To Do List Project
(Phase 1 - in memory)

PREPARED FOR

Software Engineering Course AUT

Date

Oct 2025

ToDoList – Python OOP (In-Memory)

مقدمه

می‌خواهیم یک **تودولیست** با تکیه بر **OOP پایتون** بسازیم که در فاز اول با **In-Memory Storage** (ذخیره‌سازی موقتی در حافظهٔ) و CLI کار می‌کند.
رویکرد کلی ما این است که ابتدا چند **مفهوم پایه** را مرور کنیم و سپس **به‌تدریج قابلیت‌ها** را اضافه کنیم؛ یعنی هر فاز یک خروجی قابل استفاده داشته باشد و روی شانهٔ فاز قبلی رشد کند.

همچنین می‌خواهیم در این پروژه مجموعه‌ای از **کانونشن‌های برنامه‌نویسی (Coding Conventions)** را رعایت کنیم.

نکته: در ادامه درس با این موضوعات بیشتر و مفصل‌تر آشنا خواهید شد، این بخش به عنوان مقدمه برای آشنایی با ادبیات توسعه محصول آورده شده است.

رویکرد توسعه: Incremental Development + Agile

- **Incremental Development**: سیستم را به **قطعات کوچک مستقل** خرد می‌کنیم . هر قطعه را کامل می‌سازیم و تحویل می‌دهیم، سپس قطعهٔ بعدی را اضافه می‌کنیم. این باعث می‌شود همیشه یک نسخهٔ قابل اجرا داشته باشیم.

- **Agile**:
  - **بازخورد سریع**: بعد از هر افزایش (Increment)، رفتار واقعی ابزار را می‌بینیم و بر اساس بازخورد، **اولویت‌ها/الزامات** را تنظیم می‌کنیم.
  - **تحویل مستمر**: هر فاز، قابلیت‌های مشخص و قابل تست دارد (Definition of Done روشن).
  - **سادگی در شروع**: از **In-Memory** شروع می‌کنیم؛ در فازهای بعدی به **Persistency** (مثل RDBM) و سپس Web API و تست‌های اوتومیتد می‌رسیم.
  - **ریسک کم**: خطاها و فرض‌های غلط زودتر آشکار می‌شوند چون هر Increment کوچک و قابل ارزیابی است.

یوزر استوری (مهندسی نیازمندی‌ها)

برای اینکه بفهمیم دقیقاً چه **تسک‌هایی** در پروژه باید انجام شوند، لازم است اول با مفهوم **یوزر استوری (User Story)** آشنا شویم.

یوزر استوری چیست؟

- یک روش **Agile** برای تعریف نیازمندی‌ها از دید کاربر نهایی است.
- قالب استاندارد دارد:
  «به‌عنوان یک [نوع کاربر] می‌خواهم [یک قابلیت] داشته باشم تا [ارزش/هدف] حاصل شود.»
- کمک می‌کند به‌جای تمرکز بر «چطور پیاده‌سازی می‌کنیم»، روی «چه نیازی برطرف می‌شود» تمرکز کنیم.

فانکشنال و نان‌فانکشنال ریکوایرمنت‌ها دو نوع نیازمندی نرم‌افزاری هستند؛ اولی می‌گوید «سیستم *چه کاری* انجام دهد» (رفتار و قابلیت‌ها) و دومی می‌گوید «سیستم *چطور* آن کار را انجام دهد» (ویژگی‌های کیفی و محدودیت‌ها).

فانکشنال ریکوایرمنت (Functional Requirement)

- توضیح می‌دهند سیستم **چه کارهایی باید انجام دهد**.
- معادل «ویژگی‌ها و رفتارهای قابل مشاهدهٔ سیستم».
- مثلاً: «کاربر بتواند پروژه ایجاد کند.»

