How to Use This Script:

1. **Open Google Sheets:** Go to your Google Sheet where you want to manage the email.
2. **Open Apps Script Editor:**
    - Click on Extensions in the top menu.
    - Select Apps Script. This will open a new browser tab with the Apps Script editor.
3. **Paste the Code found below these directions:**
    - In the Apps Script editor, you'll see a default Code.gs file. Delete any existing code in it.
    - Copy and paste the entire script provided above into the Code.gs file.
4. **Save the Script:** Click the floppy disk icon (Save project). You might be prompted to give your project a name (e.g., "Weekly Email Sender").
5. **Prepare Your Sheet:**
    - Go back to your Google Sheet.
    - **Create a new sheet** and name it Email Data (or whatever you set sheetName to in the script).
    - In this Email Data sheet:
        - **Cell A1:** Type the recipient's email address (e.g., your.email@example.com).
        - **Cell B1:** Type the subject of your email (e.g., Weekly Project Update).
        - **Cell C1:** Type the body of your email. You can use \n\n for new paragraphs.
6. **Run createWeeklyTrigger (First Time Only):**
    - Go back to the Apps Script editor.
    - In the toolbar, there's a dropdown menu that usually says (Select function). Click it and select createWeeklyTrigger.
    - Click the Run button (a triangle icon, similar to a play button).
    - **Authorization:** The first time you run any script that interacts with your Google services, you'll be prompted to authorize it.
        - Click Review permissions.
        - Select your Google account.
        - Click Allow to grant the necessary permissions (to send emails and manage triggers).
    - After authorization, the createWeeklyTrigger function will run and set up the weekly schedule. You'll see a success message in the execution log or an alert in the sheet.
7. **Test sendWeeklyEmail (Optional):**
    - In the Apps Script editor, select sendWeeklyEmail from the function dropdown.
    - Click the Run button. This will send an email immediately based on the data in your Email Data sheet. Check your inbox (or the recipient's inbox) to confirm it worked.

Important Notes:

- **Customization:** Remember to adjust the sheetName and cell references (A1, B1, C1) in the sendWeeklyEmail function if your email data is located elsewhere or on a different sheet name.
- **Trigger Time:** The createWeeklyTrigger function is set to run every Monday between 9 AM and 10 AM. You can modify onWeekDay(ScriptApp.WeekDay.MONDAY) to TUESDAY, WEDNESDAY, etc., and atHour(9) to any hour from 0 (midnight) to 23 (11 PM).
- **Error Handling:** The script includes basic error handling and will display alerts in the sheet if it encounters issues (e.g., sheet not found, missing data).
- **deleteAllTriggers:** I've also included a deleteAllTriggers function. If you ever need to remove all

