

MU

SEM
3

S. Y. B.Sc.
Information Technology



(Course Code : USIT3P5) (Core Subject Practical)

MOBILE PROGRAMMING PRACTICAL



Prof. Krutika H. Churi

Sonopant Dandekar Arts, V. S. Apte Commerce
and M. H. Mehata Science College, Palghar, Mumbai

ISBN

978-93-5583-436-2

Price ₹ 125/-

(Book Code : BI-06A)



TECH-NEO
PUBLICATIONS

Where Authors Inspire Innovation

A Sachin Shah Venture

www.techneobooks.in

info@techneobooks.in

Mobile Programming Practical

(Course Code : USIT3P5)

[♦ Core Core Subject Practical ♦]

S. Y. B.Sc. (Information Technology)

Semester III - Mumbai University

Choice Based Credit and Semester System with
effect from the Academic Year 2023 – 2024

Prof. Krutika H. Churi

M.Sc. IT

PG Professor,

Assistant Professor

Sonopant Dandekar Arts, V. S. Apte Commerce,
and M. H. Mehata Science College,
Palghar, Mumbai



BI-06



Mobile Programming Practical

(Course Code : USIT3P5)

[Core Core Subject Practical] (Code : USIT303)

S.Y. B.Sc. (Information Technology)

Semester III (Mumbai University)

- For New Syll. 2023-2024)

▶ **Author :** Prof. Krutika H. Churi

▶ **First Edition as per New Syllabus :**

August 2023

▶ **Tech-Neo ID :** BI-06

▶ **ISBN :** 978-93-5583-436-2

Copyright © by Prof. Krutika H. Churi

All rights reserved.

No part of this publication may be reproduced, copied, or stored in a retrieval system, distributed or transmitted in any form or by any means, including photocopy, recording, or other electronic or mechanical methods, without the prior written permission of the Publisher.

This book is sold subject to the condition that it shall not, by the way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above.

Published by

▶ **Mr. Sachin S. Shah**

Managing Director, B. E (Industrial Electronics)
An Alumnus of IIM Ahmedabad

▶ **Mrs. Nayana S. Shah, & Mr. Rahul S. Shah**

Permanent Address

Tech-Neo Publications LLP

Dugane Ind. Area, Survey No. 28/25, Dhayari,
Near Pari Company, Pune. Maharashtra, India.

Email : info@techneobooks.in

Website : www.techneobooks.in

Printed at : Image Offset (Mr. Rahul Shah)

Dugane Ind. Area, Survey No. 28/25, Dhayari,
Near Pari Company, Pune - 411041. Maharashtra
State, India.

E-mail : rahulshahimage@gmail.com

About Managing Director...

- **Mr. Sachin Shah**

▶ Over 25 years of experience in Academic Publishing...

With over two and a half decades of experience in bringing out more than 1200 titles in Engineering, Polytechnic, Pharmacy, Computer Sciences and Information Technology.

▶ A driven Educationalist...

1. B.E. (Industrial Electronics) (1992 Batch) from Bharati Vidyapeeth's College of Engineering, affiliated to University of Pune.
2. An Alumnus of IIM Ahmedabad.
3. A Co-Author of bestselling book on "Basic Electrical Engineering" Basic Electronics Engineering" for Degree Course in Engineering
4. For over a decade, been working as a Consultant for Higher Education in USA and several other countries.

▶ With path-breaking career...

- A publishing career that started with handwritten cyclostyled notes back in 1992.
- Has to his credit, setting up and expansion of one of the leading companies in higher education publishing.

▶ An experienced professional and an expert...

- An energetic, creative & resourceful professional with extensive experience of closely working with the best & the most eminent authors of Publishing Industry, ensures high standards of quality in contents.
- Helping students to attain better understanding and in-depth knowledge of the subject.
- Simplifying the methods of learning and bridging the gap between the best authors in the publishing industry and the student community for decades.



**CAUTION : PHOTOCOPYING OF
COPYRIGHTED BOOK IS ILLEGAL**

**SAVE YOURSELF,
DON'T BUY PHOTOCOPIED BOOKS**

Books Published are protected under Copyright Act 1999 and sold subject to the condition that the book and any extract thereof **shall be not photocopied** and includes the said condition being imposed on any subsequent purchaser.

Any person found selling, stocking or carrying photocopied book may be arrested for indulging in criminal offence of copyright piracy and may be imprisoned for **3 years and also fined a sum of Rs. 2,00,000/- for first offence.**

Sharing of PDF's, any Drives, Links, Storing in Hard Disks, Pendrive and Circulating on Social Media like Instagram, Telegram, Facebook, Snapchat, Google Drive & Whatsapp etc also violates the Copyright Laws and will be reported to Cyber Crime Division.

Publisher has raided many such offenders. Their Machines were Seized. Criminal case has also been registered against them. Civil Suits are also filed for recovering damages. **Police investigations of Students who are indulged in this is also in process.**

Recently, the Supreme Court of India, in M/s Knit Pro International v. The State of NCT of Delhi on 20 May 2022, has observed and held that offences under Section 63 of the Copyright Act, 1957 ("Copyright Act") are **cognizable and non-bailable.**

"Name of informer will be kept highly confidential. On successful raid he will be suitably rewarded"

Call / WhatsApp us on +91 98504 29188
Email : info@techneobooks.in



Tech-Neo Publications LLP

Sr. No. 38/1, Behind Pari Company, Khedekar Industrial Estate, Narhe,
Maharashtra, Pune - 411041.

Email : info@techneobooks.in • Website : www.techneobooks.in

Preface

Dear Students,

I am extremely happy to present the book of **“Mobile Programming Practical”** for you. I have divided the subject into small chapters so that the topics can be arranged and understood properly. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.

I am thankful to Shri. Sachin Shah for the encouragement and support that they have extended to me. I am also thankful to the staff members of Tech-Neo Publications and others for their efforts to make this book as good as it is. We have jointly made every possible efforts to eliminate all the errors in this book. However if you find any, please let me know, because that will help me to improve further.

I am also thankful to my family members and friends for their patience and encouragement.

- Prof. Krutika H. Churi



Syllabus...

Mobile Programming Practical

B. Sc. (Information Technology)		Semester – III	
Course Name: Mobile Programming Practical		Course Code: USIT3P5	
Periods per week (1 Period is 50 minutes)		3	
Credits		2	
		Hours	Marks
Evaluation System	Practical Examination	2½	50
	Internal	--	--

The practical's will be based on HTML5, CSS, Flutter. (Android will be introduced later after they learn Java)

List of Practical

	Setting up Flutter, PhoneGAP Project and environment.
1.	Program to demonstrate the features of Dart language.
2.	Designing the mobile app to implement different widgets.
3.	Designing the mobile app to implement different Layouts.
4.	Designing the mobile app to implement Gestures.
5.	Designing the mobile app to implement the theming and styling.
6.	Designing the mobile app to implement the routing.
7.	Designing the mobile app to implement the animation.
8.	Designing the mobile app to implement the state management.
9.	Designing the mobile app working with SQLite Database.
10.	Designing the mobile app working with Firebase.



LAB Manual

Flutter Basics and Configuration

1. Introduction to Flutter

- Flutter is a cross-platform UI toolkit that allows us to create fast, beautiful, natively compiled applications for mobile, web, and desktop using Dart programming languages.
- It uses a single programming language and a single codebase to create the app. It is open-source and free. It was created by Google in May 2017 and is now run according to an ECMA standard. It is a technology that is becoming more and more popular for providing excellent native experiences.

1.1 FEATURES OF FLUTER

- **Cross-platform development** : Flutter makes it possible to create desktop, web, iOS, and Android apps from a single codebase. Significant time and money are therefore saved.
- **Hot Reload** : The hot reload function of Flutter increases developer efficiency by rapidly implementing code changes without restarting the application.
- **Flexible and expressive UI** : Building adaptable and visually appealing user interfaces is made possible by Flutter's layered architecture and robust widget collection.
- **Native performance** : On both iOS and Android, Flutter compiles to native code for smooth 60 fps animations and scrolling.
- **Free and open source** : With a helpful community of developers and contributors, Flutter is free and open source.
- **Accessible widgets** : Flutter widgets come with accessibility capabilities and user interfaces to help developers make accessible apps.
- **Rapid prototyping** : Building MVPs and prototypes is made simple by Flutter's extensive widget library.

1.2 ADVANTAGES OF FLUTER

- Dart offers an extensive library of software packages that you can use to increase the functionality of your application.

- Developers only need to write one code base to support both applications (on the iOS and Android platforms).
- Flutter needs less testing.
- Fast development is possible with Flutter due to its simplicity.
- Developers have complete control over the widgets and their layout with Flutter.
- Flutter provides excellent developer tools with great hot reload.

▶▶ 1.3 DISADVANTAGES OF FLUTTER

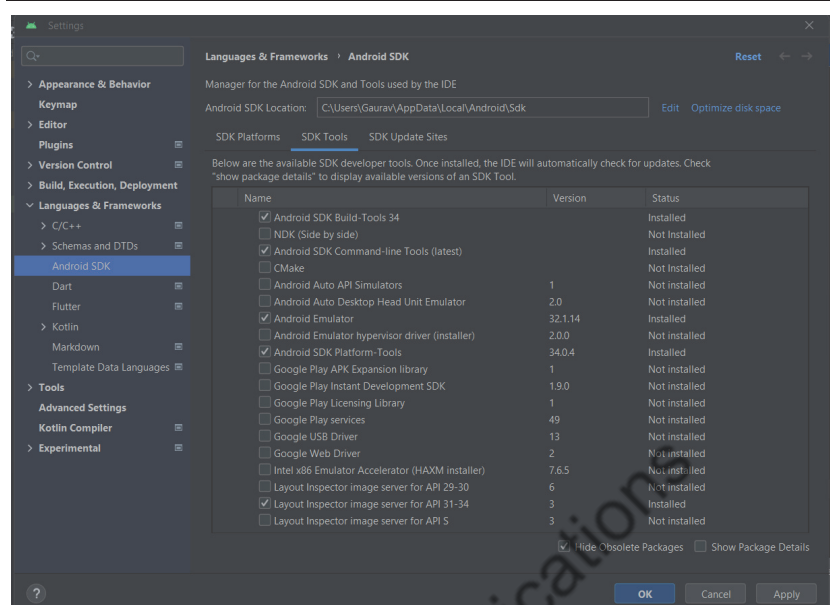
- Since it is coded in Dart language, a developer needs to learn a new language (though it is easy to learn).
- Modern frameworks make every effort to keep logic and user interface apart, whereas Flutter blends the two together. This can be avoided through the use of smart code and high-level modules to divide user interface from logic.
- Another framework for making mobile applications is Flutter. In a very overcrowded market, developers are struggling to select the best development tools.

▶▶ 1.4 LIST OF SOFTWARE'S REQUIRED FOR INSTALLATION

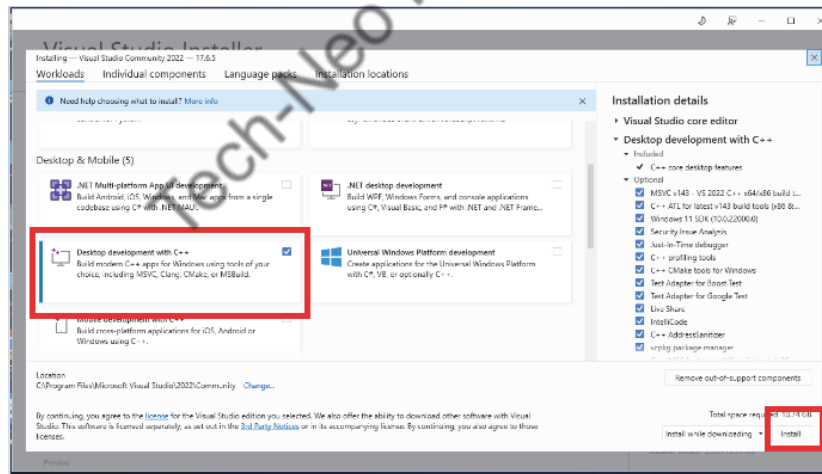
1. Flutter sdk (version 3.7.4 or latest)
2. Android Studio (version 2021.3.1.17 or latest)
3. Visual Studio (Visual Studio Community 2022 17.5.0 or latest)
4. Git
5. Google chrome browser

▶▶ 1.5 STEPS TO SET UP AN SYSTEM

1. First download Flutter sdk(you will get the downloaded file in zip format) then extract the zip file and place it in C drive.
2. Download and install Android Studio then start Android Studio, and go through the 'Android Studio Setup Wizard'. Then install the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.(Open Android studio - Tools - SDK manager - select the options given in the below screenshot and click on ok.)



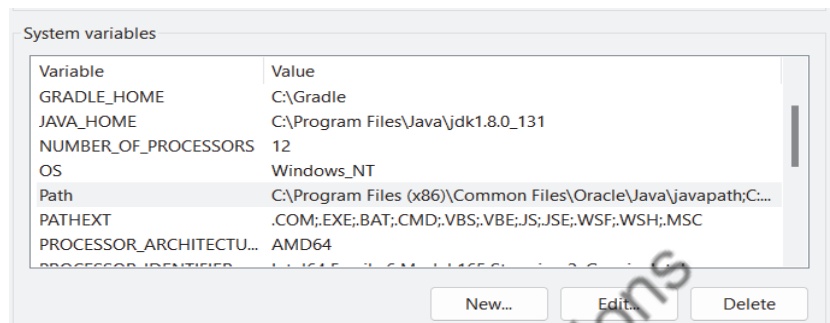
3. Install Visual Studio (Latest version).(after installation when you open Visual Studio select the field “Desktop development with C++” option and click on Install as shown below).



4. Install Git software.

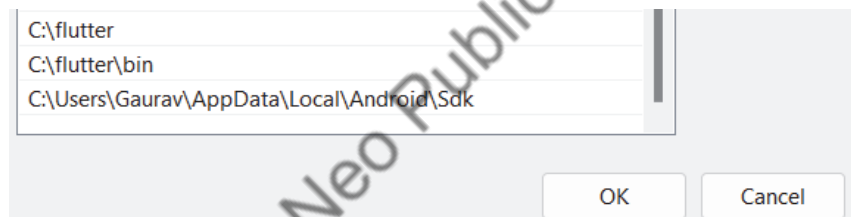
1.6 ENVIRONMENT SET UP

1. From the Start search bar, enter 'env' and select Edit environment variables for your account.
2. Under System variables check if there is an entry called Path:



Select path and click on Edit.

3. Add the following path to this variable and click on ok.



1.7 COMMAND PROMPT CODE

1. Open command prompt run the following command to see if there are any platform dependencies you need to complete the setup:
C:\flutter>flutter doctor
This command checks your environment and displays a report of the status of your Flutter installation. Check the output carefully for other software you might need to install or further tasks to perform.
2. Before you can use Flutter, you must agree to the licenses of the Android SDK platform. Run the following command to begin signing licenses.
C:\flutter>flutter doctor --android-licenses
3. Once you are done agreeing with licenses, run flutter doctor again to confirm that you are ready to use Flutter.
C:\flutter>flutter doctor
If everything is OK you will get following screen on command prompt

```
Command Prompt - flutter doctor
Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\student>flutter doctor

A new version of Flutter is available!

To update to the latest version, run "flutter upgrade".

Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.7.4, on Microsoft Windows [Version 10.0.19045.3208], locale en-IN)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 33.0.2)
[✓] Chrome - develop for the web
[✓] Visual Studio - develop for Windows (Visual Studio Community 2022 17.5.0)
[✓] Android Studio (version 2021.3)
[✓] VS Code (version 1.80.1)
[✓] Connected device (3 available)
[✓] HTTP Host Availability

• No issues found!

C:\Users\student>
```

4. To check the version of flutter installed on your system type the following command:

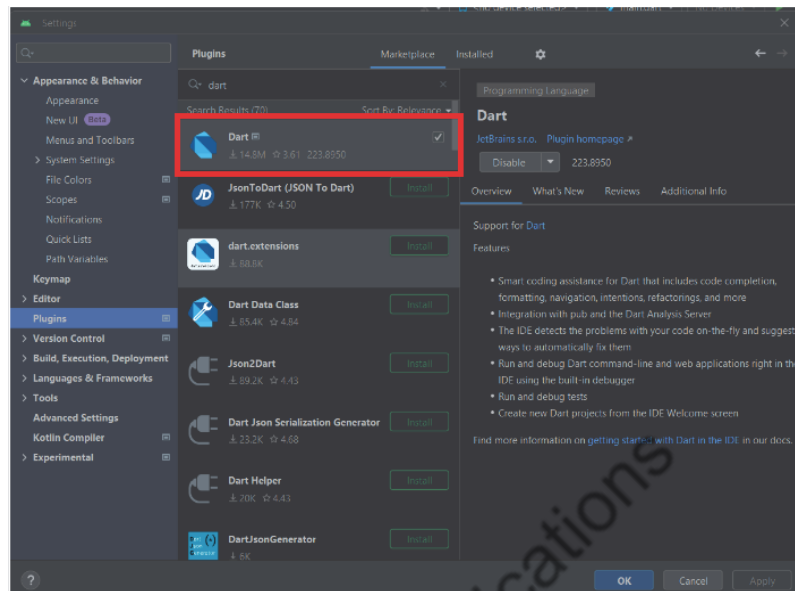
```
C:\flutter>flutter --version
```

```
C:\Users\student>flutter --version
Flutter 3.7.4 • channel stable • https://github.com/flutter/flutter.git
Framework • revision b4bce91dd0 (5 months ago) • 2023-02-21 09:50:50 +0800
Engine • revision 248290d6d5
Tools • Dart 2.19.2 • DevTools 2.20.1

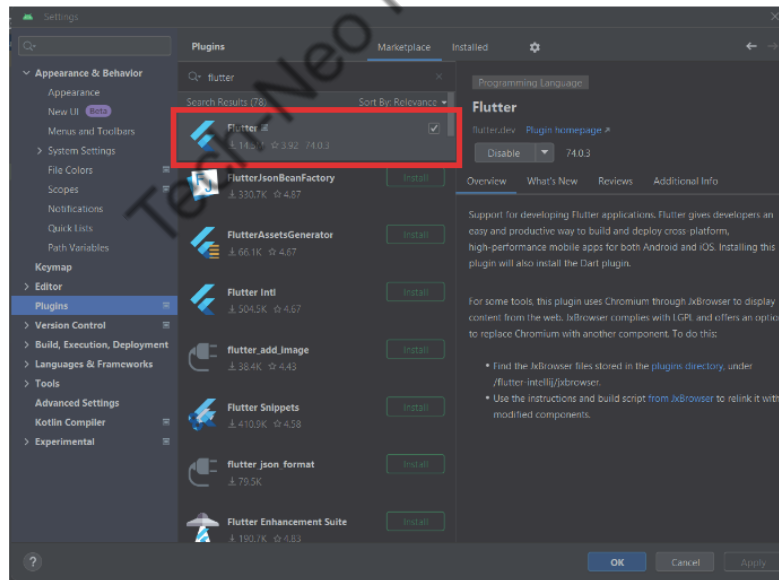
C:\Users\student>
```

➤ 1.8 INSTALLATION OF FLUTTER AND DART PLUGIN IN ANDROID STUDIO

- Flutter and Dart plugin in Android Studio provides a startup template to create a new Flutter application.
- It is used to run and debug Flutter applications in the Android studio.
- Steps to install Dart plugin in Android Studio are as follows:
 - Open android studio Go to file - setting - plugin - type dart in the search bar and then install Dart plugin and click on OK.



