Python meeting
05: Purchase orders whilst blocked
ZIP, dynamic fields

13th May 2026

The Python Meeting will start soon

COURSE PYTHON SCHEDULE

01: 20250114, 9am: 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: 20250121,10am: Melt, split, for – Case-studies: D01: Trial balance, P01_19: Supplier debtors, O01_19: Customer creditors

- Using melt, split
- For loop

03: 20250128, 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: 20250204, 9am: JSON, expressions, rolling Window – Case-study: P02_19: Split POs

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

05: 20250211, 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: 20250218, 10am: Cumulative sum, dates – Case-study: P02_18: PO slow rotation

- cum_sum for calculating future or past inventory movements

07: 20250225, 9am: Isolation forest for unusual transactions – Case-study: Unusual bank transfers

- Using isolation forest library to score for unusual transactions

08: 20250304: 9am: SpaCy – Case-study: P01_11/ O01_11: Suppliers/ customers that are people

- Using SpaCy NLP object to categorize names

09: 20250311: 9am: Contract review – Case-study: Scoring of contracts

- Using Gemini model to check for paragraphs of similar meaning

10: 20250318: 10am: Open AI API – Case-study: Generating audit reports

- Using API to OpenAI to generate text for audit reports

11: 20250325: 9am: 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: 20250401: 9am: OCR: Image recognition – Case-study: HR03_14: Expenses for others

- Using OCR python library to read travel and expense receipts

09:00 – 09:05

TODAY'S SCHEDULE

01

Case-study

→ P02_03: Purchase orders whilst blocked.

02

Calling another python script

How to call another python script? (we will call a new one for CDPOS change documents)