triggers associated with this specific script project (e.g., to reset the schedule or if you no longer need the automation), you can run this function in the same way you ran createWeeklyTrigger.

```
/**
 * @fileoverview This script automates sending weekly emails from Google Sheets using Gmail.
 * It reads email details (recipient, subject, body) from a specified sheet and sends the email.
 * A time-driven trigger can be set up to run this function weekly.
 */

/**
 * Sends a weekly email using data from a Google Sheet.
 *
 * This function assumes the following data structure in the sheet named "Email Data":
 * - Cell A1: Recipient Email Address (e.g., "your.email@example.com")
 * - Cell B1: Email Subject (e.g., "Weekly Report Update")
 * - Cell C1: Email Body (e.g., "Hi team,\n\nHere is the weekly report. Please review.\n\nThanks!")
 *
 * Make sure to adjust the sheet name and cell references if your data is organized differently.
 */
function sendWeeklyEmail() {
  try {
    // Get the active spreadsheet.
    const spreadsheet = SpreadsheetApp.getActiveSpreadsheet();

    // Get the sheet where email data is stored.
    // IMPORTANT: Change 'Email Data' to the actual name of your sheet.
    const sheetName = 'Email Data';
    const sheet = spreadsheet.getSheetByName(sheetName);

    // Check if the sheet exists.
    if (!sheet) {
      Logger.log(`Error: Sheet '${sheetName}' not found. Please create a sheet with this name.`);
```

```
      SpreadsheetApp.getUi().alert('Error', `Sheet '${sheetName}' not found. Please create a sheet with this name.`, SpreadsheetApp.getUi().ButtonSet.OK);
      return;
    }

    // Read email details from specific cells.
    // IMPORTANT: Adjust these cell references (e.g., 'A1', 'B1', 'C1') if your data is elsewhere.
    const recipient = sheet.getRange('A1').getValue(); // Cell A1 for recipient
    const subject = sheet.getRange('B1').getValue();   // Cell B1 for subject
    const body = sheet.getRange('C1').getValue();      // Cell C1 for body

    // Basic validation to ensure all required fields are present.
    if (!recipient || !subject || !body) {
      Logger.log('Error: Recipient, Subject, or Body is missing in the sheet.');
      SpreadsheetApp.getUi().alert('Error', 'Recipient, Subject, or Body is missing in the sheet. Please ensure cells A1, B1, and C1 contain data.', SpreadsheetApp.getUi().ButtonSet.OK);
      return;
    }

    // Send the email using GmailApp.
    GmailApp.sendEmail(recipient, subject, body);

    Logger.log(`Email sent successfully to ${recipient} with subject: "${subject}"`);
    SpreadsheetApp.getUi().alert('Success', `Email sent successfully to ${recipient}!`, SpreadsheetApp.getUi().ButtonSet.OK);

  } catch (e) {
    // Log any errors that occur during execution.
    Logger.log(`Failed to send email: ${e.toString()}`);
    SpreadsheetApp.getUi().alert('Error', `Failed to send email: ${e.message}`, SpreadsheetApp.getUi().ButtonSet.OK);
  }
}

/**
 * Creates a weekly time-driven trigger for the 'sendWeeklyEmail' function.
 *
```

```
 * This function should be run once to set up the trigger.
 * It will schedule 'sendWeeklyEmail' to run every Monday between 9 AM and 10 AM.
 * You can modify the day and time as needed.
 */
function createWeeklyTrigger() {
  try {
    // Delete any existing triggers for this function to prevent duplicates.
    const triggers = ScriptApp.getProjectTriggers();
    for (let i = 0; i < triggers.length; i++) {
      if (triggers[i].getHandlerFunction() === 'sendWeeklyEmail') {
        ScriptApp.deleteTrigger(triggers[i]);
      }
    }

    // Create a new weekly trigger.
    // This example sets it to run every Monday between 9 AM and 10 AM.
    ScriptApp.newTrigger('sendWeeklyEmail')
      .timeBased()
      .everyWeeks(1) // Run every week
      .onWeekDay(ScriptApp.WeekDay.MONDAY) // Specify the day of the week
      .atHour(9) // Specify the hour (e.g., 9 for 9 AM)
      .create();

    Logger.log('Weekly trigger created successfully for sendWeeklyEmail.');
    SpreadsheetApp.getUi().alert('Success', 'Weekly email trigger has been set up!
It will run every Monday between 9 AM and 10 AM.',
SpreadsheetApp.getUi().ButtonSet.OK);

  } catch (e) {
    Logger.log(`Failed to create trigger: ${e.toString()}`);
    SpreadsheetApp.getUi().alert('Error', `Failed to create trigger: ${e.message}`,
SpreadsheetApp.getUi().ButtonSet.OK);
  }
}

/**
 * Deletes all triggers associated with this Apps Script project.
 * Useful for cleaning up or resetting triggers.
 */
function deleteAllTriggers() {
```

```
  const triggers = ScriptApp.getProjectTriggers();
  for (let i = 0; i < triggers.length; i++) {
    ScriptApp.deleteTrigger(triggers[i]);
  }
  Logger.log('All project triggers deleted.');
  SpreadsheetApp.getUi().alert('Success', 'All project triggers have been
deleted.', SpreadsheetApp.getUi().ButtonSet.OK);
}
```