📝 Notes

May 24, 2025

Functional programming and code organization in Python.

Invited agaley.fesh askpyfesh@gmail.com Paschal Amah

Attachments Functional programming and code organization in Python.

Meeting records Recording

Summary

Paschal Amah led a hands-on session on functional programming and code organization in Python, demonstrating how to structure code using functions by creating and manipulating a pandas DataFrame in a Jupyter Notebook. Paschal Amah illustrated the benefits of organizing code into small, specific functions and orchestrating them in a main function, using a serverless data service example to showcase real-world application. Tochukwu and Paschal Amah discussed handling schema changes in data pipelines, covering strategies for managing different data loads and comparing custom solutions to existing tools.

Details

- **Meeting Welcome and Topic** Paschal Amah welcomed everyone to the session, apologizing for a scheduling error. The main topic of the meeting was functional programming and code organization in Python. Paschal Amah mentioned that the session would be hands-on, using a shared PDF as the primary resource.
- **Functional Programming Basics** Paschal Amah explained that functional programming involves programming constructs where functions are treated as first-class citizens. This means functions can be used as variables, passed as arguments, stored in other functions, and call other functions. The focus is primarily on functions.

- **Setting Up the Environment** Paschal Amah demonstrated how to navigate to a folder and open a Jupyter Notebook, emphasizing the importance of starting from a clean and accessible space. They also briefly explained how to install Jupyter Notebook using pip. Paschal Amah showed a shortcut to open Jupyter Notebook directly from the desired folder in Windows.
- **Jupyter Notebook and Initial Steps** Paschal Amah started a new Python 3 notebook and renamed it "code organization". They explained their approach of initially creating variables and logic in the notebook before structuring them into functions to ensure things are working. Paschal Amah then imported the pandas library as pd.
- **Creating a Pandas DataFrame** Paschal Amah proceeded to create an empty pandas DataFrame and then added columns named 'names' and 'organizations' by passing Python lists. They highlighted that all columns in a pandas DataFrame must have the same length. Paschal Amah then demonstrated how to select data from the DataFrame using a condition similar to SQL's WHERE clause.
- **Exploring DataFrame Operations** Paschal Amah showed how passing a single string to a column assigns that string to all rows. They further demonstrated how to filter the DataFrame based on a condition. Paschal Amah then illustrated how to access and slice strings within a column, specifically taking the first three letters of each name.
- **Organizing Code into a Function** Paschal Amah then took the working code for creating the DataFrame and organized it into a Python function called `DFMaker`. This function encapsulated the steps of creating an empty DataFrame and populating it with data from predefined lists. Paschal Amah showed how to define a function using `def`, give it a name, and include the code block within it.
- **Calling and Using the Function** Paschal Amah demonstrated how to call the `DFMaker` function. They explained that calling a function without assigning its return value will show its memory address. Paschal Amah then showed how to store the returned DataFrame in a variable called `data` for further use.
- **Creating a Second Function for Processing** Paschal Amah created a second function, initially working outside a function to iterate through the DataFrame rows using `data.itertuples()`. They used an f-string to create a readable message for each row. After resolving an attribute error, Paschal Amah added conditional logic to customize the message based on the 'names' column.

- **Encapsulating Processing into a Function** Paschal Amah then turned the data processing logic into a function called `pass_around` (later shortened to `pass`), which accepts a DataFrame as an argument. This function iterates through the rows and prints a formatted message. Paschal Amah demonstrated calling this function and passing the output of `DFMaker` to it.
- **Principles of Functional Code Organization** Paschal Amah emphasized the principle of having each function perform a single, specific task. They suggested that well-organized code often involves multiple small functions for different functionalities, with a main function to orchestrate their execution. Paschal Amah illustrated a scenario where different team members could work on separate functions that pass data to each other.
- **Creating an Orchestrator Function** Paschal Amah created a final function called `do` to demonstrate the complete workflow. This function calls `DFMaker` to create the DataFrame and then passes the result to the `pass` function. The `do` function also returns a string indicating completion. Paschal Amah then called the `do` function to execute the entire process.
- **Benefits of Organized Code** Paschal Amah highlighted the advantages of this organized approach, such as the ability to modify specific parts of the code (e.g., the data source) without affecting other functions. They showed an example of updating the initial data and running the main function to see the changes reflected. Paschal Amah suggested that this structure is similar to working with separate modules in a larger project, like in VS Code.
- **Example of a Real-World Application** Paschal Amah then briefly showed the structure of a serverless data service as an example of functional programming in a larger context. They pointed out a YAML file defining the service and highlighted the `src` directory containing the code. Paschal Amah mentioned that this service uses a Lambda function with a handler as the main orchestrator, triggered by a cron job.
- **Code Structure in a Larger Project** Paschal Amah explained the typical structure of their services, including a `dealer.py` file containing various functional components and `utils.py` for configuration and variables. They emphasized that the example service does not use any classes, relying entirely on functions. Paschal Amah showed examples of functions for tasks like sending messages, retrieving secrets, and getting the last run time of a process.

- **Orchestration in the Real-World Example** Paschal Amah returned to the main handler function, showing how it imports and calls various functions from `dealer.py` and `utils.py` to perform data processing tasks. They illustrated how the handler iterates through a list of tables, calls functions to get the last successful run, and fetches incremental data. The importance of error handling and logging within the functions was also noted.
- **Schema Change Handling Discussion** Tochukwu asked about how schema changes are handled in the data pipeline. Paschal Amah explained their approach of ensuring the pipeline does not fail by having a clear understanding of the expected data and destination schema from the beginning. They described a process of checking incoming data against a defined schema in `utils.py`, adding missing columns as nulls, and logging unexpected new columns. This ensures data delivery according to the agreed-upon schema.
- **Closing and Questions** Paschal Amah concluded the presentation, offering to stay for a few more minutes for questions and mentioning that the Jupyter Notebook would be shared.
- **Schema Resolution** Paschal Amah described a method for schema resolution in data pipelines. The process involves checking incoming data against the destination schema, removing unexpected columns, and creating missing ones to prevent pipeline failures. Paschal Amah explained that this approach ensures the pipeline continues to function even as schemas evolve, requiring only the modification of the changed component.
- **Handling Full Loads vs. Incremental Loads** Tochukwu raised a concern about handling schema changes during full loads versus incremental loads. Tochukwu suggested that during a full load, setting a missing column to 'none' might erase existing data, necessitating informing the relevant parties. Paschal Amah responded by stating that full loads should ideally only occur at the beginning of a data pipeline to establish the initial schema. Paschal Amah further explained that downstream analytics rely on understanding the data's structure, making awareness of full load impacts crucial.
- **Alternative Schema Handling Approaches** Tochukwu mentioned using a set-based comparison between source and destination columns to identify and handle schema differences. Paschal Amah affirmed that this approach is similar to the schema evolution handling found in tools like Airbyte. Paschal Amah noted

that while various tools offer built-in solutions, developing a custom solution provides greater control, especially for manageable data volumes.

- **Meeting Conclusion and Next Steps** Paschal Amah concluded the session, stating they would save and share the discussed content as a notebook and PDF. Paschal Amah indicated that future topics would be shared with the group and suggested a bi-weekly meeting schedule. Paschal Amah encouraged participants to remember the possibilities discussed and to explore them independently when needed.

Suggested next steps

- [ ] Paschal Amah will share the notebook used in the meeting with Moshood Akeem olayinka, Tochukwu, and the group for further discussions if needed.
- [ ] Paschal Amah will share the video recording of the current session as a notebook and a PDF once it is ready, and will share the next topic in the group.