Institute for Advanced Studies
in Basic Sciences
GavaZang, Zanjan, Iran

Contact Manager

Prof. Mahmood Shirazi

Report for Project <2>

Faeze Ahmadi

October 2025

تمرین اول: نسخه فعلی برنامه را که با فایل txt کار می‌کند، طوری تغییر دهید که با فایل CSV کار کند.

Github Link: https://github.com/Faeze-Ahmadi/AP_A2_Faeze_Ahmadi/blob/main/1_contact_manager_for_CSV_file.py

Document

من توی این پروژه یه Contact Manager خیلی ساده با پایتون پیاده‌سازی کردم که اطلاعات مخاطب‌ها مثل اسم، شماره تلفن و ایمیل رو داخل یک فایل CSV ذخیره می‌کنه، بعد می‌تونه اون‌ها رو نمایش بده یا بینشون سرچ کنه. نسخه اولیه این برنامه با یک فایل متنی ساده txt کار می‌کرد و با جدا کردن رشته‌ها بر اساس ویرگول، داده‌ها رو مدیریت می‌کردیم. اما توی این نسخه، برنامه رو طوری تغییر دادم که به جای فایل txt، از فایل contacts.csv استفاده کنه و با ماژول استاندارد csv پایتون، خواندن و نوشتن رکوردها رو انجام بده. استفاده از CSV باعث می شه ساختار داده‌ها تمیزتر، استانداردتر و قابل استفاده توسط برنامه‌های دیگه مثل اکسل و... هم باشه.

در ابتدای کد، ماژول csv رو ایمپورت کردم، چون قرار بود به جای این که خودم با split(",") و این چیزها رکوردها رو بشکنم، از امکانات آماده خود پایتون برای کار با فایل‌های CSV استفاده کنم. فایل اصلی که همه مخاطب‌ها توش ذخیره می‌شن یک متغیر به نام filename داره که مقدارش "contacts.csv" هست؛ این

طوری اگر بعدا بخوام اسم فایل رو عوض کنم، فقط همین یک جا رو تغییر میدم و بقیه کد با همون متغیر کار می‌کنن.

تابع اولی که توی این برنامه نوشتم add_contact هست. این تابع مسئول گرفتن اطلاعات مخاطب جدید و ذخیره کردن اون توی فایل CSV هست. داخل این تابع، اول با input از کاربر اسم، شماره تلفن و ایمیل رو می‌گیرم. تابع input همیشه خروجی رو به صورت رشته (str) برمی‌گردونه، حتی اگر کاربر فقط عدد تایپ کنه، اما من نمی‌خواستم اسم و ایمیل، فقط عدد خالی باشن. برای همین یک بخش اعتبارسنجی ساده اضافه کردم. به این صورت که اگر name.isdigit() درست باشه یعنی اسم فقط از رقم تشکیل شده، من با raise ValueError("Name must not be only digits.") یک خطای منطقی ایجاد می‌کنم، چون اسمی مثل "12345" از نظر برنامه ما قابل قبول نیست. همین کار رو برای ایمیل هم انجام دادم: اگر email.isdigit() بود، یعنی ایمیل هم فقط عدده، که باز غیرمنطقیه. برعکس برای شماره تلفن، برام مهم بود که فقط عدد باشه، پس اگر not phone.isdigit() بود، یعنی کاربر حروف یا چیزهای دیگه وارد کرده، من هم با یک ValueError بهش گیر میدم. کل این بخش رو داخل یک بلاک try گذاشتم تا اگر هرکدوم از این ValueErrorها رخ بده، برنامه کرش نکنه و بره توی except ValueError as e و فقط یک پیام دوستانه مثل Input error: ... پرینت کنه و ادامه بده. به این ترتیب، اگر کاربر اطلاعات اشتباه وارد کرد، برنامه خراب نمیشه، فقط بهش می‌گه چه مشکلی وجود داره.

