FLUTTER FOR

Beginners

PLATFORM: ▷

STARTED ON:
4th July 2022

TUTOR: Netninja

FINISHED ON:
11th July 2022 5:50PM

DURATION: 35 Videos Playlist

COURSE OUTLINE

LECTURE 1 : Intro & Setup
2 : Flutter Overview
3 : Dart Primer
4 : Creating a Flutter App in Android Studio
5 : ~~Colors & Fonts~~ Scaffold & AppBar Widgets
6 : Colors & Fonts
7 : Stateless Widgets & Hot Reload
8 : Image & Assets
9 : Buttons & Icons
10 : Containers & Padding
11 : Rows
12 : Columns
13 : Flutter Outline & Shortcuts
14 : Expanded Widgets
15 : Ninja 1D Project
16 : Stateful Widgets
17 : Lists of Data
18 : Custom Classes
19 : Cards
20 : Extracting Widgets

← : Flip to Continue

21 : Functions as Parameters
22 : Starting the World Time App
23 : Maps & Routing
24 : Widget Life Cycle
25 : Asynchronous Code
26 : Flutter Packages
27 : World Time API
28 : World Time Custom Class
29 : Error Handling
30 : Passing Route Data
31 : Formatting & Showing Dates
32 : Loaders Spinners
33 : Ternary Operators
34 : List View Builder
35 : Updating the Time
DONE

Intro & Setup #1

What is Flutter?

- Mobile UI Framework for creating native apps for IOS & Android.

= Now these apps that we create are all going to get access to platform API's such as the camera, microphone e.t.c.

- The apps we create are all going to use a Single Code-base (dart) means we only have to write our app once for multiple devices

Why Use Flutter?

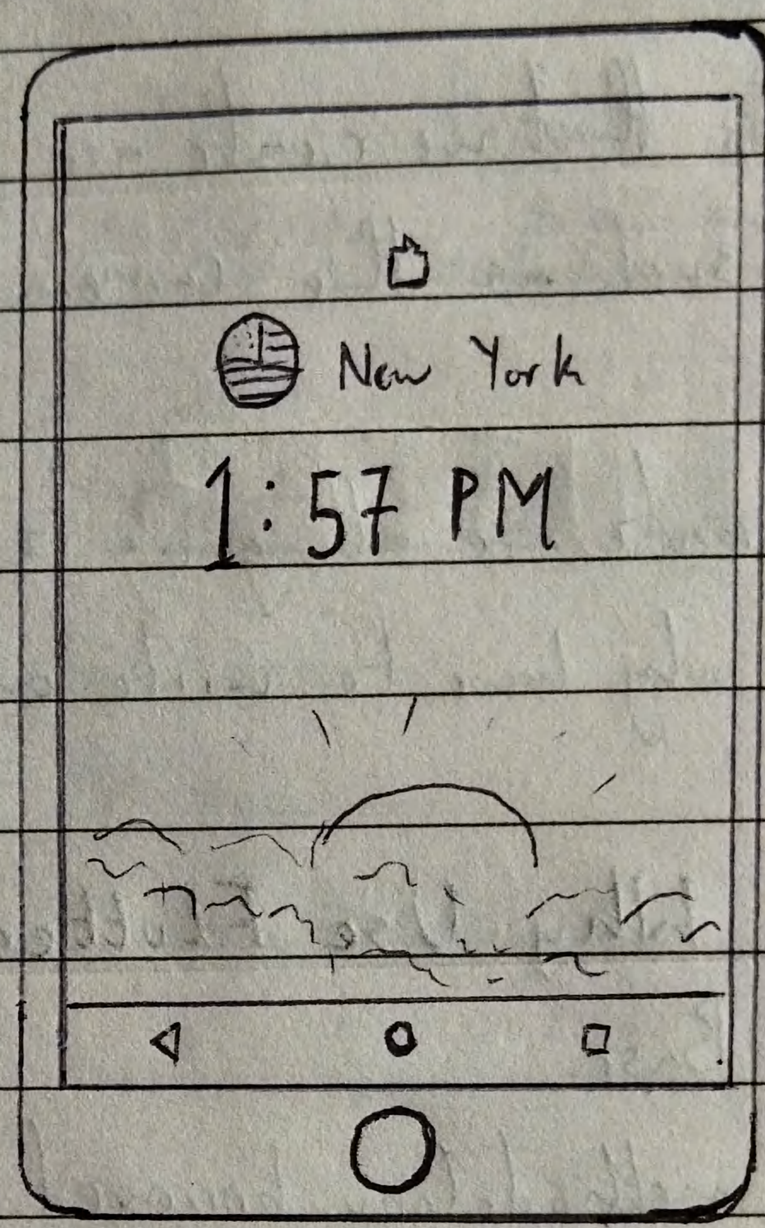
- Only 1 Code Base.
- Good layout methodology borrowed from responsive web.
- Very smooth and quick experience when running apps.
- Works well with Firebase as a backend.
- Uses Dart, Which is a really easy language to pick up.
- Uses Material Design out-of-the-box.
- Great docs & guides on the Flutter website.

OVERVIEW

These Series is going to be split up to 2 different halfs

1ST Half - We will learn all of the basic of Flutter and Create a couple of really simply apps.

2ND HALF - We will put together everything we've learnt plus a few extra pieces and then we are going to create an app that looks something like this



This App is one that tells you the time in different locations and different cities around the world.

You Should Know...

« A basic to good understanding of programming principles

- Classes
- Functions & methods
- Variables
- Asynchronous code

DOWNLOAD & INSTALL

Android Studio - This is the coding environment that we will be using. Not necessary, but if you want to follow along exactly. Use the below link to get it

developer.android.com/studio

Git - This is a version control system. we use it for tracking changes to our code. We need this on our system because Flutter depends on it and we use it to Install Flutter.

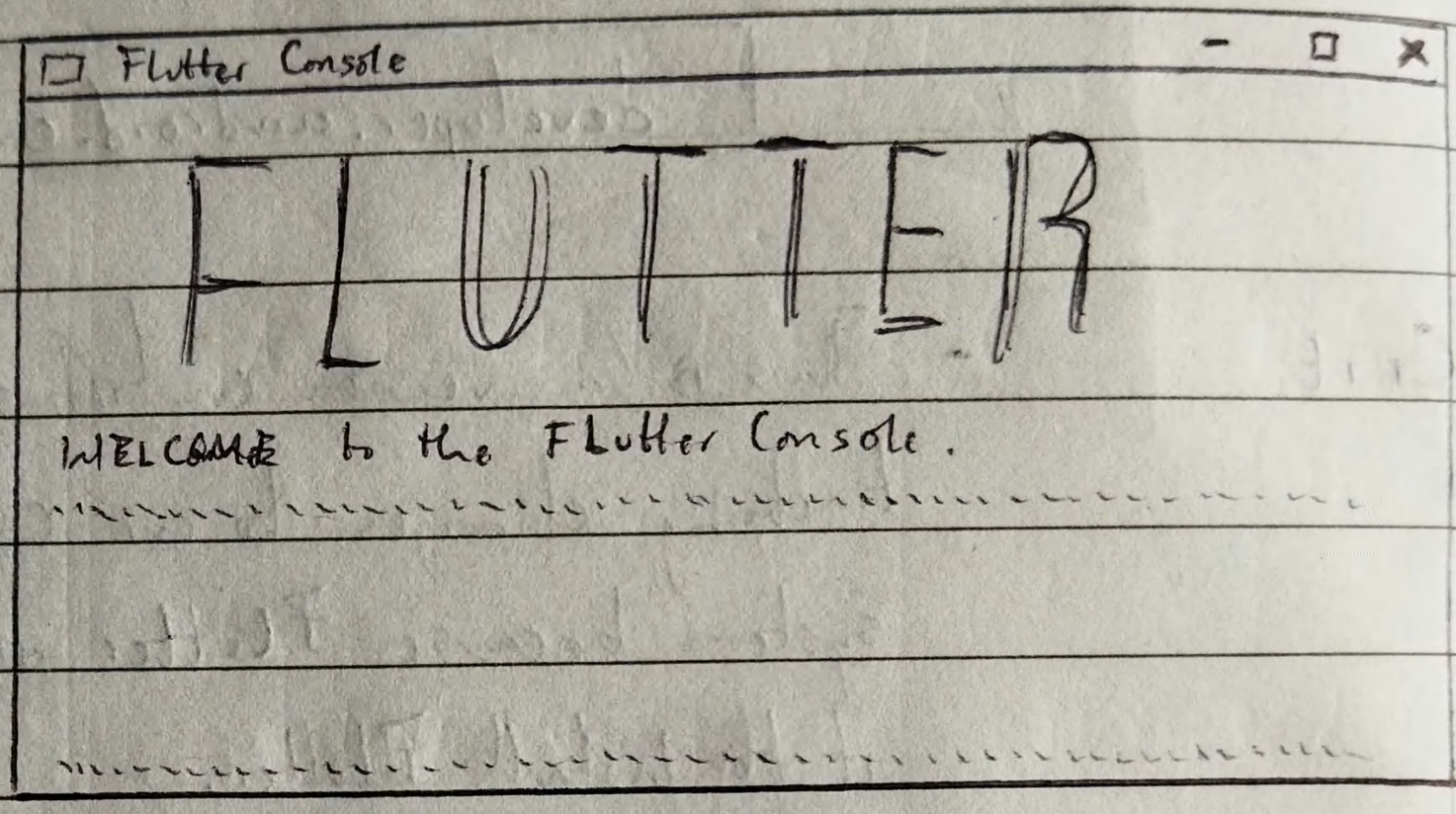
git-scm.com/downloads

Flutter - Finally the big dog. Download Flutter using the link below for whichever operating system you are on

flutter.dev/docs/get-started/install/windows

- ☐ Create [src] folder in your C:\ drive
- ☐ Copy the [git clone -b stable ...] to clone and Install flutter into your windows
- ☐ Open cmd and then navigate to created [src] folder

- Once Installed, open the [SIC] folder --> the Flutter --> and flutter-console. to See our flutter Console



- Next, we want to run a flutter command that

```
C:\Users\Shawn > flutter doctor
```

flutter doctor - basically runs a diagnostic on flutter and makes sure it is installed correctly

** NOTE: Command takes a couple of minutes

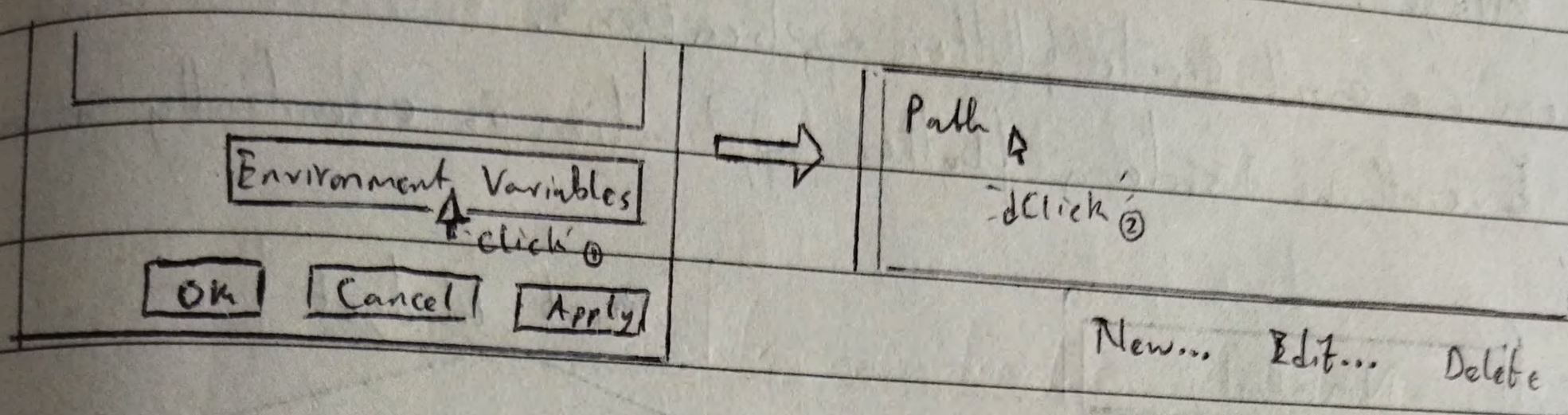
- Next, we want to go and open a new cmd and type

```
C:\Users\Shawn > flutter --version
```

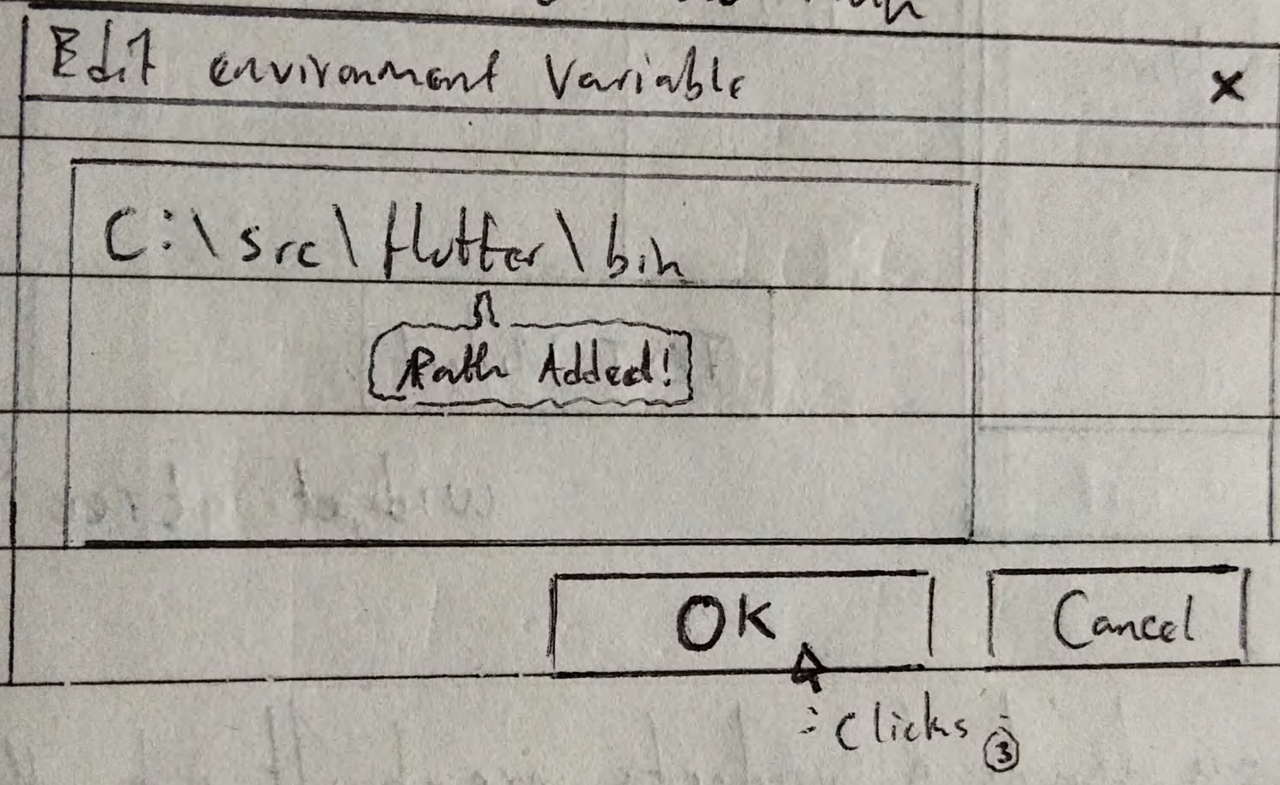
if this shows the flutter version - then we are good ☑

if this doesnt show the flutter version - we need to add a path variable

- Search Windows [env], click → Edit the System Environment variable.



... Then Add The Path



- After adding path, try checking the version again and it should work

Course Files

To get all the Course files, you can get them on my github.

[github.com/iamshaunjp/flutter-beginners-tutorial]

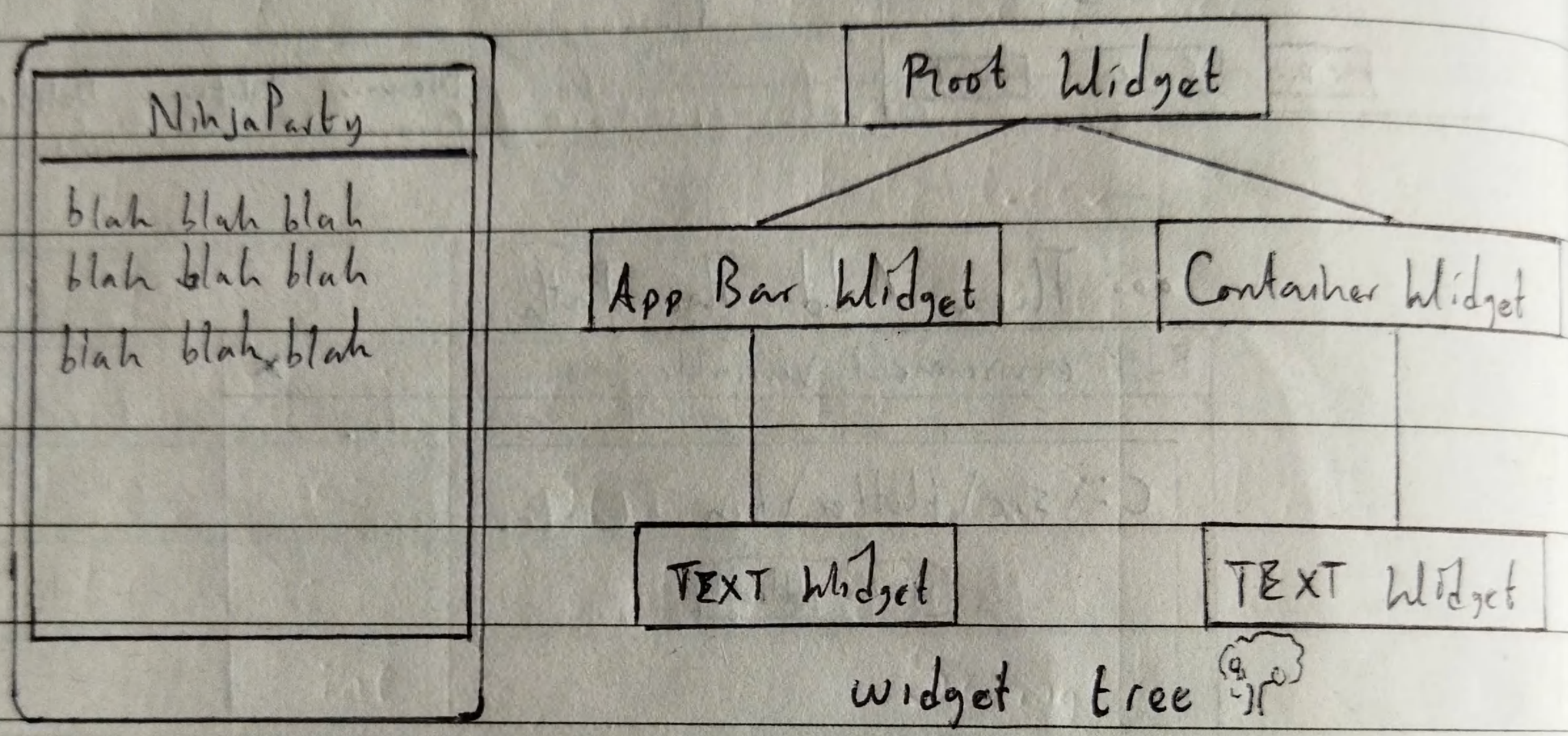
- Once you visit link don't be surprised if you see no code, simply click on [Branch:], To see a dropdown of all the different lessons codes.

FLUTTER OVERVIEW #2

WIDGETS

Widgets are the bedrock of flutter applications, because without them we can't create flutter applications.

Everything inside a flutter application is essentially a widget.



Now their are loads of widgets pre-built into the flutter framework that we can use straight out-of-the-box.

TEXT WIDGET	BUTTON WIDGET	ROW WIDGET	COLUMN WIDGET	IMAGE WI

Each of these widgets are fully customizable and we can pass them different properties to these widgets to alter how they display on their screen

E.g

style, textAlign, Overflow, maxLines etc.

color, elevation, disabledColor, enabled etc.

We also need to know that, at the end of the day, all widgets are in Flutter are classes.

Each widget has it's programmatic class that defines it's behaviour and also how it looks on the screen.

The way we implement these widgetry structure in flutter is by using a programming language called Dart.

H Dart is a programming language developed by Google. To build mobile, desktop and web applications.

Dart Primer #3

Learning Dart

Now Dart is what's known as a staticly typed programming Language and what that means is that, Once we declare a variable and we give that variable a type e.g String - We then can't change the type of that variable in the future.

Variables

dartpad.dev ← visit

DartPad		
void main(){	void main(){	void main(){
int age = 30;	String name = 'dun';	bool isNight = false;
print(age);	print(name);	print(isNight);
} int	} String	} bool

We've seen all this data types before, but Dart also comes with a new programming data type called "dynamic".

If we set a data to a type dynamic, it simply means we can change that type in the future.

Since, Dart is a staticly typed programming language -- using a data type like dynamic leads to us having some error, so it is best to avoid it where possible

Functions

Now the void main(){} function is the default function in Dart, it is the function that Dart is going to find and execute at the start of a Dart Program.

void is basically a return type and it is basically telling Dart what our main(){} function is going to return.

- DartPad

```
void main(){
  String greet = greeting();
  int age = getAge();
  print(age);
  print(greet);
}
```

```
String greeting(){
  return 'hello';
}

int getAge(){
  return 30;
}
```

NOTE: Dart also supports the use of arrow => functions just like in Javascript

- Using the Dart Arrow function to re-write our methods functions

```
String greeting() => 'hello';

int getAge() => 30;
```

List

We can use the List type to create a List in Dart

DartPad

```
void main(){
  List names = ['chun-li', 'yoshi', 'mario'];
  names.add('luigi');
  names.remove('yoshi');

  print(names);
}
```

CONSOLE

[Chun-li, mario, luigi]

NOTE: With our above List names, we can go ahead and add other different datatypes, but it is not best practice to to this in Dart. So to prevent this we can tell Dart the datatype we want in our List (Only).

...

```
List<String> names = ['chun-li', 'yoshi', 'mario', ]
```

...

Classes

Classes are basically blueprints for objects. It could be a User Object or a Shop Object.

We use these classes to describe these objects by giving them properties and methods.

```
void main() {
  User userOne = User();
  print(userOne.age);
  print(userOne.username);
  userOne.login();
}
```

CONSOLE

25

mario

User logged in

Our Class

```
class User {
  String username = 'mario';
  int age = 25;

  void login() {
    print('user logged in');
  }
}
```

CONSTRUCTORS

We don't have to hard code ~~the~~ name and age we can simply use the Constructor to initialize our properties on object creation.

Dart Pad

```
void main() {
  User userOne = User('luigi', 25);
  print(userOne.username);

  User userTwo = User('mario', 30);
  print(userTwo.username);
```

```
class User {
  String username;
  int age;

  User(String username, int age){
    this.username = username;
    this.age = age;
  };
```

INHERITANCE

This is basically when we have a class that inherits from another class (a super class) all its properties and behaviours

extends : for Inheriting

DartPad

```
class User {
  ...
}

void main() {
  ...
  SuperUser userThree = SuperUser(
    'Yoshi', 20);
  userThree.publish();
  print(userThree.username);
}
```

```
class SuperUser extends User {

  SuperUser(String username, int age
  ) : super(username, age);

  void publish() {
    print('published update!');
  }
}
```

With these Fundational knowledge in place, we are officially ready to jump into learning Flutter using our newly learnt programming language - Dart.

Creating a Flutter App in Android Studio #4

Open up your Android Studio app. First thing we want to do is create an android emulator, so that we can test our Flutter App on it

Click On {Android Virtual Device Manager}

Configure ▷ AVD Manager ▷ + Create Virtual Device... ▷ Nexus 6 ▷ Pie [#** Because In the Past I used Q and had alot of errors running the App] ▷ Next ▷ Graphics: Hardware - GLES 2.0 [# Improves Rendering] ▷ Finish

Tutor: Next It takes some time to Install this size 513 MB

Next up we want to Install some plugins that are going to help us in android studio when we are creating Flutter Apps

Click On

Configure ▷ ~~Settings~~ Plugins ▷ Marketplace ▷ Q Flutter ▷ Flutter {This Installs alot of tools into Android Studio which helps us when creating Flutter Apps} ▷ The plugins you want requires (Dart)... Yes ▷ Restart IDE {After IDE restarts we should see our new create Start a new flutter project option} ▷ + Start a new flutter project ▷ Default Flutter Application... Next ▷ Project name: myapp ▷ Next ▷ Company domain ▷ thenetninja.co.uk ▷ Finish

This runApp function runs MyApp Class which is the Root Widget

lib ▸ main.dart

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

: First function that fires when our dart file is ran

```
class MyApp extends StatelessWidget {
```

MyApp : This is us basically creating a new widget called MyApp

StatelessWidget :

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(...
```

Widget build(...) : This basically helps us build our App

MaterialApp(...) : This is like a wrapper that allows us to alot of material design widget inside it.

→ home : : The home property basically decides what widget loads on the home screen.