نان‌فانکشنال ریکوایرمنت (Non-Functional Requirement)

- توضیح می‌دهند سیستم **چطور باید عمل کند**، بدون اینکه به یک فیچر خاص اشاره کنند.
- شامل: کارایی، امنیت، مقیاس‌پذیری، خوانایی کد، تست‌پذیری، پیام خطای کاربرپسند، ...
- مثلاً: «پیام خطا باید واضح و قابل فهم باشد.» یا «کد باید مطابق PEP8 نوشته شود.»

در رویکرد اجایل، معمولا در هر پروژه، برای هر قابلیت ابتدا یوزر استوری را می‌نویسیم، بعد معیار پذیرش (Acceptance Criteria) مشخص می‌کنیم، و در نهایت آن‌ها را به فانکشنال و نان‌فانکشنال ریکوایرمنت‌ها نگاشت می‌دهیم.

یوزر استوری، **فانکشنال ریکوایرمنت** و **نان‌فانکشنال ریکوایرمنت** سه لایه یا سه «نمای مختلف» از نیازمندی‌های یک پروژه هستند و **ارتباط مستقیم** با هم دارند.

در این پروژه قرار است یک **سیستم مدیریت ToDoList** طراحی و پیاده‌سازی شود. در این سیستم، کاربر می‌تواند چندین **پروژه** ایجاد کند و برای هر پروژه مجموعه‌ای از **تسک‌ها (وظایف)** را تعریف و مدیریت نماید. در این بخش نیازمندی‌های این پروژه در فرمت یوزر استوری توضیح داده شده است.

یوزر استوری‌ها و معیارهای پذیرش

۱) ساخت پروژه

- **User Story**:

  به‌عنوان یک کاربر می‌خواهم پروژهٔ جدید بسازم تا بتوانم کارهایم را دسته‌بندی کنم.

- **Acceptance Criteria**:
  - نام ≤ ۳۰ واژه، توضیح ≤ ۱۵۰ واژه.
  - نام پروژه تکراری نباشد.
  - عبور از سقف `MAX_NUMBER_OF_PROJECT` خطا بدهد.

۲) ویرایش پروژه

- **User Story**:

به‌عنوان یک کاربر می‌خواهم بتوانم نام و توضیح پروژه را تغییر دهم تا همیشه پروژه‌هایم به‌روز باشند.

- **Acceptance Criteria**:
  - محدودیت واژه رعایت شود.
  - نام جدید نباید با پروژه‌های دیگر تکراری باشد.

۳) حذف پروژه

- **User Story**:

  به‌عنوان یک کاربر می‌خواهم پروژه را حذف کنم تا بتوانم کارهای غیرضروری را پاک‌سازی کنم.

- **Acceptance Criteria**:
  - با حذف پروژه، تمام تسک‌هایش هم حذف شوند. (**Cascade Delete**)
  - پیام موفقیت یا خطای مناسب نمایش داده شود.

مفهوم **Cascade** در این پروژه به این معناست که بعضی موجودیت‌ها به موجودیت‌های دیگر وابسته‌اند و حذف یکی باید به‌طور زنجیره‌ای باعث حذف وابسته‌ها شود. در مثال ما، یک **پروژه** به‌عنوان ظرف (Container) برای تعدادی **تسک** عمل می‌کند؛ بنابراین اگر پروژه حذف شود، نگه داشتن تسک‌های بدون پروژه منطقی نیست و باعث داده‌های یتیم (Orphan Data) می‌شود. به همین دلیل، سیاست **Cascade Delete** اعمال می‌کنیم: با حذف پروژه، همهٔ تسک‌های وابسته به آن پروژه نیز به‌طور خودکار حذف می‌شوند تا سازگاری داده‌ها (Data Consistency) حفظ شود و پیچیدگی مدیریت دستی کاهش یابد.

---

۴) افزودن تسک

- **User Story**:

  به‌عنوان یک کاربر می‌خواهم در هر پروژه تسک اضافه کنم تا بتوانم فعالیت‌هایم را ثبت کنم.

