

Week 1: Course Introduction, Dart packages, Audio Player Packages and Functions

Presented By : Lect. Mohammad Salim

2023-2024 Spring Term

Advanced
Mobile
Apps IT-
320

Syllabus

Week	Hour	Date	Topic
1	2		Xylophone App- Dart packages, audio player packages and Functions
2	2		Climate App – Location data, http package, async/await and future, exceptions, JSON, and pass data to widgets
3	2		Climate App – Location data, http package, async/await and future, exceptions, JSON, and pass data to widgets
4	2		Chat App: Integrate Flutter Apps with Firebase(Hero animation, Dart mixins, Firestore, authentication, scrolling listview, Dart Stream
5	2		Chat App: Integrate Flutter Apps with Firebase(Hero animation, Dart mixins, Firestore, authentication, scrolling listview, Dart Stream
6	2		Chat App: Integrate Flutter Apps with Firebase(Hero animation, Dart mixins, Firestore, authentication, scrolling listview, Dart Stream
7	2		Midterm Exam
8	2		TodoITApp- State Management and provider package
9	2		TodoITApp- State Management and provider package
10	2		Sensors (Accelerometer, Gyroscope, and Magnetometer)
11	2		Publishing Apps to Apps Store (Android) and (iOS)
12	2		VCS Version Control System Integration (Git and GitHub)
13	2		Localization (Multi-languages support)
14	2		Testing Flutter Apps
15	2		Final Exam
16	2		Final Exam

Prerequisites

- Passing **Mobile Apps** is important because of these some preferred prerequisites for your Advanced Mobile Apps course:
- 1. Fundamental Programming Knowledge:** Students should be comfortable with basic programming concepts such as variables, loops, conditionals, and functions.
 - 2. Introductory Dart Experience:** A foundational understanding of Dart programming language, given that Flutter is Dart-based.
 - 3. Basic Flutter Knowledge:** Familiarity with Flutter's basic widgets and concepts like the widget tree, stateless and stateful widgets, and how to create a simple Flutter application.
 - 4. Understanding of Asynchronous Programming:** Knowledge of `async-await`, `Futures`, and `Streams` in Dart, as they are crucial for handling operations like API calls.
 - 5. Version Control Systems:** Basic understanding of version control with Git and platforms like GitHub for code collaboration and versioning.
 - 6. Software Development Tools:** Experience using IDEs like Android Studio or VS Code for app development.
 - 7. Object-Oriented Programming (OOP):** Since Dart is an object-oriented language, students should be familiar with OOP principles.
 - 8. Basic Command Line Usage:** Comfort with using command-line interfaces, as Flutter often requires running shell commands.

Grading

COURSE EVALUATION CRITERIA

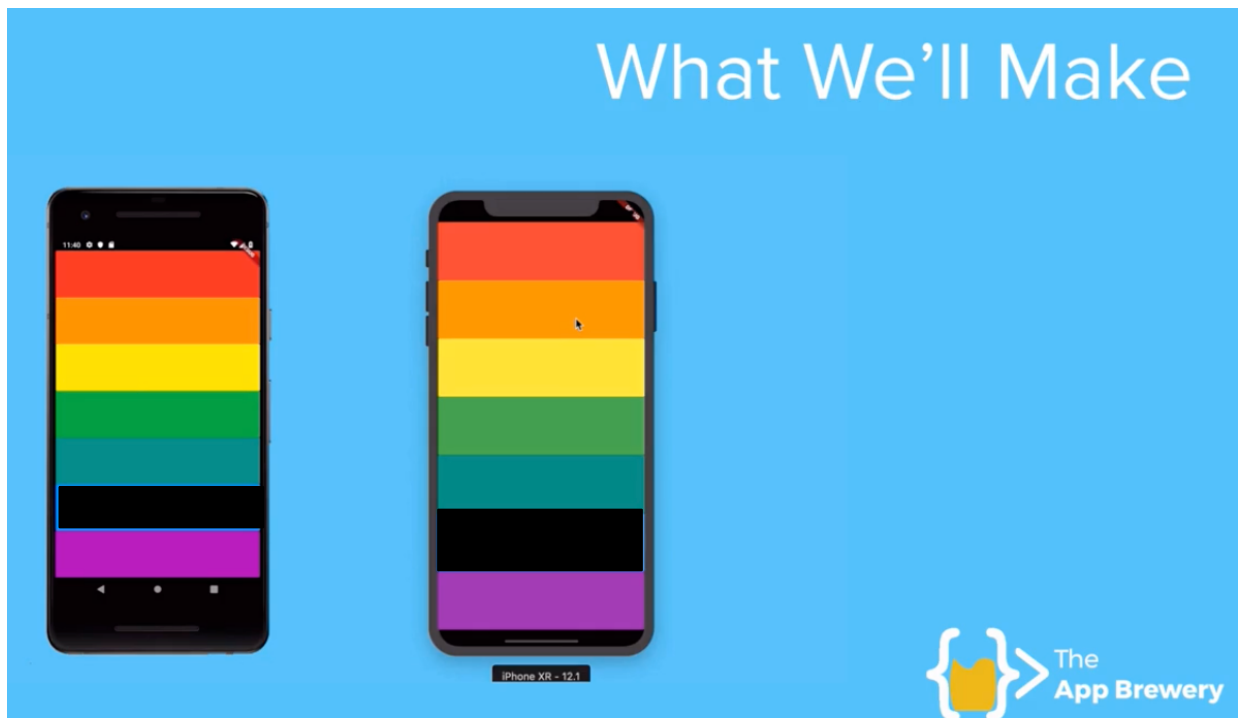
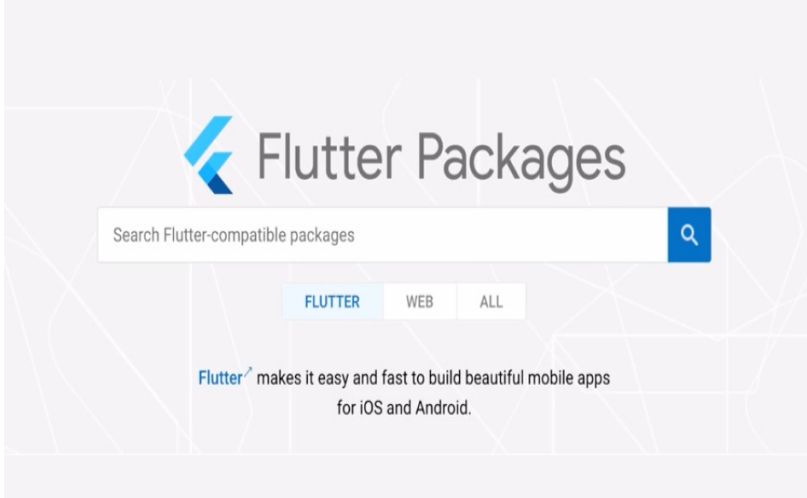
Method	Quantity	Percentage (%)
Quiz	1	10
Homework	1	5
Project	1	15
Midterm Exam	1	30
Final Exam	1	40
Total		100

