Mobile App Dev

📜 Table of Contents

Installation

1. Go to https://docs.flutter.dev/get-started/install/
2. Choose your operating system.
3. Choose the platform and follow procedure.
4. move the zip file to a folder in c : drive.
5. Extract the zip file.
6. Update windows path variable.
7. Run **flutter doctor** on the cmd.

Creating a project

command: flutter create [project name]

project name cannot use upper case letters

running the project:

command: flutter run

Open the folder in vs code.

Keywords

1. **Integer** - any positive or negative non-decimal number
2. **Double** - decimal numbers
3. **void** - no return value when used as func return type

4. **dynamic** - takes any datatype
5. **final** - any variable with final cannot be modified and is called at run time
6. **const** - any variable with const cannot be modified and is called at compile time

---

Tasks

Day 1

- installation
- project creation
- datatypes - string operations
- run the app on chrome and emulator

Day 2

- Align the column children to main axis center and cross axis center and other orientations
- Create a row widget and load children horizontally
- Draw login screen pages from google and label the widget types

Day 3

- Login Page design
- Generate the apk file

Day 4

- Create navigation buttons between 2 screens
- Load the states of India as a List View
- Show alerts on button clicks for validation

Dart Programming Language

File extension for dart files is **.dart**

dart language is used to create applications using the flutter framework.

A dart file should have a **void main(){}** function that is the entry point of the program.

To print on the console, we use the **print()** function.

Every line of code or statement should end in a semicolon (;)

```
void main(){
    print("Hello World");
}
```

The above code prints “Hello World” on the console.

Datatypes

There are 7 datatypes in dart, they are:

1. Numbers

In dart, numbers are used to represent numeric literals.

2. Strings

It is used to represent a sequence of characters. It is a sequence of UTF-16 code units. The keyword string is used to represent string literals. String is a collection of characters enclosed in single, double or triple quotes.

String str1 = “ ABC|abc ”;

string in-built methods:

toUpperCase() → str1.toUpperCase() → “ ABC|ABC ”

toLowerCase() → str1.toLowerCase() → “ abc|abc ”

trim() → str1.trim() → “ABC|abc”

trimLeft() → str1.trimLeft() → “ABC|abc ”

trimRight() → str1.trimRight() → “ ABC|abc”

split() → str1.split(”|”) → [” ABC”, “abc ”]

3. Booleans

True or False values

bool isTrue = true;

bool isFalse = false;

4. Lists

List is a collection of objects separated by commas and enclosed in square brackets.

[1, 23, 69, 7, 10] is a list of numbers or integers.

Every element is positioned with an index number. Indices start with 0 for the first element.

syntax:

List<DataType> ListName = [element1, element2, element3, element4];

if you give the datatype as **dynamic**, it can take any datatype and be a heterogenous list.

functions:

```
void main(){
    // declaration of studentNames list
    List<String> studentNames = [”safwan”];

    studentNames.add("mohammed");    // adds an element to the l
```

```
    studentNames.addAll(["jswanth", "chakradhar"]);   // adds mu
    studentNames.insert(2, "deva");    // adds "deva" to index 2
    studentNames.removeAt(2);    // removes element from index 2
    studentNames.remove("mohammed");    // removes "mohammed" el

    // printing using index
    print(studentNames[0]);
    print(studentNames[3]);
    print(studentNames.first);
    print(studentNames.last);

    studentNames.clear(); // removes all elements from the list
}
```

5. Maps

The map object is a key value pair. Keys and values on a map may be of any type. It is a dynamic collection. The key always has to be String datatype and unique. Maps are defined with curly braces.

syntax: Map<keyDataType, valueDataType> MapName = { };