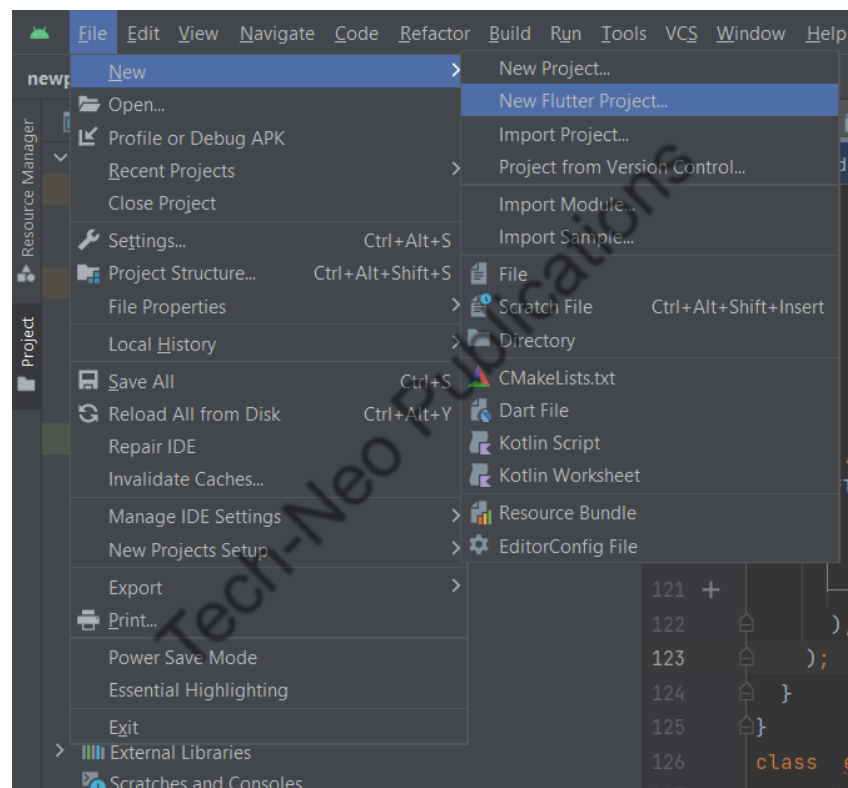
- Steps to install Flutter plugin in Android Studio are as follows:
 - Open android studio Go to file - setting - plugin - type flutter in the search bar and then install Flutter plugin and click on OK.



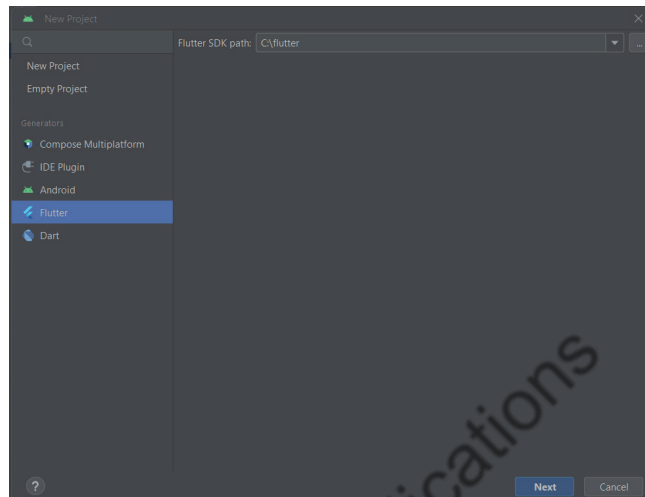
After installation of Dart and Flutter plugin Restart the android studio.

Example : Now we will create a simple application in Android Studio to understand the basics of the Flutter application. To create Flutter application, do the following steps:

- ▶ **Step 1 :** Open the Android Studio.
- ▶ **Step 2 :** Create the Flutter project. To create a project, go to File-> New->New Flutter Project.



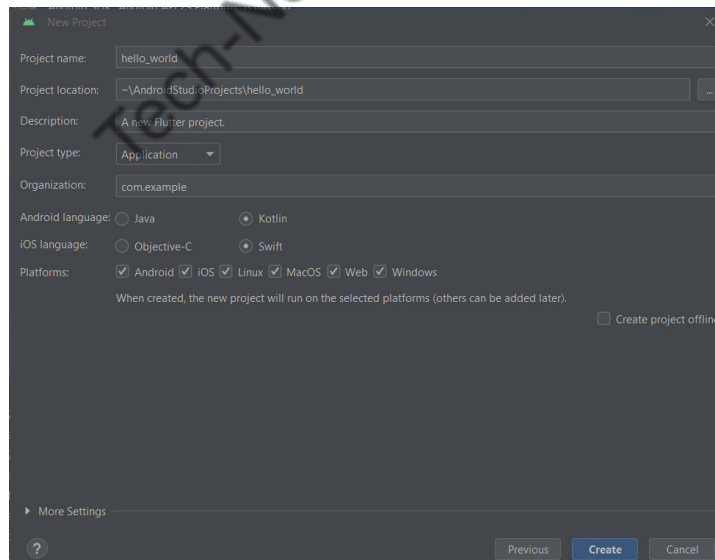
- ▶ **Step 3 :** Select Flutter and Set flutter sdk path as: c:\flutter as shown in below screenshot.



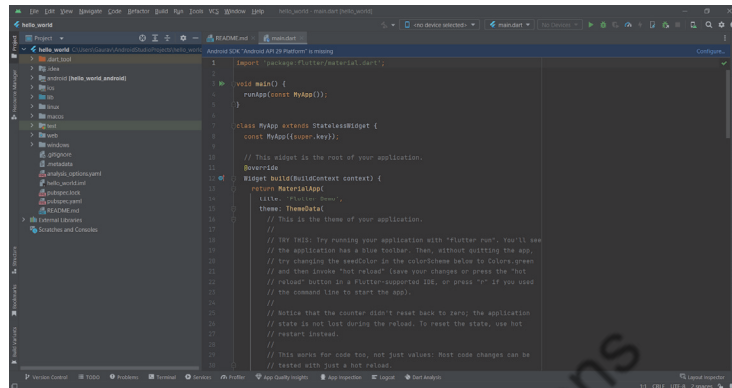
- ▶ **Step 4 :** Next, configure the application details as shown in the below screen and click on the Next button.

Set project name as hello_world and select project location.

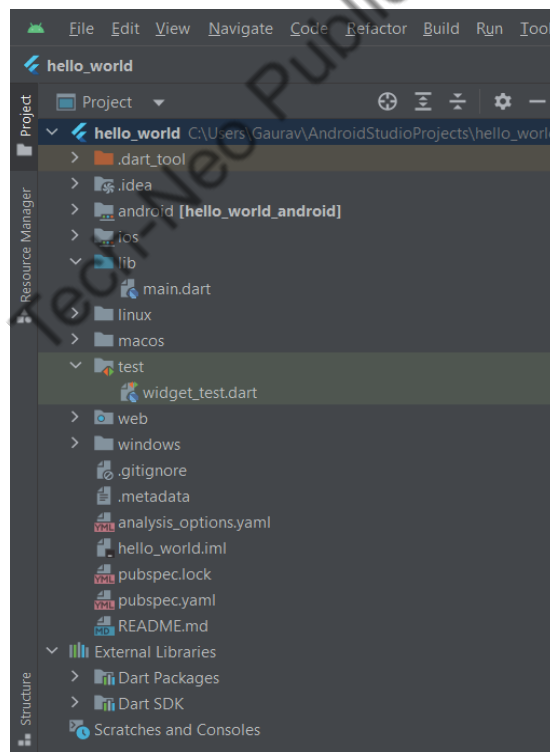
Keep the remaining field as it is and click on create.



- ▶ **Step 5 :** After clicking the Create button, it will take some time to create a project. When the project is created, you will get a fully working Flutter application with minimal functionality.



- ▶ **Step 6 :** Now, let us check the structure of the Flutter project application and its purpose. In the below image, you can see the various folders and components of the Flutter application structure.



- **android** : This folder holds a complete Android project and is used when you build the Flutter application for Android.
- **.idea** : This folder is at the very top of the project structure, which holds the configuration for Android Studio.
- **.ios** : This folder holds a complete Mac project and is used when you build the Flutter application for iOS.
- **.lib** : It is an essential folder, which stands for the library which contains Dart code written using flutter framework. By default, this folder contains the file **main.dart**, which is the entry file of the Flutter application.
- **.test** : This folder contains a Dart code, which is written for the Flutter application to perform the automated test when building the app.
- **.gitignore** : It is a text file containing a list of files, file extensions, and folders that tells Git which files should be ignored in a project. Git is a version-control file for tracking changes in source code during software development.
- **.metadata** : It is an auto-generated file by the flutter tools, which is used to track the properties of the Flutter project.
- **.packages** : It is an auto-generated file by the Flutter SDK, which is used to contain a list of dependencies for your Flutter project.
- **hello_world.iml** : It is always named according to the Flutter project's name that contains additional settings of the project. This file performs the internal tasks, which is managed by the Flutter SDK.
- **pubspec.yaml** : It is the project's configuration file that will be used a lot during working with the Flutter project. It allows you how your application works.
 - This file contains:
 - Project general setting such as name,description, and version of the project.
 - Project dependencies
 - Project assets(e.g. images)
- **pubspec.lock** : It is an auto-generated file based on the **.yaml** file. It holds a more detailed setup about all dependencies.
- **README.md** : It is an auto-generated file that holds information about the project. We can edit this file if we want to share information with the developers.

- ▶ **Step 7 :** Open the **main.dart** file (Expand lib folder -> main.dart) and replace the code with the following code.

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

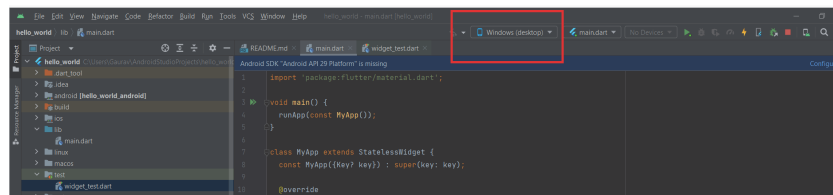
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

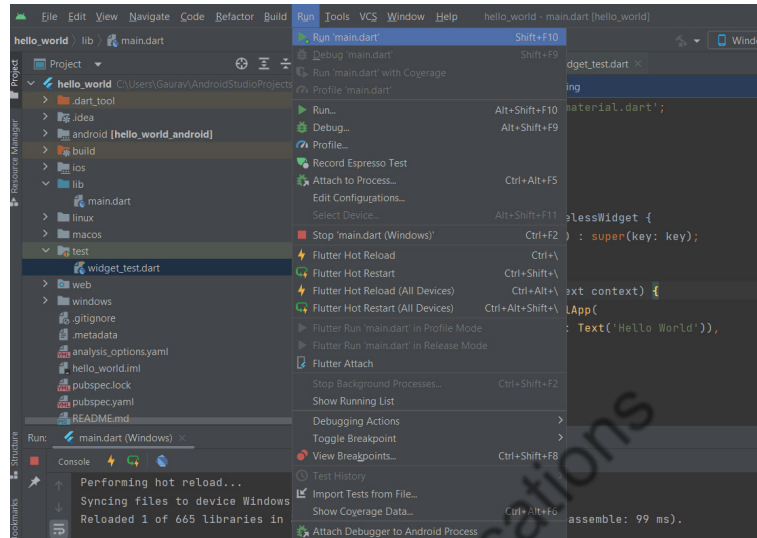
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: Center(child: Text('Hello World')),
    );
  }
}
```

- **import 'package:flutter/material.dart :** Here we are importing the package which has a definition for StatelessWidget, Center, Text, MaterialApp, and many more. It is like **#include<iostream>** in C++ program.
- **MyApp :** It is a user Defined class that inherits StatelessWidget, that is all the property of StatelessWidget is in GeeksForGeeks
- **Build :** It is a method that is responsible for drawing components on the Screen it takes a BuildContext as an argument that has information about which widget has to be displayed and in which order it has to be painted on the screen.

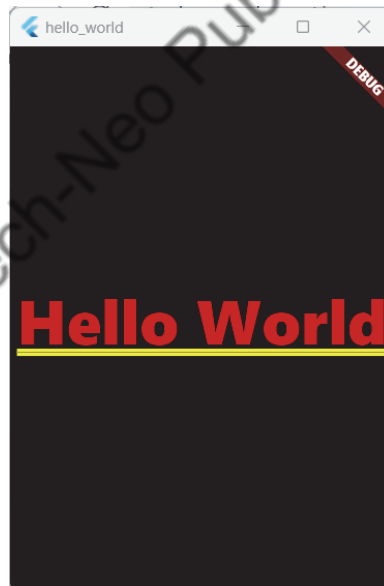
- ▶ **Step 8 :** Select windows desktop to see the output.



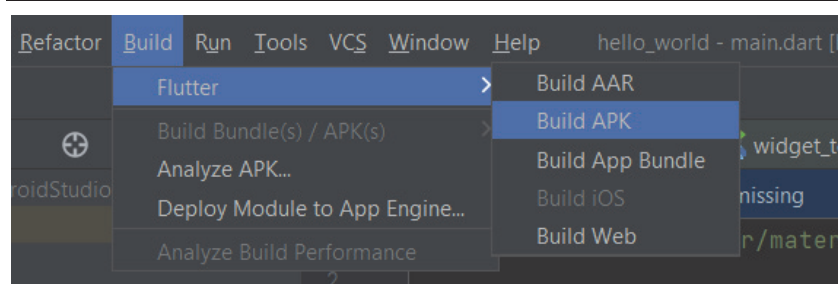
► **Step 9 :** Now run the project -> Go to Run → click on Run 'main.dart'



► **Step 10 :** Finally, you will get the output as shown below screen.



Note : To get the apk of your application you can build your project Go to Build -> Build -> Build APK as shown in below image.



After building the project you will get the path of your application apk in the terminal window on the same screen of android studio. (To get apk browse project location and follow the path given in the terminal window of android studio.)

```
Running Gradle task 'assembleRelease'... 71.6s
✓ Built build\app\outputs\flutter-apk\app-release.apk (16.9MB).
Process finished with exit code 0
```

► **Practical 1 : Aim** : Program to demonstrate the features of Dart language.

Dart Programming

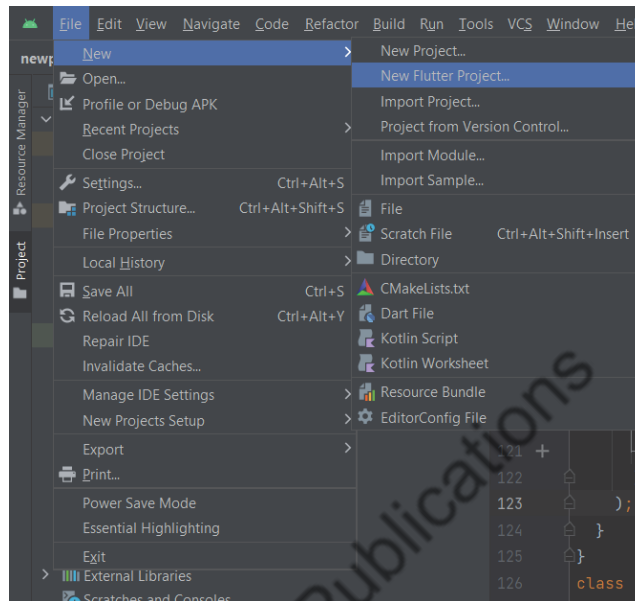
- Dart is a general-purpose, object-oriented, open-source programming language with C-style syntax that Google created in 2011.
- Dart programming is used to develop the front ends of user interfaces for websites and mobile applications.
- It is actively being developed, compiled to native machine code for creating mobile apps, Strongly Typed, and inspired by popular programming languages like Java, JavaScript, and C#.
- Dart is a compiled language, therefore you can't run your code straight away; the compiler must parse it first before converting it to machine code.
- Unlike other programming languages, it provides the majority of basic programming concepts including classes, interfaces, and functions.
- Dart is case-sensitive. This means that Dart differentiates between uppercase and lowercase characters.

The following example shows simple Dart programming.

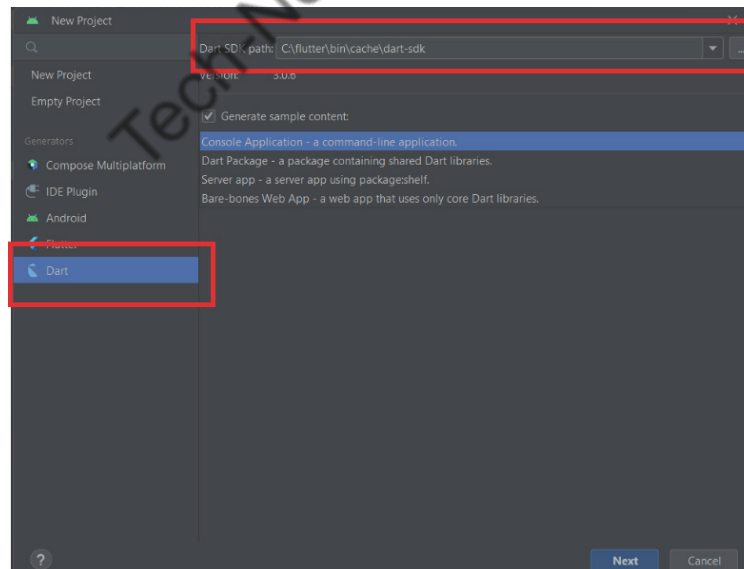
- Expand the lib folder and open the dart_program.dart file and write the below code.

