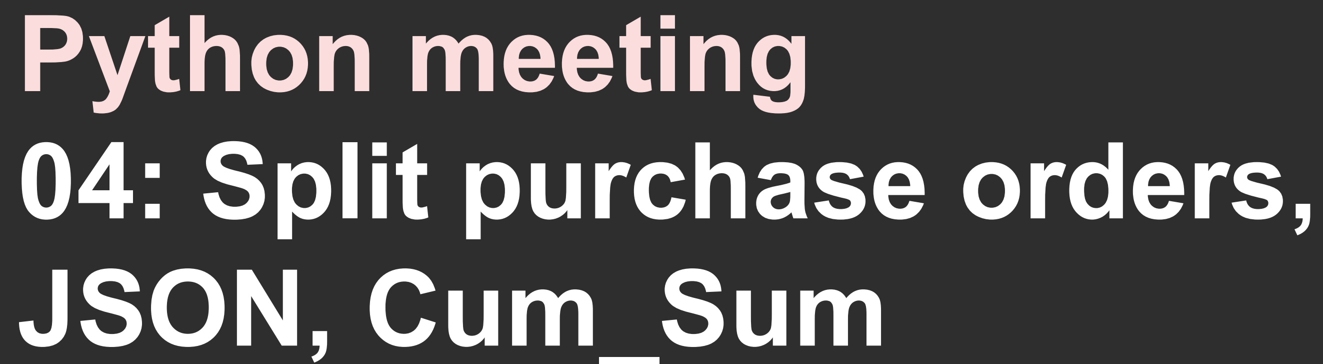


Python meeting
04: Split purchase orders, JSON, Cum_Sum

6th 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

→ P01_19: Split purchase orders

02

Calling another python script

How to call another python script?

03

Python structures

Dictionary, Group by, Expression object

Aufinia

09:00 – 09:05

Case study: Split purchase orders

09:00 – 09:05

Pre-requisites -> run programme

Pre-requisites:

1/ Set-up files

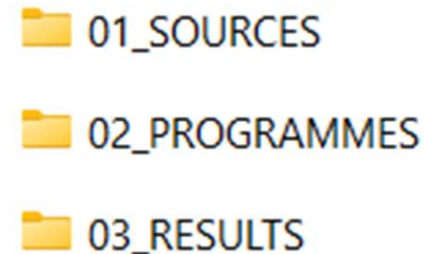
- Follow python cheat sheet for obtaining project from gitHub or organizing project structure as python cheat sheet
- Ensure that you have a Windows folder (we shall refer to it as ***root***) containing sub-folders:
- **/01_SOURCES**: should contain: 20251025_180621_1_EKPO.csv, 20240510_152615_1_EKKO.csv, A_APPROVAL_LIMITS.txt, AM_VARIABLES.txt
- **/02_PROGRAMMES**: should contain: **FC_P02_19_SPLIT_PO.py**, **pyproject.toml** and **requirements.txt**
- **/02_PROGRAMMES**: Create a **.env** file with the variable **ZV_ST_ROOT_FOLDER=C:\...\... (change to address of your root folder)**
- **/03_RESULTS**: This is where the results of the program will go

2/ Set-up environment:

- Navigate to the /02_PROGRAMMES folder and open Visual Studio Code there (**CMD** in Windows search bar, then **code .**)
- Terminal: (View-> Terminal or CTRL+SHIFT+`)
- Virtual environment: **py -3.14 -m venv venv**
- Activate virtual environment: **.\venv\Scripts\Activate.ps1**
- Run requirements: **pip install -r requirements.txt**

3/ Run

- From VSC opened at /02_PROGRAMMES, click on the script FC_P01_10.py and then execute.
- If the programme runs successfully, you will see a new Excel file in /03_RESULTS
- TROUBLESHOOTING: if the programme does not run: type in the terminal: .\venv\Scripts\python.exe <ScriptName.py> (without the <>)



09:00 – 09:05

Did you do that already?

20260204_01:Did you run the code?