```
    home: MyHomePage(title: '...'),
  );
}
```

... And So On

NEXT : Clear all the dummy code

```
void main() => runApp(MaterialApp(
  home: Text('hey ninjas!'),
));
```

main.dart ▸ ▷ run

☐ You should see changes in your Android Emulator

Scaffold & App Bar Widget #5

To add more content to the screen, the first thing we want to explore is what we call the scaffold widget.

The scaffold widget is going to allow us to implement a basic layout for our app. E.g

- An app bar at the top
- Floating Action buttons if you want
- and more ...

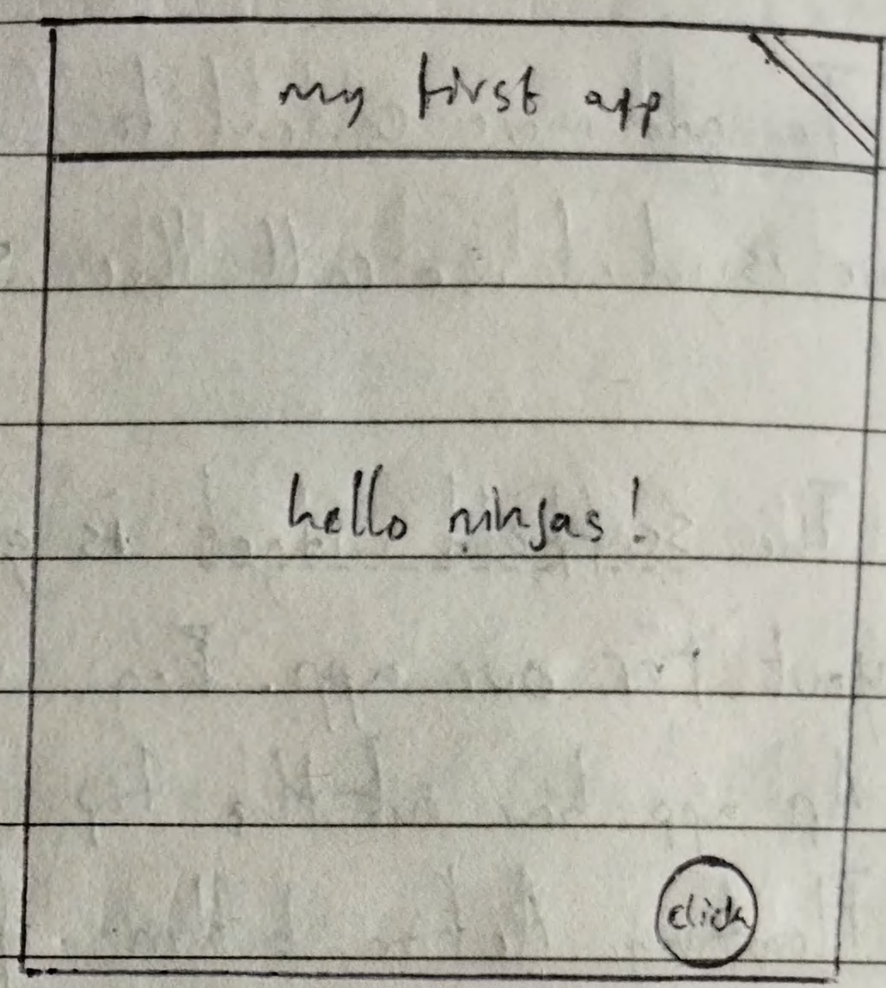
Scaffold widget takes alot of different properties that help specify different things about the base layout of our App

main.dart

```
...

void main() => runApp(MaterialApp(
  home: Scaffold(
    appBar: AppBar(
      title: Text('my first app'),
      centerTitle: true,
    ), // AppBar
    body: Center(
      child: Text('hello ninjas'),
    ), // Center
    floatingActionButton: FloatingActionButton(
      child: Text('click'),
    ), // FloatingActionButton
```

Run

Run without parent



BEST PRACTICE

- At the end of every: value pair, add a ","

- All widgets start with an uppercase because they are classes

- Widgets always take properties or content instead of just a widge

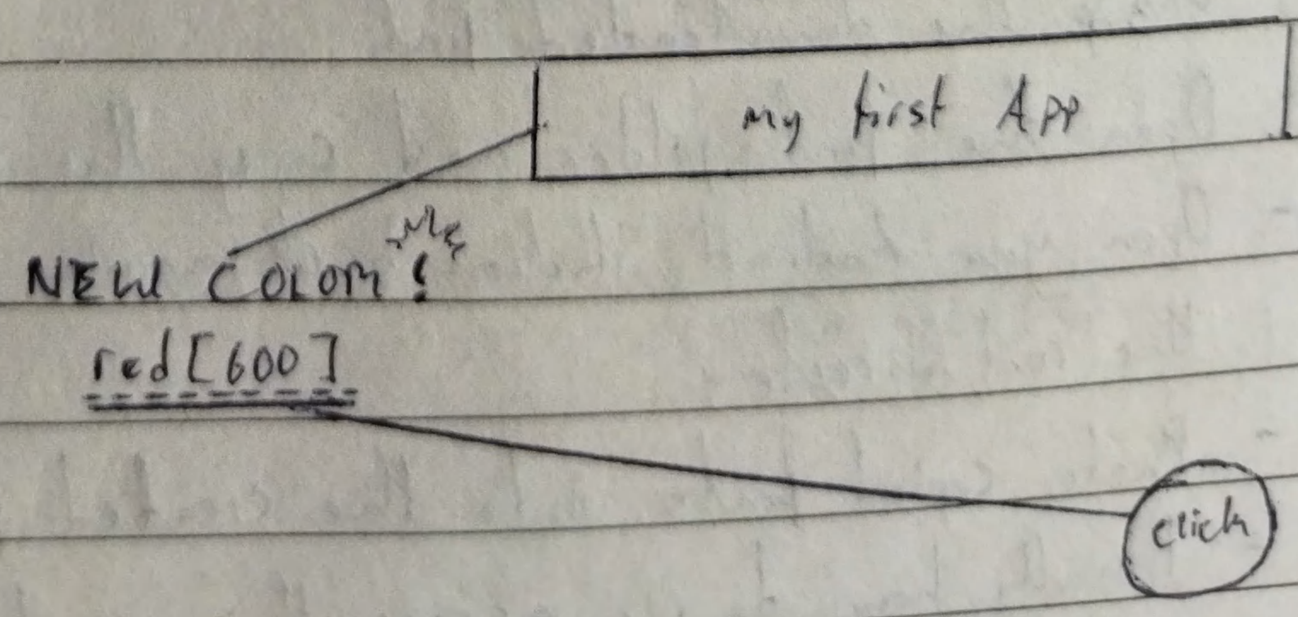
Colours & Fonts #6

Now we've been able to create something nice, but as you can see our application still uses the default designs for colors and font.
Now In this Section, we are going to look at some different colors and font styles

```
main.dart                                   // CHANGE COLOR
...

appBar: AppBar(
    ...

    backgroundColor: Colors.red[600],
),  //AppBar
...

FloatingActionButton: FloatingActionButton(
    onPressed: () {},
    child: ...,
    backgroundColor: Colors.red[600],
),  // FloatingActionButton
...
```

color class

strength of color



// Stylingn Our Centerd Text

```
...

body: Center (
  child: Text (
    'hello ninjas!',
    style: TextStyle (
      fontSize: 20.0,
      fontWeight: FontWeight.bold,
      letterSpacing: 2.0;
      color: Colors.grey[600],
    ), // TextStyle
  ), // Text
), // Center
...
```

class property

≡ To Add A Custom font family

- Visit Google fonts and download any font of your choice
- Unzip the downloaded font
- Open the font folder and copy the font files you want
- Open your Android Studion back again, and create a folder "fonts" in the root directory
- Paste copied fonts into the created folder "fonts".
- Finally how do we access this newly added font

pubspec.yaml : Is like a Configuration file for our project. It's where we can specify environment, dependencies or any assets and fonts we want to use in our Project as well. Format matters in this file.

- pubspec.yaml

Adding New Fonts | Spacebar 2x for indentation

fonts :

- family : IndieFlower # name arbitrary

  fonts :

  - asset : fonts / IndieFlower - Regular.ttf # whichever font you added in your case

  # Add more-assets if there are more

  # style and weight can also be added

  # Ctrl + S

≡ Let's Now use this font-family

- main.dart

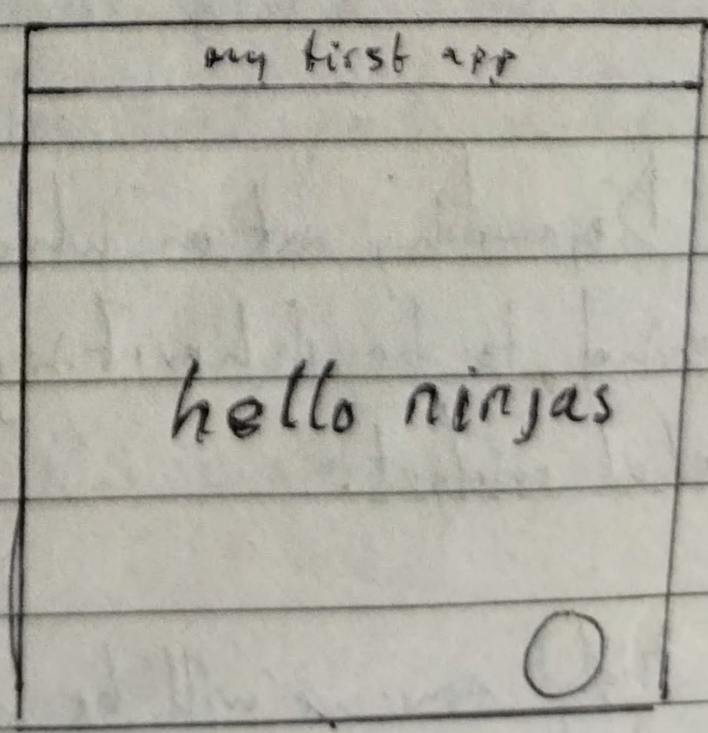
... | Pubspec has been edited Get dependencies ... |

style : TextStyle (

...

fontFamily : 'IndieFlower',

),// TextStyle



Stateless Widgets & Hot Reload #7

Now, in our application whenever we make a change and want to preview that change we have to click the HOT RESTART button icon in the Console --> Now that's okay but since we make alot of changes, it would be nice to see a preview right after we save changes.

Thankfully flutter comes with a feature called "hot Reload" that is built into Stateless Widgets & Stateful Widgets

Stateless vs Stateful

- Stateless Widgets
  the state of the widget **cannot** change over time

- Stateful Widgets
  the state of the widget **can** change over time

Depending on what we want with our custom widget, we are going to be inheriting either the stateless widget or the stateful widget.

Right now we will be working more with stateless widget since our widget to be created won't be changing

main.dart

...

« Creating a stateless widget to handle Home

Snippet: stless + ENTER

```
class Home extends StatelessWidget {
  @override
  Widget build(BuildContext context){
    // Note: Copy our widget tree inside our home: [...]
    //       then paste it in the return below

    return Scaffold(
      ...                                  // Pasted ☑
    )
  }
```

→ Copy this all!

property in void main

Now when ever we make a change to the code inside this widget tree, flutter is going to recognize/detect that and it is going to our build to re-run, when it re-runs flutter then ~~XXXX~~ goes to where the code changes and updates that part on the screen.

- Thats HOT RELOAD in action.

We can now also by doing this use this custom widget in several places in our application

// Add new widget Object to our main()

```
void main() => runApp(MaterialApp(
  home: Home(),
));  // MaterialApp
...
```

And Just Like that we are done. Now whenever we make any changes in our stateless widget, it reflects on our emulator after we save those changes.

Images & Assets #8

Hey there, so there is a couple of ways we can use Images in our flutter app

- Network Image : Image from the web
- Asset Image : Image from our projects source files

* Lets add Image to the body of our application using both methods

Using Network Images STEPS

- Grab Image URL from the web
- Go back to flutter project ⇓

- main.dart

```
class Home extends StatelessWidget {
  ...
    body: Center(
      child: Image(
        image: NetworkImage('Paste Image URL')
      ), // Image
    ), // Center
...
```

≡ Using Asset Images STEPS

- Download an Image from the web
- Go back to your flutter project and create a folder called assets in the root directory ○ assets
- Copy your downloaded Image into the assets folder
- Open pubspec and tell it we have some new Images/Image

○ pubspec.yaml

```
...

flutter:

  ...

  assets:

    - assets/          # This gives us access to all Images inside
                         this folder
```

- Lets try displaying ~~an~~ Image now

○ main.dart

```
class Home extends StatelessWidget {
  ...

    body: Center (
      child: Image (
        image: AssetImage ('assets/Image.jpg'),
                                  image name
      ), // Image
    ), // Center
```

free Img website => www.unsplash.com

Simpler way to use network Images and asset Images

- Network Images

  PASTE IMG URL

  child : Image.network('assets/image.jpg'),

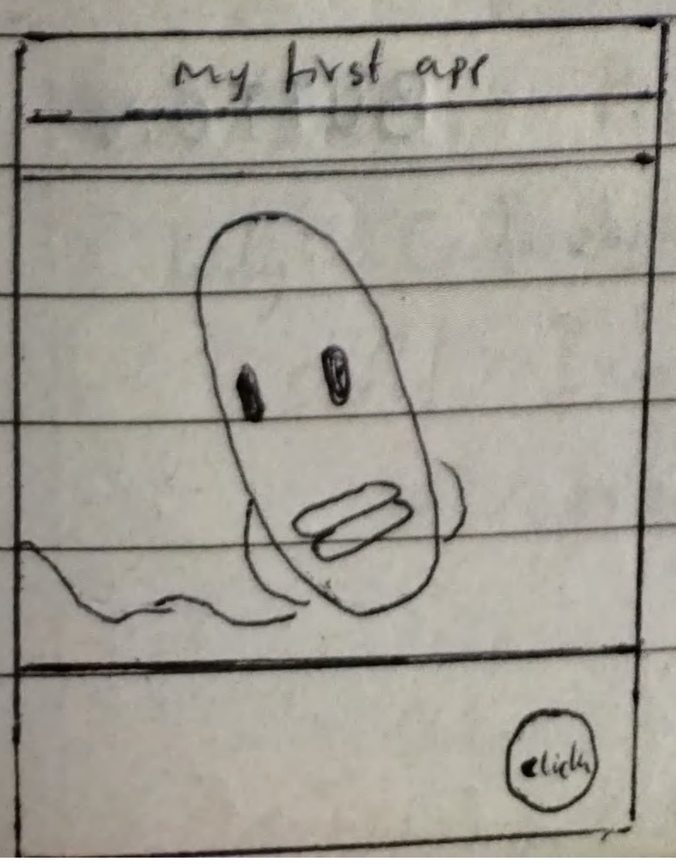
- Asset Images

  child : Image.assert('assets/Image.jpg'),

SIDE

For now we are just adding a single ~~image~~ bit of content to the page, in the body at a time. This is going to change in the future so that we can have multiple bits of text and images showing on the page at the same time.

For now I just want to show you a few basic widgets we can use first before we start playing around with the layout and then adding more stuffs to the app.



Buttons & Icons #9

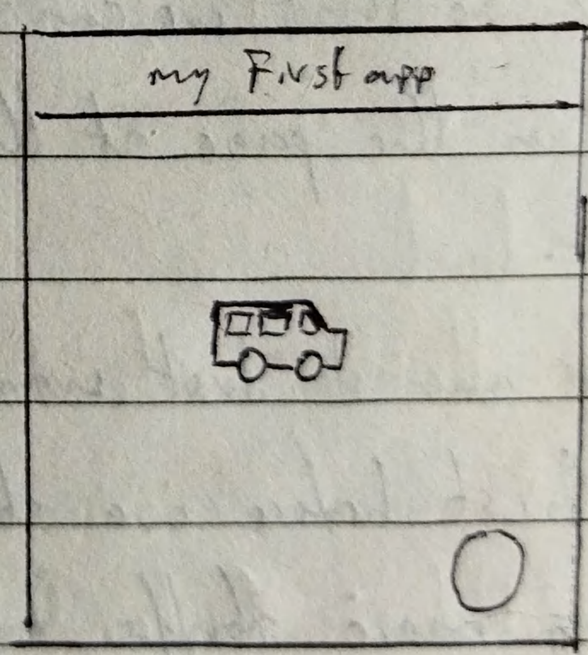
In this tutorial, we will be learning how to use buttons & Icons and then a mixture of the two as well

ICONS

main.dart

```
...

body: Center(
  child: Icon(
    Icons.airport_shuttle,
    color: Colors.lightBlue,
    size: 50.0,
  ), // Icon
), // Center
```

There are several options



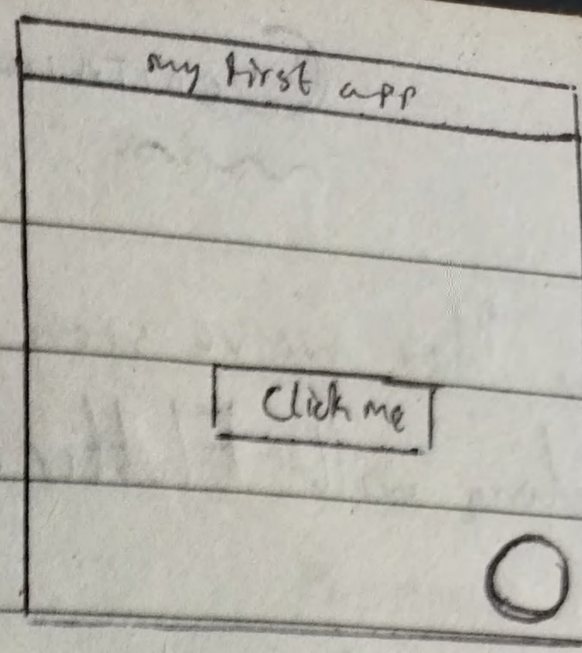
BUTTON

Next Let's look at buttons

```
...

body: Center(
  child: FlatButton(
    onPressed: () {
      print('you clicked me');
    },
```

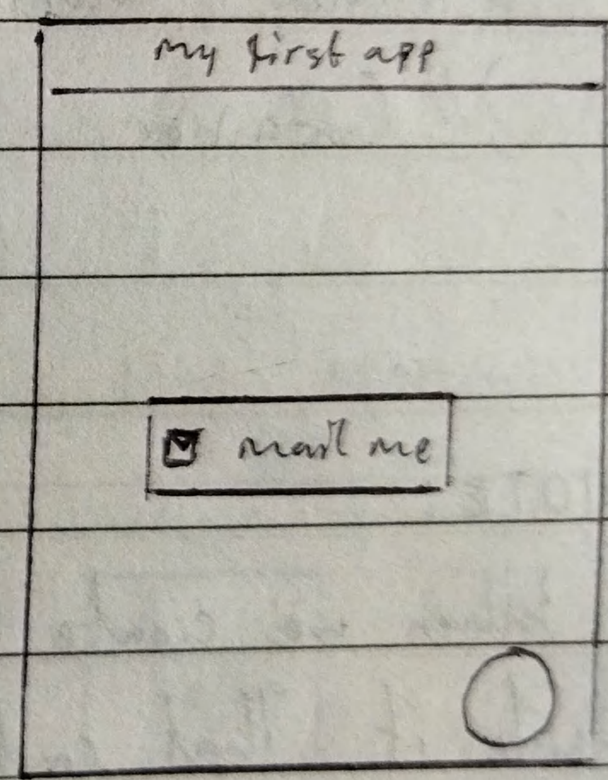
There are several kinds of buttons

```
    child : Text ('click me'),
    color : Colors.lightBlue,
  ), // FlatButton
), // Center
```



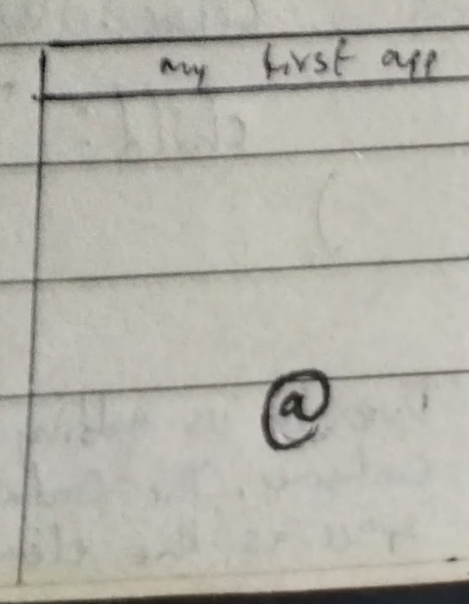
≡ NEXT: Buttons with Icons — ICONS IN BUTTON

```
...
body : Center (        --> Another kind of button
  child : RaisedButton.icon (
    onPressed: () {},
    icon: Icon (
      Icons.mail
    ), // Icon
    label: Text ('mail me')
    color : Color.amber,
  ), // RaisedButton.icon
), // Center
```



≡ NEXT/LASTLY: Pressable Icon — ICON BUTTON

```
...
body : Center (
  child : IconButton (
    onPressed : () { },
    icon : Icon (Icons.alternate_email),
    Color : Colors.amber
  ), // IconButton
```



Containers, Margin & Padding #10

Now we've seen a few of the basic UI widgets that come along with Flutter. Let us now turn our attention to Layout.

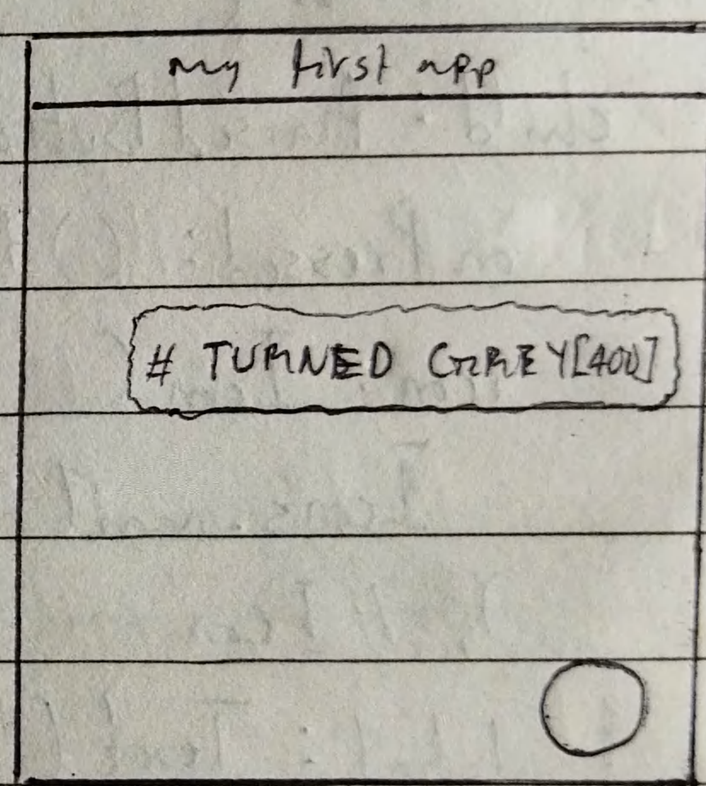
One of the simplest Layout widget that flutter gives us is the Container widget.

CONTAINER WIDGET

main.dart

```
...

body: Container(
    color: Colors.grey[400],
),// Container
```

Result →



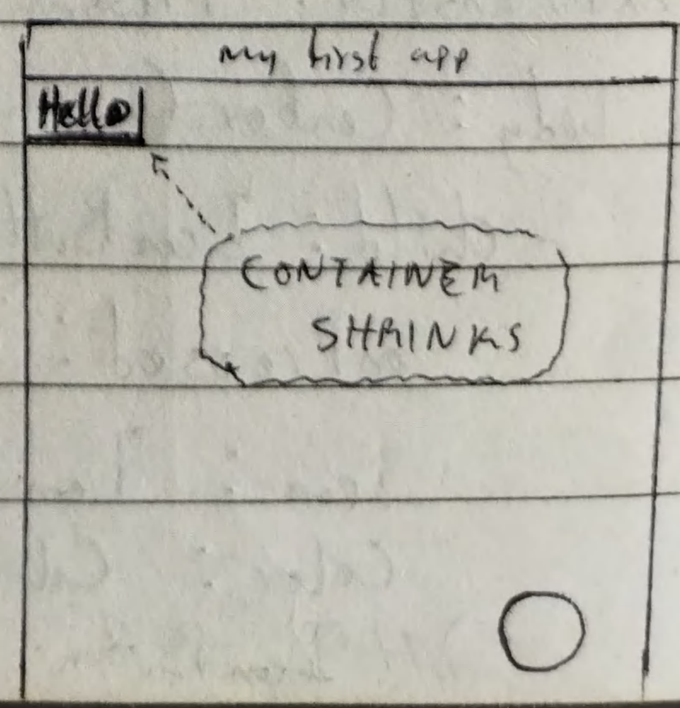
NOTE:

When we create a container and it has no children widgets inside it. That container takes up the full space available to it. -- Containers are widgets that acts as a wrapper to other widgets

...

ADD ELEMENT TO CONTAINER

```
body: Container(
    color: Colors.grey[400],
    child: Text('hello'),
),
```



Due to us adding an element/widget into our Container. The Container now only takes up as much space as the element in it has

Margin & Padding

One of the great things about containers is, we can add margin and padding to our container.

...

```
body: Container(
  padding: EdgeInsets.all(20.0),
  color: Colors.grey[400],
  child: Text('hello'),
), // Container
```

PADDING

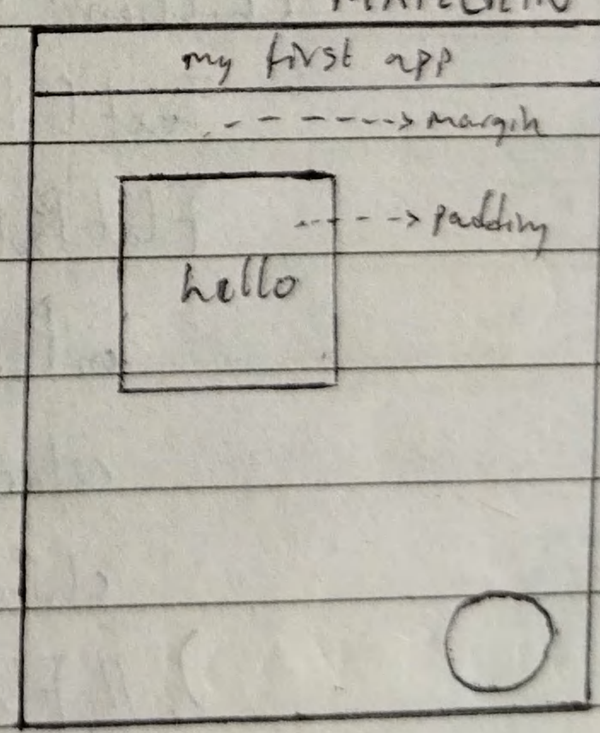
EdgeInsets : is how we control space like margin and padding inside flutter

- .symmetric : horizontal and vertical
- .fromLTRB : Left, Top, Right, Bottom
- .all : All sides

...

MARGIN

```
body: Container(
  ... 1
  margin: EdgeInsets.all(30.0),
  ... 2
), // Container
```



PADDING WIDGET

We can also use the Padding widget

...