اگر ورودی‌ها معتبر بود، نوبت به ذخیره کردن توی فایل CSV می‌رسه. این کار رو با دستور with open(filename, "r", newline="", encoding="utf-8") as file: انجام میدیم. این خط چند تا نکته داره: اول این‌که از with استفاده کردم تا هر وقت کارم با فایل تمام شد، خود پایتون خودکار فایل رو ببنده و نیازی به file.close() نداشته باشیم، خود استاد هم گفتن که این بهترین روش کار با فایل‌هاست. دوم، حالت باز کردن فایل "a" هست، یعنی append؛ یعنی اگر فایل وجود داشته باشه، داده جدید رو به انتهای فایل اضافه

می‌کنه و اگر وجود نداشته باشه، خود پایتون فایل رو میسازه. به این ترتیب هر مخاطب جدید به صورت یک سطر جدید به انتهای فایل CSV اضافه می‌شه. پارامتر newline="" رو به این خاطر گذاشتم که وقتی با ماژول csv کار می‌کنیم، اگر این پارامتر رو نذاریم، مخصوصا روی ویندوز ممکنه بین هر ردیف CSV یک خط خالی اضافه بشه. با newline="" می‌گیم مدیریت کاراکتر پایان خط رو خود ماژول csv انجام بده. پارامتر encoding="utf-8" هم برای اینه که اگر نام‌ها یا ایمیل‌ها فارسی یا شامل کاراکترهای خاص باشن، به هم نریزن و به صورت یونیکد استاندارد ذخیره بشن.

داخل بلاک with، یه نویسنده CSV می‌سازی writer =(file)csv.writer. این writer یک آبجکت مخصوص است که می تونه لیست داده‌ها رو به صورت یک ردیف CSV در فایل بنویسه. با writer.writerow([name, phone, email]) یک لیست سه تایی شامل نام، شماره و ایمیل رو به عنوان یک ردیف جدید در فایل ذخیره می‌کنم. دیگه لازم نیست خودم ویرگول بذارم و \n اضافه کنم؛ csv.writer این کار رو برای من انجام می‌ده. اگر همه چیز خوب پیش رفته باشه، یک پیام موفقیت به کاربر چاپ می‌کنم که مخاطب با موفقیت اضافه شد. اگر در مراحل اعتبارسنجی خطایی افتاده باشه، بلاک except ValueError فعال می‌شه و به جای خروج از برنامه، فقط یک پیام خطای ورودی چاپ می‌کنه.

تابع بعدی view_contacts است که وظیفه نمایش همه مخاطب‌ها رو برعهده داره. این تابع هم یک filename می‌گیره و سعی می‌کنه فایل CSV رو در حالت خوندن باز کنه: with open(filename, "r", newline="", encoding="utf-8") as file. اینجا هم از همون newline و encoding استفاده کردم برای سازگاری با csv. بعد reader = csv.reader(file) رو می‌سازم. این reader مثل یه حلقه قابل پیمایش هست که هر بار یک ردیف از CSV را به صورت لیست به ما می‌ده، مثلا چیزی شبیه ["Ali", "0912...", "ali@gmail.com"]. برای این که بتونیم بفهمیم آیا فایل مخاطب داره یا نه، کار راحت‌تر اینه که کل reader

را تبدیل به لیست کنمی: contacts = list(reader). اگر این لیست خالی باشه، یعنی هنوز هیچ مخاطبی ثبت نشده و یه پیام «No contacts found» چاپ می‌کنیم و با return از تابع خارج بیرون میایم.

اگر لیست خالی نبود، یعنی مخاطب‌ها وجود دارن. اول یه تیتر ساده چاپ می‌کنیم که مشخص بشه لیست شروع شده، بعد با for name, phone, email in contacts: روی هر ردیف حلقه می‌زنیم. چون می‌دونیم هر ردیف سه تا ستون داره، خیلی راحت خروجی هر سطر رو به سه متغیر name, phone, email نسبت می‌دیم و با یه print قشنگ به کاربر نشان می‌دیم: Name: ..., Phone: ..., Email: .... این جا دیگه نیازی به split(",") و strip() نداریم، چون csv.reader خودش داده‌ها رو تمیز و جداشده تحویل ما میده. کل این قسمت رو هم داخل یک بلاک try گذاشتم و اگر فایل اصلا وجود نداشت، یعنی کاربر هنوز هیچ مخاطبی اضافه نکرده، یک FileNotFoundError رخ می‌ده و می‌پره داخل except FileNotFoundError و توی اونجا با این پیام میگیم که «فایل مخاطب هنوز ساخته نشده، اول یک مخاطب اضافه کن».

