Utrecht University

Data Science Day 2018

Data Handling in Python

Jonathan de Bruin & Barbara Vreede
RDM Support | University Library | IT Services

20 April 2018

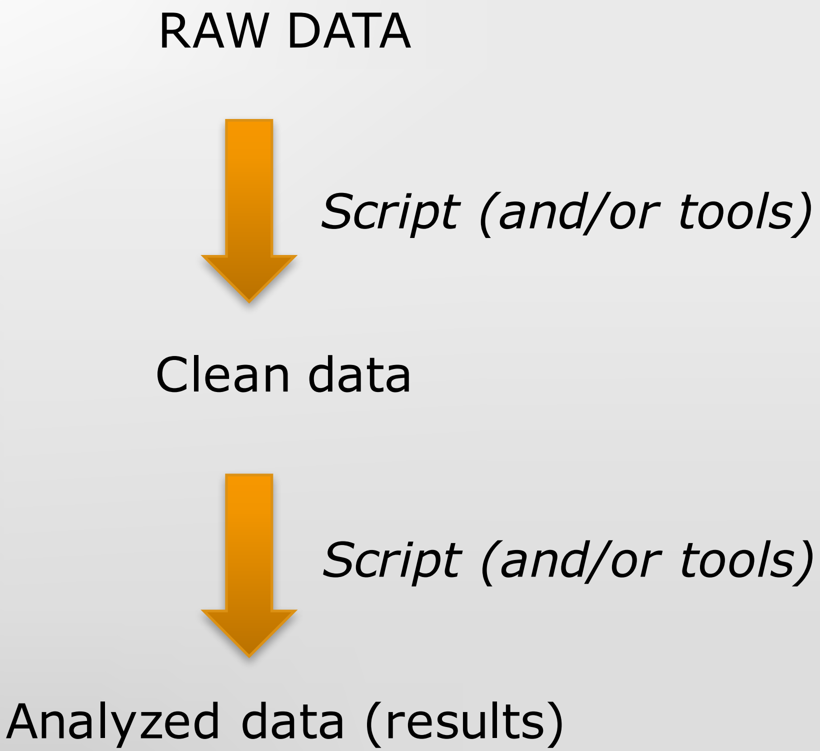
Today’s schedule:

09.30-09.40	Raw data handling, setting the stage! [presentation]
09.40-10.50	Practical Data handling in Python [Jupyter notebook]
	- 10 minute coffee break -
11.10-11.30	Formatting and commenting practices [presentation]
11.20-11.30	Anaconda: ultra quick tour [Jupyter and Spyder]
11.30-12.00	Final Exercise: comment and format the code [Spyder]

