Course: Python development

Jan-Feb, 2025, Arobs

Adi Fatol

Andrei Boros

Daniel Dutu

Introduction Round

Who you are (Name & Role/Background)

What experience do you have (Programming languages, technologies)

What you're looking forward to (What do you expect at the end of the course)

Objective

Learning the **pythonic** ways

Learning the python syntax

Getting accustomed to the python documentation

Experiencing hands-on working with python

Building a working project to showcase

Materials

- E-books
  - Learning Python
  - Python for Everybody
- Slides Python OOP, Design Patterns etc (Adi + Team)
- ChatGPT, Claude, Google

Core Python syntax and features

Project setup and packaging

OOP and functional programming

File handling and error management

Clean architecture

Testing strategies

Performance tools

Deployment and delivery

Sessions 5-8

(Data Science?)

Python Fundamentals

Web Frameworks and Design Patterns

Professional Development

Sessions 1-4

Sessions 9-12

Flask and REST APIs

Design patterns in Python

Framework comparison

Database and authentication

Tools:
Using github copilot
Github repositories

Hands On and Homework

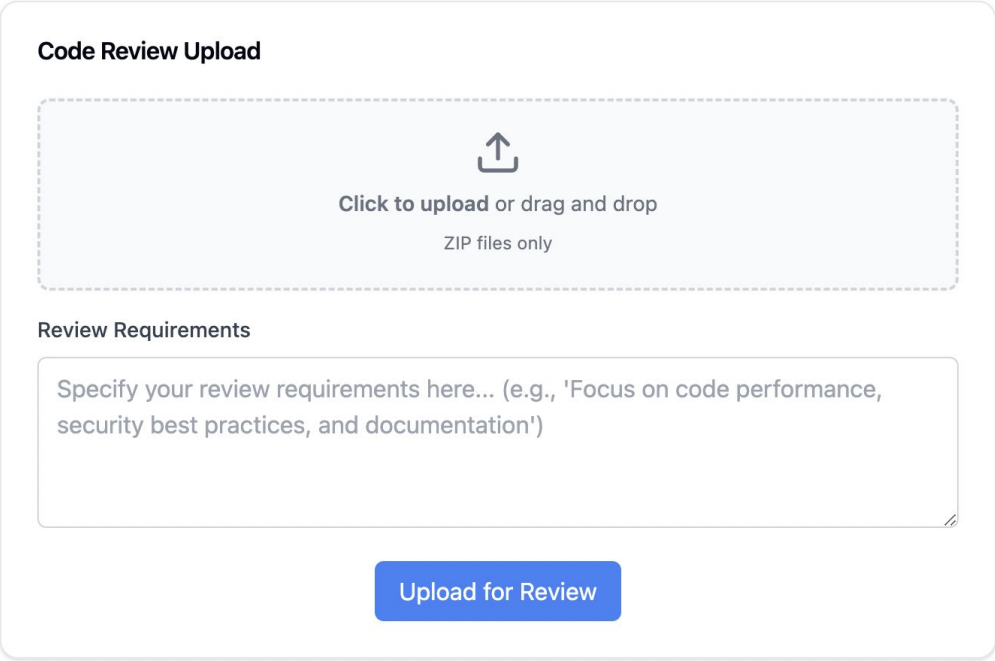
Api for a Document Analysis Tool that can:

- Accept text documents or code files as input
- Provide summaries, code explanations, or documentation generation
- Use a small LLM model like SmolLM for basic text understanding

Skills gained:

- File handling in Python
- Basic Text processing and NLP concepts
- API integration (Flask/FastAPI)
- Command-line argument parsing
- Package management with pip/poetry
- Project structure, Error handling
- Unit testing, Documentation