03

Python structures

List, Zip, Group by… using for loops

09:00 – 09:05

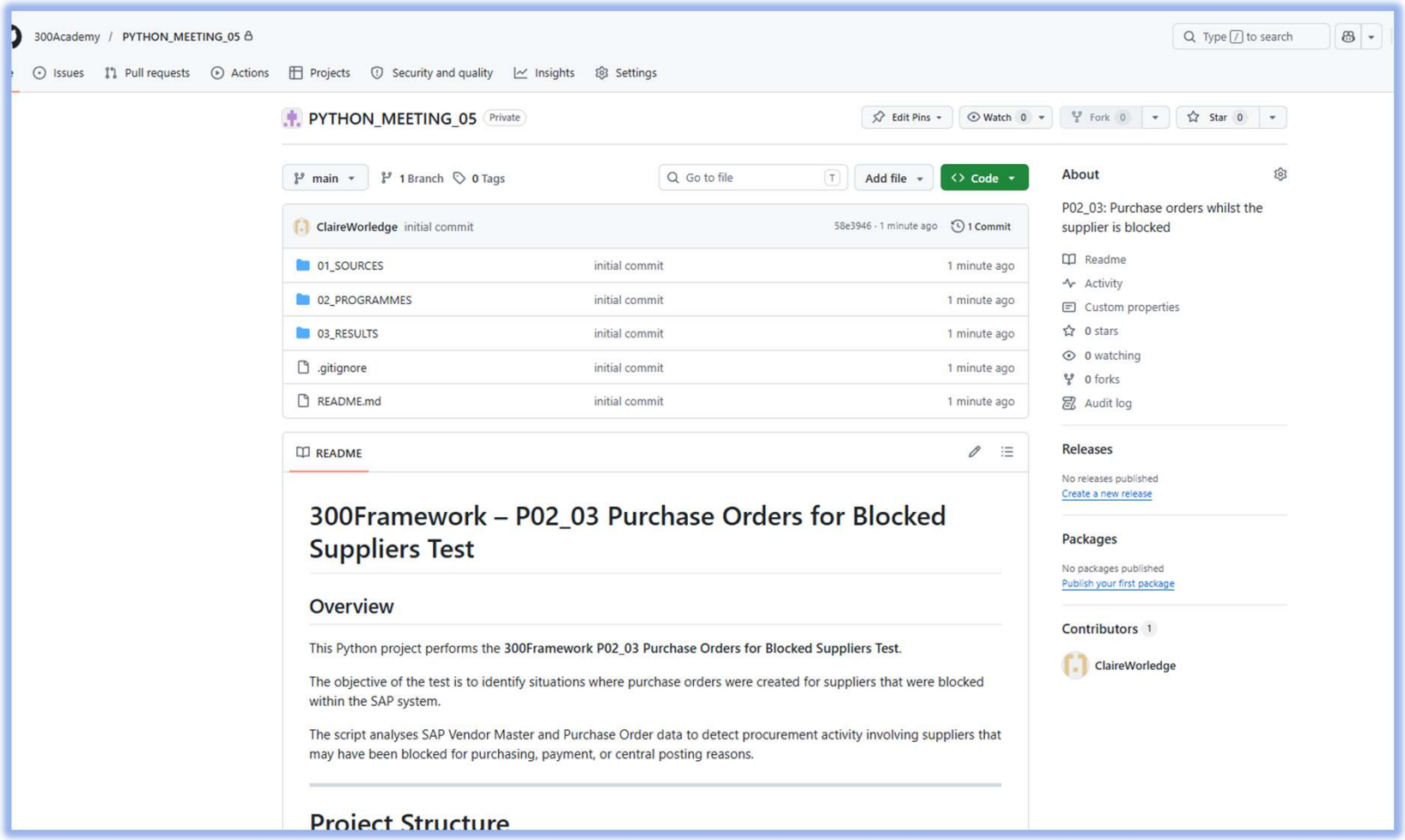
01

Case study: Split purchase orders

09:00 – 09:05

Before you get started.. New! README.md

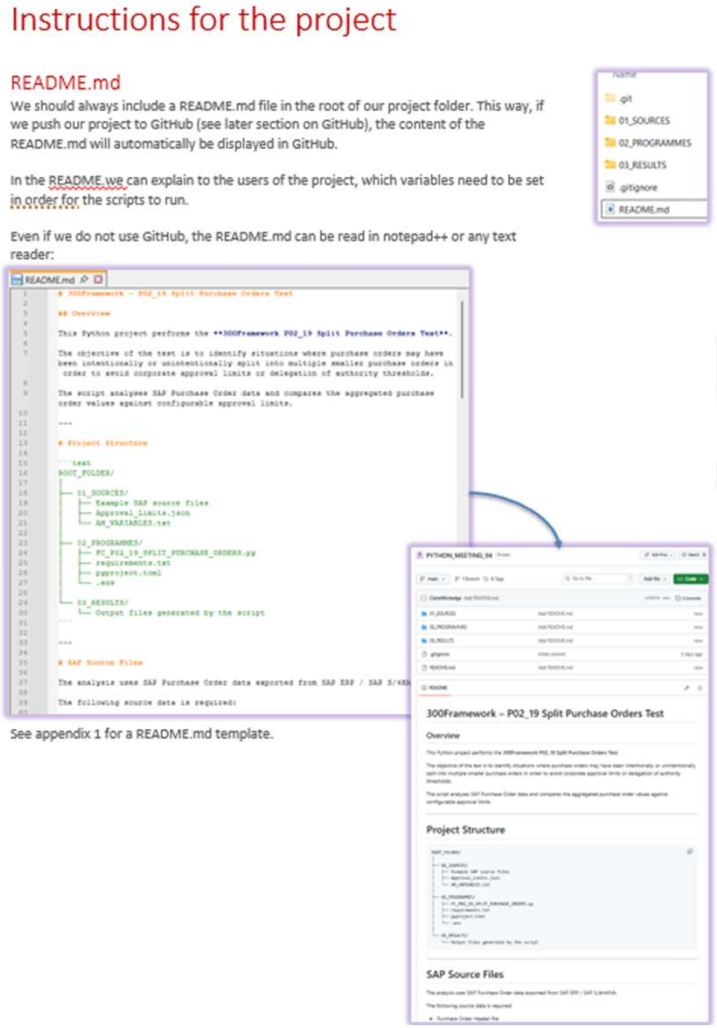
Check out the README.md



09:00 – 09:05

Before you get started.. New! README.md

Check out the updated Python Cheat Sheet:



Updates:

- README
- Artificial Intelligence
- Visual Studio Code Tips
- Some more python examples
- Some more examples of useful libraries

09:00 – 09:05

Did you do that already?