- **Acceptance Criteria**:
  - عنوان ≤ ۳۰ واژه، توضیح ≤ ۱۵۰ واژه.
  - استتوس یکی از `todo | doing | done`.
  - ددلاین اگر وارد شد، تاریخ معتبر باشد.
  - عبور از سقف `MAX_NUMBER_OF_TASK` خطا بدهد.
- **Default Behavior**:
  - هنگام ایجاد تسک، مقدار پیش‌فرض استتوس همیشه **`todo`** است؛ کاربر بعداً می‌تواند آن را به `doing` یا `done` تغییر دهد.

۵) تغییر وضعیت تسک

- **User Story**:

  به‌عنوان یک کاربر می‌خواهم وضعیت یک تسک را تغییر دهم تا روند پیشرفت آن را پیگیری کنم.
- **Acceptance Criteria**:

- فقط سه مقدار معتبر پذیرفته شود.
- تغییر ذخیره شود و قابل مشاهده باشد.

۶) ویرایش تسک

- **User Story**:

به‌عنوان یک کاربر می‌خواهم عنوان، توضیح، ددلاین و وضعیت تسک را تغییر دهم تا همیشه اطلاعات به‌روز داشته باشم.

- **Acceptance Criteria**:
  - محدودیت واژه و صحت ددلاین بررسی شود.
  - استتوس باید معتبر باشد.

۷) حذف تسک

- **User Story**:

به‌عنوان یک کاربر می‌خواهم یک تسک را حذف کنم تا کارهای بی‌استفاده پاک شوند.

- **Acceptance Criteria**:
  - حذف بر اساس شناسهٔ تسک در همان پروژه انجام شود.
  - پیام مناسب (موفق یا خطا) نمایش داده شود.

۸) نمایش لیست پروژه‌ها

User Story:

به‌عنوان یک کاربر می‌خواهم بتوانم همهٔ پروژه‌هایم را مشاهده کنم تا بدانم چه پروژه‌هایی ساخته‌ام.

Acceptance Criteria:

- فهرست پروژه‌ها همراه با شناسه، نام و توضیح نمایش داده شود.
- اگر هیچ پروژه‌ای وجود نداشت، پیام مناسب (مثلاً "پروژه‌ای وجود ندارد") نمایش داده شود.
- خروجی به‌صورت مرتب (بر اساس زمان ساخت) قابل مشاهده باشد.

۹) نمایش همه تسک‌های یک پروژه

User Story:

به‌عنوان یک کاربر می‌خواهم بتوانم تمام تسک‌های مربوط به یک پروژه را ببینم تا وضعیت کارهای آن پروژه را بررسی کنم.

Acceptance Criteria:

- تسک‌ها بر اساس پروژه فیلتر و نمایش داده شوند.
- اطلاعات هر تسک شامل شناسه، عنوان، وضعیت، و ددلاین باشد.
- اگر پروژه وجود نداشته باشد یا تسکی نداشته باشد، پیام مناسب نمایش داده شود.

استخراج نیازمندی‌ها

فانکشنال

- مدیریت پروژه‌ها: ایجاد، ویرایش، حذف، نمایش لیست پروژه‌ها.
- مدیریت تسک‌ها: ایجاد، ویرایش، حذف، تغییر وضعیت، نمایش همهٔ تسک‌های یک پروژه.
- اعمال محدودیت‌های واژه و استتوس.
- اعمال سقف تعداد پروژه/تسک از `env`.

