Python level 1

Introductory course
Class 3

Fábio Neves & Luis Dias

September 16

NATIXIS

PEA
Porto
Executive
Academy

AGENDA

Part 3 – Introduction to python

Dataframes

a. Create DataFrame and fill
b. Access to a DataFrame
c. Change rows and column names
d. Change DataFrame's values
e. Add rows and columns
f. Remove rows and columns
g. Remove rows and columns with NA-values
h. Read and import data
i. Get information about the dataset
j. Sort
k. Basic statistics (max, min, mean, median, mode, percentiles)
l. Filter DataFrames with conditions
m. Export data

Main libraries

- **Numpy:** stands for "Numerical Python". It is the most commonly used library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Array Interface is one of the key features of this library;

Focus for today

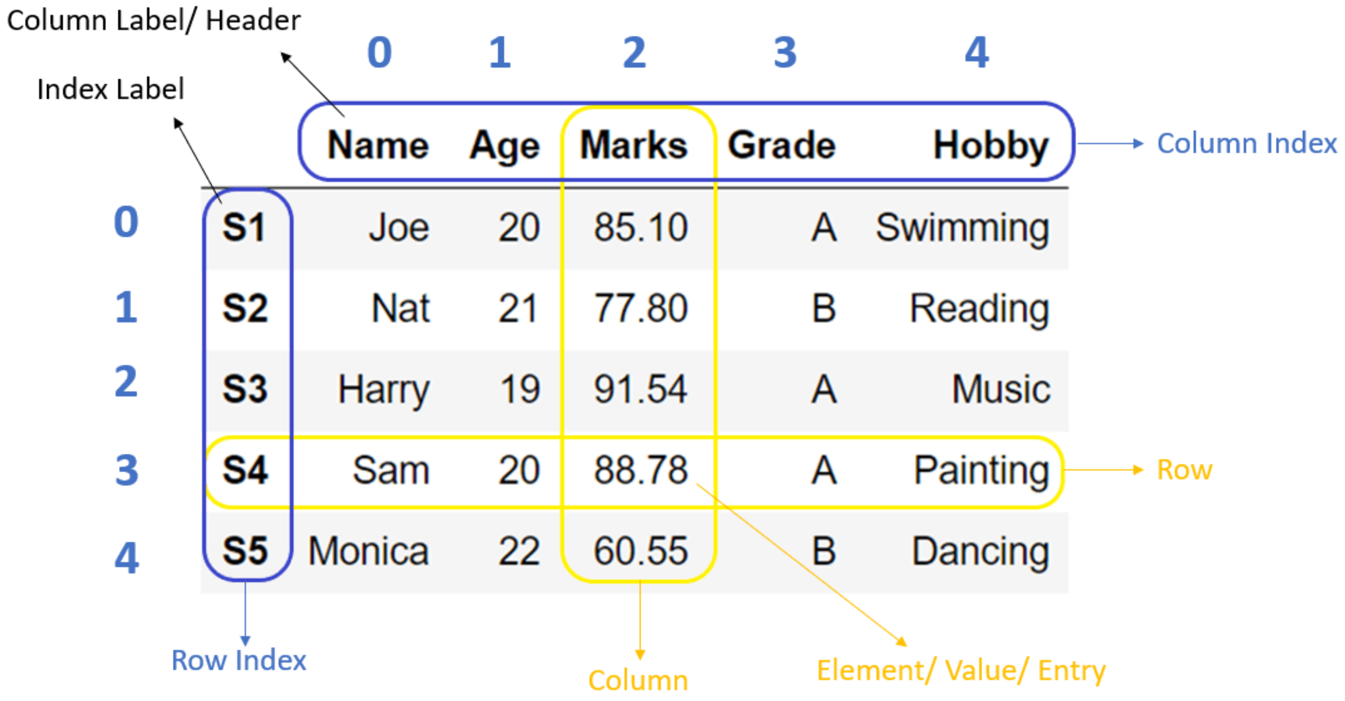
- **Pandas:** pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.;

- **Scipy:** the name "SciPy" stands for "Scientific Python". It is an open-source library used for high-level scientific computations;

- **Matplotlib:** responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library that plots high-defined figures like pie charts, histograms, scatterplots, and graphs.

DataFrames

What are DataFrames?



		0	1	2	3	4
		Name	Age	Marks	Grade	Hobby
0	S1	Joe	20	85.10	A	Swimming
1	S2	Nat	21	77.80	B	Reading
2	S3	Harry	19	91.54	A	Music
3	S4	Sam	20	88.78	A	Painting
4	S5	Monica	22	60.55	B	Dancing

A DataFrame is similar to a matrix/table, but the columns have names and can contain data of different types (string and integers, for example). A DataFrame can be seen as a table, where each line corresponds to a table record. Each column corresponds to the properties to be maintained for each record in the table. **They are used to store databases and are very useful for manipulating and analysing data**.

