Flutter Lab

1.

a) Install Flutter and Dart SDK.

To install Flutter and the Dart SDK on Windows, follow these steps:

1. Download the Flutter SDK

1. Go to the Flutter download page.
2. Download the latest stable version for Windows.
3. Extract the downloaded `.zip` file into a directory, for example, `C:\\src\\flutter` (don't install Flutter in a directory that requires elevated permissions, like `C:\\Program Files`).

2. Add Flutter to the System Path

1. Open **Control Panel > System and Security > System > Advanced system settings > Environment Variables**.
2. Under **System variables**, find the **Path** variable and click **Edit**.
3. Add a new entry with the path to the Flutter bin folder, e.g., `C:\\src\\flutter\\bin`.
4. Click **OK** to save.

3. Run `flutter doctor`

1. Open a **Command Prompt** or **PowerShell** window.
2. Run `flutter doctor` to check if there are any additional dependencies to install.
3. Follow any additional instructions provided by `flutter doctor` to complete the setup.

4. Install Visual Studio Code or Android Studio (Optional)

- Since you're using Visual Studio for Dart and Flutter development without Android Studio, you can skip installing Android Studio if you don't need the Android emulator. If you do need the emulator, you'll need to install Android Studio.

5. Verify Installation

1. Run `flutter doctor` again to ensure all checks are resolved.
2. If there are issues related to the Android toolchain, you may need to configure Android SDK paths or install missing dependencies.

Now, you should be set up to develop with Flutter and Dart on Windows using Visual Studio Code or Visual Studio. Let me know if you encounter any issues!

b) Write a simple Dart program to understand the language basics.

#in main.dart

```
void main(){
  var name="Batman";
  int x=20;
  double pi=3.14;

  print("Welcome to the team Avengers $name ");
  print("version: $x");

  if(pi>3){
    print("Value of pi is greater than 3");
  }
  else{
    print("Value of pi is less than 3");
  }

  print("Printing Numbers 1 to 5");
  for(int i=1;i<=5;i++){
    print(i);
  }
  int a=10,b=20;
  int sum=add(a,b);
  print("sum : $sum");

  var names=Car("Tesla","Mercedes");
    names.display();
}

int add(int a,int b){
  return (a+b);
}

class Car{
 late String name1,name2;

  Car(name1,name2){
    this.name1=name1;
    this.name2=name2;
  }
  void display(){
    print("car details:");
    print(name1);
    print(name2);
  }
}
```

2.

a)

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

void main(){
  runApp(const MyApp());
}
class MyApp extends StatelessWidget{
 const MyApp({super.key});
  @override
  Widget build(BuildContext context){
    return const MaterialApp(
      home:Scaffold(
        body:Center(
          child:
        )
      )
    );
  }
}
```

a)Explore various Flutter widgets (Text, Image, Container, etc.)

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

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

class Rootroute extends StatelessWidget {
  const Rootroute({super.key});

  @override
  Widget build(BuildContext context) {
    return  Scaffold(
      appBar: AppBar(
        title: const Text("Flutter App"),
        ),
        body:
        Container(
          width: double.infinity,
          child:Image.asset("images/imflutter.png"),

        ),
    );
  }
}
```

2).Implement different layout structures using Row, Column, and Stack widgets.

Row

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

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

```
class Rootroute extends StatelessWidget {
  const Rootroute({super.key});

  @override
  Widget build(BuildContext context) {
    return  Scaffold(
      appBar: AppBar(
        title: const Text("Flutter App"),
        ),
        body: Row(
          children: [
              Image.asset("images/imflutter.png"),
              const SizedBox(
                width: 10,
              ),
              const Text(
                "Hey,there!",
                style: TextStyle(
                  color: Colors.blue,
                ),
              ),

          ],


        ),
    );
  }
}
```

Column

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

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

class Rootroute extends StatelessWidget {
  const Rootroute({super.key});

