Birthday Agent Code Module 2

```
/**
 * 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 notifications.
 */
const RECIPIENT_EMAIL = "CHANGE THIS TO YOUR EMAIL ADDRESS";
/**
 * --- MAIN ENTRY POINT (CALLED BY DAILY TRIGGER) ---
 * Checks for both gift reminders (14 days out) and birthday messages (today).
 */
function mainDailyCheck() {
 Logger.log("--- Starting Two-Stage Daily Check ---");
 const today = new Date();
 sendGiftReminders(today);
 sendBirthdayMessages(today);

 Logger.log("--- Daily Check Complete ---");
}
/**
 * Checks for birthdays exactly 14 days from today for Classifications 1 & 3,
 * generates gift suggestions, and sends a gift reminder email.
 * @param {Date} today The current date.
 */
function sendGiftReminders(today) {
 Logger.log("Starting Gift Reminder Check...");
 const spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
 const sheet = spreadsheet.getSheetByName('Birthdays');
```

```
if (!sheet) {
Logger.log("ERROR: Sheet not found in sendGiftReminders.");
return;
}
// Define the target date: 14 days from now
const fourteenDaysOut = new Date(today);
fourteenDaysOut.setDate(today.getDate() + 14);
const targetMMDD = (fourteenDaysOut.getMonth() + 1).toString().padStart(2, '0') + '/' +
fourteenDaysOut.getDate().toString().padStart(2, '0');
const currentYear = today.getFullYear();
// Get data including the NEW 6th column (Last Gift Sent Year)
const dataRange = sheet.getRange(2, 1, sheet.getLastRow() - 1, sheet.getLastColumn());
const data = dataRange.getValues();

const birthdaysDueGifts = [];

// Columns: [Name, Birthday(1), Class(2), LastSentYear(3), Notes(4), LastGiftSentYear(5)]
data.forEach((row, index) => {
const [name, birthday, classification, , notes, lastGiftSentYear] = 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 1: Is today 14 days before the birthday?
const dateMatches = (birthdateString === targetMMDD);

// Check 2: Is it Classification 1 or 3?
const isPriority = (classification === 1 || classification === 3);
```

```
// Check 3: Has the gift reminder not been sent this year?
const notSentThisYear = (lastGiftSentYear !== currentYear);
if (dateMatches && isPriority && notSentThisYear) {
birthdaysDueGifts.push({
name: name,
classification: classification,
notes: notes,
rowIndex: index + 2 // Row in sheet
});
}
});
if (birthdaysDueGifts.length === 0) {
Logger.log("No gift reminders due today.");
return;
}

// --- Process and Send Gift Reminder Email ---

let emailBody = `蘆薈蔏蔏 **GIFT REMINDER: ${birthdaysDueGifts.length} BIRTHDAY(S) IN 2 WEEKS
(${targetMMDD})** 蘆薈蔏蔏\n\n`;
emailBody += "--- Gift Suggestions ---\n";
const aiGiftPromises = [];

birthdaysDueGifts.forEach(person => {
emailBody += `\n**${person.name}** (${getClassificationLabel(person.classification)})\n`;
emailBody += ` - Context: ${person.notes || 'None provided.'}\n`;
aiGiftPromises.push(generateGiftSuggestions(person));
});
const generatedGifts = generateAllMessages(aiGiftPromises);
```

```
 emailBody += "\n\n--- AI-Generated Recommendations ---\n\n";
 generatedGifts.forEach(result => {
 emailBody += `## Gift Ideas for ${result.name}:\n`;
 emailBody += '-------------------------------------------------\n';
 emailBody += result.message;
 emailBody += '\n-------------------------------------------------\n\n';
 });
 MailApp.sendEmail({
 to: RECIPIENT_EMAIL,
 subject: `麓鼓薦薦 2 Week Birthday Reminder: ${birthdaysDueGifts.length} Birthday(s) Coming Up!
