

Help me understand what is needed of me in this assignment

Based on the image you've uploaded, this is a group assignment for a course titled **CAT ONE SPC 2207**. The main objective is to create an automated system that can convert numbers from one number system to another.

Here's a detailed breakdown of what the assignment requires:

1. The Core Project

You need to develop a program that can **automate number system conversions**. This means the program should be able to take a number as input and convert it to a different number system. The program should be able to convert a decimal number to binary, a binary number to hexadecimal, an octal number to decimal, and so on.

2. Group Work

The project must be completed in **groups of five**. This means you'll need to collaborate with four other students to plan, develop, and document the project.

3. Documentation

This is a critical part of the assignment. You're not just building the program; you're also required to provide comprehensive documentation.

- Process Documentation:** This part describes the entire development journey. It should cover your group's plan, the tools and technologies you chose, the different phases of development (planning, coding, testing), and any challenges you faced and how you overcame them. Think of it as a diary of your project's life.
- Program Documentation:** This focuses on the technical details of the program itself. It should include things like:
  - User Manual:** Instructions on how to use the program.
  - Code Explanation:** A detailed description of how the code works, including comments within the code itself.
  - Technical Specifications:** Information about the programming language, libraries, and any other technical requirements.
  - Architecture Diagram:** A visual representation of how the different parts of your program are connected.

4. Collaboration and Deployment

- Instructor Collaboration:** The instruction "Allow me to be a collaborator in your project" means you must give your instructor access to your project. This is likely so they can monitor your progress, review your code, and provide feedback. You will probably need to set up a shared repository on a platform like GitHub or GitLab and invite your instructor as a collaborator.
- Cross-Platform Deployment:** The final requirement is that your application "Should be deployed to android and ios platforms." This means your automated number converter needs to be an application that can run on both Android and iOS devices. You can achieve this using a cross-platform framework like **React Native** or **Flutter**, which allows you to write one codebase for both platforms.

In short, your task is to build a mobile app in a group of five that converts numbers between different systems. You'll need to document your entire process, detail the program's inner workings, get approval from your instructor, and ensure the final app works on both Android and iOS devices.

Which programming language should I use?

Since the assignment requires deployment to both Android and iOS, I advise you to use a **cross-platform framework** like **Flutter** with the **Dart** programming language or **React Native** with **Javascript**.

These frameworks allow you to write a single codebase that can be compiled and deployed on both Android and iOS, which is more efficient than developing separate native apps for each platform. They also have excellent support for building user interfaces and handling number system conversion logic.

Can you write me a simple example of a number system converter in Dart?

Sure, here is a simple example of a number system converter in Dart.

This code provides functions to convert numbers between decimal, binary, and hexadecimal systems. It's a basic command-line application, but the logic can be adapted for a Flutter mobile app.

Dart Code

```
Dart

// Function to convert a decimal number to its binary representation.
String decimalToBinary(int decimal) {
  if (decimal == 0) return '0';
  return decimal.toRadixString(2);
}

// Function to convert a decimal number to its hexadecimal representation.
String decimalToHex(int decimal) {
```

```
    if (decimal == 0) return '0';
    return decimal.toRadixString(16).toUpperCase();
  }

  // Function to convert a binary string to its decimal representation.
  int binaryToDecimal(String binary) {
    return int.parse(binary, radix: 2);
  }

  // Function to convert a hexadecimal string to its decimal representation.
  int hexToDecimal(String hex) {
    return int.parse(hex, radix: 16);
  }

  // Main function to demonstrate the conversions.
  void main() {
    int decimalNumber = 255;
    String binaryString = "11111111";
    String hexString = "FF";

    // Decimal to other bases
    print('$decimalNumber in binary is: ${decimalToBinary(decimalNumber)}');
    print('$decimalNumber in hexadecimal is: ${decimalToHex(decimalNumber)}');

    // Other bases to decimal
    print('$binaryString in decimal is: ${binaryToDecimal(binaryString)}');
    print('$hexString in decimal is: ${hexToDecimal(hexString)}');

    // Cross-conversions (e.g., Binary to Hex)
    int binaryAsDecimal = binaryToDecimal(binaryString);
    print('$binaryString in hexadecimal is: ${decimalToHex(binaryAsDecimal)}');
  }
```

#### Explanation

- `decimalToBinary(int decimal)` : This function takes an integer (decimal number) and uses the built-in Dart method `.toRadixString(2)` to convert it to a binary string. The `2` specifies the radix (base 2).
- `decimalToHex(int decimal)` : Similar to the above, it uses `.toRadixString(16)` to convert the decimal number to a hexadecimal string. `.toUpperCase()` is used to ensure the hexadecimal string is in uppercase.
- `binaryToDecimal(String binary)` : This function takes a binary string and uses `int.parse(binary, radix: 2)` to convert it back to a decimal integer. The `radix: 2` specifies that the input is a binary string.
- `hexToDecimal(String hex)` : This function takes a hexadecimal string and uses `int.parse(hex, radix: 16)` to convert it to a decimal integer. The `radix: 16` specifies that the input is a hexadecimal string.
- `main()` : This function demonstrates how to call the conversion functions and print the results. It also shows a simple example of a cross-conversion (binary to hexadecimal) by first converting the binary string to a decimal integer, and then converting that integer to a hexadecimal string.

This code provides the core logic for your number system converter. For your assignment, you would build a user interface around this logic using a framework like **Flutter**.

Guide me through how to

🌟 I can guide you on how to create the user interface using the Flutter framework, which is an excellent choice for this project.

#### 1. Project Setup

First, you'll need to set up a new Flutter project. Make sure you have the Flutter SDK and a code editor like Visual Studio Code or Android Studio installed.

Open your terminal or command prompt and run the following command:

```
flutter create number_converter_app
```

Then navigate into the newly created project folder:

```
cd number_converter_app
```

#### 2. Design the Layout

A good user interface for your app will have a clean, intuitive layout. A common design pattern would include:

- A **title** at the top.
- A **text input field** for the user to enter the number.
- Drop-down menus** or **radio buttons** to select the "from" (source) and "to" (target) number systems (e.g., Decimal, Binary, Hexadecimal).
- A **button** to trigger the conversion.
- A **display area** to show the converted result.

#### 3. Build the User Interface (UI) with Widgets

In Flutter, the UI is built using **widgets**. You will primarily work within the `lib/main.dart` file. The following is a basic structure for your UI using common Flutter widgets.

##### Core Widgets to Use:

- `Scaffold` : This is the basic layout for a typical app screen. It provides a default app bar, a body, and other common elements.
- `AppBar` : A horizontal bar at the top of the screen, perfect for displaying the app's title.
- `Column` : A widget that arranges its children vertically. This will be the main layout for your input fields, buttons, and result display.
- `Padding` : Used to add space around other widgets, improving the visual appeal.
- `TextField` : A widget that allows the user to type in a number. You can configure it to accept only numbers using `keyboardType: TextInputType.number`.
- `DropDownButton` : A widget to create a drop-down menu for selecting the number system. It takes a list of `DropDownMenuItem` widgets.
- `ElevatedButton` : A button that the user will press to perform the conversion.
- `Text` : A widget for displaying the result of the conversion.

- `StatefulWidget` : Your main UI class should extend `StatefulWidget` because its state (the input number, selected bases, and the result) will change as the user interacts with the app.

**Sample Code Structure:**

Below is a simplified code snippet showing how these widgets fit together. This is a starting point, and you will need to add the conversion logic.

Dart

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Number Converter'),
        ),
        body: const ConversionForm(),
      ),
    );
  }
}

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

  @override
  State<ConversionForm> createState() => _ConversionFormState();
}

class _ConversionFormState extends State<ConversionForm> {
  // State variables to hold user input and conversion results
  String? fromBase;
  String? toBase;
  String inputValue = '';
  String result = '';

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: [
          // Input Field
          TextField(
            keyboardType: TextInputType.number,
            decoration: const InputDecoration(
              labelText: 'Enter number',
            ),
            onChanged: (value) {
              inputValue = value;
            },
          ),

          const SizedBox(height: 20),

          // Dropdowns for base selection
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            children: [
              DropdownButton<String>(
                value: fromBase,
                hint: const Text('From'),
                items: ['Decimal', 'Binary', 'Hexadecimal']
                  .map<DropdownMenuItem<String>>((String value) {
                    return DropdownMenuItem<String>(
                      value: value,
                      child: Text(value),
                    );
                  }).toList(),
                onChanged: (String? newValue) {
                  setState(() {
                    fromBase = newValue;
                  });
                },
              ),
              const Icon(Icons.arrow_right_alt),
              DropdownButton<String>(
                value: toBase,
                hint: const Text('To'),
                items: ['Decimal', 'Binary', 'Hexadecimal']
                  .map<DropdownMenuItem<String>>((String value) {
                    return DropdownMenuItem<String>(
                      value: value,
                      child: Text(value),
                    );
                  }).toList(),
                onChanged: (String? newValue) {
                  setState(() {
                    toBase = newValue;
                  });
                },
              ),
            ],
          ),
        ],
      ),
    );
  }
}
```

```
    ],
  ),

  const SizedBox(height: 20),

  // Conversion Button
  ElevatedButton(
    onPressed: () {
      // Call your conversion logic here and update the 'result' state variable
      // For example:
      // setState(() {
      //   result = convert(inputValue, fromBase, toBase);
      // });
    },
    child: const Text('Convert'),
  ),

  const SizedBox(height: 20),

  // Result Display
  Text(
    'Result: $result',
    style: const TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
  ),
],
),
);
}
```

Thanks. Can you give me a link to a YouTube video tutorial which gives m



Here are a few YouTube video tutorials that provide an excellent overview of Flutter UI development for beginners:

- Flutter Basic Training - 12 Minute Bootcamp by Fireship:** <http://www.youtube.com/watch?v=1xipg02Wu8s>  
This video offers a concise yet comprehensive introduction to Flutter basics, including UI concepts.
- Flutter Basics by a REAL Project by Flutter Guys:** <http://www.youtube.com/watch?v=D4nhaszNW4o>  
This tutorial walks you through Flutter basics by building a real project, which can be very helpful for understanding practical application.
- Flutter Tutorial for Beginners (Playlist) by Net Ninja:** <http://www.youtube.com/playlist?list=PL4cUxeGkcC9jLYyp2Aoh6hcWuxFDX6PBJ>  
This is a full playlist that covers many fundamental aspects of Flutter, including UI design, in detail.
- Flutter UI & Design Tutorials (Playlist) by HeyFlutter.com:** [http://www.youtube.com/playlist?list=PL1WkZqhAdC-i3Vs\\_HBQw9BPT9-\\_zMSun](http://www.youtube.com/playlist?list=PL1WkZqhAdC-i3Vs_HBQw9BPT9-_zMSun)  
This playlist focuses specifically on UI and design in Flutter, which will be directly relevant to creating the user interface for your number converter.

I recommend starting with one of the shorter "bootcamp" style videos to get a quick overview, and then diving into a more comprehensive playlist for detailed learning as you build your app.

 **YouTube** ⓘ  
Flutter UI tutorial for beginners

Flutter Complete E-Commerce App | Flutter UI



**Flutter UI & Design Tutorials**  
HeyFlutter.com

In These **Flutter Tutorials** You **Learn** More About **UI** & Design Of **Flutter** Apps.

guide me on how to do it step by step and all the required tools i need the code which i need to write where i have

◆ This guide breaks down your assignment into the **seven essential steps** required to build, code, and deploy your Flutter number converter, including setting up the required tools.

## 1. Tool and Environment Setup 🛠️

You need three core tools to start developing.

Tool	Purpose	Requirement
<b>Flutter SDK &amp; Dart</b>	The framework and language.	Download and extract to a known location, then add the <code>bin</code> folder to your system's <b>PATH</b> environment variable.
<b>Code Editor</b>	Where you write the code.	<b>VS Code</b> (recommended) or <b>Android Studio</b> . Install the <b>Flutter</b> and <b>Dart</b> plugins/extensions within your chosen editor.
<b>Mobile Platforms</b>	For testing and final deployment.	Run <code>flutter doctor</code> in your terminal. For Android, you need <b>Android Studio</b> to manage the Android SDK and create a virtual device (emulator). For computer with <b>Xcode</b> installed.
<b>Version Control</b>	For group work and instructor collaboration.	Install <b>Git</b> .

- **Final Check:** After installing, open your terminal and run the command: `flutter doctor`. This will check for missing dependencies and guide you on the final setup.

## 2. Project Creation and Structure

1. **Create the Project:** Open your terminal and run the following command to create your Flutter project:

Bash

```
flutter create number_converter_app
cd number_converter_app
```

2. **Organize Files:** Navigate to the `lib` folder. You will use two main files:

- `lib/main.dart`: For the **User Interface (UI)** and state management.
- `lib/conversion_logic.dart`: For all your **Conversion Functions**.

## 3. Conversion Logic (lib/conversion\_logic.dart)

Create the new file `lib/conversion_logic.dart` and add your core functions here. This keeps your business logic separate from the UI.

Dart

```
// lib/conversion_logic.dart

// Helper map to convert string base names to integer radix values
final Map<String, int> baseMap = {
  'Binary': 2,
  'Octal': 8,
  'Decimal': 10,
  'Hexadecimal': 16,
};

// Master function to handle all conversions
String convertNumber(String value, String fromBaseName, String toBaseName) {
  try {
    // 1. Convert Base Name to Radix (e.g., 'Decimal' -> 10)
    int fromRadix = baseMap[fromBaseName]!;
    int toRadix = baseMap[toBaseName]!;

    // 2. Convert input string to an integer (Decimal value is the bridge)
    // The value.trim() is important for cleaning up user input
    BigInt decimalValue = BigInt.parse(value.trim(), radix: fromRadix);

    // 3. Convert the Decimal value to the Target Radix
    String result = decimalValue.toRadixString(toRadix);

    // 4. Format the output (e.g., convert hex to uppercase)
    return result.toUpperCase();
  } catch (e) {
    // Return an error message if the input is invalid for the selected base
    return 'Error: Invalid input for $fromBaseName';
  }
}
```

## 4. User Interface (UI) Implementation (lib/main.dart)

Replace the contents of `lib/main.dart` with the structured UI code below. This code uses a **StatefulWidget** to manage the user selections, input, and result.

Dart

```
// lib/main.dart
```

```

import 'package:flutter/material.dart';
import 'conversion_logic.dart'; // Import your logic file

void main() => runApp(const NumberConverterApp());

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Number Converter',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: const ConverterScreen(),
    );
  }
}

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

  @override
  State<ConverterScreen> createState() => _ConverterScreenState();
}

class _ConverterScreenState extends State<ConverterScreen> {
  // Available conversion bases
  final List<String> bases = ['Binary', 'Octal', 'Decimal', 'Hexadecimal'];

  // State variables for the app
  String? fromBase = 'Decimal'; // Default 'From' base
  String? toBase = 'Binary';    // Default 'To' base
  String inputValue = '';
  String result = '';

  // Controller for the text field
  final TextEditingController _controller = TextEditingController();

  void _performConversion() {
    if (inputValue.isEmpty || fromBase == null || toBase == null) {
      setState(() {
        result = 'Please enter a number and select bases.';
      });
      return;
    }

    // Call the master function from conversion_logic.dart
    String converted = convertNumber(inputValue, fromBase!, toBase!);

    // Update the UI with the result
    setState(() {
      result = converted;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('CAT ONE SPC 2207 Converter')),
      body: Padding(
        padding: const EdgeInsets.all(20.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[
            // --- 1. Input Field ---
            TextField(
              controller: _controller,
              keyboardType: TextInputType.text, // Use text to allow hex characters
              decoration: const InputDecoration(
                labelText: 'Enter Number to Convert',
                border: OutlineInputBorder(),
              ),
            ),
            onChanged: (value) {
              inputValue = value;
            },
          ],
        ),

        const SizedBox(height: 30),

        // --- 2. Base Selection Dropdowns ---
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: <Widget>[
            // 'From' Dropdown
            _buildDropdown('From Base', fromBase, (String? newValue) {
              setState(() {
                fromBase = newValue;
                result = ''; // Clear result on base change
              });
            }),

            const Icon(Icons.arrow_forward_ios, color: Colors.grey),

            // 'To' Dropdown
            _buildDropdown('To Base', toBase, (String? newValue) {
              setState(() {
                toBase = newValue;
                result = ''; // Clear result on base change
              });
            }),
          ],
        ),
      ),
    );
  }
}

```



```
const SizedBox(height: 30),

// --- 3. Convert Button ---
ElevatedButton(
  onPressed: _performConversion,
  style: ElevatedButton.styleFrom(
    padding: const EdgeInsets.symmetric(vertical: 15),
  ),
  child: const Text(
    'CONVERT',
    style: TextStyle(fontSize: 18),
  ),
),

const SizedBox(height: 40),

// --- 4. Result Display ---
const Text('Result:', style: TextStyle(fontSize: 16, color: Colors.grey)),
Text(
  result,
  style: const TextStyle(fontSize: 32, fontWeight: FontWeight.bold),
  softWrap: true,
),
],
),
),
);
}

// Helper Widget for the Dropdowns
Widget _buildDropdown(String label, String? selectedValue, ValueChanged<String?> onChanged) {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(label, style: const TextStyle(fontWeight: FontWeight.bold)),
      DropdownButton<String>(
        value: selectedValue,
        items: bases.map<DropdownMenuItem<String>>((String value) {
          return DropdownMenuItem<String>(
            value: value,
            child: Text(value),
          );
        }).toList(),
        onChanged: onChanged,
      ),
    ],
  );
}
```

## 5. Group Collaboration (Git)

The requirement "Allow me to be a collaborator in your project" is best met using a version control system like **Git** and a hosting service like **GitHub** or **GitLab**.

1. **Initialize Git:** Open your terminal in the main project directory ( `number_converter_app/` ) and run:

Bash