1. 20260204: Did you already manage to run the code? (Single choice)

- ○ Yes
- ○ No

09:00 – 09:05

What should the result be?

	BD	BE	BF	BG	BH	BI	BJ	BK
1	MMATNRW	KPO_EBELI	ZF_EKKO_BEDAT_DT	_CASE_NU	ZF_COUNT	D_EBELN_C	O_BEDAT_DAYS_OUT	
2	-STU	2	2022-06-04 00:00:00	1	2	2	0	
3	-STU	2	2022-06-08 00:00:00	1	2	2	4	
4	00000001(	3	2023-04-28 00:00:00	2	3	3	324	
5	00000001(	3	2023-04-28 00:00:00	2	3	3	0	
6	00000001(	3	2023-04-28 00:00:00	2	3	3	0	
7	00000010(	12	2023-07-25 00:00:00	3	12	12	88	
8	00000010(	12	2023-07-27 00:00:00	3	12	12	2	

09:05 – 09:10

02

Calling another python script

Organizing scripts

09:10 – 09:15

Calling another python script

As last time, we have put some functions in a separate folder

```
> OUTLINE
v 02_PROGRAMMES
  > venv
  v Z_SHARED_FUNCTIONS
    > __pycache__
    FC_EXPORT.py
    FC_IMPORT.py
  FC_P02_19_SPLIT_PO.py
  requirements.txt
```

```
6   # Check the output
7
8
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
15  import polars as PI_POLARS
16  import sys as PI_SYS
17  import os as PI_OS
18  import json as PI_JSON
19  from datetime import datetime as PI_DATETIME
20  from pathlib import Path as PI_PATHLIBPATH
21
22  # 2/ Variables
23  ZV_ST_SOURCES_FOLDER = 'C:/2026_300FMASTERCLASS/05_PYTHON_MEETING_JAN_APRIL/04_PYTHON_MEETING_04/01_SOUR
24  ZV_04_C_PHASE_FOLDER = 'C:/2026_300FMASTERCLASS/05_PYTHON_MEETING_JAN_APRIL/04_PYTHON_MEETING_04/03_RESU
25  ZV_ST_NAME_EKPO_FILE = '20251025_180621_1_EKPO.csv' # fill in the name of your EKPO file
26  ZV_ST_NAME_EKKO_FILE = '20240510_152615_1_EKKO.csv' # fill in the name of your EKKO file
27  ZV_ST_NAME_JSON_FILE = 'A_APPROVAL_LIMITS.json' # fill in the name of your JSON file
28  ZV_NU_WINDOWDAYS = 7
```

09:10 – 09:15

Passing information to our script

We pass variables to our import script:

```
# 1/ Import purchase order detail
ZV_LI_NUM_COLS = [
    'EKPO_MENGE',
    'EKPO_NETPR',
    'EKPO_NETWR'
]
ZV_DF_EKPO = FC_IMPORT_TEXT(ZV_ST_NAME_EKPO_FILE, ZV_ST_SOURCES_FOLDER, '|','EKPO_',ZV_LI_NUM_COLS)

# 2/ Import purchase order header
ZV_LI_NUM_COLS = []
ZV_DF_EKKO = FC_IMPORT_TEXT(ZV_ST_NAME_EKKO_FILE, ZV_ST_SOURCES_FOLDER, '\t','EKKO_',ZV_LI_NUM_COLS)
```

Which returns a Dataframe.

