table
sqlite> select * from quotes;
```

92

TimeOfDay (a Flutter class, not a Dart class)

- A value representing a time during the day
- The time is represented by hour and minute pair. Once created, both values cannot be changed.
- TimeOfDay class:
- Example:

```
initialTime: TimeOfDay.fromDateTime(currentValue ?? DateTime.now())
```

- `parse` examples:

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

TimeOfDay now = TimeOfDay.now();
TimeOfDay threePm = TimeOfDay(hour: 15, minute: 0); // 3:00pm
TimeOfDay tod = TimeOfDay.fromDateTime(DateTime.parse('2019-10-04 19:30:00Z'));

_timePickerWidget('Start Time (hh:mm)', DateTime.parse('2019-09-20 06:00:00')),
_timePickerWidget('Stop Time (hh:mm)', DateTime.parse('2019-09-20 22:00:00')),
```

- Convert TimeOfDay to DateTime:

```
// you already have some `TimeOfDay tod`

// Option 1: if “today” doesn’t matter
final dt = DateTime(1969, 1, 1, tod.hour, tod.minute);

// Option 2: if “today” matters
final now = DateTime.now();
final dt = DateTime(now.year, now.month, now.day, tod.hour, tod.minute);
```

93

AnimatedContainer

- Animates multiple attributes simultaneously
  - color
  - height
  - width
- Animates from whatever the previous values were to what they are now
- Especially noticeable when you rotate your device and it animates from portrait to landscape
- Example:

```
AnimatedContainer(
    duration: Duration(milliseconds: _animationDurationMs),
    color: b ? _blue1 : _blue2,
    width: _width,
    height: _height * 0.8,
    padding: EdgeInsets.all(40.0),
    child: Row(
        children: <Widget>[
            Expanded( // Constrains AutoSizeText to the width of the Row
                child: _getSizedText(_quoteText)
            ),
        ],
    )
),
```

94

AnimatedOpacity

- Flutter cookbook doc
- Example:

```
AnimatedOpacity(
    // If the widget is visible, animate to 0.0 (invisible).
    // If the widget is hidden, animate to 1.0 (fully visible).
    opacity: _visible ? 1.0 : 0.0,
    duration: Duration(milliseconds: 500),
    // The green box must be a child of the AnimatedOpacity widget.
    child: Container(
        width: 200.0,
        height: 200.0,
        color: Colors.green,
    ),
);
```

- Example:

```
return SafeArea(
    child: Center(
      child: Column(
        children: <Widget>[
          AnimatedContainer(
              duration: Duration(milliseconds: _animationDurationMs),
              color: b ? _blue1 : _blue2,
              width: _width,
              height: _height * 0.8,
              padding: EdgeInsets.all(40.0),
              child: Row(
                children: <Widget>[
                 Expanded( // Constrains AutoSizeText to the width of the Row
```

```
                  child: _getSizedText(_quoteText)
                ),
              ],
            )
        ),

        Row(
          children: [
            AnimatedContainer(
              duration: Duration(milliseconds: _animationDurationMs),
              color: b ? _blue1 : _blue2,
              height: _height * 0.1,
              width: _width,
              child: Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  _homeButton(context),
                  _refreshButton(context)
                ]
              ),
            ),
          ],
        )

      ],
    )
  )
);
```

- Example:

```
child: Row(
  children: <Widget>[
    Expanded( // Constrains AutoSizeText to the width of the Row
      child: AnimatedOpacity(
        opacity: b ? 0.8 : 1.0,
        curve: Curves.decelerate,
        duration: Duration(milliseconds: 1500),
        child: AutoSizeText(
```

```
          _quoteText,
          maxLines: 10,
          style: TextStyle(
            color: b ? Colors.white60 : _textColor,
            fontSize: 40,
            height: 1.4
          ),
        ),
      )
    ),
  ],
)
```