```
body: Padding(
  padding: EdgeInsets.all(90.0),
  child: Text('hello'),
), // Padding
```

ROWS #11

So far, so good we've only been adding 1 widget at a time to the body of our app. Be that a Text Widget, Image, Button, Container e.t.c

Now we want to expand our Layout so that we can have multiple different widgets on the page at once. To achieve that we will be using Row & Column widgets

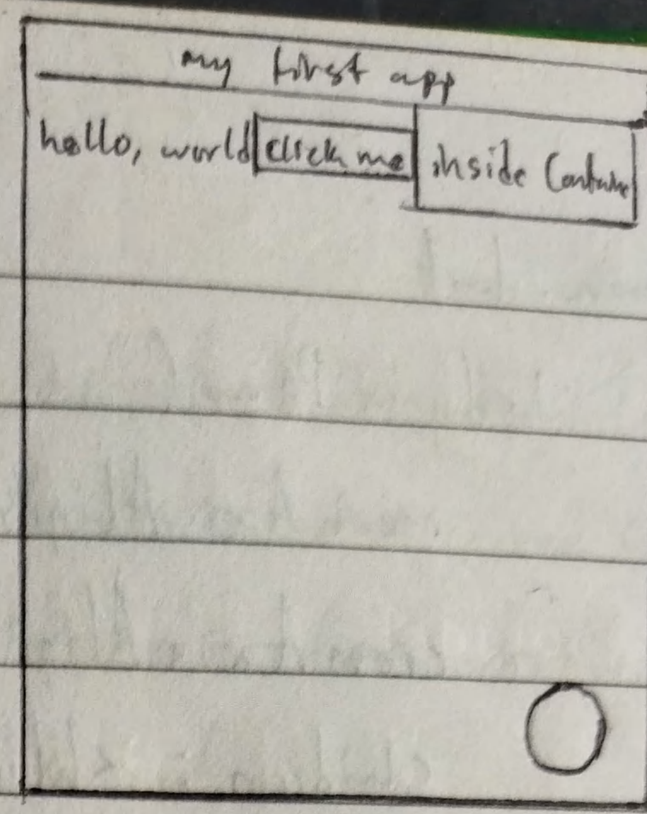
Add Row with 3 children widgets — ROW

main.dart

```
...
    body: Row (
      children: <Widget> [
        Text ('hello, world'),
        FlatButton (
          onPressed: () {},
          color: Colors.amber,
          child: Text ('click me')
        ), // FlatButton
        Container (
          color: Colors.cyan,
          padding: EdgeInsets.all (30.0),
          child: Text ('inside container'),
        ), // Container
      ], // <Widget> []
    ), // Row
```

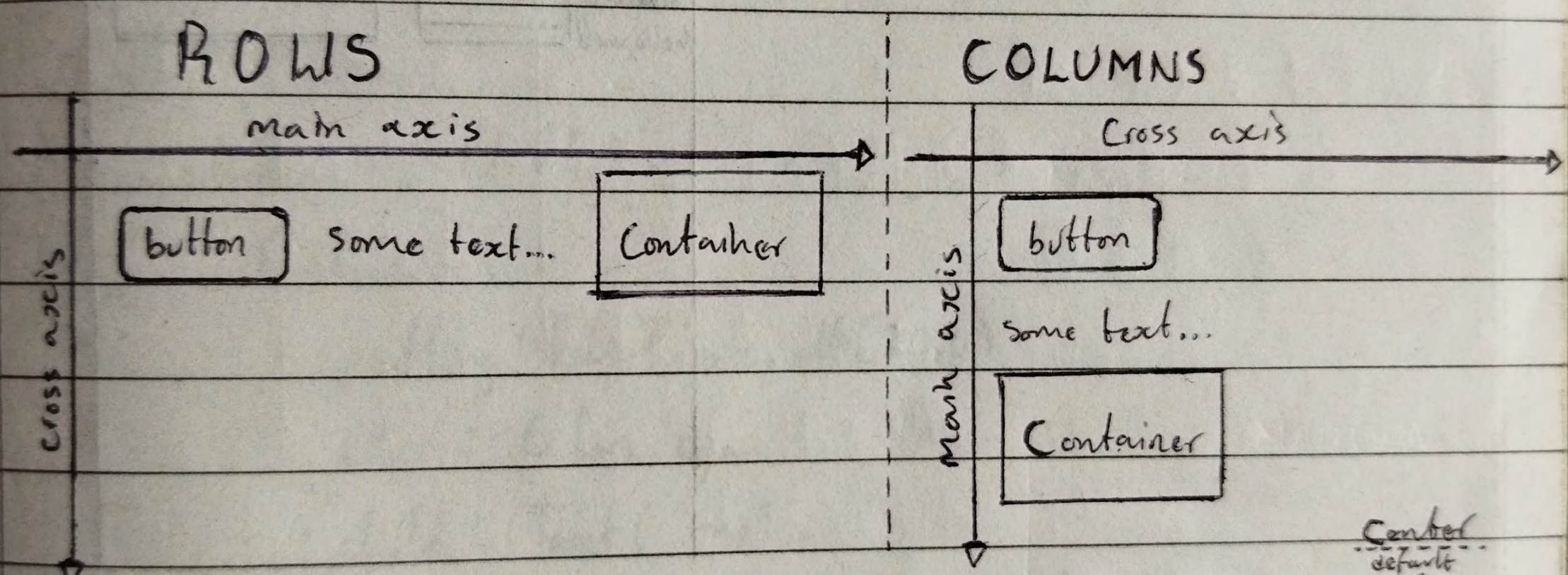
Takes children instead of child. with it's value being a List of Widgets

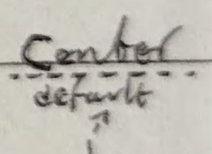
o We see that we now have several widgets. But by default, the widgets are all bunched up to the start of the Row.



To change the layout we can use the

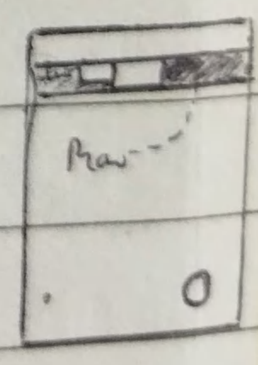
- Main Axis Alignment
- Cross Axis





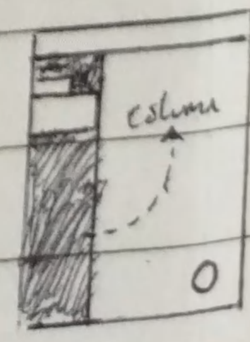
Propertise For Main Axis Alignment and Cross Axis Alignment

- Center, spaceEvenly, spaceAround, spaceBetween, start, end, stretch
  (start: default)



NOTE: When working with Rows

- The Row spans the entire width of the screen
- The hieght of the Row is the hight of the tallest widget



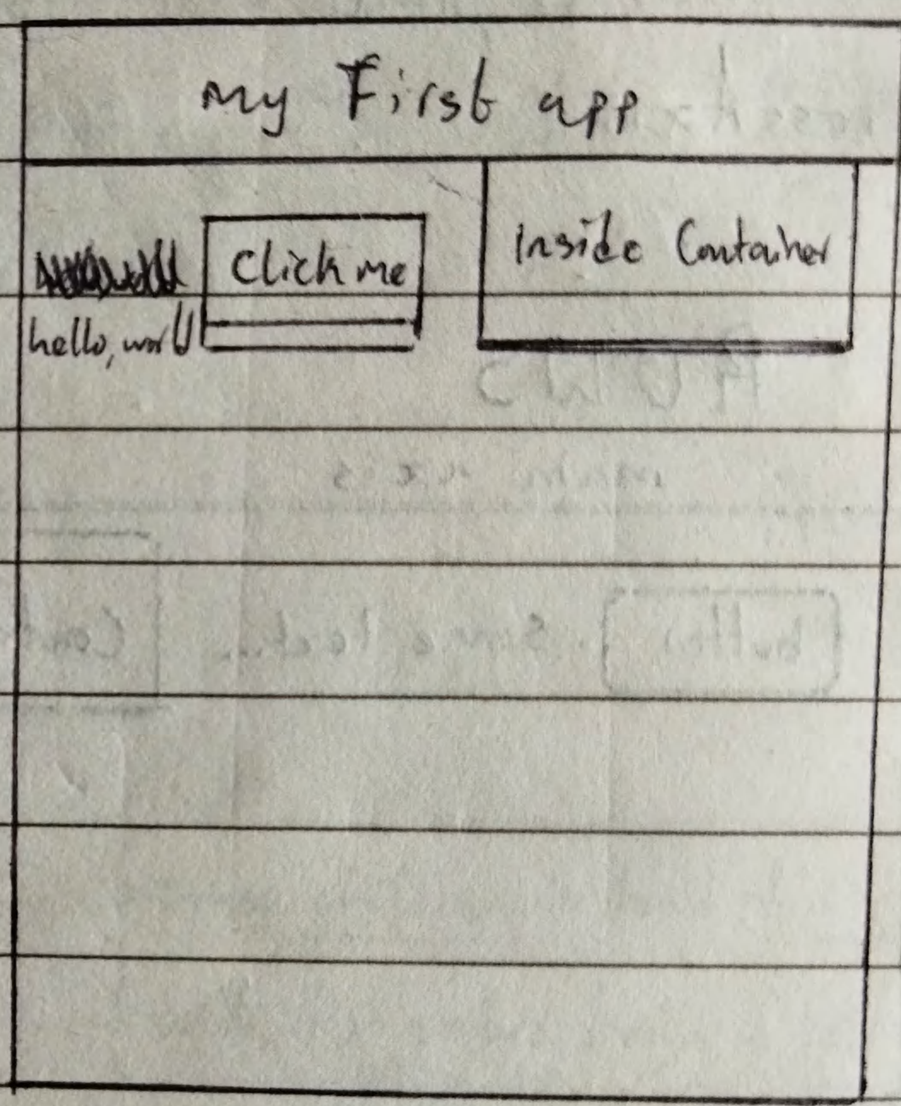
NOTE: When working with Column

- The Column Spans the entire hieght of the screen
- The width of the Column is the widht of the widest widget in the list

main.dart

```
...
    body: Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        crossAxisAlignment: CrossAxisAlignment.end,
        children: <Widget>[...], //<Widget>[]
    ), Row
```

try other properties



COLUMNS #12

Now we saw how to create a row and output different widgets in that row, and arrange the widgets across the main axis; and cross axis.

Let's now look at Columns, which stacks elements/widgets in a vertical way instead of the horizontal way.

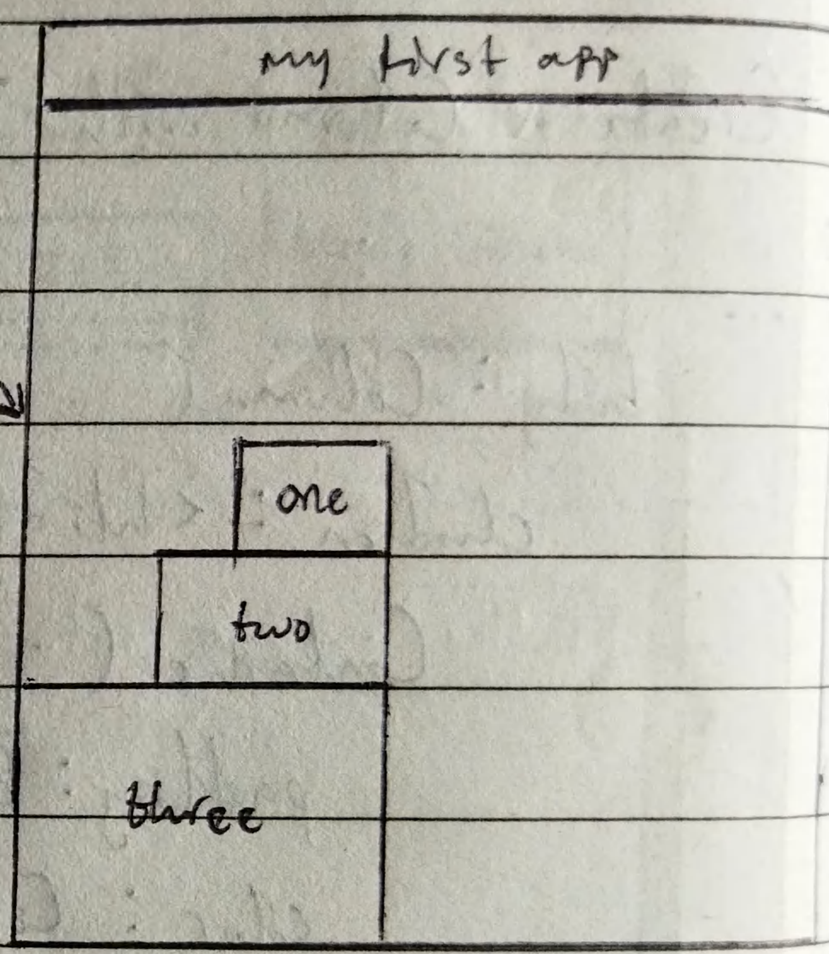
Create a Column with 3 children Containers

COLUMN

```
...
  body: Column(
    children: <Widget>[
      Container(
        padding: EdgeInsets.all(20.0),
        color: Colors.cyan,
        child: Text('one'),
      ), // Container
      Container(
        padding: EdgeInsets.all(30.0),
        color: Colors.pinkAccent,
        child: Text('two'),
      ), // Container
      Container(
        padding: EdgeInsets.all(40.0),
        color: Colors.amber,
        child: Text('three'),
      ), // Container
    ], // <Widget>[ ]
  ), // Column
```

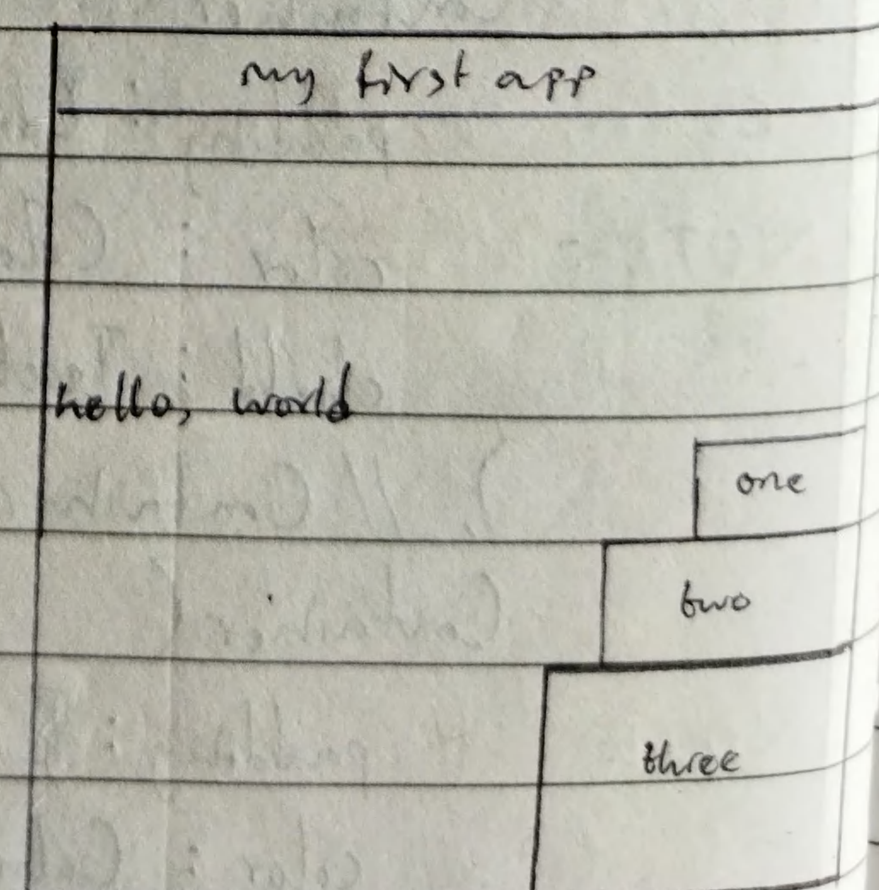
≡ Add mainAxisAlignment and CrossAxisAlignment to our Column

```
...
    body: Column(
        mainAxisAlignment: MainAxisAlignment.end,
        crossAxisAlignment: CrossAxisAlignment.end,
        children: <Widget>[...], // Widget
    ), // Column
```



≡ Add Row Inside Column

```
...
    body: Column(
        children: <Widget>[
            Row(
                children: <Widget>[
                    Text('hello,'),
                    Text('World'),
                ], //<Widget>[]
            ), //Row
            ...
        ], <Widget>[]
    ),
```



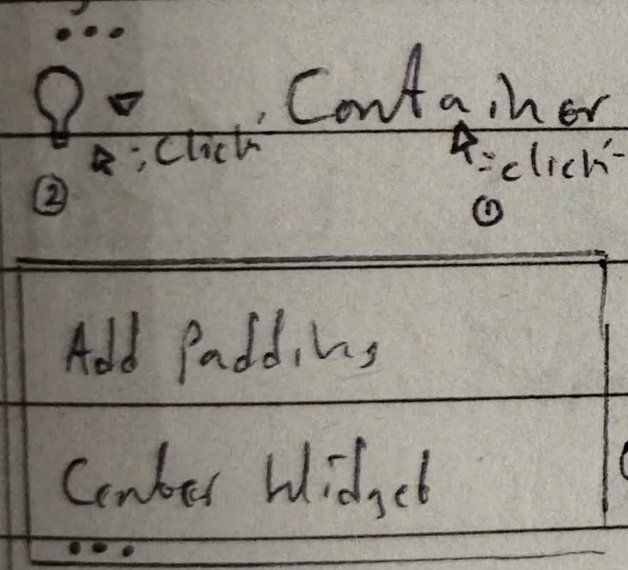
Flutter Outline & Shortcuts #13

By now you should be getting a little used to creating widgets and simple layouts using a row or a column or even a combo of both.

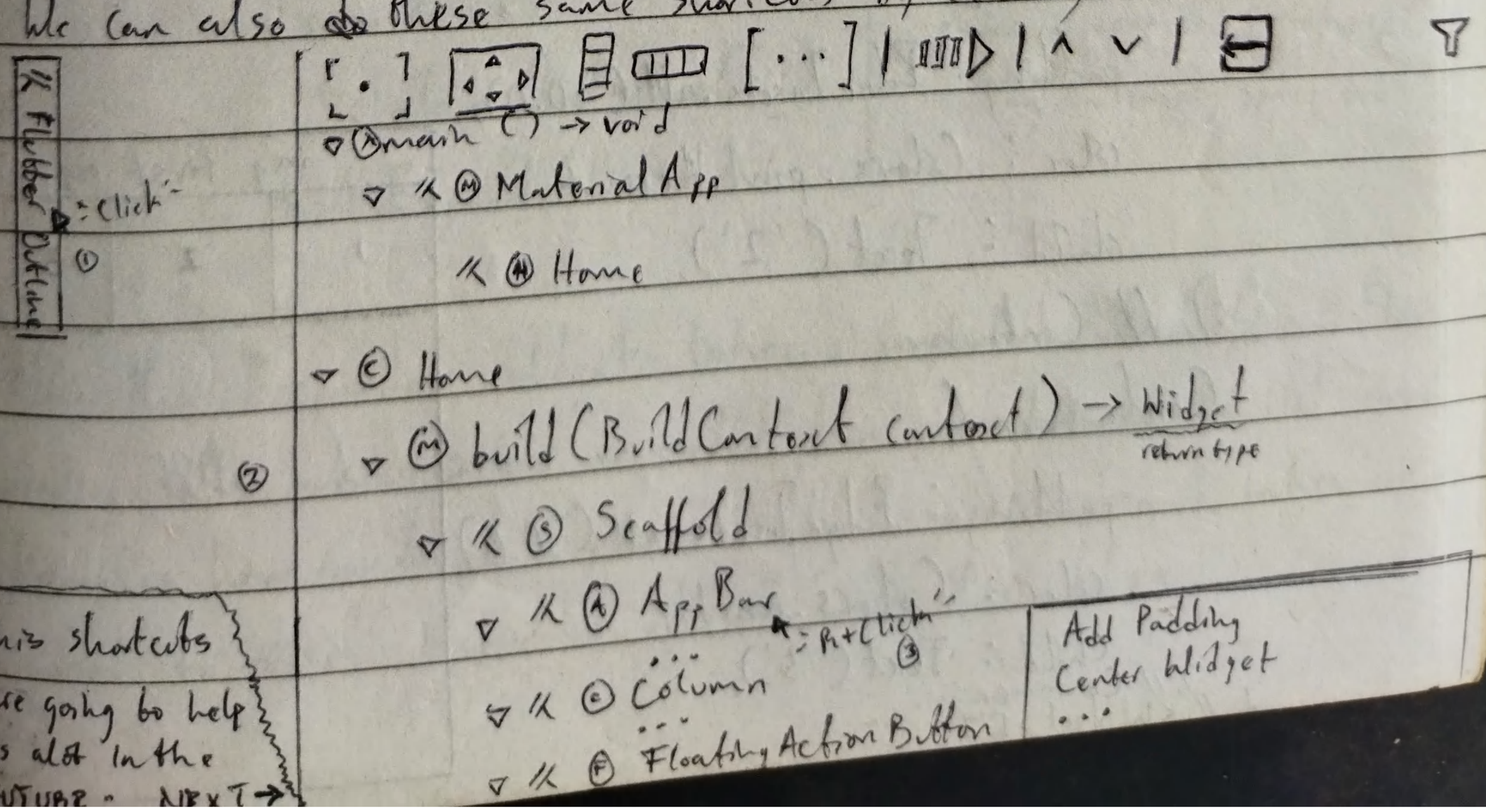
We want to now look more on the action menu. Whenever we click on a widget we should see this ------ pop up by the left hand side to the widget, it contains alot of actions like now

- Move entire widget tree up or down
- Add different kinds of wrappers rather than copying and pasting

E.g



We can also get these same shortcuts by clicking on the Flutter Outline



Expanded Widgets & Flex #14

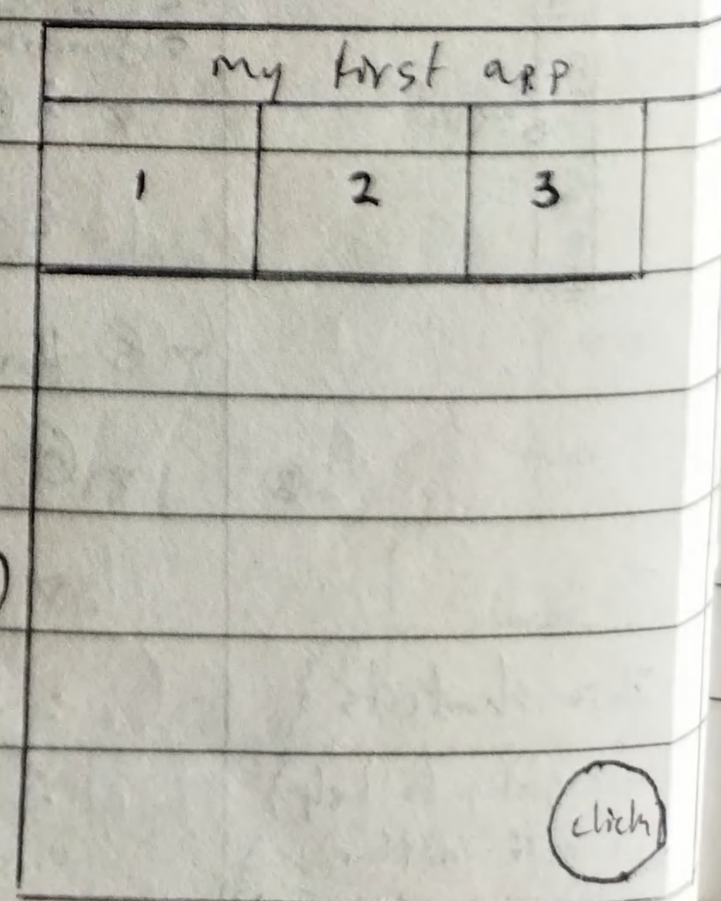
Wonderful, now we've seen containers, rows and columns, all ways to layout content on our screen.

Next let's checkout another Layout Widget called the expanded Layout Widget.

- Create a Row with 3 Containers

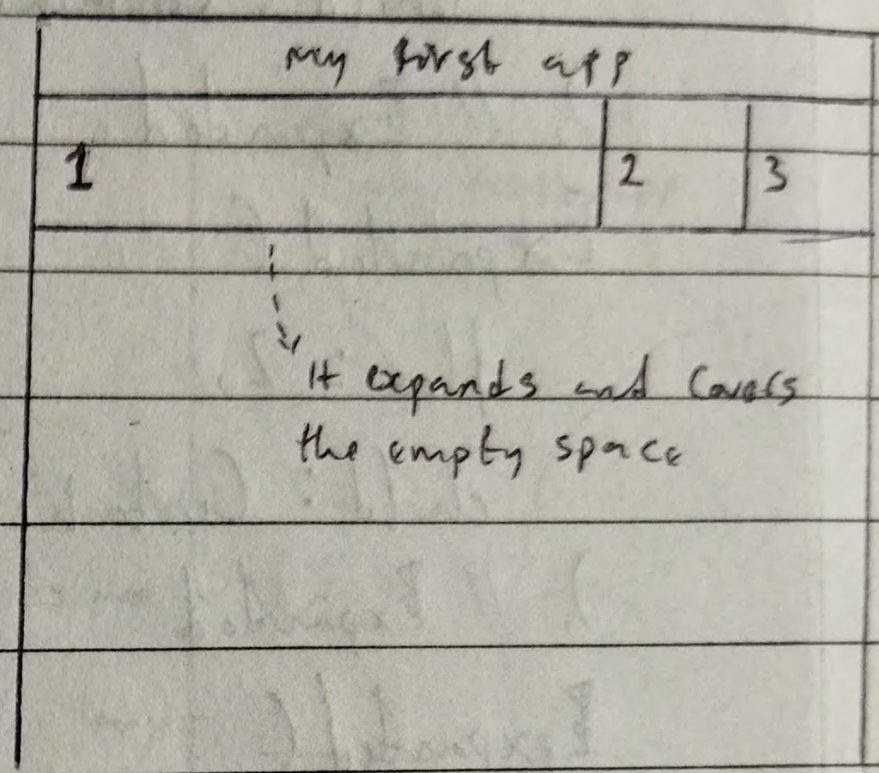
ROWS with CONTAINERS

```
...

body: Row (
  children: <Widget> [
    Container (
      padding: EdgeInsets.all(30.0),
      color: Colors.cyan,
      child: Text('1'),
    ), // Container
    Container (
      padding: EdgeInsets.all(30.0),
      color: Colors.pinkAccent,
      child: Text('2'),
    ), // Container
    Container (
      padding: EdgeInsets.all(30.0)
      color: Colors.amber,
      child: Text('3'),
    ), // Container
  ], // <Widget>[]
```



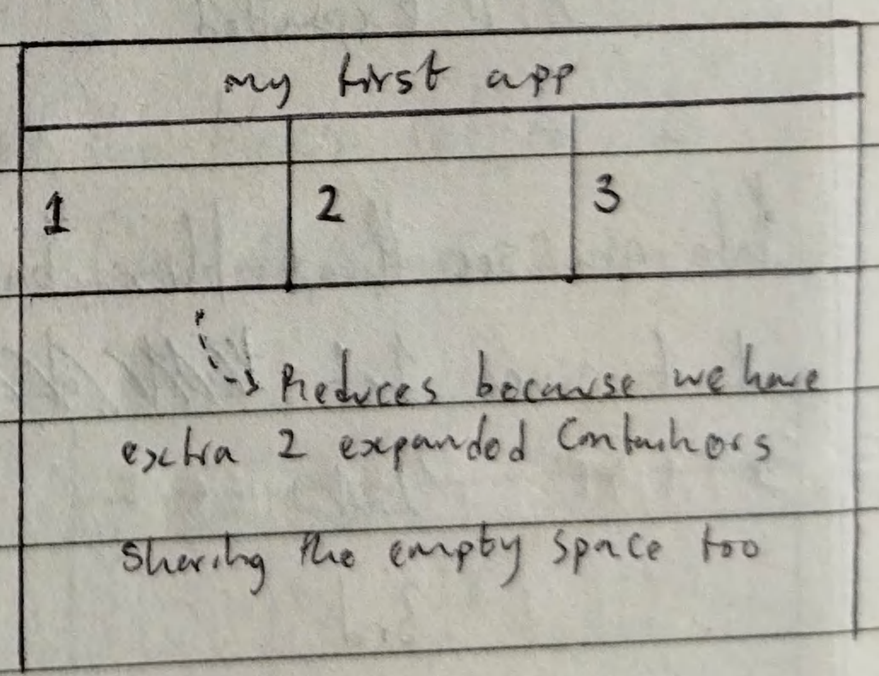
Wrap 1st Container with Expanded Widget — Expanded Widget

```
body : Row (
    children : <Widget> [
        Expanded (
            child : Container (...), //
        ), // Expanded
        Container (...),
        Container (...),
    ], // <Widget> []
), // Row
```



Wrap 2nd and 3rd Container with Expanded Widget