Map <String, String> students = {};

```
void main(){
    Map Newmap = new Map();
    Newmap['First'] = 'Dart';
    Newmap['Second'] = 'For';
    Newmap['Third'] = 'Developing apps';

    Map<String, dynamic> students = {
        "Name" : "Mohammad",
        "Age" : 18,
        "Branch" : "CSE"
    };
```

```
    print(Newmap);
    // output: {'First': 'Dart', 'Second': 'For', 'Third': 'Deve

    print(students);
    // output: {"Name" : "Mohammad", "Age" : 18, "Branch" : "CSE
}
```

6. Sets

7. Runes

Variables

variable declaration syntax: [**typeOfVariable] [nameOfVariable] = value;**

examples :

int num1 = 2;

double num2 = 1.5;

bool isRemember = true;

When a variable name starts with an underscore (_variableName), it is set to private access specifier. It cannot be accessed directly outside the local scope. It will need getter and setter methods to manipulate it.

String concatenation: to add a variable inside a string we use the dollar ($) symbol

to add more than a variable, the dollar is accompanied with curly braces { }

```
void main(){
    int x = 5;
    print("The value of x is $x");
    print("The upper case of hello is ${'hello'.toUpperCase()}");
}
```

Functions

A set of statements that perform a specific operation or task. Can be reused by calling the function.

Syntax :

```
// basic syntax
returnType functionName(parameters){
    // body of the function
    return value;
}
```

Examples :

```
// function to print on the terminal
void printValue(){
    print("Hello World");
}
// This function prints "Hello World" everytime it is called
```

```
// function to add two inputed numbers
int addTwoNumbers(int a, int b){
    return a+b;
}
// This function takes in a and b integers as input parameters a
```

Calling the function can be seen below

```
int sum = addTwoNumbers(2, 3);
print(sum);
```

We store the return output in the integer variable sum and print it in the nextline.

Flutter

Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications from a single codebase for any web browser, Fuchsia, Android, iOS, Linux, macOS, and Windows.

Every element in Flutter is considered as a **Widget**. They are classified into two types: **state-ful** and **state-less.** Stateful, meaning that it has a State object (defined below) that contains fields that affect how it looks.

Each screen has a name and it has to be defined with a route. The routes are defined in the main.dart file

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      routes : {
                "/login":(context) => const LoginScreen()
           },
           initialRoute: "/login",
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.gree
        useMaterial3: true,
```

```
      ),
      // home: const MyHomePage(title: 'Muqabil'),
    );
  }
}
```

To create a new screen, create a new dart file and use the **flutter widget snippet** extension and type the prewritten stateful widget snippet by typing fs. Then add the route to this new page in the home page after importing it. The **initialRoute** key allows you to specify the page that opens first. Or you can just put “/” in the route name key to make it the initially loaded screen.

Container

The container in Flutter is a parent widget that can contain multiple child widgets and manage them efficiently. It has attributes like:

- Color
- Decoration
- Background color
- Padding
- Margin
- Height and width

Padding is the inner spacing in a container and margin is the exterior spacing in a container. Padding is the space between the border and child elements whereas margin is the space between the border and other neighbouring containers.

The syntax is Container() and the properties are enclosed in paranthesis separated with commas. The properties are key: value pairs written with colons.

```
body: Container(
    color: Colors.amber,
    height: 500,
    width: 300,
```

```
    margin: EdgeInsets.all(50),

    // child property
    child: Text("this is a child container"),
)
```

Text

Text in Flutter is a widget that can display any written text and modify its size, font weight, font style, color etc.

```
Text(
    "Hello World",
    style: TextStyle(
        fontSize: 35,
        fontWeight: FontWeight.w600,
        color: Colors.black,
    )
)
```

Input text field

Input text field in Flutter is used to take user input in the form of written text.

Properties of text form field:

1. **controller** : to access the data in the field
2. **decoration** : styling the looks of the field
3. **obscureText** : censors the text eg: passwords
4. **validator** : defines the condition for validation

syntax:

```
TextFormField(
    controller: _passController,
    decoration: InputDecoration(
      hintText: "hello@gmail.com",
    border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(8)
      )
    ),
    obscureText: true
)
```

button

Buttons in Flutter are used to take user input in the form of clicks.

