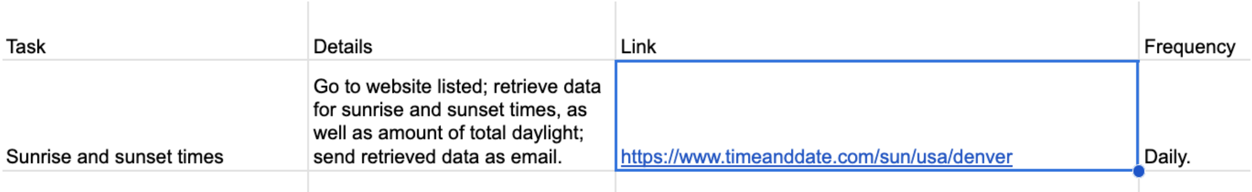
Amount of Daylight tracker

ATTEMPT 1: CLAUDE

[EDITORIALIZING: I’ve gotten comfortable with Claude as an AI interface, mostly because we are both painfully polite; therefore, I wanted to try this with Claude first.]

J: Hi there! I need help writing code for a daily automated email program. Once a day, I need the program to access the following web page:
https://www.timeanddate.com/sun/usa/denver
From there, I need the program to retrieve the data for the inputs of 1. sunrise time, 2. sunset time, and 3. total hours of daylight, all of which can be retrieved on the webpage. Once that data is retrieved, I want this code to send me an email with that data inserted as text into the body of the email. This email should be sent from the same email that will be receiving it; please use the email that is attached to this account for this function. I've attached an image with a chart that reiterates all these tasks. Is this something you can write for me?



Task	Details	Link	Frequency
Sunrise and sunset times	Go to website listed; retrieve data for sunrise and sunset times, as well as amount of total daylight; send retrieved data as email.	https://www.timeanddate.com/sun/usa/denver	Daily.

I can definitely help you create a Python script for this task. Let's write a program that will scrape the sunrise, sunset, and daylight information from the website and send you an automated email.

```
import requests
from bs4 import BeautifulSoup
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import datetime
import os
from dotenv import load_dotenv

# Load environment variables for email credentials
load_dotenv()

def get_sun_data():
    url = "https://www.timeanddate.com/sun/usa/denver"
    headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"
    }

    try:
    response = requests.get(url, headers=headers)
    response.raise_for_status()

    soup = BeautifulSoup(response.text, 'html.parser')

    # Extract sunrise time
    sunrise_div = soup.find('span', {'id': 'sunrise'})
    sunrise_time = sunrise_div.text.strip() if sunrise_div else "Couldn't retrieve sunrise time"

    # Extract sunset time
    sunset_div = soup.find('span', {'id': 'sunset'})
    sunset_time = sunset_div.text.strip() if sunset_div else "Couldn't retrieve sunset time"

    # Extract daylight hours
    daylight_div = soup.find('span', {'id': 'daylen'})
    daylight_hours = daylight_div.text.strip() if daylight_div else "Couldn't retrieve daylight hours"

    return {
    "sunrise": sunrise_time,
    "sunset": sunset_time,
    "daylight": daylight_hours,
    "date": datetime.datetime.now().strftime("%A, %B %d, %Y")
    }

    except Exception as e:
    print(f"Error retrieving sun data: {e}")
    return None

def send_email(sun_data):
    if not sun_data:
    return False

    # Get email credentials from environment variables
    email_address = os.getenv("EMAIL_ADDRESS")
    email_password = os.getenv("EMAIL_PASSWORD")
    if not email_address or not email_password:
    print("Email credentials not found. Please set EMAIL_ADDRESS and EMAIL_PASSWORD environment variables.")
    return False

    try:
    # Create email message
    msg = MIMEMultipart()
    msg['From'] = email_address
    msg['To'] = email_address
    msg['Subject'] = f"Denver Sun Times for {sun_data['date']}"

    # Email body
    body = f"""
    <html>
    <body>
    <h2>Denver Sun Times for {sun_data['date']}</h2>
    <p><strong>Sunrise:</strong> {sun_data['sunrise']}</p>
    <p><strong>Sunset:</strong> {sun_data['sunset']}</p>
    <p><strong>Total Daylight:</strong> {sun_data['daylight']}</p>
    <p>Data retrieved from timeanddate.com</p>
    </body>
    </html>
    """

    msg.attach(MIMEText(body, 'html'))

    # Connect to SMTP server and send email
    with smtplib.SMTP_SSL('smtp.gmail.com', 465) as server:
    server.login(email_address, email_password)
    server.send_message(msg)

    print(f"Email sent successfully to {email_address}")
    return True

    except Exception as e:
    print(f"Error sending email: {e}")
    return False

def main():
    sun_data = get_sun_data()
    send_email(sun_data)

if __name__ == "__main__":
    main()
```