  @override
  Widget build(BuildContext context) {
    return  Scaffold(
      appBar: AppBar(
        title: const Text("Flutter App"),
        ),
        body: Column(
          children: [
              Image.asset("images/imflutter.png"),
              const SizedBox(
                height: 10,
              ),
              const Text(
                "Hey,there!",
                style: TextStyle(
                  color: Colors.blue,
                ),
              ),

          ],


        ),
      );
    }
  }
```

Stack widget

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

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

class Rootroute extends StatelessWidget {
  const Rootroute({super.key});

  @override
  Widget build(BuildContext context) {
    return  Scaffold(
      appBar: AppBar(
        title: const Text("Flutter App"),
        ),
        body: Stack(
          children: [
              Image.asset("images/imflutter.png"),

              const Text(
                "Hey,there!",
                style: TextStyle(
                  color: Colors.blue,
                ),
              ),

          ],
        ),
      );
    }
  }
```

3.a)

Design a responsive UI that adapts to different screen sizes.

Implement media queries and breakpoints for responsiveness.

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    double width = MediaQuery.of(context).size.width;
    return MaterialApp(
        home: width < 600 ? const FirstScreen() :
        width>=600 && width<1200 ?const SecondScreen():const ThirdScreen()
        );
  }
}

class FirstScreen extends StatelessWidget {
  const FirstScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: const Text("Flutter"),
        ),
        body: const Column(
          children: [
            Text(
              "HELLO BOX",
              style: TextStyle(
                color: Colors.blue,
              ),
            ),
           DecoratedBox(decoration:
           BoxDecoration(
            color: Colors.red,

           ),
           child: SizedBox(
            width: double.infinity,
            child: Text("Box"),
           ),
           ),
          ],
        ));
  }
}

class SecondScreen extends StatelessWidget {
  const SecondScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: const Text("Flutter"),
        ),
        body: const Column(
          children: [
            Text(
              "HELLO BOX",
              style: TextStyle(
                color: Colors.blue,
              ),
            ),
           DecoratedBox(decoration:
           BoxDecoration(
            color: Colors.green,

           ),
           child: SizedBox(
            width: double.infinity,
            child: Text("Box"),
           ),
           ),
          ],
        ));
  }
}


class ThirdScreen extends StatelessWidget {
  const ThirdScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: const Text("Flutter"),
        ),
        body: const Column(
          children: [
            Text(
              "HELLO BOX",
              style: TextStyle(
                color: Colors.blue,
              ),
            ),
           DecoratedBox(decoration:
           BoxDecoration(
            color: Colors.yellow,

           ),
           child: SizedBox(
            width: double.infinity,
            child: Text("Box"),
           ),
           ),
          ],
        ));
  }
}
```

4.

a) Set up navigation between different screens using Navigator.
b) Implement navigation with named routes.

```
import 'package:flutter/material.dart';
import 'package:flutter_application_2/second_screen.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {

    return const MaterialApp(
        home:FirstScreen()
        );
  }
}

class FirstScreen extends StatelessWidget {
  const FirstScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Flutter"),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.of(context).push(
              MaterialPageRoute(builder: (BuildContext context){
               return const SecondScreen();
              })
            );
          },
           child: const Text("Go to SecondScreen"),
           ),
      ),
    );
  }
}
```

second

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

class SecondScreen extends StatelessWidget {
  const SecondScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title:  const Text("Go Back"),
      ),
      body: const Center(
        child: Text("Hello this is Second Screen"),
      ),
    );
  }
}
```

5)

a) Learn about stateful and stateless widgets.

