Week-1

1a)Introduction to FLUTTER.

Flutter is an open-source framework developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. It aims to simplify and accelerate the development process while offering a rich set of features to ensure high performance and a great user experience. Flutter is an open-source project hosted on GitHub with contributions from Google and the community. Flutter uses Dart, a modern object-oriented language that compiles to native ARM code and production ready JavaScript code.

Key Features of Flutter

1. **Single Codebase for Multiple Platforms**: With Flutter, you can write your application once and deploy it on multiple platforms, including iOS, Android, web browsers, Windows, macOS, and Linux. This reduces the need for platform-specific code and accelerates development.
2. **Dart Programming Language**: Flutter uses Dart, a language also developed by Google. Dart is designed to be easy to learn and use, with features that facilitate both high performance and a smooth development experience. It supports both just-in-time (JIT) compilation during development for quick iteration and ahead-of-time (AOT) compilation for optimized production builds.
3. **Rich Set of Widgets**: Flutter provides a comprehensive collection of widgets that help you build complex UIs quickly. These widgets are designed to be customizable and can be combined to create highly interactive and visually appealing interfaces. The widgets follow the Material Design guidelines for Android and the Cupertino design guidelines for iOS.
4. **Fast Development with Hot Reload**: One of Flutter's standout features is Hot Reload, which allows developers to see the results of changes almost instantly without restarting the application. This significantly speeds up the development process and makes experimenting with the UI easier.
5. **High Performance**: Flutter's engine, written in C++, provides high performance by rendering directly to the platform's GPU. This eliminates the need for a JavaScript bridge,

as seen in some other cross-platform frameworks, ensuring that the app runs smoothly and efficiently.

6. **Customizable and Extensible**: Flutter's architecture allows for extensive customization and the creation of complex UIs. Developers can build their own custom widgets and extend existing ones to fit their needs. Additionally, the framework supports integration with various plugins and packages to add functionality such as accessing device hardware, networking, and more.
7. **Strong Community and Ecosystem**: Since its introduction, Flutter has gained a strong community of developers and contributors. This has led to a growing ecosystem of packages and plugins that extend the framework's capabilities, making it easier to integrate with third-party services and tools.
8. **Beautiful UIs**: Flutter is known for its ability to create beautiful and smooth UIs. The framework provides a range of animations and visual effects, and its design system allows for the creation of highly responsive and adaptive interfaces.

How Flutter Works

- **Rendering Engine**: Flutter uses Skia, a 2D graphics library, as its rendering engine. This engine draws the application's UI components directly onto the screen, providing smooth and high-performance rendering across different platforms.
- **Widgets**: Everything in Flutter is a widget, from the structural elements like buttons and menus to the stylistic aspects like fonts and colors. Widgets are composed to build the UI, and Flutter's declarative approach makes it easy to manage state and build complex layouts.
- **Platform Channels**: To interact with native code or access platform-specific features, Flutter uses platform channels. These channels enable communication between Dart code and native code (Java/Kotlin for Android, Objective-C/Swift for iOS).

Widgets

In Flutter, a **widget** is a fundamental building block used to construct user interfaces. Widgets are the primary way developers define and compose the layout and behavior of their applications. Here's a comprehensive look at what widgets are and how they function within the Flutter framework:

Definition and Purpose

- **Definition**: A widget in Flutter is a description of part of a user interface. It can represent anything visible on the screen, from a simple button to a complex layout.
- **Purpose**: Widgets are used to construct the UI of a Flutter application by describing what their view should look like given their current configuration and state. They encapsulate both the visual layout (how things look) and the behavior (how things work) of a component.

Characteristics of Widgets

- **Immutable**: Widgets in Flutter are immutable, meaning once they are created, they cannot be changed. If a widget needs to change its appearance or behavior, a new widget instance is created with the updated configuration.
- **Composable**: Widgets can be combined together to build complex UIs. Flutter provides a rich set of pre-built widgets (like text, buttons, images, etc.) that can be nested and composed to create custom widgets or entire screens.

Types of Widgets

Flutter categorizes widgets into two main types based on how they manage state:

1. **Stateless Widgets**:
    - **Definition**: Stateless widgets are widgets that do not have mutable state. They are immutable and their appearance is solely a function of the configuration information in their constructor.
    - **Characteristics**: Stateless widgets are simple and lightweight because they do not need to manage state changes. Examples include Text, Icon, Image, etc.

The following sample code shows a StatelessWidget base structure

```
class JournalList extends StatelessWidget {
@override
Widget build(BuildContext context) {
return Container();
```

}
}

2. **Stateful Widgets**:
    - **Definition**: Stateful widgets are widgets that maintain state that might change during the lifetime of the widget. They are responsible for managing and updating their own state.
    - **Characteristics**: Stateful widgets are more complex compared to stateless widgets because they have mutable state. Examples include TextField, Checkbox, Slider, etc.

The following example shows a StatefulWidget base structure

```
class JournalEdit extends StatefulWidget {
@override
_ JournalEditState createState() => _ JournalEditState();
}
class _JournalEditState extends State<JournalEdit> {
@override
Widget build(BuildContext context) {
return Container();
}
}
```

Widget Lifecycle

Widgets in Flutter go through a lifecycle from creation to destruction:

- **Creation**: Widgets are instantiated using their constructor, which defines their configuration.
- **Build**: The build() method is called to obtain the widget's UI representation based on its current configuration.
- **Update**: Widgets can be rebuilt (i.e., the build() method is called again) when their configuration or the data they depend on changes.
- **Destruction**: Widgets may be destroyed when they are no longer needed (e.g., when they are removed from the widget tree).

Example of a Simple Widget

Here's an example of a stateless widget (Text widget) in Flutter:

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

class MyApp extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('My App'),
        ),
        body: Center(
          child: Text(
            'Hello, Flutter!',
            style: TextStyle(fontSize: 24.0),
          ),
        ),
      ),
    );
  }
}
```

- In this example, the `Text` widget displays the text "Hello, Flutter!" in the center of the screen.
- The `Text` widget is immutable and its appearance is determined solely by its `data` and `style` properties.

- When the app runs, Flutter constructs a widget tree starting from `MyApp` down to `Text`, and ultimately displays the UI on the device.

Widgets are the fundamental building blocks of Flutter applications, responsible for defining both the visual layout and behavior of the user interface. Understanding how to create, compose, and manage widgets effectively is crucial for developing robust and responsive Flutter applications.

1b)Installation of Flutter for Windows.

Write the steps from Observation.