Mobile Programming Practical

Practical No. 01

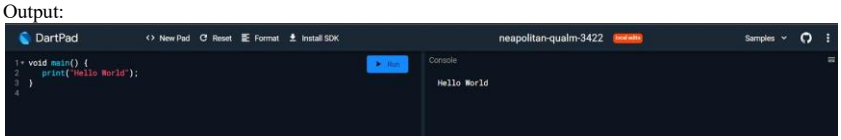
Aim: Program to demonstrate the features of Dart language.

Description:

Simple Dart program to print Hello World−

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

Output:



Variables in DART

Variables are containers used to store value in the program. There are different types of variables where you can keep different kinds of values. Here is an example of creating a variable and initializing it.

```
// here variable name contains value Meena.
var name = "Meena";
```

Variable Types

They are called data types.

String: For storing text value. E.g. “John” [Must be in quotes] **int:** For storing integer value. E.g. 10, -10, 8555 [Decimal is not included] **double:** For storing floating point values. E.g. 10.0, -10.2 , 85.698 [Decimal is included] **num:** For storing any type of number. E.g. 10, 20.2, -20 [both int and double] **bool:** For storing true or false. E.g. true, false [Only stores true or false values] **var:** For storing any value. E.g. ‘Bimal’, 12, ‘z’, true

Rules For Creating Variables In Dart

- Variable names are case sensitive, i.e., a and A are different.
- A variable name can consist of letters and alphabets.
- A variable name cannot start with a number.
- Keywords are not allowed to be used as a variable name.
- Blank spaces are not allowed in a variable name.

- Special characters are not allowed except for the underscore (_) and the dollar ($) sign.

Dart Constant

Constant is the type of variable whose value never changes. In programming, changeable values are mutable and unchangeable values are immutable. Sometimes, you don't need to change the value once declared. Like the value of PI=3.14, it never changes. To create a constant in Dart, you can use the const keyword.

```
void main(){  const pi =
3.14;  pi = 4.23; // not
possible  print("Value of
PI is $pi");
}
```

DATA TYPES IN DART

Data types help you to categorize all the different types of data you use in your code. For e.g. numbers, texts, symbols, etc. The data type specifies what type of value will be stored by the variable. Each variable has its data type. Dart supports the following built-in data types :

1. Numbers
2. Strings
3. Booleans
4. Lists
5. Maps

Data Type	**Keyword**	**Description**
Numbers	int, double, num	It represents numeric values
Strings	String	It represents a sequence of characters
Booleans	bool	It represents Boolean values true and false
Lists	List	It is an ordered group of items
Maps	Map	It represents a set of values as key-value pairs

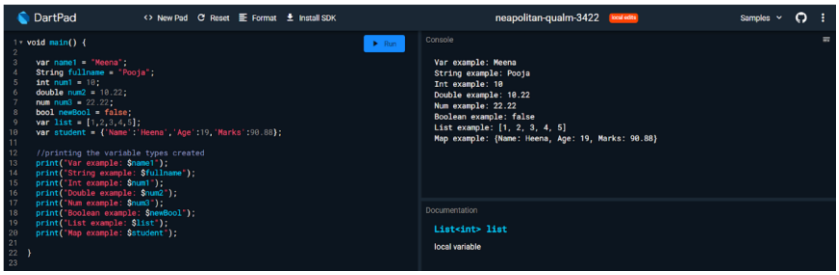
Lists and Maps − It is used to represent a collection of objects. A simple List can be defined as below −.

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

The list shown above produces [1,2,3,4,5] list.
(Also add the functions you had used for the List during practical session you can check it on https://www.javatpoint.com/dart-lists**)**

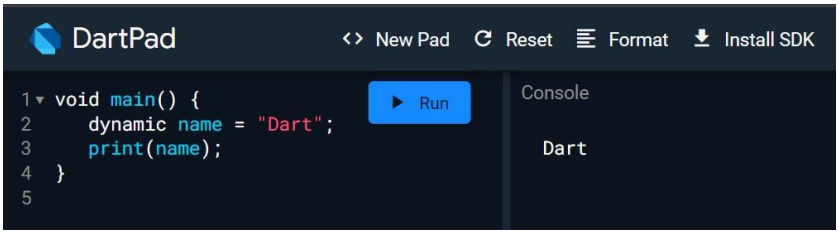
Map can be defined as shown here −

```
void main() {  var student =
  {'Name':'Heena','Age':19,'Marks':90.88};
  print(student);
}
```



Additional Map related methods - https://www.javatpoint.com/dart-map
Dynamic − If the variable type is not defined, then its default type is dynamic. The following example illustrates the dynamic type variable −

```
void main() {  dynamic
  name = "Dart";
  print(name);
}
```



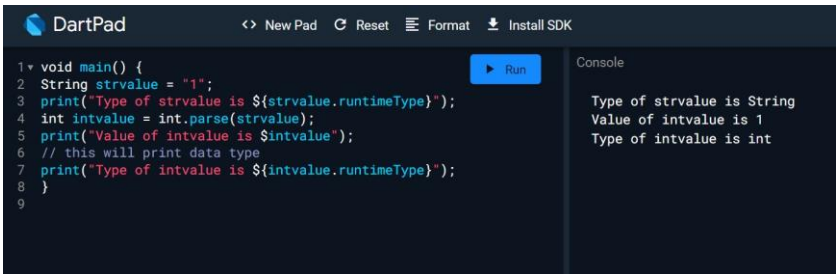
Type Conversion In Dart

In dart, type conversion allows you to convert one data type to another type. For e.g. to convert String to int, int to String or String to bool, etc.

Convert String To Int In dart

You can convert String to int using int.parse() method. The method takes String as an argument and converts it into an integer.

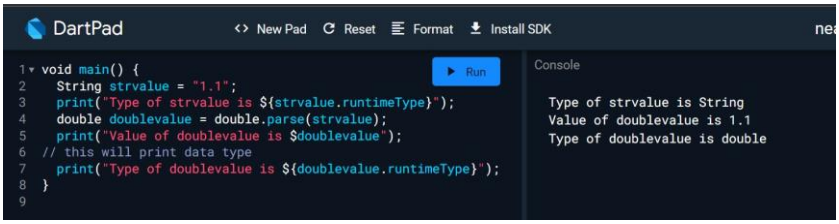
```
void main() {  String strvalue = "1";  print("Type
of strvalue is ${strvalue.runtimeType}");  int
intvalue = int.parse(strvalue);  print("Value of
intvalue is $intvalue");
// this will print data type  print("Type of intvalue is
${intvalue.runtimeType}");
}
```



Convert String To Double In Dart

You can convert String to double using double.parse() method. The method takes String as an argument and converts it into a double.