To work with DataFrames in python, it is necessary to install the library pandas. It has functions for analysing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis"

The first step is to install pandas library:

pip install pandas

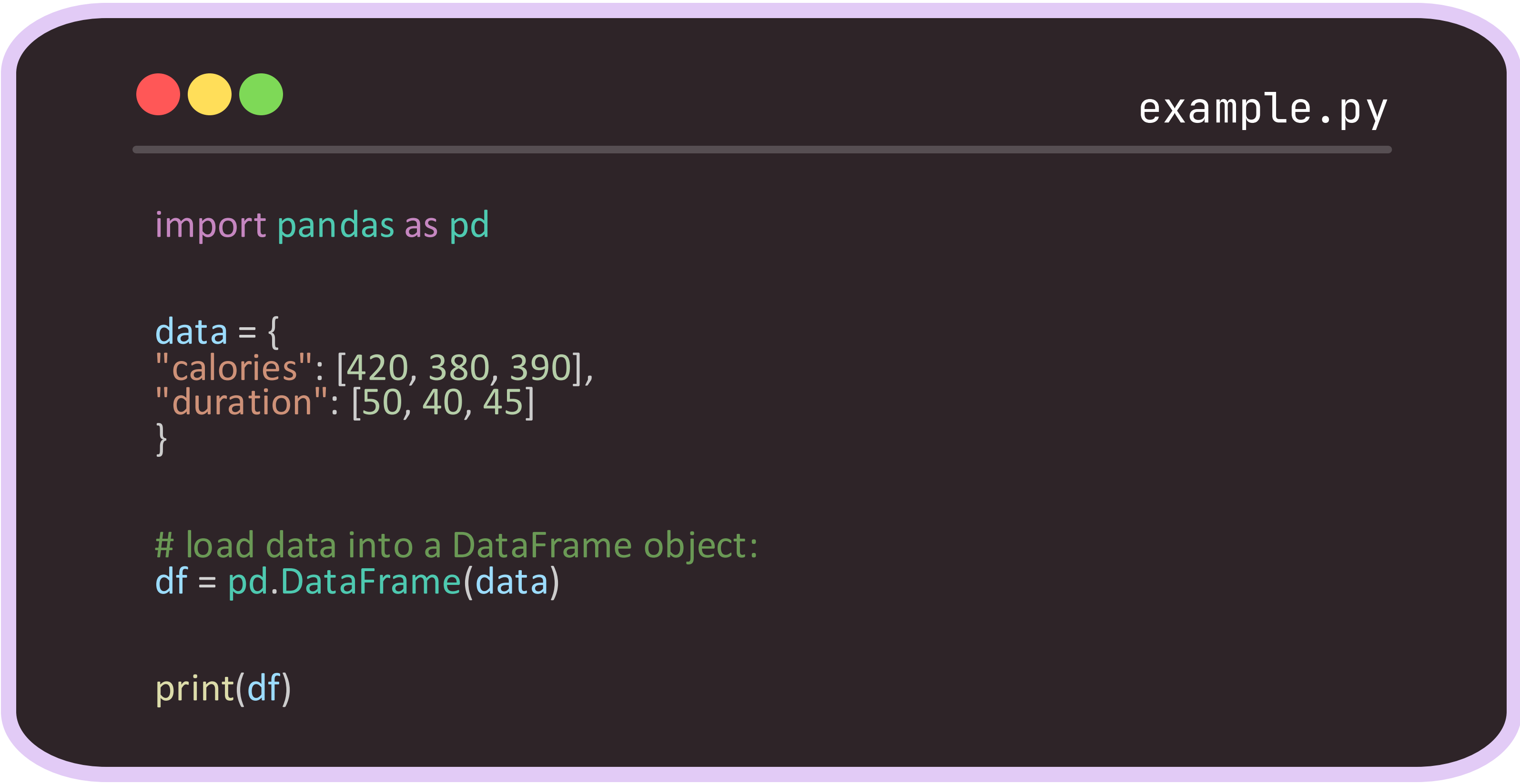
Every time we want to work with DataFrames, the script in which we are coding must include in its first lines:

import pandas

or

import pandas as pd

Create DataFrames



```python
import pandas as pd

data = {
"calories": [420, 380, 390],
"duration": [50, 40, 45]
}

# load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Result:

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Access to DataFrames

iloc vs loc

Select the columns/rows using the number of their positions/indexes

Select the columns/rows using their names/labels

Example

Column names

Row indexes

	calories	duration
0	420	50
1	380	40
2	390	45

In this DataFrame, the rows don't have names/labels. Their indexes are also their names. In turn, the columns have names – calories and duration- and have indexes – 0 and 1

Access to DataFrames

Select the columns/rows using the number of their positions/indexes

iloc vs loc

Select the columns/rows using their names/labels

Access to an entire row

	calories	duration
0	420	50
1	380	40
2	390	45

Before the comma , we select the rows

After the comma, we select the columns

With loc: df.loc[0, :]

With iloc: df.iloc[0, :]

: is to select all columns or rows

In this case, the way to access is the same because the names of the rows are the same as the indexes.

Access to DataFrames

Select the columns/rows using the number of their positions/indexes

iloc vs loc

Select the columns/rows using their names/labels

Access to an entire column

	calories	duration
0	420	50
1	380	40
2	390	45

With loc: df.loc[:, 'duration']

With iloc: df.iloc[:, 1]

Access to DataFrames

Select the columns/rows using the number of their positions/indexes

iloc vs loc

Select the columns/rows using their names/labels

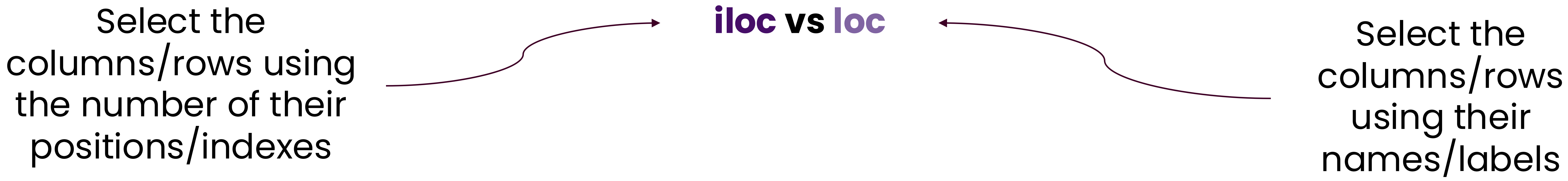
Access to an element

	calories	duration
0	420	50
1	380	40
2	390	45

With loc: df.loc[1 , 'calories']

With iloc: df.iloc[1 , 0]

Modify elements of a DataFrame



Select the columns/rows using the number of their positions/indexes

iloc vs loc

Select the columns/rows using their names/labels

Modify the value of an element

	calories	duration
0	420	50
1	380	40
2	390	60

With loc: df.loc[2 , ‘duration’] = 60

With iloc: df.iloc[2 , 1] = 60

Modify elements of a DataFrame

Select the columns/rows using the number of their positions/indexes

iloc vs loc

Select the columns/rows using their names/labels

Modify an entire row

	calories	duration
0	1000	2000
1	380	40
2	390	60

With loc: df.loc[0 , :] = 1000, 2000

With iloc: df.iloc[0 , :] = 1000, 2000

Modify elements of a DataFrame

Select the columns/rows using the number of their positions/indexes

iloc vs loc

Select the columns/rows using their names/labels

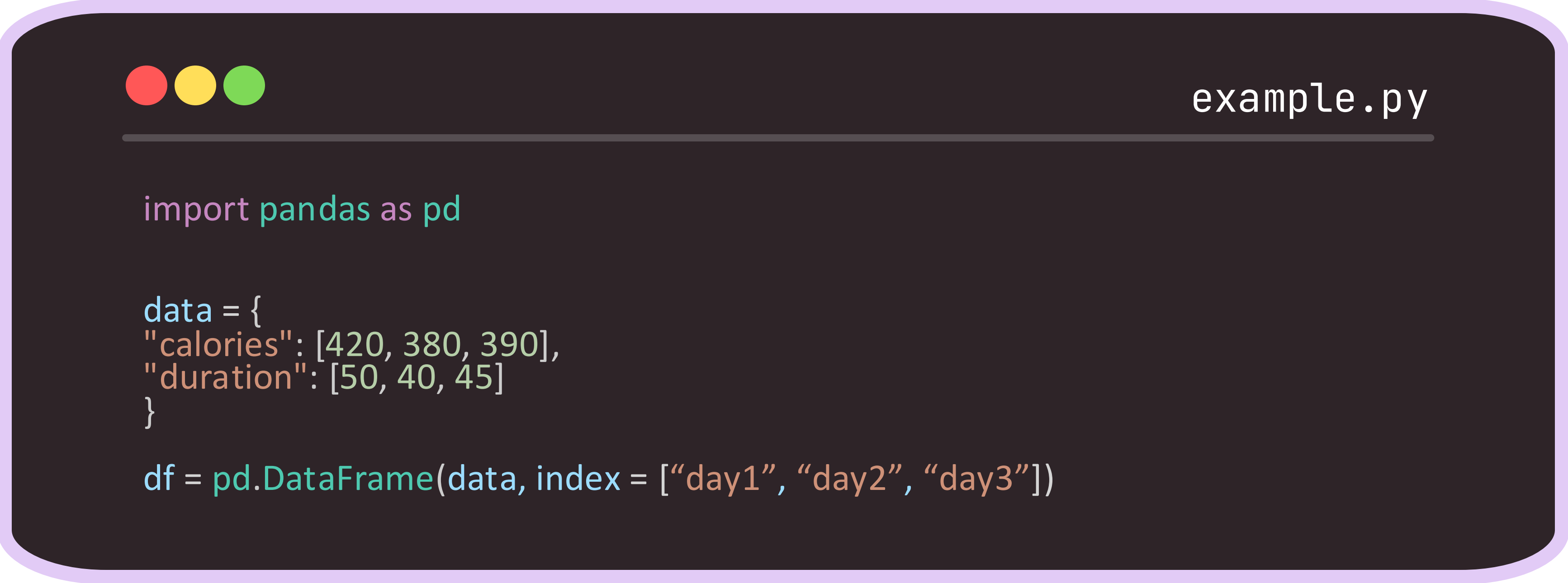
Modify an entire column

	calories	duration
0	-1	2000
1	-2	40
2	-4	60

With loc: df.loc[:, 'calories'] = -1, -2, -4

With iloc: df.iloc[:, 0] = -1, -2, -4

Name indexes of rows