تابع search_contact هم شبیه view_contacts است، فقط به جای این که همه مخاطبین رو پرینت بگیره، دنبال یه اسم خاص می‌گرده. ابتدا از کاربر می‌پرسیم دنبال چه اسمی می‌گرده: name_to_search = input("Enter name to search: ").lower(). این lower() رو می‌زنیم تا حروف رو کوچک کنیم و مقایسه نسبت به بزرگی و کوچکی حروف حساس نباشه؛ یعنی اگر در فایل «Ali» باشه و کاربر ali بزنه، باز هم پیدا بشه. یه پرچم found = False هم تعریف می‌کنیم تا بفهمیم در نهایت چیزی پیدا شده یا نه. بعد مثل قبل فایل رو در حالت "r" باز می‌کنم، یک csv.reader می‌سازم و روی هر ردیف حلقه می‌زنم: for name, phone, email in reader:. برای هر ردیف، اگر name.lower() == name_to_search بود، یعنی این همون مخاطب مورد نظر ماست، پس اطلاعاتش رو چاپ می‌کنیم، found = True رو میذاریم و با break از حلقه خارج می‌شیم تا ادامه فایل بیخود نگرده. بعد از حلقه، اگر هنوز found برابر False بود، یعنی هیچ مخاطبی با اون اسم پیدا نشده، پس یه پیام «Contact not found» چاپ می‌کنیم. این تابع هم در یک بلاک

try/except FileNotFoundError قرار گرفته تا اگر فایل CSV هنوز ایجاد نشده بود، برنامه به جای کرش کردن، فقط به کاربر بگه که هنوز فایل مخاطب ساخته نشده.

حالا طبق خواسته تمرین دو تا قابلیت update و delete مخاطب اضافه شده. توی قابلیت Update کاربر نام مخاطبی رو که می‌خواد ویرایش کنه وارد می‌کنه، برنامه اولین رکورد مطابق با اون نام رو توی فایل CSV پیدا می‌کنه، اطلاعات فعلی (نام، شماره تلفن، ایمیل) رو نمایش می‌ده و بعدش به کاربر اجازه می‌ده که برای هر فیلد مقدار جدید وارد کنه یا با زدن Enter اون فیلد رو بدون تغییر نگه داره؛ پس از تایید، فایل دوباره با داده‌های به‌روز شده ذخیره می‌شه. حالا در قابلیت Delete هم کاربر نام مخاطب رو وارد می‌کنه و برنامه اولین رکوردی رو که نام اون با ورودی برابر باشه از فایل حذف کرده و بقیه رکوردها رو بدون تغییر دوباره توی فایل می‌نویسه؛ توی هر دو حالت، اگه مخاطبی با اون نام پیدا نشه، پیام مناسب به کاربر نمایش داده می‌شه و هیچ تغییری روی فایل به وجود نمیاد.