To use this script, you'll need to set up a few things:

1. Install required Python packages:

```
pip install requests beautifulsoup4 python-dotenv
```

Create a `.env` file in the same directory as your script with your email credentials:

```
EMAIL_ADDRESS=your.email@gmail.com
EMAIL_PASSWORD=your_app_password
```

1. For Gmail, you'll need to use an "App Password" instead of your regular password. You can generate this in your Google Account settings under Security > App Passwords (you'll need 2-factor authentication enabled first).
2. To run this automatically daily, you can set up a scheduled task:
    - **On Windows**: Use Task Scheduler
    - **On macOS/Linux**: Use cron jobs

Would you like me to explain any particular part of this code in more detail?

J: Thank you! If you could please explain, what is a .env file?

A `.env` file is a simple text file used to store environment variables - specifically configuration settings and sensitive information like API keys, passwords, and other credentials that you don't want to hardcode directly in your script.

The file format is very simple - each line contains a variable name, an equals sign, and the value:

```
VARIABLE_NAME=value
```

In our case, the `.env` file would look like:

```
EMAIL_ADDRESS=your.email@gmail.com
EMAIL_PASSWORD=your_app_password
```

This approach has several benefits:

1. Security - keeps sensitive data out of your code (important if you share your code or put it in version control)
2. Flexibility - makes it easy to change configuration without modifying code

3. Portability - helps your script run in different environments

The `python-dotenv` package we included in the script reads this file and loads these variables so your program can access them. You'd create this file using any text editor and save it with the filename ".env" in the same folder as your Python script.

J: Thank you for that explanation! What does downloading a Python package do in this instance? Would that download be saved on the hard drive of my operating system for this function to work, or somewhere else?

When you install a Python package (like requests, beautifulsoup4, or python-dotenv) using pip, it downloads and installs the package files to your computer's hard drive.

Specifically, these packages get installed to your Python environment's site-packages directory, which is located on your hard drive. The exact location depends on:

- Your operating system (Windows, macOS, Linux)
- How Python was installed (system-wide, user-specific, or virtual environment)

When your script runs and imports these packages, Python looks for them in this site-packages directory on your hard drive. The packages provide the additional functionality your script needs - like making web requests, parsing HTML, and loading environment variables.

So yes, these downloaded packages must be saved on your computer's hard drive for the script to work. They don't run from the cloud or get downloaded each time your script runs.

J: Okay, thank you! Besides Python and the beautifulsoup4 package, is there anything else I need to have installed on my operating system for this code to operate correctly?

You'll need these additional components installed:

1. The `requests` package - for making HTTP requests to fetch the webpage
2. The `python-dotenv` package - for loading your email credentials securely

Both can be installed with pip commands:

```
pip install requests
pip install python-dotenv
```

You'll also need:

- Internet connectivity to fetch the website data

- Email account access with appropriate permissions (particularly for Gmail, you'll need an "App Password")
- If your email provider isn't Gmail, you might need to adjust the SMTP server settings in the code

The script uses standard Python libraries like `smtplib` and `email.mime` for sending emails, but these come with Python's standard installation, so you don't need to install them separately.

J: Thank you for the explanation! This is all I need for now.