95

FadeTransition

- TODO: discussion

```
return SafeArea(
  child: FadeTransition(
    opacity: _animation,
    child: Center(
      child: Column(
        children: <Widget>[
          AnimatedContainer(
              duration: Duration(milliseconds: _animationDurationMs),
              color: b ? _blue1 : _blue2,
              width: _width,
              height: _height * 0.8,
              padding: EdgeInsets.all(40.0),
              child: Row(
                children: <Widget>[
                Expanded( // Constrains AutoSizeText to the width of the Row
                    child: _getSizedText(_quoteText)
                  ),
                ],
              )
          ),

          Row(
            children: [
              AnimatedContainer(
                duration: Duration(milliseconds: _animationDurationMs),
                color: b ? _blue1 : _blue2,
                height: _height * 0.1,
                width: _width,
                child: Row(
                  mainAxisAlignment: MainAxisAlignment.center,
```

```
                children: [
                  _homeButton(context),
                  _refreshButton(context)
                ]
              ),
            ),
          ],
        )

      ],
    )
  )
  )
);
```

96

FadeIn

- TODO

97
HTML Widget

- There's a flutter_html library/package:

```
flutter_html: ^0.11.0
```

Here's some of my code:

```
import 'package:flutter/material.dart';
import 'package:flutter_html/flutter_html.dart';
import 'package:html/dom.dart' as dom;
import 'about_data.dart';
import 'package:url_launcher/url_launcher.dart';

class AboutWidget extends StatelessWidget {

    AboutWidget({Key key, title}) : super(key: key);

    final title = 'About';
    final _aboutText = AboutData.aboutString;

    _launchURL(String url) async {
        if (await canLaunch(url)) {
            await launch(url);
        } else {
            throw 'Could not launch $url';
        }
    }

    @override
    Widget build(BuildContext context) {

        //final hColor = Theme.of(context).splashColor;
        final hColor = Colors.grey[700];
```

```
final h1TextStyle = TextStyle(
    fontSize: 20,
    fontWeight: FontWeight.bold,
    color: hColor,
    height: 2
  );
final h2TextStyle = TextStyle(
    fontSize: 20,
    color: hColor,
    height: 2
);
final h3TextStyle = TextStyle(fontSize: 20, color:  hColor);
final brTextStyle = TextStyle(height: 0.8);
final pTextStyle = TextStyle(fontSize: 16, color: Colors.black54);
final liTextStyle = TextStyle(
    fontSize: 16,
    color: Colors.black54,
    height: 1.2
);


return new Scaffold(
    appBar: new AppBar(
        title: new Text(title),
    ),
    body:
      Center(
        child: SingleChildScrollView(
          padding: EdgeInsets.fromLTRB(15, 15, 15, 40),

          child: Html(
              data: _aboutText,
              customTextAlign: (dom.Node node) {
                  if (node is dom.Element) {
                      switch (node.localName) {
                          case "p": return TextAlign.justify;
                      }
                  }
                return null;
```

```
                    },
                    onLinkTap: (url) {
                        print("Opening $url...");
                        _launchURL(url);
                    },
                   customTextStyle: (dom.Node node, TextStyle baseStyle) {
                        if (node is dom.Element) {
                            switch (node.localName) {
                           case "br": return baseStyle.merge(brTextStyle);
                             case "p": return baseStyle.merge(pTextStyle);
                           case "li": return baseStyle.merge(liTextStyle);
                           case "h1": return baseStyle.merge(h1TextStyle);
                           case "h2": return baseStyle.merge(h2TextStyle);
                           case "h3": return baseStyle.merge(h3TextStyle);
                            }
                        }
                      return baseStyle;
                    },
                )
              )
            )
          );
    }
}
```

98
Device Rotation

I have code like this to handle the case where my app is “resumed”:

```
class MyHomePageState extends State<MyHomePage> with WidgetsBindingObserver {

    AppLifecycleState _lastLifecycleState;

    ...

    @override
    initState() {
        super.initState();
        WidgetsBinding.instance.addObserver(this);
    }

    ...

   /// https://flutter.dev/docs/get-started/flutter-for/android-devs#how-do-i-listen-to-andro
   /// https://api.flutter.dev/flutter/widgets/WidgetsBindingObserver-class.html
    /// https://api.flutter.dev/flutter/dart-ui/AppLifecycleState-class.html
    @override
    void didChangeAppLifecycleState(AppLifecycleState state) {
        setState(() {
            _lastLifecycleState = state;
        });

        /// show a random quote when the application is resumed
        if (_lastLifecycleState == AppLifecycleState.resumed) {
            Utils.getRandomQuote(_dbHelper).then((quote) {

                Navigator.push(
                    context,
              MaterialPageRoute(builder: (context) => CurrentQuoteWidget()),
```

```
            );

            // THIS ALSO WORKS
            //_showDialog(quote.quote);

            _lastLifecycleState = null;
        });
    }

    //AppLifecycleState.inactive
    //AppLifecycleState.paused
    //AppLifecycleState.resumed
    //AppLifecycleState.suspending
}

...
```