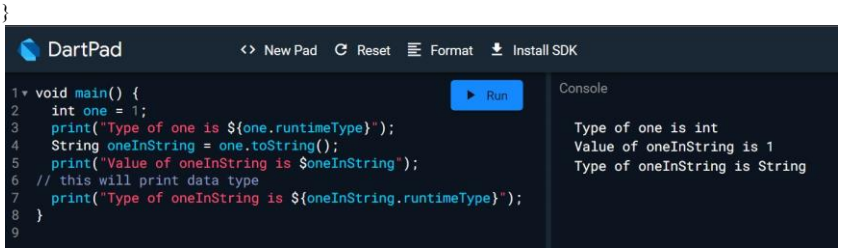
```
void main() {  String strvalue = "1.1";  print("Type
of strvalue is ${strvalue.runtimeType}");  double
doublevalue = double.parse(strvalue);  print("Value
of doublevalue is $doublevalue");
// this will print data type  print("Type of doublevalue is
${doublevalue.runtimeType}");
}
```



Convert Int To String In Dart

You can convert int to String using the toString() method. Here is example:

```
void main() {
int one = 1;
print("Type of one is ${one.runtimeType}");
String oneInString = one.toString();
print("Value of oneInString is $oneInString");
// this will print data type  print("Type of oneInString is
${oneInString.runtimeType}");
}
```



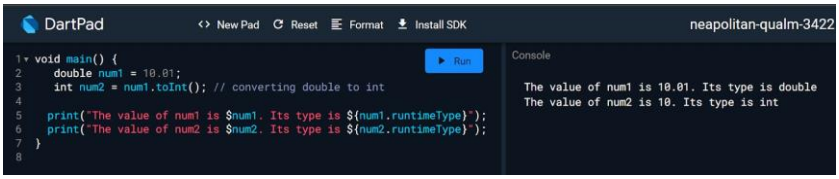
Convert Double To Int In Dart

You can convert double to int using the toInt() method.

```
void main() {
  double num1 = 10.01;  int num2 = num1.toInt();
  // converting double to int

print("The value of num1 is $num1. Its type is ${num1.runtimeType}");
print("The value of num2 is $num2. Its type is ${num2.runtimeType}");
}
```

Created by - Pooja D. Amin



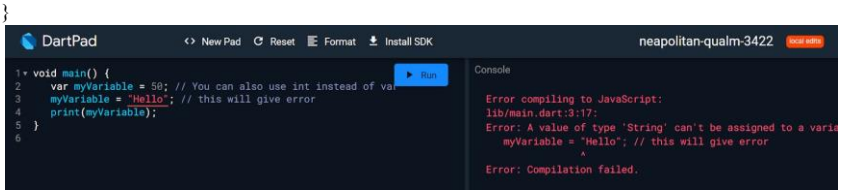
Optionally Typed Language

You may have heard of the statically-typed language. It means the data type of variables is known at compile time. Similarly, dynamically-typed language means data types of variables are known at run time. Dart supports dynamic and static types, so it is called optionally-typed language.

Statically Typed

A language is statically typed if the data type of variables is known at compile time. Its main advantage is that the compiler can quickly check the issues and detect bugs.

```
void main() {
   var myVariable = 50; // You can also use int instead of var
   myVariable = "Hello"; // this will give error
   print(myVariable);
}
```



Dynamically Typed Example

A language is dynamically typed if the data type of variables is known at run time.

```
void main() {
   dynamic myVariable = 50;
   myVariable = "Hello";
   print(myVariable);
}
```

Difference Between var and dynamic type in Dart

Use var if you expect a variable assignment to change during its lifetime:

```
var msg = "Hello world.";
msg = "Hello world again.";
```

You can change the type of x but not a

```
void main() {
dynamic x = 'hal';
x = 123;  print(x);
var a = 'hal';
a = 123;
print(a);
}
```

Note:
dynamic: can change TYPE of the variable, & can change VALUE of the variable later in code. **var:** can't change TYPE of the variable, but can change the VALUE of the variable later in code.

Decision Making and Loops

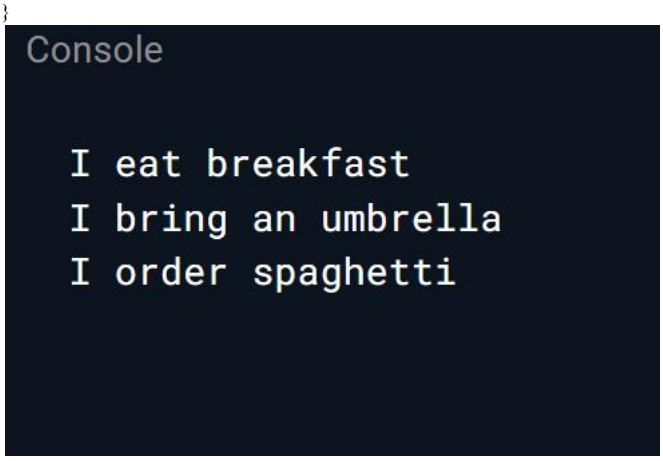
A decision making block evaluates a condition before the instructions are executed. Dart supports If, If..else and switch statements.

```
void main() {
```

```
 var wakeup = "hungry";
var leavehouse = "cloudy";
var foodorder = "pasta";
//Simple if statement  if
(wakeup == "hungry") {
    print("I eat breakfast");
}

 //if else statement  if
(leavehouse == "cloudy") {
print("I bring an umbrella");
}  else {  print("I bring
  sunglasses");
}

 //if elseif else statement  if
(foodorder == "meat") {
    print("I order butter chicken");
}  else if (foodorder == "pasta") {
  print("I order spaghetti ");
}  else {  print("I order a
  salad");
}
}
```



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.

