Open in app ↗

Medium Search

Member-only story

# 5 Python Automation Projects You Can Build This Weekend

From Beginner-Friendly to Advanced

Abdur Rahman · Follow
Published in Python in Plain English
6 min read · Nov 14, 2024

Listen Share More

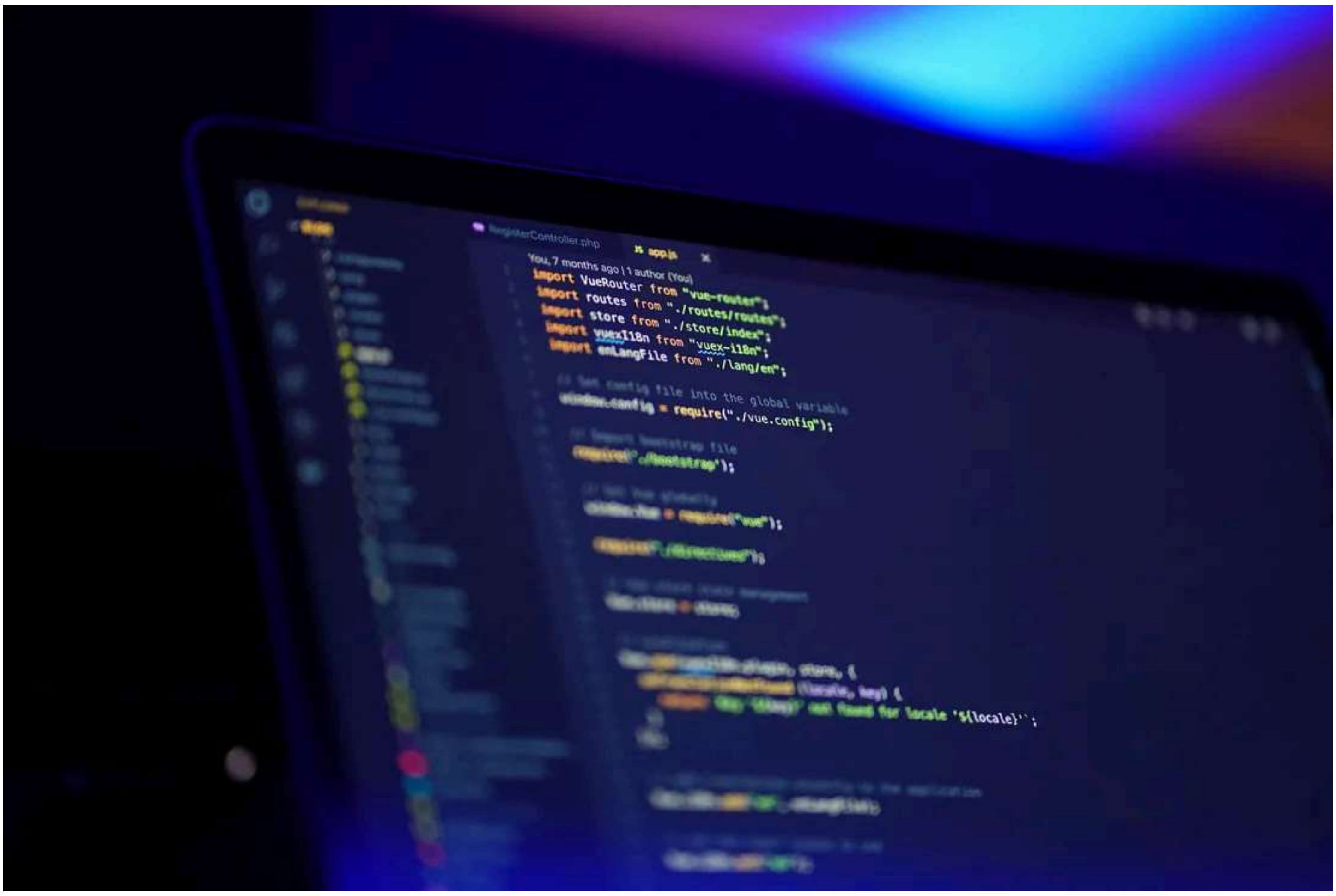

Photo by Mohammad Rahmani on Unsplash

If you are a Python developer, you probably are no stranger to the thrill of building project solutions for real-world problems.