I had this listed as a potentially-good URL:

- https://stephenmann.io/post/listening-to-device-rotations-in-flutter

99

Rename the App

- If you don't properly specify your app information when you create your project, you'll need to rename the app later
- I'm referring to the "app icon name" here
    - With Apple, you set the AppStore name in App Store Connect
- iOS
    - vi ./ios/Runner/Info.plist
    - change `CFBundleName` from `just_be` to `Just Be` (confirmed, this works on iOS)

```
<key>CFBundleName</key>
<string>Just Be</string>
```

- Android
    - vi android/app/src/main/AndroidManifest.xml
    - set the name using `android:label` (confirmed, this works on Android)

```
<application
    android:name="io.flutter.app.FlutterApplication"
    android:label="Just Be"
    ... />
```

99.1 How to change the application package name

- NOTE: You can specify your package name when you run `flutter create`

```
flutter create --org com.valleyprogramming just_be
```

- Android:
  - change:
    * *main/java/com/example…* directory to be what you want
    * modify package name in MainActivity.java
    * package name in *src/main/AndroidManifest.xml*
    * package name in *src/debug/AndroidManifest.xml*
    * `applicationId` name in build.gradle
- iOS
  - You'll find `CFBundleIdentifier` in *ios/Runner/Info.plist*:

```
<key>CFBundleIdentifier</key>
<string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>

- `PRODUCT_BUNDLE_IDENTIFIER` is set in several places in *ios/Runner.xcodeproj/project.pbxproj*
- change all those occurrences (currently three places)
```

- After making those changes:
  - run `flutter clean`
  - run `flutter run`

99.1.1 To confirm on Android

- start the emulator
- make sure the old app is deleted
- run `flutter clean`
- run `flutter run`
- adb shell ls data/data
  - should show this as part of its result:

```
com.valleyprogramming.just_be
```

Note that to run `adb shell` I have to stop VS Code. The error message is:

```
error: could not install *smartsocket* listener: Address already in use
ADB server didn't ACK
* failed to start daemon *
error: cannot connect to daemon
```

100

How to set an app icon

- Put icons in *assets/app_icon* or similar
- My app is named *Just Be*, so you'll see *just_be* in the names below
- Add this stuff to *pubspec.yaml*:

```
dev_dependencies:
    flutter_launcher_icons: "^0.7.3"

flutter_icons:
    ios: true
    android: true
    image_path_ios: "assets/app_icon/justbe_1024px.png"
    image_path_android: "assets/app_icon/justbe_1024px.png"
    #adaptive_icon_background: "assets/app_icon/background.png"
    #adaptive_icon_foreground: "assets/app_icon/foreground.png"
```

- The *flutter_launcher_icons* project is what makes this work
- Now run these commands at the command line:

```
flutter packages get
flutter packages pub run flutter_launcher_icons:main
```

101

Android manifest

- TODO: Add more information
- If you change the AndroidManifest.xml file, you may need to completely reinstall the app on your emulator/simulator
- There are two *AndroidManifest.xml* files:
  - ./android/app/src/profile/AndroidManifest.xml
  - ./android/app/src/main/AndroidManifest.xml

102
Android Debugging

- Setup:

```
PATH=/Users/al/bin/android-platform-tools:$PATH
```

- adb shell:

```
$ adb shell ls

adb server version (40) doesn't match this client (39); killing...
error: could not install *smartsocket* listener: Address already in use
ADB server didn't ACK
* failed to start daemon *
error: cannot connect to daemon
```

- I had to stop VS Code to get this to work

```
adb shell ls data/data
adb shell ls data/data/com.example.just_be
    app_flutter
    cache
    code_cache
    files
    lib
```

More:

```
> adb shell ls data/data/com.example.just_be/app_flutter

JustBe.db
JustBe.db-journal
flutter_assets
res_timestamp-1-1568224752182
```

Once I found the app directory, this command worked to copy the SQLite database to my computer, from the Android emulator:

```
> adb pull data/data/com.example.just_be/app_flutter/JustBe.db .
data/data/com.example.just_be/app_flutter/JustBe.db: 1 file pulled. 6.6 MB/s (20480 bytes in 0.00
```