```
void main() {
  print('hello world');
}
```

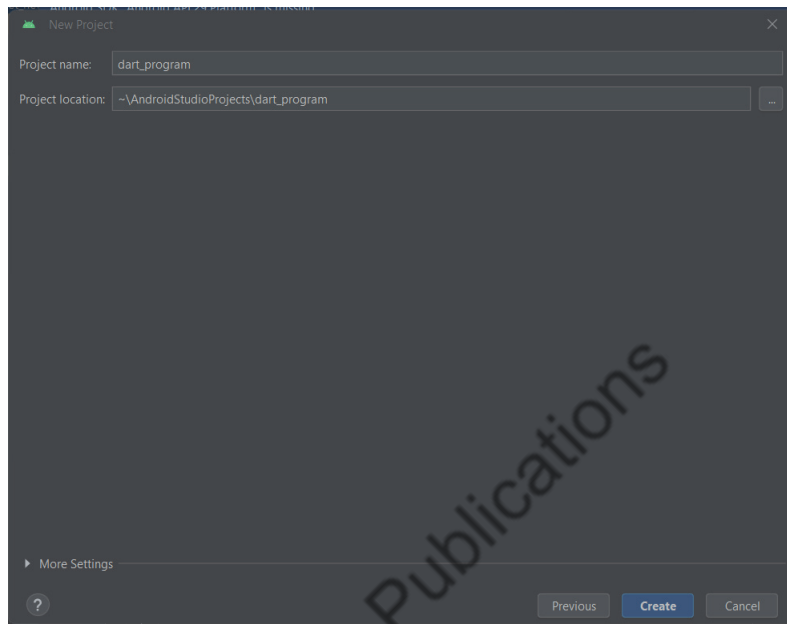
- ▶ **Step 1 :** Open android studio -> Click on file menu -> New ->New flutter project as shown in below screenshot.



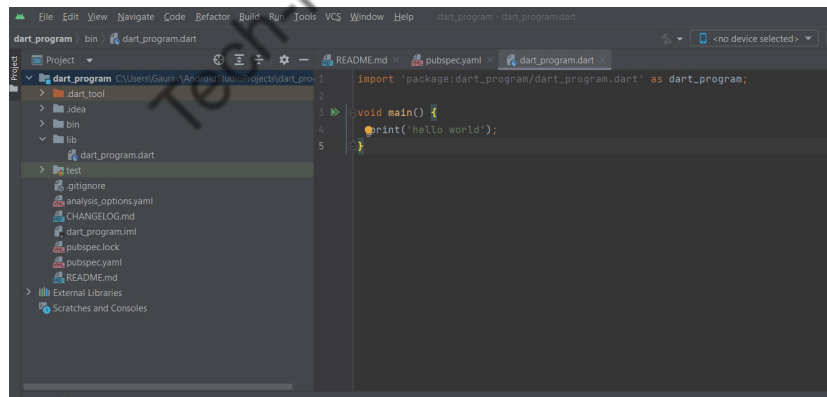
- ▶ **Step 2 :** Select Dart and Set Dart sdk path as: c:\flutter\bin\cache\dart-sdk then click on next.



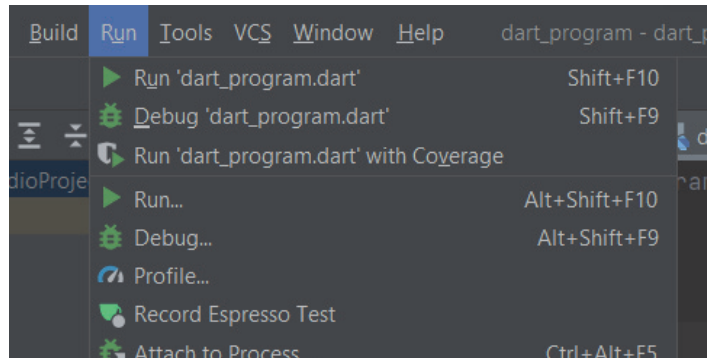
- ▶ **Step 3 :** Set project name dart_program and select project location then click on create.



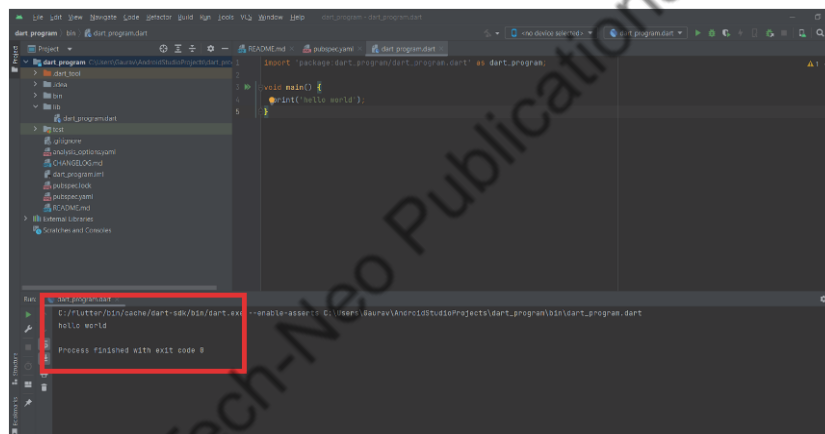
- ▶ **Step 4 :** In the dart_program.dart file type the following program shown in the following screenshot.



► **Step 5 :** Then click on Run - Run 'dart_program.dart'



► **Step 6 :** Finally, you will get the output as shown below screen.



- The main() function is a predefined method in Dart. This method acts as the entry point to the application. A Dart script needs the main() method for execution.
- print() is a predefined function that prints the specified string or value to the standard output i.e. the terminal.

1. Variables and Data types

- A variable is "a named space in the memory" that keeps values. This means that it acts like a program's value container.
- Data types simply describe the type and amount of data connected to variables and functions and are used to describe variable names.
- Following are the naming rules for an identifier –
 - Identifiers cannot be keywords.
 - Identifiers can contain alphabets and numbers.

- Identifiers cannot contain spaces and special characters, except the underscore (_) and the dollar (\$) sign.
- Variable names cannot begin with a number.
- Syntax to declare variable:
A variable must be declared before it is used. Dart uses the var keyword to achieve the same.

The syntax for declaring a variable is as given below –

```
var name = 'Neha';
```

All variables in dart store a reference to the value rather than containing the value. The variable called name contains a reference to a String object with a value of "Neha".

- The final and const keywords are used to declare constants. They are defined as Below :

```
void main() {  
  final a = 12;  
  const pi = 3.14;  
  print(a);  
  print(pi);  
}
```

- Dart language supports the following data types:
 - Numbers – It is used to represent numeric literals – Integer and Double.
 - Strings – It represents a sequence of characters. String values are specified in either single or double quotes.
 - Booleans – Dart uses the bool keyword to represent Boolean values – true and false.
 - Lists and Maps – It is used to represent a collection of objects.

A simple program for List can be defined as below :

```
void main() {  
  var list = [1,2,3,4,5];  
  print(list);  
}
```

 **Output for the above program is : 1,2,3,4,5**

1. Operators

- An expression is a specific type of statement that yields a value. Every expression is composed of:
 - Operands – Represents the data

- Operator – Defines how the operands will be processed to produce a value.
- Consider the following expression “2 + 3”. In this expression, 2 and 3 are operands and the symbol+ (plus) is the operator.

Types of Operators

1. Arithmetic Operators
2. Relational Operators
3. Type Test Operators
4. Logical Operators
5. Assignment Operators

1. Arithmetic Operators

The operators that are used to carry out arithmetic operations on the operands are included in this class of operators. They are binary operators i.e they act on two operands.

Operator Symbol	Operator Name	Operator Description
+	Addition	Use to add two operands
-	Subtraction	Use to subtract two operands
-expr	Unary Minus	It is Use to reverse the sign of the expression
*	Multiply	Use to multiply two operands
/	Division	Use to divide two operands
~/	Division	Use to divide two operands but give output in Integer
%	Modulus	Use to give remainder of two operands

Example

```
void main()
{
    int a = 2;
    int b = 3;
    // Adding a and b
    var c = a + b;
    print("Sum of a and b is $c");
    // Subtracting a and b
    var d = a - b;
    print("The difference between a and b is $d");

    // Using unary minus
    var e = -d;
    print("The negation of difference between a and b is $e");
    // Multiplication of a and b
    var f = a * b;
```



```

print("The product of a and b is $f");
// Division of a and b
var g = b / a;
print("The quotient of a and b is $g");
// Remainder of a and b
var i = b % a;
print("The remainder of a and b is $i");
}
    
```

Output

Sum of a and b is 5
 The difference between a and b is -1
 The negation of difference between a and b is 1
 Product of a and b is 6
 The quotient of a and b is 1.5
 The quotient of a and b is 1
 The remainder of a and b is 1

2. Relational Operators

The operators that carry out relational operations on the operands are included in this class of operators.

Operator Symbol	Operator Name	Operator Description
>	Greater than	Identify the larger operand and provide the result as Boolean expression.
<	Less than	Identify the smaller operand and provide the result as Boolean expression.
>=	Greater than or equal to	Identify which operand is greater or equal to each other and give the result as a boolean expression.
<=	less than equal to	Identify which operand is less than or equal to each other and give the result as a boolean expression.
==	Equal to	Identify whether the operand are equal to each other or not and give the result as a boolean expression.
!=	Not Equal to	Identify whether the operand are not equal to each other or not and give the result as a boolean expression.

Example : Using Relational Operators in the program

```
void main()
{
int a = 2;
int b = 3;

// Greater between a and b
var c = a > b;
print("a is greater than b is $c");

// Smaller between a and b
var d = a < b;
print("a is smaller than b is $d");

// Greater than or equal to between a and b
var e = a >= b;
print("a is greater than or equal to b is $e");

// Less than or equal to between a and b
var f = a <= b;
print("a is smaller than or equal to b is $f");

// Equality between a and b
var g = b == a;
print("a and b are equal is $g");

// Inequality between a and b
var h = b != a;
print("a and b are not equal is $h");
}
```

Output

```
a is greater than b is false
a is smaller than b is true
a is greater than or equal to b is false
a is smaller than or equal to b is true
a and b are equal is false
a and b are not equal is true
```

3. Type Test Operators

Operators that are used to compare operands are included in this class of operators.

Operator Symbol	Operator Name	Operator Description
is	is	Gives boolean value true as output if the object has specific type
is!	Is not	Gives boolean value false as output if the object has specific type

Example : Using Type Test Operators in the program

```
void main()
{
String a = 'JAY';
double b = 3.3;

// Using is to compare
print(a is String);

// Using is! to compare
print(b is !int);
}
```

Output

```
true
True
```

4. Logical Operators

Operators that logically combine two or more operand conditions are included in this class of operators.

Operator Symbol	Operator Name	Operator Description
&&	AND Operator	Use to add two conditions and if both are true then it will return true.
	OR Operator	Use to add two conditions and if even one of them is true then it will return true.
!	NOT Operator	It is use to reverse the result.

Example : Using Logical Operators in the program

```
void main()
{
int a = 15;
int b = 12;

// Using And Operator
bool c = a > 10 && b < 10;
print(c);

// Using Or Operator
bool d = a > 10 || b < 10;
print(d);

// Using Not Operator
bool e = !(a > 10);
print(e);
}
```

Output

false
true
false

5. Assignment Operators

The operators that are used to assign values to the operands are included in this class of operators.

Operator Symbol	Operator Name	Operator Description
=	Equal to	Use to assign values to the expression or variable
??=	Assignment operator	Assign the value only if it is null.

Example

```
void main()
{
int a = 5;
```

```
int b = 7;

// Assigning value to variable c
var c = a * b;
print(c);

// Assigning value to variable d
var d;
d ??= a + b; // Value is assign as it is null
print(d);
// Again trying to assign value to d
d ??= a - b; // Value is not assign as it is not null
print(d);
}
```

Output

```
35
12
12
```

3. Decision Making and Loops

- **Conditional statements** : A decision making block evaluates a condition before the instructions are executed. Dart supports If, If..else and switch statements.

1. If Statement

- The statements contained in this kind of statement merely check the condition, and if it is true, they are executed; however, if it is false, the statements are simply neglected by the code.

The syntax of if statement is given below.

Syntax

```
If (condition) {
    //statement(s)
}
```

The following example shows example of If statement loop:

```
void main () {
    // define a variable which hold numeric value
    var n = 35;

    // if statement check the given condition
```

```
if (n<40){
    print("The number is smaller than 40");
};
}
```

Output

The number is smaller than 40

2. If-else statement

- An optional else block can come after an if. If the Boolean expression checked by the if block returns false, the else block will be executed.

Following is the syntax.

```
if(boolean_expression){
    // statement(s) will execute if the Boolean expression is true.
} else {
    // statement(s) will execute if the Boolean expression is false.
}

void main () {
    // define a variable which holds a numeric value
    var age = 17;

    // if statement check the given condition
    if (age > 18) {
        print("You are eligible for voting");
    } else {
        print("You are not eligible for voting");
    }
}
```

Output

You are not eligible for voting

3. else...if Ladder :

- This kind of statement merely checks the condition; if it is true, the statements contained inside are executed; if not, other if conditions are checked; if they are true, they are executed; and if not, they are checked. Up to the ladder is finished, this process is repeated.

Syntax

```
if ( condition1 ){
    // body of if
}
```

```
else if ( condition2 ){  
    // body of if  
}  
.  
.  
.  
else {  
    // statement  
}
```

Example

```
void main() {  
    var num = 2;  
    if(num > 0) {  
        print("$num is positive");  
    }  
    else if(num < 0) {  
        print("$num is negative");  
    } else {  
        print("$num is neither positive nor negative");  
    }  
}
```

Output

2 is positive

- **Loops** : Loops are used to repeat a block of code until a specific condition is met. Dart supports for, for..in , while and do..while loops.

4. For loop

- The for loop is used when we know how many times a block of code will execute. It is quite the same as the C for loop.

Syntax

```
for(Initialization; condition; incr/decr) {  
    // loop body  
}
```

Example :

For loop Example :

The following example shows example of for loop:

```
void main() {
  for (int i = 0; i < 5; i++) {
    print('HI ${i + 1}');
  }
}
```

Output

```
HI 1
HI 2
HI 3
HI 4
HI 5
```

1. for... in Loop

- It only iterates each element once, using a dart object or expression as the lone iterator. The element's value is bound to var, which is valid and available for the body of the loop.
- The loop will keep going until there are no more elements in the iterator.

Syntax

```
for (var in expression) {
  //statement(s)
}
```

Example

```
void main()
{
  var abc = [1,2,3,4];
  for(var i in abc) //for..in loop to print list element
  {
    print(i); //to print the number
  }
}
```


Output

```
1
2
3
4
```

2. while loop

- The while loop repeats a section of code until the specified expression evaluates to false. When we don't know the number of executions, it is more efficient.

Syntax

```
while(condition) {
    // loop body
}
```

Example

```
void main()
{
    var a = 1;
    var maxnum = 6;
    while(a<maxnum){ // it will print until the expression return false
        print(a);
        a = a+1; // increase value 1 after each iteration
    }
}
```

Output

```
1
2
3
4
5
```

3. Functions

- A function is a group of statements that together performs a specific task.

```
void main() {
    add(3,4);
}
```

```
void add(int a,int b) {  
    int c;  
    c = a+b;  
    print(c);  
}
```

Output

7

4. Object Oriented concept

- Dart is an object-oriented language. It supports object-oriented programming features like classes, interfaces, etc.

Example

```
class Employee {  
    String name="";  
  
