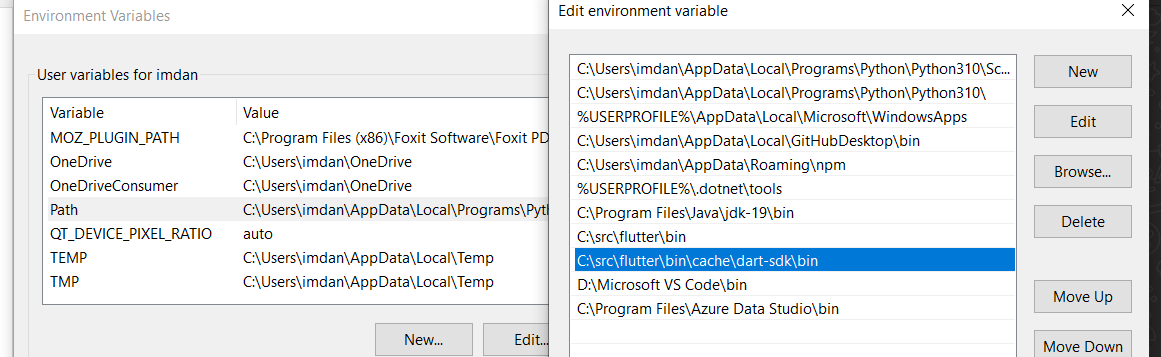
**Byte Wise Fellow Ship (Flutter)**

During my three months flutter fellowship at Byte wise I have done multiple task and projects. All of these task explained below

**Week 1 + 2**

**Environment Setup**

I installed flutter and dart and add its path to environment variable  
 

Then I downloaded android studio.

**Introduction to Flutter**

Flutter is a growing mobile framework developed by Google, designed to help in the development of mobile and web applications with only one code base and the ability to export them as native apps for Android and iOS.

Flutter is a tool that allows developers to build native cross-platform apps with just one programming language and one codebase. It will not create an app that runs in the browser or something that gets wrapped by native apps. Instead, it creates a native app for both iOS and Android that can be published to the stores later.  
Flutter is made if **2 major components; 1) SDK Software Development Kit** is a collection of tools that will assist you in developing your applications. Tools for compiling your code into native machine code are included one for ios and other for android. **2) Framework** is a widget-based UI library a set of reusable UI elements such as buttons, text inputs, and sliders and so on so that you can customize to meet your specific needs.

**Dart**

Dart is a programming language that will be used to create flutter applications. It is focused of building front-end user interfaces and front-end apps. By using it, you will be able to create either web apps or mobile apps.

It is developed by Google and is class based, object oriented and strongly typed programming language. Dart’s syntax is very similar to languages like Java or JavaScript.

Flutter and Dart are actually two independent things that work together. Flutter is a framework for Dart, and Dart uses the capabilities of Flutter to build the app. In the end, Flutter working as an SDK will help to build the app.

In this week, I learned about basics of Dart language.

I use [DartPad](https://dartpad.dev/?) for practicing dart language

**Dart- Data Types**

The Dart Language supports the following types

* Integer
* Double
* String
* Bool
* Lists
* Maps
* Dynamic

**List-** A list is an ordered group of objects. The list data type in Dart is synonymous to concept of an array in other programming languages.

**Map-** The Map data type represents a set of values as key-values pairs. The **dart: core** library enables creation and manipulation of these collections through the predefined List and Maps classes respectively.

**Dynamic Type**Dart is an optionally typed language. If the type of a variable is not explicitly specified, the variable’s type is **dynamic**. The **dynamic** keyword can also be used as a type annotation explicitly.

**Type Syntax**

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.

**Note:** Dart supports both type inference and statically type defined

var name='Daniyal';

It means value store in **name** variable is a string because Dart compiler automatically guess it because of **var** keyword.

var num=10;  
It means value store in **num** is a integer value.  
  
The above mentioned syntax are called type inference

String name='Daniyal';  
int num=10;  
This is called statically type defined as I define the data type of name and number   
If I use String Data Type and assign a integer value in it then it shows an error

**Dart Programming- Operators**

Following operators are available in dart

* Arithmetic Operators
* Equality and Relational Operators
* Assignment Operators
* Logical Operators
* Type Test Operators
* Bitwise Operators

**Dart Programming- Enumeration**

An enumeration is used for defining named constant values. An enumerated type is declared using the enum keyword.