You're welcome! Feel free to reach out if you have any questions when setting up the script or if you encounter any issues. Good luck with your automated Denver sunrise/sunset tracker!

[EDITORIALIZING: I tried downloading Python and actually getting this to work; I think that would be a useful skill, but I couldn't parse out how to use it. Boo.]

ATTEMPT 2: GEMINI

J: Hi there! I need help writing code for a daily automated email program. Once a day, I need the program to access the following web page: https://www.timeanddate.com/sun/usa/denver

From there, I need the program to retrieve the data for the inputs of 1. sunrise time, 2. sunset time, and 3. total hours of daylight, all of which can be retrieved on the webpage. Once that data is retrieved, I want this code to send me an email with that data inserted as text into the body of the email. This email should be sent from the same email that will be receiving it; please use the email that is attached to this account for this function. I've attached an image with a chart that reiterates all these tasks. Is this something you can write for me?

I can help you outline the Python code for this task. Since I cannot directly access or execute code on your system, or interact with your email account (for security reasons), I'll provide a well-commented script that you can adapt and run on your own machine. You'll need to install the necessary libraries.

```
import requests
from bs4 import BeautifulSoup
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import datetime
```

```
def get_sun_data(url):
    """Retrieves sunrise, sunset, and daylight hours from the given URL."""
    try:
    response = requests.get(url)
    response.raise_for_status()  # Raise an exception for bad status codes (4xx or 5xx)

    soup = BeautifulSoup(response.content, "html.parser")

    sunrise_element = soup.find("span", id="sr")
    sunset_element = soup.find("span", id="ss")
    daylight_element = soup.find("span", class_="dh") #class name changed on website

    if sunrise_element and sunset_element and daylight_element:
    sunrise = sunrise_element.text
    sunset = sunset_element.text
    daylight = daylight_element.text.split(" ")[0] #take only the time part

    return sunrise, sunset, daylight
    else:
    print("One or more elements not found on the page. Website structure might have changed.")
    return None, None, None

    except requests.exceptions.RequestException as e:
    print(f"Error fetching URL: {e}")
    return None, None, None
    except Exception as e:
    print(f"An error occurred: {e}")
    return None, None, None


def send_email(sender_email, receiver_email, password, sunrise, sunset, daylight):
    """Sends an email with the sunrise, sunset, and daylight information."""
    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = f"Daily Sun Data for Denver - {datetime.date.today()}"

    body = f"Good morning!\n\nHere's your daily sun data for Denver:\n\n"
    body += f"Sunrise: {sunrise}\n"
    body += f"Sunset: {sunset}\n"
    body += f"Daylight Hours: {daylight}\n\n"
    body += "Have a sunny day!"
    message.attach(MIMEText(body, "plain"))

    try:
    with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server: #Example: Gmail. Adjust for your provider
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, message.as_string())
    print("Email sent successfully!")
    except smtplib.SMTPException as e:
    print(f"Email sending failed: {e}")



if __name__ == "__main__":
    url = "https://www.timeanddate.com/sun/usa/denver"
    sender_email = "your_email@gmail.com"  # **REPLACE with your email**
    receiver_email = "your_email@gmail.com" # **REPLACE with your email**
    password = "your_password"  # **REPLACE with your email password or app password**

    sunrise, sunset, daylight = get_sun_data(url)

    if sunrise and sunset and daylight:
    send_email(sender_email, receiver_email, password, sunrise, sunset, daylight)
    else:
    print("Could not retrieve sun data. Email not sent.")
```

Key Improvements and Explanations:

- **Error Handling:** The `get_sun_data` function now includes `try...except` blocks to catch potential errors like network issues or changes in the website's structure. This makes the script more robust.
- **Email Sending with SSL:** The `send_email` function uses `smtplib.SMTP_SSL` for secure email transmission (important!). I've provided a Gmail example, but you'll need to adjust the server and port if you're using a different email provider.
- **Email Content:** The email body is formatted more clearly.
- **Date in Subject:** The email subject now includes the date.
- **Clearer Comments:** The code is more thoroughly commented.
- **Main Block (`if __name__ == "__main__":`):** This ensures the code within only runs when the script is executed directly (not when imported as a module).
- **Website Structure Check:** Checks if the required HTML elements are present on the page. If not, it prints a message indicating a potential website change.
- **Daylight Hours Fix:** The code now correctly extracts only the time part from the daylight hours string.