```
Expanded (
    child : Container (...),
), // Expanded
Expanded (
    child : Container (...),
), // Expanded
```

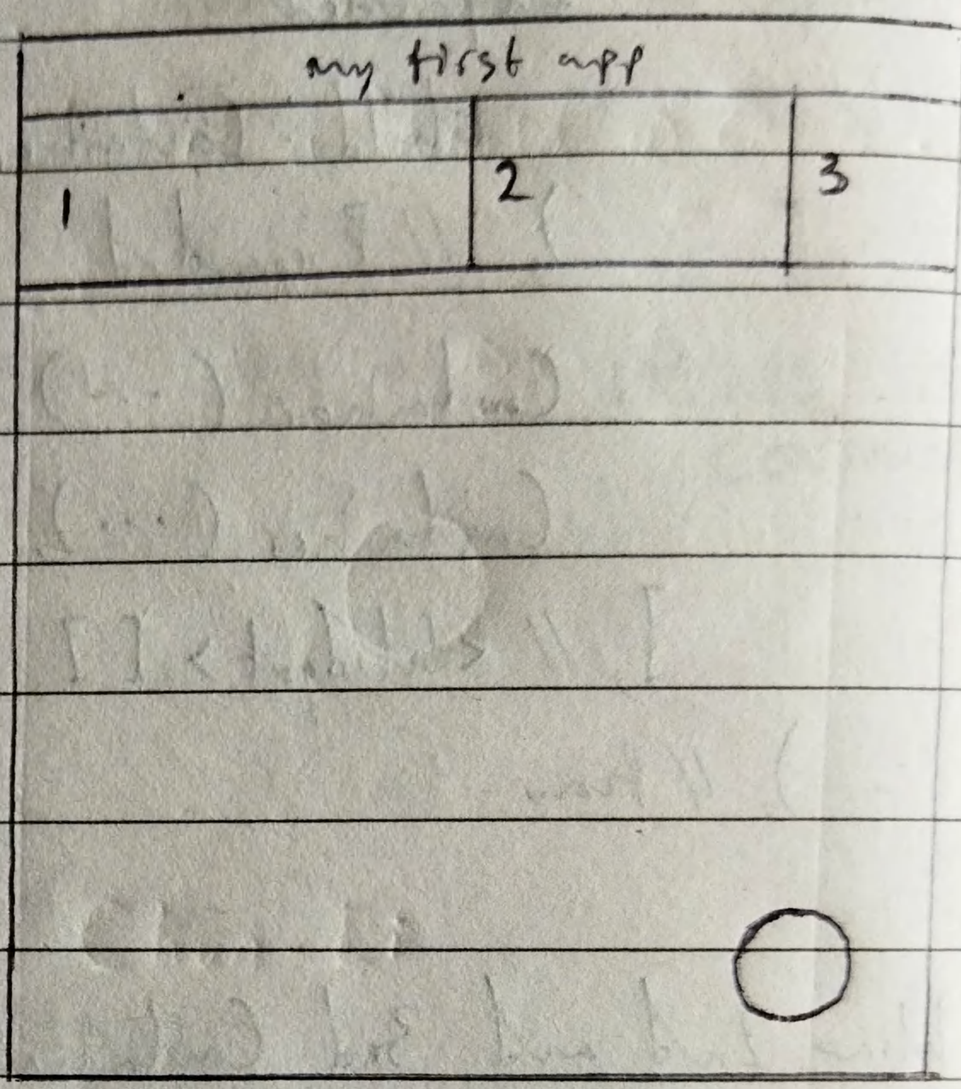


Now the Expanded Widget takes a property called flex. Flex basically ~~tells~~ takes other expanded flex properties in the list and determines how much it expands or how much space it takes up as supposed to other expanded Widgets with their flex.

Implement flex in all 3 Expanded Widgets

EXPANDED FLEX

```
Expanded(
    flex: 3,
    child: Container(...),
), // Expanded
Expanded(
    flex: 2,
    child: Container(...),
), // Expanded
Expanded(
    flex: 1,
    child: Container(...),
), // Expanded
```



We now see that flex basically adds all the flexes $3+2+1=6$

and it gives 1st : 3 r 3 . Basically fractions of the total sum

2nd : 2 r 1

3rd : 1 r 0

Expanded Widgets on Images

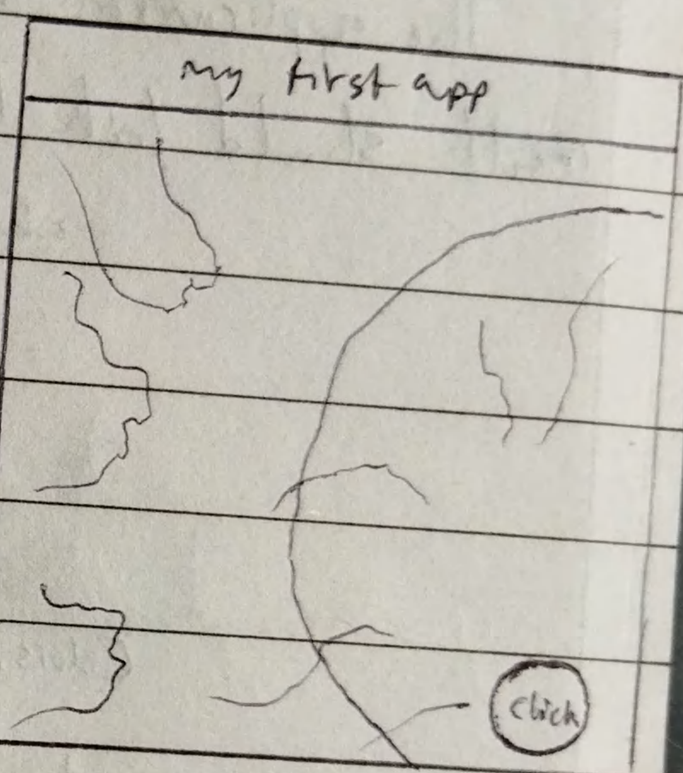
This is another good use case for expanded widgets. Usually Images we use are too big.

FLEX

≡ Add an Asset Image to Our Row ... Image In Row

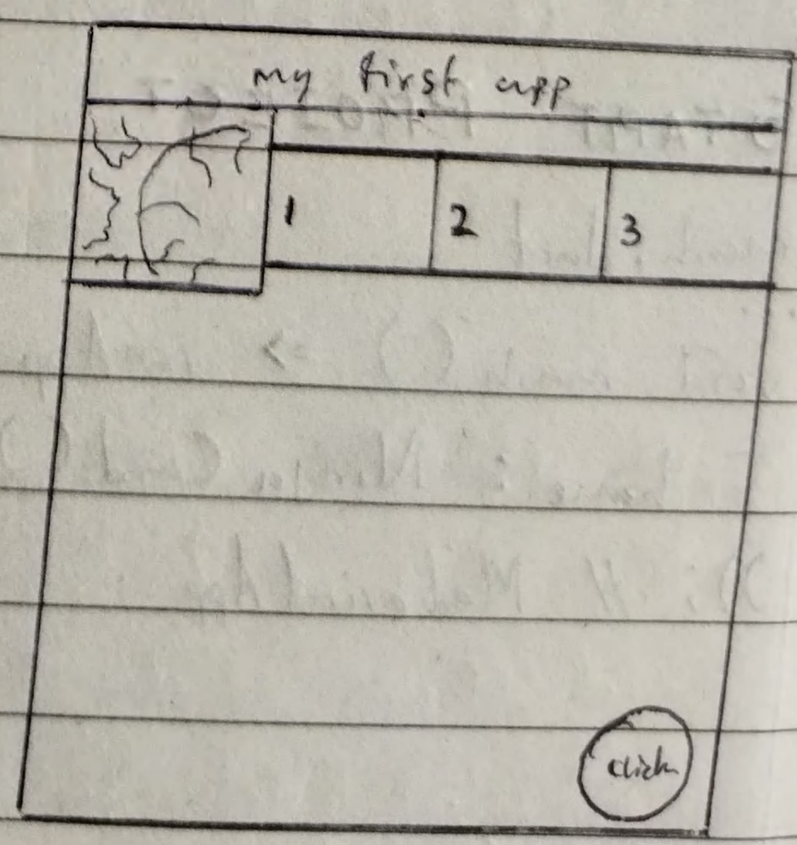
```
body: Row(
  children: <Widget>[
    Image.asset('assets/space-2.jpg'),
    Expanded(...),
    Expanded(...),
    Expanded(...),
  ],//<Widget>[]
),// Row
```

Notice how our expanded widgets are all gone. Thats because the Image covers ~~pushed~~ our expanded widgets out of the screen. We can fix these by wrapping Image in an Expanded widget



≡ Wrap Image with Expanded Widget with all the flex's set to 1

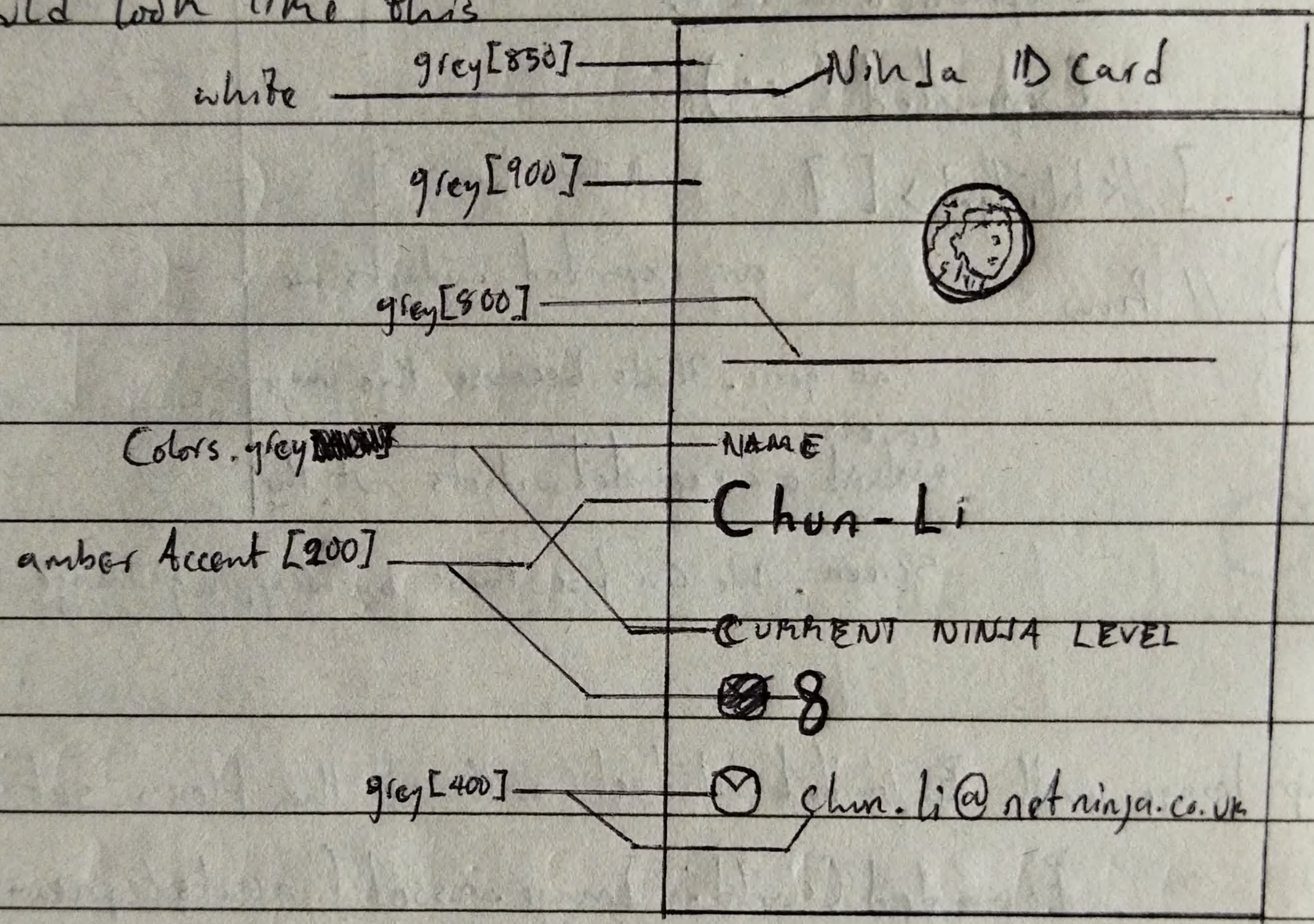
```
Expanded(child: Image.asset('assets/space-2.jpg')),
Expanded(flex: 1, ...),
Expanded(flex: 1, ...),
Expanded(flex: 1, ...),
```



NINJA ID PROJECT #15

Welcome, I think we've learnt enough of the basics of flutter to put everything together and create our very first mini application.

The application we will be building is a "NINJA ID CARD". The result should look like this



≡ START PROJECT

Create New Project

main.dart

```
> flutter create ninja_id
```

```
...
void main() => runApp(MaterialApp(
  home: NinjaCard(),
)); // MaterialApp
```

```
class NinjaCard extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[900],
      appBar: AppBar(                                          HEAD
        title: Text('Ninja ID Card'),
        centerTitle: true,
        backgroundColor: Colors.grey[850],
        elevation: 0.0,   // removes shadow
      ), // AppBar
      body: Padding(                                           BODY
        padding: EdgeInsets.fromLTRB(30.0, 40.0, 30.0, 0.0),
        child: Column(
          children: <Widget>[
            Text(                                              NAME
              'NAME',
              style: TextStyle(
                color: Colors.grey,
                letterSpacing: 2.0,   ▷ // was to closed up
              ) // TextStyle
            ), // Text
```

We will be Handling the Ninja Image later

```
Text (
    'Chun-Li',
    style: TextStyle (
        color: Colors.amberAccent[200],
        letterSpacing: 2.0,
        fontSize: 28.0,
        fontWeight: FontWeight.bold,
    ), // TextStyle
  ), // Text
], // <Widget> []
```

NAME
Chun-Li

Column
cross-axis is default center

≡ Align The "NAME" and "Chun-Li"

```
... ↑
  child: Column (
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget> [...]
```

Aligned →

Ninja ID Card

NAME
Chun-Li

Now We have 2 different values but they are kinda too close to each other, we could apply padding around this things so there is mor space. But let's see an easier way to add space between 2 elements / widgets.

We can use the SizedBox widget to create an empty box for us of a height and width that we specify

≡ Add SizedBox widget Inbetween Your Text Widgets SizedBox widget

```
...
children: <Widget> [
  Text(
    'NAME',
    style: TextStyle(...)), // TextStyle // Text
  SizedBox(height: 10.0),
  Text(
    'Chun-Li',
    style: Text(style(...)), // TextStyle // Text
  SizedBox(height: 30.0),
```

≡ Copy Both Text Widgets Above ('NAME', 'Chun-Li') and paste below, then

```
...
  Text(
    'CURRENT NINJA LEVEL',
    style: TextStyle(...)),
  SizedBox(height: 10.0),
  Text(
    '8',
    style: TextStyle(...)),
  SizedBox(height: 30.0)
], // <Widget> []
```

Ninja ID Card
NAME
Chun-Li
CURRENT NINJA LEVEL
8

≡ Add Row with email Icon and email handle

... CONTINUATION

```
Row (
  children: <Widget> [
    Icon (
      Icons.email,
      color: Colors.grey[400],
    ), // Icon
    SizedBox(width: 10.0),
    Text (
      'chun.li@thenetninja.co.uk',
      style: TextStyle (
        color: Colors.grey[400],
        fontSize: 18.0,
        letterSpacing: 1.0,
      ), // TextStyle
    ), // Text
  ], // <Widget> []
), // Row
```

vertical space between Icon and Text

Ninja ID Card
NAME
Chun-Li
CURRENT NINJA LEVEL
8
chunli@

≡ Lastly It is time to add our Image

- Copy an Image you would like to use (In our case thumb)
- Create a folder in our projects root directory, call it "assets"
- Paste Image Inside the folder → assets
- Now Lets tell flutter we want to use that Image

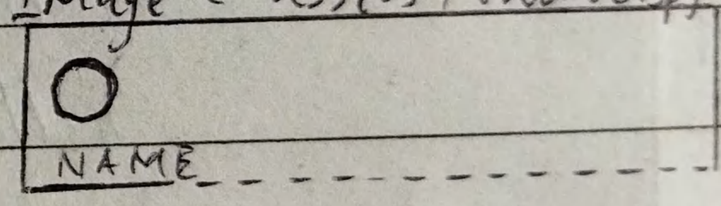
- pubspec.yaml

```
...
flutter:
  ...
  assets:
    - assets/thumb.jpg
```

uncommented

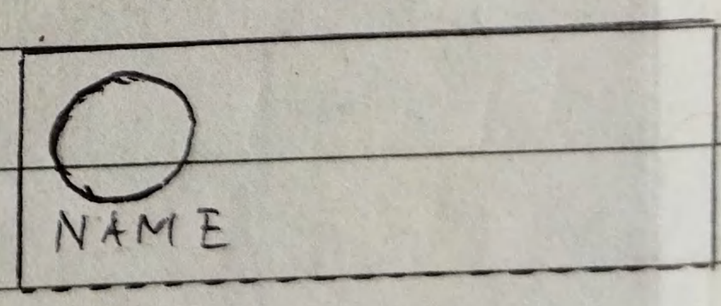
≡ Add Image Now to the top of our Column widget CIRCLE AVATAR WIDGET

```
... ↑
  child: Column(
    ...
    children: <Widget>[
      CircleAvatar(
        backgroundImage: AssetImage('assets/thumb.jpg'),
      ), // CircleAvatar
```



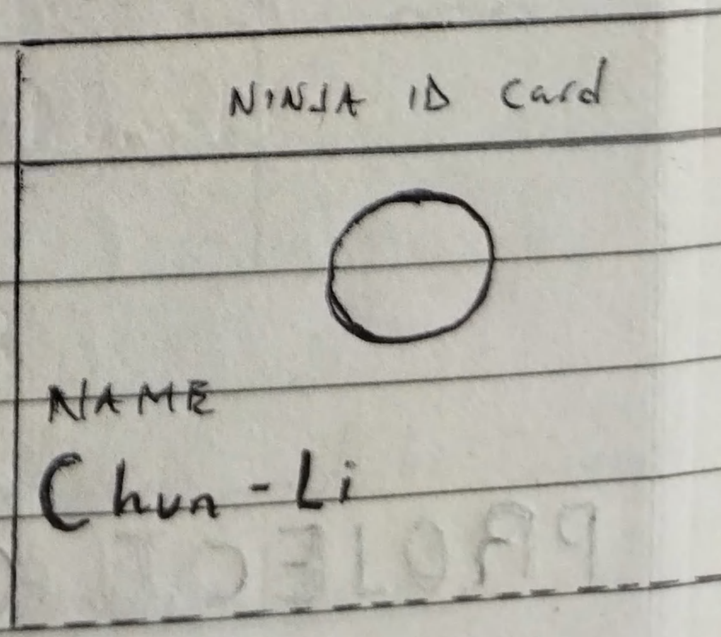
≡ Increase Size of Our Image using radius

```
      ↳ radius: 40.0,
      ), // Circle Avatar
```



≡ Center the Image

```
... ↑ children: <Widget>[
        Center(
          child: CircleAvatar(...),
        ), // Center
        Text(...),
        SizedBox(...),
        Text(...),
```



DIVIDER

≡ FINAL LASTLY: Add a Divider between Our Centered Image and the below widgets

```
... body: Padding(
    padding: EdgeInsets.fromLTRB(30.0, 40.0, 30.0, 0.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Center(...), // Center

        Divider(
          height: 90.0,
          color: Colors.grey[800],
        ), // Divider
```

Ninja ID Card

NAME

Chun-Li

CURRENT NINJA LEVEL

8

chun.li@thenotninja.co.uk

PROJECT COMPLETED

STATEFUL WIDGETS #16

FINAL PROJECT COMPLETION

Now so far we've only used Stateless Widget inside our project, because we've not needed to use any kind of dynamic data or state inside this widget, which would change on the page.

We now want to use StatefulWidget that allows us to have a changing state

Snippet : stful + ENTER

- Create a Stateful Widget In Current Project

STATEFUL WIDGET

Creates 2 classes
1st Class Initializes the 2nd (State)

... ↓

```
class Test extends StatefulWidget {
  @override
  _TestState createState() => _TestState();
}
```

This Instantiates

```
class _TestState extends State<Test> {
  @
  Widget build(BuildContext context) {
    return Container();
  }
}
```

The ACTUAL STATE

- Delete Stateful Widget above. Next we want to Convert our NinjaCard StatelessWidget into a StatefulWidget.

- Click On the NinjaCard Class and hit 💡 Convert to StatefulWidget

≡ Add a FloatingActionButton that when clicked, it increments the CURRENT NINJA LEVEL value

○ main.dart

```
void main() => runApp(MaterialApp(
    home: NinjaCard(),
)); // MaterialApp


class NinjaCard extends StatefulWidget {
  @override
  _NinjaCardState createState() => _NinjaCardState();


class _NinjaCardState extends State<NinjaCard> {

+   int ninjaLevel = 0;
```

will replace our CURRENT NINJA LEVEL constant value of 8 by adding the variable as such: '$ninjaLevel

```
  @override
  Widget build(BuildContext context) {
    return Scaffold( ... ⇓
      Text(
        'CURRENT NINJA LEVEL',
        style: TextStyle(...)), // TextStyle // Text
      SizedBox(height: 10.0),
      Text(
+       '$ninjaLevel',
        style: TextStyle(...)), // TextStyle // Text
```

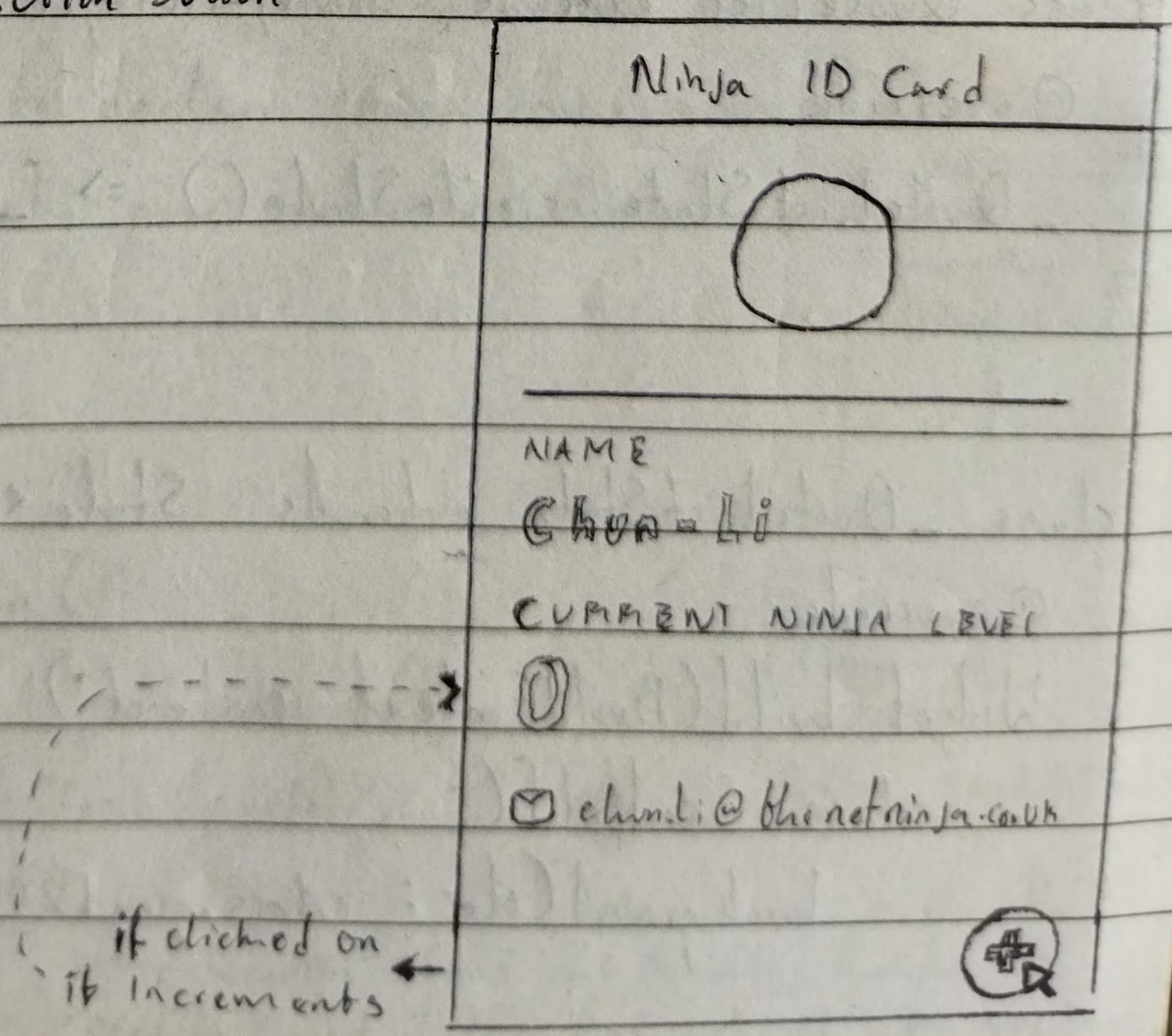
1

≡ Add Floating Action Button and Change the STATE when Clicked

2

... ↑

```
appBar: AppBar (
    ...
),  // App Bar
floatingActionButton: FloatingActionButton (
  onPressed: () {
    setState (() {
      ninjaLevel += 1;
    });
  },
  child: Icon (Icons.add),
  backgroundColor: Colors.grey[800],
),  // Floating Action Button
```



Outputting Lists of Data #17

NEW PROJECT : QUOTES

I want us to move on now and talk about something else in flutter and that's how to cycle through a list of DATA, and output that List of Data in a widget dynamically.

To do this we will be creating a new project

> flutter create quotes

Create Stateful Widget that Cycle's through and outputs a List of DATA

main.dart

```
...1
void main() => runApp(MaterialApp(
    home: QuoteList(),
)); // MaterialApp


class QuoteList extends StatefulWidget {
  @override
  _QuoteListState createState() => _QuoteListState();
}


class _QuoteListState extends State<QuoteList> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: colors.grey[200],
```

```
AppBar (
  appBar : ~~Text('Awesome Quotes')~~
    title : Text('Awesome Quotes'),
    centerTitle : true,
    backgroundColor : colors.redAccent,
  ), // AppBar
  body : Column (
    children : <Widget> [ ]
```

---> This is where we will be outputting our data. We can use a ListView widget to do this but we will learn that later on.

Create a List of about 3 Quotes

... ↑

```
class _QuoteListState extends State<QuoteList> {

  List<String> quotes = [
    'Be yourself; everyone is already taken',
    'I have nothing to declare except my genius',
    'The truth is rarely pure and never simple',
  ];

  @override
  ... ↓
```

Awesome Quotes
Be yourself; ----
I have nothing ----
The truth is ----

```
    body : Column (
      children : quotes.map ((quote) {
        return Text (quote);
      }).toList(),
    ),
```

---> map function returns an Iterable, therefore we need to convert it to a List to avoid getting any Errors.

Assignment
Write this in 1 Line. Readability

CUSTOM CLASSES #18

At the minute, we have a List of Quote and we are outputting each quote to the screen. Now at the minute each Quote is just a Single String but what we want to do now is associate each string / quote with an Author.

To do this we are going to need to create our own quote class. But first create a new dart file for this [quote.dart]

lib > quote.dart

```
class Quote {

  String text;
  String author;

  Quote (String text, String author){
    this.text = text;
    this.author = author;
  }

}
```

INCREASED MODULARIZATION :)

Assignment
Make Code more readable
using named parameter

```
Quote ({ this.text, this.author });
```

main
0 quote.dart

1
+ import 'quote.dart';

...

```
List<Quote> quotes = [
  Quote(author: 'Osca', text: 'Be yourself; everyone is already taken'),
  Quote(author: 'Oscar', text: 'I have nothing to declare ...'),
  Quote(author: 'Jim', text: 'The truth is rarely pure ...'),
]
```

...