Examinations: True-False, Fill in the Blanks, Multiple Choices, Short Answers, Matching, , ,



Xylophone

Diving Deeper into Dart Programming

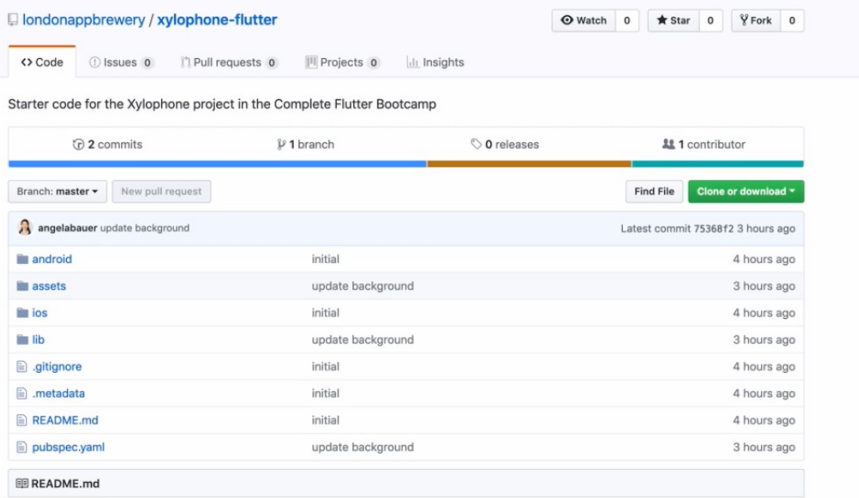



Flutter Packages

Search Flutter-compatible packages

FLUTTER WEB ALL

Flutter makes it easy and fast to build beautiful mobile apps for iOS and Android.



londonappbrewery / xylophone-flutter

Code Issues Pull requests Projects Insights

Starter code for the Xylophone project in the Complete Flutter Bootcamp

2 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Find File Clone or download

angelabauer	update background	Latest commit 75368f2 3 hours ago
android	initial	4 hours ago
assets	update background	3 hours ago
ios	initial	4 hours ago
lib	update background	3 hours ago
.gitignore	initial	4 hours ago
.metadata	initial	4 hours ago
README.md	initial	4 hours ago
pubspec.yaml	update background	3 hours ago
README.md		

What are Flutter & Dart Packages?

- In order to be able to play sounds in our Xylophone App, we are going to use a **Flutter package** for this functionality. But what is a **package** ?
- Flutter packages are open source libraires of code that other people have created which you can incoporate into your project with minimal effort.
- Flutter supports using **shared packages** contributed by other developers to the Flutter and Dart ecosystems. This allows quickly building an app without having to develop everything from scratch.
- **Packages** At a minimum, a Dart package is a directory containing a **pubspec** file.
- Additionally, a package can contain dependencies (listed in the **pubspec**), Dart libraries, apps, resources, tests, images, and examples.
- The pub.dev site lists many packages—developed by Google engineers and generous members of the Flutter and Dart community— that you can use in your

What are Flutter & Dart Packages?

What are Flutter & Dart Packages?

- Flutter and Dart packages significantly enrich the app development ecosystem, offering pre-made solutions that speed up the building process.
- **Pub.dev** hosts these packages, ensuring they undergo a vetting process for reliability and security.
- This system enables developers to utilize community and official resources confidently, integrating sophisticated functionalities with minimal effort.
- Understanding how to effectively search for and evaluate packages is key to leveraging the full potential of this ecosystem in your applications.



Flutter Packages

Search Flutter-compatible packages

FLUTTER WEB ALL

Flutter makes it easy and fast to build beautiful mobile apps for iOS and Android.