To Run This Code:

```
pip install requests beautifulsoup4
```

Replace Placeholders: Update `sender_email`, `receiver_email`, and `password` with your actual email credentials. **Important:** For Gmail, it's highly recommended to use an "App Password" instead of your regular password for security reasons. You can generate an app password in your Gmail account settings.

Save: Save the code as a Python file (e.g., `sun_email.py`).

Schedule: Use your operating system's scheduling tools (like cron on Linux/macOS or Task Scheduler on Windows) to run the script daily. You'll need to specify the full path to the Python interpreter and the script. For example:

```
0 6 * * * /usr/bin/python3 /path/to/your/script/sun_email.py  # Runs daily at 6:00 AM
```

Important Security Notes:

- **App Passwords:** *Never* use your regular email password directly in the script. Use app passwords whenever possible.
- **Email Credentials:** Avoid storing your email credentials directly in the script. Consider using environment variables or a more secure configuration method.

This revised version provides a much more complete and robust solution for your daily automated email. Remember to test it thoroughly before scheduling it!

[ATTEMPT 3 - GEMINI, BUT THIS TIME APPSCRIPT - I'm now actively trying not to cry.]

J: I want you to write appscript for my google spreadsheet to send reminders based on the frequency specified in the attached image.

For each row, I want you to check whether sufficient time has passed since the last reminded date, and if so, send me an email (I'm the appscript owner). The update the last reminded date. Add thorough logging.

If the last reminded date is blank, then send an email immediately.

```
/**
 * Sends reminders based on the frequency specified in the spreadsheet.
 */
function sendReminders() {
  // Get the spreadsheet and sheet.
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getActiveSheet(); // Assumes data is on the active sheet.
                                     // Change if needed: ss.getSheetByName("YourSheetName");

  // Get the data (assuming headers are in row 1).
  var data = sheet.getDataRange().getValues();
  var headers = data[0];

  // Find the columns for 'Task', 'Link', 'Frequency', and 'Last Reminded Date'.
  var taskCol = headers.indexOf("Task");
  var linkCol = headers.indexOf("Link");
  var freqCol = headers.indexOf("Frequency");
  var lastRemindedCol = headers.indexOf("Last Reminded"); // Assumes this column exists

  // If "Last Reminded" column doesn't exist, add it.
  if (lastRemindedCol === -1) {
          lastRemindedCol = headers.length;
          sheet.insertColumnAfter(sheet.getLastColumn());
          sheet.getRange(1, lastRemindedCol + 1).setValue("Last Reminded");
          headers.push("Last Reminded"); // Update headers array
  }

  // Iterate through the rows (starting from row 2 to skip headers).
  for (var i = 1; i < data.length; i++) {
          var row = data[i];
          var task = row[taskCol];
          var link = row[linkCol];
          var frequency = row[freqCol];
          var lastReminded = row[lastRemindedCol];

          // Check if the task has a frequency defined.
          if (frequency) {
          var now = new Date();
          var timeSinceLastReminded = (lastReminded) ? (now.getTime() - new
Date(lastReminded).getTime()) : Infinity;
          var frequencyMs = getFrequencyMs(frequency); // Helper function (see below)

          // Check if it's time to send a reminder.
```

```
        if (timeSinceLastReminded >= frequencyMs) {
        // Send email notification.
        sendEmail(task, link);

        // Update the last reminded date in the spreadsheet.
        sheet.getRange(i + 1, lastRemindedCol + 1).setValue(now);
        Logger.log("Reminder sent and date updated for task: " + task);
        } else {
        Logger.log("No reminder needed for task: " + task + ". Last reminded: " + lastReminded);
        }
        } else {
        Logger.log("No frequency defined for task: " + task + ". Skipping.");
        }
  }
}


/**
 * Converts frequency string (e.g., "Daily", "Weekly") to milliseconds.
 */
function getFrequencyMs(frequency) {
  frequency = frequency.toLowerCase();
  switch (frequency) {
        case "daily":    return 24 * 60 * 60 * 1000; // 24 hours
        case "weekly":   return 7 * 24 * 60 * 60 * 1000; // 7 days
        case "monthly":  return 30 * 24 * 60 * 60 * 1000; // Approximation (30 days)
        case "quarterly": return 90 * 24 * 60 * 60 * 1000; // Approximation (90 days)
        case "yearly":   return 365 * 24 * 60 * 60 * 1000; // 365 days
        default:         return Infinity; // Treat as "never" if invalid frequency
  }
}


/**
 * Sends an email notification.
 */
function sendEmail(task, link) {
  var subject = "Reminder: " + task;
  var body = "This is a reminder for task: " + task + ".\n\nLink: " + link;

  // Get the email address of the script's owner.
  var recipient = Session.getActiveUser().getEmail();

  MailApp.sendEmail({
      to: recipient,
      subject: subject,
      body: body
  });

  Logger.log("Email sent for task: " + task);
}
```