```
body: Column(
  children: quotes.map((quote) => Text('${quote.text}'
    '${quote.author}'))
),
```

---> when its an objects property we have to wrap it in curly braces {...}

Awesome Quotes
Be yourself; everyone else is already taken - osca
I have nothing to declare except my genius - oscar
The truth is rarely pure and never simple - Jim

CARDS #19

At the minute, what we are doing is cycling through a list of data and then outputting the text property and author property.

Now Instead of using a Text widget to output this List of data, it would be nice for us to input some kind of better template.

To do this we want to create a new function that basically helps us return that card template.

NOTE: We are going to find a better approach for this later on to generate this card templates/Widgets

Create Function that returns our Card Template Card Widget

main.dart

```
...
  List<Quote> quotes = [...];


  Widget quoteTemplate(quote) {
    return Card(
      margin: EdgeInsets.fromLTRB(16.0, 16.0, 16.0, 0),
      child: Column(
        children: <Widget>[
          Text(
            quote.text
            style: TextStyle(
              fontSize: 18.0,
              color: Colors.grey[600]
            ), // TextStyle
          ), // Text
```

```
SizedBox(height: 6.0),
Text(
  quote.author,
  style: TextStyle(
    fontSize: 14.0,
    color: Colors.grey[800]
  ), // TextStyle
), // Text
], //<Widget>[]
)) // Column // Card }
```

≡ Call Function

```
... ), AppBar
  body: Column(
    children: quotes.map((quote) => quoteTemplate(quote)).toList(),
  ) // Column
), // Scaffold
```

+

Awesome Quotes
Be yourself; everyone else is already taken - Osca
I have nothing to declare except my genius Oscar
The truth is rarely pure and never simple Ilm

≡ Next we want to fix the styling by

- wrapping/giving the card some padding;
- And making the Cards the same in width

```
... ↑ Widget quoteTemplate(quote){
        return Card(
          margin: EdgeInsets.fromLTRB(16.0,16.0,16.0,0),
+         child: Padding(
+           padding: Const EdgeInsets.all(12.0),
            child: Column(
+             crossAxisAlignment: CrossAxisAlignment.stretch,
              children: <Widget>[...]
            ), // Column
          ), // Padding
        ) // Card
      }
```

Awesome Quotes
Be yourself; everyone else is already taken Oscar
I have nothing to declare except my genius Oscar
The truth is rarely pure and never simple Jim

NOTE:

The way that we've done this is not the best way to ~~output~~ generate this card widgets and templates and output them.

A better way to do this would be Extracting our Card Widget into it's own StatelessWidget class and then reuse that widget whenever we need it

EXTRACTING WIDGETS #20

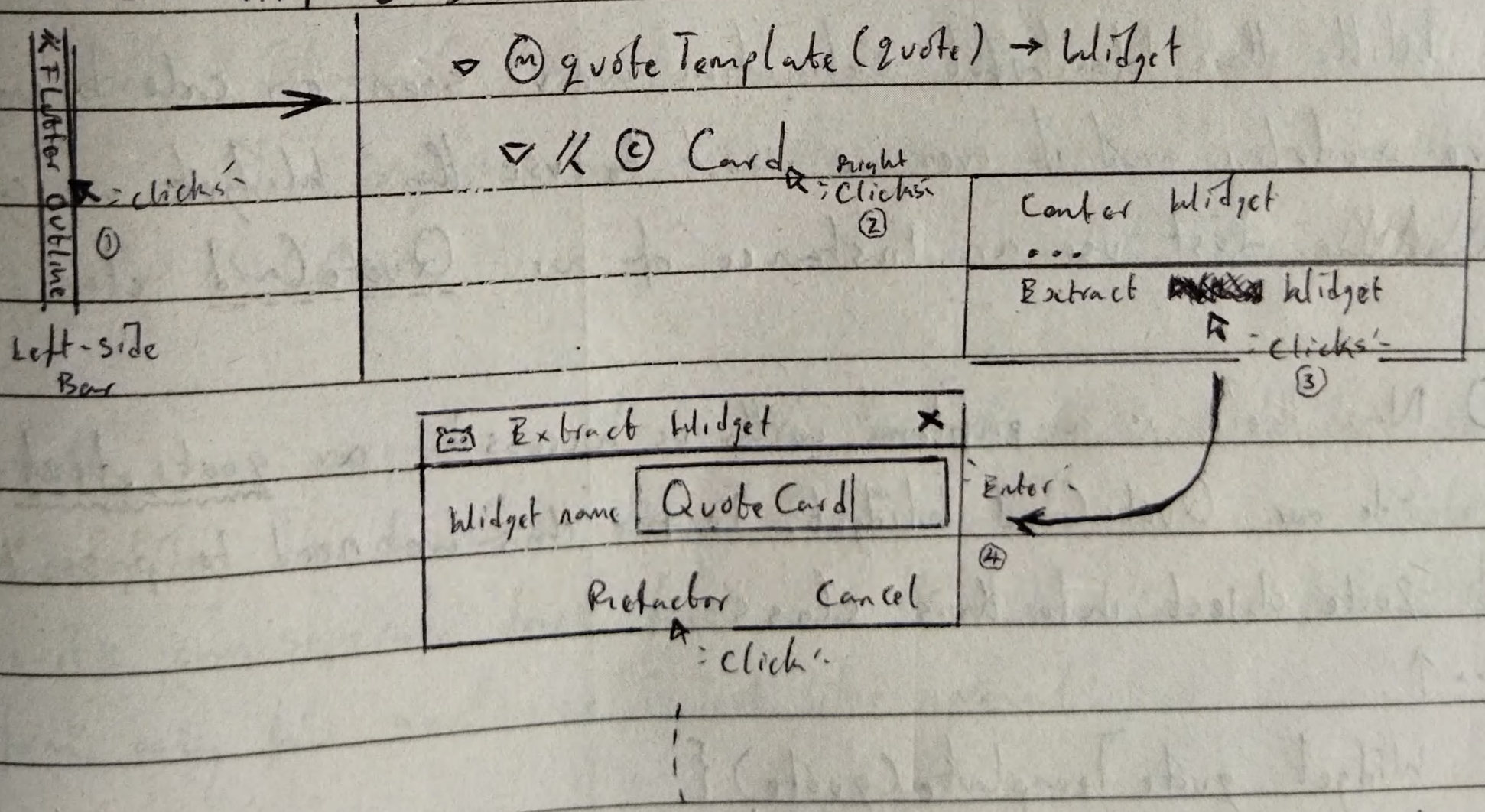
Okay gang, so far we've seen how to map through a list and then for each quote we call a quoteTemplate function that returns a Card Widget as we output that to the screen.

But thier is a better way to do this which helps us modularize our code making it more readable and reusable.

HOW CAN WE DO THIS?

- By creating our own custom stateless widget which would represent our quoteTemplate function Card Widget

A QUICK WAY TO DO THIS? is to



main.dart

... ↓ bottom

To see →

```
class QuoteCard extends StatelessWidget {
  const QuoteCard ({ key key, }) : super (key : key);

  @override
  Widget build (BuildContext context) {
    return Card (
      margin : EdgeInsets.fromLTRB (16.0, 16.0, 16.0, 0),
      child : Padding (...), // Padding
    ); // Card
  }
}
```

With this wonderful extraction we've seen our code becomes more modular and if ever we want to use this Widget, we can just use an Instance of our QuoteCard class

① Now there is a problem with us accessing our quote text inside our QuoteCard Widget. To fix this we need to pass the quote object into this class

... ↑

```
Widget quoteTemplate(quote) {
  return QuoteCard (quote : quote);
}
```

Passed

...↓

```
class QuoteCard extends StatelessWidget {

  final Quote quote;
  QuoteCard({ this.quote });
```

should it works ☑

Create QuoteCard Instance directly in our quotes.map(...).toList(),

...↑

```
Widget quoteTemplate(quote) {
  return QuoteCard(quote: quote);
}
```

---> Copy This!

DELETE!

...

```
body: Column(
  children: quotes.map((quote) => QuoteCard(quote: quote)).toList(),
), // Column
```

pasted!

Lastly, we might want to use this QuoteCard Widget in some other file. To do this means we have to move our QuoteCard to it's own seperate dart file and then we can import it from form any file and use it just like normal.

Doing this Increases Modularity...

Reusability and;
organization;
Abstraction;

Create New Dart File In (Lib) and then Copy QuoteCard and paste it here

quote_card.dart # Newly Created File

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


class QuoteCard extends StatelessWidget {
  ...
}
```

PASTED!
We can now Import it and use it anywhere

Import QuoteCard Now In main.dart to use It

main.dart

```
...2
+ import 'quote_card.dart';
```

- Now Everything Should Still work as normal

Awesome Quotes

Be yourself, everyone else ...
Osca

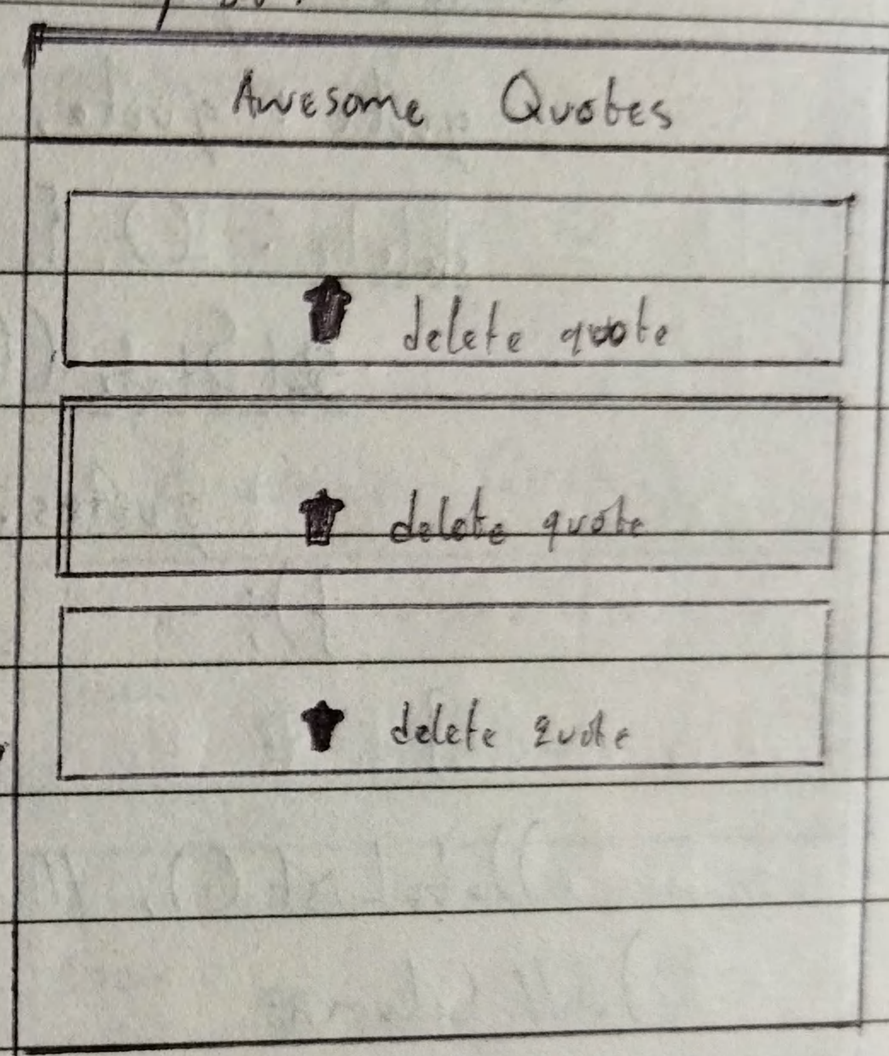
I have nothing to declare ...
Oscar

The truth is rarely pure ...
Jim

FUNCTIONS AS ARGUMENTS #21

DELETING QUOTES

Okay then, so now we have our Quote cards listed on the screen, It would be quite nice to be able to actually delete them.
We would have a button with a delete icon and text and when that gets clicked, gone is that particular post.



≡ Create a Delete Button on our QuoteCard

```
quote_card.dart
...
    return Card (
        ...

        children : <Widget> [
            ... ↓

            ), // Text

            SizedBox (height: 8.0),
            FlatButton.icon (
                onPressed: () {},
                label: Text ('delete quote'),
                icon : Icon (Icons.delete),
            ), // FlatButton.Icon
        ], // <Widget> [ ]
```

≡ Add Functionality That Allows us Actually DELETE quotes

- We can't modify a data (quotes) where it is not defined therefore
- To Delete, we must then pass in a function to quote_card.dart file that allows us delete a card.

≡ Pass Delete Function to our Card Template

○ main.dart

```
... ), // AppBar
    body: Column(
      children: quotes.map((quote) => QuoteCard(
        quote: quote,
+       delete: () {
          setState(() {
            quotes.remove(quote);
          });
        }
      )).toList(), // QuoteCard
    ) // Column
```

≡ Recieve and Declare passed In function In our Cards Template

○ quote_card.dart

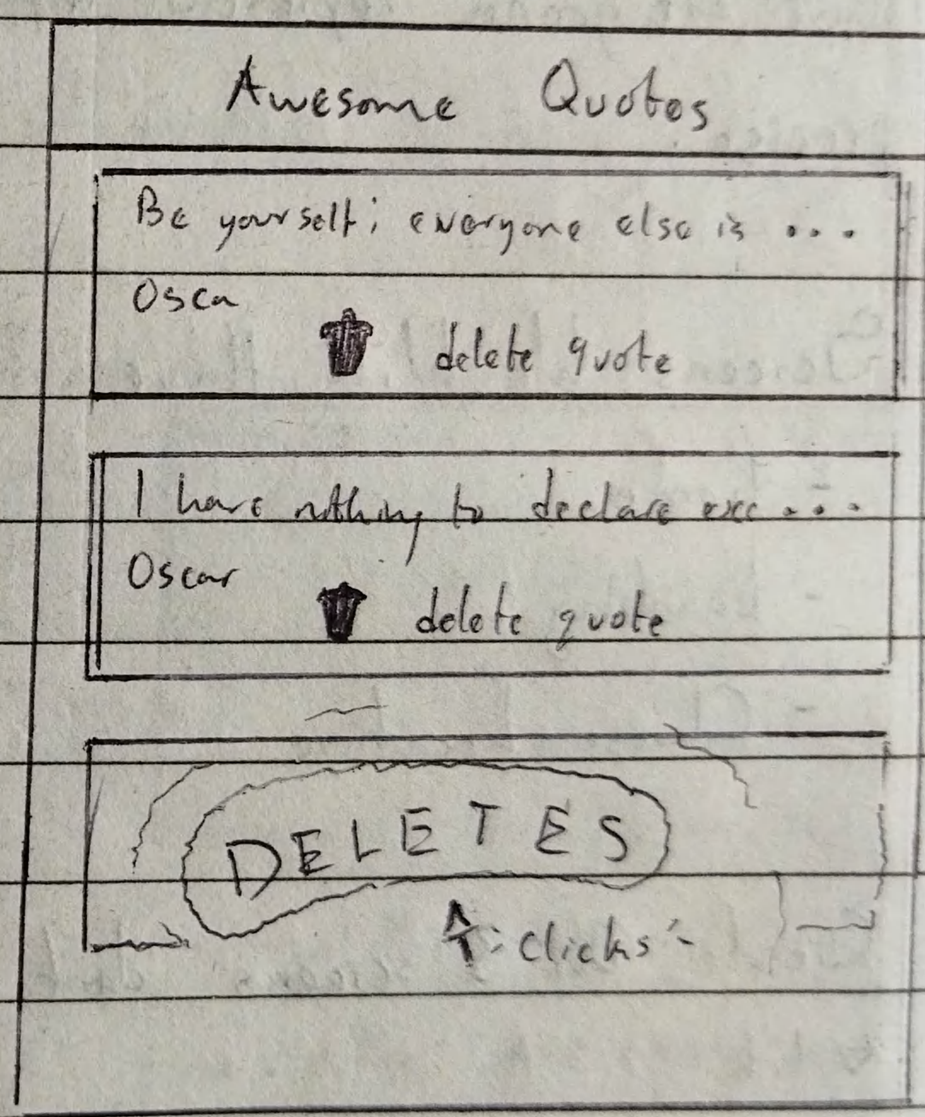
```
... 2

class QuoteCard extends StatelessWidget {

  final Quote quote;
+ final Function delete;

++ QuoteCard({ this.quote, this.delete });
```

Now Let's Call the Delete Function In Card Template To Finish This ...

```
FlatButton.icon(
  onPressed: delete,
  ...
), // FlatButton.icon
```

This now calls the delete function passed in from our main.dart file

It Works ✓



NEXT / 2ND HALF OF COURSE

On the next section we will start to build the world time App.

Starting The World Time App #22

PROJECT

NEW PROJECT : WORLD_TIME

Now I believe we've learnt enough to know about flutter now, so rott let's attack the big project of this course which is the world time App.

> flutter create world_time

We want to now create a couple of different boilerplate widgets that are gonna represent different screens. Three of them to be precise.

Screens We Will Have

- home
- Loading
- Choose Location

Create Our 3 screens dart files in a folder "Pages"

```
▽ lib
  ▽ Pages
      home.dart
      loading.dart
      choose_location.dart
  main.dart
```

≡ Fill All Screens With Their Specific Boilerplates Widgets

○ pages > home.dart

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

class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context){
    return Scaffold(
      body: Text('home screen')
    ); // Scaffold
  }
}
```

home screen

→ We don't want our home screen text entering into the mobile device header bar, because it's not really save to put text there. To fixed those we use the SAFE AREA widget

■ To Preview our Home Screen

○ main.dart

We could also use relative path > 'Pages/home.dart'

```
2
+ import 'package: world_time/pages/home.dart';

  void main() => runApp(MaterialApp(
+     home: Home(),
  )); // Material App
```

■ Fix Overlapping Text using SafeArea Widget

□ Pages > home.dart

```
...
    return Scaffold(
+       body: SafeArea(child: Text('home Screen')),
    ); // Scaffold
```

■ Boiler Plate For choose_location.dart

□ choose_location.dart

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

Because we will be using data in the future to show different locations.
It should be a StatefulWidget

```
class ChooseLocation extends StatefulWidget {
  @override
  _ChooseLocationState createState() => _ChooseLocationState();
}

class _ChooseLocationState extends State<ChooseLocation> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(child: Text('choose location screen')),
    ); // Scaffold
  }
}
```

Boiler Plate For Loading.dart

loading.dart

This will also be a stateful widget because we will have some changing states and data in this widget in the future of this app

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

class Loading extends StatefulWidget {
  @override
  _LoadingState createState() => _LoadingState();
}

class _LoadingState extends State<Loading> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Text('loading screen'),
    ); // Scaffold
  }
}
```

Assignment
Test all this widgets out on your own to see that they all work

Now We Have Our Screens and their Widgets setup, they are now ready for us to be able to navigate to them in the future

Routing & Maps #23

Okay my friend, now we've gotten to a point where we have 3 different screens, and we need a way to navigate these different screens.

To do this we need maps. Now maps in dart is simply a data structure just like object literals in Javascript, and Dictionaries in Python.

≡ Create a map and do some basic operations on it

○ Dart Pad

```
void ~~students~~ main() {
  Map students = {'name': 'chun-li', 'age': 25};
  print(student['age']);

  student['name'] = 'John doe';
  print(student['name']);
```

□ With the little knowledge we have now we can now start working with routes in flutter

Now ~~the~~ routes in flutter is basically a key value pair ~~system~~ data structure where the values are functions that take an argument "Context" and return a Widget/Screen

Create Routes For All Our Screens

main.dart

```
import 'package:flutter/material.dart';
import 'package:world_time/pages/home.dart';
+ "      "      "      "      "      "/loading.dart';
+ "      "      "      "      "      "/choose_location.dart';

void main () => runApp(MaterialApp(
+   initialRoute : '/home'  ---------->  used for testing Purposes
+   routes : {
      '/' : (context) => Loading(),  --->  should be the first screen
                                           but since we used initialRoute it
                                           overides '/' route.
      '/home' : (context) => Home(),
      '/location' : (context) => ChooseLocation(),
    },
));  // MaterialApp
```

Now We've added all our screen routes, next we need buttons that will help us navigate to the next screen or another screen

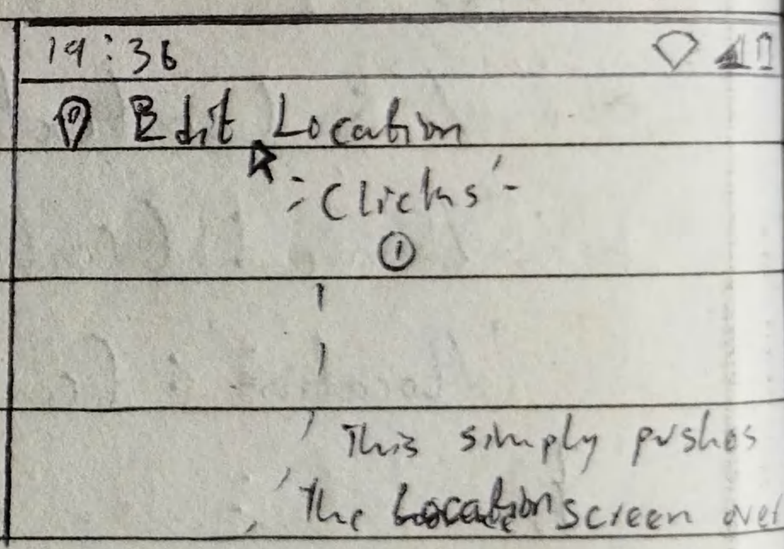
Create a Button On The Home Screen, when clicked takes us to another screen

pages > home.dart

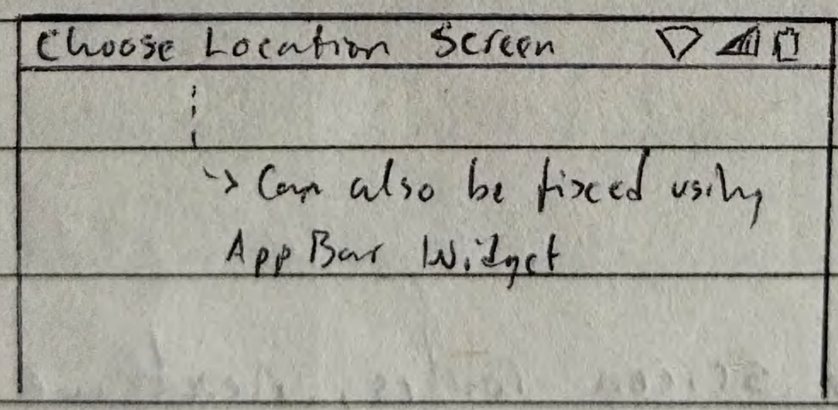
```
...
    return Scaffold(
```

```
body: SafeArea (
  child: Column (
    children: <Widget> [
      FlatButton.icon (
        onPressed: () {
          Navigator.pushNamed(context, '/location');
        },
        icon: Icon (Icons.edit_location),
        label: Text ('Edit Location'),
      ), FlatButton.icon
    ], // <Widget> []
  ), // Column
), // Safe Area
```

Home



This simply pushes the location screen over the home screen. To go back to previous screen we can use the AppBar widget



Choose Location

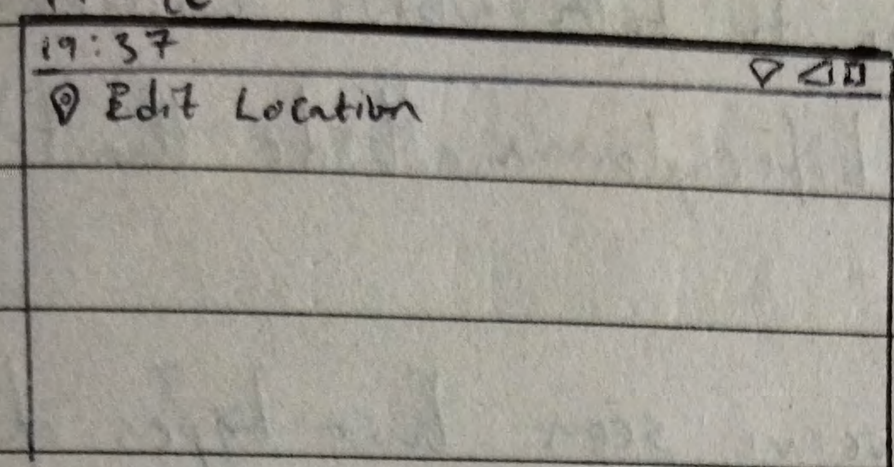
Add AppBar to choose_location.dart

choose_location.dart

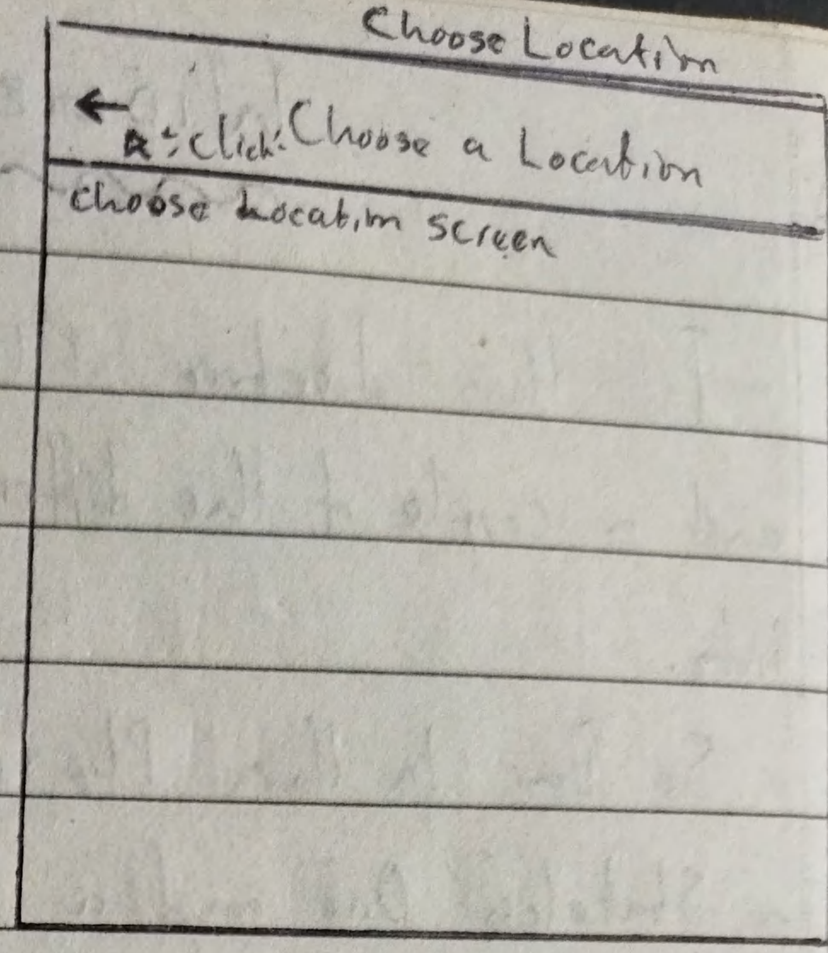
```
...  return Scaffold (
       backgroundColor: Colors.grey[200],
       appBar: AppBar (
         backgroundColor: Colors.blue[900],
         title: Text ('Choose a Location'),
         centerTitle: true,
       ), // AppBar
```

Takes us back to previous screen by pushing current screen off of the screen it's over

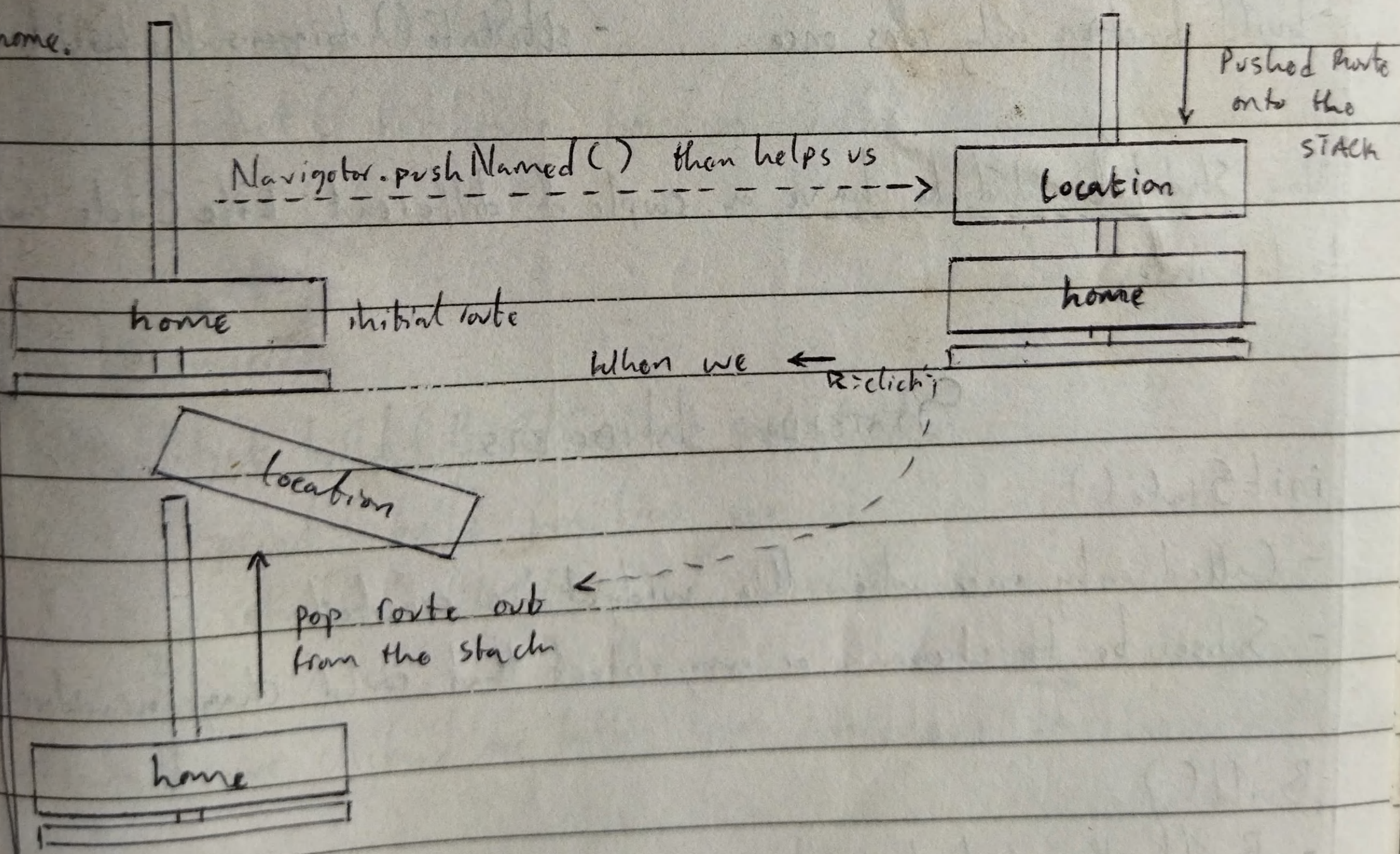
Home



Choose Location



You can think of our App as a stack of screens. Initially we have home.