(${targetMMDD})`,
 body: emailBody
 });
 Logger.log(`Successfully sent gift reminder email for ${birthdaysDueGifts.length} birthday(s).`);
 // Update the "Last Gift Sent Year" column (Column F, index 6)
 birthdaysDueGifts.forEach(person => {
 sheet.getRange(person.rowIndex, 6).setValue(currentYear);
 });
 Logger.log(`Successfully updated ${birthdaysDueGifts.length} rows for Last Gift Sent Year.`);
}
/**
 * Checks for birthdays exactly today, generates the AI message, and sends the final email.
 * @param {Date} today The current date.
 */
function sendBirthdayMessages(today) {
 Logger.log("Starting Birthday Message Check...");

 const spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
 const sheet = spreadsheet.getSheetByName('Birthdays');
```

```
if (!sheet) {
Logger.log("ERROR: Sheet not found in sendBirthdayMessages.");
return;
}
const todayMMDD = (today.getMonth() + 1).toString().padStart(2, '0') + '/' +
today.getDate().toString().padStart(2, '0');
const currentYear = today.getFullYear();
// Get data including the original 5 columns (Last Sent Year is column D, index 4)
const dataRange = sheet.getRange(2, 1, sheet.getLastRow() - 1, sheet.getLastColumn());
const data = dataRange.getValues();

const birthdaysDueMessage = [];
// Columns: [Name, Birthday(1), Class(2), LastSentYear(3), Notes(4), LastGiftSentYear(5)]
data.forEach((row, index) => {
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 1: Is today the exact birthday?
const dateMatches = (birthdateString === todayMMDD);

// Check 2: Has the main birthday email not been sent this year? (Column D)
const notSentThisYear = (lastSentYear !== currentYear);
if (dateMatches && notSentThisYear) {
birthdaysDueMessage.push({
name: name,
```

```
classification: classification,
notes: notes,
rowIndex: index + 2 // Row in sheet
});
}
});
if (birthdaysDueMessage.length === 0) {
Logger.log("No birthday messages due today.");
return;
}

// --- Process and Send Birthday Message Email ---

let emailBody = `蔑薰蔓蔔蒂蘆蔗蔘旋 **BIRTHDAY MESSAGE ALERT:
${birthdaysDueMessage.length} BIRTHDAY(S)
TODAY!** 蔑薰蔓蔔蒂蘆蔗蔘旋\n\n`;
emailBody += "--- AI-Generated Messages ---\n";
const aiMessagePromises = [];

birthdaysDueMessage.forEach(person => {
aiMessagePromises.push(generateGeminiMessage(person));
});
const generatedMessages = generateAllMessages(aiMessagePromises);

emailBody += "\n\n--- Messages Ready to Send ---\n\n";
generatedMessages.forEach(result => {
const person = birthdaysDueMessage.find(p => p.name === result.name);
const classification = person ? person.classification : 0;
const classificationLabel = getClassificationLabel(classification);

emailBody += `## Message for ${result.name} (${classificationLabel}):\n`;
```

```
 emailBody += '-------------------------------------------------\n';
 emailBody += result.message;
 emailBody += '\n-------------------------------------------------\n';

 if (classification === 5) {
 emailBody += '\n*(Reflective message. Send with care.)*\n';
 } else {
 emailBody += '\n*(Ready to copy and paste!)*\n';
 }
 emailBody += '-------------------------------------------------\n\n';
 });
 MailApp.sendEmail({
 to: RECIPIENT_EMAIL,
 subject: `蔑蔥蔓蔔蒂蘆蔗蔘蔌 Birthday Alert! Send a Message to ${birthdaysDueMessage.map(p => p.name).join(', ')}
Today!`,
 body: emailBody
 });
 Logger.log(`Successfully sent birthday message email for ${birthdaysDueMessage.length} birthday(s).`);
 // Update the "Last Sent Year" column (Column D, index 4)
 birthdaysDueMessage.forEach(person => {
 sheet.getRange(person.rowIndex, 4).setValue(currentYear);
 });
 Logger.log(`Successfully updated ${birthdaysDueMessage.length} rows for Last Sent Year.`);
}
/**
 * Helper function to find a person's classification by name.
 * (Not strictly necessary now, but good for debugging/future use)
 * @param {string} name
 * @param {Array<Object>} list
```

```
 * @returns {number | undefined}
 */
function getPersonClassification(name, list) {
 const person = list.find(p => p.name === name);
 return person ? person.classification : undefined;
}
/**
 * Reusable function to handle parallel API calls and error parsing.
