

Flutter Basics

(A basic documentation on widgets)

**Submitted by: Sabarisha K**

**College: Sri Sairam Engineering College**

**3rd Year Computer Science and Engineering**

**Last Modified: 13th Jan, 2024(version 1)**

Table of Contents

S.No Topic Pg.No

1. What is Flutter?------------------------------- 3
2. Why should we use Flutter?----------------- 3
3. What is Dart?---------------------------------- 4
4. Key Features of Dart------------------------- 4
5. Dart Syntax Basics--------------------------- 5
6. What are Widgets in Flutter?--------------- 8
7. How to build UI with Widgets?------------ 9

**1. What is Flutter?**

* Flutter is an open-source UI software development toolkit created by Google. It is used to develop natively compiled applications for mobile, web, and desktop from a single codebase.
* For mobile app development, Android traditionally relies on a native framework built on Java and Kotlin, while iOS uses a framework based on Objective-C/Swift. This leads to the challenge of managing two different languages and frameworks for creating applications for both operating systems. However, in response to this complexity, various frameworks have emerged that support cross-platform development, allowing developers to write code once and deploy it across multiple platforms, including Android, iOS, and even desktop applications.
* A more recent addition to the cross-platform development landscape is Flutter, developed by Google. Flutter stands out as a versatile and powerful framework that enables developers to create high-quality applications for multiple platforms using a single codebase

**2. Why should we use Flutter?**

* Dart has an extensive collection of software packages, enabling developers to enhance their application's functionalities by easily integrating existing solutions.
* Developers can streamline their workflow by writing a single codebase for both Android and iOS applications.
* Flutter's single codebase approach leads to decreased testing efforts. Automated tests need to be written only once, covering both Android and iOS platforms, contributing to a more efficient testing process.
* Flutter's simplicity accelerates the development process, making it an excellent choice for fast-paced projects.
* Developers using Flutter have complete control over the widgets and their layouts.

**3. What is Dart?**

* Dart is a modern, object-oriented programming language developed by Google. It serves as the foundation for Flutter, a UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase.
* Dart is a versatile programming language with a focus on simplicity, productivity, and performance. Its strong typing, support for object-oriented principles, and asynchronous programming capabilities make it a suitable language for a wide range of applications. As developers become familiar with Dart, they gain a solid foundation for building efficient and scalable applications, particularly when combined with the Flutter framework for cross-platform development.

**4. Key Features of Dart**

* **Object-Oriented:**

Dart is a fully object-oriented language where everything is an object. It supports classes and objects, allowing developers to model real-world entities in their code.

* **Strongly Typed:**

Dart is a statically-typed language, meaning variable types are declared at compile time. This helps catch errors early in the development process.

* **Garbage Collection:**

Dart has automatic garbage collection, which means developers do not need to manually manage memory. The system automatically deallocates memory that is no longer in use.

* **Isolates:**

Dart uses isolates for concurrent programming. Isolates are independent workers that do not share memory, providing a scalable approach to handling multiple tasks simultaneously.

* **Asynchronous Programming:**

Dart supports asynchronous programming using Future and Stream classes. This is crucial for handling operations that might take time, such as I/O operations or network requests, without blocking the execution of the program.

**5. Dart Syntax Basics:**

**1. Hello World Program:**

void main() {

print('Hello, Dart!');

}

Dart programs typically start execution from the **main** function.

**2. Variables:**

String name = 'John';

int age = 25;

double salary = 50000.50;

bool isStudent = false;

Dart uses explicit type annotations for variable declarations.

**3. Functions:**

int add(int a, int b) {

return a + b;

}

Functions in Dart are defined using the **functionName(parameters) => expression;**

**4. Control Flow:**

if (condition) {

// code to execute if the condition is true

}

else {

// code to execute if the condition is false

}

for (int i = 0; i < 5; i++) {

// code to repeat

}

while (condition) {

// code to execute while the condition is true

}

Dart supports common control flow structures such as if-else statements, for and while loops.

**5. Lists and Maps:**

List<int> numbers = [1, 2, 3, 4, 5];

Map<String, dynamic> person = {'name': 'Alice', 'age': 30};

Dart provides built-in support for lists and maps.