**Syntax**

enum enum\_name{

enumeration list

}

enum Status {

none,

running,

stopped,

paused

}

void main() {

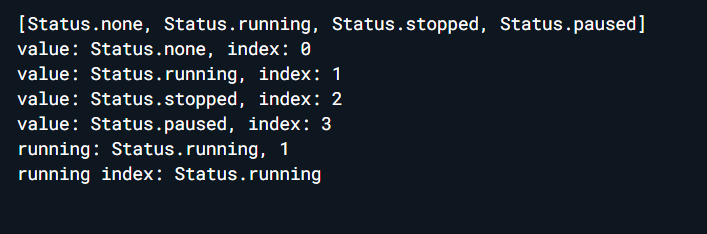
print(Status.values);

Status.values.forEach((v) => print('value: $v, index: ${v.index}'));

print('running: ${Status.running}, ${Status.running.index}');

print('running index: ${Status.values[1]}');

}



**Dart Programming - Classes**

Dart is an object-oriented language. It supports object-oriented programming features like classes, interfaces, etc. A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object. Dart gives built-in support for this concept called class.

**Week 3: Widgets & Layouts**

In this week I have learned about different widgets and layouts in flutter.

**Widgets**

Widgets are classes used to build UIs. Widgets are used for both layouts and UI elemnts.

1. Container Widget
2. Row Widget
3. Stack Widget
4. Expanded Widget
5. Circular Widget
6. Divider Widgets
7. Text Field
8. List Tile Widget
9. **Container Widget**

The Container widget lets you create a rectangular visual element. A container can be decorated with a BoxDecoration, such as a background, a border, or a shadow. A Container can also have margins, padding, and constraints applied to its size. In addition, a Container can be transformed in three dimensional space using a matrix.

1. **Row Widget**

Row widget displays its children in a horizontal array.

To cause a child to expand to fill the available horizontal space, wrap the child in an [Expanded](https://api.flutter.dev/flutter/widgets/Expanded-class.html) widget.

The [Row](https://api.flutter.dev/flutter/widgets/Row-class.html) widget does not scroll (and in general it is considered an error to have more children in a [Row](https://api.flutter.dev/flutter/widgets/Row-class.html) than will fit in the available room). If you have a line of widgets and want them to be able to scroll if there is insufficient room, consider using a [ListView](https://api.flutter.dev/flutter/widgets/ListView-class.html).

1. **Stack Widget**

Instead of being linearly oriented (either horizontally or vertically), a Stack widget lets you place widgets on top of each other in paint order. You can then use the [Positioned](https://api.flutter.dev/flutter/widgets/Positioned-class.html) widget on children of a Stack to position them relative to the top, right, bottom, or left edge of the stack. Stacks are based on the web’s absolute positioning layout model.

1. **Expanded Widget**

Expanded widget expands a child of a [Row](https://api.flutter.dev/flutter/widgets/Row-class.html), [Column](https://api.flutter.dev/flutter/widgets/Column-class.html), or [Flex](https://api.flutter.dev/flutter/widgets/Flex-class.html) so that the child fills the available space.

Using an [Expanded](https://api.flutter.dev/flutter/widgets/Expanded-class.html) widget makes a child of a [Row](https://api.flutter.dev/flutter/widgets/Row-class.html), [Column](https://api.flutter.dev/flutter/widgets/Column-class.html), or [Flex](https://api.flutter.dev/flutter/widgets/Flex-class.html) expand to fill the available space along the main axis (e.g., horizontally for a [Row](https://api.flutter.dev/flutter/widgets/Row-class.html) or vertically for a [Column](https://api.flutter.dev/flutter/widgets/Column-class.html)). If multiple children are expanded, the available space is divided among them according to the [flex](https://api.flutter.dev/flutter/widgets/Flexible/flex.html) factor.

An [Expanded](https://api.flutter.dev/flutter/widgets/Expanded-class.html) widget must be a descendant of a [Row](https://api.flutter.dev/flutter/widgets/Row-class.html), [Column](https://api.flutter.dev/flutter/widgets/Column-class.html), or [Flex](https://api.flutter.dev/flutter/widgets/Flex-class.html), and the path from the [Expanded](https://api.flutter.dev/flutter/widgets/Expanded-class.html) widget to its enclosing [Row](https://api.flutter.dev/flutter/widgets/Row-class.html), [Column](https://api.flutter.dev/flutter/widgets/Column-class.html), or [Flex](https://api.flutter.dev/flutter/widgets/Flex-class.html) must contain only [StatelessWidget](https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html)s or [StatefulWidget](https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html)s (not other kinds of widgets, like [RenderObjectWidget](https://api.flutter.dev/flutter/widgets/RenderObjectWidget-class.html)s).

1. **Circular Avatar Widget**

Circular avatar widget is used to create a Circular Image in Flutter. CircleAvatar is simply a widget used to display a user profile picture. In the absence of a user's profile picture, Circle Avatar can display the user's initials.

1. **Divider Widget**

A thin horizontal line, with padding on either side. In the Material Design language, this represents a divider. Dividers can be used in lists, [Drawer](https://api.flutter.dev/flutter/material/Drawer-class.html)s, and elsewhere to separate content.

1. **Text Form Field Widget**

TextField in Flutter is the most commonly used text input widget that allows users to collect inputs from the keyboard into an app. We can use the TextField widget in building forms, sending messages, creating search experiences, and many more. By default, Flutter decorated the TextField with an underline.

