Python meeting
02: Melt, split, For: balances Group by for general ledger

22nd April 2026

The Python Meeting will start soon

COURSE PYTHON SCHEDULE

01: 20250415, 10am: Initiation – Case-study: P02_15: Above average prices

- Set-up: Python, Visual Studio Code
- Project creation: Folder, .py, Virtual Environment, CMD
- Libraries: Import, requirements file
- Naming conventions
- Basic objects: Dataframe (table), Series (column), Variable, Function, Group_by
- Basic functions: Import, Rename, Filter, Join, Create columns, Group, Aggregate, Export

02: 20250422, 9am: Melt, split, for – Case-studies: D01: Trial balance, P01_19: Supplier debtors, O01_19: Customer creditors

- Using melt, split
- For loop

03: 20250429, 9am: Web-download – Case-study: P01_10: Suppliers in OFAC

- Request for downloading sanctions
- IO for encoding
- Regex for comparison
- Split, explode for row generation
- Levenshtein distance scoring

04: 20250506, 9am: JSON, expressions, rolling Window – Case-study: P02_19: Split POs

- Import JSON, Expression object for adding labels,
- cum_sum() for rolling window

05: 20250513, 9AM: zip, dynamic fields – Case-study: P02_03: PO whilst blocked

- Zip to group lists together
- Using dynamic fields in a for loop

09:00 – 09:05

COURSE PYTHON SCHEDULE

06: 20250520, 10am: Cumulative sum, dates – Case-study: P02_18: PO slow rotation

- cum_sum for calculating future or past inventory movements

07: 20250527, 9am: Isolation forest for unusual transactions – Case-study: Unusual bank transfers

- Using isolation forest library to score for unusual transactions

08: 20250603: 9am: SpaCy – Case-study: P01_11/ O01_11: Suppliers/ customers that are people

- Using SpaCy NLP object to categorize names

09: 20250610: 9am: Contract review – Case-study: Scoring of contracts

- Using Gemini model to check for paragraphs of similar meaning

10: 20250617: 9am: Open AI API – Case-study: Generating audit reports

- Using API to OpenAI to generate text for audit reports

11: 20250624: 10am: Regex and Sklearn matching – Case-study: HR03_12/ H403_13: Unusual T&E

- Using Regex and Sklearn to check for unexpected travel and expenses

12: 20250701: 9am: OCR: Image recognition – Case-study: HR03_14: Expenses for others