```python
import pandas as pd


data = {
"calories": [420, 380, 390],
"duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = [“day1”, “day2”, “day3”])
```

Access to an entire row

	calories	duration
day1	420	50
day2	380	40
day3	390	45

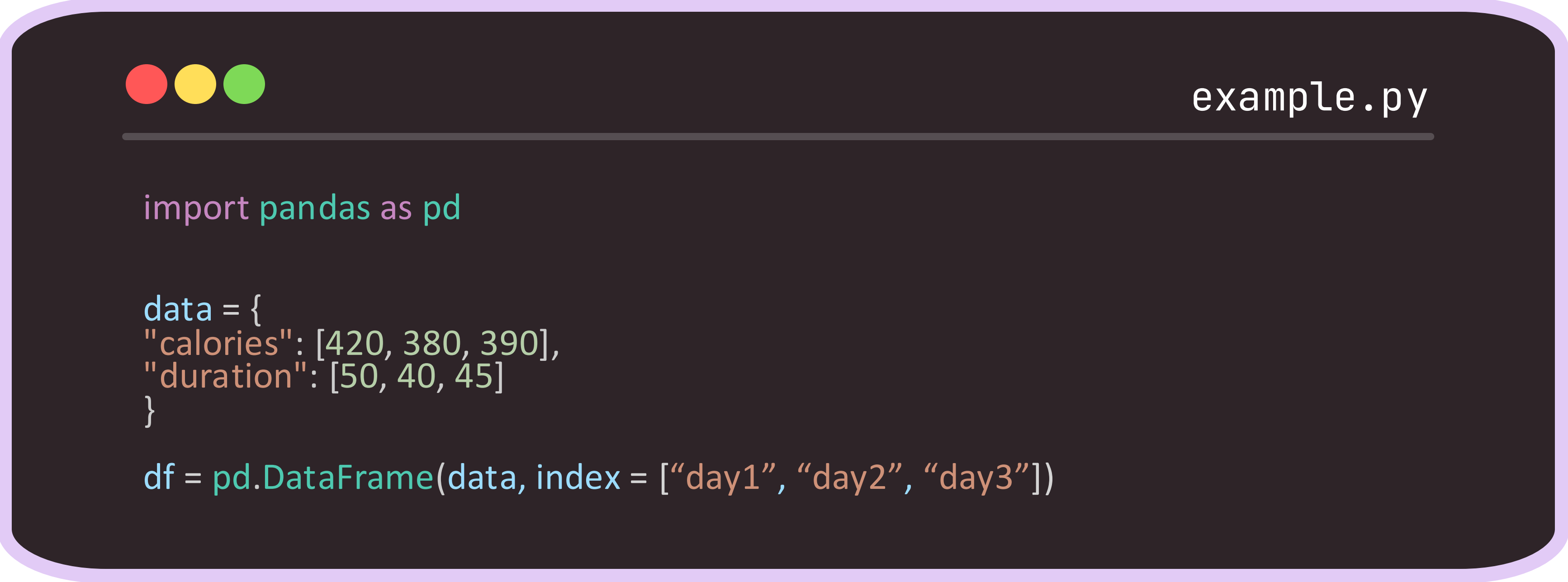
With loc: df.loc[‘day1’, :]

With iloc: df.iloc[0, :]

: is to select all columns or rows

Now, as the rows have names, loc and iloc will differ.

Add rows and columns



example.py

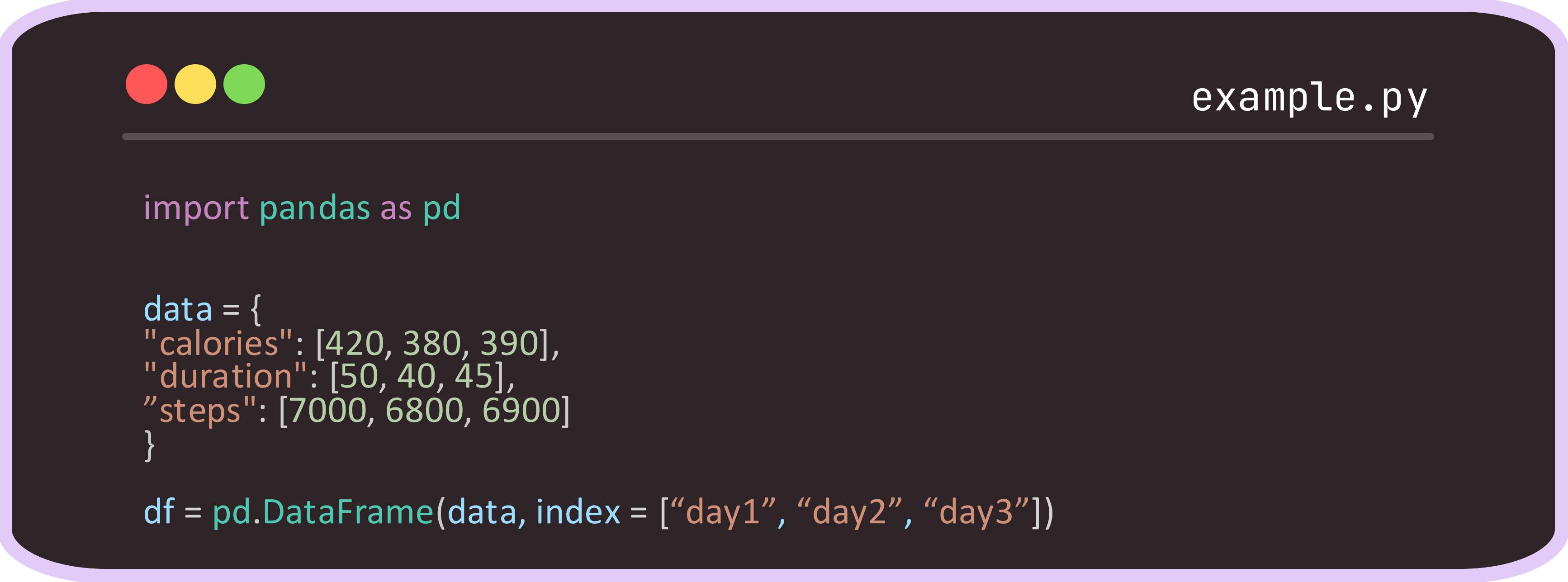
```
import pandas as pd