UI Example

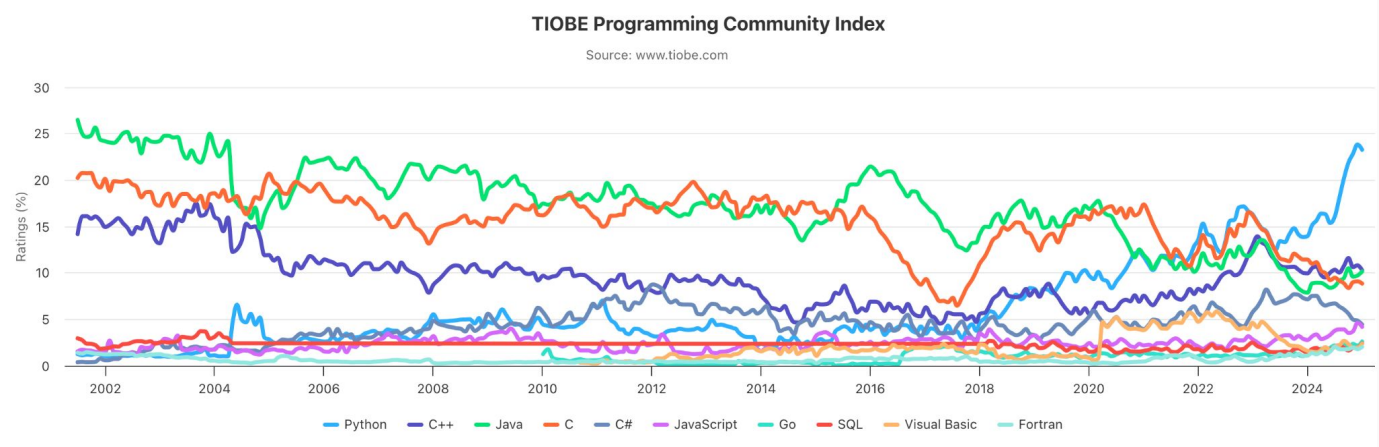

Code Review Upload
Click to upload or drag and drop
ZIP files only
Review Requirements
Specify your review requirements here... (e.g., 'Focus on code performance, security best practices, and documentation')
Upload for Review

Prerequisites

- Programming knowledge / experience with other programming languages
- Git and git commands (clone/commit/push/pull/merge/rebase/branch/remote)
- Integrated Dev Env: VSCode
- Python https://www.python.org/downloads/windows/
  Python 3.12.8 - Dec. 3, 2024
- Until next session: Github free account, ChatGPT/Claude free account (optional), Github Copilot (optional)

Why Python?

- The right programming language can significantly impact your development speed and success.
- Python has emerged as the go-to language for prototyping across various domains, from web applications to data science projects.



Python's Prototyping Advantages

Python's mature ecosystem supports rapid prototyping:

- Rich Ecosystem: Over 400,000 packages available through PyPI
- Strong community support and documentation
- Frameworks for every need (Django, Flask, Pandas, NumPy, Pytorch)
- Easy package installation with pip
- Excellent IDE support with autocomplete, error detection, debugging and lots of extensions

Core attributes

```python
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = [num for num in numbers if num % 2 == 0]
average = sum(even_numbers) / len(even_numbers)
print(f"The average of even numbers is: {average}")  # Output:
6.0
```

```java
import java.util.ArrayList;
import java.util.List;

public class NumberProcessor {
    public static void main(String[] args) {
        List<Integer> numbers = List.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        List<Integer> evenNumbers = new ArrayList<>();

        for (Integer num : numbers) {
            if (num % 2 == 0) {
                evenNumbers.add(num);
            }
        }

        double sum = 0;
        for (Integer num : evenNumbers) {
            sum += num;
        }

        double average = sum / evenNumbers.size();
        System.out.println("The average of even numbers is: " + average);
    }
}
```

Core attributes

Dynamically typed	Statically typed
List comprehensions	Explicit loops
Direct assignment	Explicit Initializations
Python doesn't use classes (although it could)	Java must use class definitions
Python is more concise and beginner-friendly	**Java enforces structure and type safety**

for quick scripting or data analysis, **Python** is better. If you need strong typing and object-oriented rigor, **Java** is preferred

```python
# First interactive Python script
name = input("What's your name? ")
mood = input("How are you feeling today? ")

print(f"Hello, {name}! It's great to have you here. Hope you're feeling {mood} today! 😊")
```

✅ Introduces **user input**
✅ Uses **string formatting**

💡 **Bonus Twist:** Add a condition to respond differently if they type "bad" or "sad," offering an uplifting message.

```python
# main.py - The main script

import greetings  # Import our custom module

def main():
    name = input("What's your name? ")
    mood = input("How are you feeling today? ")

    # Get a personalized greeting
    message = greetings.get_greeting(name, mood)

    print(message)

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

```python
# greetings.py - A module for personalized greetings


def get_greeting(name, mood):
    if mood.lower() in ["sad", "bad", "tired"]:
        return f"Hey {name}, I'm here to cheer you up! 💪😊 "
    elif mood.lower() in ["happy", "great", "awesome"]:
        return f"That's amazing, {name}! Keep the good vibes! ✨😀"
    else:
        return f"Hello, {name}! Hope you're having a nice day. 🌟"
```

A **module** in Python is simply a `.py` file containing Python code (functions, variables, classes) that can be **imported** and reused in other scripts.

Why Use Modules?

✅ **Code Organization** – Keeps your code modular and readable.
✅ **Reusability** – You can import functions without rewriting them.
✅ **Separation of Concerns** – Keeps logic separate from execution.

What is `__name__` in Python?

`__name__` is a **special built-in variable** that indicates how a Python script is being executed.

- When a script is **run directly**, `__name__` is set to `"__main__"`.
- When a script is **imported as a module**, `__name__` is set to the module's filename (without `.py`).

