internStudio

PYTHON DEVELOPMENT INTERNSHIP

TASK GUIDE

Write Python that ships. Automate, build, and scale with clean code.

SIX ADVANCED PROJECTS — CHOOSE ANY TWO

01 **Automated Web Scraping & Data Pipeline**

02 **CLI-Based Personal Finance Tracker**

03 **REST API for a Real-World Domain**

04 **Automated Report Generation & Delivery System**

05 **Concurrent Web Crawler & Content Indexer**

06 **Workflow Automation & Notification Bot**

INTERNSHIP
Python Development

REQUIRED TASKS
Complete any 2 projects

FOCUS
Scripting · APIs · Automation · Data Handling · Testing

INTERNSHIP INSTRUCTIONS

Your Internship Tasks

Select and complete **any 2 projects** from the task list. Each should be a real Python application or tool — not a script that only runs once on your machine.

Design	Build	Test	Package	Document	Deploy

EVERY PROJECT SHOULD INCLUDE

- [ ] A well-structured, modular codebase
- [ ] Proper error handling, not bare except blocks
- [ ] Command-line arguments or a real interface
- [ ] Automated tests for core logic
- [ ] Logging instead of print-statement debugging
- [ ] A requirements.txt or pyproject.toml
- [ ] Type hints where they aid clarity
- [ ] Configuration kept out of hard-coded values

BEFORE YOU SUBMIT

- [ ] Complete any 2 projects from the list
- [ ] Push the project to GitHub
- [ ] Include a requirements/dependency file
- [ ] Add a README with setup and usage instructions
- [ ] Add sample output or screenshots
- [ ] Submit details via the submission form
- [ ] Share your work on LinkedIn as required

> **Suggested technologies:** Python · requests/BeautifulSoup or Selenium · FastAPI/Flask · SQLite/PostgreSQL · pytest. Optional: Click/argparse, pandas, APScheduler.

Submit your completed work here

Once both projects are tested, documented and pushed to GitHub, submit your project details through the official submission form.

https://forms.gle/vxL5JE8cvRXXfDBv8

> **A note on quality.** A script that only works once on your machine is not a project. Aim for something another developer could clone, configure and run.

01 Automated Web Scraping & Data Pipeline

Build a pipeline that scrapes structured data from multiple sources, cleans it, and stores it on a schedule.

REQUIREMENTS

- Scrape at least two distinct real sources
- Clean and normalize the scraped data
- Detect and skip duplicate records
- Log successes, failures and retries
- Handle pagination and rate limiting responsibly
- Store results in a database, not just a CSV
- Schedule the pipeline to run periodically
- Handle site-structure changes gracefully

Focus: Web scraping at scale, data cleaning, scheduling and resilient error handling.

02 CLI-Based Personal Finance Tracker

Build a command-line tool that tracks income and expenses and generates real financial reports.

REQUIREMENTS

- A real CLI with subcommands (add, list, report)
- Categorization of transactions
- Export reports to CSV or PDF
- Undo / edit / delete of past entries
- Persistent storage in SQLite
- Monthly and category-based summaries
- Input validation with helpful error messages
- Unit tests for the core calculations

Focus: CLI design, persistent storage, reporting and well-tested business logic.

03 REST API for a Real-World Domain

Design and build a production-shaped REST API for a real domain (e.g. a library, clinic, or inventory system).

REQUIREMENTS

- Well-structured resource-based routes
- Request validation with clear error responses
- Database migrations, not ad hoc table creation
- API documentation (OpenAPI/Swagger)
- Authentication and authorization
- Pagination and filtering on list endpoints
- Automated tests for the endpoints
- Sensible logging of requests and errors

Focus: API design, authentication, testing and real documentation practices.

04 Automated Report Generation & Delivery System

Build a system that pulls data on a schedule, generates a formatted report, and delivers it automatically.

REQUIREMENTS

- Pull data from a database or API on a schedule
- Include charts or tables in the report
- Configurable report parameters
- Historical archive of generated reports
- Generate a formatted PDF or Excel report
- Email the report automatically to recipients
- Handle and log delivery failures
- Automated tests for the report logic

Focus: Scheduled automation, document generation and reliable delivery pipelines.

05 Concurrent Web Crawler & Content Indexer

Build a crawler that indexes content across many pages concurrently and makes it searchable.

REQUIREMENTS

- Concurrent or async crawling (not one request at a time)
- Extract and store page content and metadata
- Rank or score search results
- Handle broken links and timeouts gracefully
- Respect robots.txt and crawl limits
- Build a simple search index over crawled content
- Avoid re-crawling unchanged pages
- A simple query interface (CLI or minimal API)

Focus: Concurrency/async programming, indexing and search, and robust failure handling.

06 Workflow Automation & Notification Bot

Build a tool that automates a real recurring workflow (file organization, monitoring, reminders) and notifies you of results.

REQUIREMENTS

- Monitor a real source of change (files, an API, an inbox)
- Take an automated action (move, tag, summarize)
- Configuration file rather than hard-coded rules
- Log every action taken for auditability
- Apply configurable rules to what it finds
- Send notifications (email, Slack, or similar)
- Run reliably on a schedule
- Handle failures without silently doing nothing

Focus: Automation design, integration with external services, and dependable scheduled execution.

FINAL STEP

Ready to Submit?

Verify everything below before you send your work in.

SUBMISSION

- [ ] GitHub repository
- [ ] README with usage examples
- [ ] Sample output or screenshots
- [ ] Submission form completed
- [ ] LinkedIn post where required

PACKAGING

- [ ] requirements.txt or pyproject.toml included
- [ ] Clear setup and run instructions
- [ ] No secrets hard-coded or committed

CODE

- [ ] Modular, readable structure
- [ ] Proper error handling throughout
- [ ] Logging in place, not stray print statements

TESTING

- [ ] Automated tests for core logic
- [ ] Edge cases considered
- [ ] Tests actually pass on a clean checkout

Important. Complete any 2 projects. You should be able to explain your design choices — why this library, why this data structure, why this approach to error handling.

Submission Form

Submit your repository, live link and screenshots here.

https://forms.gle/vxL5JE8cvRXXfDBv8

internStudio

LEARN · BUILD · GROW

www.internstudio.in

team.internstudio@gmail.com