- Using OCR python library to read travel and expense receipts

TODAY'S SCHEDULE

01

Case-study

→ Obtain the trial balance

02

Case-study

→ Obtain the general ledger with accounts as lists

03

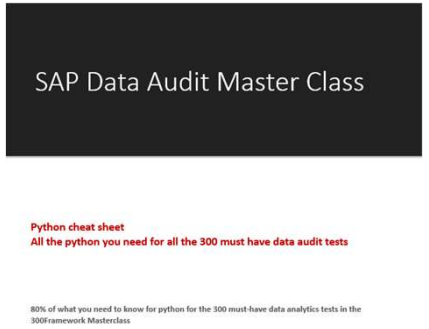
Python structures

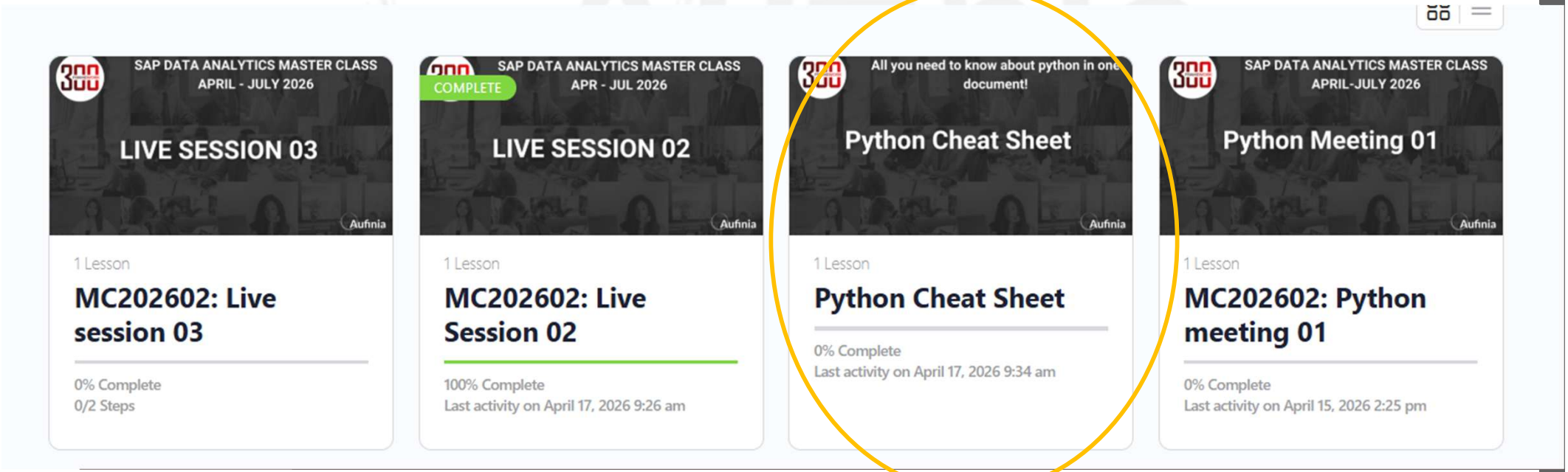
Visualize the structures we used

09:00 – 09:05

Tips: get the new Python Cheat Sheet

Get the new python cheat sheet.





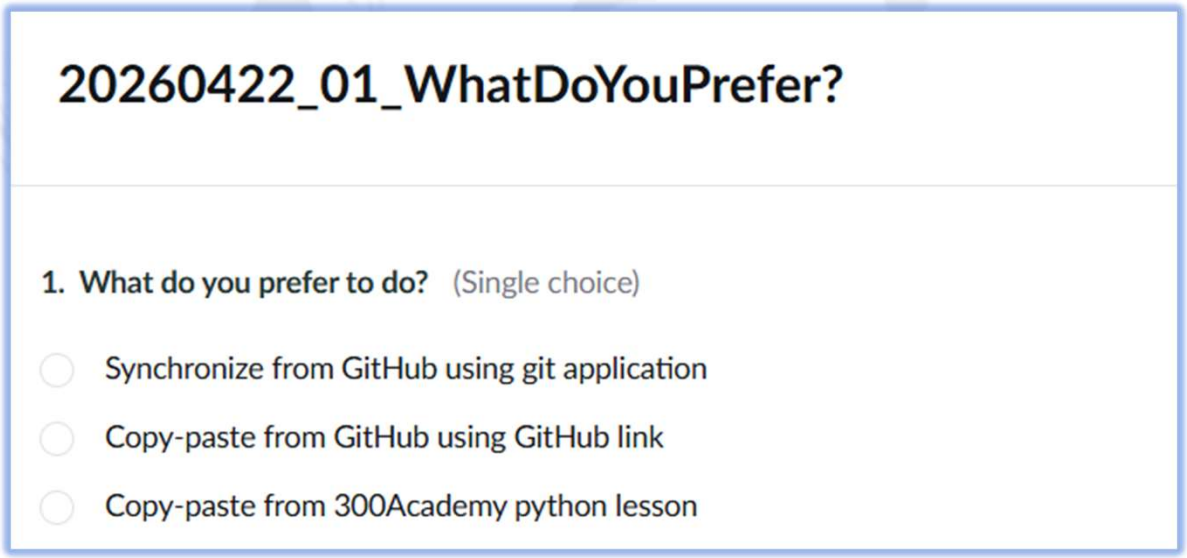
09:00 – 09:05

Obtain the code

Clone from gitHub

Copy paste from GitHub

Copy paste from 300Academy



09:05 – 09:15

Create and send me your GitHub name

Even if you do not use the git application on your PC, you can copy files from GitHub, if you have access to the repository.

1/ Create an account in GitHub

In case you don't have account, click here to create an account: **https://github.com/**

Owners of GitHub repositories will then be able to add you as members.

Note: You will need to do this step before being able to download repositories to your machine.