Z_SHARED_FUNCTIONS > FC_IMPORT.py > FC_IMPORT_TEXT

```
7   def FC_IMPORT_TEXT(
33              ignore_errors=True,
34              has_header=True,
35              schema_overrides=ZV_DI_SCHEMA_OVERRIDES,
36              infer_schema_length=0,
37              null_values=[''],
38              quote_char=None
39          ).fill_null('')
40
41      for ZV_COL in ZV_DF.columns:
42          ZV_DF = ZV_DF.rename(
43              {
44                  ZV_COL: f'{ZVFCI_ST_PREFIX}{ZV_COL}'
45              }
46          )
47
48      for ZV_COL in ZVFCI_LI_NUM_COLS:
49          ZV_DF = ZV_DF.with_columns(
50              [
51                  PI_POLARS.col(f'{ZV_COL}')
52                  .cast(PI_POLARS.Float64, strict=False)
53                  .alias(f'{ZV_COL}')
54              ]
55          )
56
57      return ZV_DF
```

09:10 – 09:15

Python objects

Dictionary

Create a dictionary

```
with open(ZV_OB_PATH_APPROVAL_LIMITS, 'r', encoding='utf-8') as ZV_OB_FILE:
    ZV_DI_APPROVAL_LIMIT_DATA = PI_JSON.load(ZV_OB_FILE)
ZV_DI_APPROVAL_LIMIT = ZV_DI_APPROVAL_LIMIT_DATA["Approval_limits"]
```

Dictionary

In Python, a dictionary is written with {} brackets. A dictionary can contain any number of items. Each item can be of a different object class (dataframe, variable, list, dictionary, ...). Items in the dictionary can be accessed based on the key, which is the name of the item.

A dictionary is written as {"<keyName>": <object>}, where object can be of any type (text, numeric, list, dictionary,...). Objects can be of different types within the same dictionary. We typically use nested dictionaries.

```
ZV_DI_MY_DICTIONARY = {
    'key': {
        'key': {
            'key': myObject
        }
    }
}
```

09:15 – 09:20

Dictionary

JSON is like a dictionary structure in a text file:

```
{
  "Approval_limits": [
    {"level": "OptionalBelowL4", "threshold_USD": 30000, "operator": "<="},
    {"level": "L4", "threshold_USD": 200000, "operator": "<="},
    {"level": "L3", "threshold_USD": 600000, "operator": "<="},
    {"level": "L2", "threshold_USD": 4000000, "operator": "<="},
    {"level": "L1", "threshold_USD": 5000000, "operator": "<="},
    {"level": "CEO_or_CFO", "threshold_USD": 10000000, "operator": "<="},
    {"level": "CEO_and_CFO_jointly", "threshold_USD": 10000000, "operator": ">"}
  ]
}
```

A dictionary that points to a lits of dictionaries

09:15 – 09:20

List

We use lists all the time. They are useful for holding any type of object, with a positional reference.

List

In Python, a list is written with [] brackets. A list can contain any number of items. Each item can be of a different object class (dataframe, variable, list, dictionary). Items in the list can be accessed based on the index (position in the list). The first position in a list has the index 0.

A list is written as [<object>, <object>, <object>], where object can be of any type (text, numeric, list, dictionary, …). Objects can be of different types within the same list. The same object can occur more than once.

```
ZV_LI_MY_LIST = [object, object, object, object]
```

Each position in the list has an index, starting at 0:

- 1st object: index 0
- 2nd object: index 1
- 3rd object: index 2
- 4th object: index 3

09:15 – 09:20

Custom object: expression

Create an expression object

```
# 5.2/ Dictionary -> expression object
ZV_OB_EXPR_BUCKET = None
for ZV_DI_LIMIT in ZV_DI_APPROVAL_LIMIT:
    ZV_ST_LEVEL = ZV_DI_LIMIT["level"]
    ZV_NU_THRESHOLD = ZV_DI_LIMIT["threshold_USD"]
    ZV_ST_OPERATOR = ZV_DI_LIMIT["operator"]

    if ZV_ST_OPERATOR == "<=":
        ZV_BO_CONDITION = PI_POLARS.col('EKPO_NETWR') <= ZV_NU_THRESHOLD
    elif ZV_ST_OPERATOR == ">":
        ZV_BO_CONDITION = PI_POLARS.col('EKPO_NETWR') > ZV_NU_THRESHOLD
    else:
        continue

    if ZV_OB_EXPR_BUCKET is None:
        ZV_OB_EXPR_BUCKET = PI_POLARS.when(ZV_BO_CONDITION).then(PI_POLARS.lit(ZV_ST_LEVEL))
    else:
        ZV_OB_EXPR_BUCKET = ZV_OB_EXPR_BUCKET.when(ZV_BO_CONDITION).then(PI_POLARS.lit(ZV_ST_LEVEL))

ZV_OB_EXPR_BUCKET = ZV_OB_EXPR_BUCKET.otherwise(PI_POLARS.lit('OutOfRange'))
```