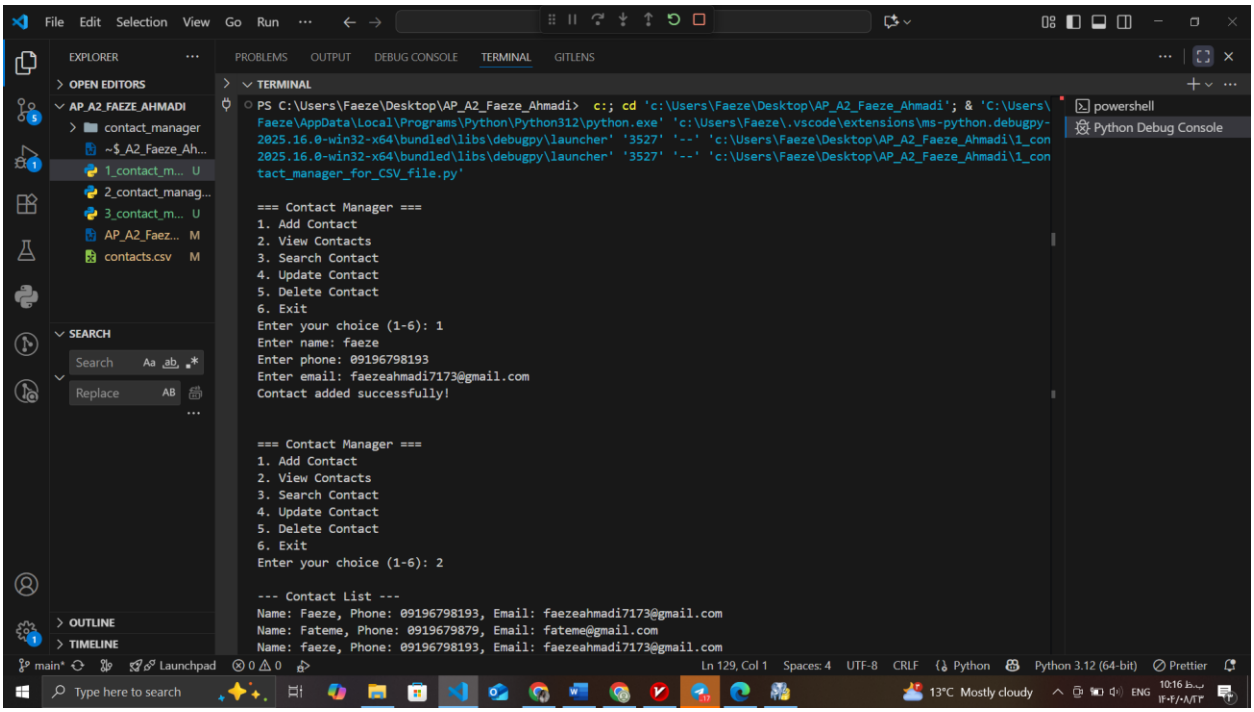
در نهایت، یه جورایی مثل تمرین 5 سری اول یه منو سیستم میسازیم. طوری که همه این تابع‌ها از داخل یک تابع main کنترل می‌شن که نقش منوی اصلی برنامه رو ایفا میکنن. در main، اول filename = ""contacts.csv رو مشخص می‌کنیم تا همه تابع‌ها بدونن با چه فایلی باید کار کنن. بعد یه حلقه بی‌نهایت while True داریم که هر بار منوی برنامه رو چاپ می‌کنه: گزینه‌ها 1 برای اضافه کردن مخاطب، 2 برای نمایش همه مخاطب‌ها، 3 برای جستجو و 4 برای خروج. از کاربر می‌پرسیم انتخابش کدومه و بسته به این که ورودی کدوم باشه، تابع مناسب را صدا می‌زنیم؛ مثلا اگر "1" بود add_contact(filename)، اگر "2" بود view_contacts(filename)، اگر "3" بود search_contact(filename). اگر "4" بود یک پیام خروج چاپ می‌کنیم و با break از حلقه خارج می‌شیم و برنامه تموم میشه. اگر کاربر هر چیز دیگه‌ای وارد کنه، پیام می‌ده که «انتخاب نامعتبره، دوباره تلاش کن». در آخر هم از الگوی استاندارد پایتون استفاده میکنیم:

```
if __name__ == "__main__":
```

```
main()
```