Top Flutter-compatible packages

shared_preferences

FLUTTER

Flutter plugin for reading and writing simple key-value pairs. Wraps NSUserDefaults on iOS and SharedPreferences on Android.

url_launcher

FLUTTER

Flutter plugin for launching a URL on Android and iOS. Supports web, phone, SMS, and email schemes.

path_provider

FLUTTER

Flutter plugin for getting commonly used locations on the Android & iOS file systems, such as the temp and app data directories.

image_picker

FLUTTER

Flutter plugin for selecting images from the

cloud_firestore

FLUTTER

Flutter plugin for Cloud Firestore, a cloud-

sqflite

FLUTTER

Flutter plugin for SQLite, a self-contained,

Packages

Packages

What is the difference between a package and a plugin?

- A plugin is a *type* of package—the full designation is *plugin package*, which is generally shortened to *plugin*.
- Existing packages enable many use cases—for example, making network requests ([http](#)), custom navigation/route handling ([fluro](#)), integration with device APIs ([url_launcher](#) and [battery](#)), and using third-party platform SDKs like Firebase ([FlutterFire](#)).
- Searching for packages: Packages are published to [pub.dev](#).

When to use packages VS plugins?

- Use **packages** when you need to **include Dart code**, libraries, or assets that do not require native platform interaction.
- Examples include implementing algorithms, working with **data** structures, or utilizing third-party services with pure Dart code, like **HTTP** requests (**http** package) or date formatting (**intl** package).
- Use **plugins** when you need to interact with native platform functionalities that Dart alone cannot handle, requiring code that uses the native **SDKs** of **iOS**, **Android**, or other **platforms**.
- Examples include accessing the device's **camera** (camera plugin), **GPS** location services (**location** plugin), or integrating with hardware features like **Bluetooth** (flutter_blue plugin) and **sensors**.

Next Slide shows an example of adding a CSS package to an app.

Adding a package!

Adding a package dependency to an app

To add the package, `css_colors`, to an app:

1. Depend on it
 - Open the `pubspec.yaml` file located inside the app folder, and add `css_colors`: under `dependencies`.
2. Install it
 - From the terminal: Run `flutter pub get`.
 - OR**
 - From Android Studio/IntelliJ: Click **Packages get** in the action ribbon at the top of `pubspec.yaml`.
 - From VS Code: Click **Get Packages** located in right side of the action ribbon at the top of `pubspec.yaml`.
3. Import it
 - Add a corresponding `import` statement in the Dart code.
4. Stop and restart the app, if necessary
 - If the package brings platform-specific code (Kotlin/Java for Android, Swift/Objective-C for iOS), that code must be built into your app. Hot reload and hot restart only update the Dart code, so a full restart of the app might be required to avoid errors like `MissingPluginException` when using the package.

The [Installing tab](#), available on any package page on pub.dev, is a handy reference for these steps.

For a complete example, see the [css_colors example](#) below.

Add Package 2

Integrating `http` Package in Flutter

Step 1: Understanding Flutter Packages

Flutter packages enhance app functionality with pre-written code. `http` is a popular package for performing network requests.

Step 2: Adding `http` Package

1. **Find the Package:** Search for the `http` package on pub.dev to get the latest version.
2. **Update `pubspec.yaml`:**

Open `pubspec.yaml` and add the `http` package under dependencies:

```
yaml Copy code
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.3 # Use the latest version
```

3. **Install the Package:**

Run `flutter pub get` in your terminal to install the package.

Step 3: Using the `http` Package in Your App

Import the package in your Dart file:

```
dart Copy code
import 'package:http/http.dart' as http;
```

Step 4: Fetching Data

Create a function to fetch data from the internet:

```
dart Copy code
Future<void> fetchData() async {
  var url = Uri.parse('https://example.com/api/data');
  var response = await http.get(url);
  if (response.statusCode == 200) {
    print('Data: ${response.body}');
  } else {
    print('Request failed with status: ${response.statusCode}.');
  }
}
```

Step 5: Handling Exceptions

Ensure to handle exceptions for reliable app performance.

How to Play Sound Across Platforms?

audioplayers

Platforms

- Android
- iOS
- Linux
- macOS
- Web
- Windows

SDKs

License

Advanced

RESULTS 180 packages

SORT BY SEARCH RELEVANCE

audioplayers

2635 LIKES | 140 PUB POINTS | 100% POPULARITY

A Flutter plugin to play multiple audio files simultaneously

v 5.2.1 (2 months ago) blue-fire.xyz MIT Dart 3 compatible

SDK | FLUTTER | PLATFORM | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS

API result: [audioplayers/audioplayers-library.html](#)

audioplayers_windows

9 LIKES | 130 PUB POINTS | 91% POPULARITY

Windows implementation of audioplayers, a Flutter plugin to play multiple audio files simultaneously

v 3.1.0 (4 months ago) blue-fire.xyz MIT Dart 3 compatible

SDK | FLUTTER | PLATFORM | WINDOWS

audioplayers_web

8 LIKES | 130 PUB POINTS | 92% POPULARITY

Web implementation of audioplayers, a Flutter plugin to play multiple audio files simultaneously

v 4.1.0 (4 months ago) blue-fire.xyz MIT Dart 3 compatible

SDK | FLUTTER | PLATFORM | WEB

API results: [▶ audioplayers_web/audioplayers_web-library.html](#)

assets_audio_player

1071 LIKES | 110 PUB POINTS | 98% POPULARITY

Play music/audio stored in assets files directly from Flutter & Network, Radio, LiveStream, Local files. Compatible with

AudioPlayers

pub v0.20.1 build passing chat 332 online

A Flutter plugin to play multiple simultaneously audio files, works for Android, iOS, macOS and web.

How to Play Sound Across Platforms?