 */
function generateAllMessages(promises) {
 const responses = [];
 promises.forEach(item => {
 const personName = item.name;
 let rawResponseText = null;
 let httpStatus = 0;
 try {
 const response = item.fetcher();
 httpStatus = response.getResponseCode();
 rawResponseText = response.getContentText();

 if (httpStatus !== 200) {
 Logger.log(` API FAILURE for ${personName}: Status ${httpStatus}. Raw Response:
${rawResponseText}` );

 let errorMessage = ` API Call failed with HTTP Status ${httpStatus}. See logs for details.` ;
 try {
 const errorData = JSON.parse(rawResponseText);
 errorMessage = ` API Error (${httpStatus}): ${errorData.error?.message || 'Unknown API Error'}` ;
 } catch (e) {
 errorMessage = ` API Error (${httpStatus}): Raw response was not JSON. Server response:
${rawResponseText.substring(0, 100)}...` ;
```

```
}
responses.push({
name: personName,
message: `AI generation failed. ${errorMessage}`
});
return;
}

const data = JSON.parse(rawResponseText);
let message = "AI 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) {
Logger.log(`SYSTEM EXCEPTION for ${personName}: ${e.toString()}`);
responses.push({
name: personName,
message: `ERROR: Failed to generate content (System Exception). Details: ${e.message}`
});
}
});
return responses;
}
/**
* Generates the specific prompt and prepares the API call structure for the message.
*/
function generateGeminiMessage(person) {
const { name, classification, notes } = person;
let systemInstruction = "";
```

```
let userQuery = "";

const safeNotes = String(notes || "no specific interests").replace(/"/g, "'");
if (classification === 5) {
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 {
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}".`;
}

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 }] },
generationConfig: {
temperature: 0.8,
},
};
const options = getApiOptions(apiKey, payload);
```

```
 const apiCallFunction = () => UrlFetchApp.fetch(url, options);
 return { name: name, fetcher: apiCallFunction };
}
/**
 * Generates the specific prompt and prepares the API call structure for gift suggestions.
 */
function generateGiftSuggestions(person) {
 const { name, classification, notes } = person;

 const safeNotes = String(notes || "general interests").replace(/"/g, "'");
 const classificationText = getClassificationLabel(classification);
 const systemInstruction = `You are an expert gift advisor. Provide three unique and thoughtful gift ideas for a person named ${name} who is a ${classificationText}. The suggestions should be practical, creative, and directly based on their listed interests. Format the response as a bulleted list with a brief description for each idea.`;

 const userQuery = `Generate three gift suggestions for ${name}. Their interests/context are: "${safeNotes}".`;

 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 }] },
 generationConfig: {
 temperature: 0.7, // Slightly lower temperature for practical suggestions
 },
 };
 const options = getApiOptions(apiKey, payload);
```

```
 const apiCallFunction = () => UrlFetchApp.fetch(url, options);
 return { name: name, fetcher: apiCallFunction };
}
/**
 * Creates the standard UrlFetchApp options object.
 */
function getApiOptions(apiKey, payload) {
 return {
 method: 'post',
 contentType: 'application/json',
 headers: { 'x-goog-api-key': apiKey },
 payload: JSON.stringify(payload),
 muteHttpExceptions: true
 };
}
/**
 * Helper function to return the full classification name.
 */
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 => {
 // IMPORTANT: Change the handler function name to the new main entry point
 if (trigger.getHandlerFunction() === 'mainDailyCheck') {
 ScriptApp.deleteTrigger(trigger);
 }
 });
 // Create a new daily trigger (runs between 7 AM and 8 AM)
 ScriptApp.newTrigger('mainDailyCheck') // Use the new main function name
 .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 Agent Setup Complete!", "Your daily two-stage
reminder app trigger has been created and will run every day between 7 AM and 8 AM.");
}
/**
 * Diagnostic function for checking API key status.
 */
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 = getApiOptions(apiKey, payload);
try {
const response = UrlFetchApp.fetch(url, options);
const rawResponseText = response.getContentText();

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 {
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 ---");
}
```