Birthday Automation Code Module 1

```
/**
 * Global constant for the Gemini model to use.
 * gemini-2.5-flash is fast and perfect for this kind of creative text generation.
 */
const GEMINI_MODEL = "gemini-2.5-flash";

/**
 * The recipient email address for all birthday notifications.
 * CHANGE THIS TO YOUR EMAIL ADDRESS.
 */
const RECIPIENT_EMAIL = "CHANGE THIS TO YOUR EMAIL ADDRESS";

/**
 * Main function run daily by a time-driven trigger.
 * It checks the spreadsheet for today's birthdays, generates AI messages,
 * and sends a single consolidated email if any birthdays are found.
 */
function sendBirthdayReminders() {
  Logger.log("Starting birthday check...");

  // 1. Setup Spreadsheet and Time Variables
  const spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
  // NOTE: If you changed the sheet name in Google Sheets, update 'Birthdays' below.
  const sheet = spreadsheet.getSheetByName('Birthdays');

  if (!sheet) {
    MailApp.sendEmail(RECIPIENT_EMAIL, "BIRTHDAY REMINDER ERROR", "Could not find the 'Birthdays' sheet. Please check the sheet name.");
    return;
```

```
  }

  // Get all data, excluding the header row
  const dataRange = sheet.getRange(2, 1, sheet.getLastRow() - 1, sheet.getLastColumn());
  const data = dataRange.getValues();
  const today = new Date();
  const todayMMDD = (today.getMonth() + 1).toString().padStart(2, '0') + '/' +
today.getDate().toString().padStart(2, '0');
  const currentYear = today.getFullYear();

  // Array to hold today's birthday details
  const todaysBirthdays = [];

  // 2. Filter for today's birthdays and check the last sent year
  data.forEach((row, index) => {
    // Columns: [Name, Birthday (MM/DD), Classification, Last Sent Year, Notes/Context]
    const [name, birthday, classification, lastSentYear, notes] = row;

    // Ensure the birthday column is formatted as a string MM/DD for comparison
    const birthdateString = birthday instanceof Date ?
      (birthday.getMonth() + 1).toString().padStart(2, '0') + '/' + birthday.getDate().toString().padStart(2,
'0') :
      String(birthday).trim();

    // Check if the date matches today (MM/DD) and if an email hasn't been sent this year
    if (birthdateString === todayMMDD && lastSentYear !== currentYear) {
      // Add data for processing and update the sheet to prevent resending today
      todaysBirthdays.push({
        name: name,
        classification: classification,
        notes: notes,
```

```
    rowIndex: index + 2 // +2 because the array is 0-indexed and data starts at row 2
   });
  }
});

// 3. Process Birthdays and Send Email
if (todaysBirthdays.length === 0) {
  Logger.log("No birthdays today. No email sent.");
  return;
}

// --- Start Email Content Construction ---
let emailBody = `🎉 **TODAY'S BIRTHDAY REMINDERS (${todayMMDD})** 🎉\n\n`;
emailBody += "--- Summary ---\n";
const aiMessagePromises = [];

todaysBirthdays.forEach(person => {
  const classificationText = getClassificationLabel(person.classification);
  emailBody += `\n**${person.name}**\n`;
  emailBody += `  - **Category:** ${classificationText} (${person.classification})\n`;
  emailBody += `  - **Context:** ${person.notes || 'None provided.'}\n`;

  // Queue up AI message generation (runs sequentially)
  aiMessagePromises.push(generateGeminiMessage(person));
});

// 4. Wait for all AI messages to be generated
Logger.log(`Generating ${aiMessagePromises.length} unique AI messages...`);
// The structure of this call now expects an array of {name, fetcher} objects
const generatedMessages = generateAllMessages(aiMessagePromises);
```

```
  emailBody += "\n\n--- AI-Generated Messages ---\n\n";

  // 5. Integrate AI Messages into the email body
  generatedMessages.forEach(result => {
    const { name, message } = result;
    emailBody += `## Message for ${name}:\n`;
    emailBody += '-------------------------------------------------\n';
    emailBody += message;
    emailBody += '\n-------------------------------------------------\n\n';
  });

  // 6. Send the final consolidated email
  MailApp.sendEmail({
    to: RECIPIENT_EMAIL,
    subject: `🎂 Birthday Alert: ${todaysBirthdays.length} Birthday(s) Today! (${todayMMDD})`,
    body: emailBody
  });

  Logger.log(`Successfully sent email with ${todaysBirthdays.length} birthday(s).`);

  // 7. Update the "Last Sent Year" column in the spreadsheet
  const rowsToUpdate = todaysBirthdays.map(p => p.rowIndex);
  rowsToUpdate.forEach(rowIndex => {
    sheet.getRange(rowIndex, 4).setValue(currentYear); // Column D (index 4) is Last Sent Year
  });

  Logger.log(`Successfully updated ${rowsToUpdate.length} rows to ${currentYear}.`);
}

/**
 * Uses a basic sequential approach with UrlFetchApp to run message generations.