**6.** **What Are Widgets in Flutter?**

* Widgets are basic building blocks used in Flutter.
* In Flutter, everything is a widget. Widgets are objects that represent the different elements of a user interface. These can range from simple components like buttons and text fields to complex layouts and entire screens. Understanding widgets is crucial to developing effective Flutter applications.

**6.1. Types of Widgets:**

1. **Stateless Widgets:**
   * Stateless widgets are immutable and don't change over time.
   * They are defined by their configuration at the time of creation.
   * Examples include **Text**, **Icon**, and other static components.
2. **Stateful Widgets:**
   * Stateful widgets can change over time and have mutable state.
   * They have a corresponding mutable state object for holding and updating data.
   * Examples include **TextField**, **ListView**, and other dynamic components.

**6.2. Widget Tree**

A Flutter application is structured as a tree of widgets. The root of this tree is typically a **MaterialApp** or **CupertinoApp** widget, depending on the design language chosen. As the tree expands, it defines the entire user interface of the application.

**7. How to build UI with Widgets?**

**7.1. Container Widget:**

* The **Container** widget is one of the most commonly used widgets in Flutter. It acts as a box model that can contain other widgets, allowing developers to customize size, padding, margin, decoration, and more.

Container(

margin: EdgeInsets.all(10.0),

padding: EdgeInsets.symmetric(vertical: 20.0, horizontal: 10.0),

color: Colors.blue, child: Text('Hello, Flutter!'),

)

**7.2. Scaffold Widget**

* The **Scaffold** widget provides a basic structure for material design apps. It includes app bars, drawers, and a bottom navigation bar, simplifying the creation of common app layouts.

Scaffold(

appBar: AppBar(title: Text('My App')),

body: Center(child: Text('Hello, Flutter!')),

)

**7.3. AppBar Widget:**

* In Flutter, the AppBar widget is commonly used to create the top app bar that typically contains the app's title, actions, and possibly other widgets like icons or buttons. Here's a basic example of how to use the AppBar widget in Flutter:

void main() {

runApp(gfgApp());

}

MaterialApp gfgApp() {

return MaterialApp(

home: Scaffold(

appBar: AppBar(

title: const Text('GeeksforGeeks'),

), //AppBar

body: const Center(

child: Text(

'AppBar',

style: TextStyle(fontSize: 24),

), //Text

), // center

), //Scaffold

debugShowCheckedModeBanner: false,

);

}

**7.4. Row and Column Widgets**

* The **Row** and **Column** widgets are used to arrange child widgets in a horizontal or vertical line, respectively. These widgets are crucial for creating flexible and responsive layouts.

Row(

children: [

Icon(Icons.star),

Text('5'),

],

)

Column(

children: [

Text('First Item'),

Text('Second Item'),

],

)

**7.5. ListView Widget**

* The **ListView** widget is employed for displaying a scrolling list of widgets. It is particularly useful when dealing with a large number of items that need to be displayed efficiently.

ListView(

children: [

ListTile(title: Text('Item 1')),

ListTile(title: Text('Item 2')),

ListTile(title: Text('Item 3')),

],

)

**7.6. Text Widget:**

* In Flutter, the Text widget is used to display a piece of text. It's a simple and commonly used widget for rendering text on the screen. Here's a basic example of using the Text widget:

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

body: Center(

child: Text(

'Hello, Flutter!',

style: TextStyle(

fontSize: 20.0,

fontWeight: FontWeight.bold,

color: Colors.blue,

),

),

),

),

);

}

}

**7.7. Icons Widget:**

* The Icon widget in Flutter is used to display graphical icons within your application. Icons can represent various actions, features, or objects and contribute to the overall user interface design.

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(

title: Text('Icon Widget Example'),

),

body: Center(

child: Icon(

Icons.star, // Using the star icon from the Icons class

size: 50.0, // Setting the size of the icon

color: Colors.yellow, // Setting the color of the icon

),

),

),

);

}

}

The widgets that were discussed here are just the beginning of what Flutter can do. Flutter is well-known for providing a wide range of building blocks, or components, that give developers the power to create complex and feature-packed user interfaces. With Flutter's flexibility and strong widget library, developers can design applications that meet various design needs and provide diverse user experiences.