20260513_01_Were you able to run the code?

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

- ○ Yes!
- ○ Not yet!

09:10 – 09:15

Did you do that already?

20260513_02_WereYouAbleRunCodeLastWeek?

1. Were you able to run the code from last week? (Single choice)

- ○ Yes!
- ○ Not yet!

09:10 – 09:15

What should the result be?

The purchase order date EKKO_BEDAT should be found between the block dates.
Note: here we changed the variable for the extraction date for testing, so our last date is end of July. Also, in our test data the suppliers are blocked and never unblocked, but if they were unblocked we would show the closest unblock date.

AH	AV	AW	AX	AY	AZ	BA	BB	BC	BD	BE	BF	BG	BH
KKO_BEDA	EKKO_LIFNR	KKO_LOEK	KKO_STAT	KO_WAE	KKO_WKU	KKO_ZTER	_TABNAM	_LOEVM_S	_LOEVM_	_NODEL_S	1_NODEL_	_SPERR_S	1_SPERR_
20220530	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220530	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220530	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220530	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220530	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220530	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220601	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220601	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220601	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220601	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220601	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220601	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220601	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220601	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220602	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220602	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220602	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731
20220602	0000003000		9	USD	1.00000	ZB01	LFA1_SPERR					20220514	20220731

09:15 – 09:30

02

Calling another python script

Organizing scripts

09:30 – 09:45

Calling another python script

As last time, we have put some functions in a separate folder.
Notice here the squiggly lines. This means that python cannot find those scripts.

```
 9  # Script objective:
10  # Purchasing split orders to avoid approbation limits, using multiple low-value purchase orders.
11
12  from Z_SHARED_FUNCTIONS.FC_IMPORT import FC_IMPORT_TEXT
13  from Z_SHARED_FUNCTIONS.FC_EXPORT import FC_EXPORT_EXCEL
14  from Z_SHARED_FUNCTIONS.FC_JOIN_CDPOS_OLD_NEW import FC_JOIN_CDPOS_OLD_NEW
15
```

When I change the extension of the files to .py, then the lines disappear.
This is the same concept as the libraries that we download and install in our virtual environment.
In this way you can see that python libraries are basically, bits of python code that is done for you.

```
 9  # Script objective:
10  # Purchasing split orders to avoid approbation limits, using multiple low-value purchase orders.
11
12  from Z_SHARED_FUNCTIONS.FC_IMPORT import FC_IMPORT_TEXT
13  from Z_SHARED_FUNCTIONS.FC_EXPORT import FC_EXPORT_EXCEL
14  from Z_SHARED_FUNCTIONS.FC_JOIN_CDPOS_OLD_NEW import FC_JOIN_CDPOS_OLD_NEW
15
```

Passing information to our script

```
# 5/ Join change document old and new together
ZV_DF_CDPOSKRED_OLDNEW= FC_JOIN_CDPOS_OLD_NEW(ZV_DF_CDPOSKRED_OLD, ZV_DF_CDPOSKRED_NEW)
```

```
1
2  import polars as PI_POLARS
3
4  def FC_JOIN_CDPOS_OLD_NEW(ZVFCI_DF_NAME_OLD,ZVFCI_DF_NAME_NEW):
5
6
7      ZV_DF_OLD = (
8          ZVFCI_DF_NAME_OLD
9          .with_columns(
10             PI_POLARS.concat_str(
11                 [
12                     'CDPOS_OBJECTCLAS',
13                     'CDPOS_CHANGENR',
14                     'CDPOS_OBJECTID',
15                     'CDPOS_TABNAME',
16                     'CDPOS_FNAME',
17                     'CDPOS_CHNGIND'
18                 ],
19                 separator=''
20             )
21             .alias('ZF_KY_JN_OLD_NEW')
22         )
23     )
```

```
67         )
68
69     return ZV_DF_CDPOS_OLD_NEW
```

Here we pass two dataframes and we get back one dataframe.