```

```
 * It iterates over an array of { name, fetcher } objects.
 * @param {Array<{name: string, fetcher: Function}>} promises An array of objects containing the person's name and the function to execute the API call.
 * @returns {Array<Object>} Array of results { name, message }.
 */
function generateAllMessages(promises) {
  const responses = [];
  // promises is an array of { name: string, fetcher: Function } objects
  promises.forEach(item => {
    const personName = item.name;
    let rawResponseText = null;
    let httpStatus = 0;

    try {
      const response = item.fetcher();
      httpStatus = response.getResponseCode();
      rawResponseText = response.getContentText();

      // Check for non-200 error code first
      if (httpStatus !== 200) {
        // This is the true API error we need to log
        Logger.log(`API FAILURE for ${personName}: Status ${httpStatus}. Raw Response: ${rawResponseText}`);

        // Attempt to parse the error message if it's JSON (most 4xx/5xx errors are)
        let errorMessage = `API Call failed with HTTP Status ${httpStatus}. See logs for details.`;
        try {
          const errorData = JSON.parse(rawResponseText);
          errorMessage = `API Error (${httpStatus}): ${errorData.error?.message || 'Unknown API Error'}`;
        } catch (e) {
          // If the error response is not JSON, just include the raw text
```

```
      errorMessage = `API Error (${httpStatus}): Raw response was not JSON. Server response: ${rawResponseText.substring(0, 100)}...`;
    }

    responses.push({
      name: personName,
      message: `AI message generation failed. ${errorMessage}`
    });
    return; // Move to the next person
  }

  // SUCCESS (HTTP 200) logic:
  const data = JSON.parse(rawResponseText);

  let message = "AI message generation failed (no content found).";
  if (data.candidates && data.candidates[0] && data.candidates[0].content && data.candidates[0].content.parts) {
    message = data.candidates[0].content.parts[0].text.trim();
  }
  responses.push({ name: personName, message: message });

 } catch (e) {
  // Handles rare network/system exceptions (not typical API key errors)
  Logger.log(`SYSTEM EXCEPTION for ${personName}: ${e.toString()}`);
  responses.push({
    name: personName,
    message: `ERROR: Failed to generate message (System Exception). Details: ${e.message}`
  });
 }
});
return responses;
```

```
}

/**
 * Generates the specific prompt and prepares the API call structure.
 * Returns an object containing the person's name and a function to execute the UrlFetchApp.
 * @param {Object} person The person object with name, classification, and notes.
 * @returns {{name: string, fetcher: Function}} An object with the person's name and the fetcher function.
 */
function generateGeminiMessage(person) {
  const { name, classification, notes } = person;
  let systemInstruction = "";
  let userQuery = "";

  // Ensure notes is a safe string for JSON inclusion
  const safeNotes = String(notes || "no specific interests").replace(/"/g, "'");

  if (classification === 5) {
    // Deceased (Reflective Message)
    systemInstruction = `You are a thoughtful and sensitive writer. Write a short, reflective, and touching message (2-3 sentences max) that focuses on gratitude, positive memories, and the enduring impact of the person named ${name}. Do not mention death or the term 'deceased'. The tone should be gentle and uplifting, perfect for private reflection.`;
    userQuery = `Write a reflective message about ${name} based on the following context: "${safeNotes}"`;
  } else {
    // All other active classifications (1-4)
    const classificationText = getClassificationLabel(classification);
    systemInstruction = `You are a creative and warm friend. Generate a unique, personalized birthday message for ${name}. The message should be warm, concise (2-3 sentences), and suitable for a ${classificationText}. Use the provided context to make it specific and avoid generic phrases like 'I hope you have a great day.' Just provide the message text, nothing else.`;
    userQuery = `Write a personalized birthday wish for ${name}. Context/Interests: "${safeNotes}".`;
```

```
}

// Add diagnostic log for the exact query being sent
Logger.log(`Prompt for ${name} (Classification ${classification}): ${userQuery}`);

const apiKey = PropertiesService.getScriptProperties().getProperty('GEMINI_API_KEY');
const url =
`https://generativelanguage.googleapis.com/v1beta/models/${GEMINI_MODEL}:generateContent`;

const payload = {
  contents: [{ parts: [{ text: userQuery }] }],
  systemInstruction: { parts: [{ text: systemInstruction }] },
  // Correct API field name
  generationConfig: {
    temperature: 0.8, // Use a higher temperature for creative, unique messages
    // maxOutputTokens: 200 // Optional: limit length
  },
};

const options = {
  method: 'post',
  contentType: 'application/json',
  headers: { 'x-goog-api-key': apiKey },
  payload: JSON.stringify(payload),
  muteHttpExceptions: true // Required to capture non-200 responses
};

// Return a function that executes the call. We execute this later in generateAllMessages.