1. **List Tile Widget**

It is mainly used to populate the scrollable views such as ListView, Column, and Row. For example, you can use the ListTile to show a list of to-do items, emails, navigation options, and more.

**Layouts**

In Flutter, a layout refers to the way widgets are arranged on the screen. Layouts are used to define the structure of a user interface by specifying the position and size of each widget in relation to other widgets.

Flutter provides a wide range of layout widgets that developers can use to create responsive and dynamic user interfaces. These widgets include:

1. Container: A widget that allows developers to specify a specific width and height for a child widget.
2. Row and Column: Widgets that display their children in a horizontal or vertical line.
3. Stack: A widget that overlays its children on top of each other.
4. Expanded: A widget that expands its child to fill the available space.
5. GridView: A widget that arranges its children in a grid.
6. ListView: A widget that displays a scrollable list of children.
7. Wrap: A widget that displays its children in a flow layout.
8. Flex: A widget that allows developers to create flexible layouts that can adapt to different screen sizes and orientations.

Using these layout widgets, developers can create complex user interfaces with ease, while ensuring that the UI is responsive and adapts to different screen sizes and orientations.

**Week 4+5+6+7+8  
Tasks: Interactivity + Assets + Navigation + Routing + Animations + Advanced UI**

**Interactivity:**

In Flutter, interactivity refers to the ability of users to interact with the user interface of a mobile or web application. Interactivity allows users to perform actions such as tapping buttons, scrolling lists, entering text, and navigating between screens.

You’ll add interactivity to an app that contains only non-interactive widgets. you’ll add interactivity to an app that contains only non-interactive widgets.

**Assets:**

Flutter apps can include both code and assets (sometimes called resources). An asset is a file that is bundled and deployed with your app, and is accessible at runtime. Common types of assets include static data (for example, JSON files), configuration files, icons, and images (JPEG, WebP, GIF, animated WebP/GIF, PNG, BMP, and WBMP).

**Navigation + Routing:**

Navigation and routing are some of the core concepts of all mobile application, which allows the user to move between different pages. We know that every mobile application contains several screens for displaying different types of information. For example, an app can have a screen that contains various products. When the user taps on that product, immediately it will display detailed information about that product.

In Flutter, the screens and pages are known as routes, and these routes are just a widget. In Android, a route is similar to an Activity, whereas, in iOS, it is equivalent to a ViewController.

**Advance UI**

Advanced UI in Flutter refers to the creation of complex and sophisticated user interfaces that go beyond the standard UI designs offered by Flutter's built-in widgets. Advanced UI designs typically require a high level of customization, creativity, and attention to detail.

Advanced UI in Flutter refers to the creation of custom, complex, and visually appealing UI designs that go beyond the standard UI designs offered by Flutter's built-in widgets. Flutter offers several tools and techniques that make it easy for developers to create advanced UI designs, including custom UI widgets, animations and transitions, custom painting, and layout constraints.

**Animations**

Animations are an essential part of building modern and engaging user interfaces in mobile applications. Flutter offers a powerful and flexible animation system that allows developers to create visually stunning animations with ease.

In Flutter, animations are created using the Animation class, which provides a simple and flexible way to define animations and specify how they should be updated over time. The Animation class is an abstract class that defines a range of values that the animation should interpolate between, along with a duration and curve that determines the speed and timing of the animation.

**Week 9+10: State Management**

**State Management**

State management is the process of tracking and updating the state of your application. In Flutter, state is anything that can change over time, such as the current user, the contents of a shopping cart, or the position of a map marker.

There are a number of different state management techniques available in Flutter, each with its own advantages and disadvantages. Some of the most popular state management techniques include:

* **Inherited Widget:**This technique allows you to share state between widgets that are not directly connected to each other.
* **StatefulWidget:** This technique allows you to create widgets that can change their state over time.
* **Provider:** This is a popular state management package that makes it easy to share state between widgets.
* **MobX:** This is another popular state management package that uses observables to track state changes.
* **RiverPod:** Riverpod is a state management library for Flutter that uses providers to manage state. Providers are objects that encapsulate state and allow listening to changes in that state. This makes it easy to update the UI when state changes. Riverpod is also very efficient, only rebuilding the UI when state actually changes.

The best state management technique for your application will depend on the specific needs of your app. If you are just starting out with Flutter, I recommend using the Inherited Widget or StatefulWidget techniques. Once you have a better understanding of how state management works, you can explore other options such as Provider or MobX.

Here are some of the benefits of using state management in Flutter:

* Easier to maintain: State management can help you keep your code organized and easier to maintain.
* Fewer bugs: State management can help you avoid bugs that are caused by changes to state.
* Better performance: State management can help improve the performance of your application by reducing the number of times the UI needs to be redrawn.