data = {
"calories": [420, 380, 390],
"duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = [“day1”, “day2”, “day3”])
```

Add an entire column

	calories	duration	steps
day1	420	50	7000
day2	380	40	6800
day3	390	45	6900

```
values = [7000, 6800, 6900]
df['steps'] = values
```

Add rows and columns



example.py

```
import pandas as pd


data = {
"calories": [420, 380, 390],
"duration": [50, 40, 45],
”steps": [7000, 6800, 6900]
}

df = pd.DataFrame(data, index = [“day1”, “day2”, “day3”])
```

Add an entire row

	calories	duration	steps
day1	420	50	7000
day2	380	40	6800
day3	390	45	6900
day4	450	60	7500

```
df.loc['day4'] = [450, 60, 7500]

#or
new_row = pd.DataFrame([[450, 60, 7500]],
columns=df.columns)

df = pd.concat([df, new_row], ignore_index=True)
```

Remove rows and columns

Remove an entire row

- The **drop()** method removes the specified row or column.
- By specifying the column axis (**axis =‘columns**’) or (**axis=1**), the **drop()** method removes the specified column.

	calories	duration	steps
day1	420	50	7000
day2	380	40	6800
day3	390	45	6900
day4	450	60	7500

```
df = df.drop('steps', axis=1)

#or

df = df.drop('steps', axis='columns')
```

Remove rows and columns

Remove an entire row

- The **drop()** method removes the specified row or column.
- By specifying the column axis (**axis =‘index**’) or (**axis=0**), the **drop()** method removes the specified row.

	calories	duration
day1	420	50
day2	380	40
day3	390	45
day4	450	60

```
df = df.drop(‘day4', axis=0)