Widget Lifecycles #24

In this Lecture, I'll like to talk about widget Lifecycle and a couple of the different Lifecycle methods that we can tap into.

So Far In this Playlist we've seen two types of widgets, a Stateless One and a Stateful One.

Stateless	Stateful
- State does not change over time	- state CAN change over time
- build function only runs once	- setState() triggers the build function

Now Statefull Widgets have a couple of different Life Cycle methods to tap into.

Statefull Widgets

initState()
- Called only once when the widget is created
- Subscribe to streams or any object that could change our widget data

Build()
- Builds the widget tree
- A build is triggered every time we use setState()

Dispose()
- When the Widget / State object is removed

EXAMPLE SHOWING THAT:

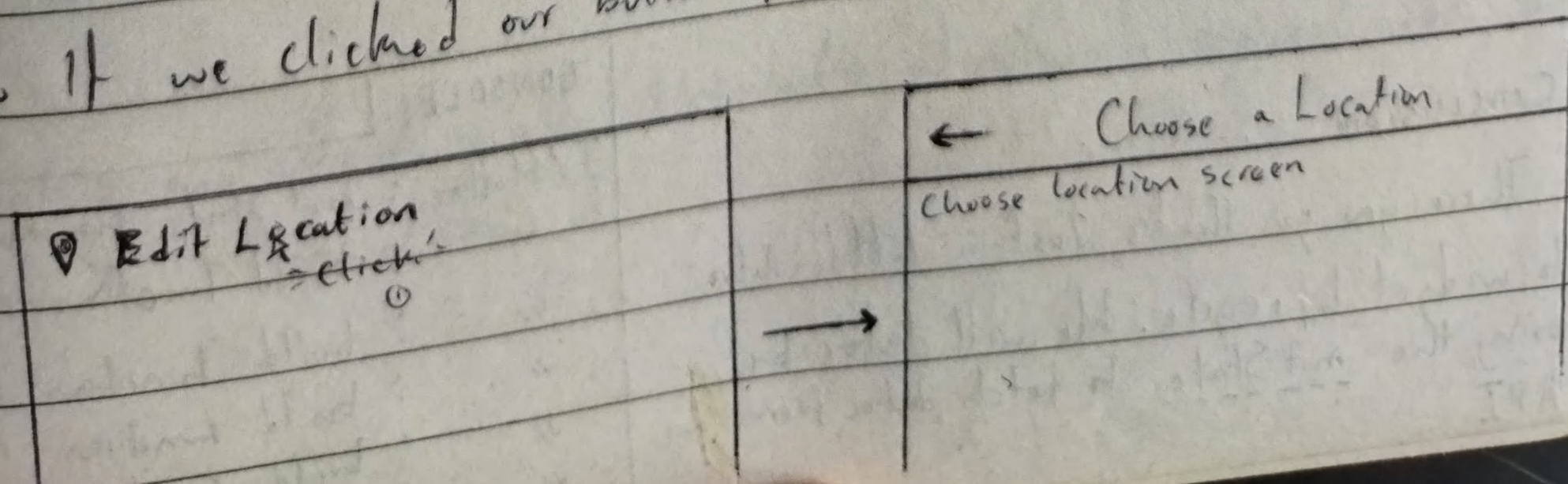
- initState() runs only once on widget creation
- Build() runs any time state changes

Show this Example

choose_location.dart

```
...
class _ChooseLocationState extends State<ChooseLocation>{

  @override
  void initState(){
    super.initState();
    print('initState function ran');
  }


  @override
  Widget build(BuildContext context){
    print('build function ran');
    return Scaffold(...) //Scaffold
  }
```

Now if we clicked our button from home.dart → to choose_location.dart



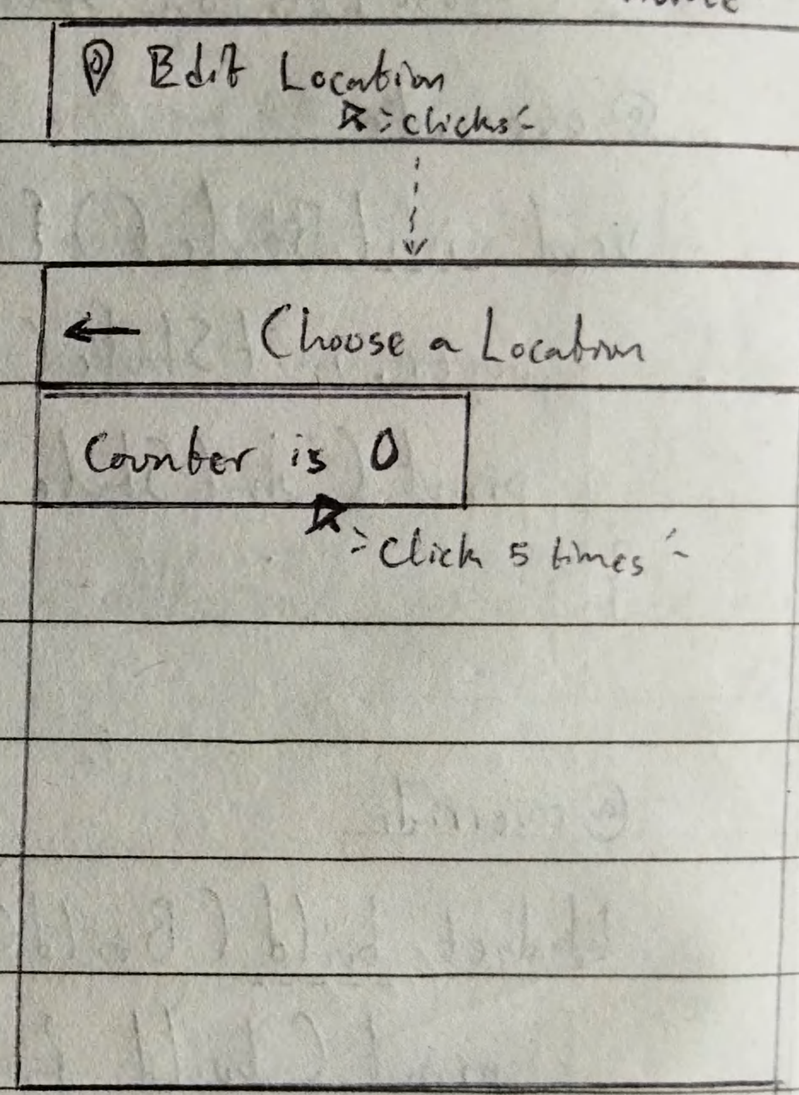
If we check the Console

```
CONSOLE
I/flutter : initState function ran
I/flutter : build function ran
```

- We see that on chooseLocation widget creation both run.

Add Button to ChooseLocation that Increments a value so we can see more clearly the difference

```
... ↑ class _ChooseLocationState extends State<ChooseLocation> {
+   int counter = 0;

    @override
    void initState(){...}
    ... ↓
+       body: RaisedButton(
            onPressed: (){
              setState(){
                counter += 1;
              });
        },
+       child: Text('counter is $counter'),
        ), // RaisedButton
```




```
CONSOLE
I/flutter : initState function ran
~~ : ~~build function ran~~
~~ : build function ran
~~ : build function ran
~~ : build function ran
~~ : build function ran
~~ : build function ran
```

CONCLUSION

There you go that's just a little intro to widget lifecycle. We will later be using the initState to fetch data from an API

Intro to Async In Flutter #25

So hopefully if you are watching this course you already know about Async and Await. You might be from a Javascript background and you might be familiar with how Async and Await works, If you are then this won't be too difficult for you since it's identical

An Asynchronous code is an action that starts now and finishes sometime in the FUTURE.

A Good example of those could be Interacting with an API endpoint to get some data from the Database, which takes some time to do as our code keeps running and doesn't stop

To Handle Asynchronous Code In Flutter, we are going to be using a Combination of

- Asynchronous Functions
- THE Await keyword and:
- Futures {Similar to Promises In Javascript}

Let's go ahead and simulate a network request with some delay In a function

≡ Simulate Api Request

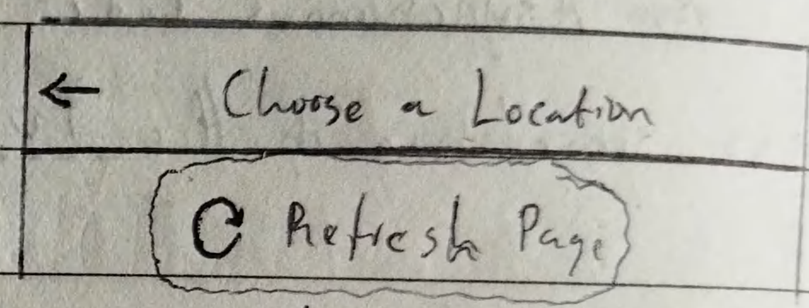
choose_location.dart

```
...
class _ChooseLocationState extends State<ChooseLocation> {

  void getData() {
    // Simulate network request for
    Future.delayed(Duration(seconds: 3), () {
      print('Yoshi');
    }); // Future.delayed
  }

  @override
  void initState() {
    super.initState();
    getData();
  }
```

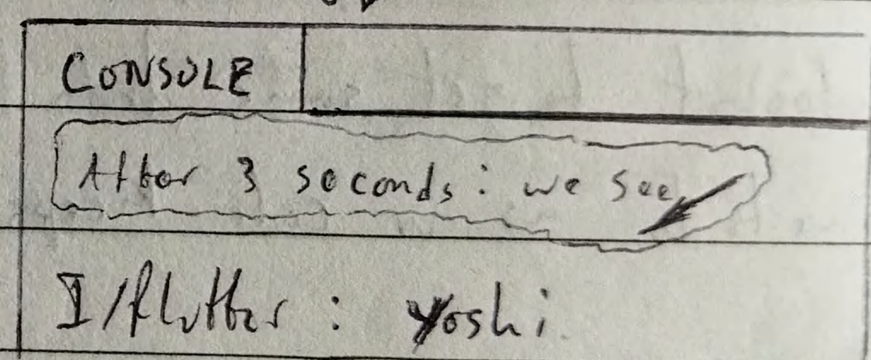
Async Func

1st: Duration object

2nd: function



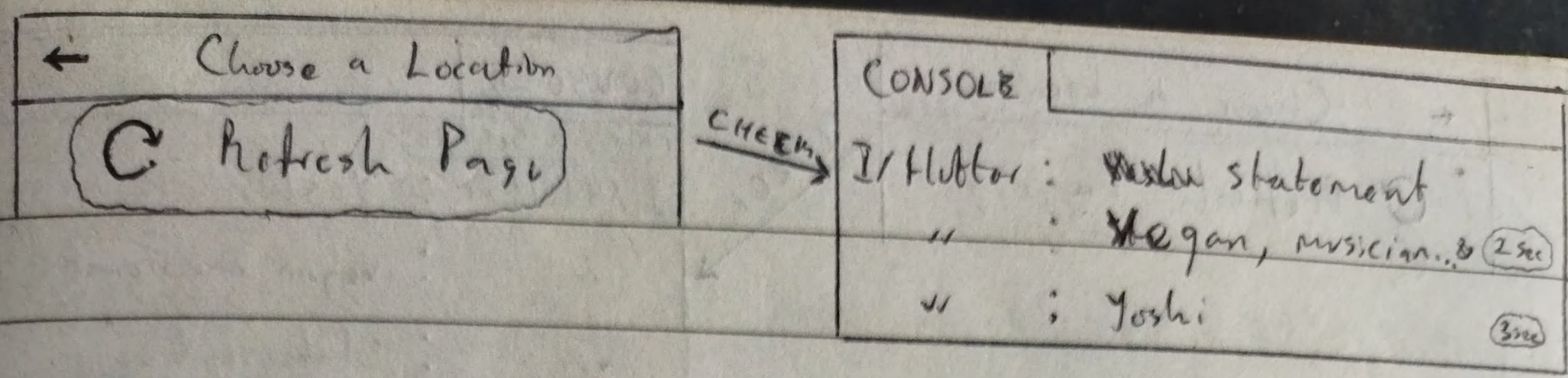
CHECK



≡ Let's Fetch another Data in our get Data Function

```
...↑ void getData() {
      ...↓
      // Simulate network request to get bio of the username
      Future.delayed(Duration(seconds: 2)), () {
        print('vegan, musician & egg collector');
      }); // Future.delayed
      print('statement');
    }
```

Run code



Due to our Future.delayed functions being Asynchronous which means non-blocking -> Statement prints first. Then the 2nd delayed finishes -> Vegan, musician & egg prints second. Then the 1st delayed finishes -> Yoshi and prints last (due to 3 seconds delay)

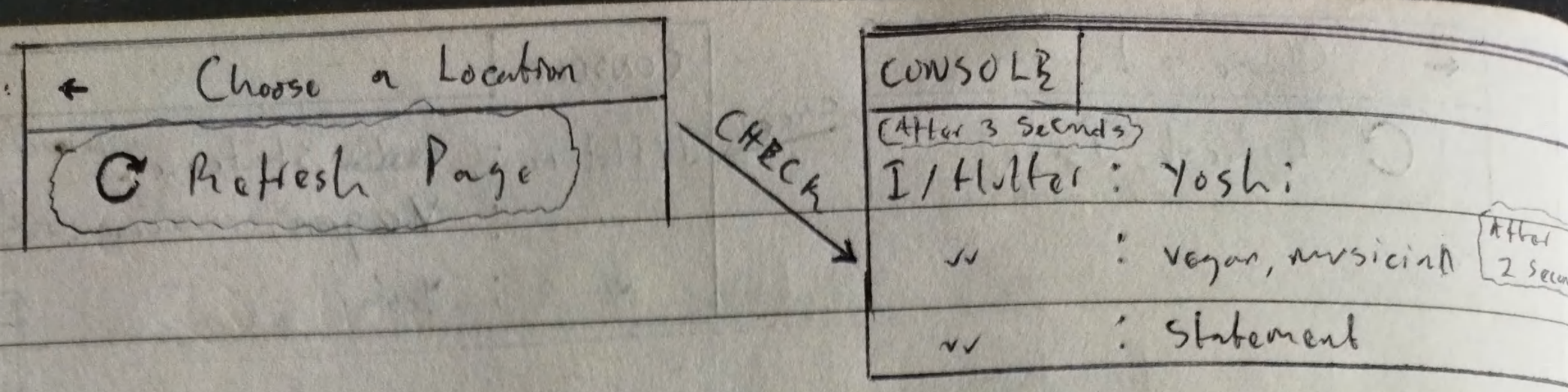
◇ Now We can use Await inside an asynchronous function and what Await Does is

Await : Simply waits for a line or function to complete before continuing the execution of the code

▽ To Do This Lets

- Make our getData() async and then await our Future.delayed

```
...
void getData() async {
  ...
  await Future.delayed(Duration(seconds: 3), () {...});

  ...
  await Future.delayed(Duration(seconds: 2), () {...});

  print('Statement');
}
```



We use Await mostly when 1 request depends on another

Return Username and bio into a variable and print them out

```
...
void getData() async {

  ....
  String username = await Future.delayed(Duration(seconds: 3), () {
    return 'Yoshi';
  }); // Future.delayed

  ....
  String bio = await Future.delayed(Duration(seconds: 2), () {
    return 'vegan, musician & egg collector';
  }); // Future.delayed

  print('$username - $bio');
}
```

Assignment
- Guess result and then run

CONCLUSION

Hopefully that explains Asynchronous code a little bit to you, As we will be using this alot in the future when working with a 3rd party Api to get the information.

NEXT UP - Flutter Packages, To make these network requests we will be using a package that's really going to help us.

FLUTTER PACKAGES #26

Ocassionally when we are creating flutter apps, we are going to need to add in some complex functionality. Which could be to implement some form of animation or maybe to work with Files and Folders on the device or just something that is going to require us to write alot of code and logic. - Yes we could do this all by ourselves, but then we could use flutter Packages ?, that other developers have already timedly written.

They are normally used to implement some kind of functionality in inside our own apps like a

- sliding menu or - date picker or - file uploader e.t.c.

Though their are several flutter packages the one that we will be using is the HTTP PACKAGE

And we could use this package to make any kind of network request to different API's and endpoints.

Eventually we are going to be using a 3rd party API to get time information for different world locations

pub.dev/flutter ; visit ①

Pub.dev

Flutter Packages

http | ; ENTER ② | Q

Pub.dev

758 results for http

http 100

click ③

A composable, cross-platform, Future-based Api for making HTTP requests

http 0.12.0+2

Published Apr 3, 2019 FLUTTER WEB OTHER

Readme changelog Example Installing Version

; Clicks ④

Use this package as a Library

1. Depend on it

Add this to your package's pubspec.yaml file:

```
dependencies:
  http: ^0.12.0+2
```

-> Copy and paste in our pubspec.yaml file -> under dependencies

pubspec.yaml

```
dependencies:
  http: ^0.12.0+2
```

// Now if we save flutter helps us get this package

≡ Instead of using the choose_location.dart file to Load our data for this app, we want to do this in the loading Screen, so let's:

- Move `void getData() async {...}` and `@override void initState() {...}` to <u>loading.dart</u>

0 ~~Main.dart~~ loading.dart

```
...
class _LoadingState extends State<Loading> {

+   void getState() async {
        // Cleaned Up
    }

+   @override
+   void initState() {
        super.initState();
        getData();
    }
```

≡ NEXT: We want to use the <u>Http</u> package to get some data from JSONPlaceholder - which gives us back some fake JSON/dummy data.

Endpoint to be Used For Testing:

'https://jsonplaceholder.typicode.com/todos/1'

≡ Import Http Package and Use that to ~~fetch~~ get data from our Endpoint

○ Loading.dart

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

void getData() async {
  get('https://jsonplaceholder.typicode.com/todos/1');
```

We want to wait for this data to be fetched and then store it into a variable of type Response which is also given to us by Http and allows us to access the Json Response Body

```
  Response response = await get('...');
    print(response.body);
}
```

≡ Change initialRoute to '/', So we can test this Out

○ main.dart

```
...
void main() => runApp(MaterialApp(
  initialRoute: '/home',
  ...
)
```

▽ Now if we refresh our app

CONSOLE

```
I/flutter : {
            "userId": 1,
            "id": 1,
            "title": "delectus aut autem
            "completed": false
            }
```

JSON

jsonDecode

Now Our Response is still in JSON format, Convert this into dart map using jsonDecode.

loading.dart

```
...2
+ import 'dart:convert';
...

void getData() async {
    Response response = await get('...');
Map data =
+   jsonDecode(response.body);
}   print(data);
```

Import required to use jsonDecode.

CONSOLE

```
I/flutter : {userId: 1, id: 1, title: delectus aut autem, completed: false}
```

Now that we've converted the JSON response to dart's map, we can now access the properties

E.g

```
print(data['title'])
```

CONSOLE

```
I/flutter : delectus aut autem
```

So Now we've seen how to work with asynchronous code and the Http package and how they work together

NEXT - Let's try getting the Data we actually need for this application using the World Time API.

WORLD TIME API & DATE TIMES #27

Ultimately we want to show time data on our app and to do that we are going to have to use some kind of API to get time data. I am going to be using a free API called

← → ⟳ worldtimeapi.org

WorldTimeAPI

Simple JSON/plain-text API to obtain the current time in, and related data about, a time zone.

00 If You Scroll To The Bottom ↓

/timezones

RESOURCES
- OpenApi Spec for the API
- ...
- Canonical Timezone List
  ↖ 'click'

00 Takes us to a List With All the Locations we can make requests

Lets checkout London

/timezones

- ...
- Europe/London
  ↖ 'click'
- ...

00 Scroll Down To UTC section

/timezone/Europe/London

UTC

Current UTC Date/Time (ISO8601)

2019-08-22T16:44:21.628814+00:00

T:h:m:s

Local Time's UTC offset

+01:00 - - - - - - - - we add this offset to our Time: hours to get the actual time in any location

00 Scroll Up To see that We Can get this Data In JSON or Plain text.

CURRENT TIME IN EUROPE/LONDON

The defaults to Europe/London are as follows. Also available ... JSON API or Plain text

:click:

Endpoint we will be Using ...

JSON FORMAT /api/timezone/Europe/London

week_number: 34

utc_offset: "+01:00"

utc_datetime: "2019-08-22T16:44:56.428718+00:00" ...↓

≡ Time To Actually Fetch this Data from the Above Endpoint

loading.dart

+ ... void getTime() async {

// Rename function and Clear all

}

@override

void initState(){

...

remove ← - - getData() -> getTime();

```
void getTime() async {
    // make the request
    Response response = await get('.../api/timezone/Europe/London');
    Map data = jsonDecode(response.body);
    print(data);
```

CONSOLE

```
I/flutter: {week_number: 34, utc_offset: +01:00, utc_date time ...
... ... ... ... ... ... ... ... ... ...
datetime: 2019-08-22T:46:37.296404+01:00, ... , ... }
```

```
// Print(data)
```

We will only be working with utc_offset and datetime

```
// get properties from data
String datetime = data['datetime'];
String offset = data['utc_offset'];
print(datetime);
print(offset);
```

CONSOLE

```
I/flutter: 2019-08-22T17:49:18.032206+01:00
           : +01:00
```

Next: We need a way to add ~~that~~ offset to our datetime in order to get the actual time.

TO DO THIS: We need to convert datetime into a DateTime object
- cf giving us access to the properties and methods in this object

```
--->// print (date time);
--->// print (offset);
```

```
// create DateTime object
DateTime now = DateTime.parse (date time);
print (now);
```

Converted ☑ CONSOLE

```
I/flutter : 2019-08-22 16:50:47.151062Z
```

add

Next: We can use a method provided by the DateTime Object to add the offset to our DateTime Object hours

TO DO THIS:

- Use the .substring( ) method to extract "01" from "+01:00"
- Convert "01" to 01 using int.parse() method
- Then add that to our DateTime Object using .add( ) method that takes Duration( ) method with takes hours, minutes, seconds

```
// Create DateTime Object
DateTime now = DateTime.parse (date time);
now =
now.add( Duration (hours: int.parse (offset. subString (1, 3)))),
print (now);
```

If we run this, we now get the Correct time

```
I/flutter : 2019-08-22 17:54:42.061946Z
```

--> updated to the correct time with offset

Creating A WorldTime Class #28

Alright Guys, so In this lecture what I want to do is to separate our getTime() function into it's own seperate class.

By doing so we make our code much more reusable because the that class can be imported to any class or widget or file that wants to use it in the future

≡ Create new folder "Services" and dart file "world_time.dart"

▽ lib

+ ▽ serviceces

+ world_time.dart

≡ Create our WorldTime class or blueprint in

○ world_time.dart

```
+ import 'package:http/http.dart';
+ import 'dart:convert';

+ class WorldTime {

  String location; // location name for the UI
  String time; // the time in that location
  String flag; // url to an assert flag icon
  String url; // location url for api endpoint
```

≡ Before Adding A WorldTime Constructor: Bring In the getTime() func

```
...
class WorldTime {
    ...

pasted
+   void getTime() async {
        // make the request
        Response response = await get('http://worldtimeapi.org/api
                                        /timezone/$url');
        Map data = JsonDecode(response.body);

        // get properties from data
        String datetime = data['datetime'];
        String offset   = data['utc_offset'].substring(1, 3);

        // Create DateTime Object
        DateTime now = DateTime.parse(datetime);
        now = now.add(Duration(hours: int.parse(offset)));
```

LASTLY: Location, ~~time~~, flag, url are going to be set by us when we create an object (using our future Constructor :D). Leaving the time blank → It would make perfect sense to set our time in this function then

```
        // set the time property
+       time = now.toString();
    }
```

Create WorldTime Constructor

```
...
class WorldTime {
    ...
    WorldTime ({this.location, this.flag, this.url});

    void getTime() async { ... }
}
```

NEXT: We want to ~~import the class and~~ create an instance/objects of the WorldTime class all in our loading screen - P.S as our application loads

loading.dart

```
...
class _LoadingState extends State<Loading> {

    void setupWorldTime() {
        WorldTime instance = WorldTime(location: 'Berlin', flag:
            'germany.png', url: 'Europe/Berlin');
        ~~instan~~
        instance.getTime();
    }

    @override
    void initState() {
        super.initState();
        setupWorldTime();
    }
```

this async function now helps you get and set the time for you in object

In other to print out the time we need to add an <u>await</u> to our custom asyn function

```
...
  void setupWorldTime() {
    ...1
    --- await instance.getTime();
        print(instance.time);
```

----> getTime() needs to Complete for this line to work

---> But this won't work because we haven't wrapped our getTime() async function with the Future<T> keyword and make current function setupWorldTime asynchronous

Wrap our getTime() function with Future

World_time.dart

```
...
  WorldTime({ ... });
```

```
Future<void> getTime() async { ... }
```

What Then is A Future?

A Future is a bit like a promise in Javascript, and it means that it is a placeholder value until the function is complete.

In our case it wraps our void and says "hey there, look at some point I am going to return void but only when our asynchronous function is fully complete"

In Summary A Future is a temporary place holder value, the kind that let's dart know when an asynchronous function is COMPLETE.

SWE knowledge :

If you ever want to use <u>await</u> on a custom asynchronous function - you must wrap it with the <u>Future keyword</u>.

LASTLY : We need to make our setupWorldTime Asynchronous

loading.dart