This is the most common type of loop. You can use for loop to run a code block multiple times according to the condition. The syntax of for loop is:

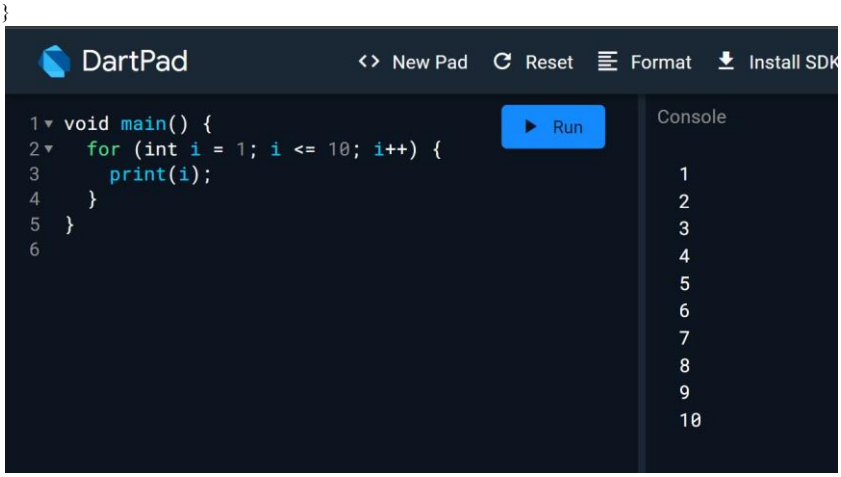
```
for(initialization; condition; increment/decrement){
        statements;
}
```

Initialization is executed (one time) before the execution of the code block.
Condition defines the condition for executing the code block.
Increment/Decrement is executed (every time) after the code block has been executed.

To Print 1 To 10 Using For Loop

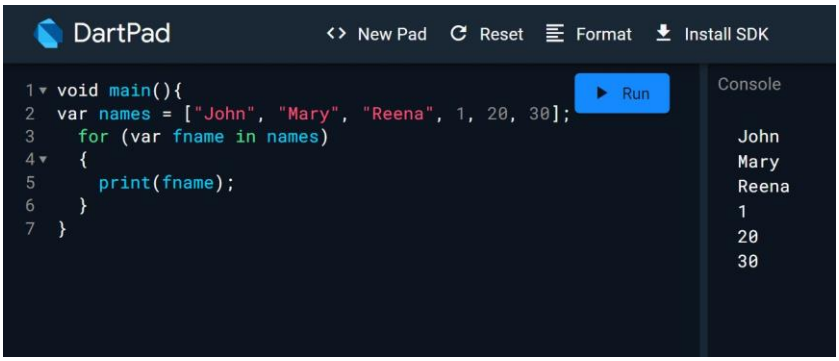
This example prints 1 to 10 using for loop. Here int i = 1; is initialization, i<=10 is condition and i++ is increment/decrement.

```
void main() {
for (int i = 1; i <= 10; i++) {
    print(i);
}
}
```



Example for For in loop

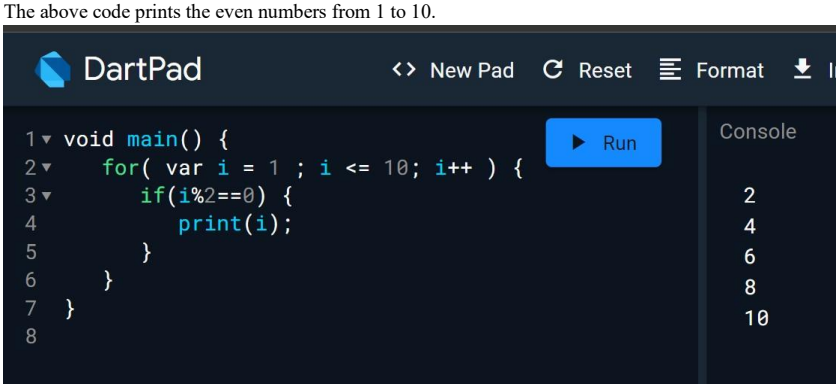
```
var names = ["John", "Mary", "Reena", 1, 20, 30];
for (var fname in names)
{
    print(fname);
}
```



Let us understand another simple example about the usage of control statements and loops −

```
void main() {  for( var i = 1 ; i
  <= 10; i++ ) {
     if(i%2==0) {
       print(i);
     }
   }
}
```

The above code prints the even numbers from 1 to 10.

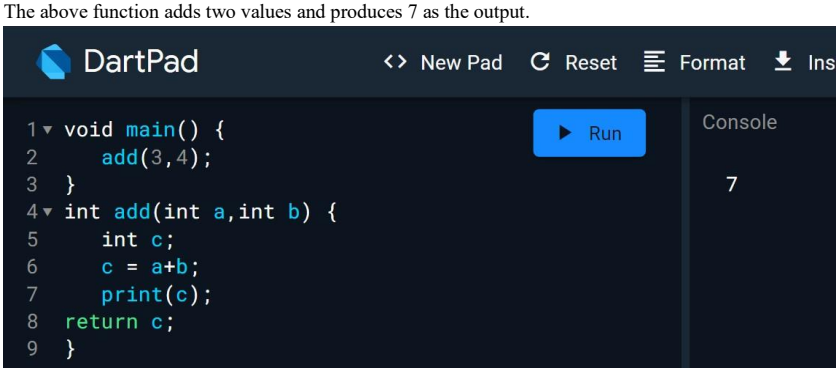


Functions

A function is a group of statements that together performs a specific task. Let us look into a simple function in Dart as shown here −

```
void main() {
  add(3,4);
}
int add(int a,int b) {
  int c;
  c =
  a+b;
  print(c)
  ;
return c;
}
```

The above function adds two values and produces 7 as the output.



Object Oriented Programming

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

A class is a blueprint for creating objects. A class definition includes the following −

- Fields
- Getters and setters
- Constructors
- Functions

Now, let us create a simple class using the above definitions −

```
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
}
```

Practical No. 02

Aim: Designing the mobile app to implement different widgets

Description:

1 . Material App(“ Hello World”)

Material App is a foundational widget in Flutter for building Android-inspired user interfaces. It provides design elements, navigation, and structure for creating visually appealing and cohesive mobile applications.