Create yourself a GitHub account and send me your account, so that I can add you as a member.

Steps to follow to be able to run the program

1. If you have git:
   - Clone the GitHub repository for today: https://github.com/300Academy/PYTHON_MEETING_02.git

2. If you do not have git:
   - Create the repositories as mentioned in the python cheat sheet
   - Copy the files from 300Academy or from GitHub
   - Put the files in the right place in the repositories: note you can copy paste from GitHub also
   - Make sure that you have the right file extension names
   - Create a .env and put the address of your AM_VARIABLES.txt in it
   - Update the variables in the AM_VARIABLES.txt file to your folder addresses

3. Go to the 02_PROGRAMMES folder
4. Type CMD in top of the Window
5. Type code .
6. Create the venv using the correct python version as mentioned in the .toml: we will use 3.14
   - py -3.14 -m venv venv
7. Activate the venv
   - .\venv\Scripts\Activate.ps1
8. Run the requirements
   - pip install -r requirements.txt
9. Run the program
10. Check the output in your results folder

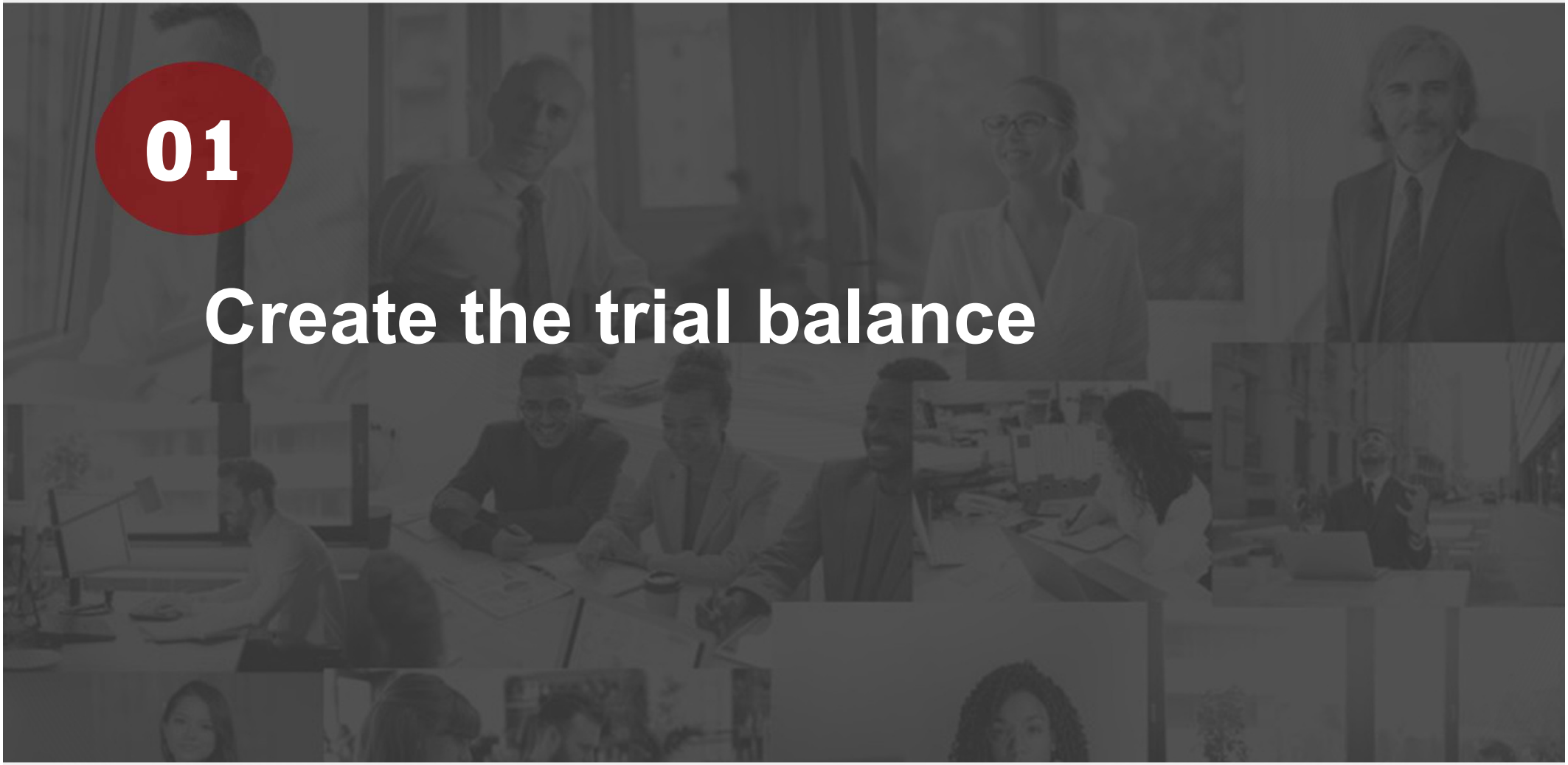
Did you do that already?