```
... void setupWorldTime () async {
        WorldTime instance = WorldTime(...);
        await instance.getTime();   {This now becomes blocking;}
        print(instance.time);   {This should now work;}
    }
```

WORKS ☑ CONSOLE

```
I/flutter : 2019-08-23 12:13:28.146562 Z
```

Create a variable that shows "Loading..." initially and then after we've gotten our <u>time</u> replace "Loading..." with "$time"

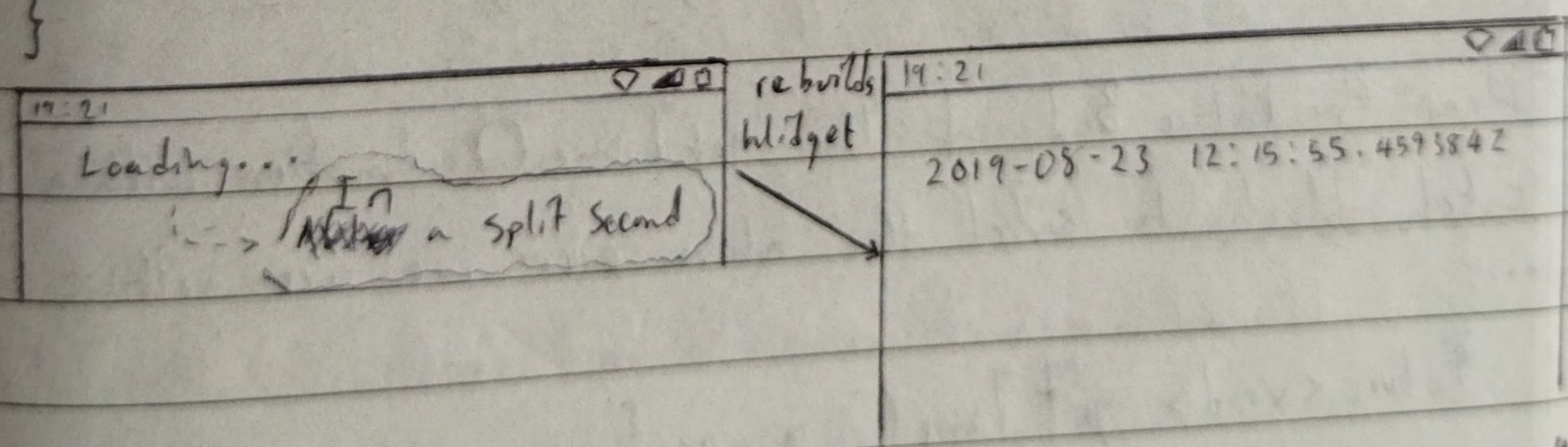
```
...↑
class _LoadingState extends State<Loading> {

    String time = 'Loading...';   // State Value to be displayed

    void setupWorldTime () async {
        ...↓
```

--> Remember setState is what triggers a rebuild of our stateful Widget

```
setState (() {
    time = instance.time;
});
}
```

Display the time Variable with a Simple UI

```
... @override
Widget build (BuildContext context) {
  return Scaffold (
    body: Padding (
      padding: EdgeInsets.all (50.0),
      child: Text (time),
    ), // Padding
  ), // Scaffold
}
```



THE IDEA

The Idea is that whenever a user chooses a different location in the future from a List of locations - Then we are going to create an instance of our WorldTime Class and then go about getting the Time of that location and updating it on the Home Screen

Thats ULTIMATELY OUR GOAL

HANDLING ERRORS #29

Currently our code all seems to be working correctly, but what if along the line we made a mistake, by maybe passing the wrong endpoint. Then our code will fail!

Demo this by making our Endpoint invalid

world_time.dart

```
... Response response = await get('http://worldtimeapi.org/api/
                                    timezones/$url');
```

CONSOLE

```
E/flutter:  Chain of Errors
 "      :  Chain of Errors
```

We can Remedy this by using a try and catch block.

Wrap The Entire Body of The getTime() function with a try and catch block to catch the above error we are getting ...↑

```
Future<void> getTime() async {
    try {
        ...        (Entire body)
    }
    catch (e) {
        print('Caught error : $e');
    }
```

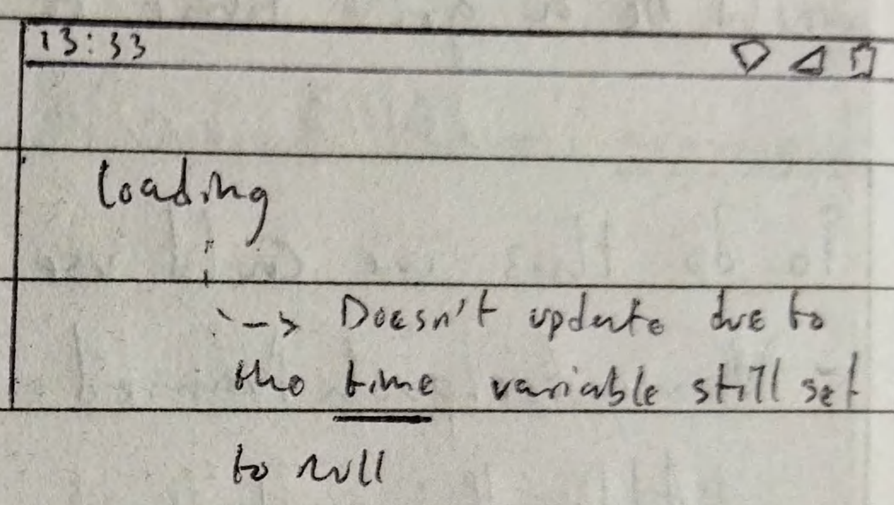
CONSOLE

```
I/flutter: caught error: type 'List<dynamic>' is not a subtype of
    type 'Map<dynamic, dynamic>'
```

which is pointing to this line that doesn't return whats expected

Map data = jsonDecode(response.body);

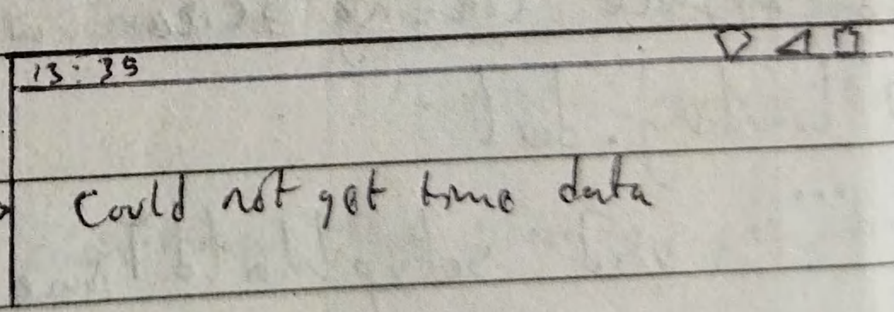
Note that our screen still displays loading



Set time to something else in the catch block. To Fix

```
...
catch (e) {
    print('caught error: $e');
    time = 'could not get time data';
```

Changes from loading to this



DONE: We can go back and fix our Endpoint now

CONCLUSSION

In the Future we will be replacing our Loading string with a spinner. The Next to do is to redirect user to the homepage when we get this data so that we can actually display it there.

To this we will learn How to pass data from a Widget to Another.

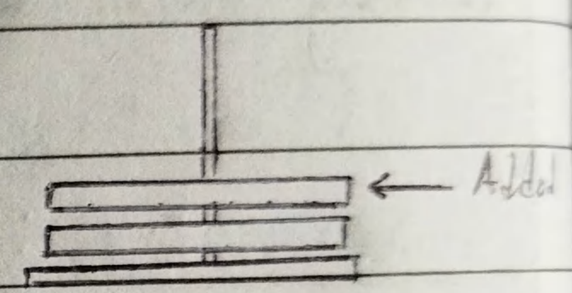
Passing Route Data #30

Our loading screen is the first widget that loads when we start the app and what we are doing is just getting the time and outputting it on our loading screen.

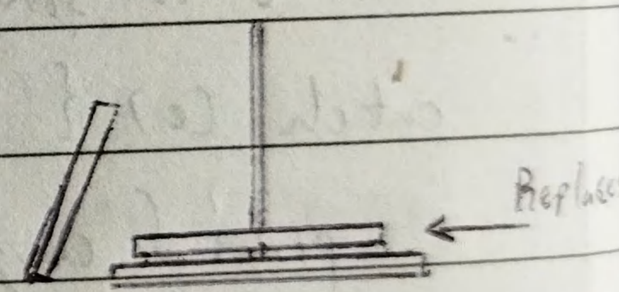
But now once we finish getting the data on our loading, it would be a good time to redirect to the home page

To do this we could use either

- Navigator.pushNamed
  Adds the route ontop of the current



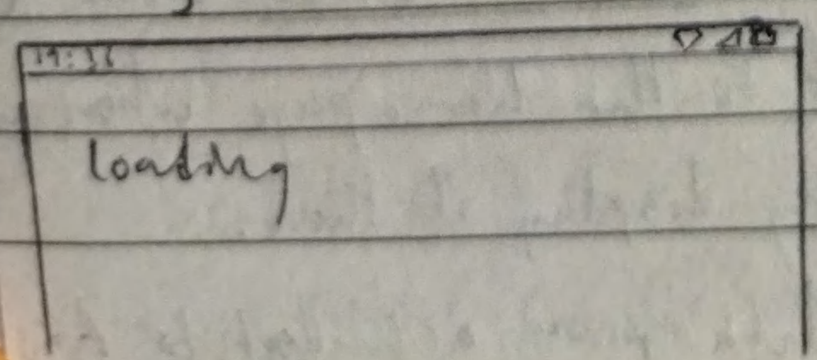
- Navigator.pushReplacementNamed
  pushes/replaces the current route



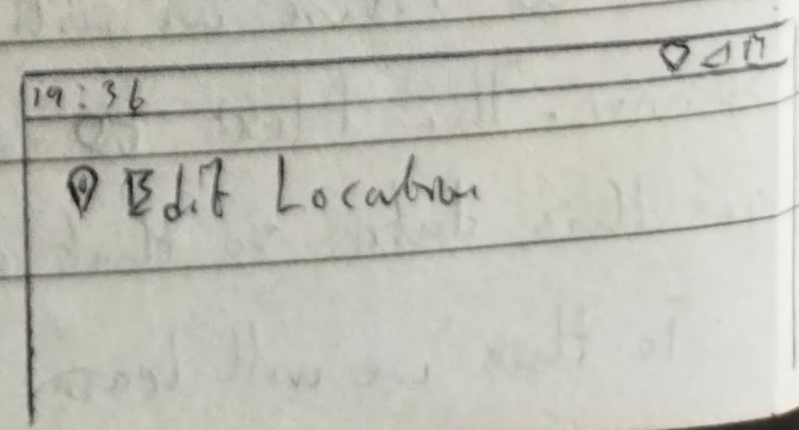
≡ Replace loading screen with home after data has been fetched

☐ loading.dart

```
...
    void setupWorldTime() async {
        WorldTime instance = WorldTime(...)
        await instance.getTime();
        Navigator.pushReplacementNamed(context, '/home');
    }
```



After Data is fetched. We replace the loading



Now We need the Data Inside our next screen but we don't have it. Luckily for us their is a way to send this data to another screen and that is by adding a 3rd parameter

Pass Data From Loading to Home

...

```
Navigator.pushReplacementNamed(context, '/home', arguments:{
  'location': instance.location,
  'flag': instance.flag,
  'time': instance.time,        // we won't be needing the url
});
```

Access the Passed In Data

home.dart

```
class _HomeState extends State<Home> {

  Map data = {};                // didn't work-remove

  @override
  Widget build(BuildContext context) {

    final data = ModalRoute.of(context).settings.arguments as Map;
    print(data);

    return Scaffold(...);
  }
```

This now grabs the passed in data which is of Type Map

CONSOLE

```
I/flutter: {location: Berlin,
flag: germany.png, time:
2019-08-23 12:58:51.19870
42}
```

Formatting & Showing Dates #31

Okay, now we've gotten to a point in our application where we have access to our data in the home screen and including the time which we ultimately want to output to the screen. But it kinda looks a bit crudy now ↓

2019-08-23 12:58:51.698704Z

We want to format this time in a different way that looks a bit better. To do this we will be using a package called

Intl

∃ Go over to pub and Install this - using the guide in Lecture #27

Pub.dev/...

intl 0.15.8

Published May 22, 2019 | FLTTER | WEB | OTHER

Readme Changelog **Installing** Versions (86)

...

```
dependencies:
  intl: ^0.15.8
```

--> Copy this and paste in our Pubspec.yaml file to Install

Next: We want to use our Intl package

≡ Format our time using the newly installed intl package

☐ world_time.dart

Let's do this in world_time.dart to keep this DRY

```
...1
+ import 'package:intl/intl.dart';
...
```

Let's now use a function this provides

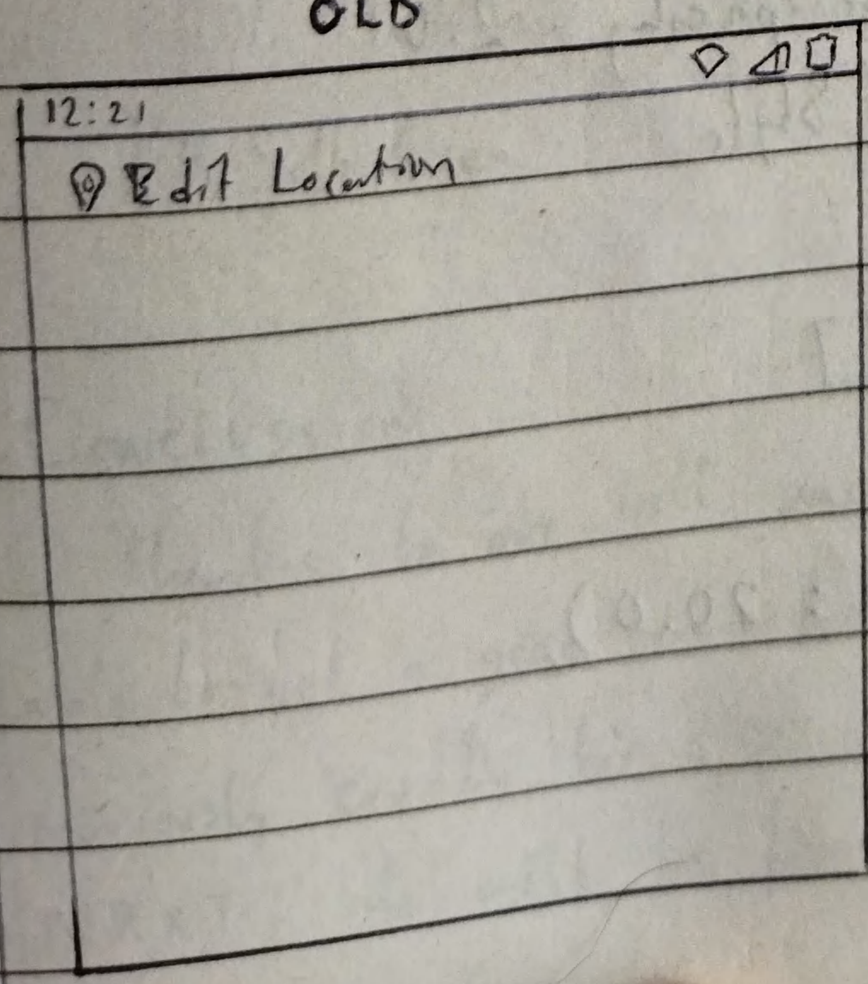
```
  // Set the time property
+ time = DateFormat.jm().format(now);
}
```

```
I/flutter: {location: Berlin, flag: germany.png, time: 1:20 PM}
```

This looks alot more readable, and this is what we are going to output to the users

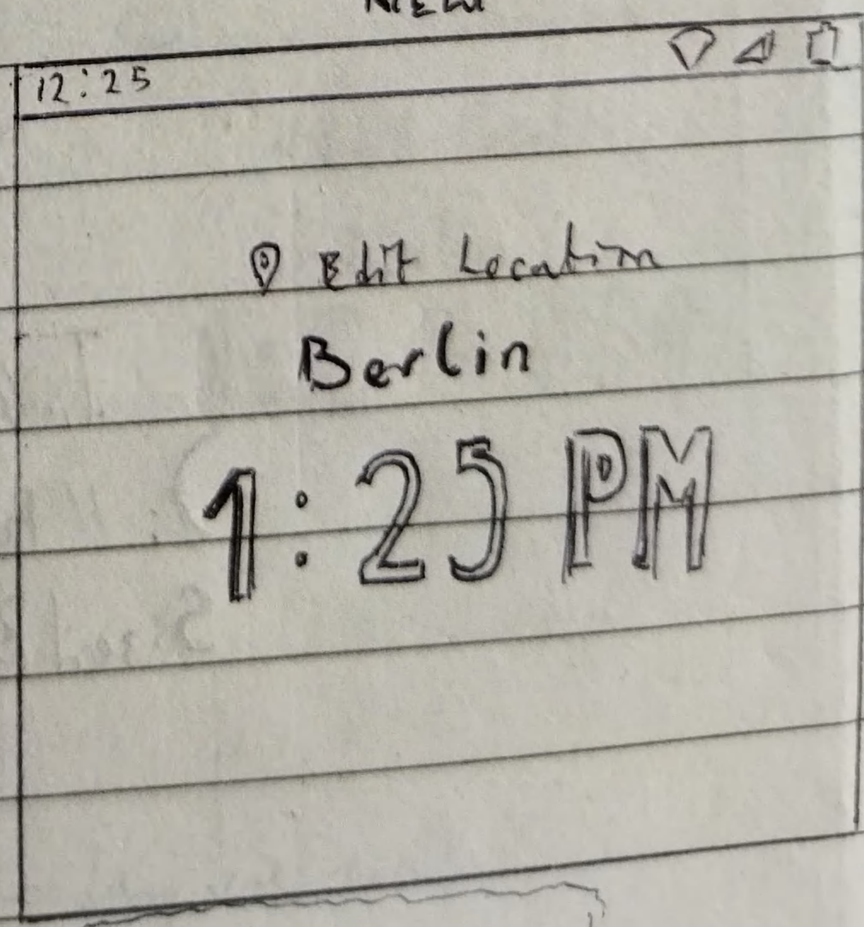
≡ NEXT: Let's Transform our Home from looking like OLD to NEW

OLD



to →

NEW



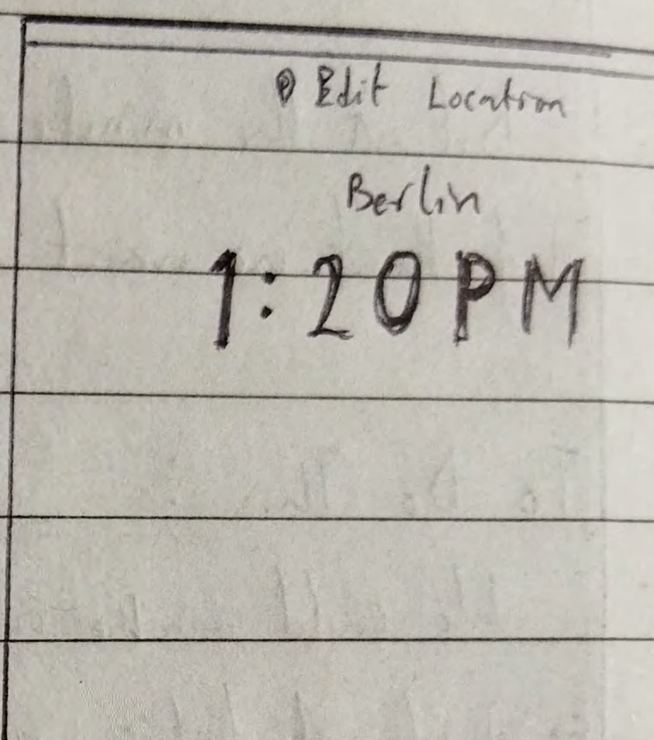
Assignment
Try Achieving this

home.dart

```
... return Scaffold(
      body: SafeArea(
        child: Column(

          children: <Widget>[
            FlatButton.icon(...), // FlatButton.icon
            SizedBox(height: 20.0),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                Text(
                  data['location'],
                  style: TextStyle(
                    fontSize: 28.0,
                    letterSpacing: 2.0
                  ) // TextStyle
                ), // Text
              ], // <Widget>[]
            ), // Row
            SizedBox(height: 20.0),
```

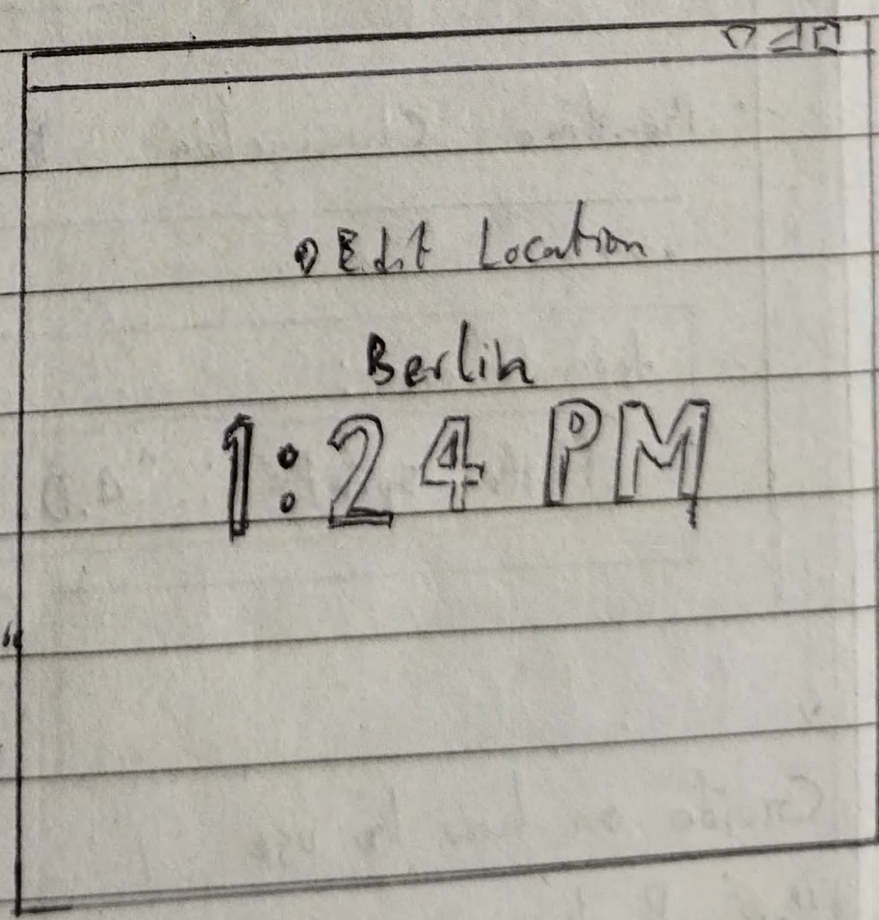
Berlin

```
Text (
    data ['time'],
    style : TextStyle (
        fontSize : 66.0,
    ), // TextStyle
), // Text
], // <Widget> []
```



Lastly: Let's Wrap our Entire Column in a Padding

```
body : SafeArea (
    child : Padding (
        padding : const EdgeInsets.fromLTRB(0, 120.0, 0, 0),
        child : Column (
            children : <Widget>[...], // <Widget>[]
        ), // Column
    ), // Padding
), // SafeArea
```



Conclussion

Thanks to our intl package we were able to get a good time format from our previously crudd time

NEXT: We will be fixing up our Landing Screen

SPINNERS #32

At the minute we have a loading screen that loads the data then redirects us to the home screen with that Data

But at the minute our loading screen looks bad. So Instead of us thinking that we want to use a Spinner

To Do This:

We could make our own widget, but that would be quite complex so instead let's use a package called "flutter_spinkit".

Go over to pub and install this - using the guide in Lecture #27

flutter_spinkit 4.0.0

published Aug 17, 2019 [FLUTTER]

----> You can check out the doc here

Readme Changelog Example Installing Versions (100)

...

dependencies:

flutter_spinkit: ^4.0.0

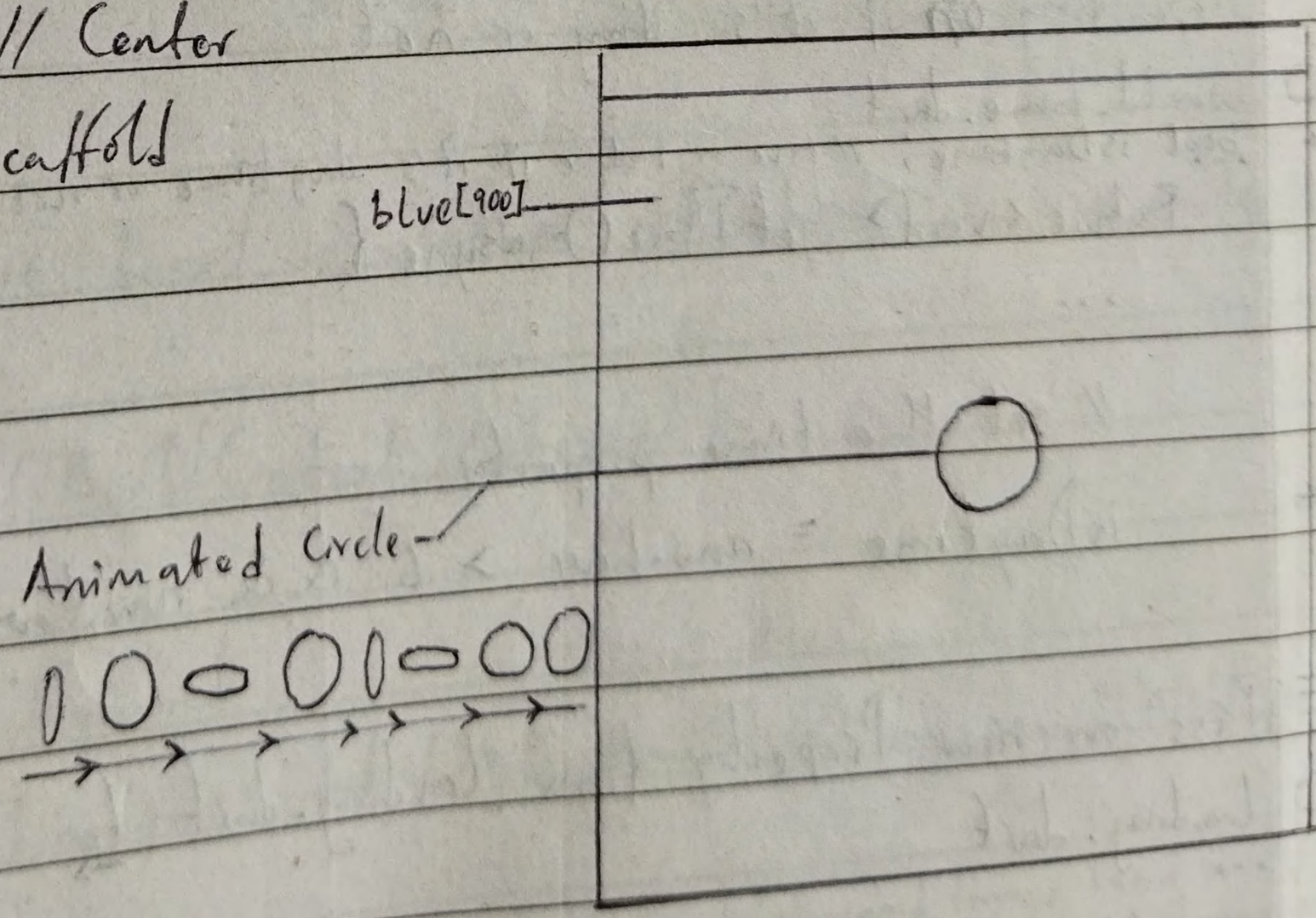
---> Copy this and paste in our pubspec.yaml file to install this package

Guide on how to use this Package

≡ Add A Spinner to our Loading Screen

O loading.dart

```
+ import 'package:flutter_spinkit/flutter_spinkit.dart';
... @override
    Widget build (BuildContext context) {
      return Scaffold (
        backgroundColor: Colors.blue[900],
        body: Center (
          child: SpinKitRotatingCircle (
            color: Colors.white,
            size: 50.0,
          ), // SpinKitRotatingCircle
        ), // Center
      ); // Scaffold
    }
```



Assignment

Go ahead and try out other SpinKits in Readme. E.g SpinKitFoldingCube, SpinKitRing, SpinKitRipple, SpinKitCircle and somany more.

Ternary Operators & Big Images #33

Alright Guys, So far on our home we've been able to show the Location, Time and Edit Location Information.

Next we want to show a background Image either night or day depending on what time it is

TO DO THIS:

We will do this in the world_time.dart, in our class for more abstraction

- Create boolean ~~variable~~ property in WorldTime class to hold whether it's day or night
- Check time in getTime() function and set our boolean ~~variable~~ property

Add Boolean Property with logic that sets it to true or False, depending ~~on~~ if it's day or not

world_time.dart

```
+ bool isDaytime; // true or false if it's daytime or not
  Future<void> getTime() async {
    ...