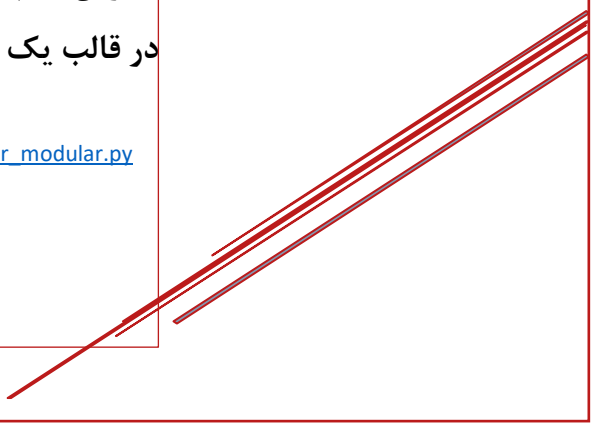
این خط یعنی اگر این فایل رو مستقیما اجرا کنیم (نه این که از جای دیگه importش کنم)، تابع main اجرا بشه و برنامه بالا بیاد. اگر بعدها خواستیم این فایل رو به صورت یک ماژول در برنامه دیگه‌ای استفاده کنیم، با import فقط به توابعش دسترسی داریم و خود برنامه خودکار اجرا نمی‌شه. به طور کلی، توی این پروژه سعی کردیم هم کار با فایل، هم استفاده پایه‌ای از ماژول csv، هم کنترل خطا با try/except و هم طراحی ماژولار با چند تابع جداگانه رو تمرین کنیم، طوری که اگر بعدا بخوایم همین ایده ساده رو تبدیل به یک پروژه جدی‌تر (مثلا اتصال به PostgreSQL توی تمرین سوم یا رابط کاربری گرافیکی) بکنیم، اسکلت اولیه‌اش آماده و قابل توسعه باشه.

Output



تمرین دوم: کد را ماژولار بنویسید و CONTACT MANAGER را در قالب یک پکیج پایتون پیاده‌سازی کنید.

Github Link: https://github.com/Faeze-Ahmadi/AP_A2_Faeze_Ahmadi/blob/main/2_contact_manager_modular.py