    //getter method  
    String get emp_name {  
        return name;  
    }  
    //setter method  
    void set emp_name(String name) {  
        this.name = name;  
    }  
    //function definition  
    void result() {  
        print(name);  
    }  
}  
void main() {  
    //object creation  
    Employee emp = new Employee();  
    emp.name = "employee1";  
    emp.result(); //function call  
}
```

Output

employee1

► **Practical 2 : Aim : Designing mobile app to implement different widgets.**

 **What are widgets in Flutter ?**

- The primary class hierarchy in the Flutter framework consists of widgets. An immutable description of a component of a user interface is referred to as a widget. It is possible to inflate widgets into elements that control the underlying render tree.
- The Flutter app's screens are made up entirely of widgets. The selection and arrangement of the widgets used to design the apps have a significant impact on how the screen is shown. A tree of widgets makes up the code structure of an app.
- Flutter offers a wide range of fundamental widgets to enable platform-independent creation of both simple and complex user interfaces.
- It includes a text widget, row widget, column widget, container widget, and many more.

 **Types of widget**

In Flutter, there are mainly two types of widget:

1. StatelessWidget
2. StatefulWidget

1. **Stateless Widget** : There is no state information available in the StatelessWidget. Throughout its entire lifespan, it doesn't change. Text, Row, Column, and Container are a few examples of StatelessWidgetes.
2. **StatefulWidget** : A StatefulWidget has state information. The state object and the widget are its two core classes. The state object and the widget are its two core classes. It is dynamic because the internal data may vary during the course of the widget's lifespan. The build() method is absent from this widget. A class that extends the Flutter's State Class is returned by the createState() method, which is present. Checkbox, Radio, Slider, InkWell, Form, and TextField are a few examples of StatefulWidgetes.

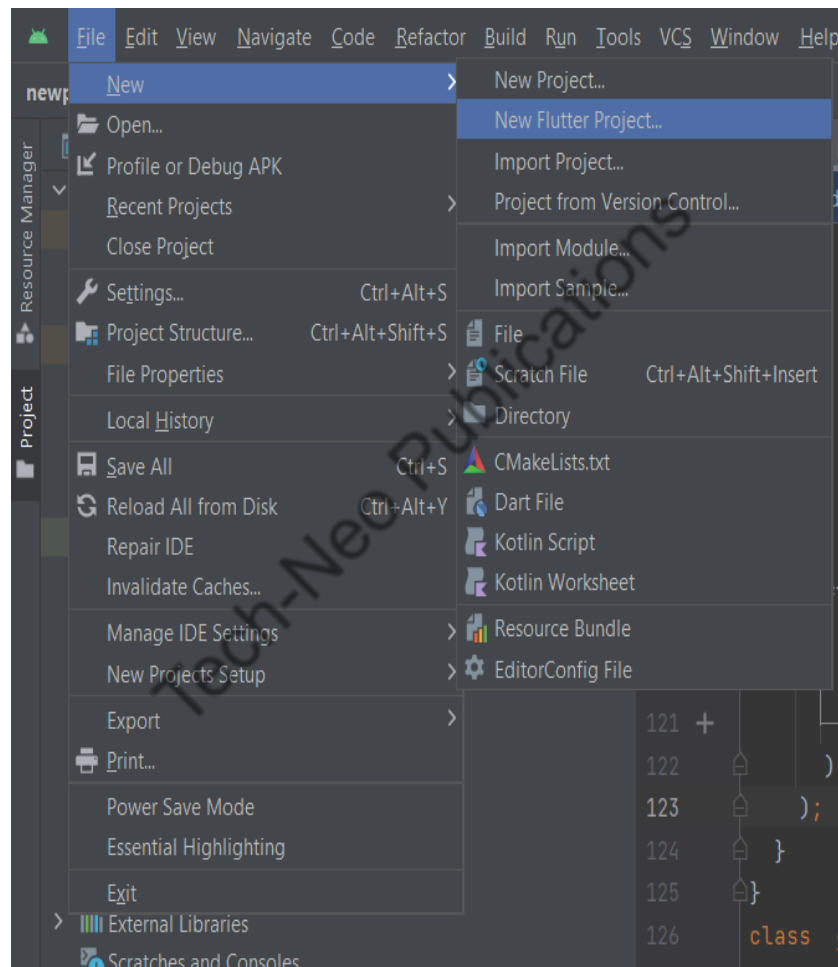
 **Basic widgets**

- **Text** : To write anything on the screen.
- **Image** : A widget that displays the image.
- **Icon** : A material design icon.
- **Row** : Layout a list of child widgets in the Horizontal direction.
- **Column** : Layout a list of child widgets in the vertical direction.
- **Elevated button** : A filled button whose material elevates when pressed.
- **Scaffold** : Implements the basic material design visual layout structure.
- **Flutter logo** : The flutter logo, in widget form.
- **AppBar** : To create a bar at the top of the screen.
- **Container** : To contain any widget that combines common painting, positioning, and sizing widgets.

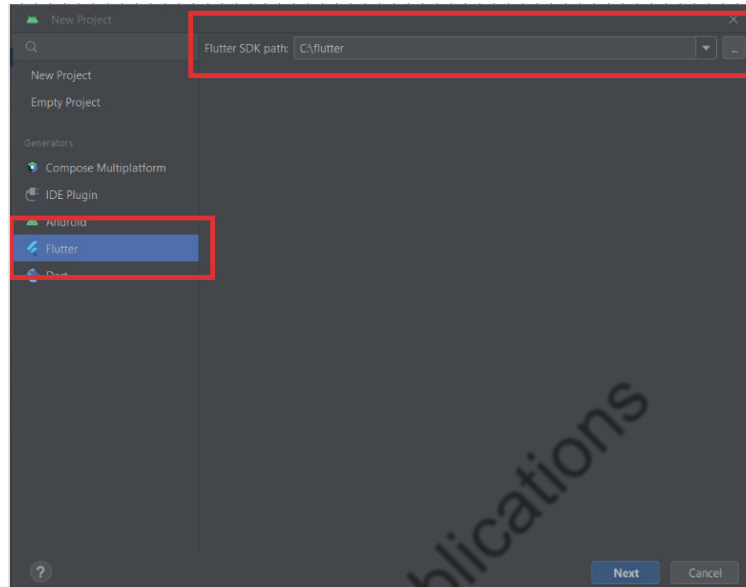
1. Following is the example to use Image in the application

The easiest option to load and display an image in *Flutter* is by including the image as assets of the application and loading it into the widget on demand.

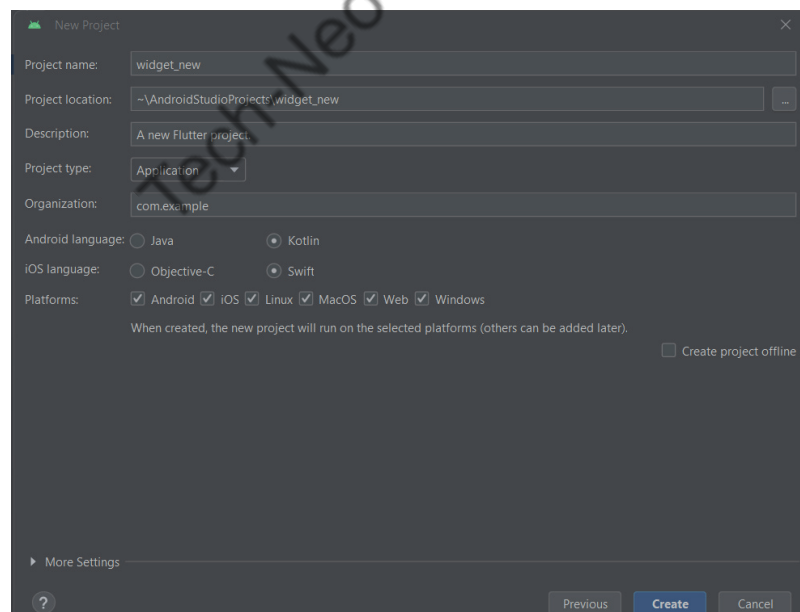
- ▶ **Step 1 :** Open android studio -> Click on file -> New -> New flutter project as shown in below screenshot.



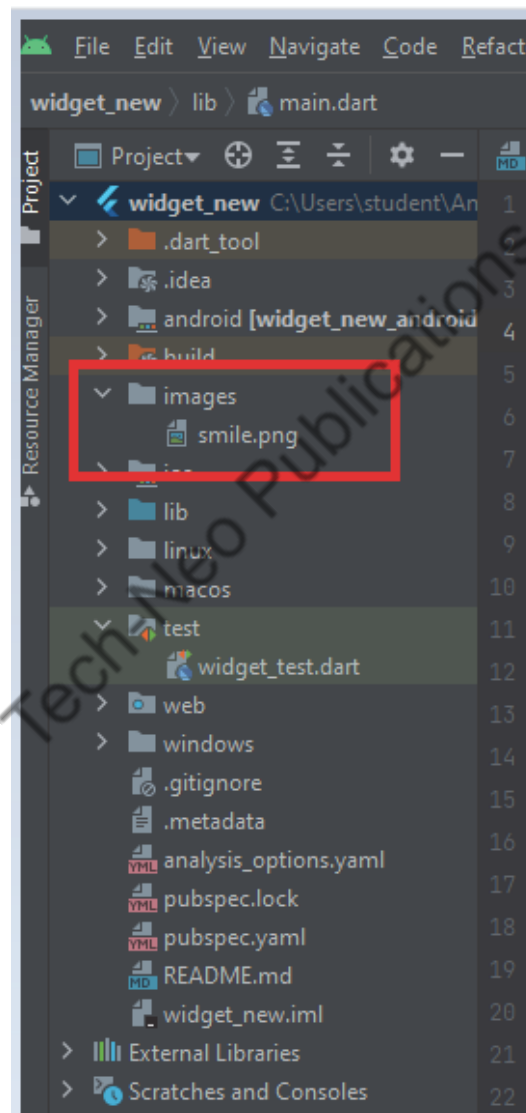
- ▶ **Step 2 :** Select Flutter and Set flutter sdk path as: c:\flutter as shown below.



- ▶ **Step 3 :** Set project name widget_new and select project location. Keep the remaining field as it is and click on create.



- ▶ **Step 4 :** Create a folder, images in the project folder and place the necessary images. (Right click on project name – click on New – Directory – Set name as images then copy any image to this folder images.) after creating the image folder, the project structure will look like the following.



- **Step 5:** Specify the assets in the pubspec.yaml as shown below –

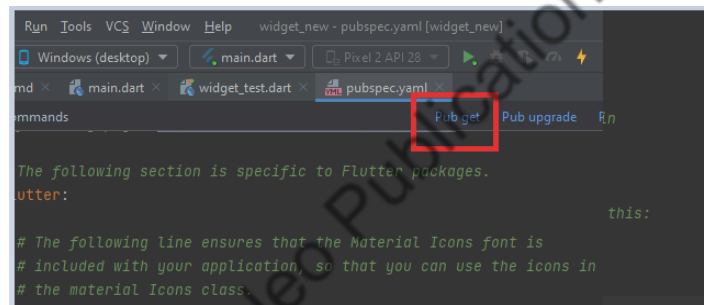
pubspec.yaml

```
# The following section is specific to Flutter packages.
flutter:

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

  # To add assets to your application, add an assets section, like this:
  assets:
    - images/smile.png
```

- **Step 6:** Click on Pub get.



If everything is OK you will get following message in terminal:

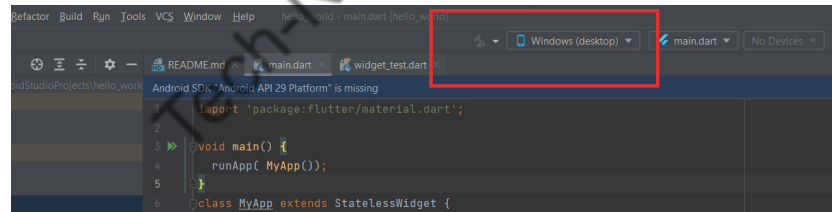
```
77 # fonts:
78 # family: Sabulor
Document 1/1 > flutter:
Messages: [widget_new] Flutter x
lints 2.0.1 (2.1.1 available)
matcher 0.12.13 (0.12.16 available)
material_color_utilities 0.2.0 (0.8.0 available)
meta 1.8.0 (1.9.1 available)
path 1.8.2 (1.8.3 available)
source_span 1.9.1 (1.10.0 available)
stack_trace 1.11.0 (1.11.1 available)
stream_channel 2.1.1 (2.1.2 available)
test_api 0.4.16 (0.6.1 available)
Got dependencies!
Process finished with exit code 0
```

- ▶ **Step 7 :** Type following code in main.dart file.

main.dart

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("My first App"),
        ),
        body: Image(
          image: AssetImage('images/smile.png'),
        ),
      ),
    );
  }
}
```

- ▶ **Step 8 :** To see the output select **windows desktop** as shown in below screenshot.



- ▶ **Step 9 :** Now Run the project -> Run -> run main.dart file
- ▶ **Step 10 :** When you run the program you will get the following output.



- Code to demonstrate the use of Text, Icon, ElevatedButton, Row, column widget.

main.dart :

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const Home(),
    );
  }
}
```

```
}  
class Home extends StatelessWidget {  
  const Home({super.key});  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(title: const Text('Sample Application')),  
      body: const Text('Text1', style: TextStyle(color: Colors.red, fontSize: 20)),  
    );  
  }  
}
```

Note : Following is the Sample code of individual Widget.To use it, replace the below-listed code with the above-highlighted code.

2. Text: It is used to display text.

Code :

```
const Text('Text1', style: TextStyle(color: Colors.red, fontSize: 20)),
```

Output



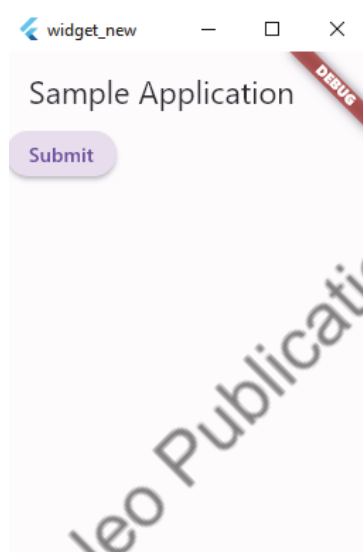
3. ElevatedButton

A filled button whose material elevates when pressed.

Code :

```
ElevatedButton(onPressed: (){}), child: Text('Submit')),
```

Output



4. Icon

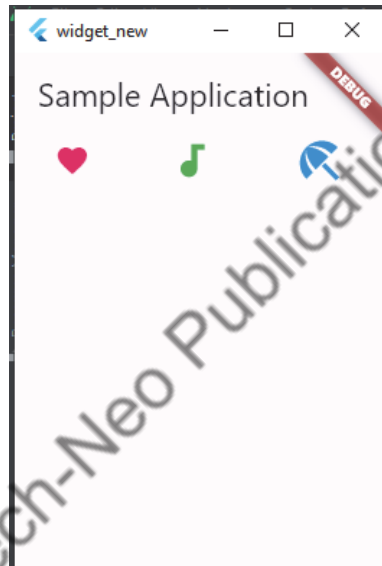
Icon widget is used to display a glyph from a font described in *IconData* class.

Code :

```
Row(  
  mainAxisAlignment: MainAxisAlignment.spaceAround,  
  children: <Widget>[  
    Icon(  
      Icons.favorite,  
      color: Colors.pink,  
      size: 24.0,  
      semanticLabel: 'Text to announce in accessibility modes',  
    ),  
    Icon(  
      Icons.audiotrack,  
      color: Colors.green,
```

```
    size: 30.0,  
  ),  
  Icon(  
    Icons.beach_access,  
    color: Colors.blue,  
    size: 36.0,  
  ),  
],  
)
```

Output



5. Row

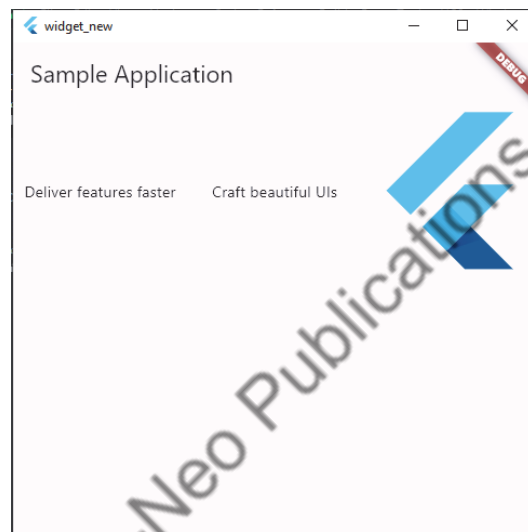
Layout a list of child widgets in the Horizontal direction.

Code

```
Row(  
  children: <Widget>[  
    Expanded(  
      child: Text('Deliver features faster', textAlign: TextAlign.center),  
    ),  
    Expanded(  
      child: Text('Craft beautiful UIs', textAlign: TextAlign.center),  
    ),  
  ],  
)
```

```
Expanded(  
  child: FittedBox(  
    child: FlutterLogo(),  
  ),  
),  
],  
)
```

Output



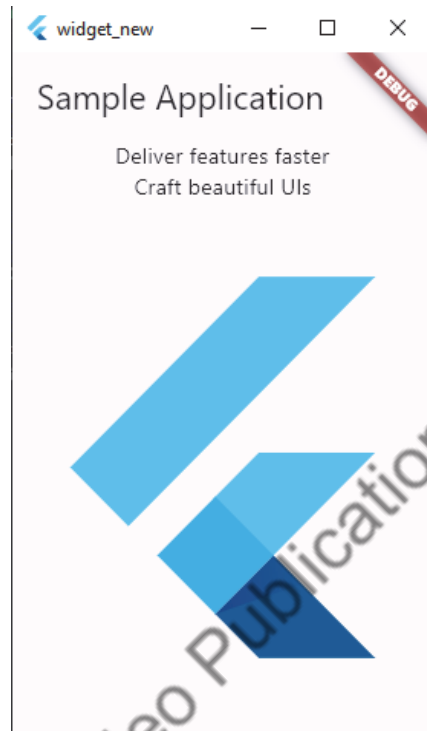
6. Column

Layout a list of child widgets in the Vertical direction.

Code

```
Column(  
  children: <Widget>[  
    Text('Deliver features faster'),  
    Text('Craft beautiful UIs'),  
    Expanded(  
      child: FittedBox(  
        child: FlutterLogo(),  
      ),  
    ),  
  ],  
)
```

Output



► **Practical 3 : Aim :** Designing the mobile app to implement different Layout.

Layout

- As the application is used on several platforms, including the web and mobile devices with varying screen sizes, layout in Flutter describes how the material expands in a certain area. As a result, dynamic content must be properly displayed to the user.
- Widgets are the foundation of the layout in Flutter. Everything in Flutter, including images, icons, text, and other elements, is a widget, including the layout models.
- It enables us to create layouts by combining several widgets. Flutter also has a few unique widgets, such as Center, Align, Container, and others, for organizing the user interface.

Types of Layout Widgets

1. Single Child Widget

- Single Child Widget is a type of widget that has only a single or one widget inside the parent widget. These single child widgets are very simple to use and make the code readable for the programmer.
- Flutter offers the programmer a large number of single child widgets so they may easily and quickly create a beautiful user interface.
- The following is a description of a few of Single Child widgets that are often used:
 - Container
 - Center
 - Align
 - Padding
 - BaseLine
 - SizedBox
 - ConstrainedBox

2. Multiple Child Widget

- Multiple Child Widgets are the kind of widgets that have more than one child widget, and each child widget has a different layout.
- For example, To create a table with rows and columns, for instance, a row widget is used, which lays its child object in the horizontal direction, while a column widget, on the other hand, lays its child object in the vertical direction. Composing this will create a whole new level of a complex widget.
- The following is a description of a few of Single Child widgets that are often used :
 - Column
 - Row
 - ListView
 - Stack
 - GridView

Example :

Code : main.dart

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
```

```
// This widget is the root of your application.
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
      useMaterial3: true,
    ),
    home: const Home(),
  );
}

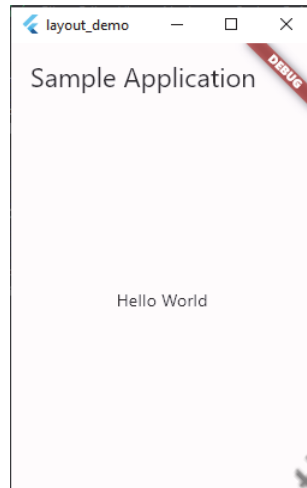
class Home extends StatelessWidget {
  const Home({super.key});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Sample Application')),
      body: const Center(
        child: Text('Hello World'),
      ),
    );
  }
}
```

Note : Following is the code of individual layout, to try this code replace the below code with the above highlighted code.