102.1 SQLite commands

- The SQLite filename is *JustBe.db*
- I used these commands to look at my database and its one table:

```
# connect/open the sqlite database
sqlite3 JustBe.db

# list tables (mine is named quotes)
sqlite> .tables
android_metadata  quotes

# look at the quotes schema
sqlite> .schema quotes
CREATE TABLE quotes (
    _id INTEGER PRIMARY KEY,
    quote TEXT UNIQUE,
    author TEXT
);

# show the database in the quotes table
sqlite> select * from quotes;
```

103

Notifications (Local)

- Note: This chapter needs a lot of work
- TODO: organize this information
- Resources:
  - https://pub.dev/packages/flutter_local_notifications
  - https://github.com/MaikuB/flutter_local_notifications
  - https://medium.com/@nitishk72/flutter-local-notification-1e43a353877b
  - https://inducesmile.com/google-flutter/how-to-send-foreground-push-notifications-in-flutter/
- Some example code:

```
var minutePrior = int.parse(mins) - 1;

var platformChannelSpecifics = NotificationDetails(
    androidPlatformChannelSpecifics,
    iOSPlatformChannelSpecifics
);


// first attempt at scheduling something
var scheduledNotificationDateTime = DateTime
    .now()
    .add(new Duration(minutes: 1));

await flutterLocalNotificationsPlugin.schedule(
    0,
    'Thich Nhat Hanh',
    'Walk as if you are kissing the Earth with your feet.',
    scheduledNotificationDateTime,
    payload: 'Default_Sound',
```

```
);
```

103.1 Options I Can Use

- schedule
  - schedule at a delta-t from now (now+5, now+10, etc.)
- periodicallyShow
  - this is limited because of the limits of RepeatInterval
- showDailyAtTime
  - looks like an option

103.2 flutter_local_notifications project

- Project: https://github.com/MaikuB/flutter_local_notifications
- On Android, you'll need to know about channels:
  - https://developer.android.com/training/notify-user/channels
- Example AndroidManifest.xml config:
  - https://github.com/MaikuB/flutter_local_notifications/blob/master/example/android/app/src/
- some of my code:

```
var androidPlatformChannelSpecifics = AndroidNotificationDetails(
    'just_be_channel',
    'Just Be',
    'Mindfulness reminders',
    importance: Importance.Default,
    priority: Priority.Default
);
var iOSPlatformChannelSpecifics = IOSNotificationDetails();
var platformChannelSpecifics = NotificationDetails(
    androidPlatformChannelSpecifics,
```

```
    iOSPlatformChannelSpecifics
);

// works
// await flutterLocalNotificationsPlugin.show(
//     0,
//     'Thich Nhat Hanh',
//     'Walk as if you are kissing the Earth with your feet.',
//     platformChannelSpecifics,
//     payload: 'Default_Sound',
// );

// send a notification in one minute
var scheduledNotificationDateTime = DateTime
    .now()
    .add(new Duration(minutes: 1));

await flutterLocalNotificationsPlugin.schedule(
    0,
    'Thich Nhat Hanh',
    'Walk as if you are kissing the Earth with your feet.',
    scheduledNotificationDateTime,
    platformChannelSpecifics,
    payload: 'Default_Sound',
);
```

103.3 Android channels (need to know this for flutter_local_notifications)

- Android 8.0 (Oreo, API 26) and newer
- https://developer.android.com/training/notify-user/channels
- notification channels (also called Categories in the UI)
    - collections of associated notifications
    - every notification needs channel, or it won't show up
    - one app can have multiple channels
    - see 1:15 of this Google video
    - on the phone on that video, it looks like a channel can have:

    * a title
    * something that shows “make sound”, “pop on screen”

- to set up a notification channel:

```
// create channel object with the unique id FOLLOWERS_CHANNEL
NotificationChannel followersChannel = new NotificationChannel(
    FOLLOWERS_CHANNEL_ID,                   // unique channel id
    "Followers",                            // user-visible name
    NotificationManager.IMPORTANCE_DEFAULT   // importance level
);

// configure the channel's initial settings
followersChannel.setLightColor(Color.GREEN);

// submit the notification channel object to the notification manager
NotificationManager nm = (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
nm.createNotificationChannel(followersChannel);
```