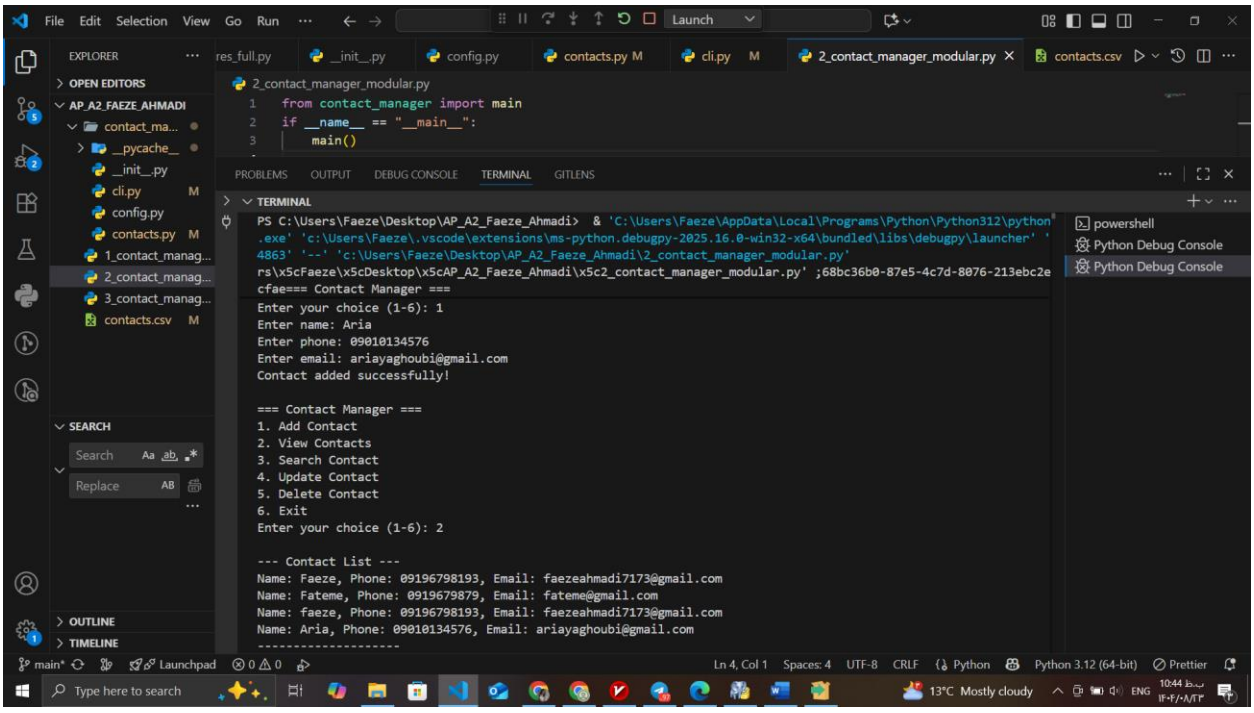


Document

توی این نسخه جدید از برنامه، ایده و منطق اصلی همون نسخه قبلی مدیریت مخاطب‌هاست، اما ساختار پروژه رو کاملا ماژولار و استاندارد پایتونی کردیم و اون رو در قالب یک پکیج پیاده‌سازی کردیم. توی نسخه قبلی، همه توابع و منوی برنامه توی یه فایل قرار داشتن، اما اینجا برای مرتب‌تر شدن، قابلیت توسعه آینده و خوانایی بهتر، کد رو به چند فایل تقسیم کردیم و هر بخش رو در جای مناسب خودش قرار دادیم. برای شروع، یک پوشه به عنوان پکیج ساختیم و فایل‌های اون رو مشخص میکنیم. این پکیج شامل سه فایل اصلی هست: یه فایل مخصوص تنظیمات که فقط نام فایل مخاطب‌ها در اون قرار داره، یه فایل که تمام منطق مربوط به اضافه کردن مخاطب، نمایش لیست مخاطب‌ها، حذف مخاطب و سرچ رو مدیریت می‌کنه و یک فایل مخصوص رابط خط فرمان که منوی برنامه رو نشون می‌ده و تصمیم می‌گیره کدوم تابع اجرا بشه. این جداسازی باعث می‌شه اگر بعدا برنامه بزرگ‌تر شد، یا بخوایم قابلیت‌های جدید مثل حذف مخاطب یا ویرایش اطلاعات اضافه کنیم، همه چیز خیلی منظم و قابل کنترل باشه. یکی از مهم‌ترین نکات این نسخه اینه که فایل اصلی برنامه فقط نقش اجراکننده رو داره و خودش هیچ منطق پردازشی نداره. این فایل فقط پکیج ساخته شده رو وارد می‌کنه و تابع اصلی رو صدا

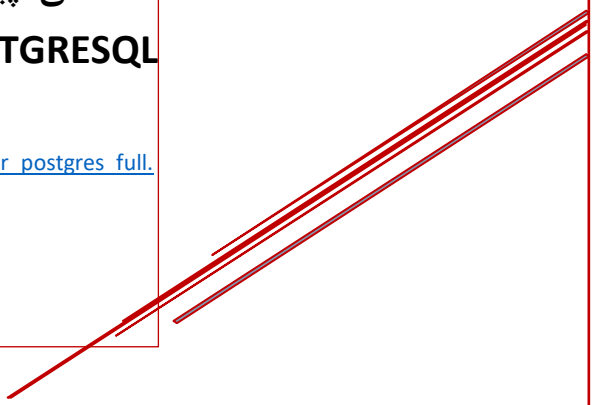
میزنه. این رویکرد همون استانداردیه که توی پروژه‌های واقعی پایتون استفاده می‌شه و کمک می‌کنه که برنامه تمیز و قابل فهم بمونه و استاد هم دقیقا همین رو گفتن. من قبلا در یه مقاله خونده بودم که ماژولار کد زدن از اینکه برنامه ما اسپاگتی کد بشه جلوگیری می‌کنه. یعنی دیگه برنامه‌مون مثل ماکارونی پیچ در پیچ و توی هم نمی‌شه. در ادامه، توی بخش مدیریت مخاطب‌ها همون اعتبارسنجی ساده‌ای رو که برای ورودی‌ها داشتیم حفظ کردیم؛ یعنی اگر نام یا ایمیل فقط عدد باشند خطا می‌دیم و شماره تلفن رو مجبور کردیم که فقط عدد باشه. بعد، تمام عملیات مربوط به فایل را همچنان با فایل CSV انجام می‌دهیم، اما این بار این کار رو در یک فایل مستقل داخل پکیج انجام میدیم. البته که استفاده از فایل CSV کمک می‌کنه اطلاعات مخاطب‌ها به شکل استاندارد ذخیره بشن و اگر بعدها بخوایم برنامه رو به یه پایگاه داده مثل PostgreSQL منتقل کنیم، انتقال خیلی راحت‌تر میشه. به طور خلاصه، این نسخه جدید همون برنامه قبلیه اما با یک ساختار حرفه‌ای‌تر و ماژولار نوشته شده. هر بخش از برنامه در فایل مخصوص به خودش قرار گرفته تا کد تمیزتر، قابل‌نگهداری‌تر و آماده توسعه باشه.