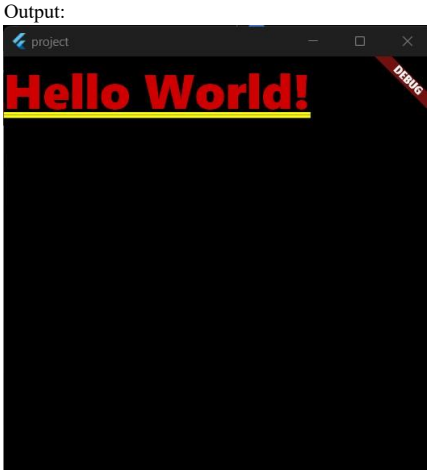
Code:

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



void main        ()  {
 runApp          (
   MaterialApp   (
     home :  Text ( 'Hello World!'        ) ,
   ) , //MaterialApp
 ) ;
}
```

Output:



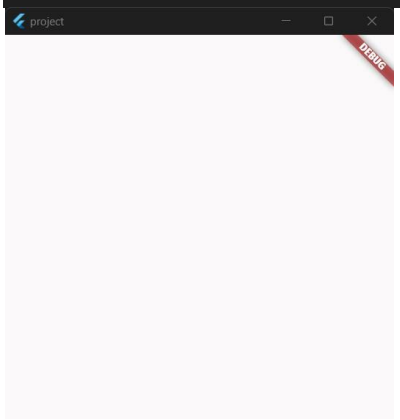
2 . Scaffold

A scaffold in Flutter is a fundamental widget that provides a basic app structure, including an app bar, body content area, and optional floating action button. It serves as a foundational layout for building mobile applications.

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



void main       ()  {
 runApp         (
   MaterialApp  (
     home :  Scaffold    () ,
   ) ,
 ) ;
}
```



3 . AppBar

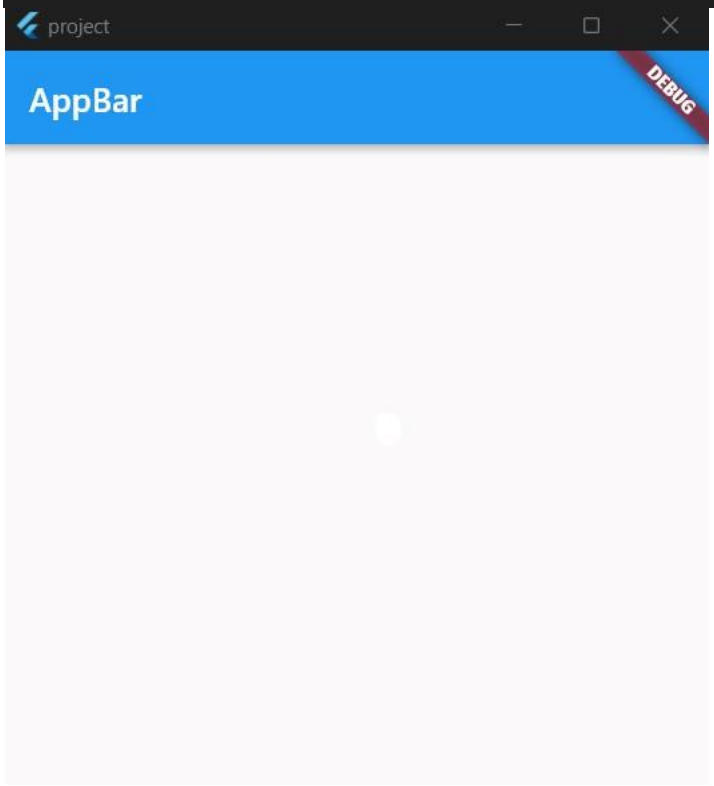
An AppBar in Flutter is a customizable top app bar widget used for navigation, branding, and actions. It typically contains icons, text, or other widgets and is commonly used in mobile app layouts for easy access to app features.

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



void main       ()  {
 runApp         (
```

```
    MaterialApp  (
      home :  Scaffold    (
        appBar :  AppBar  (
          title :  Text ( 'AppBar'         ) ,
        ) ,//AppBar
      ) , //Scaffold
    ) , //MaterialApp
  ) ;
}
```

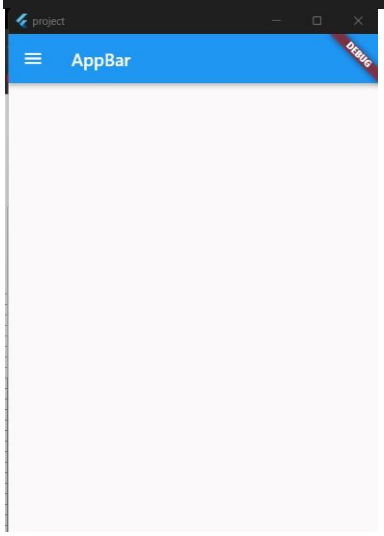


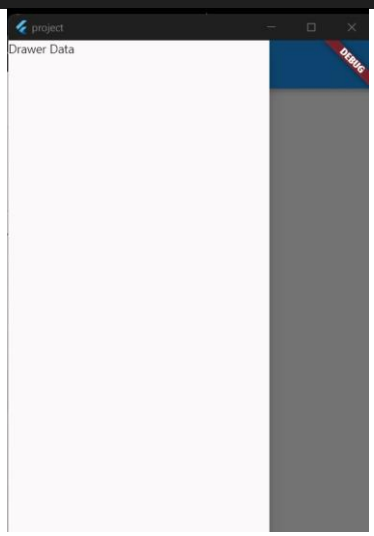
4 . Drawer

A Flutter "Drawer" is a user interface element that slides in from the side, typically used for navigation or displaying additional content options in mobile apps.

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