    // set the time property
+   isDaytime = now.hour > 6 && now.hour < 20 ? true : false;
```

Pass our new Property from loading.dart to home.dart

loading.dart

```
...
  Navigator.pushReplacementNamed(context, 'home', arguments:{
    ...
+   'isDaytime': instance.isDaytime,
  });
```

we can now use this to display either a day or night time image

Now You Can Find The Images we are going to be using In my Github

<> Code.

Branch: lesson-330 flutter-beginners-tutorial/world_time_app/

:SELECT:

...

assets

:click:

day.png

night.png

Download both and move them into our projects ▽ assets // paste both and then tell pubspec.yaml where to find those Images

pubspec.yaml

...

assets:

- assets/

☑ Now we Can access our Images in our project

Pick An Image based on whether is Daytime is true or false

home.dart

...

Widget build(BuildContext context) {

...

// set background

String bgImage = data['isDaytime'] ? 'day.png' : 'night.png';

--> Now this will hold our background Image based on if it is night or day

return Scaffold(...);

}

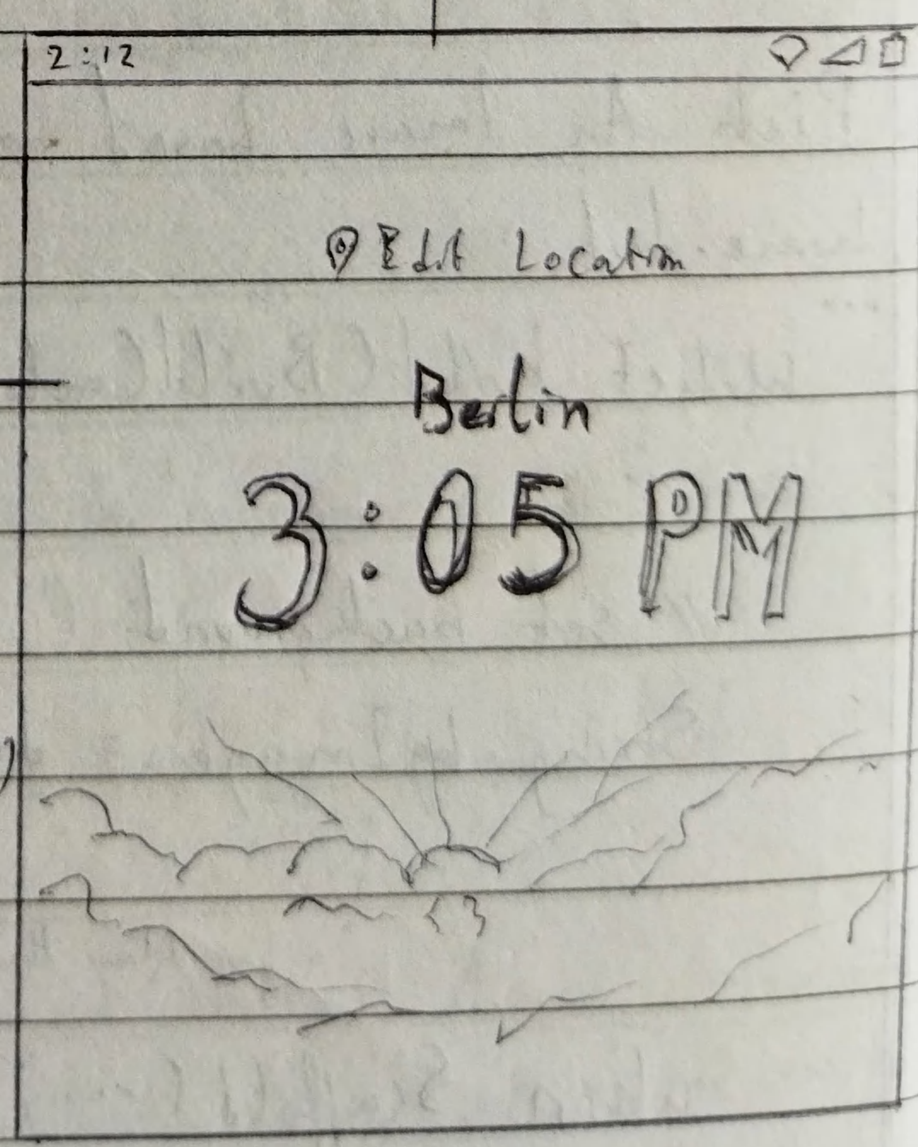
Next: We want to output the bgImage on the background. To do this we use a Combination of a

- Container and a;
- BoxDecoration

```
...
return Scaffold(
  body: SafeArea(
    child: Container(                    [Add wrapper]
      decoration: BoxDecoration(
        image: DecorationImage(
          image: AssetImage('assets/$bgImage'),
          fit: BoxFit.cover,             [Covers screen]
        ) // DecorationImage
      ), // BoxDecoration
      child: Padding(...)
    ), // Container
```

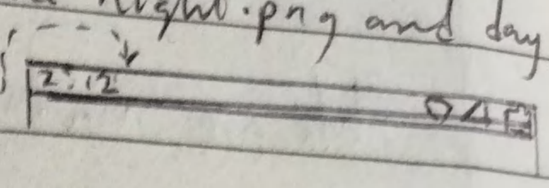
No color here
Let's Fix this

day.png



Assignment
Tweak the world_time.dart isDaytime property to see that day.png and night.png WORKS

Add Scaffold Background Color that Compliments the night.png and day .png Images – It will only display in the top bar.



```
...
  // set background
  String bgImage = ...;
  Color bgColor = data['isDaytime'] ? Colors.blue : Colors.indigo

  return Scaffold(
    backgroundColor: bgColor,
    body: SafeArea(...)
  )
```

Assignment
Find your own Complimentary colors

LASTLY: We want to change the Colors of our texts depending on the background

```
...
  FloatButton.icon(
    onPressed: () {...},
    icon: Icon(
      Icons.edit_location,
      color: Colors.grey[300],
    ), // Icon
    label: Text(
      'Edit Location',
      style: TextStyle(
        color: Colors.grey[300],
      ), // TextStyle
    ), // label
```

Edit Location

Edit Location

Assignment

Berlin
3:14 Pm

Color both of them White

ListView Builder #34

We are making some progress with this App, But there's still a couple of things left to do.

What we want to do next is in our location screen we want to have multiple locations listed on and whenever we click on any of those - It's going to update the location by creating a new World Time class for that location, getting the time data and returning it back to the Home Screen to show that updated data

In this section we will be focusing on the TEMPLATE of the Page, we will be using the ListView for this.

Create 8 Locations : That are objects of the WorldTime class

choose-location.dart

```
class _ChooseLocationState extends State<ChooseLocation> {

  List<WorldTime> locations = [
    WorldTime(url: 'Europe/London', location: 'London', flag: 'uk.png'),
    "  (url: 'Europe/Berlin', location: 'Athens', flag: 'greece.png'),
    "  (url: 'Africa/Cairo', location: 'Cairo', flag: 'egypt.png'),
    "  (url: 'Africa/Nairobi', location: 'Nairobi', flag: 'kenya.png'),
    "  (url: 'America/Chicago', location: 'Chicago', flag: 'usa.png'),
    "  (url: 'Asia/Seoul', location: 'seoul', flag: 'south_korea'),
    "  (url: 'Asia/Jakarta', location: 'Jakarta', flag: 'indonesia'),
    "  (url: 'America/New_York', location: 'New York', flag: 'usa.png'),
  ];
```

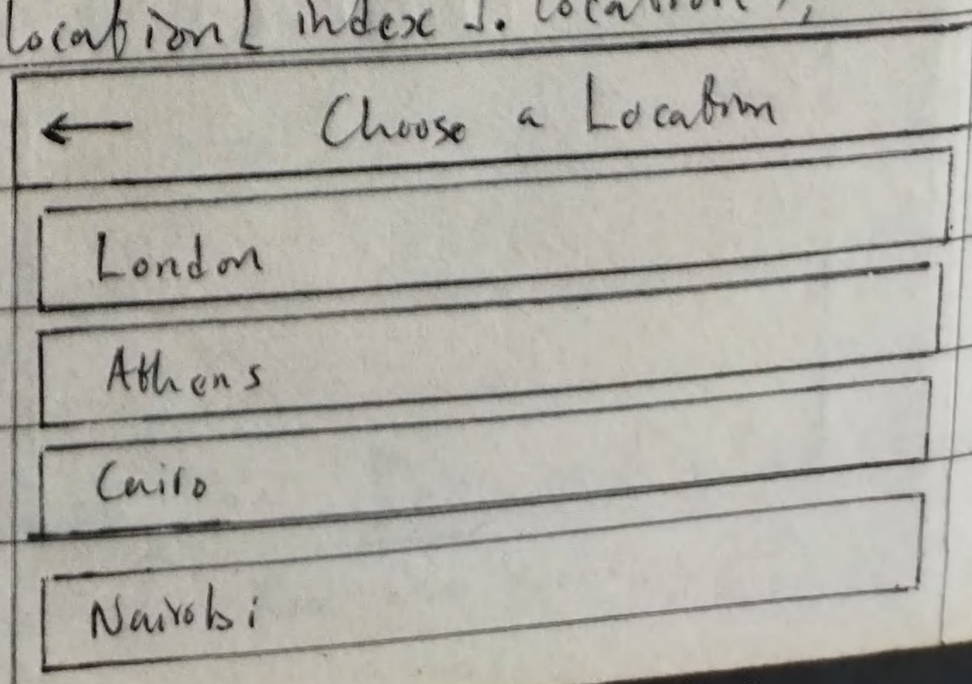
Cycle through the locations List and output a template for each

- For this we will be using the List View Builder - which allows us to use an inline anonymous function to return a widget template for each item in our list

PREVIOUSLY: If you can remember, earlier we used a map method to output a template for items in the list

That was one way of doing it, and the ListView Builder is another

```
...
return Scaffold (
  backgroundColor : Colors.gray[200],
  appBar : AppBar (...), //AppBar
  body : ListView.builder (
    itemCount: locations.length,
    itemBuilder : (context, index) {
      return Card (                    ; ---> checkout docs
        child : ~~ListTile~~ ListTile (
          onTap : () {},               [Similar to onPressed]
          title : Text ( location[ index ]. location ),
        ), // List Tile
```



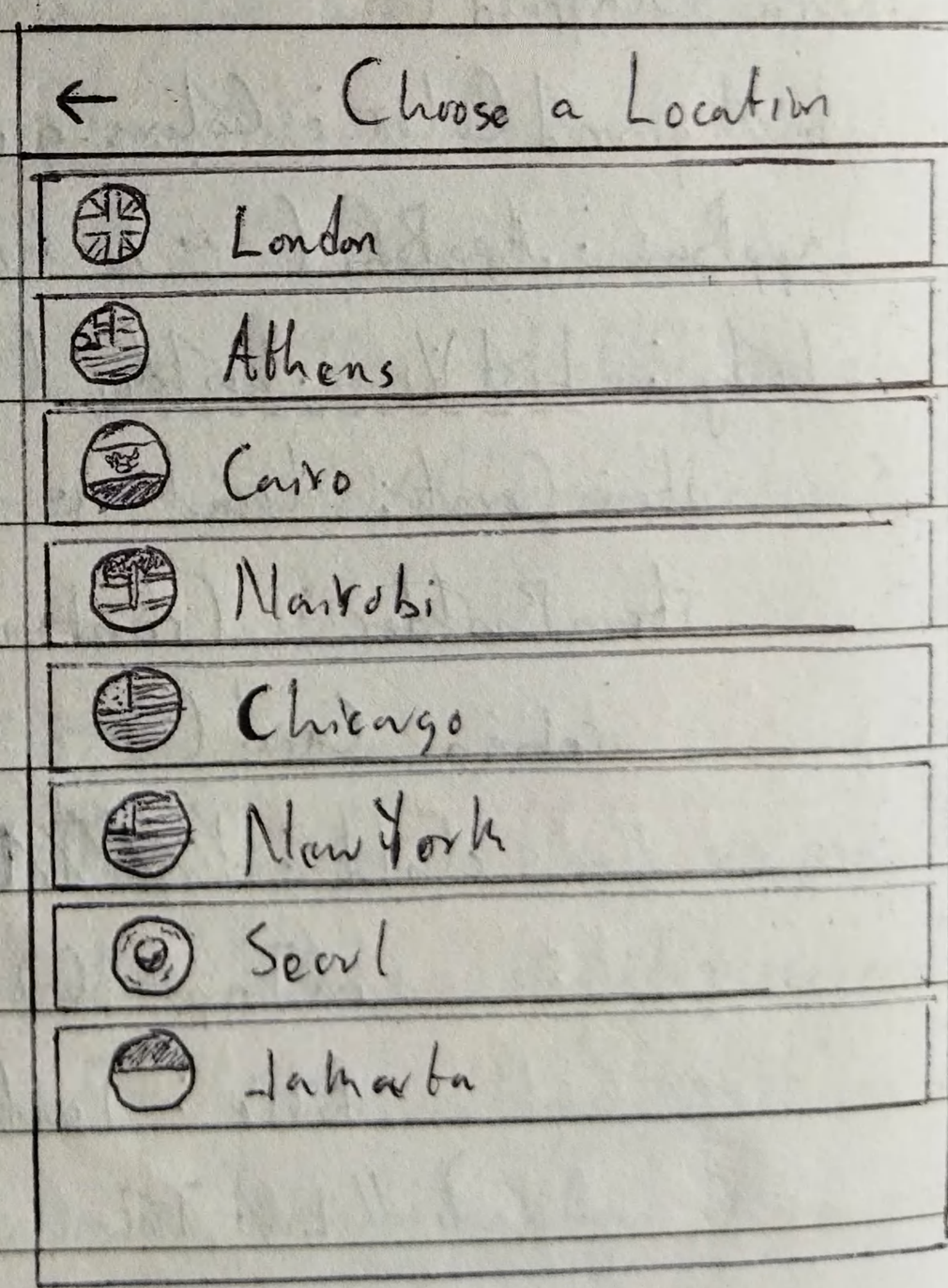
- Grab all the Flags Image from the Github repo of this Course
  - Paste them Inside our project's **assets** folder

- Output The Flag of each location beside it

```
... return Card(
      child: ListTile(
        ...
        leading: CircleAvatar(                    ${locations[index].flag}
          backgroundImage: AssetImage('assets/ ^ '),
        ), // CircleAvatar
      ), // ListTile
    ); // Card
```

Assignment

Add Symetrical Padding to card



UPDATING THE LOCATION #35

LAST LECTURE

Welcome to the showdown. So there is only 1 more piece to the puzzle when it comes to building this app

Which is to update the Location data of any one of our listed Locations

- Create Boilerplate for Updating ~~the~~ any Location
- choose_location.dart

```
class _ChooseLocationState extends State<ChooseLocation> {
  List<WorldTime> locations = [...];


  void updateTime(index) async {                    Actual Function
    // Update Location Time and Navigate to Home Screen
  }
  ...

  return Card(
    child: ListTile(
      onTap: () {
        updateTime(index);                          function Call
      }
    ), // List Title
  );
```

NOTE:

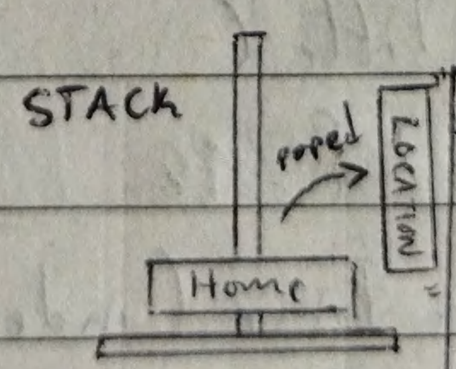
- The Navigator method we will be using is **Pop** and what it does is what it says. It pops current screen from ~~the~~ Our View

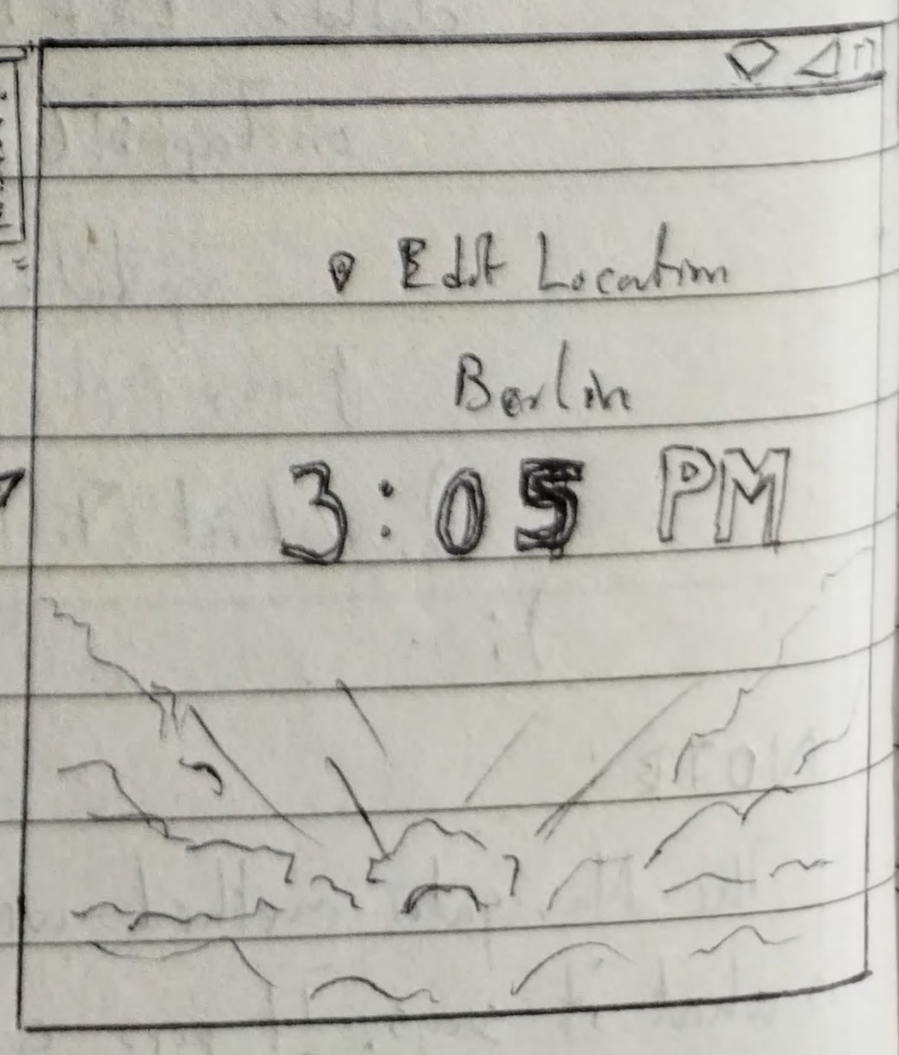
- pop only takes (context, { })
  ↳ it knows that this represents arguments

≡ Implement Our UpdateTime function

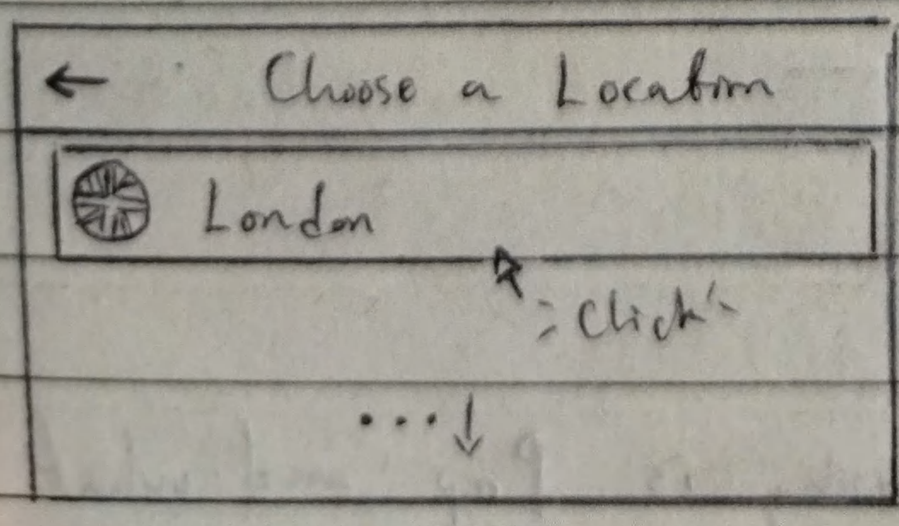
...

```
void updateTime(index) async {
+   WorldTime instance = locations[index];
+   await instance.getTime();


+   // navigate to home screen
+   Navigator.pop(context, {
        'location': instance.location,
        'flag': instance.flag,
        'time': instance.time,
        'isDaytime': instance.isDaytime,
    });
}
```





▽ There Is a PROBLEM



pops but doesn't

Rebuild the below SCREEN

This is because we are not rebuilding the widget nor are we using setState(( ) { }) function to trigger a re-render with setting the data variable to our our newly passed In data.

To Fix This:

Where we use Navigator.pushNamed('/location') on to the stack we want to right there <u>await</u> a result from '/location'
- Blocking until we get a result (our new data), and then use setSta-te to set new data which should then trigger a re-render.

home.dart

```
class _HomeState extends State<Home> {

  Map data = {};

  @override
  Widget build(BuildContext context) {

    data = ModalRoute.of(context).settings.arguments;

    ...

    FlatButton.icon(
      onPressed: () async {
        dynamic result = await Navigator.pushNamed(...);
        setState(() {
          data = {
            'time': result['time'],
            'location': result['location'],
```

```
        'isDaytime': result['isDaytime'],
        'flag': result['flag'],
      };
    });
  }
```

With this we override the old Map with the new Map but th

while rebuilding, we run into

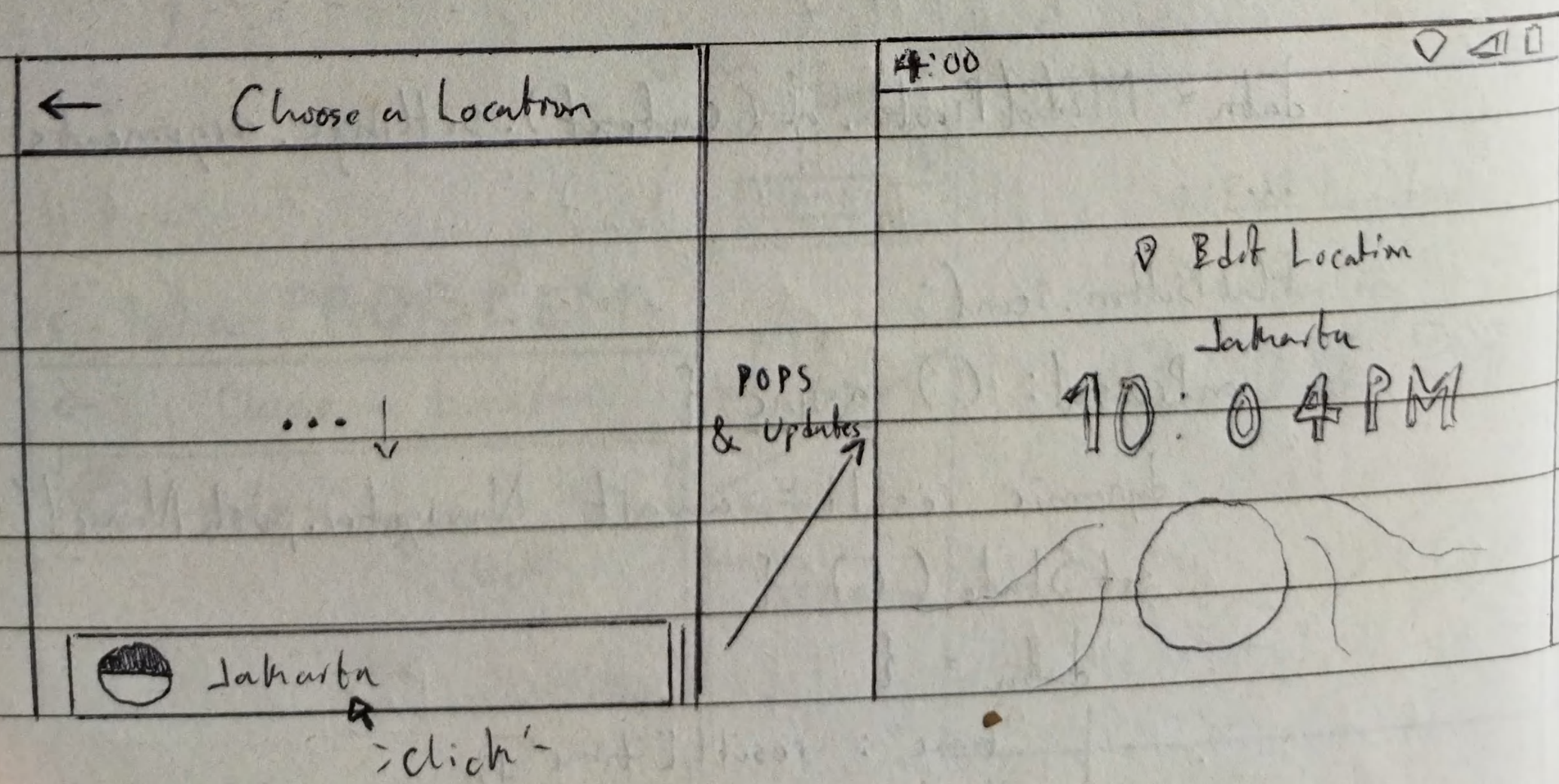
```
@override
Widget build(Build ...)
  data = ModalRoute.of(Context).settings.arguments;
```

Which overwrites the data again with the old data

To Fix This :

using Ternary Operato

```
+ data = data.isNotEmpty ? data : ModalRoute.of(context).settings.arguments
```



CONCLUSION

LAST PAGE

That is all for now, I hope you enjoyed this course. There is soo much more to flutter than what we've learnt.

So please also take time to read the documentation on the Flutter website for more ideas

flutter.dev/docs/cookbook

Cookbook

DOCS > Cookbook

This cookbook contains reciepes that demonstrate how to solve common problems while writing Flutter apps. Each reciepe is self-contained and can be used as a reference to help you build an application.

Animation

...↓

--> Also check out this Cookbook as it is pure Gold if you are looking to improve your flutter skills

NETNINJA - Thankyou soo much for watching. See you Later. Goodluck! 👍 😁