Experiment 3

Aim: To include icons, images, fonts in Flutter app

Theory:

- Flutter allows developers to enhance the visual appeal and branding of their applications by integrating custom icons, images, and fonts. These assets help in building visually engaging UIs and are essential for delivering a polished user experience. To use these assets in a Flutter project, they must first be added to the project directory and then declared in the pubspec.yaml file under the assets or fonts section.
- **Icons:**
  Flutter provides a rich set of built-in Material icons accessible through the Icons class. These are vector-based and scale well across devices. However, developers can also use custom icons, which can be added using external packages like flutter_svg (for SVG icon support) or by importing them as PNG assets. This flexibility allows for better theming and customization in apps.
- **Images:**
  Images can be sourced either from local assets or the internet. Local images must be placed in the project's assets directory (commonly assets/images/) and listed in the pubspec.yaml file. These can be rendered using the Image.asset() widget. For dynamic or online content, Image.network() can be used to fetch and display images directly from the web. This capability supports responsive and interactive UI design.
- **Fonts:**
  Flutter allows the use of custom fonts. These font files (such as .ttf or .otf) are usually stored in a directory like assets/fonts/, and must be registered in the pubspec.yaml file under the fonts section. Once registered, these fonts can be applied globally through the ThemeData or locally using the TextStyle widget. Custom fonts contribute significantly to app branding and aesthetics.

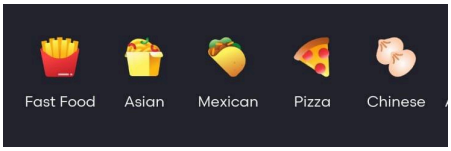
Code: import 'package:flutter/material.dart';

```
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Icons & Fonts Demo',
      theme: ThemeData(
        textTheme: TextTheme(
          bodyText2: TextStyle(fontFamily: 'CustomFont'),
        ),
      ),
      home: CategoryScreen(),
      debugShowCheckedModeBanner: false,
    );
  }
}

class CategoryScreen extends StatelessWidget {
  final List<Map<String, String>> categories = [
    {'image': 'fast_food.png', 'label': 'Fast Food'},
```

```
  {'image': 'asian.png', 'label': 'Asian'},
  {'image': 'mexican.png', 'label': 'Mexican'},
  {'image': 'pizza.png', 'label': 'Pizza'},
  {'image': 'chinese.png', 'label': 'Chinese'},
];

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Color(0xFF1C1C1E),
    appBar: AppBar(title: Text("Categories")),
    body: Padding(
      padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 20),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: categories.map((category) {
          return Column(
            children: [
              Image.asset(
                'assets/images/${category['image']}',
                width: 40,
                height: 40,
              ),
```

```
            SizedBox(height: 6),
            Text(
              category['label']!,
              style: TextStyle(
                fontFamily: 'CustomFont',
                fontSize: 12,
                color: Colors.white,
              ),
            ),
          ]
        }).toList(),
      ),
    ),
  );
```

Output:



Conclusion: This experiment successfully demonstrated how to include and use icons, images, and custom fonts in a Flutter app. With proper asset management and configuration in pubspec.yaml, Flutter enables seamless customization of the UI, allowing for more brand-consistent and visually appealing applications.