#or

df = df.drop(‘day4', axis=‘index')
```

Remove rows and columns

Remove an entire row

- The **dropna()** method removes rows or columns that contain missing values.
- By specifying the column axis (**axis =0**) or (**axis=1**), the define if we want to remove the rows or the columns with missing values.

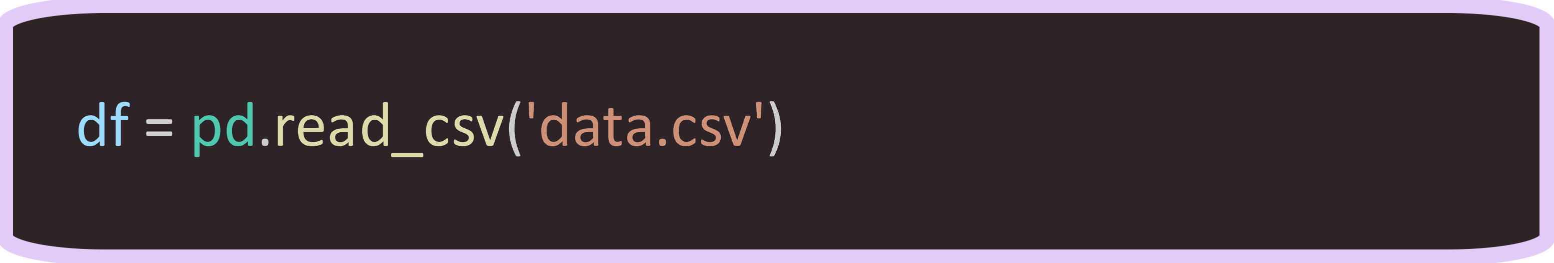
	calories	duration
day1	420	50
day2	380	Nan
day3	390	45

```
df = df.dropna(axis=0)


df = df.dropna(axis=1)
```

Read and import data

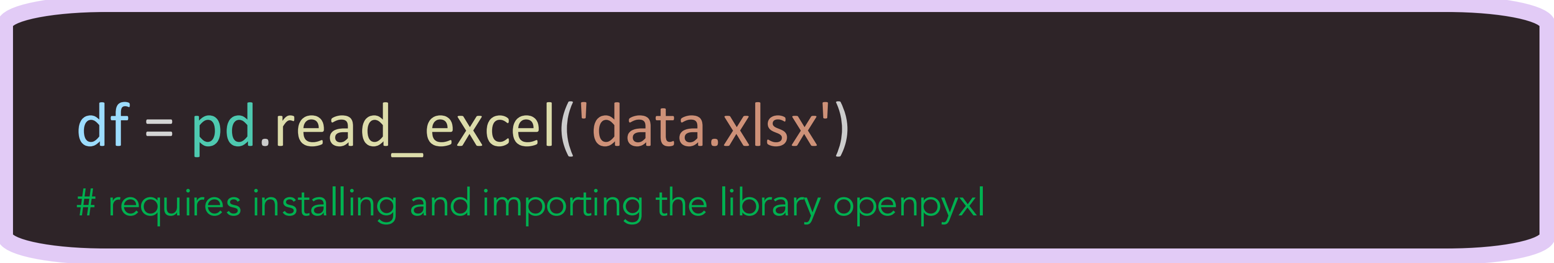
If your data sets are stored in a file, Pandas can load them into a DataFrame

- Data sets stored in **csv**



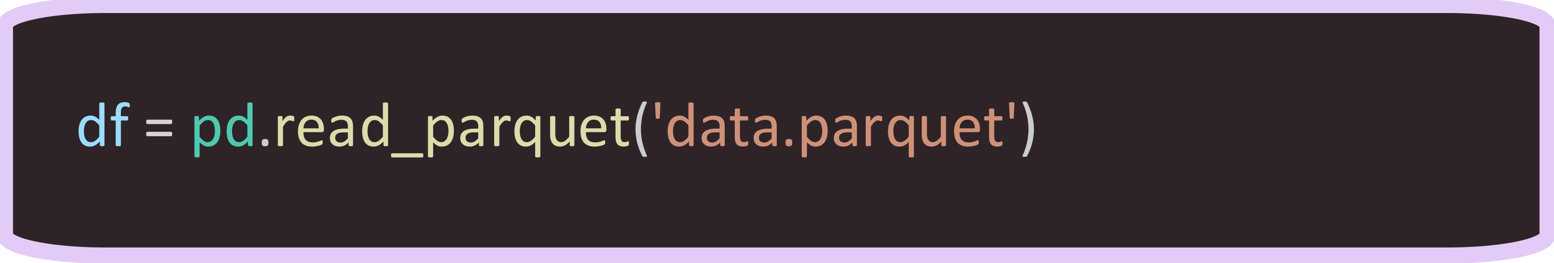
```
df = pd.read_csv('data.csv')
```

- Data sets stored in **xlsx**