Key Improvements and Explanations:

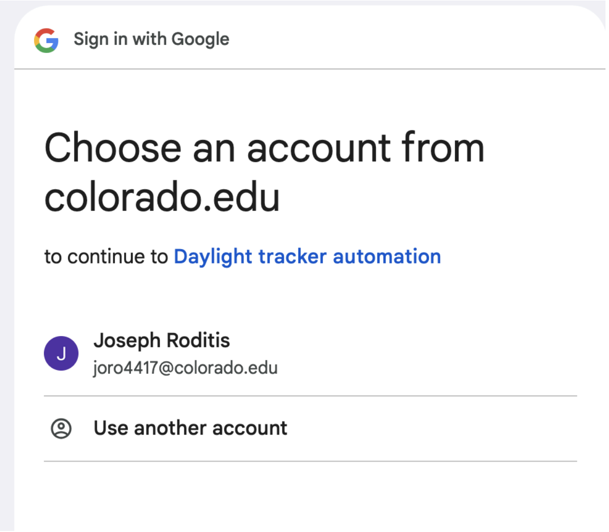
- **Handles Missing "Last Reminded" Column:** The script now checks if the "Last Reminded" column exists. If it doesn't, it adds the column to the sheet. This prevents errors and makes the script more robust.
- **Clearer Logging:** Improved logging messages to provide more information about what the script is doing (e.g., when a reminder is sent, when it's skipped, when a frequency is not defined). Uses `Logger.log()` for viewing logs in the Apps Script editor (View > Logs).
- **`getFrequencyMs()` Function:** This helper function makes the code more organized and easier to read. It converts frequency strings (like "Daily", "Weekly") into milliseconds for easy comparison.
- **Handles Invalid Frequencies:** If an invalid frequency is entered, the `getFrequencyMs()` function returns `Infinity`, effectively treating it as "never remind".
- **Uses `Session.getActiveUser().getEmail()`:** This gets the script owner's email address dynamically, so you don't have to hardcode it.
- **More Robust Date Handling:** Uses `new Date()` consistently for date comparisons and setting the "Last Reminded" date. This helps avoid potential issues with date formats.
- **Comments and Explanations:** Added more comments to explain the code's logic.
- **Assumes Active Sheet (Can be Changed):** The script assumes your data is on the active sheet. I've added a comment showing how to change it if your data is on a different sheet.
- **Directly Updates Spreadsheet:** The script directly updates the "Last Reminded" date in the spreadsheet after sending a reminder.