AudioPlayer

An `AudioPlayer` instance can play a single audio at a time (think of it as a single boombox). To create it, simply call the constructor:

```
final player = AudioPlayer();
```



You can create as many instances as you wish to play multiple audios simultaneously, or just to more easily control separate sources.

Sources

Each `AudioPlayer` is created empty and has to be configured with an audio source (and it can only have one; changing it will replace the previous source).

The source (cf. `packages/audioplayers/lib/src/source.dart`) is basically what audio you are playing (a song, sound effect, radio stream, etc), and it can have one of 4 types:

1. **UrlSource**: get the audio from a remote URL from the Internet. This can be a direct link to a supported file to be downloaded, or a radio stream.
2. **DeviceFileSource**: access a file in the user's device, probably selected by a file picker.
3. **AssetSource**: play an asset bundled with your app, by default within the `assets` directory. To customize the prefix, see [AudioCache](#).
4. **BytesSource** (only some platforms): pass in the bytes of your audio directly (read it from anywhere).

AudioPlayers

pub v0.20.1 build passing chat 332 online

A Flutter plugin to play multiple simultaneously audio files, works for Android, iOS, macOS and web.

How to Play Sound Across Platforms?

Advanced Concepts

AudioCache

Flutter does not provide an easy way to play audio on your local assets, but that's where the `AudioCache` class comes into play. It actually copies the asset to a temporary folder in the device, where it is then played as a Local File. It works as a cache because it keeps track of the copied files so that you can replay them without delay.

If desired, you can change the `AudioCache` per player via the `AudioPlayer().audioCache` property or for all players via `AudioCache.instance`.

Local Assets

When playing local assets, by default every instance of `AudioPlayers` uses a [shared global instance of `AudioCache`](#), that will have a [default prefix `"/assets"`](#) configured, as per Flutter conventions. However, you can easily change that by specifying your own instance of `AudioCache` with any other (or no) prefix.

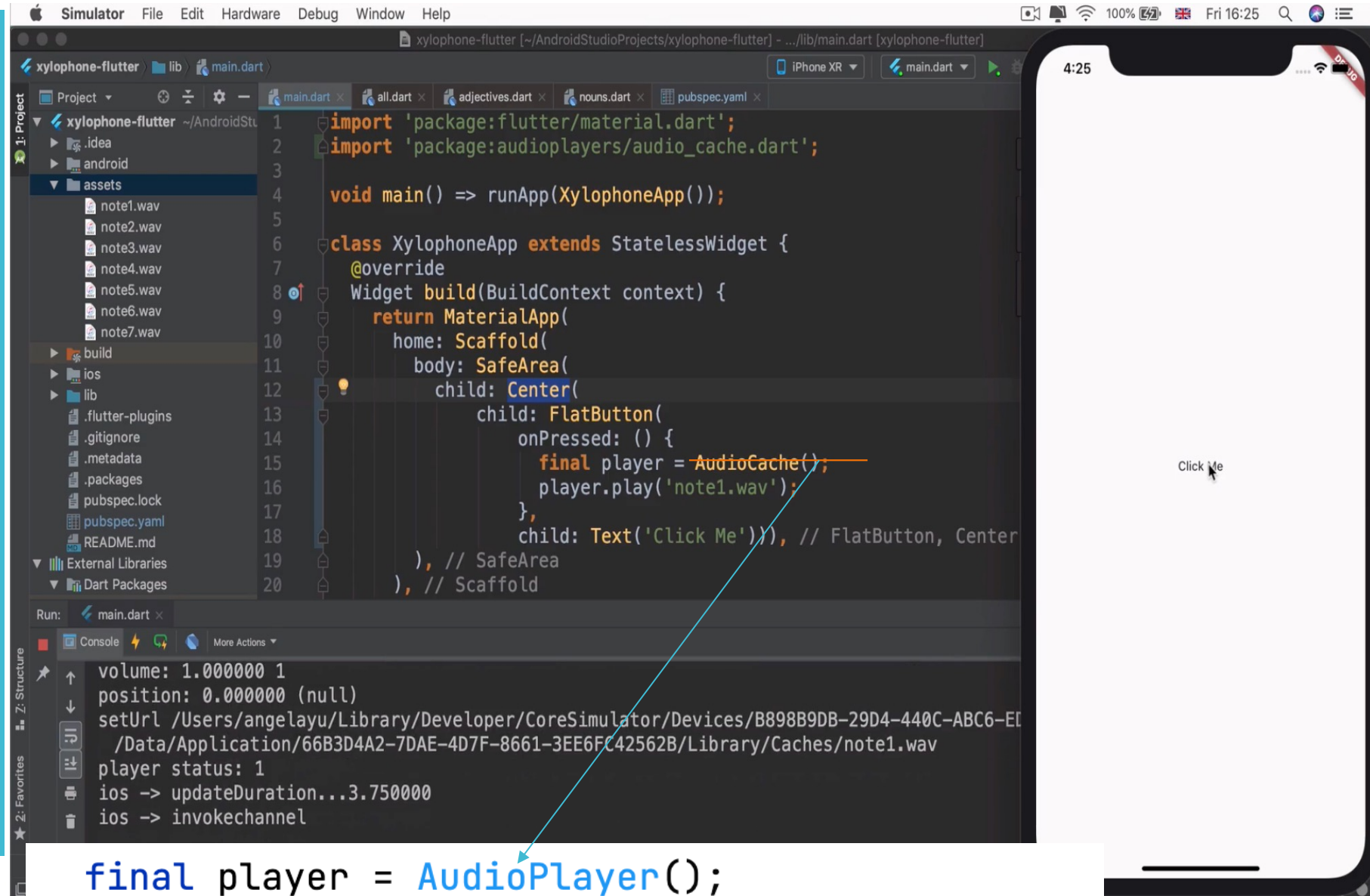
Default behavior, presuming that your audio is stored in `/assets/audio/my-audio.wav` :

```
final player = AudioPlayer();
await player.play(AssetSource('audio/my-audio.wav'));
```

Remove the asset prefix for all players:

```
AudioCache.instance = AudioCache(prefix: '')
final player = AudioPlayer();
await player.play(AssetSource('assets/audio/my-audio.wav'));
```

How to Play Sound Across Platforms?