```
TextButton(
    onPressed: () {
        // functionality of the button
    },
    child: Text("Click me")
)
```

row

Column widgets in flutter are used for horizontal allignment of its children. It can take multiple widgets and display them horizontally. The main axis in row is X axis and the cross axis is Y axis.

```
Row(
    children:[
        Container(
            color: Colors.blue,
            width: 300,
            height: 200
```

```
        ),
        Container(
            color: Colors.orange,
            width: 300,
            height: 200
        ),
        Container(
            color: Colors.black,
            width: 300,
            height: 200
        )
    ]
)
```

column

Column widgets in flutter are used for vertical allignment of its children. It can take multiple widgets and display them vertically. The main axis in column is Y axis and the cross axis is X axis.

There are nine points defined in a column layout. Defined with respect to Main axis and Cross axis as start, centre or end. By default a column is at Main axis start and Cross axis centre. We can change these values.

Main axis and Cross axis also has properties called “spaceAround”, “spaceBetween”, “spaceEvenly”

spaceAround - adds equal spaces between the children and also the borders

spaceEvenly - adds equal spaces between the children containers

spaceBetween - adds equal spaces between the children but no space from the borders

```
Column(
    mainAxisAlignment: MainAxisAlignment.center,
    crossAxisAlignment: CrossAxisAlignment.center,
    children:[
        Container(
            color: Colors.blue,
```

```
            width: 300,
            height: 200
        ),
        Container(
            color: Colors.orange,
            width: 300,
            height: 200
        ),
        Container(
            color: Colors.black,
            width: 300,
            height: 200
        )
    ]
)
```

List view

List view is to align the items or widgets vertically or horizontally on the screen. We can give the item count. List view allows for dynamic scrolling with items being retrieved from the server.

Example of list view with a users list page:

1. Create a new screen called *usersListView.dart* and add the route to main.dart
2. Create a list called *users* in the class of *usersListView.dart* file

```
List<String> users = [
    "Ramesh",
    "Anil",
    "Sai",
    "Chintu",
    "Manasa",
    "Deva",
    "Samantha",
    "Vishnu",
```

```
    "Ranjith",
    "Charan",
    "Patel",
    "Hemanth",
    "Balayya"
];
```

3. Create the ListView widget in the page scaffold

```
@override
  Widget build(BuildContext context) {
    return Scaffold(
            appBar: AppBar(title: Text("User List View")),
            body: Container(
                width: MediaQuery.of(context).size.width,
                height: MediaQuery.of(context).size.height,
                child: ListView.builder(
                    itemCount: users.length,
                    itemBuilder: (context, index) {
                        final name = users[index];
                        return Card(
                            child: Text(name),
                        );
                    },
                ),
            )
        )
    }
```

Login Page design

Every screen returns a Scaffolding, Scaffolding can have appBar (Top bar), Body and other keys.

Here, we add a container inside the body and define its properties of color, width and height. The color is green, height is 200 and width is taken dynamically from the MediaQuery.

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

class LoginScreen extends StatelessWidget {
  const LoginScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Login Screen")),
            body: Container(
                color: Color.fromARGB(255, 255, 235, 164),
                width: MediaQuery.of(context).size.width,
        height: MediaQuery.of(context).size.height,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceAround,
          crossAxisAlignment: CrossAxisAlignment.center,
          children:[
            Container(
              color: Colors.blue,
              width: 300,
              height: 100
            ),
            Container(
              color: Colors.orange,
              width: 300,
              height: 100
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
```

```
              crossAxisAlignment: CrossAxisAlignment.center,
              children: [
                Container(
                  color: Colors.black,
                  width: 120,
                  height: 100
                ),
                Container(
                  color: Colors.brown,
                  width: 120,
                  height: 100
                ),
              ],
            )
          ]
        )
            ),
    );
  }
}
```

Creating the Login Page

We are going to see how to design a login page with 11 children widgets in a column layout and also validate the input of email and password. We need the following components on the page: useer profile image, heading text, paragraph text, email address text input, password text input, forgot password button, login button, “create an account” link and “book a demo” page.