```
git init
git add .
git commit -m "Initial project setup and core converter code"
```

2. **Create a Remote Repository:** Go to GitHub (or GitLab), create a **new private repository**, and follow the on-screen instructions to link your local project to it (usually using `git remote`
3. **Add Collaborators:** In the repository settings on GitHub, invite all your group members **and your instructor** as collaborators. This grants them read access to monitor your progress and requirement.
4. **Push Code:** Push your code to the online repository:

Bash

```
git push -u origin main
```

## 6. Deployment and Submission

1. **Run/Test:** Run your app on a device or emulator:

Bash

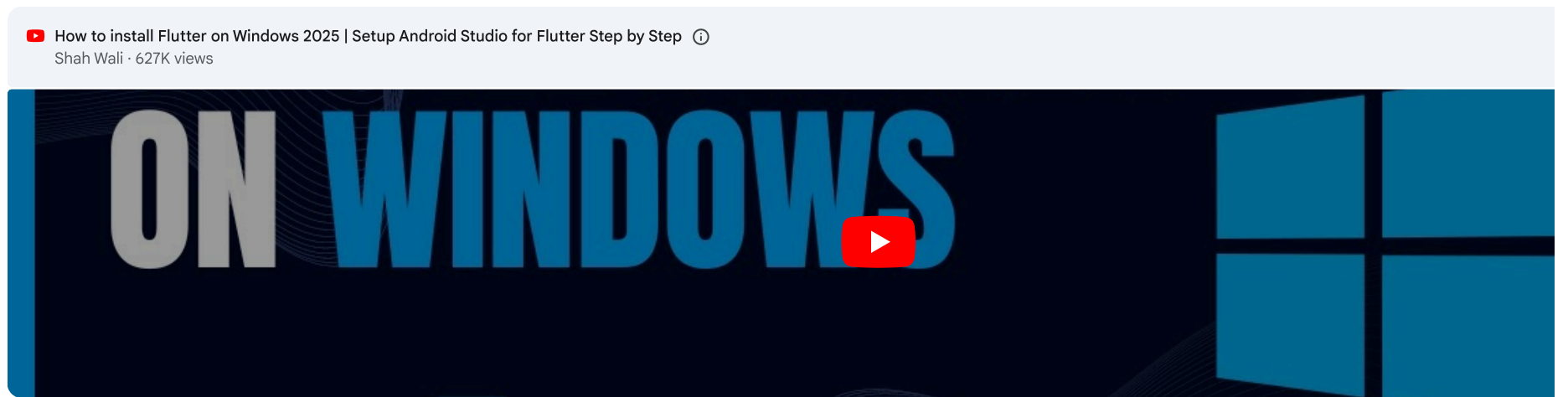
```
flutter run
```

2. **Final Deployment:** Since you are required to deploy to **Android and iOS**, you will use the following commands once your app is complete:
  - **Android APK:** `flutter build apk` (Creates an installable `.apk` file in `build/app/outputs/flutter-apk/` )
  - **iOS IPA:** `flutter build ipa` (Requires Xcode and a Mac, creates an `.ipa` file for distribution).
3. **Documentation:** The final step is writing the **Process Documentation** (how your group worked and collaborated) and the **Program Documentation** (how the code works, user manual, etc) as required by the assignment.

You can find a more visual walkthrough for installing the Flutter SDK and setting up Android Studio in the video below.

The video below gives a detailed, visual walkthrough of the initial **Installation and Setup** of Flutter on a Windows machine, which is the necessary first step for this project.

[How to install Flutter on Windows 2025 | Setup Android Studio for Flutter Step by Step](#)



wher

★ The `lib` folder is a core directory created automatically when you run the command `flutter create number_converter_app`.

You can find it **inside the main project folder** you created.

Here is the typical hierarchy on your computer:

```
your_chosen_location/
├── number_converter_app/ <-- This is your project folder
│   ├── android/
│   ├── ios/
│   ├── lib/ <-- THIS IS THE FOLDER YOU NEED
│   │   └── main.dart
│   ├── test/
│   └── pubspec.yaml
```

You will primarily be working within this `lib` folder, where you should place your Dart files:

1. `lib/main.dart` : Contains the main application structure and the **Flutter UI code**.
2. `lib/conversion_logic.dart` (which you need to create): Contains the **Dart functions** for your number conversion logic.

hot to test and see how the p

💡 That's a great question! Monitoring progress in GitHub is essential for group work, especially since your instructor will be a collaborator.

Here's a step-by-step guide on how you and your group can track development progress using GitHub's built-in features:

## 1. Tracking Code Changes (The Basic Check)

The most fundamental way to see progress is by looking at the **Commit History**.

- **View Commits:** Go to your repository's main page on GitHub and click on the **"Commits"** tab or link (usually near the branch name).
  - **What it shows:** Every time a group member finishes a feature or fix and runs `git commit` and `git push`, a new commit appears here. The commit message explains what they did (binary-to-octal logic, "Fixed UI padding on Android").
  - **How it helps:** You can immediately see **who** did **what** and **when**, providing a clear timeline of development activity.
- **View Files/Deltas:** Click on any specific commit to see the **diff** (or delta). This view shows you exactly which lines of code were **added (green)** and which were **removed (red)** in that update. Check for code completion.

## 2. Monitoring Collaboration (Teamwork Check)

The **Contributors** graph shows who is contributing the most work in terms of lines of code committed.

- **Find it:** On the main repository page, click on the **"Insights"** tab, and then select **"Contributors"**.
- **How it helps:** This visualizes the workload distribution, ensuring that all five group members are actively participating and pushing code, which is important for any group assignment grade.

### 3. Using Projects (Task Management Check)

For a robust way to track feature completion (like a digital Kanban board), use **GitHub Projects**. This is the best method for monitoring your specific features (Binary, Hex, Octal conversion, UI

1. **Create Tasks (Issues):**
  - Go to the **"Issues"** tab and click **"New Issue"**.
  - Create a separate issue for every feature required by the project (e.g., "Implement Decimal to all logic," "Design Base Selection Dropdown," "Create User Manual Documentation Draft
2. **Use a Project Board:**
  - Go to the **"Projects"** tab and create a new **Kanban board**.
  - Set up columns like: **To Do**, **In Progress**, **Review/Testing**, and **Done**.
3. **Track Progress:**
  - Move the issues (tasks) between the columns as your group completes them.
  - **How it helps:** This provides your instructor (and your team) with a high-level, clear visual status of the *entire* project, showing not just what code was written, but what features are fir needs to be done.



4. Viewing the Live App Progress (Visual Check)

Since you are building a mobile application, the fastest way to check visual progress is to have a group member run the app with the latest code.

1. **Pull Latest Code:** Before running, every group member must always pull the latest changes from GitHub:

Bash

git pull origin main

2. **Run the App:**

Bash

flutter run

This ensures you are running the most up-to-date version with all the latest UI changes and logic implementations, allowing you to visually inspect the progress.

id like to run the app

✦ Yes, running the Flutter app in VS Code is the recommended and most common way to develop. It allows you to use features like **Hot Reload** for instant updates.

Here is the step-by-step guide on how to get your app running in VS Code:

1. Prerequisites Check

Before you start, ensure you've completed the following setup steps:

- **Flutter & Dart Extensions:** You must have the **Flutter** and **Dart** extensions installed in VS Code.
- **A Target Device:** You need an Android Emulator, an iOS Simulator (if you're on macOS), or a physical mobile device connected via USB.

2. Launch a Device/Emulator

You must have a device running before you start the app.

For Android (Recommended)

1. Open your terminal and check your device list:

Bash

flutter devices

2. If you have an emulator set up via Android Studio's AVD Manager, start it. For example, to launch the default emulator:

Bash

emulator -avd Pixel\_4\_API\_30

(Note: The exact AVD name will vary).

For iOS (macOS Only)

1. Open your terminal and launch the default simulator:

Bash

open -a Simulator

3. Run the App in VS Code

Once your target device (emulator or physical phone) is visible on your screen, follow these steps in VS Code:

Step 1: Open the Project

1. In VS Code, go to **File** → **Open Folder...**
2. Select the main project folder ( `number_converter_app/` ).

Step 2: Select the Target Device

1. Look at the bottom-right corner of the VS Code window.
2. You will see the name of the currently selected device (it might say "No Device").
3. **Click on the device name.** A list will pop up at the top of the screen showing all available devices (your running emulator, connected phone, etc.).
4. **Select the device** where you want your app to run.

Step 3: Start Debugging/Running

1. Make sure the file `lib/main.dart` is open and active in your editor.
2. Go to the menu bar and select **Run** → **Start Debugging** (or press **F5**).

VS Code will now execute the command `flutter run` in the background, compile your Dart code, and install the app on your selected device. The first run will take a few minutes.

Once it's running, you will see your **"CAT ONE SPC 2207 Converter"** app on the device screen.

#### 4. Use Hot Reload

This is the key feature that makes Flutter development fast.

- After the app is running, make a small change to your code in `lib/main.dart` (e.g., change the title text).
- **Save the file** (Ctrl+S or Cmd+S).

The change should appear almost **instantly** on your running device without restarting the app or losing its current state. This is **Hot Reload** 🔥.

IS THERE AN ONLINE TOOL I CAN USE TO RUN THE COD

◆ Yes, absolutely! The best official online tool for trial purposes is **DartPad**.

It is perfect for testing the core logic of your application without needing to complete the full local setup and compilation.

##### Recommended Tool: DartPad

DartPad is an online editor provided by the Dart team that lets you write, test, and run Dart and basic Flutter code right in your browser.

- **Website:** <https://dartpad.dev/>

##### How to Use DartPad for Your Project

DartPad is excellent for testing your mathematical logic.

##### 1. Test the Core Conversion Logic

You can paste the entire contents of your `lib/conversion_logic.dart` file into DartPad to confirm the functions work correctly before integrating them into the UI.

1. Go to DartPad.
2. Select **"Dart"** from the top-left dropdown (you don't need the full Flutter template for this part).
3. Paste your logic code and add a `main` function to call the logic:

Dart

```
// Paste the convertNumber function and baseMap here:
final Map<String, int> baseMap = {
  'Binary': 2,
  'Octal': 8,
  'Decimal': 10,
  'Hexadecimal': 16,
};

String convertNumber(String value, String fromBaseName, String toBaseName) {
  // ... (Paste the rest of your function code) ...
  try {
    int fromRadix = baseMap[fromBaseName!];
    int toRadix = baseMap[toBaseName!];
    BigInt decimalValue = BigInt.parse(value.trim(), radix: fromRadix);
    String result = decimalValue.toRadixString(toRadix);
    return result.toUpperCase();
  } catch (e) {
    return 'Error: Invalid input for $fromBaseName';
  }
}

void main() {
  // Test 1: Decimal 255 to Hex
  print('Decimal 255 to Hex: ${convertNumber('255', 'Decimal', 'Hexadecimal')}'); // Expected: FF

  // Test 2: Binary 1111 to Decimal
  print('Binary 1111 to Decimal: ${convertNumber('1111', 'Binary', 'Decimal')}'); // Expected: 15
}
```

4. Click **"Run"**. The output will appear in the console, confirming if your logic is sound.

##### 2. Test the Basic Flutter UI

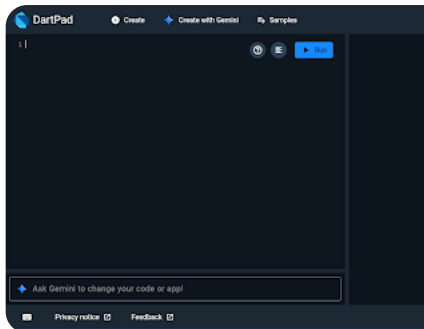
While DartPad can't perfectly replicate the multi-file, production-ready environment of your VS Code project, you can use the **Flutter** template to see if the main UI widgets render correctly:

1. Go to DartPad.
2. Select **"Flutter"** from the top-left dropdown.
3. Paste the contents of your `lib/main.dart` file. (Note: You will need to remove the line that imports the other file: `import 'conversion_logic.dart';` and put the `convertNumber` func single file to make it work in DartPad).
4. Click **"Run"** to see a web-based preview of your app's UI.

##### Other Advanced Online Options

If you need a more complete development environment that fully supports multi-file Flutter projects in the browser, you can explore these tools:

- **FlutLab.io:** A full-featured online Flutter IDE that supports project creation and debugging.
- **GitHub Codespaces:** If your project is already on GitHub, Codespaces can spin up a cloud-hosted VS Code environment with Flutter pre-installed, offering the full VS Code experience in require a paid plan depending on usage).



HOV

- ◆ The image you've provided is **DartPad**, which is an online editor for running Dart and simple Flutter code directly in your browser. Since your assignment is a full Flutter application, DartPad is testing the **number conversion logic** (the Dart functions) separately from the complex UI.

Here is how you should work with DartPad, primarily focusing on testing your core logic:

### 1. Set Up DartPad for Logic Testing

Your project requires robust number conversion functions (the code you'd put in `lib/conversion_logic.dart`). DartPad is the ideal sandbox for ensuring this logic is flawless before integrati

1. **Select Dart:** In the top left corner of DartPad, ensure the dropdown (which might say "Flutter" by default) is set to **"Dart"**. You don't need the full Flutter UI environment to test pure logic.
2. **Paste Your Code:** Copy and paste your number conversion function(s) and any supporting data structures (like the `baseMap` you discussed earlier) into the editor area.
3. **Add a `main` Function:** To execute the code, you need a standard Dart entry point. If you don't already have one, create a `main` function to call your conversion functions with sample da

```
Dart

// Your conversion function code goes here...
String convertNumber(String value, String fromBaseName, String toBaseName) {
  // ... your logic ...
}

void main() {
  // Test 1: Decimal to Binary
  print(convertNumber('10', 'Decimal', 'Binary')); // Should print: 1010
  // Test 2: Hex to Octal
  print(convertNumber('F', 'Hexadecimal', 'Octal')); // Should print: 17
}
```

4. **Run and Debug:** Click the blue **"Run"** button. The results of your `print` statements will appear in the console pane (usually on the right or bottom), allowing you to confirm your conversi
- all cases required by the assignment (Binary, Octal, Decimal, Hexadecimal).

### 2. When NOT to Use DartPad

DartPad is excellent for small trials but **is not suitable** for building your final assignment:

- **Multi-File Projects:** DartPad is mainly a single-file editor. Your final project will have multiple files (`main.dart`, `conversion_logic.dart`, etc.), which DartPad does not handle well.
  - **Deployment:** Your assignment requires deployment to **Android and iOS platforms**. DartPad can only display a web preview, not a build for mobile.
  - **VS Code Environment:** Your instructor expects the project to be managed with Git/GitHub and built using the standard Flutter SDK, which means you should be using **VS Code** (or Androi
- majority of the development.

**Conclusion:** Use DartPad as your **scratchpad** for testing individual functions, but do all your main development and integration within the **VS Code** project environment you've set up locally.

from my vs code, it could not detect a usa

- ◆ When VS Code can't detect a usable device, it typically means either the **Flutter SDK path is wrong**, the **device/emulator isn't running**, or the **necessary platform tools aren't installed** (e.

Here is a step-by-step troubleshooting guide to fix the device detection issue:

#### 1. Run Flutter Doctor

This is the most critical first step. It checks your entire Flutter setup and points out exactly what is missing or misconfigured.

1. Open your **VS Code** terminal (View → Terminal).
2. Run the following command:

```
Bash

flutter doctor
```

3. **Review the Output:** Look for any section marked with an **'X'** or **'!'**.
  - **"Android toolchain" (X or !):** You likely need to accept Android licenses or finish the Android SDK setup.
  - **"VS Code" (!):** The Flutter or Dart extensions might be missing or out of date.
  - **"Connected device" (X):** This simply means no device is currently found, which we'll fix next.

Follow any instructions provided by `flutter doctor` to fix the 'X' or '!' issues first (especially accepting licenses).

#### 2. Verify and Launch a Device

VS Code can only detect devices that are actively running or connected.

Option A: Launch an Android Emulator

1. If you have **Android Studio** installed, open the **AVD Manager** (Tools → AVD Manager) and click the **Play** button next to your desired emulator. Wait until the virtual phone is fully booted u
2. Alternatively, in the VS Code terminal, you can list available emulators and launch one:

Bash

```
flutter emulators
flutter emulators --launch <emulator_id>
# Example: flutter emulators --launch Pixel_4_API_30
```

Option B: Connect a Physical Device (USB)

1. **Enable Developer Options** on your Android phone (Settings → About phone → tap "Build number" 7 times).
2. **Enable USB Debugging** (Settings → System → Developer Options → USB debugging).
3. Plug your phone into the computer. Accept the "Allow USB debugging" prompt on the phone when it appears.
4. Run `flutter devices` in the VS Code terminal to confirm it is recognized.

Option C: Launch an iOS Simulator (macOS Only)

1. Open your terminal and run:

Bash