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

void main() => runApp(XylophoneApp());

class XylophoneApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: SafeArea(
          child: Center(
            child: FlatButton(
              onPressed: () {
                final player = AudioCache();
                player.play('note1.wav');
              },
              child: Text('Click Me')), // FlatButton, Center
          ), // SafeArea
        ), // Scaffold
      ),
    );
```

Run: main.dart x

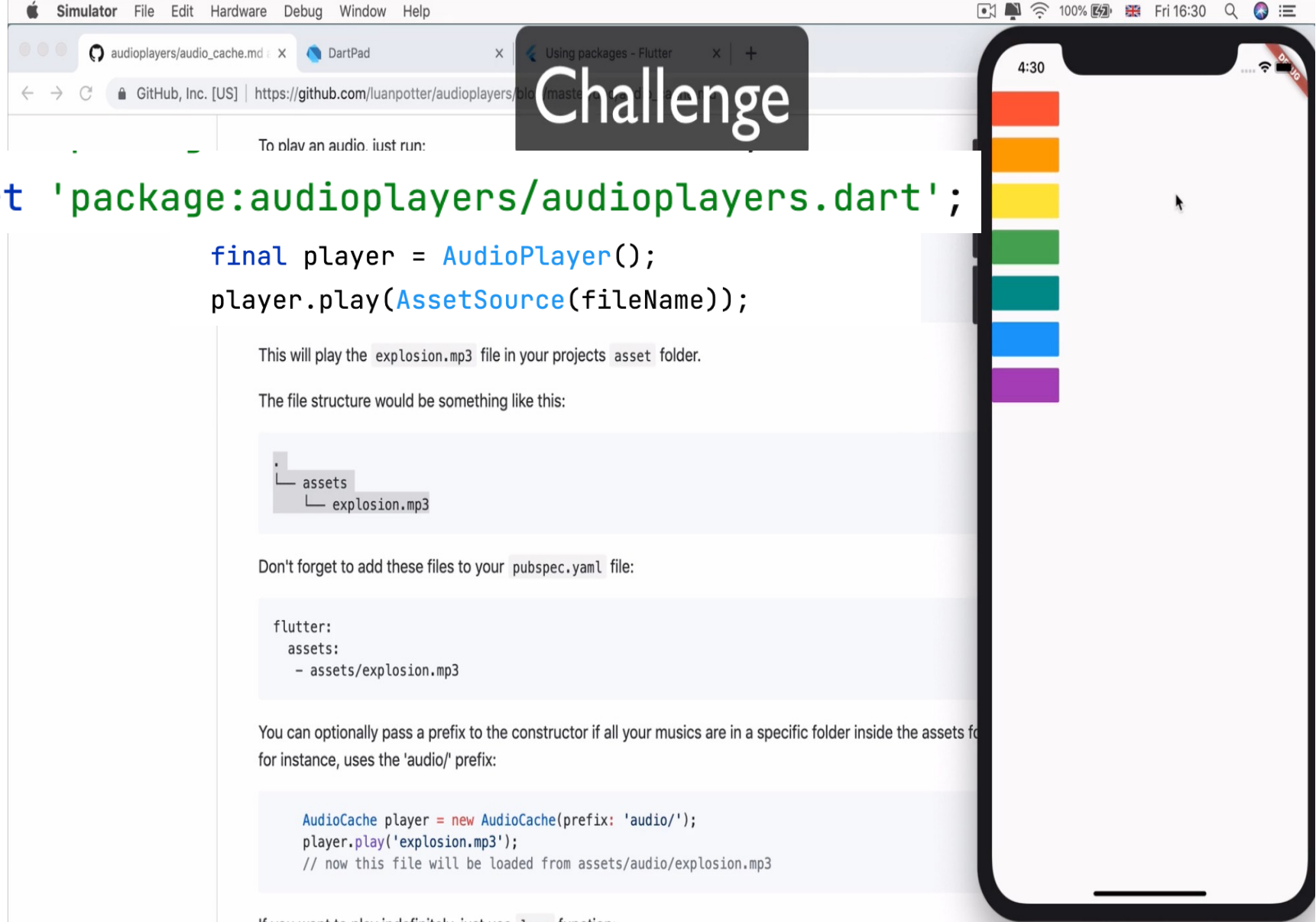
Console

```
volume: 1.000000 1
position: 0.000000 (null)
setUrl /Users/angelayu/Library/Developer/CoreSimulator/Devices/B898B9DB-29D4-440C-ABC6-E1
/Data/Application/66B3D4A2-7DAE-4D7F-8661-3EE6FC42562B/Library/Caches/note1.wav
player status: 1
ios -> updateDuration...3.750000
ios -> invokechannel
```

4:25

Click Me

```
final player = AudioPlayer();
player.play(AssetSource(fileName));
```



Challenge