```dart
void main       ()  {
 runApp         (
   MaterialApp  (
     home :  Scaffold    (
       appBar :  AppBar  (
         title :  Text ( 'AppBar'         ) ,
       ) ,//AppBar
       drawer :  Drawer  (
         child :  Text ( 'Drawer Data'    ) ,
       ) ,//Drawer
     ) ,//Scaffold
   ) ,//MaterialApp
  ) ;
}
```





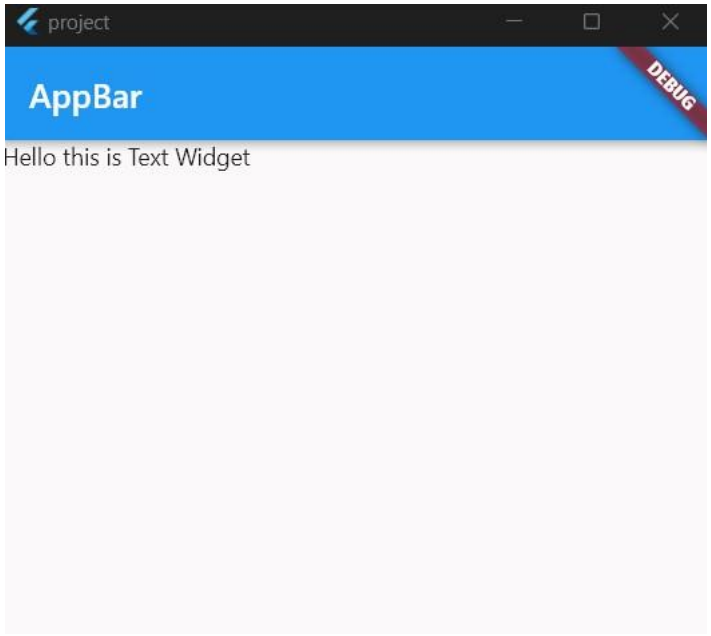
5 . Text

"Text" is a widget used to display simple, styled text on the user interface. It's customizable with various fonts, styles, and alignments, making it essential for app UIs.

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



void main       ()  {
 runApp         (
   MaterialApp  (
     home :  Scaffold    (
       appBar :  AppBar  (
         title :  Text ( 'AppBar'         ) ,
       ) ,//AppBar
       body :  Text ( 'Hello this is Text Widget' ) ,//Text  Widget
     ) ,//Scaffold
   ) ,//MaterialApp
 ) ;
}
```



6 . Image (using URL & local image)

In Flutter, an "Image" widget is used to display static or network images within an app, supporting various formats and customization options for layout and appearance.

Image.network()

"image.network" in Flutter is a widget that loads and displays an image from a network URL, making it easy to show remote images in your app's interface.

[for getting image URL → go to your browser → type any image name in search bar → select a image of your choice → right-click on that image → in the selection dialogue select Copy image address → then paste it in your code]