03
Python structures

List

Here we create a list.

A list can be a list of any type of object in any order and also with duplication.

Here we have a list of strings. That means a list of words surrounded by quotation marks.

```
92      # 7/ Block flags - list
93      ZV_LI_LFA1LFB1LM1_BLOCK_FLAGS = [
94          'LFA1_LOEVM',
95          'LFA1_NODEL',
96          'LFA1_SPERR',
97          'LFA1_SPERM',
98          'LFA1_SPERQ',
99          'LFA1_SPERZ',
00          'LFB1_LOEVM',
01          'LFB1_NODEL',
02          'LFB1_SPERR',
03          'LFB1_ZAHLS',
04          'LFM1_SPERM',
05          'LFM1_LOEVM'
06      ]
07
```

For… each item in our list

For each item in our list, we will do a lot of tasks:

```
# 8/ Add start and end dates for each block/unblock flag: join start-end, filter
ZV_LI_DFS =[]

for ZV_ST_LFA1LFB1LM1_BLOCK_FLAG in ZV_LI_LFA1LFB1LM1_BLOCK_FLAGS:

    # Variables
    ZV_ST_LFA1LFB1LM1_BLOCK_FLAG_START = f'{ZV_ST_LFA1LFB1LM1_BLOCK_FLAG}_START'
    ZV_ST_LFA1LFB1LM1_BLOCK_FLAG_END = f'{ZV_ST_LFA1LFB1LM1_BLOCK_FLAG}_END'

    # Filter start, add field
    ZV_DF_CDPOSCDHDR_KRED_START = (
        ZV_DF_CDPOSCDHDR_KRED
        .filter(
            (
                PI_POLARS.col('ZF_CDPOS_TABNAME_FNAME')
                == ZV_ST_LFA1LFB1LM1_BLOCK_FLAG
            ) &
            (PI_POLARS.col('CDPOS_VALUE_OLD') == '') &
            (PI_POLARS.col('CDPOS_VALUE_NEW') == 'X')
        )
```

We do the following tasks inside the for loop (which is quite long):

- Create variables
- Filter the change documents on start and end
- Join the block start and block end change documents together
- If there is no end (unblock), then we replace the NULL with the date of the extraction
- Filter where the start date is prior the end date
- Get the earliest end date per supplier, block flag and start block date
- Add the resulting dataframe to a list of dataframes

09:45 – 09:50

Group by

We use a group by so that we can obtain the earliest end date per start date.

Meaning, for each block start, we want to know what was the block end. We want to know the first block end, in order to not confuse with another block event that might start later.

To do this, we organize our data into groups and then we get the earliest end date in the group.

Within a group, we can think of the records as mini dataframes, on which we can perform tasks, such as sort, and first.

