Flutter

1. Setting up
   1. install Flutter [here](https://docs.flutter.dev/get-started/install)
   2. in CLI test flutter by  
      flutter --version  
      flutter doctor
2. create a new project
   1. In CLI type in:  
      flutter create [project name]
3. Project structure
   1. an application in Flutter is always a bunch of nested widgets (sort of like Russian dolls)

Widget – each widget is a Class

1. general idea
   1. each widget is a class and to create it, just call the constructor
   2. to add item into the class/widget, specify the item that is to be added as arguments in the constructor
   3. some useful arguments
      1. child 🡪 take another (single) widget
      2. children 🡪 take a list of widgets
2. Stateless widget
   1. Parent class for widgets
   2. Only have one state
   3. Used to create a custom widget by extending this class
   4. Ex:  
      class MyScrapElement extends StatelessWidget {

const MyScrapElement({super.key});

@override

Widget build(BuildContext context) {

return const Text('this widgets is a text');

}

}

1. Stateful widget
   1. Create widget with multiple states, such as input box
   2. Vscode shorthand: stful
   3. All the override below must be written to make stateful widget
   4. To update a state of a stateful widget (redraw the whole widget), setState(void func()) must be called
   5. Ex:  
      class MyWidget extends StatefulWidget {

const MyWidget({super.key});

@override

State<MyWidget> createState() => \_MyWidgetState();

}

class \_MyWidgetState extends State<MyWidget> {

@override

Widget build(BuildContext context) {

return const Placeholder();

}

}

1. Container()
   1. basically, a box container
2. Text()
   1. a text widget
3. MaterialApp()
   1. has home, doesn't have child/children
4. Material()
   1. is an opensource widget
   2. all children of material will have component of Material
5. Column()
   1. give a column widget (vertical array of widgets)
6. TextField()
   1. Is stateful widget.
   2. To control the widget use controller parameter that takes of type TextEditingController
   3. To redraw/change the state of TextField, use onChanged parameter, that takes a void Function(String) as a parameter. Remember to use setState(void Function()) to change a state.
   4. Ex:  
      class TextInputWidget extends StatefulWidget {

const TextInputWidget({super.key});

@override

State<TextInputWidget> createState() => \_TextInputWidgetState();

}

class \_TextInputWidgetState extends State<TextInputWidget> {

final TextEditingController myController = TextEditingController();

String text = 'as';

@override

void dispose() {

super.dispose();

myController.dispose();

}

void changeStateText(String msg) {

setState(() => this.text = msg); //setState will change the state of the widget

if (this.text == 'Hello World'){

setState(() => this.text = '');

}

}

void Function(String) changeStateText2(){

return (msg) => changeStateText(msg);

}//alternative

@override

Widget build(BuildContext context) {

return Column(

children: [

TextField(

controller: myController,

decoration: const InputDecoration(

prefixIcon: Icon(Icons.message),

labelText: 'Put something here',

),

onChanged: changeStateText, //this will call the function with the text from the textbox as the function argument

),

),

Text(this.text),

],

);

}

}