- 1. Center :** The child is centered using this widget. This widget derives from the class Align. It is one of the straightforward yet highly practical widgets used in Flutter.

```
const Center(
  child: Text('Hello World'),
),
```

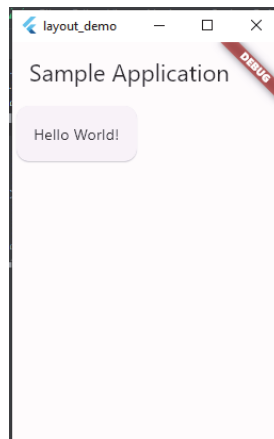

Output



2. **Padding** : One of the most popular widgets for giving child widgets the necessary padding to fit inside the layout widget is padding. Padding in Flutter can be provided using the EdgeInsets for the required sides.

```
const Card(  
  child: Padding(  
    padding: EdgeInsets.all(16.0),  
    child: Text('Hello World!'),  
  ),  
)
```

Output



- 3. Container :** One of the simple, well-known, and box-based widgets with lots of customizing options. It first surrounds the child with its child with padding and itself with the empty space called margin. It applies the additional constraints to the padded child.

```
Center(  
  child: Container(  
    margin: const EdgeInsets.all(10.0),  
    color: Colors.amber[600],  
    width: 48.0,  
    height: 48.0,  
  ),  
)
```

Output



- 4. Align :** As the name implies, this widget uses the alignment attribute to align each of its children within it. It has the option of sizing itself according to the size of its child.

```
Center(  
  child: Container(  
    height: 120.0,  
    width: 120.0,  
    color: Colors.blue[50],  
  ),  
)
```

```
child: const Align(  
  alignment: Alignment.topRight,  
  child: FlutterLogo(  
    size: 60,  
  ),  
),  
),  
)
```

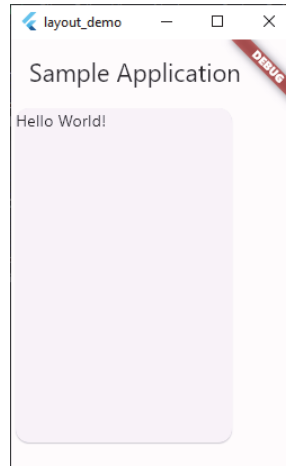
Output



5. **SizedBox** : This widget allows you to give the specified size to the child widget through all screens.

```
const SizedBox(  
  width: 200.0,  
  height: 300.0,  
  child: Card(child: Text('Hello World!'))  
)
```

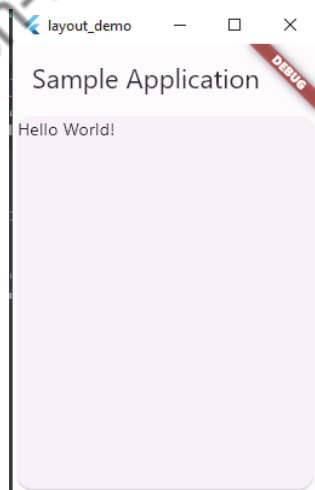
Output



6. **ConstrainedBox** : It is a widget that allows you to force additional constraints on its child widget.

```
ConstrainedBox(  
  constraints: const BoxConstraints.expand(),  
  child: const Card(child: Text('Hello World!')),  
)
```

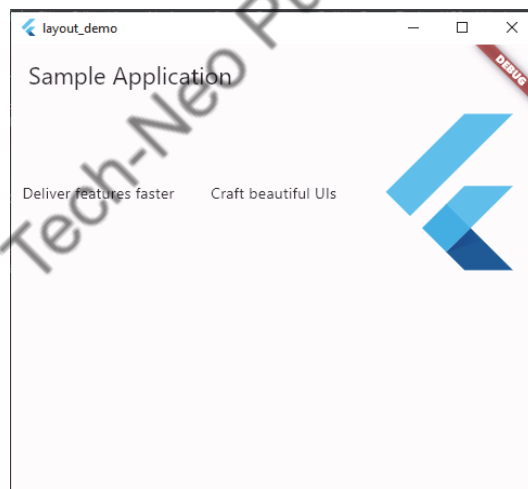
Output



7. **Row** : Row widgets, in contrast to Column widgets, allow the horizontal display of their child widgets. As it is considered incorrect to have more than one child in the row, the Flutter Row widget also does not scroll.

```
Row(  
  children: <Widget>[  
    Expanded(  
      child: Text('Deliver features faster', textAlign: TextAlign.center),  
    ),  
    Expanded(  
      child: Text('Craft beautiful UIs', textAlign: TextAlign.center),  
    ),  
    Expanded(  
      child: FittedBox(  
        child: FlutterLogo(),  
      ),  
    ),  
  ],  
)
```

Output



8. **Column** : This widget enables the vertical display of its child widget, much like a regular column. As it is seen to be an error to have more children than the allocated space in the room, the Column widget in Flutter does not scroll.

```
Column(  
  children: <Widget>[  
    Text('Deliver features faster'),  
    Text('Craft beautiful UIs'),  
    Expanded(  
      child: FittedBox(  
        child: FlutterLogo(),  
      ),  
    ),  
  ],  
)
```

Output

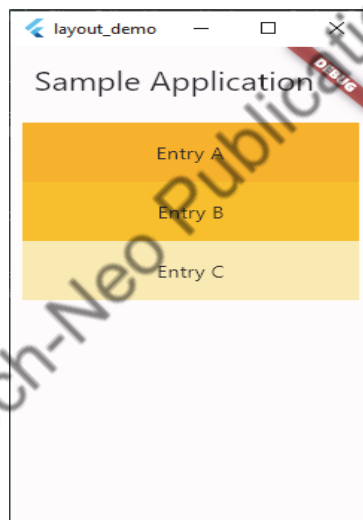


9. **ListView** : It is one of the widgets in Flutter that programmers use the most frequently. It permits the sequential display of its children in a scrolling fashion. The programmer has a lot of options when using Flutter to construct a ListView.

```
ListView(  
  padding: const EdgeInsets.all(8),  
  children: <Widget>[  
    Container(  
      height: 50,  
      color: Colors.amber[600],  
      child: const Center(child: Text('Entry A')),  
    ),  
  ],  
)
```

```
),  
  Container(  
    height: 50,  
    color: Colors.amber[500],  
    child: const Center(child: Text('Entry B')),  
  ),  
  Container(  
    height: 50,  
    color: Colors.amber[100],  
    child: const Center(child: Text('Entry C')),  
  ),  
],  
)
```

Output

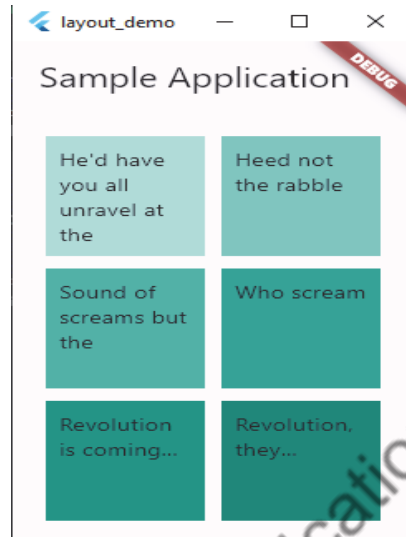


- 10. GridView :** With the help of this widget, programmers can make a scrollable 2D array of widgets. It enables the horizontal and vertical arrangement of the cells with repetitive patterns.

```
GridView.count(  
  primary: false,  
  padding: const EdgeInsets.all(20),  
  crossAxisSpacing: 10,  
  mainAxisSpacing: 10,  
  crossAxisCount: 2,
```

```
children: <Widget>[
  Container(
    padding: const EdgeInsets.all(8),
    color: Colors.teal[100],
    child: const Text("He'd have you all unravel at the"),
  ),
  Container(
    padding: const EdgeInsets.all(8),
    color: Colors.teal[200],
    child: const Text('Heed not the rabble'),
  ),
  Container(
    padding: const EdgeInsets.all(8),
    color: Colors.teal[300],
    child: const Text('Sound of screams but the'),
  ),
  Container(
    padding: const EdgeInsets.all(8),
    color: Colors.teal[400],
    child: const Text('Who scream'),
  ),
  Container(
    padding: const EdgeInsets.all(8),
    color: Colors.teal[500],
    child: const Text('Revolution is coming...'),
  ),
  Container(
    padding: const EdgeInsets.all(8),
    color: Colors.teal[600],
    child: const Text('Revolution, they...'),
  ),
],
)
```

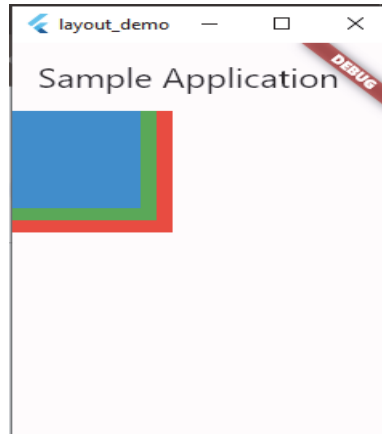

Output



- 11. Stack :** This widget enables the programmer to stack the various child widgets in relation to the box's edge. It is one of the most practical yet straightforward methods used by programmers to overlap the child widgets in Flutter.

```
Stack(  
  children: <Widget>[  
    Container(  
      width: 100,  
      height: 100,  
      color: Colors.red,  
    ),  
    Container(  
      width: 90,  
      height: 90,  
      color: Colors.green,  
    ),  
    Container(  
      width: 80,  
      height: 80,  
      color: Colors.blue,  
    ),  
  ],  
)
```

Output



► **Practical 4 : Aim :** Designing mobile app to implement Gestures.

Gestures

- Gestures are used to interact with an application. To physically interact with the program, it is typically utilized in touch-based devices.
- Simple physical interactions like a single tap on the screen or more difficult ones like swiping in a certain direction or scrolling along an application are all possible with gestures.
- Following are a few commonly used gestures:
 - **Tap :** putting a fingertip on the device's surface, holding it there for just a little moment, and then releasing it.
 - **Double Tap :** Tapping twice in a short time.
 - **Drag :** Drag is the act of touching the device's surface with the tip of the finger, moving it gradually, and then letting go.
 - **Flick :** A quicker method of performing something similar to dragging.
 - **Pinch :** Pinching the surface of the device using two fingers.
 - **Zoom :** Opposite of pinching.
 - **Panning :** Touching the device surface with the fingertip and moving it in the desired direction without releasing the fingertip.

GestureDetector

- Flutter uses the GestureDetector widget to track physical interaction with the UI of the application.
- Flutter provides excellent support for all types of gestures through its exclusive widget, GestureDetector.
- A widget can be inserted inside the GestureDetector widget in order to recognise a gesture that is targeted on it.

- GestureDetector will capture the gesture and dispatch multiple events based on the gesture.

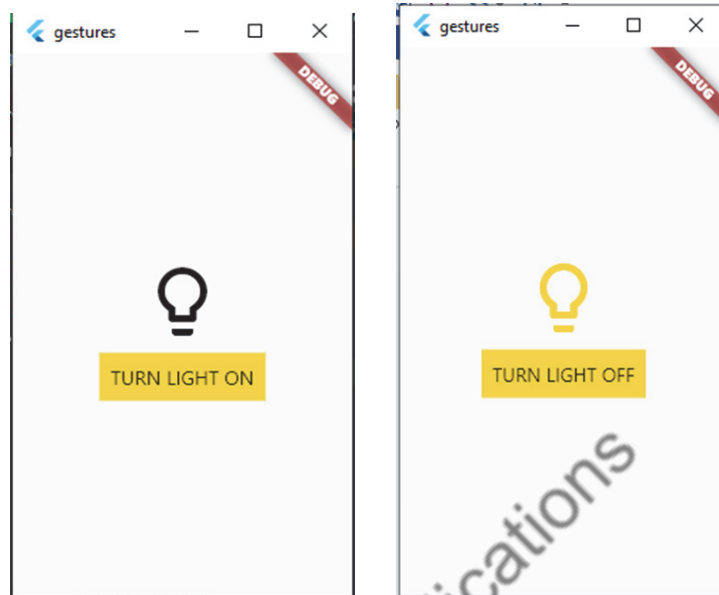
Example 1

This example contains a black light bulb wrapped in a GestureDetector. It turns the light bulb yellow when the "TURN LIGHT ON" button is tapped by setting the `_lights` field, and off again when "TURN LIGHT OFF" is tapped.

Code : main.dart

```
import 'package:flutter/material.dart';
/// Flutter code sample for [GestureDetector].
void main() => runApp(const GestureDetectorExampleApp());
class GestureDetectorExampleApp extends StatelessWidget {
  const GestureDetectorExampleApp({super.key});
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: GestureDetectorExample(),
    );
  }
}
class GestureDetectorExample extends StatefulWidget {
  const GestureDetectorExample({super.key});
  @override
  State<GestureDetectorExample> createState() =>
  _GestureDetectorExampleState();
}
class _GestureDetectorExampleState extends
State<GestureDetectorExample> {
  bool _lightIsOn = false;
  @override
  Widget build(BuildContext context) {
```

```
child: Column(  
  mainAxisAlignment: MainAxisAlignment.center,  
  children: <Widget>[  
    Padding(  
      padding: const EdgeInsets.all(8.0),  
      child: Icon(  
        Icons.lightbulb_outline,  
        color: _lightIsOn ? Colors.yellow.shade600 : Colors.black,  
        size: 60,  
      ),  
    ),  
    GestureDetector(  
      onTap: () {  
        setState(() {  
          // Toggle light when tapped.  
          _lightIsOn = !_lightIsOn;  
        });  
      },  
      child: Container(  
        color: Colors.yellow.shade600,  
        padding: const EdgeInsets.all(8),  
        // Change button text when light changes state.  
        child: Text(_lightIsOn ? 'TURN LIGHT OFF' : 'TURN LIGHT ON'),  
      ),  
    ),  
  ],  
),  
),  
);  
}  
}
```



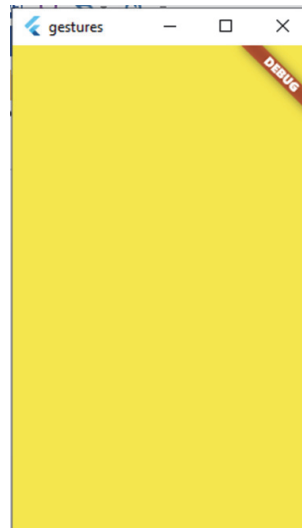
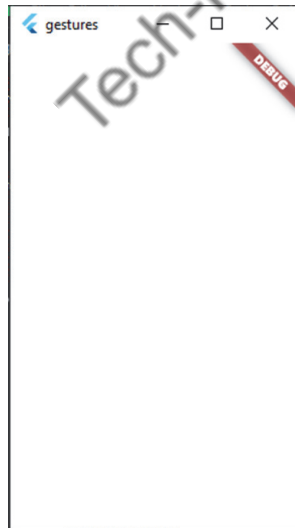
Example 2 :

This example uses a Container that wraps a GestureDetector widget which detects a tap. Since the GestureDetector does not have a child, it takes on the size of its parent, making the entire area of the surrounding Container clickable. When tapped, the Container turns yellow by setting the `_color` field. When tapped again, it goes back to white.

Code : main.dart

```
import 'package:flutter/material.dart';
/// Flutter code sample for [GestureDetector].
void main() => runApp(const GestureDetectorExampleApp());
class GestureDetectorExampleApp extends StatelessWidget {
  const GestureDetectorExampleApp({super.key});
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: GestureDetectorExample(),
    );
  }
}
class GestureDetectorExample extends StatefulWidget {
  const GestureDetectorExample({super.key});
  @override
```

```
State<GestureDetectorExample> createState() =>
  _GestureDetectorExampleState();
}
class _GestureDetectorExampleState extends State<GestureDetectorExample>
{
  Color _color = Colors.white;
  @override
  Widget build(BuildContext context) {
    return Container(
      color: _color,
      height: 200.0,
      width: 200.0,
      child: GestureDetector(
        onTap: () {
          setState(() {
            _color == Colors.yellow
            ? _color = Colors.white
            : _color = Colors.yellow;
          });
        },
      ),
    );
  }
}
```



► **Practical 5 : Aim :** Designing mobile app to implement the theming and styling.

Theming and Styling

- In order to share colors and font styles across an application, themes are a crucial component of the user interface (UI).
- An application's fonts and colors are designed using themes to make it look more professional.
- The Theme widget in Flutter is used to put themes into an application.
- It can be used for the entire program or only a specific area of it, such as the buttons and navigation bar, by defining it in the application's root.
- We can define app-wide themes, or use Theme widgets that define the colors and font styles for a particular part of the application.

Properties of Theme Widget

- **Child :** The *child* property takes in a widget as the object to show below the *Theme* widget in the widget tree.
- **Data :** This property takes in ThemeData class as the object to specify the styling, colors and typography to be used.
- **isMaterialAppTheme :** This property takes in a boolean (final) as the object. If it is set to true then the theme uses the material design.

Example :

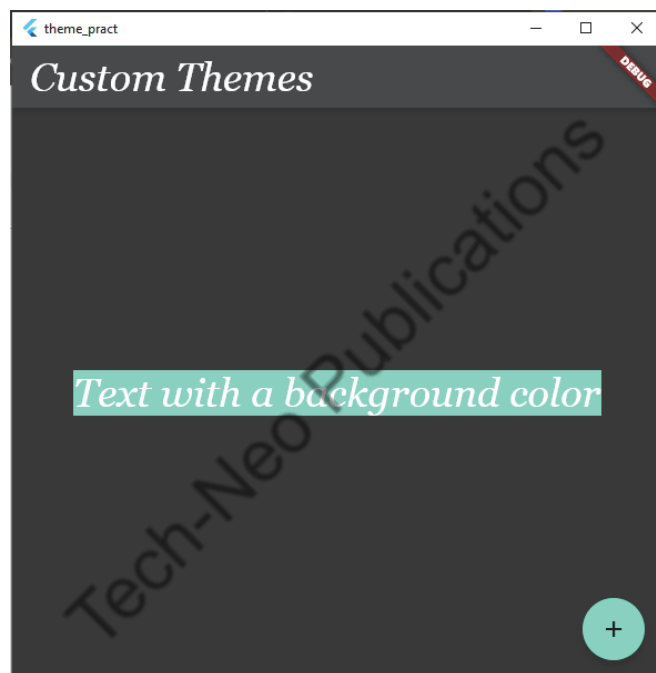
Code : main.dart:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    const appName = 'Custom Themes';
    return MaterialApp(
      title: appName,
      theme: ThemeData(
        // Define the default brightness and colors.
        brightness: Brightness.dark,
        primaryColor: Colors.lightBlue[800],
        // Define the default font family.
        fontFamily: 'Georgia',
        // Define the default `TextTheme`. Use this to specify the default
```