Output



نسخه‌ای پیاده‌سازی کنید که داده‌ها را در پایگاه‌داده POSTGRESQL ذخیره کند.

Github Link: https://github.com/Faeze-Ahmadi/AP_A2_Faeze_Ahmadi/blob/main/3_contact_manager_postgres_full.py



Document

توی تمرین سوم، نسخه جدیدی از برنامه مدیریت مخاطب رو پیاده‌سازی کردیم که به جای ذخیره‌سازی داده‌ها در فایل متنی (CSV)، اطلاعات رو توی یه پایگاه‌داده واقعی یعنی PostgreSQL نگهداری می‌کنه. این کار باعث می‌شه برنامه ساختارمندتر، قابل‌اعتمادتر و مناسب برای پروژه‌های حرفه‌ای‌تر باشه.

دلیل استفاده از PostgreSQL

توی نسخه‌های قبلی، مخاطب‌ها داخل یک فایل ذخیره می‌شدن. این روش برای برنامه‌های کوچیک کافیه اما مشکلاتی مانند سختی در سرچ کردن، افزایش احتمال خطا و دشوار بودن به‌روزرسانی داده‌ها رو به همراه داره. به همین دلیل توی این تمرین از PostgreSQL استفاده شد؛ یک سیستم مدیریت پایگاه‌داده قدرتمند که داده‌ها رو به صورت جدولی ذخیره می‌کنه و امکان جستجو، ویرایش و حذف مطمئن و آسون رو فراهم میاره.

ساختار پایگاه‌داده

برای این تمرین یه پایگاه داده با نام contacts_db ایجاد شد.

داخل این پایگاه‌داده، جدولی به نام contacts ساخته شد که شامل ستون‌های زیر هست:

- id: شناسه منحصر به فرد برای هر مخاطب
- name: نام مخاطب
- phone: شماره تماس
- email: ایمیل

این جدول نقش همون فایل قبلی رو داره، اما با ساختار استاندارد، قابل جستجو و قابل مدیریت‌تر.

اتصال پایتون به دیتابیس

برای اینکه برنامه پایتون بتونه با PostgreSQL ارتباط برقرار کنه، از کتابخانه psycopg2 استفاده شده. این کتابخانه به برنامه اجازه می‌ده که به دیتابیس وصل بشه، دستورهای درج، جستجو، ویرایش و حذف رو اجرا کنه و نتایج رو دریافت کرده و به کاربر نشون بده. همچنین اتصال به دیتابیس فقط یک بار تنظیم می‌شه و در تمام بخش‌های برنامه مورد استفاده قرار می‌گیره.

عملکرد برنامه در نسخه دیتابیس

توی این نسخه تمامی قابلیت‌ها (add، view، search، update و delete) روی پایگاه داده انجام می‌شه.

وقتی کاربر گزینه‌ای مثل «Add Contact» رو انتخاب می‌کنه:

- اطلاعات ورودی دریافت می‌شه
- برنامه از طریق psycopg2 یه دستور SQL به دیتابیس ارسال می‌کنه
- داده در جدول ذخیره می‌شه

برای جستجو، ویرایش و حذف هم به همین ترتیب دستورهای مناسب SQL اجرا می‌ششن و تغییرات مستقیما روی دیتابیس میاد.

تفاوت‌های اصلی این نسخه با نسخه‌های قبلی

- داده‌ها در جدول پایگاه‌داده ذخیره می‌شن نه توی فایل.
- هر مخاطب یک شناسه عددی (ID) داره که برای آپدیت و حذف استفاده می‌شه.
- داده‌ها حتی اگر فایل‌ها جا به جا بشن یا نام پروژه تغییر کنه، توی دیتابیس هستن.
- ساختار ماژولار کد کاملا حفظ شده و فقط منبع داده از فایل به دیتابیس تغییر پیدا کرده.

Output