```
import 'package:audioplayers/audioplayers.dart';  
  
final player = AudioPlayer();  
player.play(AssetSource(fileName));
```

To play an audio, just run:

This will play the `explosion.mp3` file in your projects `asset` folder.

The file structure would be something like this:

```
├── assets  
│   └── explosion.mp3
```

Don't forget to add these files to your `pubspec.yaml` file:

```
flutter:  
  assets:  
    - assets/explosion.mp3
```

You can optionally pass a prefix to the constructor if all your musics are in a specific folder inside the assets folder. For instance, uses the 'audio/' prefix:

```
AudioCache player = new AudioCache(prefix: 'audio/');  
player.play('explosion.mp3');  
// now this file will be loaded from assets/audio/explosion.mp3
```

If you want to play indefinitely, just use `playFrom` function:

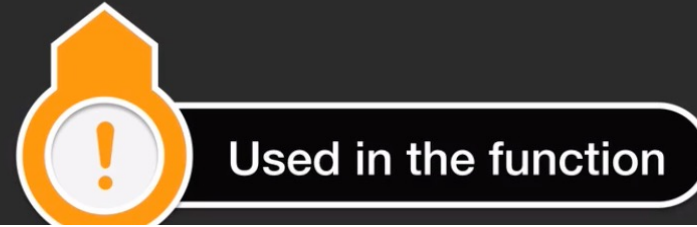
`import 'package:audioplayers/audioplayers.dart';`

How to Play Multiple Sounds?

Dart Functions



```
void playSound(String name) {  
  final AudioCache player = AudioCache();  
  player.play('$name.wav');  
}
```



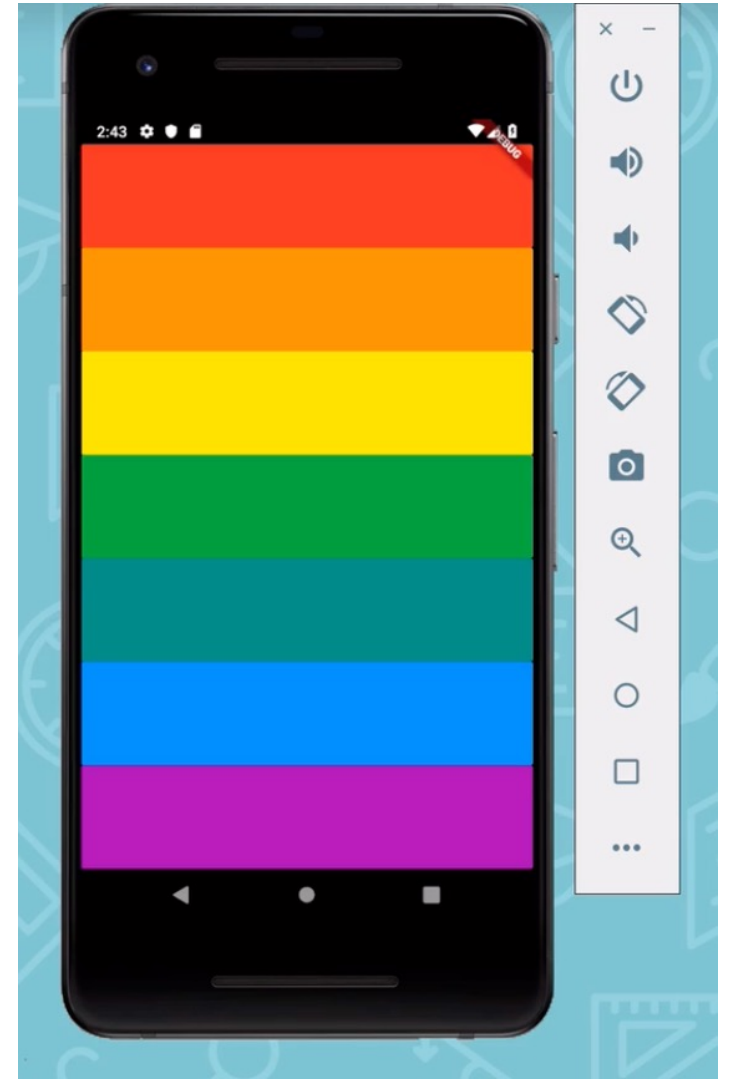
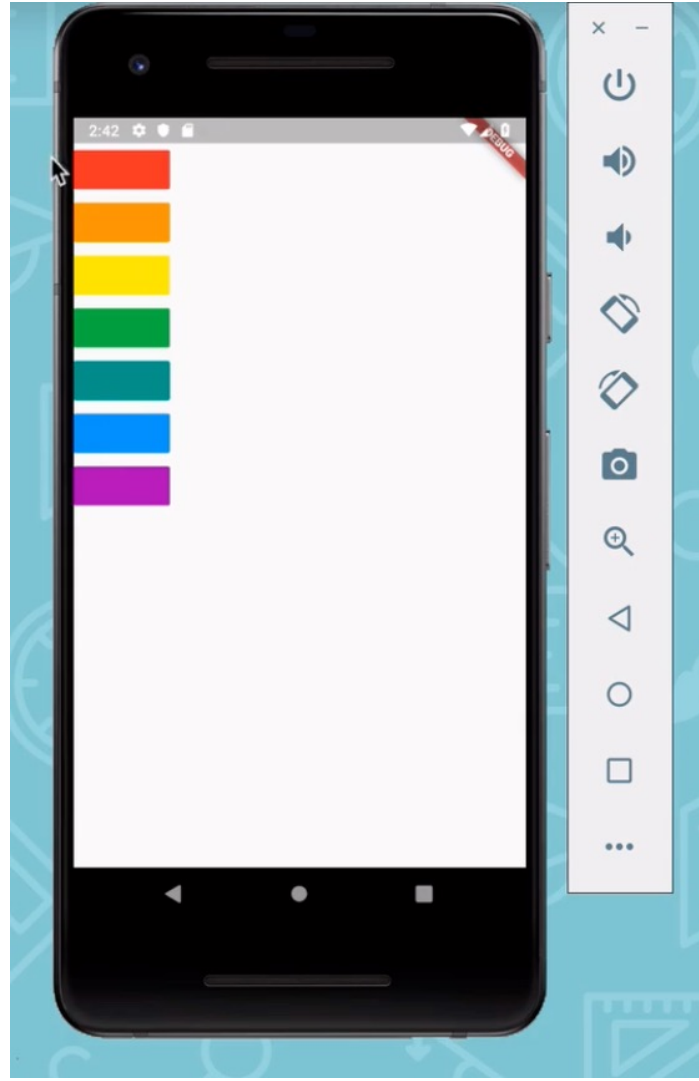
DART