20260422_02_AreYouAbleToRunTheCode?

1. **Are you able to run the code?** (Single choice)

- ○ Yes
- ○ No - but I did not try yet
- ○ No - even though I tried - will let you know in the chat / book a meeting with Hanh

09:10 – 09:15



01

Create the trial balance

09:15 – 09:30

Recompute the trial balance
For: Repeat an action many times

Repeat an action many times – using variables

```
# 5.1/ Initialize opening balance
ZV_SE_EXP_NU_HSVLT_HSLXX = PI_POLARS.col('FAGLFLEXT_HSLVT')

# 5.2 / Generate PERIOD_01 to PERIOD_11:
for i in range(1, 12):
    ZV_ST_MM = f'{i:02}'
    ZV_SE_EXP_NU_HSLXX = PI_POLARS.col(f'FAGLFLEXT_HSL{ZV_ST_MM}')

    ZV_DF_FAGLFLEXT = (
        ZV_DF_FAGLFLEXT
        .with_columns(
            [
                PI_POLARS.concat_str(
                    [
                        ZV_SE_EXP_NU_HSVLT_HSLXX.cast(PI_POLARS.Utf8),
                        ZV_SE_EXP_NU_HSLXX.cast(PI_POLARS.Utf8),
                    ],
                    separator='|'
                )
                .alias(f'ZE_PERIOD_{ZV_ST_MM}')
```

09:15 – 09:30

Recompute the trial balance

Melt: convert a pivot table to a normal table

Melt: convert a pivot table to a normal table

```
# 5.5/ Convert pivot table to vertical table
ZV_LI_EXP_PERIOD_COLUMNS = [f'ZF_PERIOD_{i:02}' for i in range(1, 13)]

ZV_DF_FAGLFLEXT_MELTED = (
    ZV_DF_FAGLFLEXT
    .melt(
        id_vars=[
            'FAGLFLEXT_RBUKRS',
            'FAGLFLEXT_RYEAR',
            'FAGLFLEXT_RACCT',
            'FAGLFLEXT_DRCRK',
            'FAGLFLEXT_HSLVT'
        ],
        value_vars=ZV_LI_EXP_PERIOD_COLUMNS,
        variable_name='ZF_FAGLFLEXT_PERIOD',
        value_name='ZF_FAGLFLEXT_HSLXX_HSLVTXX'
    )
)
```

09:15 – 09:30

Recompute the trial balance

Split: Like Data-> Text to columns in Excel

Split: we use it to isolate certain information in a column

```
ZV_DF_FAGLFLEXT_MELTED
.with_columns(
    [
        PI_POLARS.col('ZF_FAGLFLEXT_HSLXX_HSLVTXX')
        .str.split_exact('|', 1)
        .struct.rename_fields(['ZF_FAGLFLEXT_HSVLTHSLXX_OPENING', 'ZF_FAGL
        .struct.field('ZF_FAGLFLEXT_HSVLTHSLXX_OPENING')
        .cast(PI_POLARS.Float64)
        .alias('ZF_FAGLFLEXT_HSVLTHSLXX_OPENING'),

        PI_POLARS.col('ZF_FAGLFLEXT_HSLXX_HSLVTXX')
        .str.split_exact('|', 1)
        .struct.rename_fields(['ZF_FAGLFLEXT_HSVLTHSLXX_OPENING', 'ZF_FAGL
        .struct.field('ZF_FAGLFLEXT_HSLXX')
        .cast(PI_POLARS.Float64)
        .alias('ZF_FAGLFLEXT_HSLXX'),

        PI_POLARS.col('ZF_FAGLFLEXT_PERIOD')
        .str.split('_')
```

09:15 – 09:30

02

General ledger – list of accounts

How do we make a list of accounts?