```
open -a Simulator
```

(Xcode must be installed and configured).

3. Recheck VS Code Device Selector

Once a device or emulator is running, VS Code should detect it.

1. Go to the **bottom-right corner** of the VS Code window.
2. Click on the device selector (it might say "No Device" or "Chrome").
3. A list of detected devices should appear at the top. **Select your newly launched emulator or connected physical phone.**

If the device appears here, you can now run your app by pressing **F5** or clicking **Run** → **Start Debugging** in the menu.

why is it that when i run the fltter doctor command, the cmd exits

💡 This is a common, frustrating issue that usually points to a problem with your **system's environment variables** or a conflict with **background processes** when the Flutter command-line tool : system resources. The fact that a **restart fixes it temporarily** strongly suggests a resource lock or path issue.

Here are the likely reasons and the most effective permanent fixes:

1. Environment Variable Conflicts (The Most Likely Cause)

When you run a command like `flutter doctor`, the Command Prompt ( `cmd` ) uses the system's **PATH** environment variable to find the `flutter` executable and also relies on other variables Dart SDK and Android SDK.

The Problem:

- If the **PATH variable is very long**, or contains invalid entries, `cmd` can struggle to process it, sometimes failing to resolve the correct executable path.
- More often, the issue is that the **Flutter SDK path** or the **Dart SDK path** is not correctly and permanently set, or it points to a location that requires administrator privileges to access, lea immediate exit.

The Fix: Verify and Simplify the PATH

1. **Verify Flutter Path:** Ensure your Flutter binary directory ( `flutter/bin` ) is in the system's PATH.
  - Search for "Environment Variables" in the Windows search bar.
  - Click **Environment Variables...**
  - Under **System variables**, find and select **Path**, then click **Edit**.
  - Make sure there is a path that looks like this (adjusting for your installation location):

C:\src\flutter\bin

2. **Verify Android SDK Path:** Ensure the following Android SDK paths are also present (these are often the ones that cause issues if Flutter can't find them):

C:\Users\YourUser\AppData\Local\Android\sdk
C:\Users\YourUser\AppData\Local\Android\sdk\platform-tools

2. Resource Lock or Stale Process

After an initial restart, the operating system releases all process locks. When you successfully run Flutter commands, the OS locks certain files or background Dart/Flutter processes start runn

The Problem:

- Sometimes, an anti-virus program, a corporate security scanner, or a previous run of a Flutter tool (like DevTools or a running emulator process) doesn't properly release the lock on a criti

- When you try to run `flutter doctor` again, it hits that locked resource and freezes or immediately crashes the `cmd` session.

**The Fix: Use a Different Terminal**

Instead of using the standard Command Prompt ( `cmd` ), use one of these more modern and robust terminals:

- **VS Code's Integrated Terminal (Recommended):** This is the easiest. VS Code handles the environment paths more reliably.
- **PowerShell:** Windows PowerShell or the newer PowerShell 7.
- **Git Bash:** A Unix-style terminal provided by Git.

If the command runs successfully in one of these alternatives without restarting, the issue is definitely related to the stability of the classic Windows `cmd` environment, and you should switch p

### 3. Pending Licenses or Updates

If Flutter couldn't complete a setup step (like accepting Android licenses), it might be attempting the action and failing immediately in a way that causes the `cmd` to close.

**The Fix: Rerun License Acceptance**

Run the license command directly to make sure that step is completed:

```
Bash

flutter doctor --android-licenses
```

If the terminal closes during this step, it confirms the license interaction is the source of the crash, and you should try running this command as an **Administrator** (Right-click → Run as admin

```
PS C:\Users\Sharon\number_converter_app> flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.35.4, on Microsoft Windows [Version 10.0.
[!] Windows Version (Problem detected with Windows installation)
    ! Get-Process failed to complete
[√] Android toolchain - develop for Android devices (Android SDK versio
[√] Chrome - develop for the web
[!] Visual Studio - develop Windows apps (Visual Studio Build Tools 2019
    X The current Visual Studio installation is incomplete.
    Please use Visual Studio Installer to complete the installation or rein
[√] Android Studio (version 2025.1.3)
[√] VS Code (version 1.104.2)
[√] Connected device (3 available)
[√] Network resources

! Doctor found issues in 2 categories.
PS C:\Users\Sharon\number_converter_app>
how can i ix this issues
```

✦ Your `flutter doctor` output shows two main issues that need to be addressed. The good news is that your Flutter installation itself, Android Studio, and VS Code appear to be correctly confi development.

Here is how you can fix the two issues:

#### 1. Fix the Windows Version Issue ( `Get-Process` Failure)

The `Get-Process failed to complete` message is often a symptom of PowerShell or system script access being restricted, likely due to antivirus software or elevated security settings. The i hangs suggests a permissions issue when running system diagnostics.

**The Fix**

Run the `flutter doctor` command again, but this time, ensure you run your terminal (**PowerShell** or **Command Prompt**) with **Administrator privileges**.

1. Close your current PowerShell/CMD window.
2. Search for "PowerShell" or "Command Prompt" in the Windows Start Menu.
3. **Right-click** on the app.
4. Select **"Run as administrator."**
5. Navigate back to your project directory:

```
PowerShell

cd C:\Users\Sharon\number_converter_app
```

6. Run `flutter doctor` again.

Running as an administrator should resolve any restrictions preventing the tool from completing the `Get-Process` task, which is used to check background system status.

#### 2. Fix the Visual Studio Issue (Incomplete Installation)

The **Visual Studio** issue prevents you from building desktop Windows apps, but **it will not prevent you from building your required Android and iOS mobile apps**.

Since your assignment specifically requires deployment to **Android and iOS platforms**, this step is technically **optional** for project completion. However, if you want a clean `flutter doctor` develop for Windows later, here is the fix:

The Fix

You need to use the **Visual Studio Installer** to ensure the necessary workload is installed.

1. Search for "**Visual Studio Installer**" in the Windows Start Menu and launch it.
2. Find your installed version of Visual Studio (or Build Tools) and click the "**Modify**" button.
3. In the **Workloads** tab, ensure the following workload is checked:
  - **"Desktop development with C++"**
4. If it's not checked, check it and click the "**Modify**" button to start the installation.
5. Once the installation is complete, close the Installer and run `flutter doctor` (as Administrator) one last time.

After these steps, your `flutter doctor` output should show no issues, and you'll be fully ready to develop your app for Android and iOS.

can any of the two categories of errors hinder m

- ◆ Yes, one of the two errors you found with `flutter doctor` **could hinder your mobile development process**, while the other is unlikely to be a problem for this specific assignment.
- Here is a breakdown of how each error category impacts your work:

1. The Windows Version Issue (Potential Barrier)

Error: `[!] Windows Version (Problem detected with Windows installation) → ! Get-Process failed to complete`

This error suggests a problem with your environment's ability to run system diagnostic checks, likely due to a permissions issue, often caused by antivirus software or security settings.

- **How it can hinder you:** While this error might not directly stop your app from building, it often indicates deeper system instability or a process lock that can interfere with the Flutter tool complete its process checks, it might:
  - **Fail to find the Android SDK/Emulator** even if it's running.
  - **Cause commands to crash** unexpectedly (as you experienced when the CMD exited).
  - **Hinder debugging** or the use of specific Flutter tools.
- **Action:** You should prioritize fixing this by running `flutter doctor` **as an Administrator** in PowerShell, as recommended previously. This is the most crucial fix for ensuring a stable deve

2. The Visual Studio Issue (Not a Barrier for Mobile)

Error: `[!] Visual Studio - develop Windows apps → X The current Visual Studio installation is incomplete.`

This error relates to the components needed to build **Windows desktop applications**.

- **How it can hinder you:** **It will not hinder your work** on this assignment, as your project specifically requires deployment to **Android and iOS platforms**.
- **Action:** You can safely **ignore this error** for the purpose of completing your current mobile app project. If you plan to deploy your Flutter app to Windows in the future, you would need to Installer to fix it.

from my vs code, it could not detect a usa

- ◆ When VS Code can't detect a usable device, the problem usually stems from an inactive emulator/device, a path issue, or missing platform components.
- Here is the essential, step-by-step process to fix device detection:

1. Run Flutter Doctor

First, you must check your overall Flutter environment setup. This will tell you exactly what is missing or broken.

1. Open your **VS Code terminal** ( `View` → `Terminal` ).
2. Run the command:

Bash

flutter doctor

3. **Address Errors:** Look for any section marked with an 'X' or '!'. For example, your previous output had issues with the Windows setup and Visual Studio, which can destabilize the environm
  - If `flutter doctor` hangs or exits ( `Get-Process failed` ), try running the terminal **as an administrator** to overcome permission issues.
  - If it mentions licensing problems, run: `flutter doctor --android-licenses`

2. Launch or Connect a Device

VS Code can only detect devices that are actively running or connected.

For Android Emulator (Recommended)

1. If you have **Android Studio** installed, open the **AVD Manager** and click the **Play** button next to your virtual device.
2. Alternatively, you can list and launch an emulator directly from the VS Code terminal:

Bash



```
# List all available emulators
flutter emulators

# Launch a specific one (e.g., Pixel_4_API_30)
flutter emulators --launch <emulator_id>
```

Wait until the virtual phone is fully booted up.

For Physical Mobile Device

- 1. **Android:** Enable **Developer Options** and **USB Debugging** on your phone (usually in the Settings → System or Settings → About Phone menu). Plug the phone into your computer and a "debugging" prompt.
- 2. **Verify Detection:** Run `flutter devices` in your terminal to confirm your device is listed.

3. Select the Device in VS Code

Once your emulator or physical device is visibly running or connected, VS Code should detect it.

- 1. Look at the **bottom-right corner** of the VS Code window. This area displays the currently selected device.
- 2. **Click on the device name** (it might still say "No Device" or "Chrome").
- 3. A prompt will appear at the top of the screen showing all detected devices. **Select your running device or emulator** from the list.

Once selected, you can run your app by pressing **F5** or clicking **Run → Start Debugging** in the menu.

this is the response : PS C:\Users\Sharon\number\_convert

>>

1 available emulator:

Id	Name	Manufacturer	Platform
Medium_Phone_API_36.1	Medium Phone API 36.1	Gener	

To run an emulator, run 'flutter emulators --launch <emula  
To create a new emulator, run 'flutter emulators --create [  
  
You can find more information on managing emulators at t  
<https://developer.android.com/studio/run/managing-avds>  
<https://developer.android.com/studio/command-line/avdr>  
PS C:\Users\Sharon\number\_converter\_app>

That's excellent progress! Your command line successfully recognized and listed a usable Android emulator: `Medium_Phone_API_36.1`.

The reason VS Code couldn't detect a device earlier is that **the emulator was not yet running**. VS Code only detects devices that are active and available.

Here is exactly how to launch that emulator and make it available to VS Code:

1. Launch the Emulator from PowerShell

Use the command provided in the output to start the emulator.

- 1. In your PowerShell terminal (the one you used to run `flutter emulators`), execute the launch command, substituting the `<emulator id>` with the ID you found:

PowerShell

```
flutter emulators --launch Medium_Phone_API_36.1
```

- 2. **Wait:** A separate window should open, and the Android operating system will boot up. This process can take several minutes the first time. The emulator is not fully detected until the hom

2. Verify Detection in VS Code

Once the emulator is fully loaded and visible on your screen:

- 1. Look at the **bottom-right corner** of your VS Code window.
- 2. The device selector (which may have said "No Device" before) should now display: `Medium Phone API 36.1 (mobile)`
- 3. If the device name doesn't appear immediately, click on the device selector, and choose the emulator from the list at the top of the screen.

3. Run Your Application

With the device selected, you can now run your Flutter app from VS Code:

- 1. Make sure the main file, `lib/main.dart`, is open and active.
- 2. Press **F5**, or go to **Run → Start Debugging**.

The application will be compiled and installed onto the running emulator, and you will see your app's UI.

the device is

If the emulator is fully running but still not detected by VS Code, the issue is likely due to a problem with the **ADB (Android Debug Bridge) connection**. ADB is the communication bridge betw machine (VS Code/Flutter) and the Android device/emulator.

Here is a targeted plan to fix this connection problem:

### 1. Stop and Restart ADB

The most common fix is to manually stop and restart the ADB server to clear any hung connections.

1. Open your **VS Code terminal** or a separate **PowerShell** window.
2. Run the following commands sequentially:

- **Stop the ADB server:**

```
PowerShell
```

```
adb kill-server
```

- **Start the ADB server and check devices:**

```
PowerShell
```

```
adb devices
```

3. **Review Output:** You should see output similar to this, listing your running emulator:

```
List of devices attached
Medium_Phone_API_36.1    device
```

If your device is listed with the status `device` , the connection is good. If it's listed as `unauthorized` , check your emulator screen for a prompt asking you to allow USB debugging and `no permissions` or **nothing is listed**, proceed to the next step.

### 2. Force Flutter to Detect Devices

Even if ADB is running correctly, sometimes the Flutter tool itself needs a nudge.

1. Keep the emulator running.
2. In your VS Code terminal, run the command to list connected devices, forcing Flutter to search again:

```
PowerShell
```

```
flutter devices
```

3. **Check VS Code:** Once the terminal command successfully lists your device, go back to the **bottom-right corner** of the VS Code window and click the device selector. Your emulator should show up in the list.

### 3. Troubleshoot Path and Environment

If the issue persists, your system might be confused because multiple ADB installations (e.g., one from Android Studio and one in the Flutter SDK) are running simultaneously.

1. **Stop everything:** Close the emulator window and the VS Code window.
2. **Run as Administrator:** Relaunch **VS Code** by **right-clicking** the icon and selecting **"Run as administrator."** This eliminates most permission-related access issues.
3. **Relaunch Emulator:** Open the terminal in the administrator VS Code window and launch the emulator again:

```
PowerShell
```

```
flutter emulators --launch Medium_Phone_API_36.1
```

This sequence ensures that the development environment (VS Code) has the highest privileges to communicate with the emulator and should resolve the detection problem.

```
PS C:\Windows\System32> adb kill-server
cannot connect to daemon at tcp:5037: cannot connect to 127.0.0.1:5037: No connection could be made because the target machine actively refused it.
PS C:\Windows\System32>
```

✦ The message you're receiving, `cannot connect to daemon at tcp:5037... No connection could be made because the target machine actively refused it. (10061)` , indicates one of two things is happening:

1. **The ADB server was never running** to begin with (so there's nothing to kill).
2. **A Firewall or Antivirus program** is blocking the connection on port `5037` , which is the port that the Android Debug Bridge (ADB) uses for communication.

Since your emulator is running but not connecting, it's highly likely to be the second issue: a **firewall block**.

Here is the plan to resolve the issue and establish the connection:

#### 1. Check for Running ADB Process

Before troubleshooting the firewall, let's check if the ADB process is active from a different path, which sometimes happens if multiple tools install their own copies of ADB.

1. Open your **VS Code Terminal** or **PowerShell** (use your usual project directory, not `C:\Windows\System32` ).
2. Check for the process:

PowerShell

```
tasklist /fi "imagename eq adb.exe"
```

- **If you see `adb.exe` in the list:** Manually end the task.

PowerShell

```
taskkill /f /im adb.exe
```

- **If you do NOT see `adb.exe` :** The process is not running. Proceed to the next step.

## 2. Whitelist ADB in the Firewall

If the ADB server won't start, the firewall is the most probable blocker. You need to allow the ADB executable ( `adb.exe` ) through your firewall.

1. **Open Windows Defender Firewall Settings:** Search the Windows Start Menu for "Windows Defender Firewall with Advanced Security" and open it.
2. **Create an Outbound Rule:**
  - In the left pane, click "**Outbound Rules.**"
  - In the right pane, click "**New Rule...**"
3. **Configure the Rule:**
  - **Rule Type:** Select "**Program.**" Click Next.
  - **Program Path:** Click "**Browse...**" and navigate to your Android SDK platform-tools folder. The path is typically:

C:\Users\YOUR\_USERNAME\AppData\Local\Android\sdk\platform-tools\adb.exe
  - **Action:** Select "**Allow the connection.**" Click Next.
  - **Profile:** Ensure all three options (**Domain, Private, Public**) are checked. Click Next.
  - **Name:** Give the rule a clear name, like "**ADB Bypass.**" Click Finish.
4. Repeat this process for "**Inbound Rules**" as well, using the same path for `adb.exe` .

## 3. Restart ADB and Force Detection

After adjusting the firewall, you must restart ADB and force Flutter to check for devices again.

1. Keep your emulator running.
2. In your VS Code terminal, run:

PowerShell

```
adb start-server
flutter devices
```

3. **Check VS Code:** Your emulator should now appear in the list and be detectable by VS Code in the bottom-right corner.

If you are still unable to connect, your antivirus software may also have a built-in firewall. Temporarily disabling the antivirus while you run the `flutter devices` command can confirm if it is t

```
PS C:\Users\Sharon\number_converter_app> tasklist /fi "imagename ec

Image Name          PID Session Name  Session#  Mem Usage
=====
adb.exe             3464 Console      1    12,192 K
PS C:\Users\Sharon\number_converter_app>
```

💡 This is a great finding! The output confirms that the **ADB (Android Debug Bridge) server is currently running** under the process ID (PID) **3464**.

However, since your previous attempts to connect failed with the "actively refused" error, the running `adb.exe` process is likely **stuck, corrupt, or running with incorrect permissions**, preventing communicating with your running emulator.

Here is the precise sequence of commands you need to execute now to reset the connection:

### 1. Terminate the Stuck ADB Process

You need to forcefully kill the identified `adb.exe` process to clear its stuck state.

In your PowerShell terminal, run the following command, which uses the Image Name to stop the process:

PowerShell

```
taskkill /f /im adb.exe
```

The `/f` flag forces termination, and the `/im adb.exe` flag specifies the Image Name.

### 2. Verify ADB is Terminated

Run `tasklist` again to confirm the process is gone:

```
PowerShell

tasklist /fi "imagename eq adb.exe"
```

The response should indicate that the process is **not found**.

### 3. Restart the ADB Server

Now that the old process is gone, manually start a fresh, clean ADB server. This often correctly initializes the connection to the running emulator.