```
void main() {  
  greet(personToGreet: 'Jackie', greeting: 'How do you do');  
}  
  
void greet({String personToGreet, String greeting}) {  
  print('$greeting $personToGreet');  
}
```

Dart Functions

- In the **Xylophone** app, **functions** are utilized to modularize and simplify code, making it more readable and easier to maintain.
- Dart's first-class function support allows functions to be passed as arguments, enhancing code flexibility and reusability.
- By This feature is particularly useful in the app for creating reusable components, such as buttons that play different sounds.
- Abstracting functionality into functions, developers can easily adjust and expand their apps without repetitive code, demonstrating the power of Dart's function capabilities in practical applications.

Challenge



Updating the UI of Our App

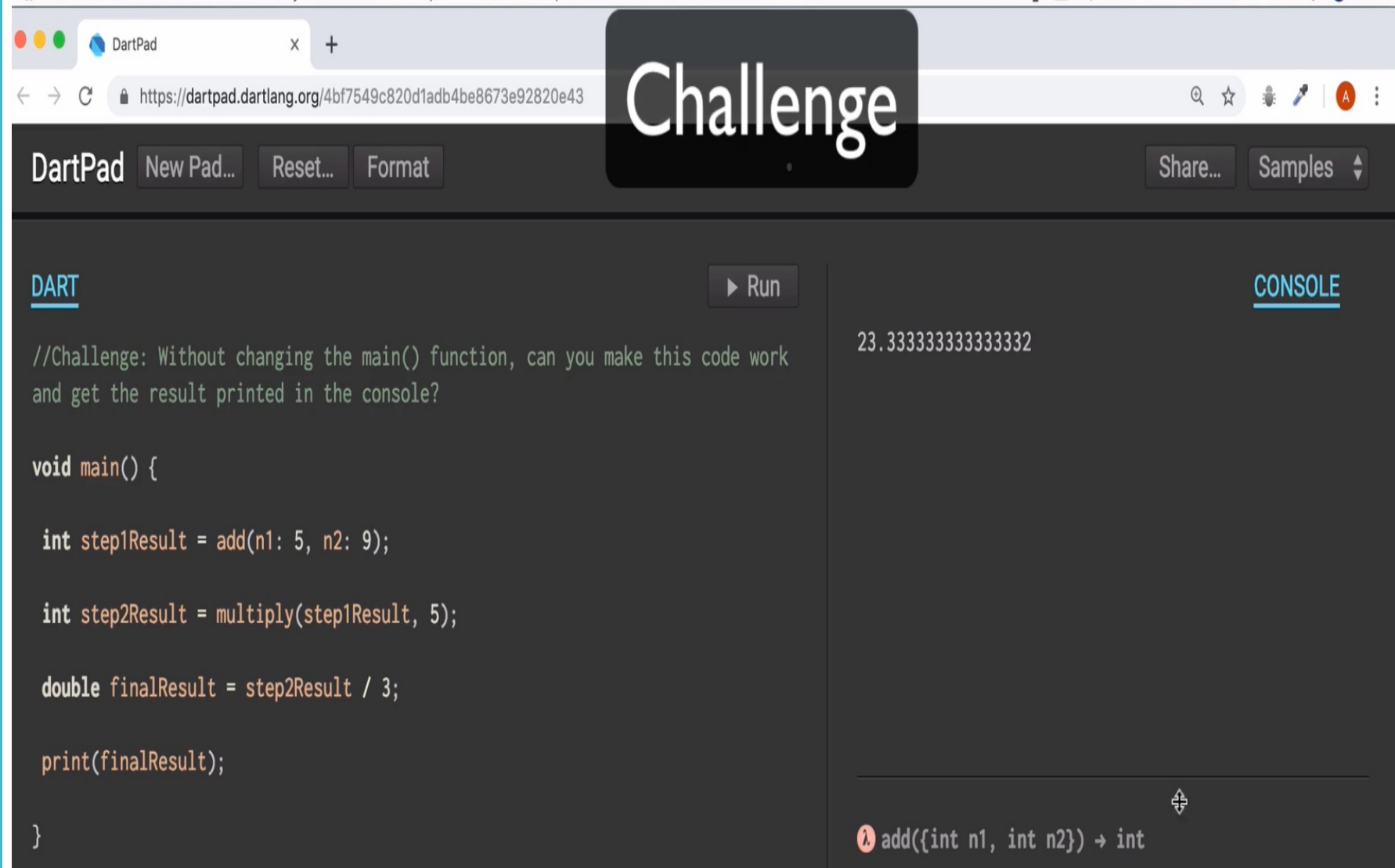
- Try using **Expanded** Widget to improve the design.
- The **Expanded widget** in Flutter is crucial for creating responsive UIs that adapt to various screen sizes.
- It works by letting child widgets flexibly occupy the available space, adjusting their size according to the surrounding layout.
- You should cleanup the code by creating a function to call everytime you want to build a key instead of writing the key code 7 times!