09:30 – 09:45

General ledger – list accounts

.agg() within a group_by()

If our data is not too big - we can do a simple join with .agg(): inside the group_by()

```
)
.agg(
    [
        (
            PI_POLARS.col('BSEG_HKONT')
            .filter(PI_POLARS.col('BSEG_SHKZG') == 'S')
            .sort_by(
                (
                    PI_POLARS.col('BSEG_DMBTR')
                    .filter(PI_POLARS.col('BSEG_SHKZG') == 'S')
                ),
                descending=True
            )
            .cast(PI_POLARS.Utf8)
            .implode()
            .list.join('_')
            .alias('ZF_BSEG_HKONT_DEBIT')
        ),
```

General ledger – list accounts
.agg() within a group_by()

Some problems we tend to encounter:
- Too many accounts / rows per journal entry
- Accounts that are repeated

The .implode() is giving us a list for the row that is inside the group that we are currently looking at.

We can limit this list. Group by with sum and then .slice()

```
# 1) Sum DMBTR per account per document (debit/credi
ZV_DF_BSEG_ACC_SUM = (
    ZV_DF_BSEG
    .group_by(
        [
            'BSEG_BUKRS',
            'BSEG_GJAHR',
            'BSEG_BELNR',
            'BSEG_SHKZG',
            'BSEG_HKONT',
        ]
    )
    .agg(
        [
            PI_POLARS.col('BSEG_DMBTR')
            .sum()
            .alias('ZF_BSEG_DMBTR_SUM')
        ]
    )
)
```

```
    .group_by(
        [
            'BSEG_BUKRS',
            'BSEG_GJAHR',
            'BSEG_BELNR',
        ],
        maintain_order=True
    )
    .agg(
        [
            (
                PI_POLARS.col('BSEG_HKONT')
                .filter(PI_POLARS.col('BSEG_SHKZG') == 'S')
                .cast(PI_POLARS.Utf8)
                .implode()
                .list.slice(0, 5)
                .list.join('_')
                .alias('ZF_BSEG_HKONT_DEBIT')
            ),
```