فایل **`.env`** یک محل ساده و استاندارد برای ذخیره‌ی **تنظیمات و متغیرهای محیطی** پروژه است؛ مثل سقف تعداد پروژه‌ها (`MAX_NUMBER_OF_PROJECT`) یا تسک‌ها (`MAX_NUMBER_OF_TASK`). دلیل نیاز ما به `.env` این است که **کد و تنظیمات** را از هم جدا کنیم: کد همیشه ثابت می‌ماند، ولی مقادیر تنظیمات (مثلاً در محیط توسعه، تست یا محصول) ممکن است فرق داشته باشند. این روش یک **Best Practice** در توسعهٔ نرم‌افزار است چون: ۱) امنیت را بالا می‌برد (مثلاً رمزها داخل کد هاردکد نمی‌شوند)، ۲) تغییرات تنظیمات بدون تغییر در سورس‌کد امکان‌پذیر است، ۳) قابل حمل و یکپارچه با ابزارهایی مثل Docker یا Poetry است.

پرکتیس‌های مهم شامل: تعریف یک فایل نمونه (`env.example.`) برای اعضای تیم، عدم انتشار فایل واقعی `env.` در مخزن گیت، و بارگذاری مقادیر با کتابخانه‌هایی مثل `python-dotenv`.

نان‌فانکشنال

- **خوانایی کد** (PEP8, conventions).
- **قابلیت نگه‌داری** (OOP، لایه‌بندی تمیز).
- **کارایی کافی** (In-Memory سبک و سریع).
- **کاربرپسندی پیام‌ها** (خطاها واضح باشند).
- **قابلیت توسعه** (آماده برای افزودن Persistency در فاز بعد).

مفهوم **Persistency** (پایداری داده) به این معناست که اطلاعات برنامه پس از پایان اجرای آن هم حفظ شوند و با هر بار اجرای مجدد از بین نروند. در فاز اول پروژه ما داده‌ها به‌صورت **In-Memory** نگهداری می‌شوند؛ یعنی با بستن برنامه همه چیز پاک می‌شود. اما در فازهای بعدی برای واقعی‌تر شدن سیستم، نیاز به مکانیزم‌های Persistency داریم مثل ذخیره در فایل‌های JSON، دیتابیس‌های سبک (SQLite)، یا حتی دیتابیس‌های کامل (PostgreSQL, MySQL). افزودن Persistency باعث می‌شود کاربر بتواند پس از بستن و باز کردن دوباره برنامه، همچنان پروژه‌ها و تسک‌هایش را در همان وضعیت قبلی مشاهده کند.

نکته: برخی از موارد فنی مورد نیاز پروژه در ادامه آورده شده است. رعایت این موارد الزامی (mandatory) است.

کانونشن برنامه‌نویسی (Coding Conventions)

تعریف

کانونشن‌ها مجموعه‌ای از **قواعد و الگوهای استاندارد** هستند که توسعه‌دهندگان در یک پروژه رعایت می‌کنند تا کد:

- **خواناتر** شود،

- **نگه‌داری آسان‌تر** باشد،
- تیم‌های مختلف بتوانند **یکپارچه کار کنند**،
- و خطاهای ناشی از سلیقه‌های متفاوت کمتر شود.

فایل کانونشن‌ها رو به دقت مطالعه کنید. (مطالعه فایل ضروری بوده و منبع بررسی و ارزیابی کیفیت کد شما خواهد بود.)

لینک فایل کانونشن‌ها

مدیریت نسخه و همکاری تیمی (Version Control & Workflow)

برای پیشبرد این پروژه لازم است از Git به‌عنوان سیستم کنترل نسخه استفاده شود و مخزن پروژه نیز بر روی GitHub قرار گیرد.

کامیت‌ها (Commits)

- باید پالیسی گفته شده در پروژه قبل (پورتفولیو وب سایت) به طور کامل رعایت شود.

برنچ‌ها (Branches)

- توسعه‌ی اصلی روی برنچ `develop` انجام می‌شود.
- برنچ `main` یا `master` همیشه فقط نسخه‌های پایدار و آماده‌ی انتشار را نگه می‌دارد.
- برای هر فیچر یا باگ‌فیکس یک برنچ جدید از `develop` ساخته می‌شود (`feature/add-task-deadline`). بعد از تکمیل، با Pull Request یا Merge به `develop` برمی‌گردد. سپس بعد از بررسی دولوپ و سازگاری فیچر جدید با کلیت محصول، با `main` مرج می‌شود.
- دلیل این روش:
  - **کاهش ریسک** (تغییرات ناپایدار وارد `main` نمی‌شوند).
  - **تست و مرور کد** قبل از ادغام در خط اصلی توسعه.
  - **همکاری تیمی تمیز** بدون درگیری و تداخل تغییرات.