```
df = pd.read_excel('data.xlsx')
# requires installing and importing the library openpyxl
```

- Data sets stored in **parquet**



```
df = pd.read_parquet('data.parquet')
```

Pandas can read numerous formats, you just have to search for the right Pandas functions that allow you to read your DataFrame format.

Get information about the dataset

The DataFrame object has a method called **info()**, that gives you more information about the data set.

	calories	duration
day1	420	50
day2	380	40
day3	390	45

```
print(df.info())
```

Result:

```
<class 'pandas.core.frame.DataFrame'>
Index: 3 entries, day1 to day3
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   calories  3 non-null      int64
 1   duration  3 non-null      int64
dtypes: int64(2)
memory usage: 72.0+ bytes
None
```

Get information about the dataset

The **shape** property returns the number of rows and columns of the DataFrame.

	calories	duration
day1	420	50
day2	380	40
day3	390	45

```
print(df.shape)
```

Result: (3, 2)

```
print(df.shape[0])
```

Result: 3

```
print(df.shape[1])
```

Result: 2

The **columns** property returns the column names of the DataFrame.

```
print(df.columns)
```

Result: Index(['calories', 'duration'], dtype='object')

Sort values

The **sort_values()** method sorts the DataFrame by the specified label.

```
newdf = df.sort_values(by='duration')
```

	calories	duration
day1	420	50
day2	380	40
day3	390	45

	calories	duration
day1	380	40
day2	390	45
day3	420	50

```
newdf = df.sort_values(by=‘calories', ascending=False)
```

	calories	duration
day1	420	50
day2	380	40
day3	390	45

	calories	duration
day1	420	50
day2	390	45
day3	380	40

Basic statistics

The **describe()** method returns a description of the data in the DataFrame.

	calories	duration
day1	420	50
day2	380	40
day3	390	45

```
print(df.describe())
```

Result:

```
        calories    duration
count    3.000000       3.0
mean   396.666667       45.0
std     20.816660      5.0
min    380.000000      40.0
25%    385.000000      42.5
50%    390.000000      45.0
75%    405.000000      47.5
max    420.000000      50.0e
```

Filter

Filter DataFrame by all rows where the column “calories” is less than 400:

	calories	duration
day1	420	50
day2	380	40
day3	390	45

```
print(df[df['calories'] < 400])
```

Result:

```
        calories  duration
day2     380       40
day3     390       45
```

Filter DataFrame by all rows where the column “calories” is less than 400 and the columns “duration” is greater 40:

	calories	duration
day1	420	50
day2	380	40
day3	390	45

```
print(df[(df['calories'] < 400) & (df['duration'] > 40)])
```

Result:

```
        calories  duration
day3     390       45
```

Filter

Filter the DataFrame by all cells where the column “calories” contain the values 420 and 380:

	calories	duration
day1	420	50
day2	380	40
day3	390	45

```
print(df[df['calories'].isin([420, 380])])
```

Result:

```
      calories  duration
day1    420        50
day2    380        40
```

Filter the DataFrame by all cells that contain the values 50, 380 and 420

	calories	duration
day1	420	50
day2	380	40
day3	390	45

```
print(df[df.isin([420, 380, 50])])
```

Result:

```
   calories  duration
0   420.0     50.0
1   380.0     NaN
2    NaN      NaN
```

Export DataFrames

Pandas DataFrames can be exported to any format:

- To **csv** →

```
df.to_csv('data.csv')
```

- To **xlsx** →

```
df.to_excel('data.xlsx')
# requires installing and importing the library openpyxl
```

- To **parquet** →