09:20 – 09:30

Custom object: expression

An expression object can be used to create a field

```
# 5.4/ Apply expression object
ZV_DF_EKPO_EKKO_GROUPBY = (
    ZV_DF_EKPO_EKKO_GROUPBY
    .with_columns(
        [
            ZV_OB_EXPR_BUCKET
            .alias('ZF_ST_APPROVAL_BUCKET')
        ]
    )
)
```

09:30 – 09:40

Group by

Group by

```
# 5.3/ Group by
ZV_DF_EKPO_EKKO_GROUPBY = (
    ZV_DF_EKPO_EKKO
    .group_by('EKPO_EBELN')
    .agg([
        PI_POLARS.col('EKPO_NETWR')
        .sum()
        .alias('EKPO_NETWR')
    ])
)
```

09:40 – 09:45

Group by

Group_by()

When we use group_by(), we obtain a group_by() object. A group_by() object is a dataframe that has been divided up into sub-dataframes.

```
ZV_OB_GROUPBY = (
    ZV_DF
    .group_by(
        [
          'Material',
          'Month',
        ]
    )
)
```

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

As soon as we apply a method on the group_by() object, it is transformed back into a dataframe. Typically, we apply the method agg() to a group_by() object and use other methods on columns inside agg():

```
    .group_by(…)
    .agg(
      [
       PI_POLARS.col('..').sum().alias(''),
       PI_POLARS.col('..').mean().alias(''),
       PI_POLARS.col('..').first().alias(''),
       PI_POLARS.col('..').n_unique().alias('')
      ]
    )
```

In the above code, you can see that we can use the Polars col() function to bring up a column and then we can apply chained methods on that column. This gives us a lot of flexibility.

09:40 – 09:45

Rolling window

Use .cum_sum() to create a rolling window:

True is also equal to 1.

When either condition is True, then the overall result will be 1

This will increment cum_sum()... so purchase orders that are more than ZV_NU_WINDOWDAYS from lat purchase order or that are for a different supplier / plant / material, will get a new case number.

```
# 7.2/ Rolling window
ZV_DF_EKPO_EKKO = (
    ZV_DF_EKPO_EKKO
    .sort(
        [
            'ZF_LIFNRERNAMMATNRWERKSAPPBUCK',
            'ZF_EKKO_BEDAT_DT'
        ]
    )
    .with_columns(
        [
            (
                (
                    (
                        (
                            (
                                PI_POLARS.col('ZF_EKKO_BEDAT_DT').diff() /
                                PI_POLARS.duration(days=1)
                            )
                            .fill_null(0)
                            .abs()
                        ) >= int(ZV_NU_WINDOWDAYS)
                    ) |
                    (
                        PI_POLARS.col('ZF_LIFNRERNAMMATNRWERKSAPPBUCK')  !=
                        PI_POLARS.col('ZF_LIFNRERNAMMATNRWERKSAPPBUCK').shift(1)
                    )
                )
                .cum_sum()
            )
            .alias('ZF_NU_CASE_NUMBER')

        ]
    )
)
```

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)

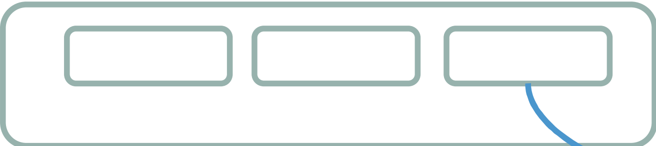
A tuple is like a list but it cannot be changed or indexed

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?