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

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

class FirstScreen extends StatefulWidget {
  const FirstScreen({super.key});

  @override
  State<FirstScreen> createState() => _FirstScreenState();
}

class _FirstScreenState extends State<FirstScreen> {
   int val=0;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Stateful Widget"),
      ),
      floatingActionButton: FloatingActionButton(onPressed: (){},child:const Icon(Icon
s.add),),
      bottomNavigationBar: NavigationBar(
        destinations: const[
           NavigationDestination(icon: Icon(Icons.home), label: 'home'),
          NavigationDestination(icon: Icon(Icons.person), label: 'contact'),

        ],
        onDestinationSelected: (val1){
          setState(() {
            val=val1;
          });
        },
        selectedIndex: val,
      ),
    );
  }
}
```

b)Implement state management using set State and Provider.

using set State

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

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

class Rootroute extends StatefulWidget {
  const Rootroute({super.key});

  @override
  State<Rootroute> createState() => _RootrouteState();
}

class _RootrouteState extends State<Rootroute> {

  late int val;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Flutter"),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {

        },
        child: const Icon(Icons.add),
      ),
      bottomNavigationBar: NavigationBar(
        destinations: const [
          NavigationDestination(icon: Icon(Icons.home), label: 'home'),
          NavigationDestination(icon: Icon(Icons.person), label: 'person'),
        ],
        onDestinationSelected: (value) {
          setState(() {
            val=value;
          });
        },
        selectedIndex: val,
        ),
    );
  }
}
```

using Provider

dependencies:
flutter:
sdk: flutter
provider: ^6.1.2 # Use the latest version available

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

void main() {
  runApp(ChangeNotifierProvider(
    create: (context)=>Npv(),
    child: const MyApp(),
  ));
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

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

class FirstScreen extends StatefulWidget {
  const FirstScreen({super.key});

  @override
  State<FirstScreen> createState() => _FirstScreenState();
}

class _FirstScreenState extends State<FirstScreen> {
  int val = 0;
  @override
  Widget build(BuildContext context) {
    final npv=context.watch<Npv>();
    return Scaffold(
      appBar: AppBar(
        title: const Text("Provider Example"),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          context.read<Npv>().increment();
        },
        child: const Icon(Icons.add),
      ),
      bottomNavigationBar: NavigationBar(
        destinations: const [
          NavigationDestination(icon: Icon(Icons.home), label: 'home'),
          NavigationDestination(icon: Icon(Icons.person), label: 'contact'),
          NavigationDestination(icon: Icon(Icons.phone), label: 'phone'),
          NavigationDestination(icon: Icon(Icons.email), label: 'email'),

        ],

        selectedIndex: npv.v,
      ),
    );
  }
}


class Npv extends ChangeNotifier{
  int v=0;
  void increment(){
    v++;
    notifyListeners();
  }
}
```

6).

a.Create custom widgets for specific UI elements.

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: CustomWidget(
        text:"Hello Batman",
        textColor:Colors.blue,
      ),
    );
  }
}

class CustomWidget extends StatelessWidget {
  final String text;
   final Color textColor;
  const CustomWidget({super.key,required this.text,required this.textColor});
  @override
  Widget build(BuildContext context) {
    return  Scaffold(
      appBar: AppBar(
        title: const Text("Flutter App"),
        ),
        body: Text(
          text,
          style: TextStyle(
             color: textColor,
             fontSize:24,
              fontWeight: FontWeight.bold,
          ),
        ),

    );
  }
}
```

b. Apply styling using themes and custom styles.

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return  MaterialApp(
      theme: ThemeData(
        primarySwatch: Colors.blue,
         textTheme: TextTheme(
          bodyLarge: TextStyle(
            fontSize: 20,
            color: Colors.black,
          )
         )
      ),
      home: CustomWidget(
        text:"Hello Batman",
        textColor:Colors.blue,
      ),
    );
  }
}

class CustomWidget extends StatelessWidget {
  final String text;
   final Color textColor;
  const CustomWidget({super.key,required this.text,required this.textColor});


  @override
  Widget build(BuildContext context) {
    return  Scaffold(
      appBar: AppBar(
        title: const Text("Flutter App"),
        ),
        body: Text(
          text,
          style:Theme.of(context).textTheme.bodyLarge?.copyWith(

             color: textColor,
             fontSize:24,
              fontWeight: FontWeight.bold,