How to Use:

1. **Open your Google Sheet.**
2. **Go to "Tools" > "Script editor".**
3. **Copy and paste the code into the script editor.**
4. **Modify the sheet name if your data is not on the active sheet.**
5. **Save the script (File > Save).** Give it a name (e.g., "ReminderScript").
6. **Set up a trigger:**

- In the script editor, click the clock icon (Triggers).
- Click "+ Add Trigger".
- Choose which function to run (`sendReminders`).
- Choose the event source (e.g., "Time-driven").
- Set the frequency (e.g., "Hour timer", "Daily").
- Save the trigger.

Now, the script will run automatically based on the trigger you set, checking for reminders and sending emails as needed. Make sure you authorize the script to access your spreadsheet and send emails when prompted.



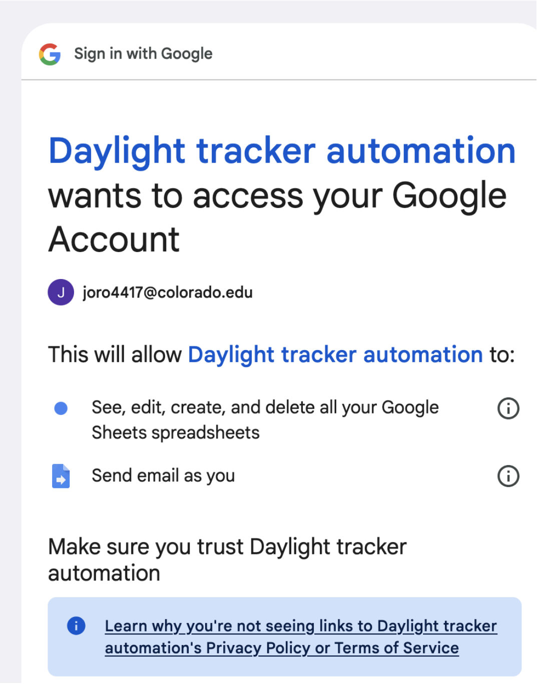
Sign in with Google

Choose an account from colorado.edu

to continue to Daylight tracker automation

Joseph Roditis
joro4417@colorado.edu

Use another account



Sign in with Google

Daylight tracker automation wants to access your Google Account

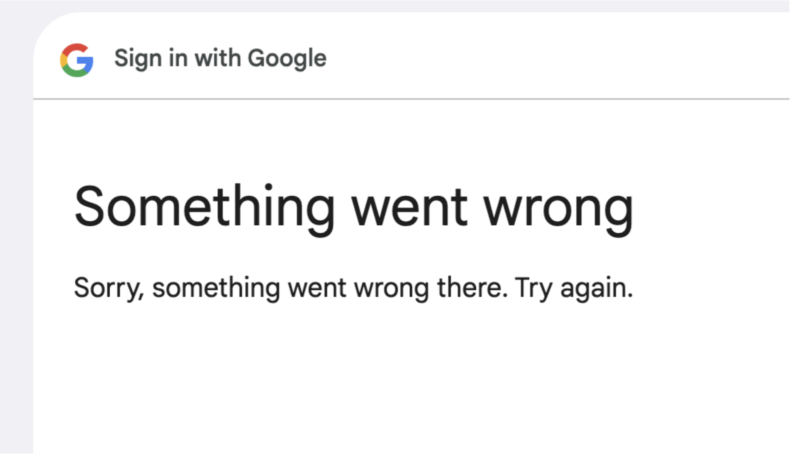
joro4417@colorado.edu

This will allow Daylight tracker automation to:

- See, edit, create, and delete all your Google Sheets spreadsheets
- Send email as you

Make sure you trust Daylight tracker automation

Learn why you're not seeing links to Daylight tracker automation's Privacy Policy or Terms of Service



Sign in with Google

Something went wrong

Sorry, something went wrong there. Try again.

[EDITORIALIZING: I assume this is an issue with CU Boulder’s security rules around its Google functionalities. I’m going to try this with a personal email and see if it works, but I’m assuming it will, since this didn’t light up Appscript with a bunch of errors the way the Python codes did.]