در این مستند، فاز اول پروژه (**ToDoList - Python OOP (In-Memory**) را تعریف کردیم.
مسیر توسعه بر اساس **Incremental Development** و **Agile** طراحی شد تا در هر گام یک خروجی قابل استفاده داشته باشیم. نیازمندی‌ها به‌صورت **یوزر استوری**، **فانکشنال** و **نان‌فانکشنال** بیان شدند، و قواعد مهم مثل **Persistency**، **Cascade Delete** و استفاده از فایل `.env` برای تنظیمات معرفی شد. همچنین مجموعه‌ای از **کانونشن‌های برنامه‌نویسی** برای نام‌گذاری، تایپینگ، مستندسازی و مدیریت خطا ارائه گردید تا کیفیت و یکپارچگی کد تضمین شود.

برای پیشبرد پروژه از **Git** استفاده می‌کنیم. همهٔ توسعه‌ها روی برنچ `develop` انجام می‌شوند و تنها نسخه‌های پایدار به `main` منتقل می‌گردند. هر تغییر از طریق **Merge Request** انجام می‌شود و قبل از ادغام، کد خود را ریویو کنید تا اطمینان حاصل شود کد تمیز، پایدار و مطابق با کانونشن‌هاست. این رویکرد باعث می‌شود تاریخچهٔ پروژه شفاف، کیفیت کد بالا باشد.

مدیریت وابستگی‌ها با Poetry

برای این پروژه تصمیم گرفتیم از **Poetry** استفاده کنیم. Poetry یک ابزار مدرن برای **مدیریت وابستگی‌ها (Dependencies)** و **محیط‌های مجازی (Virtual Environments)** در پایتون است که جایگزین روش‌های سنتی مثل `pip + requirements.txt` می‌شود.

دلیل انتخاب Poetry این است که:

- همهٔ وابستگی‌ها به‌صورت دقیق در فایل `pyproject.toml` مدیریت می‌شوند.
- هر پروژه محیط مجازی جداگانه دارد و وابستگی‌ها با پروژه‌های دیگر تداخل پیدا نمی‌کنند.
- اضافه یا حذف کردن پکیج‌ها ساده و قابل پیگیری است (`poetry add/remove`).
- انتشار یا اجرای پروژه نیز به‌راحتی از طریق خود Poetry انجام می‌شود.

به این ترتیب، کل تیم می‌تواند مطمئن باشد که محیط اجرا و کتابخانه‌های موردنیاز پروژه همیشه یکسان و هماهنگ هستند.

چی کار می‌کنه؟

1. **مدیریت وابستگی‌ها (Dependencies):**

- به جای اینکه دستی `pip install` بزنی و بعدش لیست پکیج‌ها رو توی `requirements.txt` نگه داری، با Poetry فقط می‌گی `poetry add requests` و خودش نسخه‌ها رو مدیریت می‌کنه.
- همه‌چی داخل فایل `pyproject.toml` ذخیره می‌شه (مثل `package.json` توی جاوااسکریپت).

2. **ساخت محیط مجازی (Virtualenv):**
   - Poetry خودش برای پروژه یه virtualenv جدا می‌سازه، پس لازم نیست با `venv` یا `pipenv` درگیر بشی.
   - یعنی هر پروژه پکیج‌های مخصوص به خودش رو داره.

3. **اجرای پروژه:**
   - به‌راحتی می‌تونی دستورها رو با `poetry run python main.py` اجرا کنی.