        ),
        )
    );
  }
}
```

7.

a) Design a form with various input fields.

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

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

class LoginPage extends StatelessWidget {
  const LoginPage({super.key});

  @override
  Widget build(BuildContext context) {
    final TextEditingController controller1 = TextEditingController();
    final TextEditingController controller2 = TextEditingController();
    return Scaffold(
      appBar: AppBar(
        title: const Text("Login Page"),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            TextField(
              controller: controller1,
              decoration: const InputDecoration(
                labelText: 'UserName',
              ),
            ),
            const SizedBox(
              height: 16,
            ),
            TextField(
              controller: controller2,
              obscureText: true,
              decoration: const InputDecoration(labelText: "Password"),
            ),
            ElevatedButton(
              onPressed: () {
                final name = controller1.text;
                final pass = controller2.text;
                print('Entered username:  $name');
                print('Entered password: $pass');
              },
              child: const Text("Submit"),
            )
          ],
        ),
      ),
    );
  }
}
```

b) Implement form validation and error handling.

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return  MaterialApp(

      home: LoginPage(),
    );
  }
}

class LoginPage extends StatelessWidget {
  const LoginPage({super.key});

  @override
  Widget build(BuildContext context) {
    final TextEditingController controller1=TextEditingController();
    final TextEditingController controller2=TextEditingController();
    return Scaffold(
      appBar: AppBar(
        title: Text("Login Page"),
      ),
      body: Padding(
        padding: EdgeInsets.all(16),
        child: Column(
          children: [
            TextField(
             controller: controller1,
             decoration: InputDecoration(
              labelText: 'UserName',
             ),
            ),
            SizedBox(height: 16,),
            TextField(
              obscureText: true,
              controller: controller2,
              decoration: InputDecoration(
                labelText: "Password"
              ),
            ),
            ElevatedButton(
              onPressed: (){
                 final username =controller1.text;
                final password = controller2.text;

                if (username.isEmpty || password.isEmpty) {
                  ScaffoldMessenger.of(context).showSnackBar(
                    SnackBar(content: Text("Please fill all details")),
                  );
                } else if(password.length<8){
                  ScaffoldMessenger.of(context).showSnackBar(
                    SnackBar(content: Text("Enter atleast 8 characters"))
                  );
                }
                else {
                  print('Entered username: $username');
                  print('Entered password: $password');
                }


              },
            child:Text("Submit"),
             )
          ],


        ),
        ),
    );
  }
}
```

8.

. a) Add animations to UI elements using Flutter's animation framework.
b) Experiment with different types of animations (fade, slide, etc.).

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: AnimatedPage(),
    );
  }
}

class AnimatedPage extends StatefulWidget {
  const AnimatedPage({super.key});

  @override
  State<AnimatedPage> createState() => _AnimatedPageState();
}

class _AnimatedPageState extends State<AnimatedPage>
    with TickerProviderStateMixin {
  late final AnimationController controller = AnimationController(
    duration: Duration(seconds: 2),
    vsync: this,
  )..repeat(reverse: true);
  @override
  void dispose() {
    controller.dispose();
    super.dispose();
  }

  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Flutter"),
      ),
      body: Center(
        child:
              SlideTransition(
                position: Tween<Offset>(
              begin: Offset(-1, 1),
              end: Offset(0, 0)
            ).animate(controller),


            child: FadeTransition(
              opacity: controller,
              child: Text(
                "Hello Animation !",
                style: TextStyle(
                  fontSize: 24,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),



        ),
      ),
    );
  }
}
```

9.

a) Fetch data from a REST API.
b) Display the fetched data in a meaningful way in the UI.