```
// text styling for headlines, titles, bodies of text, and more.
textTheme: const TextTheme(
  displayLarge: TextStyle(fontSize: 72, fontWeight: FontWeight.bold),
  titleLarge: TextStyle(fontSize: 36, fontStyle: FontStyle.italic),
  bodyMedium: TextStyle(fontSize: 14, fontFamily: 'Hind'),
),
),
home: const MyHomePage(
  title: appName,
),
);
}
}
class MyHomePage extends StatelessWidget {
  final String title;
  const MyHomePage({super.key, required this.title});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(title),
      ),
      body: Center(
        child: Container(
          color: Theme.of(context).colorScheme.secondary,
          child: Text(
            'Text with a background color',
            style: Theme.of(context).textTheme.titleLarge,
          ),
        ),
      ),
      floatingActionButton: Theme(
        data: Theme.of(context).copyWith(splashColor: Colors.yellow),
        child: FloatingActionButton(
          onPressed: () {},
          child: const Icon(Icons.add),
        ),
      ),
    );
  }
}
```


- In above code to share a Theme across an entire app, provide a ThemeData to the MaterialApp constructor. If no theme is provided, Flutter creates a default theme for you. If you don't want to inherit any application colors or font styles, create a ThemeData() instance and pass that to the Theme widget. Now that you've defined a theme, use it within the widgets' build() methods by using the Theme.of(context) method.
- The Theme.of(context) method looks up the widget tree and returns the nearest Theme in the tree. If you have a standalone Theme defined above your widget, that's returned. If not, the app's theme is returned.

Output



► **Practical 6 : Aim :** Design mobile app to implement the routing.

Route

- The apps show their content in full-screen containers known as pages or screens. The screens or pages are known as Routes in flutter.
- In a flutter, routes are referred to as Widgets.
- Using object-oriented principles, a route in Dart can be written in the form of "Class". Each route has its own contents and user interface (UI) and can be created as a distinct class.

Navigator

- Navigator is a widget that helps us to navigate between the routes.
- The navigator follows the stack method when dealing with the routes. Based on the actions made by the user, the routes are stacked one over the other and when pressed back, it goes to the most recently visited route.

Defining Home

- The "home page" must first be defined or initialized before we can begin navigating.
- The home page can be any route according to our needs. The home usually will be placed at the bottom of the navigator stack.

Navigating to a Page

- The navigator widget contains a method named `Navigator.push()` that allows users to go from their home to another route of the app.
- This method pushes the route on top of the home, thereby displaying the second route.

Navigating Back to Home

- To navigate back to home the navigator has a method called `Navigator.pop()`.
- This helps us to remove the present route from the stack so that we go back to our home route.

Example

Code: `main.dart`

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

void main() {
  runApp(const MaterialApp(
    home: HomeRoute(),
  ));
}

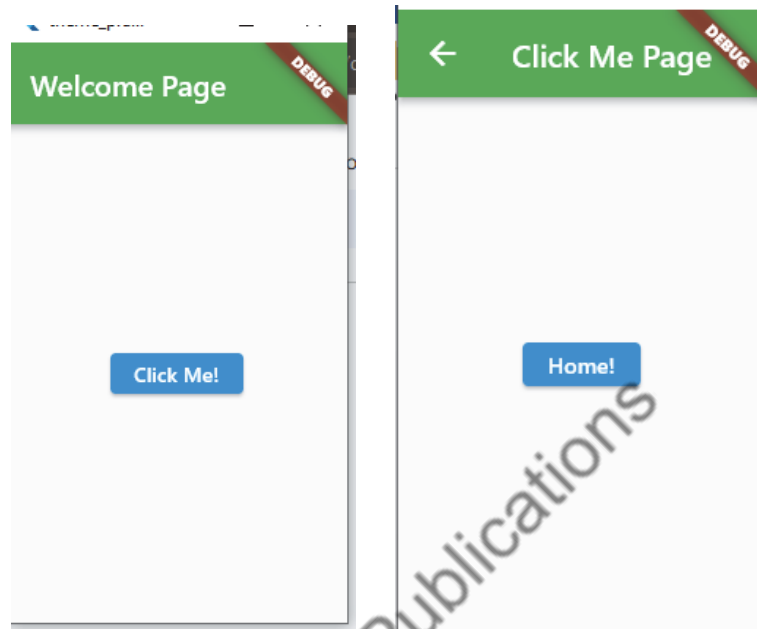
class HomeRoute extends StatelessWidget {
  const HomeRoute({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

```
        title: const Text('Welcome Page'),
        backgroundColor: Colors.green,
    ),
    body: Center(
      child: ElevatedButton(
        child: const Text('Click Me!'),
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => const SecondRoute()),
          );
        },
      ),
    );
}
}
```

```
class SecondRoute extends StatelessWidget {
  const SecondRoute({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Click Me Page"),
        backgroundColor: Colors.green,
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Home!'),
        ),
      ),
    );
  }
}
```

Output



► **Practical 7 : Aim :** Designing the mobile app to implement the animation.

Animation

- Animations improve user experiences and increase the level of interactivity in the applications.
- The Flutter Package offers several techniques for creating and utilizing animation in our app.
- Explicit animations like FadeTransition, SizeTransition, and SlideTransition are also included in the Flutter SDK. These simple animations are triggered by setting a beginning and ending point.

1. **Fade In transition**

Code : main.dart:

```
import 'package:flutter/material.dart';  
/// Flutter code sample for [FadeTransition].  
void main() => runApp(const FadeTransitionExampleApp());  
class FadeTransitionExampleApp extends StatelessWidget {  
  const FadeTransitionExampleApp({super.key});
```

```

@override
Widget build(BuildContext context) {
return const MaterialApp(
home: FadeTransitionExample(),
);
}
}

class FadeTransitionExample extends StatefulWidget {
const FadeTransitionExample({super.key});
@override
State<FadeTransitionExample> createState() =>
_FadeTransitionExampleState();
}

/// [AnimationController]s can be created with `vsync: this` because of
/// [TickerProviderStateMixin].
class _FadeTransitionExampleState extends State<FadeTransitionExample>
with TickerProviderStateMixin {
late final AnimationController _controller = AnimationController(
duration: const Duration(seconds: 2),
vsync: this,
)..repeat(reverse: true);
late final Animation<double> _animation = CurvedAnimation(
parent: _controller,
curve: Curves.easeIn,
);
@override
void dispose() {
_controller.dispose();
super.dispose();
}
@override
Widget build(BuildContext context) {
return ColoredBox(
color: Colors.white,
child: FadeTransition(
opacity: _animation,
child: const Padding(padding: EdgeInsets.all(8), child: FlutterLogo()),
),
); }
}

```

Output



2. Size transition

Code : main.dart :

```
import 'package:flutter/material.dart';  
/// Flutter code sample for [SizeTransition].  
void main() => runApp(const SizeTransitionExampleApp());  
class SizeTransitionExampleApp extends StatelessWidget {  
  const SizeTransitionExampleApp({super.key});  
  @override  
  Widget build(BuildContext context) {  
    return const MaterialApp(  
      home: SizeTransitionExample(), );  
  } }  
class SizeTransitionExample extends StatefulWidget {  
  const SizeTransitionExample({super.key});  
  @override  
  State<SizeTransitionExample> createState() =>  
    _SizeTransitionExampleState();  
}  
/// [AnimationController]s can be created with `vsync: this` because of  
/// [TickerProviderStateMixin].  
class _SizeTransitionExampleState extends State<SizeTransitionExample>  
  with TickerProviderStateMixin {  
  late final AnimationController _controller = AnimationController(  
    duration: const Duration(seconds: 3),
```

```
vsync: this,  
)..repeat();  
late final Animation<double> _animation = CurvedAnimation(  
parent: _controller,  
curve: Curves.fastOutSlowIn,  
);  
@override  
void dispose() {  
  _controller.dispose();  
  super.dispose();  
}  
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    body: SizeTransition(  
      sizeFactor: _animation,  
      axis: Axis.horizontal,  
      axisAlignment: -1,  
      child: const Center(  
        child: FlutterLogo(size: 500.0),  
      ), ), ), );  
} }
```

Output



2. Slide transition

Code : main.dart:

```
import 'package:flutter/material.dart';
/// Flutter code sample for [SlideTransition].
void main() => runApp(const SlideTransitionExampleApp());
class SlideTransitionExampleApp extends StatelessWidget {
  const SlideTransitionExampleApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('SlideTransition Sample')),
        body: const Center(
          child: SlideTransitionExample(),
        ),
      ),
    );
  }
}
class SlideTransitionExample extends StatefulWidget {
  const SlideTransitionExample({super.key});
  @override
  State<SlideTransitionExample> createState() =>
    _SlideTransitionExampleState();
}
class _SlideTransitionExampleState extends State<SlideTransitionExample>
  with SingleTickerProviderStateMixin {
  late final AnimationController _controller = AnimationController(
    duration: const Duration(seconds: 2),
    vsync: this,
  )..repeat(reverse: true);
  late final Animation<Offset> _offsetAnimation = Tween<Offset>(
    begin: Offset.zero,
    end: const Offset(1.5, 0.0),
  ).animate(CurvedAnimation(
    parent: _controller,
    curve: Curves.elasticIn,
  ));
  @override
```



```
void dispose() {  
  _controller.dispose();  
  super.dispose();  
}  
  
@override  
Widget build(BuildContext context) {  
  return SlideTransition(  
    position: _offsetAnimation,  
    child: const Padding(  
      padding: EdgeInsets.all(8.0),  
      child: FlutterLogo(size: 150.0),  
    ),  
  );  
}
```

Output



► **Practical 8 : Aim :** Designing the mobile app to implement the state management.

State management

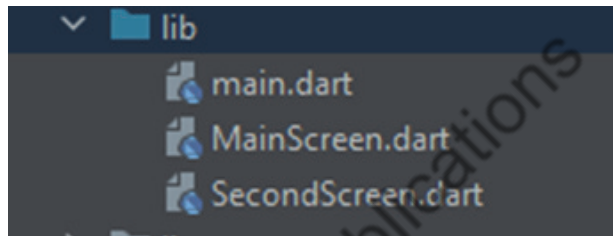
- One of the most crucial and essential processes in the life cycle of an application is managing state.
- So, to put it simply, when a user opens an application, they see one state, and when they click a button, they view another UI screen. The screen is now a state. Consequently, an application is made up of several states, or UI. Navigation and Route are in charge of respective states.
- Everything in memory at the time a program is running constitutes its state. These contain all of the data, including graphics, UI information, animation state, and asset information. Most user states are managed by the framework. Ephemeral state and app state are two different categories of the state that we control ourselves.
- Based on how long a given state persists in an application, state management can be split into two groups.
 - Ephemeral –a single page, like the current rating of a product, or a few seconds like the condition of an animation at the moment.StatefulWidget in Flutter provides support for it.
 - app state –The App state or Shared state refers to states that the user desires to share across numerous application components.

Example

1. Code: main.dart

```
import 'package:flutter/material.dart';
import 'mainscreen.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
// This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ),
      home: MainScreen(),
    );
  }
}
```

- Here we have created a StatelessWidget because we didn't want to make a dynamic application. If you want to make a dynamic application, then use StatefulWidget.
- We have added the theme, which makes the primary color of the application blue. The blue color is the default color of flutter, but we will give a green color here. And we have set the home as mainScreen(). This means that the 1st screen that will be visible to the user will open the UI of the MAIN SCREEN.
- To create mainScreen and SecondScreen right click on -> lib folder then click on -> new -> dart file -> then set name as mainScreen -> then press Enter.(follow the same process to create SecondScreen file.) after creation of this two files user will get following structure.



2. Code : mainScreen.dart

```
import 'package:flutter/material.dart';
import 'secondscreen.dart';
class mainScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("State_Management Example SCREEN 1"),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('SCREEN 2'),
          onPressed: (){
            Navigator.push(context,
              MaterialPageRoute(builder: (context)=>SecondScreen()));
          },
        ),
      );
  }
}
```

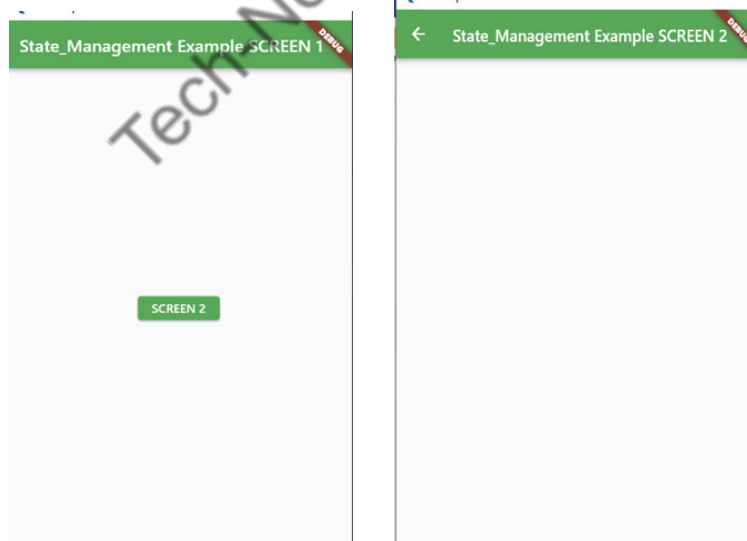
- This code created the 1st screen of our application whose title we have given as State_Management Example SCREEN 1. And in the middle, we had added a ElevatedButton which when clicked will take us to SCREEN 2.
- The line ” MaterialPageRoute(builder: (context)=>SecondScreen())” is mainline that tells our app to move to screen 2 when the button is pressed. Now create a new dart file named Secondscreen.dart . Creating this file will help us to import this in the mainscreen.dart.

3. Code:SecondScreen.dart

```
import 'package:flutter/material.dart';
class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("State_Management Example SCREEN 2"),
      ),
    );
  }
}
```

- In this file, we only have added the title bar that tells you that now you are on the second screen. The flutter itself adds the back icon button. No need to code that. Run this application and press the button in the Centre. It will take you to screen 2.

Output

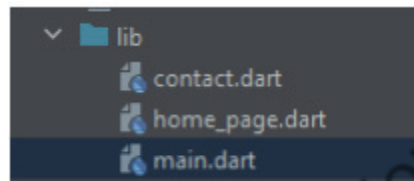


► **Practical 9 : Aim :** Designing the mobile app working with SQLite database.

In this practical we will learn how to use SQLite in Flutter to create, read, update and delete data operations. Here we can perform database operations on ListView With this sqflite we would be able to store data in the mobile local storage.

- **Step 1 :** Create home_page.dart and contact.dart file under lib folder.

Right click on -> lib folder then click on -> new -> dart file -> then set name as home_page -> then press Enter.(follow the same process to create a contact file.) after creation of these two files, the user will get the following structure.



- **Step 2 :** Type command on cmd -> flutter pub add sqflite

```
C:\Users\student\AndroidStudioProjects\database_pract2>flutter pub add sqflite
Resolving dependencies...
  async 2.10.0 (2.11.0 available)
  characters 1.2.1 (1.3.0 available)
  collection 1.17.0 (1.17.2 available)
  js 0.6.5 (0.6.7 available)
  lints 2.0.1 (2.1.1 available)
  matcher 0.12.13 (0.12.16 available)
  material_color_utilities 0.2.0 (0.8.0 available)
  meta 1.8.0 (1.9.1 available)
  path 1.8.2 (1.8.3 available)
  source_span 1.9.1 (1.10.0 available)
+ sqflite 2.2.8+4
+ sqflite_common 2.4.5+1
  stack_trace 1.11.0 (1.11.1 available)
  stream_channel 2.1.1 (2.1.2 available)
+ synchronized 3.1.0
  test_api 0.4.16 (0.6.1 available)
Changed 3 dependencies!

C:\Users\student\AndroidStudioProjects\database_pract2>
```

- **Step 3 :** Write below code in home_page.dart file.

Code : home_page.dart

```
import 'package:database_pract/contact.dart';
import 'package:flutter/material.dart';
class HomePage extends StatefulWidget {
  const HomePage({Key? key}) : super(key: key);
  @override
  State<HomePage> createState() => _HomePageState();
```

```
}  
class _HomePageState extends State<HomePage> {  
  TextEditingController nameController = TextEditingController();  
  TextEditingController contactController = TextEditingController();  
  List<Contact> contacts = List.empty(growable: true);  
  int selectedIndex = -1;  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        centerTitle: true,  
        title: const Text('Contacts List'),  
      ),  
      body: Padding(  
        padding: const EdgeInsets.all(8.0),  
        child: Column(  
          children: [  
            const SizedBox(height: 10),  
            TextField(  
              controller: nameController,  
              decoration: const InputDecoration(  
                hintText: 'Contact Name',  
                border: OutlineInputBorder(  
                  borderRadius: BorderRadius.all(  
                    Radius.circular(10),  
                )),  
            ),  
            const SizedBox(height: 10),  
            TextField(  
              controller: contactController,  
              keyboardType: TextInputType.number,  
              maxLength: 10,  
              decoration: const InputDecoration(  
                hintText: 'Contact Number',  
                border: OutlineInputBorder(  
                  borderRadius: BorderRadius.all(  
                    Radius.circular(10),  
                )),  
            ),  
          ],  
        ),  
      ),  
    );  
  }  
}
```

```
),
const SizedBox(height: 10),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    ElevatedButton(
      onPressed: () {
        //
        String name = nameController.text.trim();
        String contact = contactController.text.trim();
        if (name.isNotEmpty && contact.isNotEmpty) {
          setState(() {
            nameController.text = "";
            contactController.text = "";
            contacts.add(Contact(name: name, contact: contact));
          });
        }
        //
      },
      child: const Text('Save')),
    ElevatedButton(
      onPressed: () {
        //
        String name = nameController.text.trim();
        String contact = contactController.text.trim();
        if (name.isNotEmpty && contact.isNotEmpty) {
          setState(() {
            nameController.text = "";
            contactController.text = "";
            contacts[selectedIndex].name = name;
            contacts[selectedIndex].contact = contact;
            selectedIndex = -1;
          });
        }
        //
      },
      child: const Text('Update')),
  ],
```

```
    ),
    const SizedBox(height: 10),
    contacts.isEmpty
    ? const Text(
      'No Contact yet..',
      style: TextStyle(fontSize: 22),
    )
    : Expanded(
      child: ListView.builder(
        itemCount: contacts.length,
        itemBuilder: (context, index) => getRow(index),
      ),
    )
  ],
),
),
);
}
Widget getRow(int index) {
  return Card(
    child: ListTile(
      leading: CircleAvatar(
        backgroundColor:
        index % 2 == 0 ? Colors.deepPurpleAccent : Colors.purple,
        foregroundColor: Colors.white,
      child: Text(
        contacts[index].name[0],
        style: const TextStyle(fontWeight: FontWeight.bold),
      ),
    ),
    title: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          contacts[index].name,
          style: const TextStyle(fontWeight: FontWeight.bold),
        ),
        Text(contacts[index].contact),
      ],
    ),
  ),
),
);
}
```