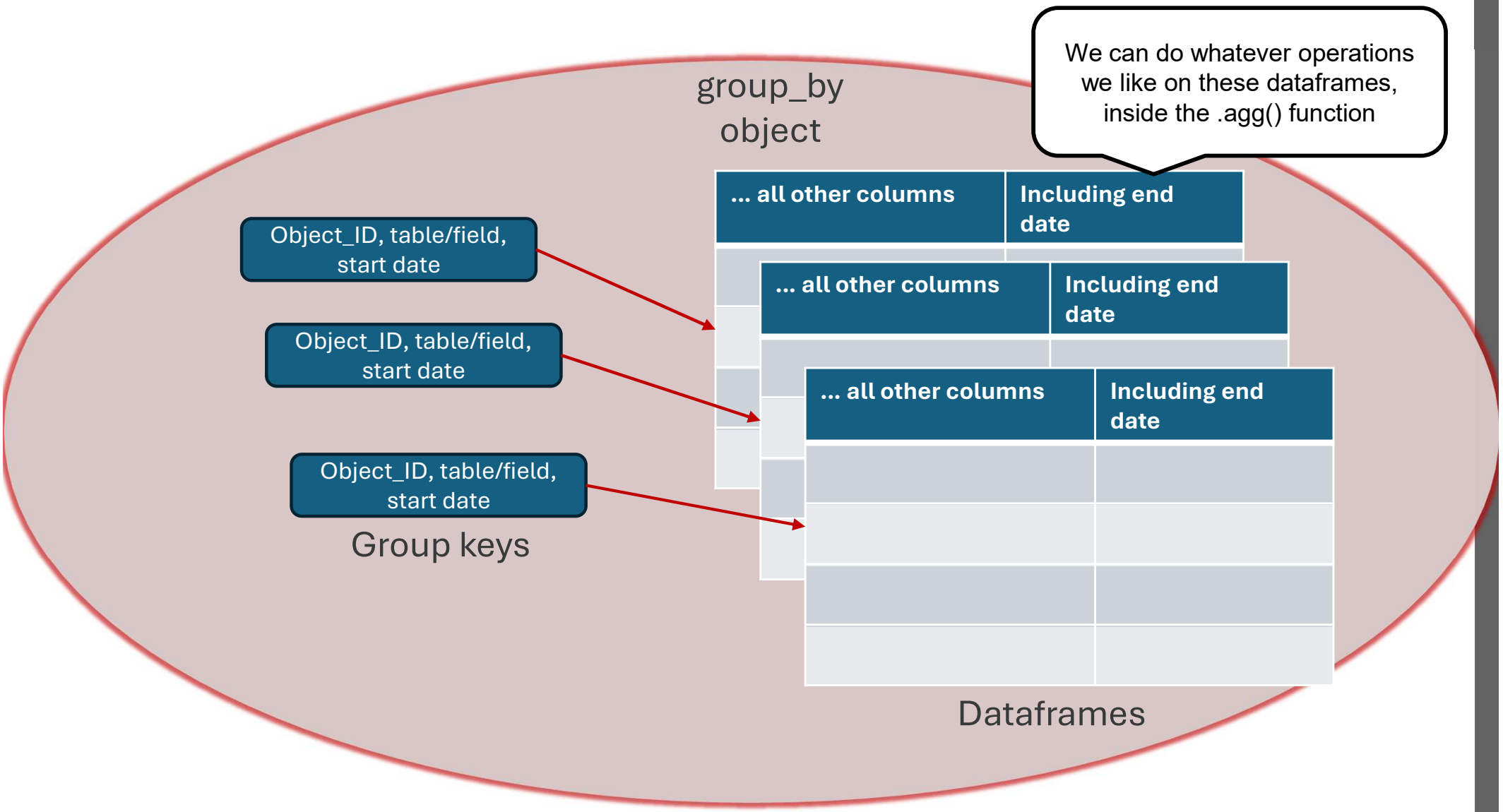
These are the group keys

```
# Group by for first end date - for each start block, find the first unblock date
ZV_DF_CDPOSCDHDR_KRED_START_END = (
    ZV_DF_CDPOSCDHDR_KRED_START_END
    .group_by(
        'CDPOS_OBJECTID',
        'ZF_CDPOS_TABNAME_FNAME',
        ZV_ST_LFA1LFB1LM1_BLOCK_FLAG_START
    )
    .agg(
        PI_POLARS.col(ZV_ST_LFA1LFB1LM1_BLOCK_FLAG_END)
        .sort()
        .first()
        .alias(ZV_ST_LFA1LFB1LM1_BLOCK_FLAG_END)
    )
)
```

We can do whatever operations we like, inside the .agg() function

09:45 – 09:50

Python structures



09:50 – 09:55

List of dataframes

At the end we make a list of dataframes.
We often use lists of dataframes.

- Each dataframe holds the result of one go of the for loop.
- When we have finished the for loop we concatenate all the dataframes into one.
- We use this method very often.