Add

http: ^1.0.0
below dependencies

make sure to install cmake for windows

check installation

```
cmake --version
```

```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

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

class HomeRoute extends StatefulWidget {
  const HomeRoute({super.key});

  @override
  State<HomeRoute> createState() => _HomeRouteState();
}

class _HomeRouteState extends State<HomeRoute> {
  List<dynamic> users=[];
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("REST Api"),
      ),
      body: ListView.builder(
        itemCount: users.length,
        itemBuilder: (context,index) {
          final user=users[index];
          final email=user['email'];
          return ListTile(
         title: Text(email),
          );
        },
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          fetchUsers();
        },
        child: Icon(Icons.add),
      ),
    );
  }

  void fetchUsers() async {
    print("Fetch users called:");
    const url = "https://randomuser.me/api/?results=10";
    final uri = Uri.parse(url);
    final response = await http.get(uri);
    final body = response.body;
    final json = jsonDecode(body);
    setState(() {
      users = json['results'];
    });
    print("fetching users completed");
  }
}
```

10. a) Writing Unit Tests for UI Components

How to Write Unit Tests for UI Components in Flutter

Unit tests are useful for verifying the behavior of individual functions, methods, or classes. The `test` package provides the core framework for writing unit tests, and the `flutter_test` package offers additional utilities for testing widgets. Below is a step-by-step guide on how to write unit tests for your Flutter UI components.

1. Add the Test Dependency

To write unit tests in Dart, you need to add the `test` or `flutter_test` dependency. For Flutter apps, use `flutter_test` to test widgets.

Run the following command to add the `test` package as a dev dependency:

```
flutter pub add dev:test
```

2. Create a Test File

Once the dependencies are added, create two files:

- `counter.dart`: This file contains the class you want to test, located in the `lib` folder.
- `counter_test.dart`: This file contains the actual tests and resides in the `test` folder.

Ensure that all test files end with `_test.dart`, which is the convention for the test runner.

Folder Structure:

```
counter_app/
  lib/
    counter.dart
  test/
    counter_test.dart
```

3. Create a Class to Test

Next, define the "unit" (i.e., a class, function, or method) you want to test. For this example, create a `Counter` class in `lib/counter.dart`. The `Counter` class manages an integer value, providing methods to increment and decrement it.

```
class Counter {
  int value = 0;

  void increment() => value++;
  void decrement() => value--;
}
```

4. Write a Test for the Class

Now, write your first unit test in the `counter_test.dart` file. Use the `test` function to define the test and the `expect` function to verify the behavior.

```
// Import the test package and Counter class
import 'package:counter_app/counter.dart';
import 'package:test/test.dart';

void main() {
  test('Counter value should be incremented', () {
    final counter = Counter();
    counter.increment();
    expect(counter.value, 1);
  });
}
```

In this example:

- `test` defines a unit test.
- `expect` is used to assert that the counter value is `1` after calling the `increment()` method.

5. Combine Multiple Tests in a Group

If you have multiple related tests, you can group them together using the `group` function. Grouping tests helps organize them and run all related tests with a single command.

```
import 'package:counter_app/counter.dart';
import 'package:test/test.dart';

void main() {
  group('Test start, increment, decrement', () {
    test('value should start at 0', () {
      expect(Counter().value, 0);
    });

    test('value should be incremented', () {
      final counter = Counter();
      counter.increment();
      expect(counter.value, 1);
    });

    test('value should be decremented', () {
      final counter = Counter();
      counter.decrement();
      expect(counter.value, -1);
    });
  });
}
```

In this example:

- The `group` function groups related tests together.
- The `test` functions check different aspects of the `Counter` class, like its initial value and behaviors after calling `increment()` and `decrement()`.

6. Run the Tests

Once you've written the tests, you can run them in different ways:

a) Using IntelliJ or VSCode

- **IntelliJ**: Open the `counter_test.dart` file and run the tests by going to `Run > Run 'tests in counter_test.dart'`.
- **VSCode**: Open the `counter_test.dart` file and run the tests by going to `Run > Start Debugging`.

b) From the Terminal

To run all tests in the `counter_test.dart` file, run:

```
flutter test test/counter_test.dart
```

To run all tests in a specific group, use the `--plain-name` option:

```
flutter test --plain-name "Test start, increment, decrement"
```

This will run the tests grouped under "Test start, increment, decrement."