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



void main       ()  {
 runApp         (
   MaterialApp  (
     home :  Scaffold    (
       appBar :  AppBar  (
         title :  Text ( 'AppBar'         ) ,
       ) ,
body :
Image . network      (  "https://img.freepik.com/free-photo/puppy-that-is-
walking-sn
ow_1340-37228.jpg?q=10&h=200"     ) ,
     ) ,
   ) ,
 ) ;
}
```

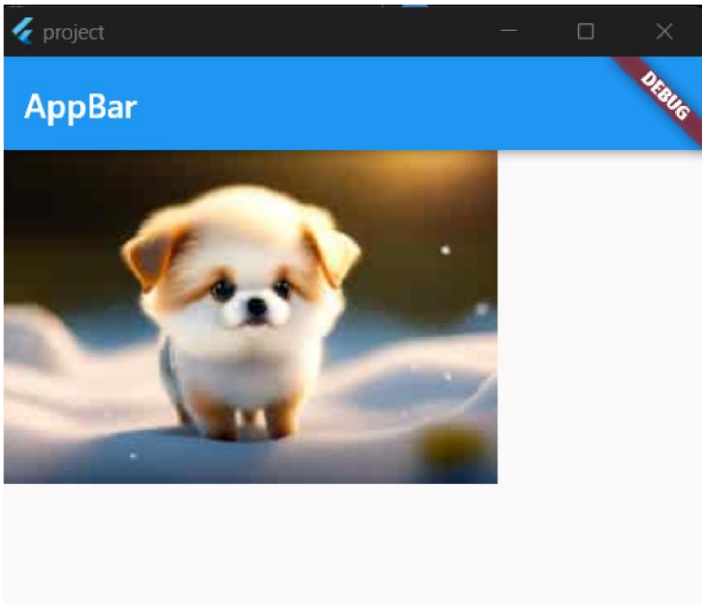
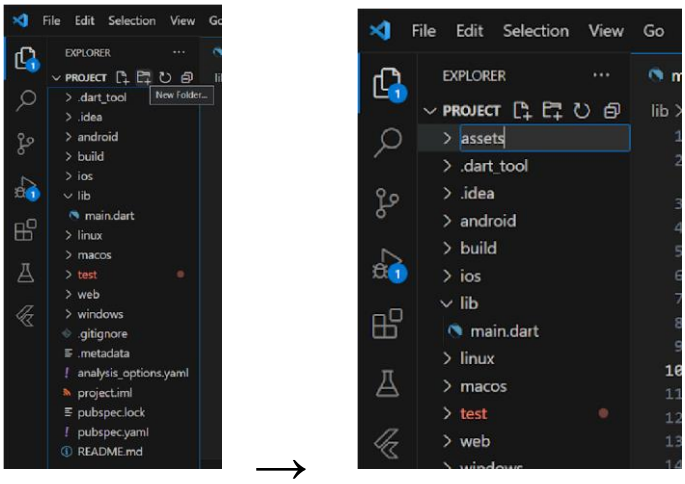
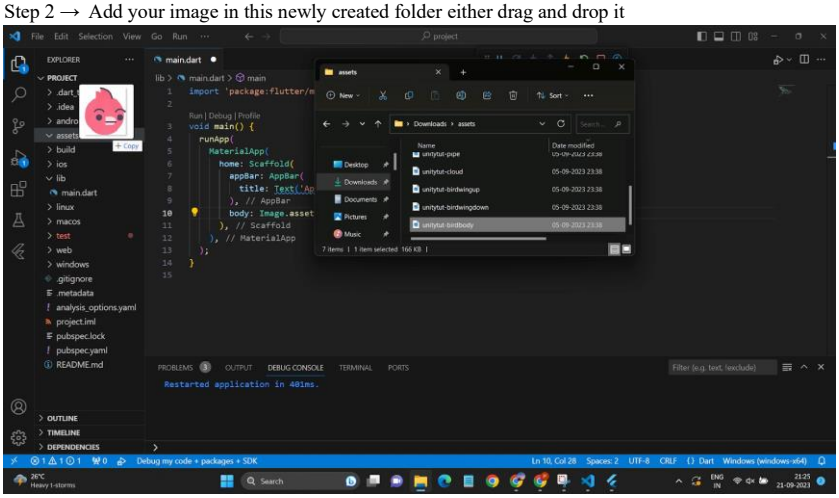


Image.asset():
"image.asset" in Flutter is a method to display static images from the app's assets directory, allowing you to embed and showcase images within your Flutter application.

Step 1 → Create an asset folder for your local images

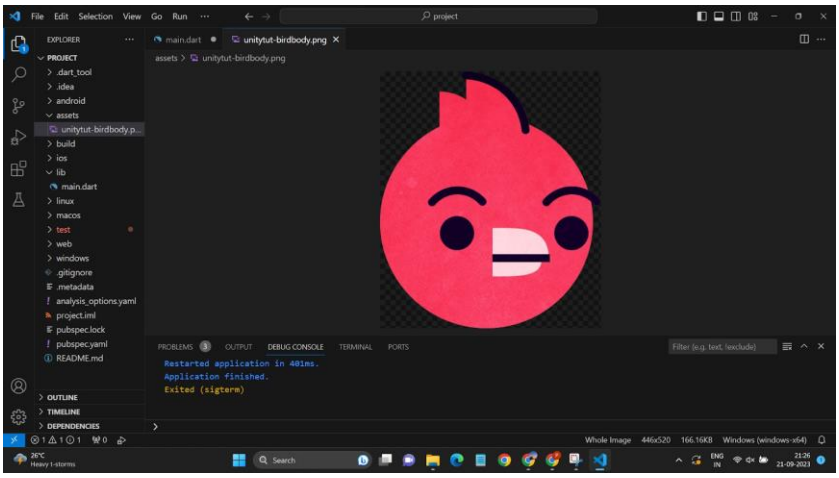


Step 2 → Add your image in this newly created folder either drag and drop it



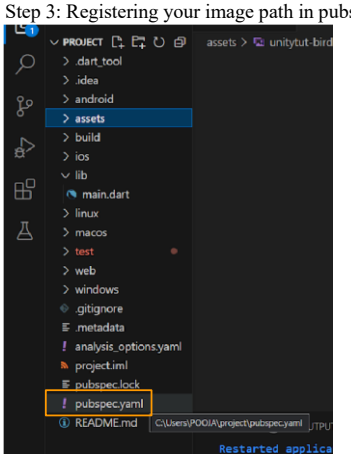
Once added successfully it will be shown like this 👇

Created by - Pooja D. Amin

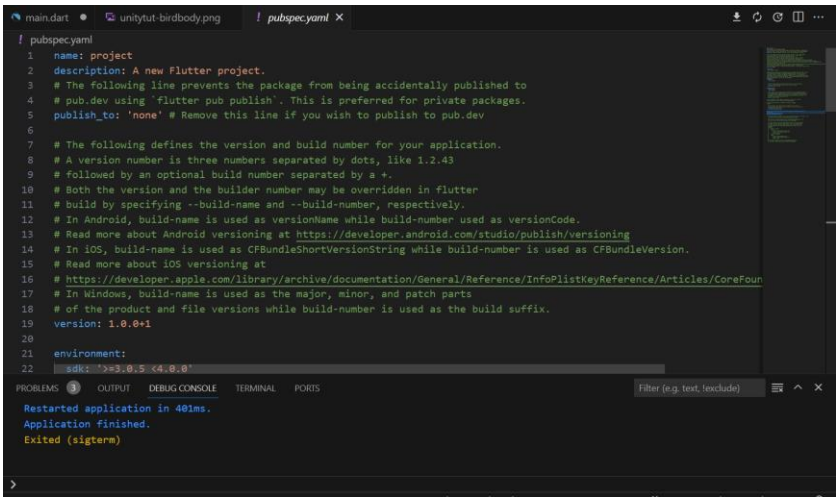


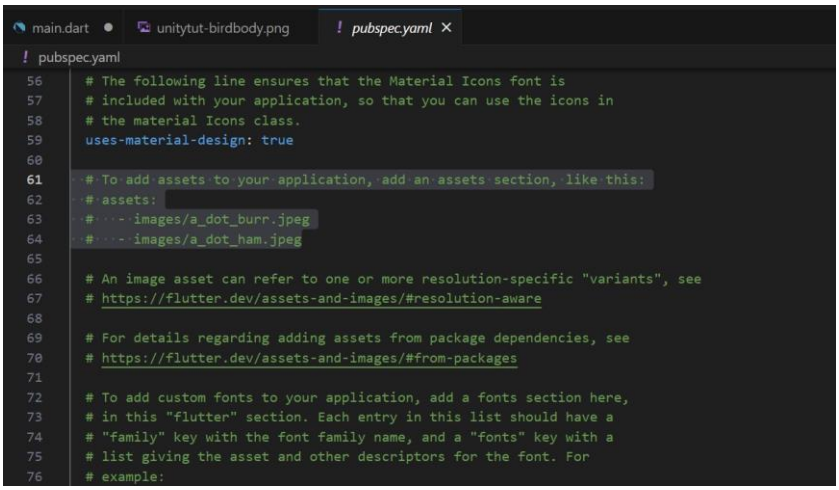
Note: I had later renamed the image name to bird, for my preference

Step 3: Registering your image path in pubspec.yaml file (you will find it in Explorer)



Open the pubspec.yaml file in your project and add the following code to the file:





Check for screenshot line of code

To add assets to your application, add an assets section, like this:

assets:

- images/a_dot_burr.jpeg

- images/a_dot_ham.jpeg

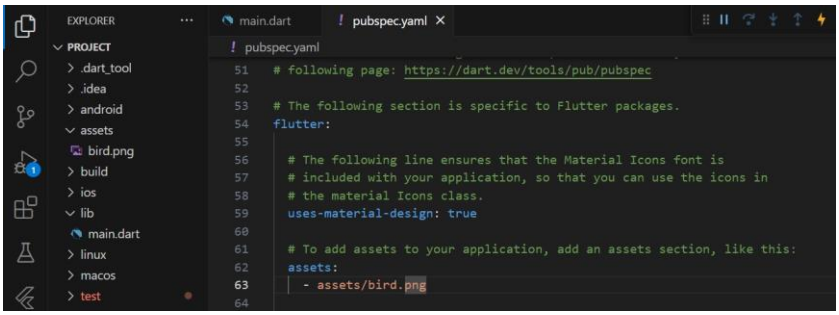
Uncomment lines of code from assets:

Created by - Pooja D. Amin

To uncomment highlighted lines of code → simply select the highlighted line of code → use shortcut key

Ctrl + / to uncomment

- images/a_dot_ham.jpeg, replace this with your assets path location In my case this how it looks



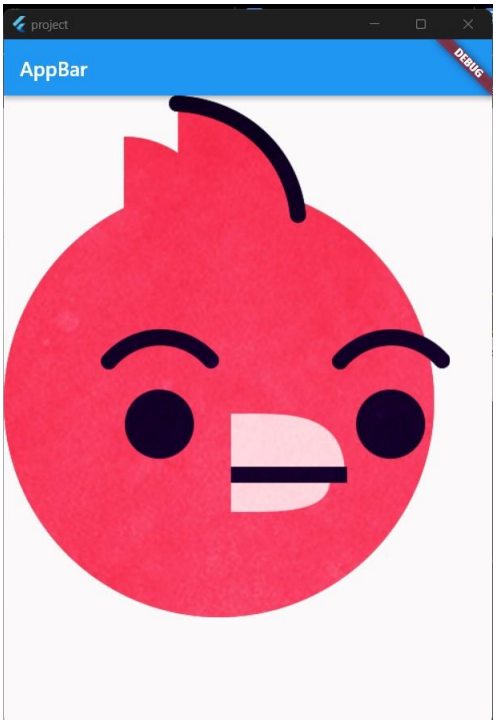
Once added your path → Save this file (Ctrl + S)

Now come back to your main.dart file

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



void main        ()  {
 runApp          (
   MaterialApp   (
     home :  Scaffold    (
       appBar :  AppBar  (
         title :  Text ( 'AppBar'          ) ,
       ) , //AppBar
       body :  Image . asset      (
       "assets/bird.png"         ) ,
     ) , //Scaffold
   ) , //MaterialApp
 ) ;
}
```