- later:

```
Notification n = new Notification.Builder(MainActivity.this)
        .setContentTitle(title)
        .setContentText(message)
        .setSmallIcon(R.drawablw.notification_icon)
        .setChannelId(FOLLOWERS_CHANNEL)  //must do this, or get error
        .build();

// issue the notification
mNotificationManager.notify(
    notificationId,
    notification
);
```

- Oreo also introduced Notification Badges, also known as dots
- shows Facebook-like badges (dots) on your app icon
- users can see a count of outstanding notifications, and can swipe through them
- can control when to show badges:

```
// turn off badges for channel
mChannel.setShowBadge(false);
```

- can control display of number of outstanding notifications:

```
// change number of outstanding notifications for channel
Notification n = new Notification.Builder(MainActivity.this)
        .setNumber(messageCount)
        // other options
```

- there are other features (see the docs):
  - `enableLights()`, `setLightColor()`, and `setVibrationPattern()`
  - once you create the channel you can't change these settings, and the user has final control of them
- can also set up a "channel group," but I don't need that
- notes:
  - notification channels APIs are not available in the support library
- current notification code (in *main.dart*):

```
futureListOfQuotes.then((quotes) {
    var q = ListUtils.getRandomElement<Quote>(quotes);
   //TODO there used to be an 'await' before calling 'schedule'; is it needed?
    flutterLocalNotificationsPlugin.schedule(
        0,
        q.author,
        q.quote,
        scheduledNotificationDateTime,
        platformChannelSpecifics,
        payload: 'Default_Sound',
    );
});
```

104

Going Live on Android

- Main URL:
  - https://flutter.dev/docs/deployment/android
  - The following text follows the contents of that URL, adds a few notes to it
- Note: Remember that you can use this command to test on a physical Android device:
  - `flutter run --profile`

104.1 The process

104.1.1 Creating a keystore

This is the input/output from using the keytool command for the Android version of the app. The storePassword and keyPassword are the same that I used on the Motify app.

See this URL for details: https://flutter.dev/docs/deployment/android

```
> keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key

Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Alvin Alexander
What is the name of your organizational unit?
  [Unknown]:  Software Development
What is the name of your organization?
  [Unknown]:  Valley Programming
What is the name of your City or Locality?
  [Unknown]:  Broomfield
```

```
What is the name of your State or Province?
  [Unknown]:  CO
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Alvin Alexander, OU=Software Development, O=Valley Programming, L=Broomfield, ST=CO, C=US c
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10
     for: CN=Alvin Alexander, OU=Software Development, O=Valley Programming, L=Broomfield, ST=CO
Enter key password for <key>
        (RETURN if same as keystore password):
Re-enter new password:
[Storing /Users/al/key.jks]

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an indus
```

104.1.2 Reference the keystore from the app

These notes are directly from the flutter.dev Android URL.

Create a file named /android/key.properties that contains a reference to your keystore:

```
storePassword=<password from previous step>
keyPassword=<password from previous step>
keyAlias=key
storeFile=<location of the key store file, such as /Users/<username>/key.jks>
```

- Warning: Keep the *key.properties* file private; do not check it into public source control

104.1.3 Configure signing in gradle

- Edit the *android/app/build.gradle* file (see the URL)

104.1.4 Use ProGuard (optional)

104.1.5 Review AndroidManifest.xml

- vi *android/app/src/main/AndroidManifest.xml*
  - set the app name: `android:label="Back to Now"`
  - review `uses-permission`

104.1.6 Review the build configuration

- IMPORTANT: Set the build number in *pubspec.yaml*
  - 1.0.0+1 #1st build
  - 1.0.0+2 #2nd build
  - 1.0.0+3 #3rd build
- vi android/app/build.gradle
  - applicationId
  - compileSdkVersion
  - minSdkVersion
  - targetSdkVersion
  - flutterVersionCode = '1' #set in pubspec.yaml
  - flutterVersionName = '1.0.0' #set in pubspec.yaml

104.2 Build the app for release

104.2.1 Build an app bundle

- command:

```
flutter build appbundle
```

- output goes to:
  - *build/app/outputs/bundle/release/app.aab*

104.2.2 Test the app bundle

Two ways:

- Use `bundletool`, build an APK, test it locally
- Test through the Google Play Store