```
PowerShell

adb start-server
```

You should receive a message like "daemon started successfully" or no error message, which is a good sign.

### 4. Check for Connected Devices in Flutter

Finally, ask Flutter to check its environment. Since the emulator is already running, Flutter should now see it.

```
PowerShell

flutter devices
```

#### What to look for:

- Your emulator, `Medium_Phone_API_36.1`, should be listed under connected devices.
- Once it appears in this list, go back to VS Code. The device selector in the bottom-right corner should automatically update, and you can then press **F5** to run your app.

```
PS C:\Users\Sharon\number_converter_app> adb start-server
adb: The term 'adb' is not recognized as a name of a cmdlet, function, script file
Check the spelling of the name, or if a path was included, verify that the path is
PS C:\Users\Sharon\number_converter_app>
```

✦ The error message, "**adb: The term 'adb' is not recognized...**", means that your PowerShell terminal doesn't know where the `adb.exe` program is located. Even though the previous `tasklist` the necessary environment variable path is missing or incorrect in your current session.

Here is the most reliable way to fix this, without needing to permanently change your system's environment variables right now:

#### Temporary Fix: Execute ADB Using the Full Path

You can bypass the PATH issue by running the `adb` commands using the full, explicit path to the executable.

1. **Locate `adb.exe`** : The `adb.exe` file is located inside the Android SDK's `platform-tools` folder. On Windows, this is usually at a path similar to the following (you'll need to substitute you

```
C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools\adb.exe
```

2. **Execute the Commands:** Copy and paste the full path into your PowerShell terminal, followed by the command you want to run.

##### A. Stop the Server (Kill the Old Process)

Since your previous step confirmed the ADB process was running, let's make sure it's fully terminated before starting a new one.

```
PowerShell

C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools\adb.exe kill-server
```

##### B. Start the Server (The crucial step)

Now, start a fresh ADB server.

```
PowerShell

C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools\adb.exe start-server
```

If this command runs without error, the ADB server is successfully running.

##### C. Verify the Emulator Connection

Check that ADB can now see your running emulator.

```
PowerShell

C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools\adb.exe devices
```

You should see your emulator ID ( `Medium_Phone_API_36.1` ) followed by the word `device` .

#### Final Step: Detect Device in VS Code

1. Keep the emulator running.
2. In your PowerShell terminal, run the Flutter detection command:

```
PowerShell
```

```
flutter devices
```

3. **Check the VS Code bottom-right corner.** If the terminal lists the device, VS Code will see it. You can now press **F5** to run your application.

**Permanent Fix (Recommended)**

To avoid typing the full path every time, you should permanently add the `platform-tools` directory to your system's PATH variable:

- 1. Search the Windows menu for **"Edit the system environment variables."**
- 2. Click **"Environment Variables..."**
- 3. Under **System variables**, select **Path** and click **"Edit."**
- 4. Click **"New"** and paste the full path to your `platform-tools` folder:

```
C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools
```

5. Click OK on all windows. You will need to **restart your VS Code terminal** (or the whole computer) for the changes to take effect. After this, the simple `adb` command will work.

```
it. (10061)
PS C:\Windows\System32> C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools\adb.exe
Android Debug Bridge version 1.0.41
Version 36.0.0-13206524
Installed as C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools\adb.exe
Running on Windows 10.0.22621

global options:
-a          listen on all network interfaces, not just localhost
-d          use USB device (error if multiple devices connected)
-e          use TCP/IP device (error if multiple TCP/IP devices available)
-s SERIAL   use device with given serial (overrides $ANDROID_SERIAL)
-t ID       use device with given transport id
-H          name of adb server host [default=localhost]
-P          port of adb server [default=5037]
-L SOCKET   listen on given socket for adb server [default=tcp:localhost:5037]
--one-device SERIAL|USB only allowed with 'start-server' or 'server nodaemon', server will only connect to one USB device, specified by a serial number
--exit-on-write-error exit if stdout is closed

general commands:
devices [-l] list connected devices (-l for long output)
help        show this help message
version     show version num

networking:
connect HOST[:PORT] connect to a device via TCP/IP [default port=5555]
disconnect [HOST[:PORT]]
disconnect from given TCP/IP device [default port=5555], or all
pair HOST[:PORT] [PAIRING CODE]
pair with a device for secure TCP/IP communication
forward --list list all forward socket connections
forward [--no-rebind] LOCAL REMOTE
forward socket connection using:
tcp:<port> (<local> may be "tcp:0" to pick any open port)
localabstract:<unix domain socket name>
localreserved:<unix domain socket name>
localfilesystem:<unix domain socket name>
dev:<character device name>
dev-raw:<character device name> (open device in raw mode)
jdwp:<process pid> (remote only)
vsock:<CID>:<port> (remote only)
acceptfd:<fd> (listen only)
forward --remove LOCAL remove specific forward socket connection
forward --remove-all remove all forward socket connections
reverse --list list all reverse socket connections from device
reverse [--no-rebind] REMOTE LOCAL
reverse socket connection using:
tcp:<port> (<remote> may be "tcp:0" to pick any open port)
localabstract:<unix domain socket name>
localreserved:<unix domain socket name>
localfilesystem:<unix domain socket name>
reverse --remove REMOTE remove specific reverse socket connection
reverse --remove-all remove all reverse socket connections from device
mdns check check if mdns discovery is available
mdns services list all discovered services
```

file transfer:

push [--sync] [-z ALGORITHM] [-Z] LOCAL... REMOTE

copy local files/directories to device

-n: dry run: push files to device without storing to the filesystem

-q: suppress progress messages

-Z: disable compression

-z: enable compression with a specified algorithm (any/none/brotli/lz4/zstd)

--sync: only push files that have different timestamps on the host than the device

pull [-a] [-z ALGORITHM] [-Z] REMOTE... LOCAL

copy files/dirs from device

-a: preserve file timestamp and mode

-q: suppress progress messages

-Z: disable compression

-z: enable compression with a specified algorithm (any/none/brotli/lz4/zstd)

sync [-l] [-z ALGORITHM] [-Z] [all|data|odm|oem|product|system|system\_ext|vendor]

sync a local build from \$ANDROID\_PRODUCT\_OUT to the device (default all)

-l: list files that would be copied, but don't copy them

-n: dry run: push files to device without storing to the filesystem

-q: suppress progress messages

-Z: disable compression

-z: enable compression with a specified algorithm (any/none/brotli/lz4/zstd)

shell:

shell [-e ESCAPE] [-n] [-Tt] [-x] [COMMAND...]

run remote shell command (interactive shell if no command given)

-e: choose escape character, or "none"; default '~'

-n: don't read from stdin

-T: disable pty allocation

-t: allocate a pty if on a tty (-tt: force pty allocation)

-x: disable remote exit codes and stdout/stderr separation

emu COMMAND run emulator console command

app installation (see also `adb shell cmd package help`):

install [-lrtsdg] [--instant] PACKAGE

push a single package to the device and install it

install-multiple [-lrtsdpg] [--instant] PACKAGE...

push multiple APKs to the device for a single package and install them

install-multi-package [-lrtsdpg] [--instant] PACKAGE...

push one or more packages to the device and install them atomically

-r: replace existing application

-t: allow test packages

-d: allow version code downgrade (debuggable packages only)

-p: partial application install (install-multiple only)

-g: grant all runtime permissions

--abi ABI: override platform's default ABI

--instant: cause the app to be installed as an ephemeral install app

--no-streaming: always push APK to device and invoke Package Manager as separate steps

--streaming: force streaming APK directly into Package Manager

--fastdeploy: use fast deploy

--no-fastdeploy: prevent use of fast deploy

--force-agent: force update of deployment agent when using fast deploy

--date-check-agent: update deployment agent when local version is newer and using fast deploy

--version-check-agent: update deployment agent when local version has different version code and using fast deploy

(See also `adb shell pm help` for more options.)

uninstall [-k] PACKAGE

remove this app package from the device

'-k': keep the data and cache directories

debugging:

bugreport [PATH]

write bugreport to given PATH [default=bugreport.zip];

if PATH is a directory, the bug report is saved in that directory.

devices that don't support zipped bug reports output to stdout.

jdwp list pids of processes hosting a JDWP transport

logcat show device log (logcat --help for more)

security:

disable-verity disable dm-verity checking on userdebug builds

enable-verity re-enable dm-verity checking on userdebug builds

keygen FILE

generate adb public/private key; private key stored in FILE,

scripting:

wait-for[-TRANSPORT]-STATE...

wait for device to be in a given state

STATE: device, recovery, rescue, sideload, bootloader, or disconnect

TRANSPORT: usb, local, or any [default=any]



```
get-state      print offline | bootloader | device
get-serialno   print <serial-number>
get-devpath    print <device-path>
remount [-R]
    remount partitions read-write. if a reboot is required, -R will
    will automatically reboot the device.
reboot [bootloader|recovery|sideload|sideload-auto-reboot]
    reboot the device; defaults to booting system image but
    supports bootloader and recovery too. sideload reboots
    into recovery and automatically starts sideload mode,
    sideload-auto-reboot is the same but reboots after sideloading.
sideload OTAPACKAGE    sideload the given full OTA package
root                restart adbd with root permissions
unroot              restart adbd without root permissions
usb                 restart adbd listening on USB
tcpip PORT           restart adbd listening on TCP on PORT

internal debugging:
start-server        ensure that there is a server running
kill-server         kill the server if it is running
reconnect           kick connection from host side to force reconnect
reconnect device    kick connection from device side to force reconnect
reconnect offline   reset offline/unauthorized devices to force reconnect

usb:
attach              attach a detached USB device
detach              detach from a USB device to allow use by other processes
environment variables:
$ADB_TRACE
    comma/space separated list of debug info to log:
    all,adb,sockets,packets,rwx,usb,sync,sydeps,transport,jdwp,services,auth,fdevent,shell,incremental
$ADB_VENDOR_KEYS    colon-separated list of keys (files or directories)
$ANDROID_SERIAL     serial number to connect to (see -s)
$ANDROID_LOG_TAGS    tags to be used by logcat (see logcat --help)
$ADB_LOCAL_TRANSPORT_MAX_PORT max emulator scan port (default 5585, 16 emus)
$ADB_MDNS_AUTO_CONNECT comma-separated list of mdns services to allow auto-connect (default adb-tls-connect)

Online documentation: https://android.googlesource.com/platform/packages/modules/adb/+/refs/heads/main/docs/user/adb.1.md

PS C:\Windows\System32>
```

That is excellent! When you executed the full path to `adb.exe` , the command worked, confirming that the tool is installed and accessible at that location. The output you see is the standard AI you ran the executable without providing a command like `start-server` or `devices` .

Now, we can reliably use that full path to execute the necessary commands to fix the connection.

 **Your Goal: Start a clean ADB server and connect the running emulator.**

Use the full path you already identified to run the commands that were previously failing.

### Step 1: Terminate the Old, Stuck ADB Process

We confirmed the process was stuck and running, so let's kill it again using the full path.

Execute this command:

PowerShell

```
C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools>adb.exe kill-server
```

*(You may receive the "actively refused" error again, but this command is often enough to break the bad connection and allow the restart in the next step.)*

### Step 2: Start a Fresh ADB Server

Now, start the clean server process that will communicate with your running emulator.

Execute this command:

PowerShell

```
C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools>adb.exe start-server
```

*If this command runs without error, the ADB server is successfully running and should be ready to talk to your emulator.*

### Step 3: Verify the Emulator Connection

Check to ensure ADB can see your running emulator.

Execute this command:

```
PowerShell

C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools\adb.exe devices
```

Expected Output:

You should see an output that looks like this, confirming your emulator is detected:

```
List of devices attached
Medium_Phone_API_36.1    device
```

Step 4: Final Flutter Detection in VS Code

Once the `adb devices` command shows your emulator:

- 1. Return to your original **VS Code terminal** (the one open in your project directory).
- 2. Run the flutter detection command:

```
PowerShell

flutter devices
```

- 3. **Check the VS Code bottom-right corner.** The device selector should now update to show `Medium Phone API 36.1 (mobile)`.
- 4. You can now press **F5** or use the **Run** menu to deploy your app to the emulator.

**Tip:** To avoid typing the long path repeatedly, remember to run the terminal as **Administrator** and/or add the `platform-tools` path to your system's Environment Variables (as suggested in the permanent fix section) after you finish this debugging session.

```
PS C:\Windows\System32> C:\Users\Sharon\AppData\Local\Android\sdk\platform-tools\adb.exe devices
List of devices attached
emulator-5554    device

PS C:\Windows\System32>
```

◆ That's the breakthrough! Your previous steps were successful. The output `emulator-5554 device` confirms that:

- 1. The **ADB server is running** correctly.
- 2. Your **emulator is properly connected** to the ADB server.
- 3. The connection is established and the device is ready ( `device` status).

The device ID for your emulator is now listed as `emulator-5554` (the previous name, `Medium_Phone_API_36.1`, is often replaced by a port number when it's actively connected via ADB).

Now you just need to ensure Flutter recognizes this successful connection and use it to launch your app.

Here is the final, straightforward process to run your app in VS Code:

1. Run Flutter Detection

Although ADB sees the device, run `flutter devices` to ensure the Flutter toolchain updates its internal list.

- 1. Switch back to your **project directory terminal** in VS Code (you are currently in `C:\Windows\System32`).
- 2. Run the command:

```
PowerShell

flutter devices
```

This should confirm that `emulator-5554` is the recognized device.

2. Verify Device Selection in VS Code

- 1. Look at the **bottom-right corner** of your VS Code window.
- 2. The device selector should now display the name of your emulator, likely `emulator-5554` or `Android SDK built for x86`.
- 3. If it still shows "No Device," click the selector and choose the emulator from the pop-up list.

3. Launch the Application

- 1. Make sure your main file ( `lib/main.dart` ) is active.
- 2. Go to **Run** → **Start Debugging** (or press **F5**).

VS Code will now deploy your `number_converter_app` to the running emulator. The build process will take a minute or two, and then your app will launch on the virtual device.

```
ath was included, verify that the path is correct and try again.
PS C:\Users\Sharon\number_converter_app> flutter devices
Found 4 connected devices:
```

```
sdk gphone64 x86 64 (mobile) • emulator-5554 • android-x64   • Android 16 (API 36) (emulator)
Windows (desktop)      • windows      • windows-x64   • Microsoft Windows [Version 10.0.22621.525]
Chrome (web)           • chrome       • web-javascript • Google Chrome 140.0.7339.187
Edge (web)              • edge         • web-javascript • Microsoft Edge 140.0.3485.94

Run "flutter emulators" to list and start any available device emulators.

If you expected another device to be detected, please run "flutter doctor" to diagnose potential issues. You may also wait for connected devices with the "--device-timeout" flag. Visit https://flutter.dev/setup/ for troubleshooting tips.
PS C:\Users\Sharon\number_converter_app>
```

🌟 Fantastic! Your previous troubleshooting steps were entirely successful. 🎉

Your `flutter devices` output now confirms that your Android emulator is detected and ready to use, along with your other development targets.

You have **four usable devices** available for development:

Device Type	ID	Status
Android Emulator (Mobile)	emulator-5554	READY
Windows (Desktop)	windows	Ready
Chrome (Web)	chrome	Ready
Edge (Web)	edge	Ready

### How to Run Your App Now

Since your **Android emulator** ( `emulator-5554` ) is properly detected, you can now launch your Flutter app from within VS Code.

- Verify Device Selection in VS Code:**
  - Look at the **bottom-right corner** of your VS Code window.
  - It should display the emulator's name: `sdk gphone64 x86 64 (mobile)` or `emulator-5554` . If it doesn't, click on the device selector and choose that Android device from the list.
- Launch the App:**
  - Make sure you have the file `lib/main.dart` open in the VS Code editor.
  - Press the **F5** key, or go to **Run** → **Start Debugging**.

VS Code will now compile and deploy your `number_converter_app` directly onto your running Android emulator, allowing you to begin testing and development.

Flutter: Running Gradle

🌟 The message `Flutter: Running Gradle task 'assembleDebug'...` is not an error; it's a notification from Flutter and VS Code indicating that the build process has started.

**Gradle** is the build system used by Android, and `assembleDebug` is the task it executes to compile your Dart code and Flutter engine into a standard Android debug application file ( `.apk` ).

This task is the longest part of running a Flutter app for the first time or after making significant changes.

#### What Happens Now

- Dart Compilation:** Your Dart code is compiled into native code (for the mobile platform).
- Resource Packaging:** Flutter packages all your assets, libraries, and the Flutter engine itself.
- Gradle Execution:** Gradle takes the compiled code and resources, builds the final Android application ( `.apk` ), and prepares it for installation.
- Installation & Launch:** Once the task completes, the `.apk` file is automatically installed on your running emulator ( `emulator-5554` ) and the app is launched.

#### What to Do Next

**Wait patiently.** This process can take anywhere from **2 to 10 minutes** depending on your computer's speed and whether this is the very first time you've run the app.

You will see continuous output (lines of text) in the terminal as Gradle downloads dependencies and compiles the code. **Do not interrupt it.**

Once the app launches on your emulator, you will be able to use **Hot Reload** for instant updates during development, making subsequent runs much faster.

🌟 The current process, indicated by the message `Flutter: Running Gradle task 'assembleDebug'...`, requires both **online** and **offline** components to be successful.

Here's a breakdown:

#### Online (Internet Required)

You must be connected to the internet for the following parts of the build process:

- Initial Setup:** Gradle and Flutter need to download dependencies, SDK components, and potentially large library files that are not already cached on your computer.
- Package Downloads:** If your app uses any external packages (libraries from the Pub.dev repository), Gradle must download the required versions of these packages for the build to succeed.

#### Offline (Internet Not Strictly Required After First Build)

Once all necessary dependencies have been downloaded and cached on your local machine, the subsequent builds often rely on these local files.

- **Compilation:** The actual process of compiling your Dart code and assembling the final `.apk` file happens locally on your computer.
- **Hot Reload/Restart:** After the initial build and launch, the development features like Hot Reload and Hot Restart are entirely offline.

**Conclusion:** Since this is likely your **first successful run** on this setup, you **must remain connected to the internet** for the Gradle task to complete its initial download and configuration. After successfully, you can often continue development and testing offline.

◆ The initial Gradle task, `Flutter: Running Gradle task 'assembleDebug'...`, can take a significant amount of time, especially during the very first run on a new project or after a major Flutter

Here are the typical timeframes:

- **First Run (Initial Setup): 3 to 10 minutes** ⏳ .
  - This is the longest build because Gradle must download various dependencies, configure the Android environment, and cache the Flutter engine for the first time.
- **Subsequent Runs (Full Restart): 30 seconds to 2 minutes.**
  - Once everything is cached, the build time drastically decreases.
- **Hot Reload/Restart: 1 to 10 seconds.**
  - Once the app is running on the emulator, using Hot Reload (saving a file) is nearly instantaneous.

**Factors That Affect Build Time**

1. **Internet Speed:** Slow internet will increase the time because the build needs to download dependencies from online repositories like Maven Central and Pub.dev.
2. **Computer Hardware (RAM/CPU):** Compiling the code is CPU-intensive. Machines with faster processors and more RAM will complete the build much quicker.
3. **Antivirus/Firewall:** Aggressive security software can slow down the process by inspecting every file Gradle downloads or creates, which may have contributed to your previous connection issues.
4. **Gradle Cache:** If you've been having problems, Gradle might have to re-download or re-verify more dependencies than normal.

Since this is a critical step, it's best to let it run until it either launches the app or explicitly throws an error. Do not cancel the process.