We first take a Scaffold widget, and in the body we have a container containing a column and use it’s children property to load the 11 children components.

Loading the profile image

1. select an image for the profile and load it
2. open the project file in vs code
3. create a new folder called ***assets***
4. drag and drop the image into the ***assets*** folder
5. open the ***pubspec.yaml*** file
6. uncomment the assets section in the file
7. write the path of your image in the assets section
8. In the code, Image can be loaded with **Image.asset(”assets/logo.png”)** or **Image.network(”*path*”)** for loading dynamic images from the server.

Login Page code

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

// ignore: must_be_immutable
class LoginScreen extends StatelessWidget {
  LoginScreen({super.key});

    bool isVisibilityOff = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
            body: Container(
            width: MediaQuery.of(context).size.width,
            height: MediaQuery.of(context).size.height,
            margin: EdgeInsets.all(10),
      child: Column(
        children: [
          // Profile Image
          Column(
```

```
              mainAxisAlignment: MainAxisAlignment.start,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Image.asset("assets/logo.png", height: 100, width
                const SizedBox(height: 12,),

                // Welcome Back text
                const Text("Welcome Back", style: TextStyle(
                  fontSize: 32,
                  fontWeight: FontWeight.w600
                ),),
                const SizedBox(height: 12,),

                // Paragraph text
                const Text("Welcome Back. Enter your credentials t
                    style: TextStyle(
                      fontSize: 14,
                      fontWeight: FontWeight.w400,
                      color: Color.fromRGBO(138, 144, 162, 1)
                  )),
                const SizedBox( height: 24 ),

                // Email Input
                const Text("Email Address", style: TextStyle(
                  fontSize: 14,
                    fontWeight: FontWeight.w400,
                  color: Colors.black
                )),
                const SizedBox( height: 8 ),

                TextFormField(
                  decoration: InputDecoration(
                      hintText: "hello@gmail.com",
                      border: OutlineInputBorder(
                          borderRadius: BorderRadius.circular(8)
                )))),
```

```
            const SizedBox( height: 20 ),

            // Password Input
            const Text("Password", style: TextStyle(
                fontSize: 14,
              fontWeight: FontWeight.w400,
              color: Colors.black
            )),
            const SizedBox( height: 8 ),

            TextFormField(
              obscureText: isVisibilityOff,
              decoration: InputDecoration(
                  hintText: "password",
                suffixIcon: InkWell(
                  onTap: (){
                    if (isVisibilityOff){ isVisibilityOff=fals
                    else{ isVisibilityOff = true; }
                  },
                  child: isVisibilityOff ?
                  const Icon(Icons.visibility_off) :
                  const Icon(Icons.visibility),
                ),
                  border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(8)
            ))),
            const SizedBox( height: 20 ),
          ],
        ),

        Row(
          mainAxisAlignment: MainAxisAlignment.end,
          children: [
            TextButton(onPressed: (){}, child: Text("Forgot Pa
          ],
        ),
```

```
          const SizedBox(height: 39),

          TextButton(
            onPressed: (){},
            child: Container(
              decoration: BoxDecoration(
                color: const Color.fromRGBO(89, 86, 233, 1),
                borderRadius: BorderRadius.circular(100)
              ),
              width: 280,
              height: 50,
              alignment: Alignment.center,
              child: const Text("Login", style: TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.w600,
                color: Colors.white
              ),)
            ),
          ),
          const SizedBox( height:  24),

                    // Create an account
                    Row(
              mainAxisAlignment: MainAxisAlignment.center,
            children: [
                        const Text("New Here? "),
                        TextButton(onPressed: () {},
                        child: Text("Create an account")
                    )
                    ])
            ]
          )
            )
      );
```