چرا خوبه؟

- کار با وابستگی‌ها تمیزتره (دیگه مشکل conflict بین نسخه‌ها کمتر می‌شه).
- همه‌چی توی یه فایل `pyproject.toml` مدیریت می‌شه.
- از `requirements.txt` و نصب دستی راحت می‌شی.
- توی پروژه‌های تیمی یا بزرگ، حرفه‌ای‌تر و پایدارتره.

Poetry هم یه package manager هست، هم یه project manager.

یه جوری می‌شه گفت نقش همزمان `pip` + `venv` + `setup.py` + `requirements.txt` رو با هم بازی می‌کنه، ولی خیلی تمیزتر و مدرن‌تر.

هدف و ددلاین پروژه

در این فایل با یکی از روش‌های توسعه اجایل در حدی آشنا شدیم (در فازهای بعدی ملموس‌تر خواهد شد.)

هدف از این پروژه این است که مهارت شما در کار با Python OOP سنجیده شود و همچنین با Poetry و کانوشن‌های پایتونی آشنا شوید.

نکته‌ای که در طول توسعه این فاز باید به آن توجه داشته باشید این است که در فازهای بعدی لازم است داده‌ها به‌صورت پایدار (persistent) ذخیره شوند. همچنین این ToDoList قرار است به یک وب‌اپلیکیشن تبدیل شود که با FastAPI پیاده‌سازی خواهد شد و برای آن تست نیز نوشته می‌شود. بنابراین کد را طوری طراحی و پیاده‌سازی کنید که این قابلیت‌ها در آینده به‌سادگی اضافه شوند و نیاز به تغییرات گسترده نباشد. در جلسات پنج‌شنبه نیز سعی می‌کنیم روی این موضوع تمرکز کنیم که چگونه این کار را انجام دهیم. البته توجه داشته باشید که این مسیر لزوماً کم‌هزینه نیست و گاهی برای یادگیری، عمداً کار را سخت‌تر پیش می‌بریم تا تجربه‌ی بیشتری به دست بیاید.

همچنین یک هدف دیگر این پروژه این است که در هنگام پیاده‌سازی روی **تفکیک بیزینس لاجیک** تمرکز کنید. منظور از بیزنس لاجیک همان بخش اصلی منطق و قوانین سیستم است که مشخص می‌کند برنامه دقیقاً چه کاری انجام می‌دهد (مثل اضافه‌کردن تسک جدید، ویرایش یا تغییر وضعیت آن در ToDoList). این منطق باید از جزئیات مربوط به **ورودی/خروجی** (مثل نحوه‌ی گرفتن داده از کاربر در CLI یا نمایش آن در وب) و همچنین از **زیرساخت ذخیره‌سازی** (مثل فایل، دیتابیس یا API) جدا باشد.

بنابراین کد خود را به‌صورت **ماژولار** و **لایه‌لایه** طراحی کنید کد تا هر بخش وظیفه‌ی مشخصی داشته باشد. یک لایه برای منطق اصلی (core/business logic)، یک لایه برای دسترسی به داده (data access/storage)، و یک لایه برای رابط کاربری یا API.

همچنین توجه کنید که **نحوه‌ی ارتباط این لایه‌ها و مدیریت وابستگی بین آن‌ها** از مسائل مهم در طراحی نرم‌افزار است؛ یعنی هر بخش باید فقط به اندازه‌ی لازم از بخش‌های دیگر خبر داشته باشد و از درهم‌تنیدگی زیاد اجتناب شود.

این موضوع در جلسه‌ی پنج‌شنبه بیشتر بررسی خواهد شد. **رعایت فرمت پیشنهادی و ملزومات ذکر شده در پروژه الزامی است.**

اگر تا امروز با Python کار نکردید، فرصت خوبی است که با انجام این پروژه یادگیری را به‌صورت عملی شروع کنید (**learn by doing**).

ددلاین نهایی ۲۵ مهر می‌باشد.