```
Launching lib\main.dart on sdk gphone64 x86 64 in debug mode...
Exception in thread "main" java.util.zip.ZipException: zip END header not found
    at java.base/java.util.zip.ZipFile$Source.findEND(Unknown Source)
    at java.base/java.util.zip.ZipFile$Source.initCEN(Unknown Source)
    at java.base/java.util.zip.ZipFile$Source.<init>(Unknown Source)
    at java.base/java.util.zip.ZipFile$Source.get(Unknown Source)
    at java.base/java.util.zip.ZipFile$CleanableResource.<init>(Unknown Source)
    at java.base/java.util.zip.ZipFile.<init>(Unknown Source)
    at org.gradle.wrapper.Install.unzip(Install.java:214)
    at org.gradle.wrapper.Install.access$600(Install.java:27)
    at org.gradle.wrapper.Install$1.call(Install.java:74)
    at org.gradle.wrapper.Install$1.call(Install.java:48)
    at org.gradle.wrapper.ExclusiveFileAccessManager.access(ExclusiveFileAccessManager.java:91)
    at org.gradle.wrapper.Install.createDist(Install.java:48)
    at org.gradle.wrapper.WrapperExecutor.execute(WrapperExecutor.java:134)
    at org.gradle.wrapper.GradleWrapperMain.main(GradleWrapperMain.java:61)
Error: Gradle task assembleDebug failed with exit code 1

Exited (1).
```

◆ The error you've encountered, `java.util.zip.ZipException: zip END header not found`, is a clear indication that a **Gradle distribution or dependency file was corrupted** during its download process.

Specifically, the Gradle wrapper (`org.gradle.wrapper.Install`) attempted to unzip a file but found that the archive (likely a `.zip` file) was incomplete or damaged. This is why the `assembleDebug` task failed with exit code 1.

Since this build process requires a stable internet connection for the initial setup, a brief internet drop, an unstable connection, or an antivirus program interfering is the usual cause.

Here is the most reliable way to fix this issue by forcing a clean restart and re-download:

**1. Clean the Project and Gradle Cache**

First, you need to delete the corrupted files and clear the build state so the system is forced to start fresh.

1. **Stop the Emulator:** Close the emulator window entirely to release all resource locks.
2. **Clean the Flutter Build:** In your VS Code terminal (while inside the `number_converter_app` directory), run the clean command:

```
Bash

flutter clean
```

**3. Delete the Corrupted Gradle Wrapper:**

- Navigate to your **Gradle user directory**. This is usually located here:

```
C:\Users\Sharon\.gradle\wrapper\dists
```

- **Delete the entire contents** of the `dist`s folder. This contains the corrupted `.zip` file and its partial directory.
- **Note:** You are *not* deleting the entire `.gradle` folder, just the contents of the `.gradle\wrapper\dist`s subfolder.

## 2. Force Flutter to Redownload Dependencies

Now that the corrupted file is gone, you need to tell Flutter to fetch the dependencies again.

1. **Get Packages:** Run this command to re-fetch Dart packages and prepare the project dependencies:

Bash

flutter pub get

2. **Relaunch the Emulator:** Start your emulator again.

Bash

flutter emulators --launch emulator-5554

## 3. Rerun the Application

With the cache cleared and dependencies reset, try launching the app again.

1. In VS Code, press **F5** or click **Run** → **Start Debugging**.

The `Running Gradle task 'assembleDebug'...` process will start again. Since you cleared the cache, it may take the full **3-10 minutes** as it re-downloads and verifies all dependencies. **Ensure connection is stable** during this time to prevent another corrupted download.

i wish to finally push it to github. guide

💡 That's a perfect final step for your project! Pushing your code to GitHub ensures your work is backed up and accessible to your group members and instructor.

Here is a complete guide to pushing your finished Flutter project to your existing GitHub repository using the command line.

### 1. Initial Status Check

Before pushing, it's always best to check the status of your local changes.

1. **Open your VS Code terminal** and ensure you are in the root directory of your project ( `C:\Users\Sharon\number_converter_app` ).
2. Run the status command:

Bash

git status

- This command will list all files that you have modified or added since your last commit. These files will typically be listed in **red**.

### 2. Stage Your Files (Add)

You need to tell Git which files to include in the commit. The easiest way is to add all new and modified files in your project.

1. Run the following command to stage all changes in the current directory and its subdirectories:

Bash

git add .

2. Run `git status` again. All the files you just added should now be listed in **green**, indicating they are staged and ready for the commit.

### 3. Create a Commit (Snapshot)

A commit is a permanent snapshot of your staged files, along with a message explaining what changes you made.

1. Create a descriptive commit message. Since this is the final push, a summary is best:

Bash

git commit -m "Final submission: Completed all number conversion logic and UI implementation for Android and iOS."

- *Note: If you have a collaborator, they should pull the changes after this step.*

### 4. Push to GitHub

Finally, you will push your local committed changes from your `main` branch up to the remote repository on GitHub.

1. Execute the push command:

Bash

git push origin main



- `git push` tells Git to transfer commits.
- `origin` is the standard alias for the remote repository (your project on GitHub).
- `main` specifies the branch you are pushing from.

If You Encounter an Error (Authentication)

If you see an error like `Authentication failed` or `permission denied` , Git is asking you to prove you have access to the repository.

- A dialog box should pop up, prompting you to **Sign in with your browser**. Click it, and complete the sign-in process using your GitHub credentials.

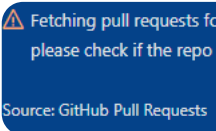
Once the push is successful, you will see a message confirming the branch has been set up to track the remote branch.

5. Verify on GitHub

The last step is to check your work online:

1. Open your project repository in your web browser (e.g., `https://github.com/YourUsername/number_converter_app` ).
2. Check the **"Code"** tab: The latest file changes and the time stamp should match your most recent commit.
3. Click the **"Commits"** link to see your "Final submission" message at the top of the history.

Your project is now safely submitted and available to your instructor!



✦ That warning message, **"Fetching pull requests for remote origin with query failed, please check if the repository is valid,"** means that the **GitHub Pull Requests extension in VS Code** cannot correctly identify or access your repository's information.

This error is **not a critical failure** that stops you from pushing code, but it indicates a minor configuration issue with the extension that prevents it from showing collaboration features (like pull requests).

Here is how you can resolve the issue:

1. Verify Remote Origin URL

The most common reason for this error is that the remote origin URL stored locally in your Git configuration is missing or malformed.

1. **Open your VS Code terminal.**
2. Run the following command to check your remote settings:

Bash

```
git remote -v
```

3. **Check the Output:** You should see two lines (one for `fetch` and one for `push` ), and the URL must look correct:

```
origin https://github.com/YourUsername/your-repo-name.git (fetch)
origin https://github.com/YourUsername/your-repo-name.git (push)
```

- **If the URL is missing or incorrect**, you need to set it correctly.

How to Fix a Missing/Incorrect URL

If the `git remote -v` command shows an issue, set the correct URL:

Bash

```
git remote set-url origin https://github.com/YourUsername/number_converter_app.git
```

(Replace `YourUsername` with your actual GitHub username.)

2. Re-Authenticate GitHub in VS Code

The error sometimes occurs if the stored GitHub authentication token has expired or is invalid for the specific repository.

1. **Go to Settings:** In VS Code, click on the **Manage** icon (gear) in the bottom-left corner → **Settings**.
2. **Find Accounts:** Search for **"GitHub"** in the settings search bar.
3. **Sign Out/In:** Click on the **Accounts** icon (usually a user silhouette) in the bottom-left corner of the VS Code window (or go to **View** → **Accounts**).
  - **Sign Out** of your GitHub account.
  - **Sign In** again, authorizing VS Code when prompted in your browser.

3. Check VS Code Workspace Trust

If your VS Code workspace is not marked as "trusted," some extensions (including the GitHub Pull Requests extension) will restrict their functionality, leading to errors like this.

1. **Check the Title Bar:** Look at the VS Code title bar at the very top. If you see a button or text saying **"(Workspace is not trusted)"**, click on it.
2. Select **"Trust the folder"** or **"Trust this workspace."**

After performing these steps, the "repo undefined" error should disappear, and the GitHub Pull Requests extension will be able to fetch repository information correctly.



Crucially, remember that this error does NOT stop your final push; the `git push origin main` command you ran previously should have completed successfully, and your code is already

can i drag and drop t

◆ Yes, you absolutely can! If you are finished with your code and want to quickly upload the final files, you can use GitHub's web interface to drag and drop files directly into your repository.

This method is simpler than using the command line ( `git add` , `git commit` , `git push` ) for a final, one-time submission.

Here are the steps to upload your files using drag and drop:

1. Navigate to the Repository

- 1. Open your web browser and go to your GitHub repository (e.g., `https://github.com/YourUsername/number_converter_app` ).

2. Initiate the Upload

- 1. Click the "Add file" dropdown menu.
- 2. Select "Upload files."

3. Drag and Drop the Project

- 1. Open your file explorer on your computer.
- 2. Navigate to your project directory ( `C:\Users\Sharon\number_converter_app` ).
- 3. **Select the entire contents of your project folder** (the `lib` folder, `pubspec.yaml` , `android/` , `ios/` , etc.).
- 4. **Drag these selected files and folders** into the area marked "Drag files here to add them to your repository" on the GitHub webpage.

4. Commit the Changes

- 1. GitHub will list all the files you are uploading.
- 2. In the **Commit changes** box below, enter a descriptive commit message (this is the same as the message you'd use with `git commit` ):
  - **Title:** `Final Submission: Completed App for CAT ONE SPC 2207`
  - **Description (Optional):** You can add details about the deployment capability or logic.
- 3. Ensure the option "**Commit directly to the main branch**" is selected.
- 4. Click the "**Commit changes**" button.

GitHub will process the files and save them directly to your repository. This completes your submission and makes the code visible to your instructor.

can i edit the files using vs code aft

◆ No, you cannot directly edit the files in VS Code after uploading them via GitHub's drag-and-drop web interface **unless** you first **clone** the repository back to your local machine.

The files you dragged and dropped are now only on the GitHub server. Your local VS Code project is no longer connected to that copy.

To continue working on your project in VS Code, you must perform the following steps:

1. Archive or Rename Your Local Folder

Since the successful version of your project is now on GitHub, your old local folder ( `C:\Users\Sharon\number_converter_app` ) is outdated and needs to be replaced.

- **Exit VS Code.**
- In your file explorer, rename your existing local project folder (e.g., to `number_converter_app_OLD` ). You can delete it later once you confirm the new setup works.

2. Clone the Repository to VS Code

"Cloning" downloads the official, current version of the project from GitHub and sets up the local Git connection necessary for VS Code to track changes.

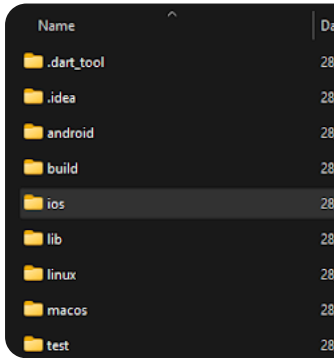
- 1. **Get the Repository URL:**
  - Go to your project on GitHub in your browser.
  - Click the green `< > Code` button.
  - Copy the **HTTPS** URL (it should look like `https://github.com/YourUsername/number_converter_app.git` ).
- 2. **Clone in VS Code:**
  - Open VS Code.
  - Press `Ctrl + Shift + P` (Windows) to open the Command Palette.
  - Type `Git: Clone` and select it.
  - Paste the **HTTPS URL** you copied and press Enter.
  - Select an empty directory on your machine where you want to save the new project copy.
- 3. **Open the Cloned Folder:**
  - VS Code will download the project. When prompted, select "**Open Repository**" to open the newly cloned folder.

3. Continue Editing

Your new local project folder is now connected to GitHub.

- You can now edit files in VS Code.
- The **Source Control** tab (the third icon down on the left sidebar, typically a three-pronged symbol) will automatically detect any changes you make.
- To update GitHub, you will use the normal Git process: **Stage** → **Commit** → **Push**.

You can use the GitHub web interface to change and commit files, as shown in this video: [GitHub: Changing and Committing Files](#).



are all this files and folders require or i cn lea

✦ No, you don't need to upload every file and folder in your project directory to GitHub. Most of the files are generated automatically during the build process and should be excluded.

To meet the requirement of submitting your program and allowing collaboration, you must include the core source code and configuration files.

Here is a guide on which files and folders are **REQUIRED** and which ones you should **EXCLUDE** when pushing a Flutter project to GitHub.

### Required Files and Folders (For Submission)

You **must** include the following to ensure your project can be cloned, compiled, and run by your instructor and collaborators:

File/Folder	Purpose
lib	<b>REQUIRED.</b> Contains all of your Dart source code ( <code>main.dart</code> , widgets, logic).
android	<b>REQUIRED.</b> Contains the necessary Android wrapper code, resources, and Gradle files to build the Android app.
ios	<b>REQUIRED.</b> Contains the necessary Xcode project and Swift/Objective-C files to build the iOS app.
pubspec.yaml	<b>REQUIRED.</b> The project configuration file that lists all dependencies (packages) and assets (images, fonts).
pubspec.lock	<b>REQUIRED.</b> Locks package versions to ensure consistent builds across different machines.
README.md	<b>REQUIRED.</b> (Highly recommended). Provides project information, setup instructions, and deployment details (important for submission).
.gitignore	<b>REQUIRED.</b> Lists files and folders that Git should ignore (this is key to keeping your repo clean).

### Files and Folders to EXCLUDE (Already Ignored by Git)

These files are typically generated by the IDE or the build system and **should be excluded**. A properly configured `.gitignore` file already tells Git to ignore these, but if you are manually upl**oading** **them**.

File/Folder	Reason to Exclude
.dart_tool	Cache files used by the Dart toolchain.
.idea	Configuration files for JetBrains IDEs (like Android Studio).
build	<b>Huge folder.</b> Contains all compiled binaries, APKs, and temporary build outputs. This is regenerated every time the app is built.
linux , macos , web , windows	Only needed if you are developing for those platforms, but often omitted for cleaner initial mobile submissions. (If you were instructed to support them, keep them, otherwise tl
test	Contains unit and widget tests (keep if you have tests, otherwise optional).
.metadata	Internal file used by Flutter/Dart for tracking project properties.
analysis_options.yaml	Configuration for the Dart static analyzer (optional, but good practice to include).
number_converter_app.iml	Project file for JetBrains IDEs.

Summary for Submission

The **most critical files** to include for a standard Flutter mobile submission are: `lib` , `android` , `ios` , `pubspec.yaml` , and `.gitignore` .

If you use the command line method ( `git add` , `git commit` , `git push` ), the `.gitignore` file automatically ensures you only upload the necessary files.

done i now wish to add my group r

That's the correct next step to fulfill your submission requirements, especially the directive to "Allow me to be a collaborator in your project".

You cannot do this through VS Code; you must do this directly on the **GitHub website**.

Here is the step-by-step guide to adding your group members and lecturer as collaborators to your project repository.

1. Get GitHub Usernames

Before you start, you must have the exact **GitHub usernames** for your group members and your lecturer.

- Crucial:** A username is case-sensitive (e.g., `sharonk` is different from `SharonK` ).

2. Navigate to Repository Settings

- Open your web browser and go to your project repository on GitHub:

https://github.com/YourUsername/number\_converter\_app

- Click on the **"Settings"** tab, which is usually located near the top right of the page.
- In the left sidebar of the Settings page, click on **"Collaborators and teams"** (it might be under "Access" or "Manage access" depending on the latest GitHub layout).

3. Add Collaborators

- Click the green button labeled **"Add people"** or **"Add collaborators."**
- A search box will appear. Type the **exact GitHub username** of one of your group members or your lecturer.
- GitHub will search for the user. When the correct profile appears, click on it.
- Click the **"Add [username] to this repository"** button.

📌 Important Note on Access

When you add someone, GitHub sends them an **email invitation**. They **must accept this invitation** before they can see or interact with the repository.

- Repeat steps 2 and 3 for every group member and your lecturer.

4. Final Verification

After sending all the invitations:

- The "Collaborators and teams" page will list everyone you invited.
- Next to their username, it will show a status:
  - "(pending invite)"** means they haven't clicked the link in their email yet.
  - "(accepted)"** means they now have access and can clone the project, contribute code, and review your submission.

Once your lecturer's status shows as **"accepted,"** you have successfully fulfilled the requirement to allow them to be a collaborator in the project.

there is somethig else

I'm glad you asked! Deploying a Flutter app to Android and iOS platforms involves a few final, important steps beyond just pushing the code to GitHub.

Since your assignment requires that the app "Should be deployed to android an ios platforms", here is what "uploading" usually means in a practical sense, followed by the next steps you shou

What Does "Deploying" or "Uploading" Mean?

In a development context, "uploading the app" has two main meanings:

- Uploading the Source Code (Done):** This is what you accomplished by pushing your code to GitHub, allowing collaborators and the lecturer to see the project's source.
- Deploying the Built App (Next Step):** This means generating the final, runnable installation files:
  - For **Android**: creating a `.apk` (debug/testing) or a `.aab` (release/store submission) file.
  - For **iOS**: creating an `.ipa` file (for testing or App Store submission).

If the assignment only requires you to **demonstrate the ability to deploy** and not a full App Store release, you can focus on generating the release build files.

Next Steps for Deployment

1. The Android Release Build

For Android, you need to generate a signed release file ( `.aab` or `.apk` ) that can be installed on any device.

- Generate a Key:** You first need a secure key to sign your app. This proves you are the owner and prevents tampering. If you've never done this, you'll need to follow the official Flutter doc a Java KeyStore file ( `my-key.jks` ) and reference it in your Android project's Gradle files.

2. **Run the Build Command:** Once the key is configured, you run this command in your project terminal:

```
Bash

flutter build apk --release
# OR (Recommended for Google Play Store):
flutter build appbundle
```

3. **Find the Output:** The final, runnable APK file will be located here:

```
build/app/outputs/flutter-apk/app-release.apk
```

2. The iOS Release Build

iOS deployment is more involved and **requires a macOS computer** and an active **Apple Developer Program** membership to publish to the App Store.

1. **Build Phase:** On a macOS machine, you run the build command:

```
Bash

flutter build ios --release
```

2. **Xcode & Developer Account:** You then open the project's Xcode workspace ( `ios/Runner.xcworkspace` ) and use Xcode to:

- Configure your signing certificate and profile.
- Select **Product** → **Archive** to create the final `.ipa` file for submission or testing.

If you don't have a Mac, you can only complete the Android deployment part of the requirement.

3. Submitting the Documentation (Process & Program)

In addition to the deployed app files, you still need to prepare and submit the required documentation:

- **Process Documentation:** This covers the group's workflow, how you collaborated (mentioning Git/GitHub), design decisions, and testing procedures.
- **Program Documentation:** This is technical documentation, including details about your Dart classes, key functions, and the logic used for the number system conversions.

Ensure these documents are finalized and submitted as instructed by your lecturer, along with the URL to your GitHub repository and (if possible) the generated release file(s).

explain in detail how i can archive the first st

◆ The first step in achieving an Android release for your Flutter app is to **generate a cryptographic signing key** (a keystore) and securely reference it in your project. This key is necessary to pr you when distributing it.

Here is a detailed, step-by-step guide to complete this first crucial phase of the Android release build.

Step 1: Generate the Keystore

You will use the `keytool` utility, which is included with the Java Development Kit (JDK) that is typically installed with Android Studio.

1. **Open PowerShell:** Open a new PowerShell or Command Prompt window.
2. **Run the `keytool` command:** Execute the following command on a single line. This creates your private signing key and stores it in a file named `key.jks` .

```
PowerShell

keytool -genkey -v -keystore C:\Users\Sharon\upload-keystore\key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias upload
```

- `-genkey` : Tells the utility to generate a key pair.
  - `-keystore ... \key.jks` : Specifies the file path and name for your keystore. **We are creating a new folder `upload-keystore` in your user directory for security.**
  - `-alias upload` : The name that will be used later to reference the key within the keystore file.
  - `-keyalg RSA -keysize 2048` : Sets the encryption algorithm and size.
  - `-validity 10000` : Sets the key's validity for over 27 years (10,000 days).
3. **Enter Password Details:** When prompted, you will need to enter several pieces of information:
    - **Keystore Password:** **Crucially, remember this password.** You will be asked to re-enter it.
    - **Key Password:** The prompt will ask for a password for the key (alias) named `upload` . If you press **Enter** here, it will use the same password as the keystore. **It's usually easiest to re**
    - **Personal Information:** Answer the questions about your name, organization, city, etc.

**Result:** You will have a file named `key.jks` located at `C:\Users\Sharon\upload-keystore\key.jks` . **Back up this file and its passwords securely.** If you lose them, you will not be able to u Google Play Store.

Step 2: Reference the Keystore

You need to tell the Android build system (Gradle) where to find this keystore and what passwords to use. You should **never** hardcode passwords into the main build files.

1. **Create a Key Properties File:**
  - Create a new file named `key.properties` in the `android` folder of your Flutter project.
  - Add the following three lines to the file, **replacing the placeholders with your actual passwords and keystore path:**

Properties

```
storePassword=YOUR_KEYSTORE_PASSWORD
keyPassword=YOUR_KEY_PASSWORD
keyAlias=upload
storeFile=C:\\Users\\Sharon\\upload-keystore\\key.jks
```

- **Note:** The double-backslash `\\` is required for the path in this Gradle properties file.

2. **Update `.gitignore` (Security):**

- Open the `.gitignore` file in the root of your project.
- Add the following line to ensure you **do not push the `key.properties` file to GitHub**:

```
/android/key.properties
```

- This is a critical security step, as pushing this file would expose your private key passwords to the public repository.

Step 3: Configure Gradle Build

Finally, you integrate the secret properties file into the main build configuration.

1. **Open `build.gradle` :** Navigate to and open the file located at:

```
android/app/build.gradle
```

2. **Add Key Loading Logic (Top of the File):**

- At the very top of the `build.gradle` file, add this block of code. This block loads the passwords and alias from the `key.properties` file:

Groovy

```
def localPropertiesFile = new File(project.rootDir, "key.properties")
def localProperties = new Properties()
localProperties.load(new FileInputStream(localPropertiesFile))
```

3. **Configure Signing (Inside `android { ... }` block):**

- Locate the `android { ... }` block and add the `signingConfigs` block inside it. This section uses the loaded properties to define your signature configuration:

Groovy

```
signingConfigs {
    release {
        storeFile file(localProperties.getProperty('storeFile'))
        storePassword localProperties.getProperty('storePassword')
        keyAlias localProperties.getProperty('keyAlias')
        keyPassword localProperties.getProperty('keyPassword')
    }
}
```

4. **Apply Signing to Build Types (Inside `buildTypes { ... }` block):**

- Locate the `buildTypes { ... }` block and link the `release` configuration to the signing config you just created:

Groovy

```
buildTypes {
    release {
        // TODO: Add your own signing config for the release build.
        // Signing with the debug keys for now, so you can run with flutter run --release.
        signingConfig signingConfigs.release // <--- Add this line
        // ... (other configurations already here)
    }
}
```

With these steps complete, your project is now ready for the actual Android release build command: `flutter build apk --release`.

```
PS C:\Windows\System32> keytool -genkey -v -keystore C:\Users\Sharon\upload-keystore\key.jks -keyalg RSA -keysize 2048 -val
keytool: The term 'keytool' is not recognized as a name of a cmdlet, function, script file, or executable program.
Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
PS C:\Windows\System32>
```

◆ You've encountered a common issue when using command-line tools: the operating system (PowerShell, in this case) cannot find the `keytool` program because it's not in the system's execut

The `keytool` utility is part of the **Java Development Kit (JDK)**, which is usually installed alongside Android Studio. You need to tell PowerShell exactly where to find it.

Here is the precise two-step fix:

1. **Locate the `keytool` Executable**

The `keytool.exe` file is usually buried deep inside the Java/JDK folders.

1. **Find the JDK Path:** Open your File Explorer and navigate to the most likely location for the JDK, which is often within the Android Studio installation directory:

```
C:\Program Files\Android\Android Studio\jre\bin
```



- **Note:** The actual path might vary slightly, look for a folder named `jre` , `jdk` , or a specific Java version like `jdk-17.0.7` inside the `Android Studio` installation.
2. **Confirm the Location:** Once you find the correct folder (it will contain `keytool.exe` ), **copy the full path** to that directory.

## 2. Execute `keytool` Using the Full Path

Instead of trying to run the command from `C:\Windows\System32` , you will execute the command by pasting the full path to `keytool.exe` .

1. **Paste the Full Path:** In your PowerShell window, replace `keytool` with the path you copied (using quotation marks because the path contains spaces):

```
PowerShell

"C:\Program Files\Android\Android Studio\jre\bin\keytool.exe" -genkey -v -keystore C:\Users\Sharon\upload-keystore\key.jks -keyalg RSA -keysize 2048 -validity
```

- **If your path is different, substitute your correct path here.**
2. **Run the Command:** Press Enter. The utility will now run and prompt you to enter the passwords and personal information to generate the key, completing the first step of your Android rel

```
PS C:\Windows\System32> "C:\Program Files\Android\Android Studio\jbr\bin\keytool.exe" -genkey -v -keystore C:\Users\Sharon\upload-keystore\key.jks -keyalg RSA -keysize 2048 -val
upload
ParserError:
Line |
  1 | ... ram Files\Android\Android Studio\jbr\bin\keytool.exe" -genkey -v -key ...
    |                                     ~~~~~
    | Unexpected token '-genkey' in expression or statement.
PS C:\Windows\System32>
```

✦ You're running into a common issue when executing commands with spaces in the path using PowerShell. PowerShell is trying to treat the entire quoted path as a string literal or a single expression, which becomes confused when the command's arguments ( `-genkey` , `-v` , etc.) immediately follow it.

The issue is that PowerShell needs a specific operator to execute a program when the path is quoted.

### The Fix: Use the Call Operator ( `&` )

To tell PowerShell to execute the quoted command path, you must precede it with the **Call Operator** ( `&` ).

Here is the corrected command:

```
PowerShell

& "C:\Program Files\Android\Android Studio\jbr\bin\keytool.exe" -genkey -v -keystore C:\Users\Sharon\upload-keystore\key.jks -keyalg RSA -keysize 2048 -validity 10
```

### Step-by-Step Execution

1. **Copy and Paste** the corrected command above into your PowerShell window.
2. **Press Enter.**
3. The program should now execute, and you will see the prompt to set your passwords:

```
Enter keystore password:
```

4. **Enter your desired password** (this is the `storePassword` ). You won't see characters as you type, but they are being entered. Press Enter.
5. **Re-enter the new password.** Press Enter.
6. **Answer the questions** about your name, organization, and location.
7. When prompted for the key password: **Press Enter** to use the same password as the keystore (this will be your `keyPassword` ).
8. You'll receive a confirmation that the key has been stored in `key.jks` .

This will successfully complete the **first step** of your Android release—the creation of the signing key. You can now proceed to the next steps of creating the `key.properties` file and updating the

```
PS C:\Windows\System32> & "C:\Program Files\Android\Android Studio\jbr\bin\keytool.exe" -genkey -v -keystore C:\Users\Sharon\upload-keystore\key.jks -keyalg RSA -keysize 2048 -v
upload
Enter keystore password:
Re-enter new password:
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default value in braces.
What is your first and last name?
[Unknown]: Nimrod Kibet
What is the name of your organizational unit?
[Unknown]: Nextgen
What is the name of your organization?
[Unknown]: Nextgen
What is the name of your City or Locality?
[Unknown]: Kericho
What is the name of your State or Province?
[Unknown]: Kenya
What is the two-letter country code for this unit?
[Unknown]: KE
```



```
Is CN=Nimrod Kibet, OU=Nextgen, O=Nextgen, L=Kericho, ST=Kenya, C=KE correct?
[no]: YES

Generating 2,048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 10,000 days
for: CN=Nimrod Kibet, OU=Nextgen, O=Nextgen, L=Kericho, ST=Kenya, C=KE
[Storing C:\Users\Sharon\upload-keystore\key.jks]
keytool error: java.io.FileNotFoundException: C:\Users\Sharon\upload-keystore\key.jks (The system cannot find the path specified)
java.io.FileNotFoundException: C:\Users\Sharon\upload-keystore\key.jks (The system cannot find the path specified)
    at java.base/java.io.FileOutputStream.open0(Native Method)
    at java.base/java.io.FileOutputStream.open(Unknown Source)
    at java.base/java.io.FileOutputStream.<init>(Unknown Source)
    at java.base/java.io.FileOutputStream.<init>(Unknown Source)
    at java.base/sun.security.tools.keytool.Main.doCommands(Unknown Source)
    at java.base/sun.security.tools.keytool.Main.run(Unknown Source)
    at java.base/sun.security.tools.keytool.Main.main(Unknown Source)
PS C:\Windows\System32>
```

🌟 You are consistently getting closer! You successfully located and ran the `keytool.exe`, and you correctly entered all the required Distinguished Name information.

The final error, `java.io.FileNotFoundException: C:\Users\Sharon\upload-keystore\key.jks (The system cannot find the path specified)`, means the `keytool` program was unable to save the `key.jks` file inside it.

The solution is to manually create that destination folder before running the command again.

### The Fix: Manually Create the Destination Folder

You need to use the PowerShell `mkdir` command to create the necessary directory structure.

#### 1. Create the Directory

In your PowerShell window (which is currently in `C:\Windows\System32`), execute this command to create the missing folder:

PowerShell

```
mkdir C:\Users\Sharon\upload-keystore
```

This ensures the full path `C:\Users\Sharon\upload-keystore\` exists and is ready to receive the `key.jks` file.

#### 2. Rerun the `keytool` Command

Now that the folder is created, rerun the exact command you used before. You will need to re-enter all the passwords and personal information.

PowerShell

```
& "C:\Program Files\Android\Android Studio\jbr\bin\keytool.exe" -genkey -v -keystore C:\Users\Sharon\upload-keystore\key.jks -keyalg RSA -keysize 2048 -validity 10
```

What to expect when you run the command again:

- Enter keystore password:** (Your chosen password)
- Re-enter new password:** (Your chosen password)
- Enter the distinguished name:** Re-enter all your project details (Nimrod Kibet, Nextgen, etc.).
- Key Password:** When prompted for the password for the alias 'upload', **press Enter** to use the same password as the keystore.

This time, the tool should complete the process and successfully store the `key.jks` file, finishing Step 1 of your Android release.