However, if you are like me, you would have spent hours brainstorming the perfect project idea and gotten stuck on how to start.

Well, if eager to learn and wanting to kickstart your automation game this weekend, here are 5 Python automation projects each ramping up in difficulty. These will not only improve your Python skills, but give you something really useful at the end of the weekend.

Just a heads up, all of these projects are based on one simple but effective guiding principle: **try to solve a problem, not to use a tool**. That way you will learn more, and you will build something useful.

## 1) Email Auto Cleaner (Beginner)

Ever feel your inbox blow up with unread newsletters, promotion emails, and spam? What if I told you it was possible to automate all of this with Python?

Instead of doing all that by hand, why not let Python do the job? Here's a simple e-mail cleaner that will weed out promo emails from Gmail.

**What You'll Need:**

- Google Gmail API (To access your inbox)
- Regular expressions (To filter out junk)

```
import re
import imaplib
import email
from email.header import decode_header

# Connect to the Gmail server
mail = imaplib.IMAP4_SSL("imap.gmail.com")
mail.login("your_email@gmail.com", "your_password")
mail.select("inbox")

# Search for all emails
status, messages = mail.search(None, "ALL")
email_ids = messages[0].split()
```

```
    # Loop through all emails and filter out unwanted ones
    for email_id in email_ids:
        status, msg_data = mail.fetch(email_id, "(RFC822)")
        for response_part in msg_data:
            if isinstance(response_part, tuple):
                msg = email.message_from_bytes(response_part[1])
                subject, encoding = decode_header(msg["Subject"])[0]

                # Check if the subject contains a promotional keyword
                if re.search(r"(sale|promotion|offer|newsletter)", subject, re.IGNC
                    mail.store(email_id, '+FLAGS', '\\Deleted')  # Mark as deleted
                    print(f"Deleted: {subject}")

    # Expunge deleted messages
    mail.expunge()
    mail.logout()
```

This program will look through your inbox for keywords in the subject lines of emails, such as “sale” or “promotion.” Then it automatically deletes those kinds of emails. You can expand this project by adding the feature to auto-unsubscribe from these lists!

**2) Social Media Post Scheduler (Beginner)**

Any content creator, marketer, or any businessman servicing on-line knows how important Social Media is. Therein lays the problem: manually scheduling posts on social media is pretty much a time-sink and productivity killer. The following is an automation project that will enable you to schedule posts to Twitter at specific times.

### What You’ll Need:

- Twitter Developer API (Access to your Twitter account)
- `schedule` Python library (For scheduling posts)

```
    import tweepy
    import schedule
    import time

    # Twitter API authentication
    auth = tweepy.OAuth1UserHandler(
        consumer_key="your_consumer_key",
        consumer_secret="your_consumer_secret",
        access_token="your_access_token",
        access_token_secret="your_access_token_secret"
    )
    api = tweepy.API(auth)

    # Function to post a tweet
    def post_tweet():
        tweet = "Automated tweet using Python! #PythonRocks"
        api.update_status(tweet)
        print("Tweet posted!")

    # Schedule the tweet to be posted at 9 AM every day
    schedule.every().day.at("09:00").do(post_tweet)

    # Keep the script running
    while True:
        schedule.run_pending()
        time.sleep(60)
```

This simple scheduler will automatically post a tweet every day at 9:00 AM. Want to schedule multiple posts? Easy, just modify the scheduling logic. That's it, now this is going to be great in automating your social media presence.

**Quick Pause**

If you're eager to learn Python through hands-on, project-based examples, ***PYTHON CRASH COURSE by Eric Matthes*** is the world's bestselling programming book, with over **1,500,000** copies sold to date!

**3) Automatic Data Scraper (Intermediate)**

Need to gather any web data that you have piled up? Probably find some price data on an e-commerce site or some stock market data? With Python, BeautifulSoup, and requests libraries, it's really easy to scrape and automate this process of data extraction.