104.2.3 Build an APK

- can also build an APK

104.3 Create the app on the Google Play Store

- do all the usual things on the Play Store
- URL: https://play.google.com/apps

104.3.1 Store listing

- Title: Back To Now
- Short Description
- Full Description
- Graphic Assets
  - Hi-res icon (512x512 px, PNG)
  - Screenshots
  - Feature Graphic (person meditating, Sun, orange sky)
- Don't forget to make the app available for all countries; it does not default to this

105

Going Live on iOS

- Most everything you need:
  - https://flutter.dev/docs/deployment/ios

105.1 A step-by-step example

105.1.1 Step 1

```
flutter build ios
```

105.1.2 Step 2

Then in Xcode, configure the app version and build:

- In Xcode, open Runner.xcworkspace in your app's ios folder.
- Select Product > Scheme > Runner.
- Select Product > Destination > Generic iOS Device.
- Select Runner in the Xcode project navigator, then select the Runner target in the settings view sidebar.
- In the Identity section, update the Version to the user-facing version number you wish to publish
  - 1.0.0 for Just Be
- In the Identity section, update the Build identifier to a unique build number used to track this build on App Store Connect. Each upload requires a unique build number.
  - Initially 1, then 2, etc.

- Very important!:
  - Don't change the *Display Name*. It must be "Runner" for `flutter run` to continue working
  - Display Name is *not* the same as your app/icon name

105.1.3 Step 3

Finally, create a build archive and upload it to App Store Connect:

- Select Product > Archive to produce a build archive.
  - Runs 178 tasks to build an archive
  - Opens a new window/dialog after the archive is built (shows "Archive Information" on the right side of the dialog)
    * TODO: IMAGE: xcode-product-archive-dialog.jpg
    * note: you can end up playing "hide and seek" with this dialog. if you can't find it, run the build again (i don't see how to get to it from the menu system)
- In this window, make sure your current build is selected
- Click the *Validate App* button. If any issues are reported, address them and produce another build. You can reuse the same build ID until you upload an archive.
  - Runs a small wizard
    * Automatically manage signing
    * Review Runner.ipa content (TODO: IMAGE: review-runner.ipa-content.jpg)
    * Includes App.framework and Flutter.framework (which is why you will find *Info.plist* files in your directory related to App and Flutter)
  - Click Validate at the end of the wizard
    * Hopefully you'll see Success at the end (TODO: IMAGE: app-runner-successfully-validated.jpg)
- After the archive has been successfully validated, click *Distribute App*.
  - Select method of distribution:

        * App Store Connect (SELECT THIS)
        * Other options are:
            · Ad Hoc (install on designated devices)
            · Enterprise (distribute to your organization)
            · Development (distribute to members of your team)
    - Select a destination:
        * Upload (send app to App Store Connect) (SELECT THIS)
        * Export (sign and export without uploading)
    - Something about "app symbols" for debugging
    - Re-sign "Runner"
        * Needs to be re-signed for App Store Connect distribution
        * Automatically manage signing
    - Another opportunity to review everything. If it's all good, click *Upload*
        * Hopefully you'll see TODO: IMAGE: uploading-app-success.jpg

105.1.4 Step 4

After this you'll get some emails. You can also follow the status in your app's detail page on App Store Connect. Once your app goes through the initial acceptance process you can manage it there, such as by making it available through TestFlight.

105.2 Flutter iOS app submission issue warning: Missing Push Notification Entitlement

You'll get this error message when your app is submitted to the Apple App Store:

```
Dear Developer,

We identified one or more issues with a recent delivery for your app, "Back to Now" 1.0.0 (3). You

ITMS-90078: Missing Push Notification Entitlement - Your app appears to register with the Apple P

For more information, see https://developer.apple.com/library/content/documentation/Networkin

After you've corrected the issues, you can use Xcode or Application Loader to upload a new binary
```

```
Best regards,

The App Store Team
```

This is a known Flutter problem/issue:

- SO disucssion
- Flutter discussion
- Hack/solution

106
The End

As I mentioned at the beginning of this book, at the moment this book is pretty much just a large cheat sheet. That being said, I hope some things in here are helpful, as some of the examples demonstrate things you won't find elsewhere.

106.1 My other (complete) books

Other books I have written (and completed) include:

- Scala Cookbook
- Functional Programming, Simplified
- How I Sold My Business: A Personal Diary
- A Survival Guide for New Consultants

I also wrote the initial version of *Scala Book*, which is on the docs.scala-lang.org website.

All the best,
Alvin Alexander
alvinalexander.com