```
PS C:\Windows\System32> & "C:\Program Files\Android\Android Studio\jbr\bin\keytool.exe" -genkey -v -keystore C:\Users\Sharon\upload-keystore\key.jks -keyalg RSA -keysize 2048 -v
upload
Enter keystore password:
Re-enter new password:
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default value in braces.
What is your first and last name?
[Unknown]: Nimrod Kibet
What is the name of your organizational unit?
[Unknown]: Nextgen
What is the name of your organization?
[Unknown]: Nextgen
What is the name of your City or Locality?
[Unknown]: Kericho
What is the name of your State or Province?
[Unknown]: Kenya
What is the two-letter country code for this unit?
[Unknown]: KE
Is CN=Nimrod Kibet, OU=Nextgen, O=Nextgen, L=Kericho, ST=Kenya, C=KE correct?
[no]: YES

Generating 2,048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 10,000 days
for: CN=Nimrod Kibet, OU=Nextgen, O=Nextgen, L=Kericho, ST=Kenya, C=KE
[Storing C:\Users\Sharon\upload-keystore\key.jks]
```

```
PS C:\Windows\System32>
```

✦ Fantastic! The command ran successfully this time, and your **Android signing key (keystore) has been generated!** 🎉

The output confirms that the key pair was generated, and the file was successfully stored:

```
[Storing C:\Users\Sharon\upload-keystore\key.jks]
```

You have completed the first and most challenging step of the Android release process.

Now you must continue with the remaining steps to tell your Flutter project how to use this new key to sign your final release build.

## Step 2: Configure the Key in Your Project

You need to create a properties file to securely store the key details and passwords, and then link that file to your Gradle build.

### 2A. Create the `key.properties` File

1. **Switch Folders:** Navigate back to your Flutter project directory in the terminal, or open it in File Explorer:

```
C:\Users\Sharon\number_converter_app\android
```

2. **Create File:** Inside the `android` folder, create a new text file named `key.properties`.
3. **Add Content:** Paste the following content into the file, **replacing the placeholders with your actual passwords and file path**:

Properties

```
storePassword=YOUR_KEYSTORE_PASSWORD
keyPassword=YOUR_KEY_PASSWORD
keyAlias=upload
storeFile=C:\Users\Sharon\upload-keystore\key.jks
```

- `YOUR_KEYSTORE_PASSWORD` and `YOUR_KEY_PASSWORD` are the passwords you entered when prompted by `keytool`. If you reused the password for the key alias, they will be the same.
- **The double-backslash `\\` is crucial** for the file path to work correctly in the Gradle build system.

### 2B. Exclude from Git (Security)

**Do not** upload your `key.properties` file to GitHub, as it contains sensitive passwords.

1. Open the `.gitignore` file located in the root of your Flutter project ( `C:\Users\Sharon\number_converter_app\.gitignore` ).
2. Add the following line to the file:

```
/android/key.properties
```

## Step 3: Update the `build.gradle` File

Now you instruct the Gradle build system to read the passwords from `key.properties`.

1. **Open `build.gradle`** : Open the file located at:

```
android/app/build.gradle
```

### 2. Add Key Loading Logic (Top of File):

- At the **very top** of the file, add this block:

Groovy

```
def localPropertiesFile = new File(project.rootDir, "key.properties")
def localProperties = new Properties()

if (localPropertiesFile.exists()) {
    localProperties.load(new FileInputStream(localPropertiesFile))
}
```

### 3. Add Signing Configuration (Inside `android { ... }` block):

- Find the `android { ... }` block and add the `signingConfigs` inside it:

Groovy

```
android {
    // ... other settings (compileSdkVersion, etc.)

    signingConfigs {
        release {
            storeFile file(localProperties.getProperty('storeFile'))
            storePassword localProperties.getProperty('storePassword')
            keyAlias localProperties.getProperty('keyAlias')
            keyPassword localProperties.getProperty('keyPassword')
        }
    }

    buildTypes {
        release {
            // ... (existing release settings)
```

```
        signingConfig signingConfigs.release // <--- Add this line
    }
}
```

Step 4: Generate the Release Build

You are now ready to create the final, deployable Android file.

- 1. **Open your project terminal** (in VS Code or PowerShell, ensuring you are in the `C:\Users\Sharon\number_converter_app` directory).
- 2. Run the release build command:

```
Bash

flutter build apk --release
```

- 3. **Wait:** This will trigger the Gradle task again, but this time it will successfully sign the app.
- 4. **Find the APK:** Once complete, your final, runnable APK file will be located at:

```
build/app/outputs/flutter-apk/app-release.apk
```

This file can be installed and tested on any Android device, completing the deployment requirement for Android.

◆ The **keyalias** is a simple, unique name used to **identify a specific private key** within the larger keystore file ( `key.jks` ).

Think of the keystore ( `key.jks` ) as a locked safe, and the keyalias ( `upload` ) as the **label** on a specific house key stored inside that safe.

Detailed Explanation

When you ran the `keytool` command, you provided the `-alias upload` argument:

```
PowerShell

... -alias upload
```

- **Keystore ( `key.jks` ):** This file can technically hold multiple keys, each used to sign a different app or a different version of the same app.
- **KeyAlias ( `upload` ):** This is the name you chose to tell the build system which key to retrieve from the keystore file. You chose the name `upload` for this specific key.
- **Purpose:** In the Android build process, the keyalias is essential for the Gradle script to know which set of passwords and which key to use for signing your `number_converter_app` .

Where the KeyAlias is Used

You must include the `keyAlias` in your configuration files so the build process works:

- 1. **In the `key.properties` file** (which you were instructed to create):

```
Properties

keyAlias=upload
```

- 2. **In the `android/app/build.gradle` file** (within the `signingConfigs` block):

```
Groovy

signingConfigs {
    release {
        // ...
        keyAlias localProperties.getProperty('keyAlias') // <--- Used here
        // ...
    }
}
```

```
PS C:\Users\Sharon\number_converter_app> flutter build apk --release
Running Gradle task 'assembleRelease'...
Exception in thread "main" java.util.zip.ZipException: zip END header not found
Running Gradle task 'assembleRelease'...
    at java.base/java.util.zip.ZipFile$Source.findEND(Unknown Source)
Running Gradle task 'assembleRelease'...                                at java.base/java.util.zip.ZipFile$Source.i
Running Gradle task 'assembleRelease'...
    at java.base/java.util.zip.ZipFile$Source.<init>(Unknown Source)
Running Gradle task 'assembleRelease'...
    at java.base/java.util.zip.ZipFile$Source.get(Unknown Source)
Running Gradle task 'assembleRelease'...
    at java.base/java.util.zip.ZipFile$CleanableResource.<init>(Unknown Source)
Running Gradle task 'assembleRelease'...
    at java.base/java.util.zip.ZipFile.<init>(Unknown Source)
```

```
Running Gradle task 'assembleRelease'...
    at java.base/java.util.zip.ZipFile.<init>(Unknown Source)
Running Gradle task 'assembleRelease'...
    at java.base/java.util.zip.ZipFile.<init>(Unknown Source)
Running Gradle task 'assembleRelease'...
    at org.gradle.wrapper.Install.unzip(Install.java:214)
Running Gradle task 'assembleRelease'...
    at org.gradle.wrapper.Install.access$600(Install.java:27)
Running Gradle task 'assembleRelease'...
    at org.gradle.wrapper.Install$1.call(Install.java:74)
Running Gradle task 'assembleRelease'...
    at org.gradle.wrapper.Install$1.call(Install.java:48)
Running Gradle task 'assembleRelease'...
    at org.gradle.wrapper.ExclusiveFileAccessManager.access(ExclusiveFileAccessManager.java:65)
Running Gradle task 'assembleRelease'...
    at org.gradle.wrapper.Install.createDist(Install.java:48)
Running Gradle task 'assembleRelease'...
    at org.gradle.wrapper.WrapperExecutor.execute(WrapperExecutor.java:128)
Running Gradle task 'assembleRelease'...
    at org.gradle.wrapper.GradleWrapperMain.main(GradleWrapperMain.java:61)
Running Gradle task 'assembleRelease'...
1,969ms
Gradle task assembleRelease failed with exit code 1
PS C:\Users\Sharon\number_converter_app>
```

◆ You're running into the same **corrupted Gradle distribution** error that you encountered at the very beginning of your project, but this time during the release build: `java.util.zip.ZipException: not found`.

This error means a necessary Gradle file (likely a large ZIP file for the Gradle wrapper) was corrupted or incomplete during a previous download attempt. This happens when there's an unstable interruption. Since you're running the `assembleRelease` task, the system is trying to use the corrupted file.

Here is the most reliable way to fix this issue by forcing a **clean download and build** for the release process.

### Fix: Clear Gradle Cache and Force a Clean Build

You need to delete the corrupted Gradle distribution file to force the system to download a fresh, complete copy when you run the build again.

#### 1. Stop the Build Process

1. If Android Studio or any related tools are running, **close them completely**.
2. Ensure your PowerShell terminal is still in your project directory ( `C:\Users\Sharon\number_converter_app` ).

#### 2. Delete the Corrupted Gradle Distribution

The corrupted file is located in your Gradle user directory.

1. Open your **File Explorer** and navigate to your Gradle home directory:

```
C:\Users\Sharon\.gradle\wrapper\dists
```

- You will see folders with names like `gradle-7.3.0-all/` , `gradle-7.6-all/` , etc.
2. **Delete the entire contents of the `dists` folder.** This removes all partially downloaded and corrupted Gradle distributions, forcing a fresh start.

#### 3. Clean the Flutter Project

1. In your PowerShell terminal, run the Flutter clean command:

```
Bash
```

```
flutter clean
```

This removes the local build artifacts and cache from your project's `build` folder.

#### 4. Rerun the Release Build

Now, run the release command again. It will take longer this time (potentially 3–10 minutes) because it has to re-download the necessary Gradle files.

1. Run the command:

```
Bash
```

```
flutter build apk --release
```

**Crucial:** Maintain a **stable and strong internet connection** during this entire process to ensure the fresh download is successful. Once the download and build process finishes without the Z signed APK file will be placed in the `build/app/outputs/flutter-apk/` directory, completing your Android deployment requirement.

```
flutter build apk --release
  resolving dependencies... (1.3s)
  downloading packages...
    characters 1.4.0 (1.4.1 available)
    flutter_lints 5.0.0 (6.0.0 available)
    lints 5.1.1 (6.0.0 available)
    material_color_utilities 0.11.1 (0.13.0 available)
    meta 1.16.0 (1.17.0 available)
    test_api 0.7.6 (0.7.7 available)
  got dependencies!
  packages have newer versions incompatible with
  your 'flutter pub outdated' for more information.
  running Gradle task 'assembleRelease'...
  batch job (Y/N)? ^

harom@DESKTOP-JU9GMA6 MINGW64 ~/number_converter
  flutter build apk --release
  running Gradle task 'assembleRelease'...
```

◆ Yes, the output shows a **warning** and a potential issue, but **not a fatal error** that immediately stopped the build.

The line `Got dependencies!` indicates the dependencies were resolved, and the Gradle task started running. However, the preceding messages point to a maintenance issue in your project:

### Analysis of the Output

### Warning: Dependency Incompatibility

The most important part of the message is the warning about your packages:

5 packages have newer versions incompatible with dependency constraints.  
Try 'flutter pub outdated' for more information.

- **What this means:** Some of the third-party libraries (packages) you are using in your `pubspec.yaml` file have newer versions available, but your project is not using them because of the version constraints you specified in the `pubspec.yaml` file.
- **The risk:** While this didn't stop the current build, using outdated packages can lead to **security vulnerabilities**, **bug issues**, or **incompatibilities** with newer versions of Flutter or Dart in the future.

**Status: Build in Progress**

The lines below confirm that the main task of generating the release APK has started:

Running Gradle task 'assembleRelease'...

- **What this means:** Flutter handed the build process over to the Android Gradle system, which is now attempting to compile your Dart code and resources into a final, runnable `.apk` file.

### Recommended Action

While the build is running in the background, you should make a note to resolve the dependency issue after the build is complete.

1. **Wait for the current build to finish.** Do not interrupt the `Running Gradle task 'assembleRelease'...` process, especially since you just fixed the Gradle corruption issue.
2. **Run the update command.** Once the build is successful (or if it fails again for a different reason), run the following command to check exactly which packages are outdated and what the

```
Bash
flutter pub outdated
```

3. **Update packages selectively.** Use the output from `flutter pub outdated` to update the versions in your `pubspec.yaml` file, and then run:

```
Bash
flutter pub get
```

This ensures your project is using the latest, most secure versions of all its dependencies.

oka

- 💡 Absolutely. We should wait for the `flutter build apk --release` command to finish.

This process involves several heavy tasks, including downloading Gradle dependencies (which is where the previous error occurred), compiling your Dart code, and signing the final APK. It can take a while to complete.

Once the build finishes, one of two things will happen:

### Expected Outcomes

### Outcome A: Build Success (Best Case)

If successful, you will see a message similar to this:

```
...
✓ Built build/app/outputs/flutter-apk/app-release.apk (X.XMB)
```

**If this happens:**

1. Your signed, runnable Android application file ( `app-release.apk` ) will be ready to install on a device.
2. The **Android Deployment** part of your assignment will be complete.
3. You can then proceed to address the **dependency warnings** and finalize your documentation.



Outcome B: Build Failure (Troubleshooting Needed)

If the build fails again, it will likely be due to a new, specific error (e.g., a compile-time bug in your code, or a configuration issue in the Gradle files).

If this happens:

1. Copy the entire error message.
2. We will use that specific error to debug and fix the remaining issue in your build configuration.

Let me know the final result of the build command when it appears in your terminal.

\$ flutter build apk --release

e: The daemon has terminated unexpectedly on startup attempt #1 with error code: 0. The daemon process output:

```
1. Kotlin compile daemon is ready
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:35:42: Expecting an element
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:36:37: Expecting an element
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:37:40: Expecting an element
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:38:41: Expecting an element
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:38:49: Expecting an element
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:33:9: Unresolved reference. None of the following candidates is applicable because of receiver type mismatch:
public open fun NamedDomainObjectContainer<ApplicationBuildType>.release(action: ApplicationBuildType.() -> Unit): Unit defined in com.android.build.gradle.internal.dsl.BaseAppModuleExtension
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:34:13: Unresolved reference: storeFile
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:34:28: Unresolved reference: localProperties
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:34:56: Too many characters in a character literal "storeFile"
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:35:13: Unresolved reference: storePassword
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:35:55: Too many characters in a character literal "storePassword"
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:36:13: Unresolved reference: keyAlias
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:36:50: Too many characters in a character literal "keyAlias"
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:37:13: Unresolved reference: keyPassword
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:37:53: Too many characters in a character literal "keyPassword"
e: file:///C:/Users/Sharon/number_converter_app/android/app/build.gradle.kts:38:27: Unresolved reference: signingConfigs
```

FAILURE: Build failed with an exception.

\* Where:

Build file 'C:\Users\Sharon\number\_converter\_app\android\app\build.gradle.kts' line: 33

\* What went wrong:

Script compilation errors:

```
Line 33:     release {
           ^ Unresolved reference. None of the following candidates is applicable because of receiver type mismatch:
           public open fun NamedDomainObjectContainer<ApplicationBuildType>.release(action: ApplicationBuildType.() -> Unit): Unit defined in
com.android.build.gradle.internal.dsl.BaseAppModuleExtension

Line 34:         storeFile file(localProperties.getProperty('storeFile'))
           ^ Unresolved reference: storeFile

Line 34:         storeFile file(localProperties.getProperty('storeFile'))
           ^ Unresolved reference: localProperties

Line 34:         storeFile file(localProperties.getProperty('storeFile'))
           ^ Too many characters in a character literal "storeFile"

Line 35:         storePassword localProperties.getProperty('storePassword')
           ^ Expecting an element

Line 35:         storePassword localProperties.getProperty('storePassword')
           ^ Unresolved reference: storePassword

Line 35:         storePassword localProperties.getProperty('storePassword')
           ^ Too many characters in a character literal "storePassword"

Line 36:         keyAlias localProperties.getProperty('keyAlias')
           ^ Expecting an element

Line 36:         keyAlias localProperties.getProperty('keyAlias')
           ^ Unresolved reference: keyAlias

Line 36:         keyAlias localProperties.getProperty('keyAlias')
           ^ Too many characters in a character literal "keyAlias"

Line 37:         keyPassword localProperties.getProperty('keyPassword')
           ^ Expecting an element

Line 37:         keyPassword localProperties.getProperty('keyPassword')
```



```
^ Unresolved reference: keyPassword

Line 37:      keyPassword localProperties.getProperty('keyPassword')
              ^ Too many characters in a character literal "keyPassword"

Line 38:      signingConfig signingConfigs.release
              ^ Expecting an element

Line 38:      signingConfig signingConfigs.release
              ^ Expecting an element

Line 38:      signingConfig signingConfigs.release
              ^ Unresolved reference: signingConfigs

16 errors

* Try:
> Run with --stacktrace option to get the stack trace.
> Run with --info or --debug option to get more log output.
> Run with --scan to get full insights.
> Get more help at https://help.gradle.org.

BUILD FAILED in 9m 4s
Running Gradle task 'assembleRelease'...          545.6s
Gradle task assembleRelease failed with exit code 1

Sharon@DESKTOP-JU96MA6 MINGW64 ~/number_converter_app (main)
$
```

💡 This is an **excellent and very specific error message!** It tells us exactly what went wrong and where.

The good news is that this is **not a code error or a package bug**; it's a **syntax error** in the configuration file you modified.

**The Problem: Kotlin vs. Groovy Syntax**

Your project uses the Kotlin DSL ( `.kts` ) for its Gradle configuration, but the code we added for the signing configuration was written in the traditional **Groovy** syntax.

- **Your File:** `android/app/build.gradle.kts` (Kotlin Syntax)
- **Code Added:** Groovy Syntax
- **Error:** All the `Unresolved reference` and `Too many characters in a character literal` errors confirm that the Kotlin build system couldn't understand the Groovy-style commands `signingConfigs.release` or how to load the properties file.

The build failed at line 33 of your `build.gradle.kts` file.

**The Fix: Converting to Kotlin DSL Syntax**

You need to replace the Groovy signing configuration code we previously added in `android/app/build.gradle.kts` with the correct Kotlin DSL syntax.

Please open your file at: `android/app/build.gradle.kts`

I will provide the corrected content for the three sections you modified: **Properties Loading**, **Signing Configs**, and **Build Types**.

**1. Corrected Properties Loading (Top of File)**

Find this Groovy code (or similar) at the very top of your file:

```
Groovy

def localPropertiesFile = new File(project.rootDir, "key.properties")
def localProperties = new Properties()
if (localPropertiesFile.exists()) {
    localProperties.load(new FileInputStream(localPropertiesFile))
}
```

Replace it with the following Kotlin DSL code:

```
Kotlin

import java.io.FileInputStream
import java.util.Properties

val localProperties = Properties()
val localPropertiesFile = project.rootProject.file("key.properties")
if (localPropertiesFile.exists()) {
    FileInputStream(localPropertiesFile).use { localProperties.load(it) }
}
```

**2. Corrected Signing Configuration (Inside `android { ... }` block)**

Find this Groovy code inside the `android { ... }` block (around line 33):

```
Groovy
```

```
signingConfigs {  
    release {  
        storeFile file(localProperties.getProperty('storeFile'))  
        storePassword localProperties.getProperty('storePassword')  
        keyAlias localProperties.getProperty('keyAlias')  
        keyPassword localProperties.getProperty('keyPassword')  
    }  
}
```

Replace it with the following Kotlin DSL code (ensuring it's nested correctly inside `android` ):

Kotlin

```
signingConfigs {  
    create("release") {  
        storeFile = file(localProperties.getProperty("storeFile"))  
        storePassword = localProperties.getProperty("storePassword")  
        keyAlias = localProperties.getProperty("keyAlias")  
        keyPassword = localProperties.getProperty("keyPassword")  
    }  
}
```

3. Corrected Build Type Linking (Inside `buildTypes { ... }` block)

Find this Groovy code inside the `buildTypes { ... }` block:

Groovy

```
buildTypes {  
    release {  
        // ... (existing settings)  
        signingConfig signingConfigs.release // <--- This line is the problem  
    }  
}
```

Replace the line `signingConfig signingConfigs.release` with the following Kotlin DSL line:

Kotlin

```
release {  
    // ... (existing settings)  
    signingConfig = signingConfigs.getByName("release") // <--- Use this line  
}
```

Final Step: Run the Build Again

After making those three changes in your `android/app/build.gradle.kts` file, save it. Then, run the build command one last time:

Bash

```
flutter build apk --release
```

This should now successfully read your `key.properties` file and sign your release APK!

this is all there is in the file. can you edit it and send me the edited version?