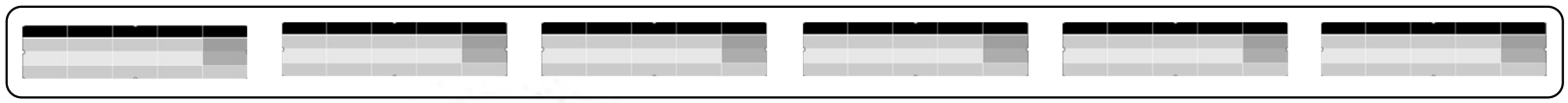
```
    # Add the dataframe to the list
    ZV_LI_DFS.append(ZV_DF_CDPOSCDHDR_KRED_START_END)

# 9/ Append all of the dataframes together to get start and end dates for each block field
ZV_DF_CDPOSCDHDR_KRED_START_END = PI_POLARS.concat(ZV_LI_DFS, how = 'diagonal_relaxed')
```

Diagonal relaxed enables us to concatenate, even if the dataframes have a different structure

09:45 – 09:50

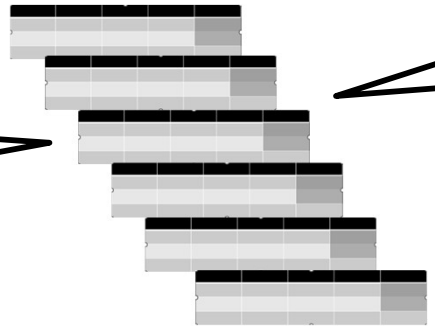
List of dataframes

A list of dataframes :



In our program, we created a list of dataframes, with one data frame for each block flag.

Then we concatenate into one dataframe:



We do diagonal because our dataframes do not have the same fields.

Where a field is found in another dataframe and not in this one, the value will be NUL, which means “didn´t match”

Zip: when you want to get related values from more than one list..typically dates

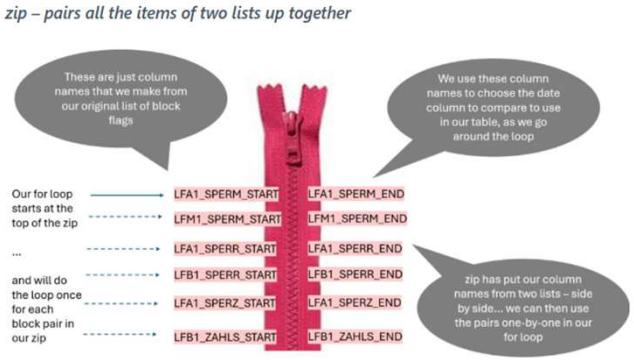
Here we use Zip.

We use zip quite often in 300Framework because it enables us to do a for loop on values from two different lists at the same time.

It is especially useful, when we have two lists of values that relate to each other.

We have a list of field names for each block flag start date and a list of field names for each block flag end date.

We typically use zip when we want to check if a transaction is between dates, where we can´t use the date field in the join key.




```
# 11/ Filter where purchase order date is between start and end date
ZV_LI_LFA1LFB1LM1_BLOCK_FLAGS_START = [f'{ZV_ST_FLAG}_START' for ZV_ST_FLAG in ZV_LI_LFA1LFB1LM1_BLOCK_FLAGS]
ZV_LI_LFA1LFB1LM1_BLOCK_FLAGS_END = [f'{ZV_ST_FLAG}_END' for ZV_ST_FLAG in ZV_LI_LFA1LFB1LM1_BLOCK_FLAGS]
ZV_LI_DF_EKPOEKKO_BETWEEN_START_END = []

for ZV_ST_DATE_BLOCKSTART, ZV_ST_DATE_BLOCKEND in zip(ZV_LI_LFA1LFB1LM1_BLOCK_FLAGS_START, ZV_LI_LFA1LFB1LM1_BLOCK_FLAGS_END):
    ZV_DF_EKPO_EKKO_BET_ST_END = (
        ZV_DF_EKPO_EKKO
        .filter(
            (
                PI_POLARS.col('EKKO_BEDAT')
                >= PI_POLARS.col(ZV_ST_DATE_BLOCKSTART)
            ) &
            (
                PI_POLARS.col('EKKO_BEDAT')
                <= PI_POLARS.col(ZV_ST_DATE_BLOCKEND)
            ) &
```

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


Questions?