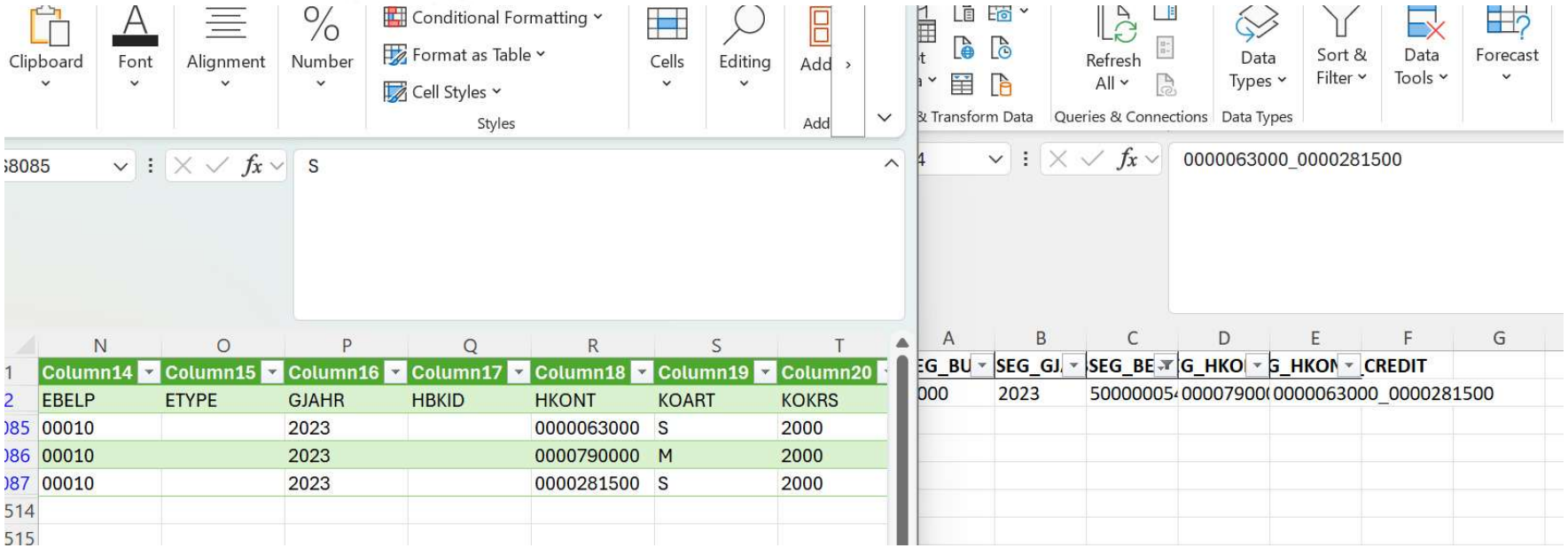
09:30 – 09:45

General ledger – list accounts

.agg() within a group_by()

We should check our results:

- Check we get the right accounts for the right journal entry
- Check the accounts are in the correct order



09:30 – 09:45


03
Python structures

Visualize the structures we used

Which structures did we use?

09:45 – 09:50

Python structures

A list:

[1, ‘2’,[‘a’,’b’,’c’], 1]

A list is a list of objects: can be different types

A set:

{1, ‘2’,[‘a’,’b’,’c’]}

A set is a list – but no duplicates allowed

An array:

[1, 2,3,4,5]

An array is like a list, but all objects of the same type

A tuple

(1, ‘2’,[‘a’,’b’,’c’], 1)

A tuple is like a list but it cannot be changed or indexed

A dictionary:

{
‘Suppliers’: (‘a’,’b’,’c’)
‘Number’: (1,2,3)
‘Value’:(100,150,300)
}

A dictionary has names that reference objects.

A dataframe:

A dataframe (~table) is like a dictionary, where the names are column names and the objects are arrays (have same type within them).

A series:

If we take a column out it is called a series

09:45 – 09:50

Python structures

(GBP, EUR)

(GBP, USD)

(GBP, ZAF)

Group keys

group_by object

TCURR_GDATU	TCURR_UKURS
20250101	0.61
...	...

TCURR_GDATU	TCURR_UKURS
20250101	0.93
...	...

TCURR_GDATU	TCURR_UKURS
20250101	20.01
20250102	20.03
20250103	20.02
...	...

Dataframes

Python structures

```
group_by([
        Company,
        Material,
        Month
])
```

3 Sub dataframes

Material	Month	Posting date	Input date	Input time	Document	Item	Quantity	Value
Bolts	Jan 2025	1 Jan 2025	1 Jan 2025	00:01:30	00001234	0001	10	200
Bolts	Jan 2025	2 Jan 2025	2 Jan 2025	00:05:20	00001235	0001	3	60
Bolts	Jan 2025	3 Jan 2025	3 Jan 2025	00:16:10	00001236	0001	2	400
Bolts	Jan 2025	25 Jan 2025	25 Jan 2025	00:01:30	00001237	0001	14	280
Bolts	Feb 2025	1 Feb 2025	1 Jan 2025	00:01:30	00001238	0001	5	1000
Bolts	Feb 2025	2 Feb 2025	1 Jan 2025	00:05:20	00001241	0001	6	1200
Bolts	Feb 2025	3 Feb 2025	1 Jan 2025	00:16:10	00001242	0001	17	3400
Bolts	Feb 2025	25 Feb 2025	1 Jan 2025	00:01:30	00001256	0001	2	400
Bolts	Mar 2025	1 Mar 2025	1 Mar 2025	00:01:30	00001281	0001	21	4200
Bolts	Mar 2025	28 Mar 2025	28 Mar 2025	00:05:20	00001290	0001	32	6400


Questions?