7 . Stateless Widgets

A "StatelessWidget" in Flutter is a fundamental component that represents a static part of a user interface. It doesn't maintain any mutable state and is rebuilt whenever its parent's state changes. Stateless widgets are efficient for displaying static content, like icons, text, or images, in a Flutter application.

Stateless Widget code explaination

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

class MyStatelessWidget extends StatelessWidget {
@override
Widget build(context) {  return  // Your
 widget content goes here ;
}
}
```

Now, let's break down the code template:

1. Import Required Packages:

import 'package:flutter/material.dart';

This line imports the Flutter material package, which provides widgets and tools for building user interfaces.

2. Define a Stateless Widget Class:

class MyStatelessWidget extends StatelessWidget:

This defines a new class named MyStatelessWidget that extends StatelessWidget. You can replace MyStatelessWidget with a meaningful name for your widget.

extends is a keyword used to create a subclass (In our case MyStatelessWidget) that inherits properties and behaviors from a superclass (StatelessWidget).

3. Override the build Method:

@override:

This annotation indicates that you are overriding a method from the superclass.

Widget build(context):

This is the overridden build method, which is called when the widget needs to be built.

Inside the build method, you return a widget (in this case, a Container). You can replace the Container with any other widget that suits your UI needs.

Customize the Widget Content:

Within the Container or other widget you return, you can define the content of your stateless widget. This could include text, images, buttons, or any other Flutter widgets.

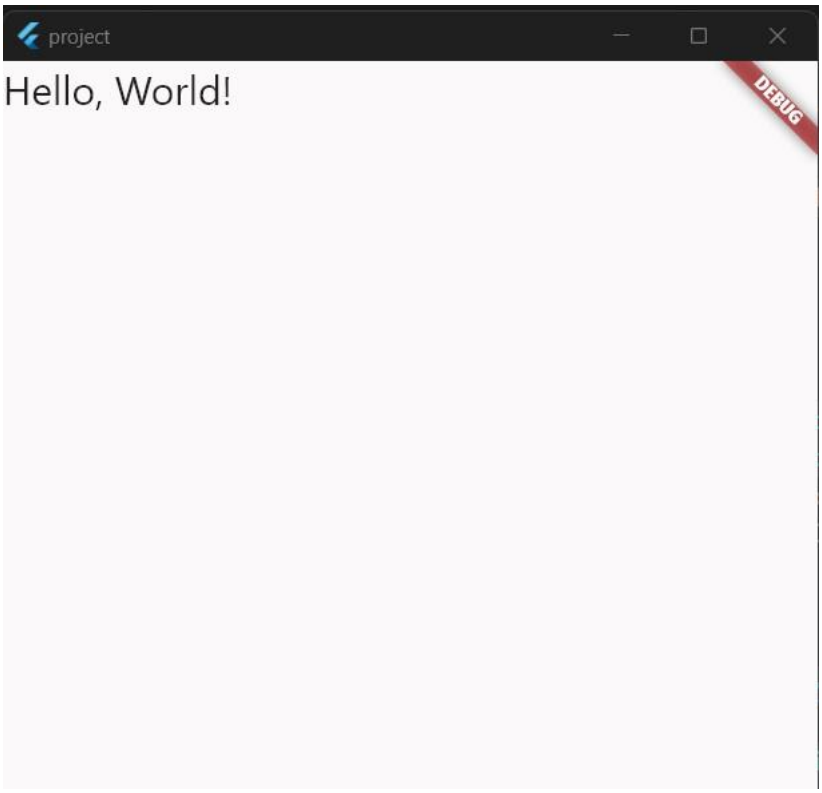
```
import 'package:flutter/material.dart'     ;



void main       ()  {
 runApp         (
   MaterialApp  (
     home :  Scaffold    (
       body :  MyStatelessWidget () ,
/*You can then use this MyStatelessWidget in your Flutter app by creating
an instance of it and including it in your widget tree, just like you
would with any other Flutter widget.*/
     ) ,
   ) ,
 ) ;
}
// Created a customized class
```

```
class  MyStatelessWidget  extends  StatelessWidget {
  @override
  Widget build (  BuildContext context    )  {
    return Container     (
     child :  Text       (
```