```
xylophone-flutter [-~/Desktop/Flutter Dev/xylophone-flutter] - .../lib/main.dart [xylophone-flutter]
pubspec.yaml x
void buildKey() {
  Expanded(
    child: FlatButton(
      color: Colors.red,
      onPressed: () {
        playSound(1);
      },
    ), // FlatButton
  ); // Expanded
}

@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      backgroundColor: Colors.black,
      body: SafeArea(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[
            buildKey(),
            buildKey(),
            buildKey(),
            buildKey(),
            buildKey(),
            buildKey(),
            buildKey(),
          ], // <Widget>[]
        ), // Column
      ), // Scaffold
    ), // MaterialApp
  );
}
```

The expression here has a type of 'void', and therefore cannot be used.

Dart Functions Challenge



Challenge

DartPad [New Pad...](#) [Reset...](#) [Format](#) [Share...](#) [Samples](#)

[DART](#) [Run](#) [CONSOLE](#)

```
//Challenge: Without changing the main() function, can you make this code work  
and get the result printed in the console?  
  
void main() {  
  
  int step1Result = add(n1: 5, n2: 9);  
  
  int step2Result = multiply(step1Result, 5);  
  
  double finalResult = step2Result / 3;  
  
  print(finalResult);  
  
}
```

23.33333333333332

[add\({int n1, int n2}\) → int](#)

Refactor and Clean Up Our Code

```
12 Expanded buildKey({Color color, int soundNumber}) {
13   return Expanded(
14     child: FlatButton(
15       color: color,
16       onPressed: () {
17         playSound(soundNumber);
18       },
19     ), // FlatButton
20   ); // Expanded
21 }
22
23 @override
24 Widget build(BuildContext context) {
25   return MaterialApp(
26     home: Scaffold(
27       backgroundColor: Colors.black,
28       body: SafeArea(
29         child: Column(
30           crossAxisAlignment: CrossAxisAlignment.stretch,
31           children: <Widget>[
32             buildKey(color: Colors.red, soundNumber: 1),
33             buildKey(color: Colors.orange, soundNumber: 2),
34             buildKey(color: Colors.yellow, soundNumber: 3),
35             buildKey(color: Colors.green, soundNumber: 4),
36             buildKey(color: Colors.teal, soundNumber: 5),
37             buildKey(color: Colors.blue, soundNumber: 6),
38             buildKey(color: Colors.purple, soundNumber: 7),
39             // <Widaget>[]
```

Refactor and Clean Up Our Code



- When refactoring and cleaning up your code, prioritize readability and maintainability.
- This involves organizing code logically, such as grouping related functions together, and choosing descriptive, meaningful names for variables and functions.
- These practices not only make your code more understandable to others (and your future self) but also facilitate easier updates and debugging.
- This approach ensures that your code remains scalable, efficient, and less prone to errors, contributing to the overall quality and longevity of your application.
- In refactoring the Xylophone app, it's advisable to use **widgets** instead of **functions** for creating xylophone keys.
- Because using **widgets** leverages Flutter's reactive style, improving the app's performance and compatibility with the framework's architecture.
- Widgets provide a more structured way to manage the UI and state, enhancing code readability and maintainability.
- They also facilitate the reuse of UI components and make it easier to apply themes and styles consistently across the app.
- This practice aligns with Flutter's design principles, leading to a more efficient and scalable application.

Customize It!



- Now that you've built the app, it's time to customize it to make it your own.
- Explore your creativity in app development by making the Xylophone app uniquely yours.
- Consider adding unique sound sets, vibrant color schemes, and interactive elements to enhance your app.
- This approach not only sets your project apart but also deepens your understanding of Flutter's capabilities, encouraging innovative thinking in design.
- There's a large collection of free sounds at <https://freesound.org/>
- You can download collections of various sounds, e.g. birds/ cats sounds/atmospheric sounds.
- We emphasize the significance of user experience and interface design in customization.
- Maybe you'll want to build a personal sound track to turn your life into a movie.
- Want to investigate a strange movement in your backyard? Click on the tense string ensemble. Hit with a flash of inspiration? Click on the corresponding sound effect.
- Perhaps a bit too much of a narcissistic app idea, I'll leave you to come up with the ideas to customise the app.

Summary



- Today, we dove into **Flutter**, learning more about its **packages** and plugins.
- We saw how to pick and use an **Audio Player package** to add cool features to our apps.
- We also got a closer look at **Dart Functions**, discovering how they make our code cleaner and easier to read, especially when we use **functions** and **parameters** smartly.
- You got to apply what we learned by starting a **Xylophone** app.
- This project isn't just about practicing **coding**; it's about seeing how all these pieces come together in a real app you're building.
- Remember, this is just the **start**. Keep playing around with new **packages**, tweaking your app's **look**, and cleaning up your **code**. And don't forget to join in on the fun with the **Flutter community**.
- Share what you make, pick up tips from others, and maybe help out someone else. Keep being curious and creative – it's the best way to learn and make awesome stuff.