```
df.to_parquet('data.parquet')
```

Pandas can export to numerous formats, you just have to search for the right Pandas functions that allow you to export your DataFrame to the format you need.

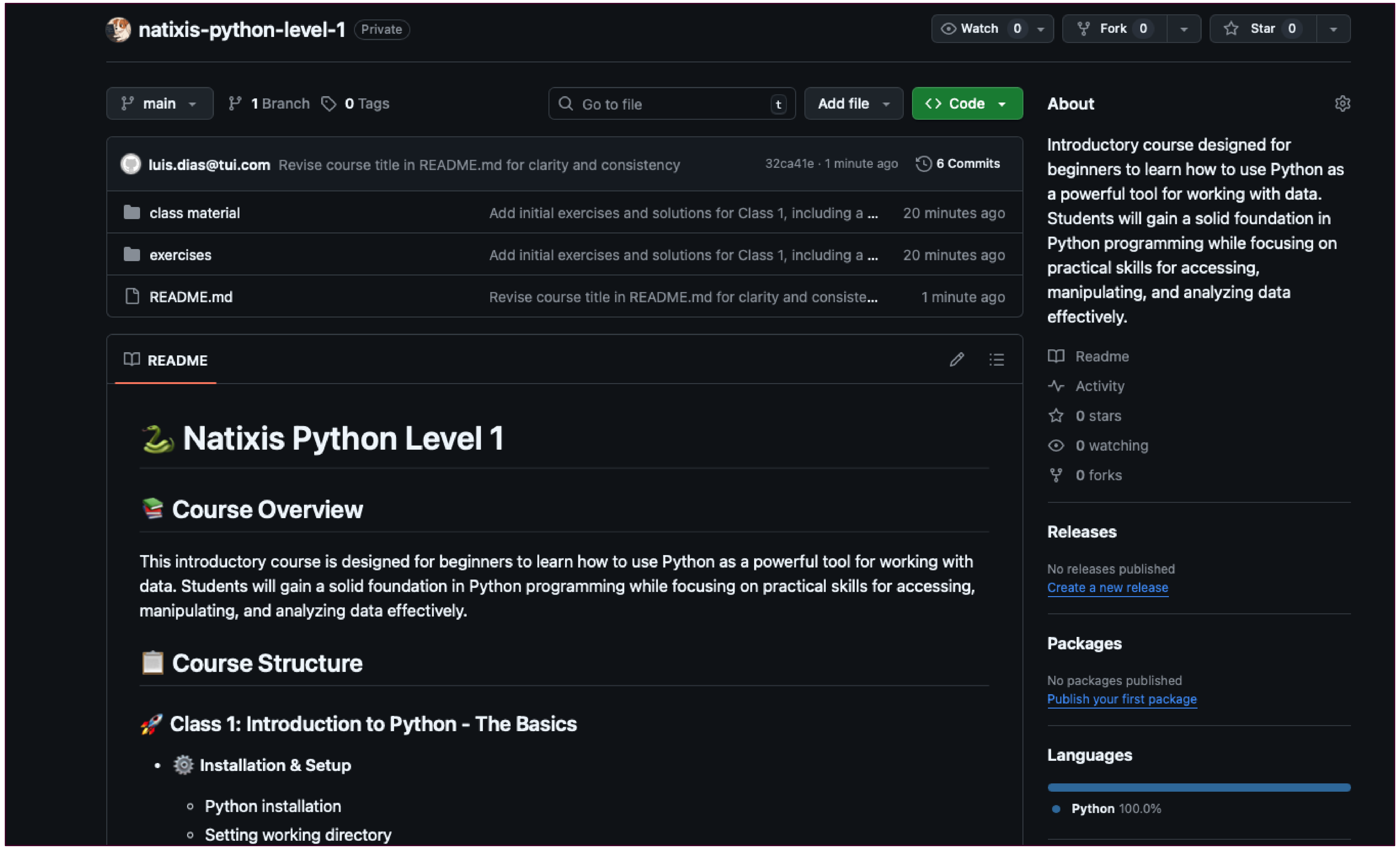
Class Wrap-up

What have we learned today?

- Learned about Python **advanced data structures**: Pandas Dataframe
- Learned about the **power of pandas dataframe** for data wrangling
- Learned how to **read** from a file into a pandas dataframe
- Learned how to **manipulate** a pandas dataframe
- Learned how to **write** to a pandas dataframe

PRACTICE
PRACTICE
PRACTICE

Exercises for today



natixis-python-level-1 Private

luis.dias@tui.com Revise course title in README.md for clarity and consistency 32ca41e · 1 minute ago 6 Commits

class material	Add initial exercises and solutions for Class 1, including a ...	20 minutes ago
exercises	Add initial exercises and solutions for Class 1, including a ...	20 minutes ago
README.md	Revise course title in README.md for clarity and consiste...	1 minute ago

README

🐍 Natixis Python Level 1

📚 Course Overview

This introductory course is designed for beginners to learn how to use Python as a powerful tool for working with data. Students will gain a solid foundation in Python programming while focusing on practical skills for accessing, manipulating, and analyzing data effectively.

📋 Course Structure

🚀 Class 1: Introduction to Python - The Basics

- ⚙️ Installation & Setup
  - Python installation
  - Setting working directory

About

Introductory course designed for beginners to learn how to use Python as a powerful tool for working with data. Students will gain a solid foundation in Python programming while focusing on practical skills for accessing, manipulating, and analyzing data effectively.

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

Python 100.0%

Link to exercises: https://github.com/diaxz12/natixis-python-level-1/blob/main/exercises/Class3_exercises.py

THANK YOU 😊

Questions?

Fábio Neves ✉ fnssm26@gmail.com

Luis Dias ✉ luis.f.s.m.dias@gmail.com