7. Conclusion

Unit tests ensure that individual functions and components in your Flutter app work as expected. By using the `test` package for logic and the `flutter_test` package for widget testing, you can validate both the functionality and the behavior of your Flutter app. The steps outlined above show how to write and organize unit tests, run them, and verify that your code behaves correctly.

10 b) Use Flutter's debugging tools to identify and fix issues.

Using Flutter's Debugging Tools to Identify and Fix Issues

Flutter provides several powerful debugging tools that can help you identify and fix issues in your app. Here's a step-by-step guide on how to effectively use these tools:

1. Flutter DevTools

- **Activate DevTools**: To use Flutter DevTools, run your app with the following command:
  Then, activate DevTools with these commands:

  ```
  flutter run
  ```

  ```
  flutter pub global activate devtools
  flutter pub global run devtools
  ```

- **Access DevTools**: Open your app in a Chrome browser and connect it to DevTools. Click on the "Open DevTools" button in the terminal, or manually navigate to http://127.0.0.1:9100/.
- **DevTools Tabs**: DevTools provides several useful tabs like:
  - **Inspector**: Inspect the widget tree and modify properties.
  - **Timeline**: Analyze app performance and frame rendering.
  - **Memory**: Monitor memory usage and detect memory leaks.

2. Flutter Inspector

- **Use in IDE**: The Flutter Inspector is available in IDEs like Android Studio and Visual Studio Code.

- **Toggle Inspector**:
  - In Android Studio: Press `Alt + Shift + D` (Windows/Linux) or `Option + Shift + D` (Mac).
  - In Visual Studio Code: Use the "Flutter: Toggle Widget Inspector" command.
- **Inspect Widget Tree**: The Inspector lets you view and interact with the widget tree, modify widget properties, and observe relationships between widgets in the app.

3. Hot Reload

- **Instant Feedback**: Hot Reload allows you to make changes to the code and immediately see the effect in your app without restarting the entire app.
- **Trigger Hot Reload**:
  - Press `R` in the terminal or click the "Hot Reload" button in your IDE.

4. Debugging with Breakpoints

- **Set Breakpoints**: Use breakpoints to pause the execution of your code at a specific line, allowing you to inspect variables and step through the code.
- **IDE Debugger**: Most IDEs support debugging with breakpoints. Use the debugger to step through your code and identify issues.

5. Logging

- **Print Statements**: Utilize the `print` function to log messages to the console, which can help track variable values and flow.

```
print('Debugging message');
```

- **View Logs**: Logs can be viewed in the terminal or the "Logs" tab in DevTools.

6. Debug Paint

- **Enable Debug Paint**: Debug paint helps visualize the layout and rendering of widgets by showing bounding boxes, lines, and other layout details.

```
void main() {
  debugPaintSizeEnabled = true; // Shows bounding boxes of widgets
  runApp(MyApp());
}
```

- **Other Paint Options**: You can also enable `debugPaintBaselinesEnabled` to show widget baselines.

7. Memory Profiling

- **Memory Tab in DevTools**: Use the "Memory" tab in DevTools to monitor memory usage and identify potential memory leaks.
- **Analyze Memory Usage**: Track object allocations and deallocations to ensure optimal memory usage in your app.

8. Performance Profiling (Timeline)

- **Timeline Tab**: Use the "Timeline" tab in DevTools to analyze the performance of your app. This helps you identify UI jank, slow frames, and performance bottlenecks.
- **Analyze Frame Rendering**: The Timeline provides a detailed view of frame rendering, helping you pinpoint performance issues.

9. Flutter Driver Tests

- **Automated UI Tests**: Flutter Driver allows you to write automated UI tests to simulate user interactions and validate the correctness of your UI.
- **Simulate Interactions**: You can use Flutter Driver to simulate taps, swipes, and other user interactions to ensure that your app responds correctly.

By leveraging these debugging tools, you can effectively diagnose and fix issues in your Flutter app, improving both its functionality and performance.