```
    ],
  ),
  trailing: SizedBox(
    width: 70,
    child: Row(
      children: [
        InkWell(
          onTap: () {
            //
            nameController.text = contacts[index].name;
            contactController.text = contacts[index].contact;
            setState(() {
              selectedIndex = index;
            });
            //
          },
          child: const Icon(Icons.edit)),
        InkWell(
          onTap: () {
            //
            setState(() {
              contacts.removeAt(index);
            });
            //
          },
          child: const Icon(Icons.delete)),
      ],
    ),
  ),
);
} }
```

► **Step 4 :** Write below code in contact.dart file

Code : contact.dart

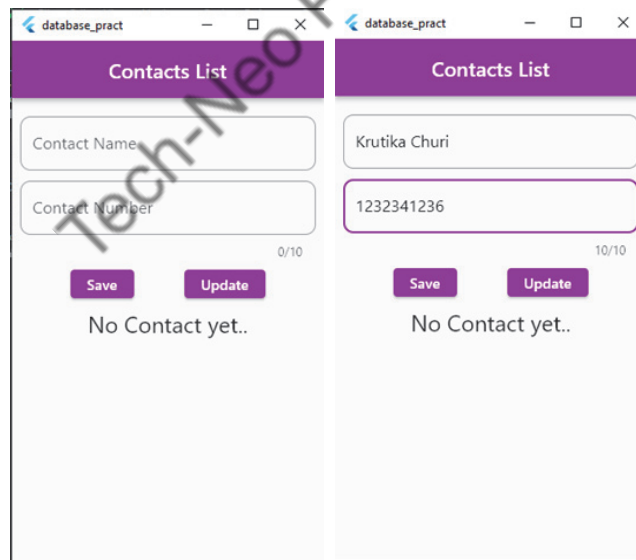
```
class Contact {
  String name;
  String contact;
  Contact({required this.name, required this.contact});
}
```

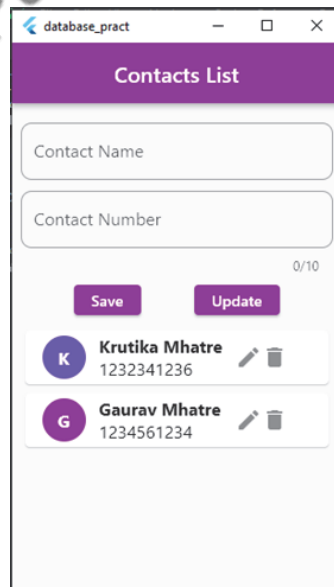
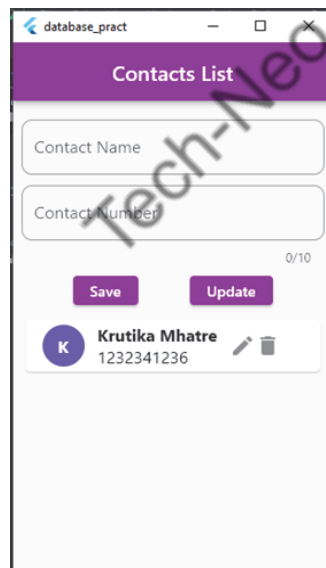
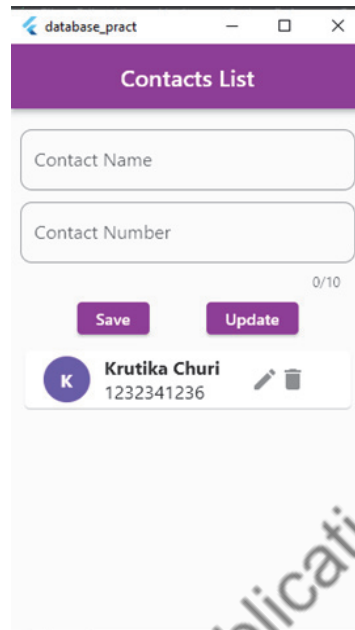
► **Step 5 :** Write below code in main.dart file.

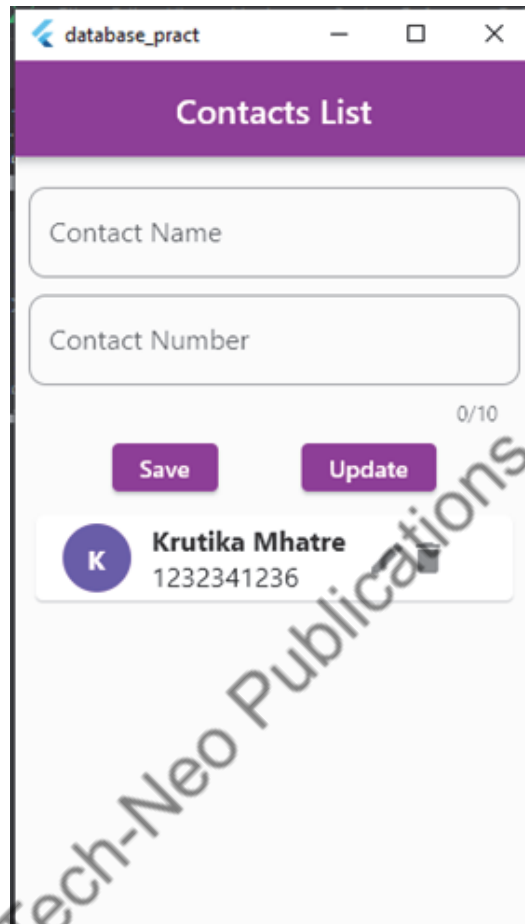
Code : main.dart

```
import 'package:database_pract/home_page.dart';
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp()); }
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Contacts List',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.purple,
      ),
      home: const HomePage(), );
  } }
```

Output







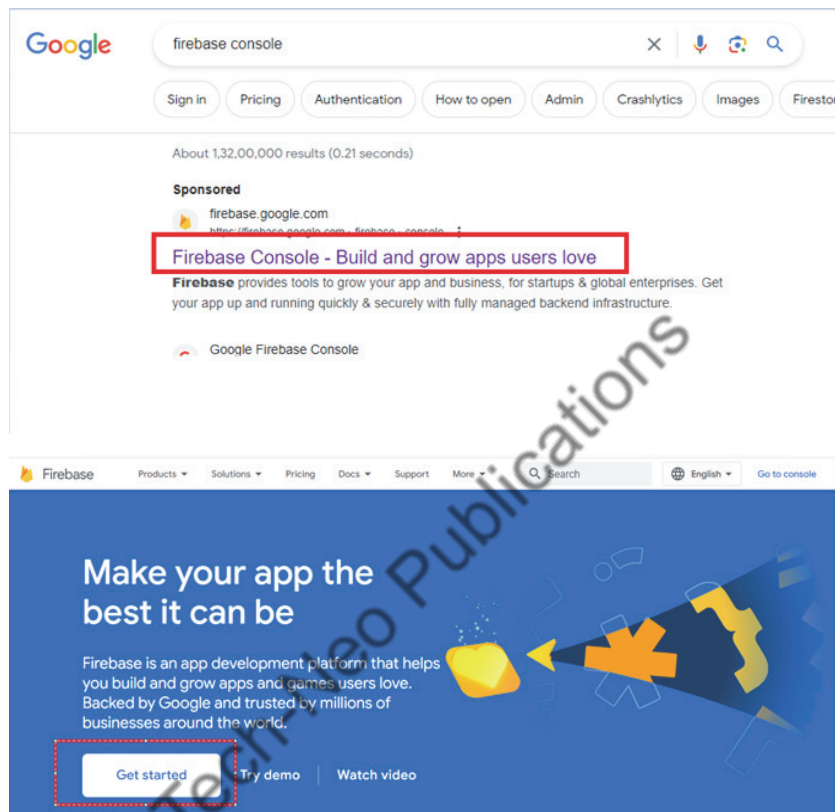
► **Practical 10 : Aim :** Designing the mobile app working with Firebase.

Firestore

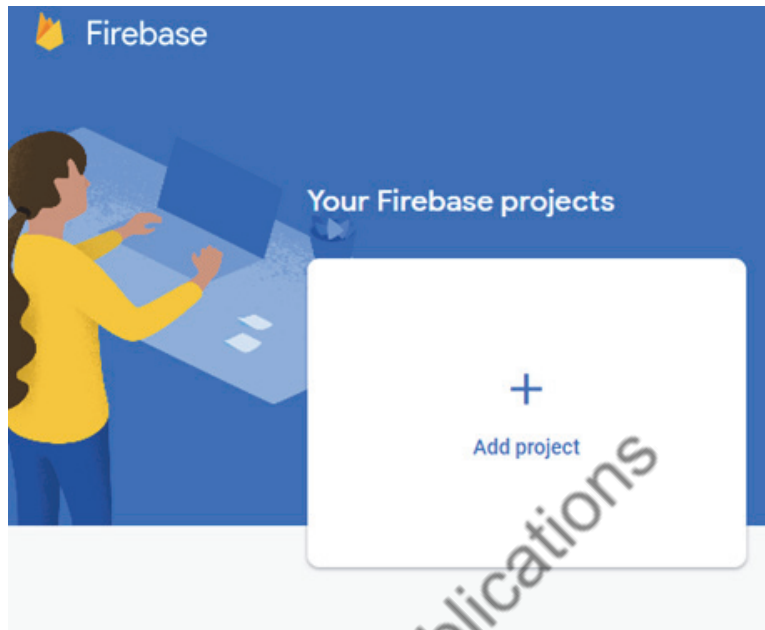
- Google's Firestore is a tool that makes it simple for developers to create, maintain, and expand their apps.
- It makes it easier for developers to create apps more quickly and securely.
- Because there is no programming required on the firestore side, it is simple to use the features more effectively. It offers services to web, unity, android, and ios.
- It offers cloud storage. The database used for data storage is a NoSQL one.

Step by Step Implementation

- ▶ **Step 1 :** First, you have to visit the Firebase console. Now let's move to the next step. Click on the "Add project" as shown in the below image.



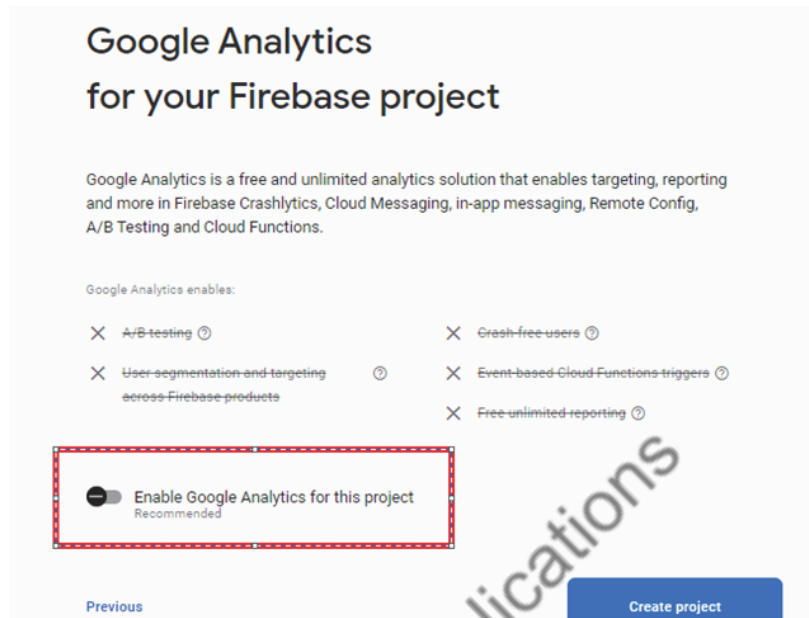
Google sign-In is required to get started.



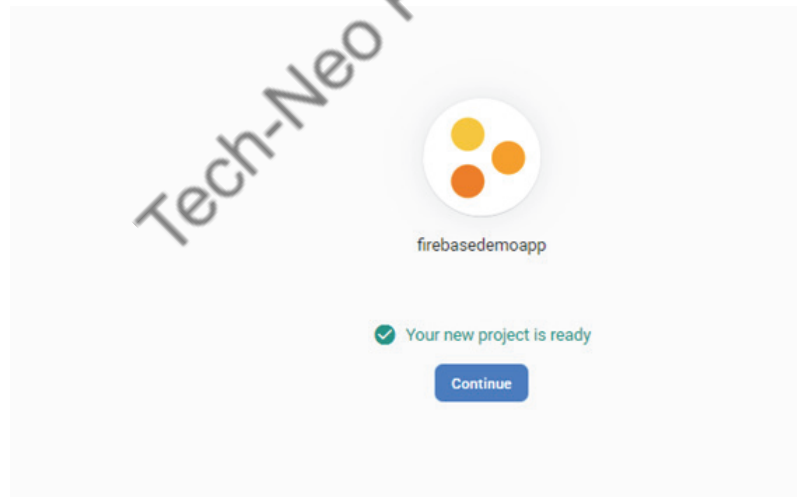
Click on Add project.



Assign the name for your project and click on continue.

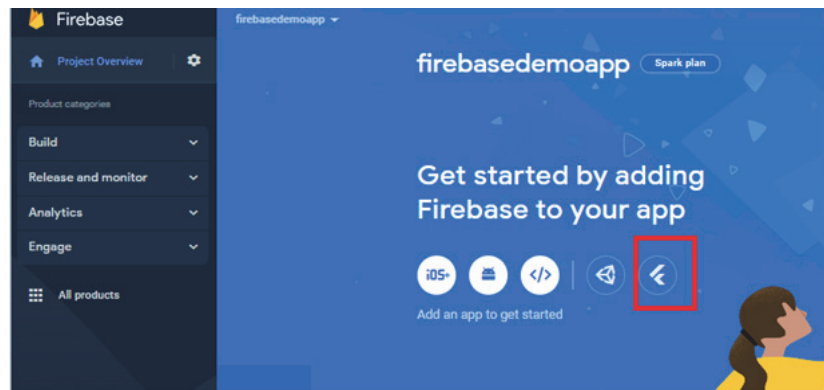


Disable the highlighted option as shown in above screenshot and click on Create project.



Click on Continue.

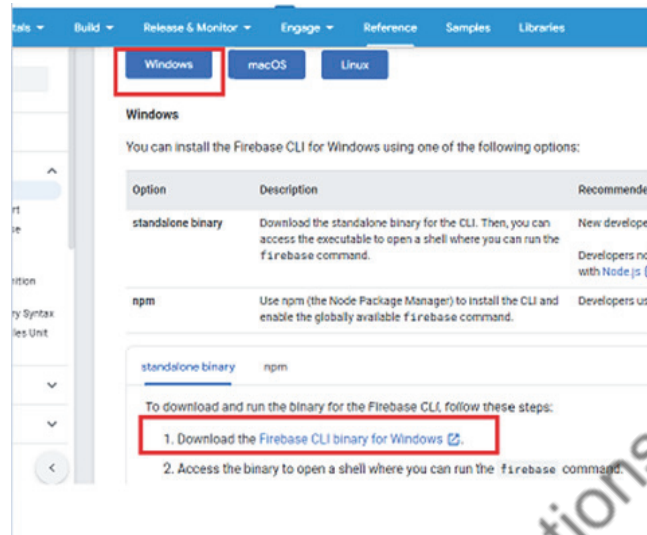
- ▶ **Step 2 :** Now there will be a screen, You can find the Flutter button and click on it as shown in the below image.



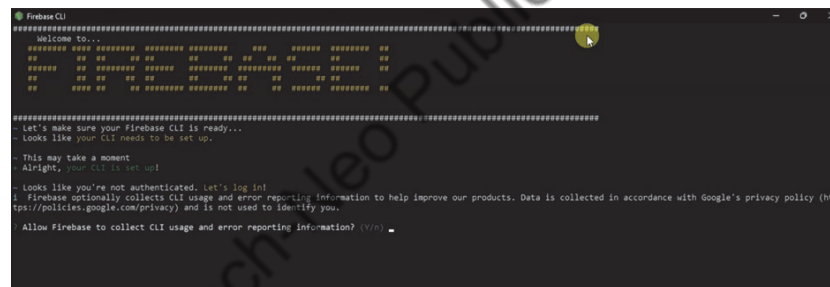
Now time to add firebase to your Flutter App.



Now click on Firebase CLI then you will redirect to the following page select windows and download Firebase CLI for windows.



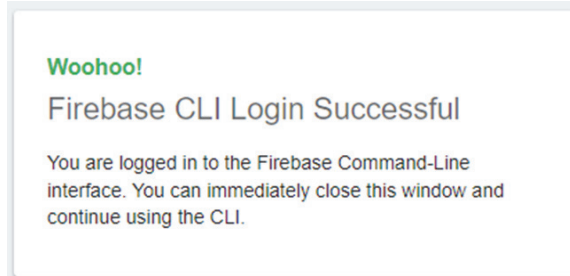
► **Step 3 :** Now go to downloads and click on downloaded Firebase CLI library



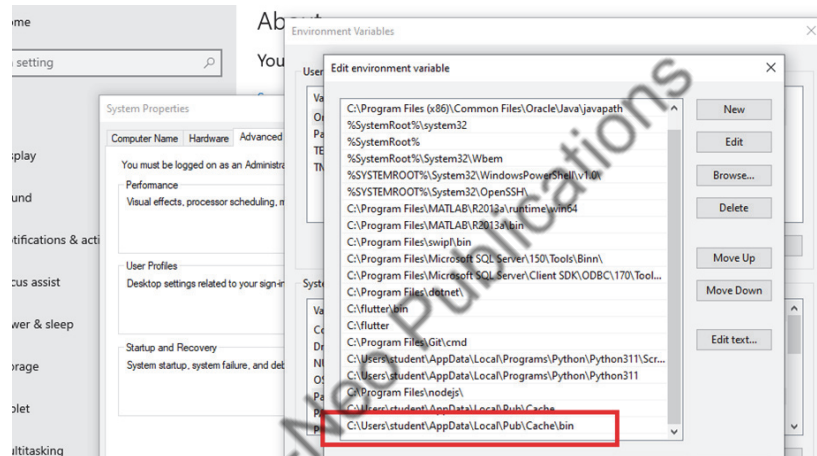
On firebase CLI you will get a message: Allow firebase to collect CLI usage and error reporting information? (Y/N)

Then type Y and press enter.

Then windows security alert box will come, click on allow access and at the same time firebase CLI login is required to provide it, after login you will get the following message.

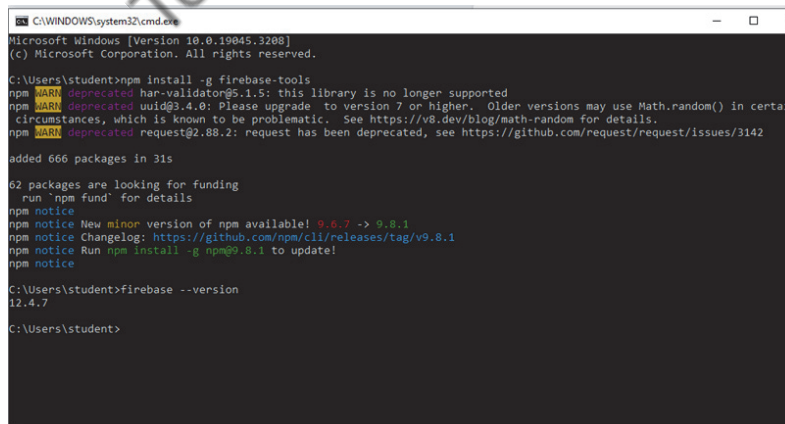


► **Step 4 :** Then install nodejs software and add the following highlighted path to the path variable in the environment variable.



Click on OK

Open cmd and type command: `npm install -g firebase-tools`



► **Step 5 :** Again open cmd and type: dart pub global activate flutterfire_cli

```
Command Prompt - dart pub global activate flutterfire_cli
Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\student>dart pub global activate flutterfire_cli

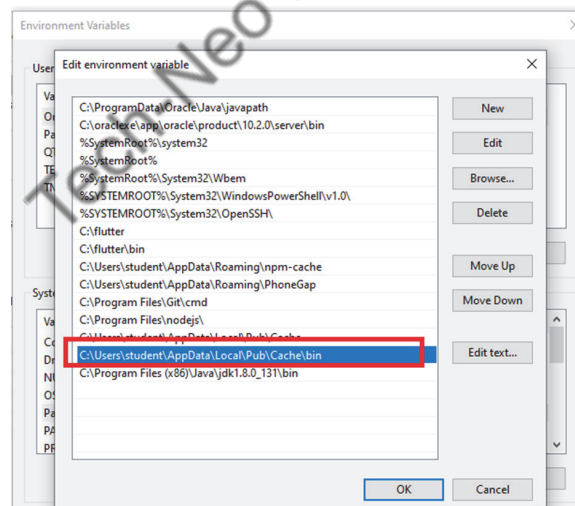
The Dart tool uses Google Analytics to report feature usage statistics
and to send basic crash reports. This data is used to help improve the
Dart platform and tools over time.