```
  }
}
```

Navigation

Navigation between screens is done using the routes written in main.dart, The route names are used to switch between screens.

In Android, a route is equivalent to an Activity. In iOS, a route is equivalent to a ViewController. In Flutter, a route is just a widget.

This recipe uses the `Navigator` to navigate to a new route.

The next few sections show how to navigate between two routes, using these steps:

1. Create two routes.
2. Navigate to the second route using Navigator.push().
3. Return to the first route using Navigator.pop().

1. Create two routes

First, create two routes to work with. Since this is a basic example, each route contains only a single button. Tapping the button on the first route navigates to the second route. Tapping the button on the second route returns to the first route.

First, set up the visual structure:

```
class FirstRoute extends StatelessWidget {
  const FirstRoute({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ),
      body: Container(
        child: TextButton(
          child: const Text('Open route'),
```

```
          onPressed: () {
            // Navigate to second route when tapped.
          },
        ),
      ),
    );
  }
}

class SecondRoute extends StatelessWidget {
  const SecondRoute({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Route'),
      ),
      body: Container(
        child: TextButton(
          onPressed: () {
            // Navigate back to first route when tapped.
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```

2. Navigate to the second route using Navigator.pushNamed()

To switch to a new route, use the `Navigator.pushNamed()` method. The `push()` method adds a `Route` to the stack of routes managed by the `Navigator`.

In the `build()` method of the `FirstRoute` widget, update the `onPressed()` callback:

```
// Within the `FirstRoute` widget
onPressed: () {
  Navigator.pushNamed(
    context, "/SecondRoute"),
  );
}
```

3. Return to the first route using Navigator.pop()

How do you close the second route and return to the first? By using the `Navigator.pop()` method. The `pop()` method removes the current `Route` from the stack of routes managed by the `Navigator`.

To implement a return to the original route, update the `onPressed()` callback in the `SecondRoute` widget:

```
// Within the SecondRoute widget
onPressed: () {
  Navigator.pop(context);
}
```

Responsive Design

If the widgets overflow from the screen, we can wrap the content in a widget called **SingleChildScrollView.** This allows the screen to be scrollable and the widgets will be below the visible viewport.

```
body: Container(
    width: MediaQuery.of(context).size.width,
    height: MediaQuery.of(context).size.height,
    child: SingleChildScrollView(
        child: Column(
            ...
        )
```

```
    )
)
```

The above code shows a column child in the **SingleChildScrollView** widget.

Login Form Validation

1. Create a global form key
2. Wrap the column with the widget **Form**
3. Assign the key attribute to the Form
4. Create textEditingController for each text field
5. Assign the controller to the attributes in each text field
6. Add the validator property to each text field

```
// top of the class
final _formKey = Global...;
textEditingController _emailController = TextEditingController(
textEditingController _passwordController = TextEditingControlle

// in the textForm
TextFormField(
    validator: (value){
        ...
    },
    controller: _passwordController,
    obscureText: true,
)

// in the login button
TextButton(
    onPressed: (){
        showValidationAlert();
    },
```

```
  child: Container(
    decoration: BoxDecoration(
      color: const Color.fromRGBO(89, 86, 233, 1),
    borderRadius: BorderRadius.circular(100)
  ),
  width: 280,
  height: 50,
  alignment: Alignment.center,
  child: const Text("Login", style: TextStyle(
        fontSize: 16,
    fontWeight: FontWeight.w600,
    color: Colors.white
    ),)
    ),
)

// showValidationAlert function at the bottom of the class
void showValidationAlert() {
    if (_emailController.text.isEmpty){
        showDialog(
            context: context,
            builder: (context) {
                return AlertDialog(
                    title: Text("Alert"),
                    content: Text("Please Enter Email"),
                    actions: [
                        TextButton( onPressed: () {
                            Navigator.pop(context);
                        }, child: Text("ok")
                        )
                    ]
                )
            }
        )
```

```
    }
}
```