```
       'Hello, World!'    ,
       style :  TextStyle ( fontSize      :  24 ) ,
     ) ,
   ) ;
  }
}
```



Practical No. 03

Aim: Designing the mobile app to implement different Layouts.

3 A. Single Child Widget

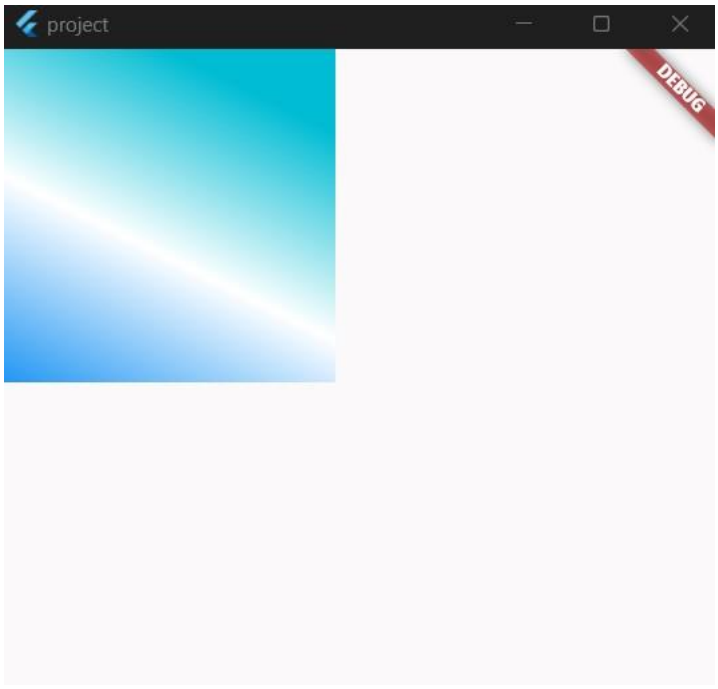
It is a UI element that can contain only one child widget. It's commonly used for components like containers, buttons, and text, allowing precise control over layout and appearance within the app's user interface.

1. Container

It's a widget is a versatile element that provides layout, padding, margin, and decoration to its child widget, allowing for precise control over its appearance and positioning within the user interface.

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

```
void main       ()  {
 runApp         (
   MaterialApp  (
     home :  Scaffold    (
       body :  Container         (
     child:Text('This is Container'),
        width :  200  ,
        height :  200  ,
        decoration :  BoxDecoration      (
          gradient :  LinearGradient     (
            colors :  [ Colors   . blue  , Colors    . white
           , Colors . cyan  ] ,
            begin :  Alignment . bottomLeft       ,
           end :  Alignment . topCenter   ,
          ) , //LinearGradient
        ) , //BoxDecoration
      ) , //Container
    ) , //Scaffold
   ) , // MaterialApp
 ) ;
}
```



2 . SizedBox

It's a widget in Flutter is used to create a box with specified dimensions, providing control over empty space in a layout. It helps in managing the size and spacing of child widgets.

Eg. of SizedBox widget is used with along Column Wiget

3 . Padding

Its a widget in Flutter adds space around its child widget, creating margins. It controls the empty space between the child and its parent, helping with layout and design spacing within the app.

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



void main        ()  {
 runApp          (
   MaterialApp   (
     home :  Scaffold    (
       body :  Padding   (
```

```
          padding :  EdgeInsets . all ( 20.22       ) ,
          child :  Container       (
            width :  200  ,
            height :  200  ,
            decoration :  BoxDecoration     (
              gradient :  LinearGradient   (
                colors :  [ Colors    . blue  , Colors     . white
               , Colors . cyan  ] ,
                begin :  Alignment . bottomLeft      ,
               end :  Alignment . topCenter          ,
              ) , //LinearGradient
            ) , //BoxDecoration
          ) , //Container
        ) , //Padding
      ) , //Scaffold
    ) , //MaterialApp
  ) ;
}
```

4 . Center

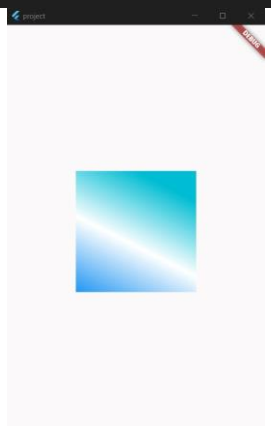
The "Center" widget in Flutter is used to position its child widget in the center of its parent, both horizontally and vertically, simplifying the alignment of content within a user interface.

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



void main        ()  {
 runApp          (
   MaterialApp   (
     home :  Scaffold     (
       body :  Center     (
         child :  Container       (
           width :  200  ,
           height :  200  ,
           decoration :   const BoxDecoration      (
             gradient :  LinearGradient   (
               colors :  [ Colors    . blue  , Colors     . white
, Colors         . cyan    ] ,
```

```
              begin :  Alignment . bottomLeft      ,
             end :  Alignment . topCenter          ,
            ) , //LinearGradient
          ) , //BoxDecoration
        ) , //Container
      ) , //Center
    ) , //Scaffold
  ) , //MaterialApp
 ) ;
}
```



5 . Expanded

The "Expanded" widget in Flutter is used to make child widgets within a column, row, or flex container expand to fill available space, enabling flexible and responsive layouts by distributing space proportionally among its children.

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



void main       ()  {
 runApp ( MaterialApp    (
   home :  Scaffold      (
     body :  Center      (
       child :  Column   (
         children :  [
           Expanded      (
```

```
            child :  Container  (
                decoration :   const BoxDecoration        (
                gradient :  LinearGradient       (
                 colors :  [ Colors   . blue  , Colors    . white
, Colors          . cyan   ] ,
                  begin :  Alignment . bottomLeft          ,
                 end :  Alignment . topCenter     ,
                ) ,
              ) ,
            ) ,
          ) ,
          Expanded      (
            child :  Container  (
                decoration :   const BoxDecoration        (
                gradient :  LinearGradient       (
```

```
              colors :  [ Colors    . yellow  , Colors    . white
, Colors        . green   ] ,
                  begin :  Alignment . bottomLeft          ,
                 end :  Alignment . topCenter     ,
                ) ,
              ) ,
            ) ,
          ) ,
          Expanded      (
            child :  Container  (
                decoration :   const BoxDecoration        (
                gradient :  LinearGradient       (
               colors :  [ Colors   . cyan  , Colors . white     ,
Colors          . purple  ] ,
                  begin :  Alignment . bottomRight         ,
                 end :  Alignment . topLeft       ,
                ) ,
              ) ,
            ) ,
          ) ,
        ] ,
      ) ,
    ) ,
```

Created by - Pooja D. Amin

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