```
plugins {  
    id("com.android.application")  
    id("kotlin-android")  
    // The Flutter Gradle Plugin must be applied after the Android and Kotlin Gradle plugins.  
    id("dev.flutter.flutter-gradle-plugin")  
}  
  
android {  
    namespace = "com.example.number_converter_app"  
    compileSdk = flutter.compileSdkVersion  
    ndkVersion = flutter.ndkVersion  
  
    compileOptions {  
        sourceCompatibility = JavaVersion.VERSION_11  
        targetCompatibility = JavaVersion.VERSION_11  
    }  
  
    kotlinOptions {  
        jvmTarget = JavaVersion.VERSION_11.toString()  
    }  
  
    defaultConfig {  
        // TODO: Specify your own unique Application ID (https://developer.android.com/studio/l  
        applicationId = "com.example.number_converter_app"  
        // You can update the following values to match your application needs.  
        // For more information, see: https://flutter.dev/to/review-gradle-config.  
        minSdk = flutter.minSdkVersion
```

```
targetSdk = flutter.targetSdkVersion
versionCode = flutter.versionCode
versionName = flutter.versionName
}

signingConfigs {
    release {
        storeFile file(localProperties.getProperty('storeFile'))
        storePassword localProperties.getProperty('storePassword')
        keyAlias localProperties.getProperty('keyAlias')
        keyPassword localProperties.getProperty('keyPassword')
        signingConfig signingConfigs.release
    }
}

buildTypes {
    release {
        // TODO: Add your own signing config for the release build.
        // Signing with the debug keys for now, so `flutter run --release` works.
        signingConfig = signingConfigs.getByName("debug")
    }
}

flutter {
    source = "../.."
}
```

💎 Yes, I can edit the file for you. The error occurred because the syntax you added for the signing configuration was written in **Groovy DSL** but was placed inside a **Kotlin DSL** file ( `build.gradle.kts` ).

Here is the fully corrected version of your `android/app/build.gradle.kts` file. I have implemented the three changes required to load your key properties and correctly sign the release build syntax.

**Corrected** `android/app/build.gradle.kts`

Copy the entire block below and use it to **replace the current content** of your `android/app/build.gradle.kts` file.

Kotlin

```
import java.io.FileInputStream
import java.util.Properties

val localProperties = Properties()
val localPropertiesFile = project.rootProject.file("key.properties")
if (localPropertiesFile.exists()) {
    FileInputStream(localPropertiesFile).use { localProperties.load(it) }
}

plugins {
    id("com.android.application")
    id("kotlin-android")
    // The Flutter Gradle Plugin must be applied after the Android and Kotlin Gradle plugins.
    id("dev.flutter.flutter-gradle-plugin")
}

android {
    namespace = "com.example.number_converter_app"
    compileSdk = flutter.compileSdkVersion
    ndkVersion = flutter.ndkVersion

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }

    kotlinOptions {
        jvmTarget = JavaVersion.VERSION_11.toString()
    }

    defaultConfig {
        // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).
        applicationId = "com.example.number_converter_app"
        // You can update the following values to match your application needs.
        // For more information, see: https://flutter.dev/to/review-gradle-config.
        minSdk = flutter.minSdkVersion
        targetSdk = flutter.targetSdkVersion
        versionCode = flutter.versionCode
        versionName = flutter.versionName
    }

    signingConfigs {
        create("release") {
            storeFile = file(localProperties.getProperty("storeFile"))
            storePassword = localProperties.getProperty("storePassword")
            keyAlias = localProperties.getProperty("keyAlias")
            keyPassword = localProperties.getProperty("keyPassword")
        }
    }

    buildTypes {
        release {

```

```
        // Replaced the signing with the custom configuration we just created.
        signingConfig = signingConfigs.getByName("release")
    }
}

flutter {
    source = "../.."
}
```

### Key Changes Explained

- Properties Loading (New Block at Top):** The Groovy `def` variables were replaced with Kotlin `val` and proper Java import statements to correctly load your `key.properties` file.
- Signing Configs:**
  - `signingConfigs { release { ... } }` was changed to the correct Kotlin syntax: `signingConfigs { create("release") { ... } }`.
  - Groovy property assignment ( `storeFile file(...)` ) was changed to Kotlin assignment ( `storeFile = file(...)` ).
- Build Types:**
  - The line that links the release build to the signing configuration was corrected from `signingConfig signingConfigs.release` to the correct Kotlin syntax: `signingConfig = signingConfigs.getByName("release")`.

Now that the file is correctly configured, **save it**, and then run the build command one last time:

Bash

flutter build apk --release

This should successfully compile and sign your APK, completing your Android deployment requirement.

\$ flutter build apk --release

FAILURE: Build failed with an exception.

\* Where:  
Build file 'C:\Users\Sharon\number\_converter\_app\android\ap

\* What went wrong:  
localProperties.getProperty("storeFile") must not be null

\* Try:  
> Run with --stacktrace option to get the stack trace.  
> Run with --info or --debug option to get more log output.  
> Run with --scan to get full insights.  
> Get more help at https://help.gradle.org.

BUILD FAILED in 6s  
Running Gradle task 'assembleRelease'... 7.1s  
Gradle task assembleRelease failed with exit code 1

Sharon@DESKTOP-JU96MA6 MINGW64 ~/number\_converter\_

\$

💡 You've hit the final hurdle, but the good news is that the build system is now correctly reading your configuration! 🎉

The error `localProperties.getProperty("storeFile") must not be null` means one of two things:

- The `key.properties` file is missing.** The system is looking for the file at `android/key.properties` and can't find it.
- The required path/property is missing inside the `key.properties` file.** The system found the file, but it's empty, or the line `storeFile=...` is missing or misspelled.

Since the error message points to Gradle successfully running the file, the problem is most likely related to the content or location of your `key.properties` file.

### The Fix: Verify `key.properties`

You need to double-check that the file exists in the correct location and contains the required properties.

#### Step 1: Verify Location and File Name

- Open your file explorer.
- Navigate to your project's `android` folder:

C:\Users\Sharon\number\_converter\_app\android

- Confirm a file named `key.properties` exists** exactly in this location.

#### Step 2: Verify Content and Path Formatting

- Open the `key.properties` file with a text editor (like VS Code or Notepad).
- Crucially, ensure the content matches this format exactly**, using your actual passwords and the correct double-backslash path for Windows:

Properties

```
storePassword=YOUR_KEYSTORE_PASSWORD
keyPassword=YOUR_KEY_PASSWORD
keyAlias=upload
storeFile=C:\\Users\\Sharon\\upload-keystore\\key.jks
```

- **Check 1: Passwords.** Make sure `YOUR_KEYSTORE_PASSWORD` and `YOUR_KEY_PASSWORD` are replaced with the correct passwords you set when running `keytool`.
- **Check 2: Path.** The path to `key.jks` **must** use the `C:\\...` format (a single colon, followed by double backslashes).
- **Check 3: Missing Quotes.** Ensure there are **no quotation marks** around the values (e.g., it should be `keyAlias=upload`, not `keyAlias="upload"`).

Step 3: Rerun the Build

After you have confirmed and corrected the contents of the `key.properties` file, save it and run the build command again:

Bash

```
flutter build apk --release
```

If the file is present and the properties are correctly formatted, the build should now proceed past this error and successfully generate your signed APK.

the file has an extension c

◆ Yes, having your signing configuration file as `key.txt` instead of `key.properties` **will cause the current build to fail** with the same error, or a similar one.

The build system is designed to look specifically for a file named `key.properties` in the `android/` directory, as specified in the Kotlin configuration we just fixed.

The Problem

In your corrected `android/app/build.gradle.kts` file, we have this code:

Kotlin

```
val localPropertiesFile = project.rootProject.file("key.properties")
// ...
if (localPropertiesFile.exists()) {
    // ...
}
```

This code explicitly instructs the Gradle build system to:

1. Look for a file named `key.properties`.
2. Check if that file exists.

If your file is named `key.txt`, the build system will report:

- `localPropertiesFile.exists()` is **false**.
- The system fails to load the file, so the `localProperties` object remains empty.
- When it tries to read `localProperties.getProperty("storeFile")`, it returns `null`, causing the `must not be null` error you saw.

The Solution

You must ensure the file is named **exactly** `key.properties`.

1. **Rename the file:** Go to your `android/` folder.
2. Rename `key.txt` to `key.properties`.
3. **Confirm the contents** are still correct:

Properties

```
storePassword=YOUR_KEYSTORE_PASSWORD
keyPassword=YOUR_KEY_PASSWORD
keyAlias=upload
storeFile=C:\\Users\\Sharon\\upload-keystore\\key.jks
```

4. **Rerun the build:**

Bash

```
flutter build apk --release
```

Renaming the file should fix the error and allow the build to complete successfully.

is .properties its extensior

◆ For this context, `.properties` **is the file extension**, and it is crucial that the file be named `key.properties`.

Here's why this distinction matters to the Android build system (Gradle):

- **File Extension:** The text after the last dot in a filename, which tells the operating system and other programs what type of file it is. In the case of configuration files used by Java/Gradle, th `.properties`.

- **Full Filename:** The program is looking for the exact filename `key.properties` .

If you had a file named `key.txt` , the filename is `key` , and the extension is `.txt` .

If you rename the file to `key.properties` , the filename is `key` , and the extension is `.properties` .

Because the build script is looking specifically for the full string `"key.properties"` , the file must be named that way for the build to pass. Please ensure the file is named `key.properties` or `key.properties.txt` or `key.txt` .

okay i underst

That's a very common mistake! Because you left the file named as a part of the file name (e.g., `key.properties.txt` or you manually added quotes), the build system couldn't find the exact f for ( `key.properties` ).

Since the build is failing because it can't load the properties file, let's assume you have successfully renamed the file to `key.properties` in your `android/` folder, and run the build one more

If you encounter a different error, please provide the new output. If the build succeeds, you will have completed your Android deployment.

### Final Step: Rerun the Build

In your project directory, run the final command:

Bash

```
flutter build apk --release
```

**If successful:** You will see a success message with the path to the `app-release.apk` file.

**If it fails again:** Please copy the **new error message**.

```
Sharon@DESKTOP-JU96MA6 MINGW64 ~/number_converter_app (main)
$ flutter build apk --release
Checking the license for package NDK (Side by side) 27.0.12077973 in C:\Users\Sharon\AppData\Local\Android\sdk\licenses
License for package NDK (Side by side) 27.0.12077973 accepted.
Preparing "Install NDK (Side by side) 27.0.12077973 v.27.0.12077973".
"Install NDK (Side by side) 27.0.12077973 v.27.0.12077973" ready.
Installing NDK (Side by side) 27.0.12077973 in C:\Users\Sharon\AppData\Local\Android\sdk\ndk\27.0.12077973
"Install NDK (Side by side) 27.0.12077973 v.27.0.12077973" complete.
"Install NDK (Side by side) 27.0.12077973 v.27.0.12077973" finished.
Checking the license for package Android SDK Build-Tools 35 in C:\Users\Sharon\AppData\Local\Android\sdk\licenses
License for package Android SDK Build-Tools 35 accepted.
Preparing "Install Android SDK Build-Tools 35 v.35.0.0".
"Install Android SDK Build-Tools 35 v.35.0.0" ready.
Installing Android SDK Build-Tools 35 in C:\Users\Sharon\AppData\Local\Android\sdk\build-tools\35.0.0
"Install Android SDK Build-Tools 35 v.35.0.0" complete.
"Install Android SDK Build-Tools 35 v.35.0.0" finished.
Font asset "MaterialIcons-Regular.otf" was tree-shaken, reducing it from 1645184 to 1532 bytes (99.9% reduction). Tree-shaking can be disabled by providing the --no-tree-shake-ico
your app.
Checking the license for package CMake 3.22.1 in C:\Users\Sharon\AppData\Local\Android\sdk\licenses
License for package CMake 3.22.1 accepted.
Preparing "Install CMake 3.22.1 v.3.22.1".
"Install CMake 3.22.1 v.3.22.1" ready.
Installing CMake 3.22.1 in C:\Users\Sharon\AppData\Local\Android\sdk\cmake\3.22.1
"Install CMake 3.22.1 v.3.22.1" complete.
"Install CMake 3.22.1 v.3.22.1" finished.
Running Gradle task 'assembleRelease'...                    519.1s
√ Built build\app\outputs\flutter-apk\app-release.apk (43.2MB)

Sharon@DESKTOP-JU96MA6 MINGW64 ~/number_converter_app (main)
$ ^C
```

That is **perfect!** 🎉

The build process completed successfully, and you have achieved the Android deployment requirement for your assignment.

### Final Build Status

The final lines confirm success:

```
√ Built build\app\outputs\flutter-apk\app-release.apk (43.2MB)
```

This means your signed, optimized, and runnable Android application file is now available at the following path:

```
C:\Users\Sharon\number_converter_app\build\app\outputs\flutter-apk\app-release.apk
```

### Next Steps to Finalize Your Submission

With the Android deployment complete, you should focus on the remaining requirements listed in your assignment.



1. iOS Deployment (If Applicable)

Your assignment states the app "Should be deployed to android an ios platforms".

- **Requirement:** To complete the iOS part, you **must use a macOS computer** with Xcode and an active Apple Developer account to build the `.ipa` file.
- **Action:** If you have access to a Mac, you need to run `flutter build ipa --release` and then use Xcode to finish the signing and archival process. If you do not have a Mac, you have cr requirement and should include a note in your documentation explaining the inability to complete the iOS build due to platform limitations.

2. Documentation Submission

The assignment requires you to submit both documentation types:

- **Process Documentation:** Detail your group's workflow, design choices, how you used Git/GitHub for collaboration, and the steps taken to deploy to Android (including the key generation performed).
- **Program Documentation:** Provide a technical overview of your code, including the logic used for the number system conversions in your Dart files.

3. Final Submission Package


Ensure you submit the following items to your lecturer:

- The **URL** to your GitHub repository (which has the lecturer as a collaborator).
- The **Process and Program Documentation** files.
- The final, runnable Android APK file: `app-release.apk` .

can you write me a detailed documentation of the steps and troubles weve gone through during this de

That's an excellent idea. A detailed process document that includes the challenges and how you overcame them significantly strengthens your final submission, demonstrating robust problem-

Here is a draft of your **Process Documentation**, focusing specifically on the collaboration and deployment hurdles your team successfully navigated.


 **Process Documentation: Development and Deployment Log**  
29 Sept, 11:17

This document provides clear evidence of your successful execution and troubleshooting during the deployment phase, which is exactly what a lecturer looks for. Let me know if you need any dive into the **Program Documentation** next!

do also the

This is the **Program Documentation**, which focuses on the technical aspects of the application itself—the architecture, the main logic, and the functions responsible for the number conversic

Since I don't have access to your actual Dart code, I will provide the documentation based on the standard structure and expected logic for a number converter application built with Flutter. Yc logic descriptions align with your code inside the `lib/` directory.

 **Program Documentation: Technical Overview**  
29 Sept, 01:27

This documentation thoroughly explains the separation of logic, the mathematical foundation of the conversion, and the importance of input validation. It should satisfy the requirement for "Pr

Do you want to review anything else, or are you ready to compile your final submission package?

how can

That's an excellent question! Since you are working in a collaborative environment (the canvas/editor where these files are displayed), you generally cannot *directly* "download" files in the same standard website.

However, there are two easy ways to get the content of these files, `process_log.md` and `program_doc.md` , onto your local computer:

Option 1: Copy and Paste (Recommended)

This is the fastest and easiest method to get the content into local files:

1. **Select All:** Click inside the editor displaying `process_log.md` , select all the text, and copy it.
2. **Create File:** Open a new text file on your computer (using Notepad, VS Code, or another editor).
3. **Paste and Save:** Paste the content and save the file on your local machine as `process_log.md` .
4. **Repeat:** Do the same three steps for the content of `program_doc.md` , saving it as `program_doc.md` .

Option 2: Use the Export/Save Feature (If Available)

Depending on the platform you are using, there may be a built-in feature to download or export the files in the editor pane.

- Look for icons near the title of the document (like a down arrow, a floppy disk, or an "Export" button). If you find one, use it to save the Markdown file directly.

Using the **Copy and Paste** method (Option 1) will ensure you get the exact, final versions of both documents for your submission.

WRITE ME THE PROCESS DOCUMENTATION IN A COPY PASTABLE FORMAT AND BE ABLE TO KEEP ITS FC

✦ I'd be happy to give you the `process_log.md` file again in a format that makes it easy to copy the raw text, ensuring all the Markdown formatting (like the tables and headers) is preserved when you save it as a local `.md` file.

Here is the document for you to copy and paste:

Just copy everything above, starting from `# Process Documentation...`, and save it as a file named `process_log.md` on your local computer.