```python
import random  # Import the random module

# Generate a random integer between 1 and 10
rand_int = random.randint(1, 10)

# Pick a random item from a list
choices = ["Python", "Java", "C++", "Rust"]
random_choice = random.choice(choices)

print(f"Random number: {rand_int}")
print(f"Random programming language: {random_choice}")
```

```python
from math import sqrt, pi  # Import only sqrt and pi from math

num = 49
square_root = sqrt(num)  # No need to use math.sqrt()

circle_radius = 5
circle_area = pi * (circle_radius ** 2)  # Using pi directly

print(f"The square root of {num} is {square_root}")
print(f"The area of a circle with radius {circle_radius} is
{circle_area}")
```

```python
from greetings import get_greeting  # Import only get_greeting

def main():
    name = input("What's your name? ")
    mood = input("How are you feeling today? ")

    # Call the imported function directly
    message = get_greeting(name, mood)

    print(message)

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

```
from greetings import *  # Imports all functions


print(get_greeting("Alice", "happy"))  # Works fine, but...
```

⚠️ **Why is this discouraged?**

- It can **overwrite existing variables/functions** without warning.
- Makes it **hard to track** where functions come from.
- Not explicit, which reduces readability.

✅ **Best practice:** Import only what you need (`from greetings import get_greeting`).

📌 **Python only looks for modules in specific directories (`sys.path`). If your module is elsewhere, you need to adjust the path.**

```
import sys

sys.path.append('/path/to/your/module') # Add a module path


import my_custom_module
```

✅ **Best practice:** Keep modules in the same project folder or package.

Python packages

📌 A **package** is just a folder containing multiple modules with an `__init__.py` file.

```
my_project/
|── main.py
|── utilities/
|    |── __init__.py  # Marks this as a package
|    |── greetings.py
|    |── math_helpers.py


# Now you can do:
from utilities.greetings import get_greeting
```

✅ **Best practice:** Use packages to organize large projects.

pip

📌 **Python has thousands of external libraries you can install with `pip`.**

Example:

sh
```
$> pip install requests
```

```
import requests

response = requests.get("https://api.github.com")

print(response.status_code)
```

✅ **Best practice:** Use `requirements.txt` or `pyproject.toml` to manage dependencies.

https://pypi.org/

Help Sponsors Log in Register

Find, install and publish Python packages with the Python Package Index

Search projects

Or browse projects

604,673 projects 6,539,259 releases 13,205,688 files 897,437 users

python™ Package Index

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.

Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.

How to draw an owl

1.

2.

1. Draw some circles

2. Draw the rest of the fucking owl

📌 Homework: Even & Squared Filter

Objective: Write a Python program that takes a list of numbers from the user, filters only the even numbers using **list comprehension**, squares them, and prints the result.

Part 1: Create a Simple Module (number_utils.py)

1. Create a package called `math_tools` with a module named `number_utils.py`.
2. Inside `number_utils.py`, write two functions:
   - One that filters even numbers using **list comprehension**.
   - One that squares a list of numbers using **list comprehension**.

Part 2: Install and Use an External Package (pip)

Use `pip` to install **numpy** (for an alternative method).

```
pip install numpy
```

Modify the script to:

1. Process numbers using **your module**.
2. Also show an alternative **NumPy-based** approach (for comparison).

Bonus Challenge (Optional)

1 Visualize the Squared Numbers (Matplotlib)

- Instead of just printing numbers, **plot a bar chart** where:
  - The x-axis shows the original even numbers.
  - The y-axis shows their squared values.
- **Install `matplotlib` if needed:**
  bash:
  ```
  pip install matplotlib
  ```
- **Hint:** Use `plt.bar()` to create a simple bar chart.

2 Generate a Histogram of Even Numbers

- Show how many even numbers fall within specific value ranges using a **histogram**.
- Example: If the user enters **20 numbers**, the histogram can show the distribution of **even numbers**.
- **Hint:** Use `plt.hist()`.

3 Add a User Choice: Even or Odd?

- Modify the program to let the user **choose** whether they want:
  - Even numbers **or** Odd numbers.
  - Squared **or** Cubed results.

Want More!?

Working with files

Reading Files

```
with open("example.txt", "r") as file:
    content = file.read()  # Reads the entire file
    print(content)
```

Other methods:

- `.readline()` → Reads one line at a time.
- `.readlines()` → Returns a list of lines.

Writing Files

```
with open("example.txt", "w") as file:
    file.write("Hello, world!\n This is a new line.")
```

Appending to a file:

```
with open("example.txt", "a") as file:
    file.write("\n Appending another line.")
```

Context Managers

The **`with`** statement is part of **context managers** in Python.

It's used to properly manage resources like **files, sockets, database connections,** and **locks**.

What is a Context Manager?

A **context manager** is an object that properly sets up and cleans up resources automatically using two special methods:

- `__enter__()` → Code before `with` runs (e.g., opening a file).
- `__exit__()` → Code after `with` runs (e.g., closing the file, handling errors).

Context Managers

Example: File Handling Without `with`

```
file = open("example.txt", "r")
content = file.read()
file.close()  # You must manually close the file
```

The issue? If an **exception** happens before `file.close()`, the file remains open and locked.

Using `with` (Automatic Cleanup)

```
with open("example.txt", "r") as file:
    content = file.read()  # File is open inside this block
# File is automatically closed after the block
```

✅ No need to manually call `file.close()`.
✅ Even if an error occurs, Python **ensures cleanup**.