const apiCallFunction = () => {
  // Executes the API call
  return UrlFetchApp.fetch(url, options);
```

```
  };

  // Return the name and the fetcher function
  return { name: name, fetcher: apiCallFunction };
}

/**
 * Helper function to return the full classification name.
 * @param {number} classification The classification number (1-5).
 * @returns {string} The descriptive label.
 */
function getClassificationLabel(classification) {
  switch (classification) {
    case 1: return "Close Family";
    case 2: return "Extended Family";
    case 3: return "Close Friend";
    case 4: return "Friend";
    case 5: return "Deceased (Reflection)";
    default: return "Unknown Category";
  }
}

/**
 * Sets up a daily time-driven trigger for the main function.
 * RUN THIS FUNCTION *ONCE* TO START THE AUTOMATION.
 */
function createDailyTrigger() {
  // Delete any existing triggers to prevent duplicates
  const triggers = ScriptApp.getProjectTriggers();
  triggers.forEach(trigger => {
    if (trigger.getHandlerFunction() === 'sendBirthdayReminders') {
```

```
    ScriptApp.deleteTrigger(trigger);
  }
 });

 // Create a new daily trigger (runs between 7 AM and 8 AM)
 ScriptApp.newTrigger('sendBirthdayReminders')
    .timeBased()
    .everyDays(1)
    .atHour(7) // You can adjust the hour (e.g., 6 for 6 AM)
    .create();

 Logger.log("Daily birthday reminder trigger created successfully.");
 // Optional: Send a confirmation email that the trigger is set
 MailApp.sendEmail(RECIPIENT_EMAIL, "Birthday App Setup Complete!", "Your daily birthday reminder app trigger has been created and will run every day between 7 AM and 8 AM.");
}

/**
 * NEW DIAGNOSTIC FUNCTION: Tests the API key validity directly.
 * Run this function and check the logs for the outcome.
 */
function testApiKey() {
 Logger.log("--- Starting API Key Test ---");
 const apiKey = PropertiesService.getScriptProperties().getProperty('GEMINI_API_KEY');
 const url = `https://generativelanguage.googleapis.com/v1beta/models/${GEMINI_MODEL}:generateContent`;

 if (!apiKey) {
  Logger.log("ERROR: GEMINI_API_KEY not found in Script Properties. Please add it.");
  return;
 }
```

```
 const payload = {
  contents: [{ parts: [{ text: "Explain why this test call should succeed in 3 words." }] }],
 };

 const options = {
  method: 'post',
  contentType: 'application/json',
  headers: { 'x-goog-api-key': apiKey },
  payload: JSON.stringify(payload),
  muteHttpExceptions: true
 };

 try {
  const response = UrlFetchApp.fetch(url, options);
  const rawResponseText = response.getContentText();

  // Check for success (200 OK)
  if (response.getResponseCode() === 200) {
   const data = JSON.parse(rawResponseText);
   const generatedText = data.candidates?.[0]?.content?.parts?.[0]?.text?.trim() || "Could not extract text.";

   Logger.log("API TEST RESULT: SUCCESS!");
   Logger.log(`Generated Text: "${generatedText}"`);
   MailApp.sendEmail(RECIPIENT_EMAIL, "API Key Test SUCCESS!", "The Gemini API call succeeded. The problem is likely in your spreadsheet data or configuration.");
  } else {
   // Log failure response for diagnosis
   Logger.log(`API TEST RESULT: FAILURE! HTTP Status: ${response.getResponseCode()}`);
   Logger.log(`RAW FAILURE RESPONSE: ${rawResponseText}`);
```

```
    MailApp.sendEmail(RECIPIENT_EMAIL, "API Key Test FAILED", `The Gemini API call failed with HTTP Status ${response.getResponseCode()}. Check the Apps Script logs for the RAW FAILURE RESPONSE to see the exact error message (e.g., invalid key, billing, or quota).`);
   }
  } catch (e) {
   Logger.log(`API TEST EXCEPTION: ${e.toString()}`);
   MailApp.sendEmail(RECIPIENT_EMAIL, "API Key Test EXCEPTION", `A network or system error prevented the API call. Exception: ${e.toString()}`);
  }
  Logger.log("--- API Key Test Complete ---");
}
```