To disable reporting of analytics, run:

  dart --disable-analytics

+ ansi_styles 0.3.2+1s... (2.1s)
+ args 2.4.2
+ async 2.11.0
+ boolean_selector 2.1.1
+ characters 1.3.0
+ ci 0.1.0
+ cli_util 0.3.5 (0.4.0 available)
+ clock 1.1.1
+ collection 1.18.0
+ dart_console 1.1.2 (1.2.0 available)
+ deep_pick 0.10.0 (1.0.0 available)
+ ffi 2.0.2
+ file 6.1.4 (7.0.0 available)
+ flutterfire_cli 0.2.7
+ http 0.13.6 (1.1.0 available)
```

Now add below highlighted path to path variable in environment variable as shown below.



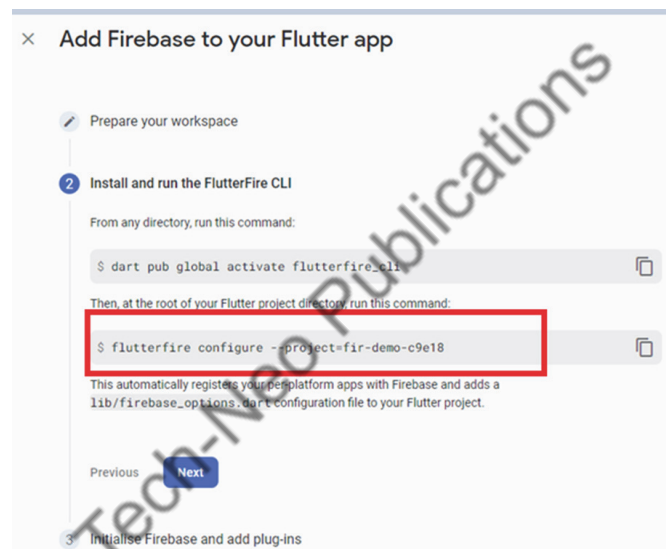
Open cmd and again type: dart pub global activate flutterfire_cli

```
Command Prompt - dart pub global activate flutterfire_cli
Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\student>dart pub global activate flutterfire_cli
Package flutterfire_cli is currently active at version 0.2.7.
The package flutterfire_cli is already activated at newest available version.
To recompile executables, first run `dart pub global deactivate flutterfire_cli`.
Installed executable flutterfire.
Activated flutterfire_cli 0.2.7.

C:\Users\student>
```

► **Step 6 :** Now register your platform app with firebase



Now copy the above highlighted command of your project and run on cmd as shown below.

```
Command Prompt - flutterfire configure --project=fir-demo-c9e18
Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\student>cd C:\Users\student\AndroidStudioProjects\firebaseedemo
C:\Users\student\AndroidStudioProjects\firebaseedemo>flutterfire configure --project=fir-demo-c9e18
! Found 2 Firebase projects. Selecting project fir-demo-c9e18.
? Which platforms should your configuration support (use arrow keys & space to select)? >
  android
  ios
  macos
  web
```

Select android press enter.

```
Command Prompt
Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\student>cd C:\Users\student\AndroidStudioProjects\firebasedemo

C:\Users\student\AndroidStudioProjects\firebasedemo>flutterfire configure --project=fir-demo-c9e18
Found 2 Firebase projects. Selecting project fir-demo-c9e18.
Which platforms should your configuration support (use arrow keys & space to select)? · ios, macos, web
Firebase ios app com.example.firebasedemo is not registered on Firebase project fir-demo-c9e18.
Registered a new Firebase ios app on Firebase project fir-demo-c9e18.
Firebase macos app com.example.firebasedemo registered.
Firebase web app firebasedemo (web) is not registered on Firebase project fir-demo-c9e18.
Registered a new Firebase web app on Firebase project fir-demo-c9e18.

Firebase configuration file lib\firebase_options.dart generated successfully with the following Firebase apps:

Platform  Firebase App Id
web       1:771579595639:web:89a68170d8e65601c7ef72
ios       1:771579595639:ios:4b8bb95d7463bcd1c7ef72
macos    1:771579595639:ios:4b8bb95d7463bcd1c7ef72

Learn more about using this file and next steps from the documentation:
> https://firebase.google.com/docs/flutter/setup

C:\Users\student\AndroidStudioProjects\firebasedemo>
```

► **Step 7 :** Now copy below highlighted code and paste in main.dart file in android studio.

× Add Firebase to your Flutter app

- Prepare your workspace
- Install and run the FlutterFire CLI
- 3** Initialise Firebase and add plug-ins

To initialise Firebase, call `Firebase.initializeApp` from the `firebase_core` package with the configuration from your new `firebase_options.dart` file:

```
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

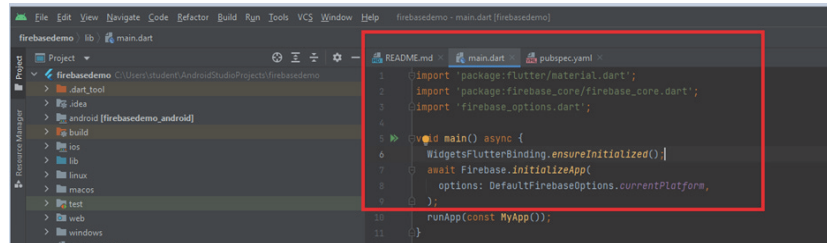
// ...

await Firebase.initializeApp(
  options: DefaultFirebaseOptions.currentPlatform,
);
```

Then, add and begin using the [Flutter plug-ins](#) for the Firebase products that you'd like to use.

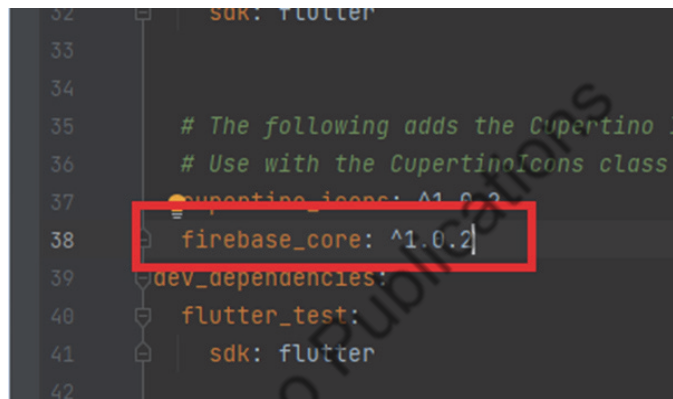
Note: If you're using Analytics or performance monitoring, you may need to follow a few additional set-up steps.

Previous [Continue to the console](#)



```
1 import 'package:flutter/material.dart';
2 import 'package:firebase_core/firebase_core.dart';
3 import 'firebase_options.dart';
4
5 void main() async {
6   WidgetsFlutterBinding.ensureInitialized();
7   await Firebase.initializeApp(
8     options: DefaultFirebaseOptions.currentPlatform,
9   );
10  runApp(const MyApp());
11 }
```

Open pubspec.yaml and add firebase_core: ^1.0.2 as shown below and click on pub get.



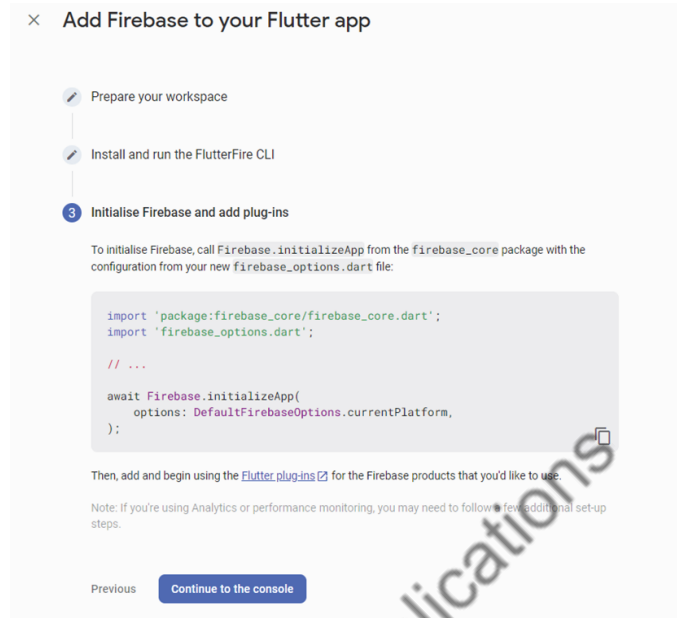
```
32  sdk: flutter
33
34
35  # The following adds the Cupertino Icons class
36  # Use with the CupertinoIcons class
37  cupertino_icons: ^1.0.2
38  firebase_core: ^1.0.2
39  dev_dependencies:
40    flutter_test:
41      sdk: flutter
42
```

► **Step 8:** In terminal section of android studio Type following commands:

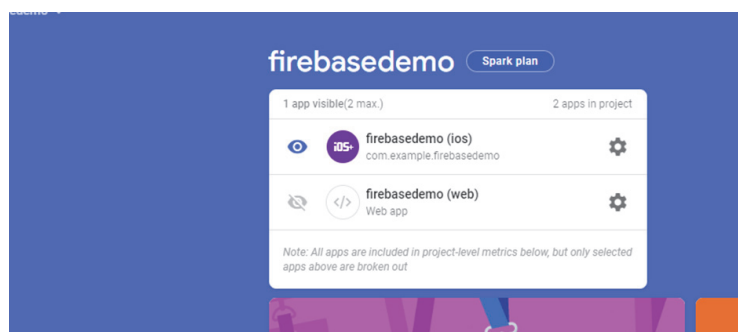
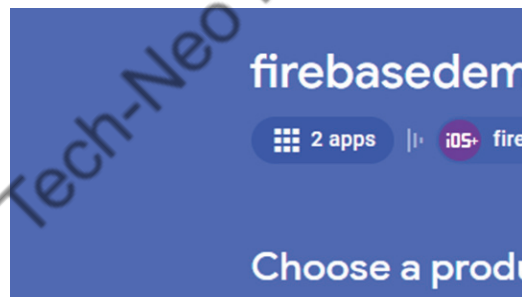
1. flutter pub add firebase_core
2. flutterfire configure

👉 **Run your flutter project at the end.**

Now in firebase console click on continue to the console as shown below:



Then refresh this page you will see here iOS Platform has been added to your project.



► **Practical 11 : Aim :** Designing the mobile app to develop a calculator.

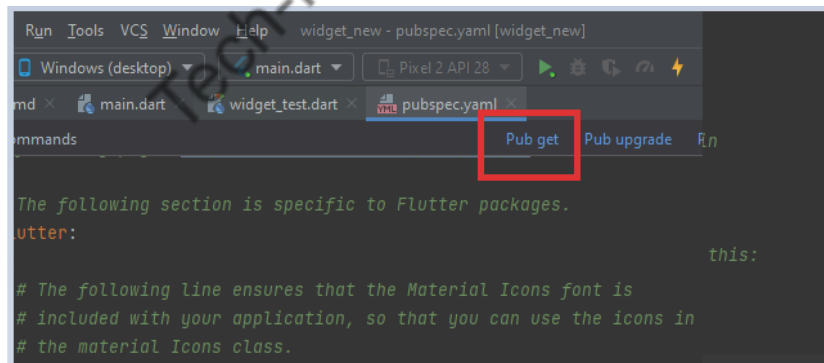
In this Practical we will build a Simple Calculator App that can perform basic arithmetic operations like addition, subtraction, multiplication or division depending upon the user input.

► **Step 1 :** Adding dependencies in pubspec.yaml file

Add math_expressions: ^2.0.0 package in pubspec.yaml file as shown in below screen.

```
28 # the latest version available on pub.dev. To see which dep
29 # versions available, run `flutter pub outdated`.
30 dependencies:
31   flutter:
32     sdk: flutter
33
34
35 # The following adds the Cupertino Icons font to your app
36 # Use with the CupertinoIcons class for iOS style icons.
37 cupertino_icons: ^1.0.2
38 math_expressions: ^2.0.0
39 dev_dependencies:
40   flutter_test:
41     sdk: flutter
42
```

► **Step 2 :** After adding this package click on Pub get.



If everything is OK you will get following message in terminal:


```
Document 1/
Messages: [calculator_app] Flutter x
C:\flutter\bin\flutter.bat --no-color pub get
Resolving dependencies...
  collection 1.17.1 (1.18.0 available)
  matcher 0.12.15 (0.12.16 available)
  material_color_utilities 0.2.0 (0.8.0 available)
  source_span 1.9.1 (1.10.0 available)
  stack_trace 1.11.0 (1.11.1 available)
  stream_channel 2.1.1 (2.1.2 available)
  test_api 0.5.1 (0.6.1 available)
Got dependencies!
Process finished with exit code 0
```

- ▶ **Step 3 :** In the Lib folder, there is a main.dart file already present. And now in the same folder create a new file named **buttons.dart** and write below code inside it.

Code: buttons.dart

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

// creating Stateless Widget for buttons
class MyButton extends StatelessWidget {

  // declaring variables
  final color;
  final textColor;
  final String buttonText;
  final buttontapped;

  //Constructor
  MyButton({this.color, this.textColor, required this.buttonText,
  this.buttontapped});

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: buttontapped,
```

```
child: Padding(
  padding: const EdgeInsets.all(0.2),
  child: ClipRRect(
    // borderRadius: BorderRadius.circular(25),
    child: Container(
      color: color,
      child: Center(
        child: Text(
          buttonText,
          style: TextStyle(
            color: textColor,
            fontSize: 25,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ),
  ),
);
}
```

► **Step 4 :** Write below code in main.dart file.

Code: main.dart

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: HomePage(),
    ); // MaterialApp
  }
}
```

```
}  
}  
  
class HomePage extends StatefulWidget {  
  @override  
  _HomePageState createState() => _HomePageState();  
}  
  
class _HomePageState extends State<HomePage> {  
  var userInput = "";  
  var answer = "";  
  
  // Array of button  
  final List<String> buttons = [  
    'C',  
    '+/-' ,  
    '%',  
    'DEL',  
    '7',  
    '8',  
    '9',  
    '/',  
    '4',  
    '5',  
    '6',  
    'x',  
    '1',  
    '2',  
    '3',  
    '-',  
    '0',  
    '.',  
    '=',  
    '+',  
  ];  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: new AppBar(  

```

```
        title: new Text("Calculator"),
      ), //AppBar
      backgroundColor: Colors.white38,
      body: Column(
        children: <Widget>[
          Expanded(
            child: Container(
              child: Column(
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                children: <Widget>[
                  Container(
                    padding: EdgeInsets.all(20),
                    alignment: Alignment.centerRight,
                    child: Text(
                      userInput,
                      style: TextStyle(fontSize: 18, color: Colors.white),
                    ),
                  ),
                  Container(
                    padding: EdgeInsets.all(15),
                    alignment: Alignment.centerRight,
                    child: Text(
                      answer,
                      style: TextStyle(
                        fontSize: 30,
                        color: Colors.white,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                  ),
                ],
              ),
            ),
          ),
          Expanded(
            flex: 3,
            child: Container(
              child: GridView.builder(
                itemCount: buttons.length,
                gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                  crossAxisCount: 4,
                  itemBuilder: (BuildContext context, int index) {
```

```
// Clear Button
if (index == 0) {
  return MyButton(
    buttontapped: () {
      setState() {
        userInput = "";
        answer = '0';
      }
    },
    buttonText: buttons[index],
    color: Colors.blue[50],
    textColor: Colors.black,
  );
}

// +/- button
else if (index == 1) {
  return MyButton(
    buttonText: buttons[index],
    color: Colors.blue[50],
    textColor: Colors.black,
  );
}

// % Button
else if (index == 2) {
  return MyButton(
    buttontapped: () {
      setState() {
        userInput += buttons[index];
      }
    },
    buttonText: buttons[index],
    color: Colors.blue[50],
    textColor: Colors.black,
  );
}

// Delete Button
else if (index == 3) {
  return MyButton(
    buttontapped: () {
```

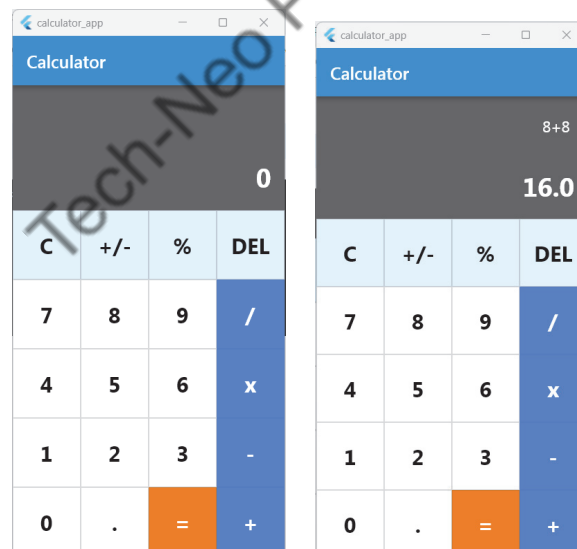
```
        setState() {
            userInput =
                userInput.substring(0, userInput.length - 1);
        });
    },
    buttonText: buttons[index],
    color: Colors.blue[50],
    textColor: Colors.black,
);
}
// Equal_to Button
else if (index == 18) {
    return MyButton(
        buttontapped: () {
            setState() {
                equalPressed();
            });
        },
        buttonText: buttons[index],
        color: Colors.orange[700],
        textColor: Colors.white,
    );
}
// other buttons
else {
    return MyButton(
        buttontapped: () {
            setState() {
                userInput += buttons[index];
            });
        },
        buttonText: buttons[index],
        color: isOperator(buttons[index])
            ? Colors.blueAccent
            : Colors.white,
        textColor: isOperator(buttons[index])
            ? Colors.white
            : Colors.black,
    );
}
}), // GridView.builder
```

```

    ),    ),
    ],
    ),
); }
bool isOperator(String x) {
    if (x == '/' || x == 'x' || x == '-' || x == '+' || x == '=') {
        return true; }
    return false;
}
// function to calculate the input operation
void equalPressed() {
    String finaluserinput = userInput;
    finaluserinput = userInput.replaceAll('x', '*');
    Parser p = Parser();
    Expression exp = p.parse(finaluserinput);
    ContextModel cm = ContextModel();
    double eval = exp.evaluate(EvaluationType.REAL, cm);
    answer = eval.toString();
}}

```

Output



Lab Manual Ends...