Here, a scraper will be implemented that will scrape product names along with their prices from an e-commerce website.

**What You'll Need:**

- `requests` library (For making HTTP requests)
- `BeautifulSoup` from `bs4` (For parsing HTML)

```
import requests
from bs4 import BeautifulSoup

# URL of the website to scrape
url = "https://example.com/products"

# Send a GET request to the website
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

# Find product names and prices
products = soup.find_all("div", class_="product")

for product in products:
    name = product.find("h2").text
    price = product.find("span", class_="price").text
    print(f"Product: {name}, Price: {price}")
```

This script fetches the product name and its price from the website. It allows one to automate the gathering of data for tracking prices or monitoring market trends.

**4) File Organizer (Intermediate)**

If you have lots of projects, then you know that your file system can get out of hand pretty quick. Here's an automation project to put all those files in order. This scripting project was made in Python, sorting the files in a directory according to their file type (PDF, images, docs, etc.) and moved into appropriate folders:.

**What You'll Need:**

- `os` library (For interacting with the operating system)
- `shutil` library (For moving files)

```
import os
import shutil

# Path to the directory you want to organize
source_dir = "your_directory_path"

# Define file type categories
categories = {
    "Images": [".jpg", ".png", ".gif"],
    "PDFs": [".pdf"],
    "Documents": [".docx", ".txt"],
        "Spreadsheets": [".xls", ".csv"]
    }

    # Loop through all files in the source directory
    for filename in os.listdir(source_dir):
        file_path = os.path.join(source_dir, filename)

        if os.path.isfile(file_path):
            file_extension = os.path.splitext(filename)[1].lower()

            # Check the file extension and move to the appropriate folder
            for category, extensions in categories.items():
                if file_extension in extensions:
                    category_path = os.path.join(source_dir, category)
                    if not os.path.exists(category_path):
                        os.makedirs(category_path)
                    shutil.move(file_path, os.path.join(category_path, filename))
                    print(f"Moved {filename} to {category}")
                    break
```

The script sorts your files into folders based on type, automatically. It's simple, but works for a clean workspace.

**5. Personal Movie Recommendation System (Advanced)**

Okay, now it gets a bit interesting. If you're into machine learning, here is a project using your past movie ratings to recommend movies you might like. A personalized recommendation system, it takes its core Ranking principles from the most powerful way of making suggestions based on user preferences: collaborative filtering.

**What You'll Need:**

- `pandas` (For data handling)
- `sklearn` (For collaborative filtering)
- `numpy` (For numerical operations)

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Example user-movie rating data (you can load your own dataset)
data = {
        'user': ['Alice', 'Bob', 'Charlie', 'David'],
        'Movie A': [5, 4, 1, 0],
        'Movie B': [3, 5, 2, 0],
        'Movie C': [4, 3, 5, 0],
        'Movie D': [0, 1, 3, 5]
    }
    df = pd.DataFrame(data)

    # Compute the cosine similarity between users
    user_ratings = df.drop('user', axis=1).values
    cosine_sim = cosine_similarity(user_ratings)

    # Create a function to recommend movies based on similarity
    def recommend_movies(user_id):
        similar_users = list(enumerate(cosine_sim[user_id]))
        similar_users = sorted(similar_users, key=lambda x: x[1], reverse=True)

        recommended_movies = []
        for i, _ in similar_users[1:]:
            recommended_movies.append(df.columns[i+1])

        return recommended_movies

    # Example: Recommend movies for user 0 (Alice)
    recommended = recommend_movies(0)
    print(f"Recommended movies for Alice: {recommended}")
```

This system works out similarities between users of their ratings of movies and recommends movies based on these similarities. If you have ever used Netflix or Spotify, you have interacted with a similar recommendation engine.

## Final Thoughts

All these projects push your skills in some way. Pick one that excites you, that way learning won't feel like a chore.

> When you're motivated, learning doesn't feel like work — it feels like progress.

So, which one are you going to start with this weekend?