Raw data handling

20 April 2018

Data analysis pipeline



RAW DATA

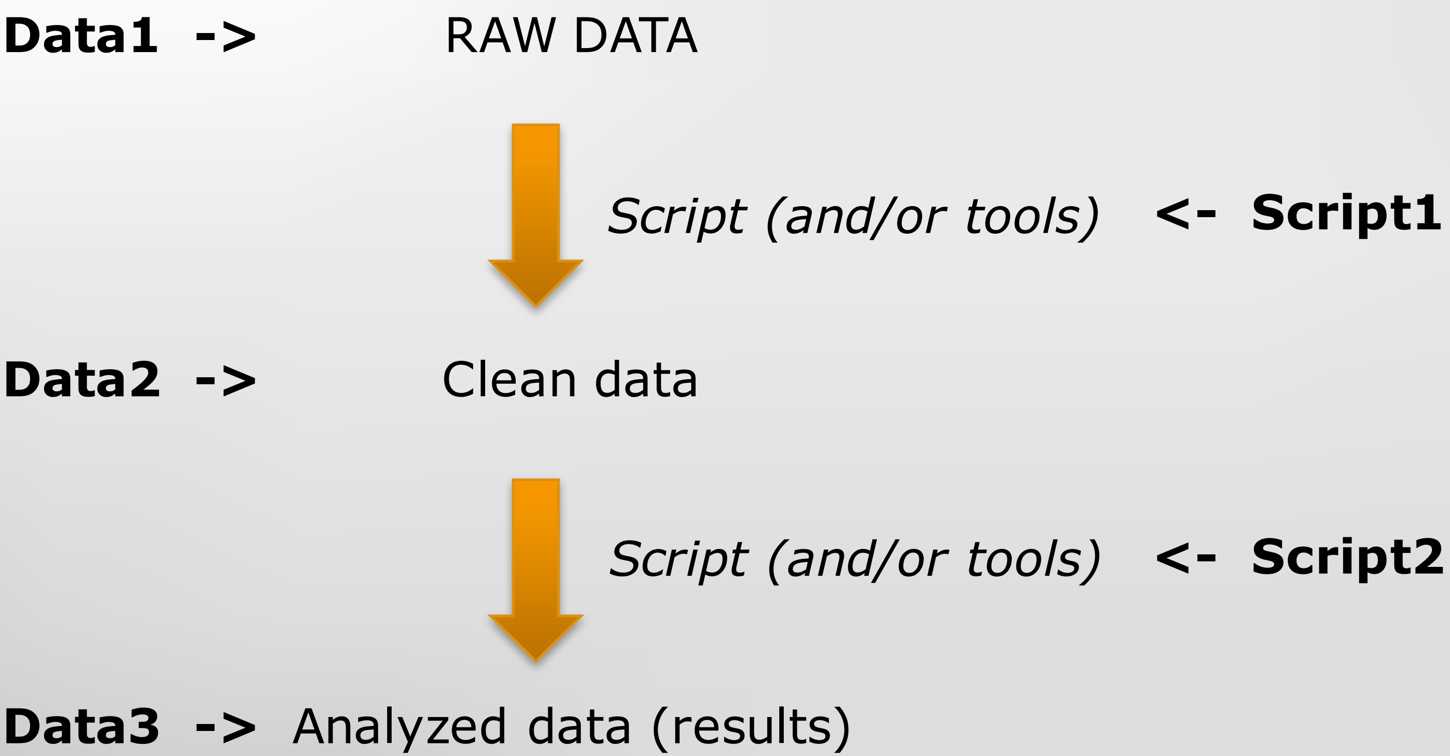
Script (and/or tools)

Clean data

Script (and/or tools)

Analyzed data (results)

Keep copies of your data and your scripts at every step!



Data1 -> RAW DATA

Script (and/or tools) **<- Script1**

Data2 -> Clean data

Script (and/or tools) **<- Script2**

Data3 -> Analyzed data (results)

Cleaning your data

Load and view

- *Are data types correctly encoded?*

Complete dataset

- *Stack to make tidy*
- *Merge with other datasets*

Quality control

- *Explore basic statistics: mean, stdev, etc.*
- *Are the values consistent and in range?*
- *Make some plots and look at the data*

Save clean dataset

Tidy data

- Each *variable* is a column and contains *values*
- Each *observation* is a row
- Each type of *observational unit* forms a table

Messy:

	Treatment A	Treatment B
John Smith	-	2
Jane Doe	16	11
Mary Johnson	3	1

Tidy:

Name	Treatment	Result
John Smith	a	-
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

(*Examples from:* ***http://www.jeannicholashould.com/tidy-data-in-python.html***)

Tools & dataset for this workshop

- We will work in Python…
- Which is embedded in a Jupyter notebook…
- Which runs on Surf facility.

- *Dataset:* we will work with a small fictitious patient dataset, with some general measurements (heart rate, blood pressure) and some factors per patient (i.e. hospital code, date).

Practical Data Handling in Python: your turn!

Part 1:

- Go to the SURF environment at https://uu-its.jove.surfsara.nl
- Log in with the details on your handout
- Open Jupyter notebook

Part 2:

- Install Anaconda if necessary
- Open Anaconda Launcher
- The files (notebook, data1, data2, .py script) can be found in the SURF environment.
- Store these in a folder on your computer
- Open file ‘datahandling.py’ in Spyder on your computer

Best practices for Python coding

20 April 2018

In this (short) overview:

- Script setup
- Script layout and formatting
- Commenting
- File naming conventions

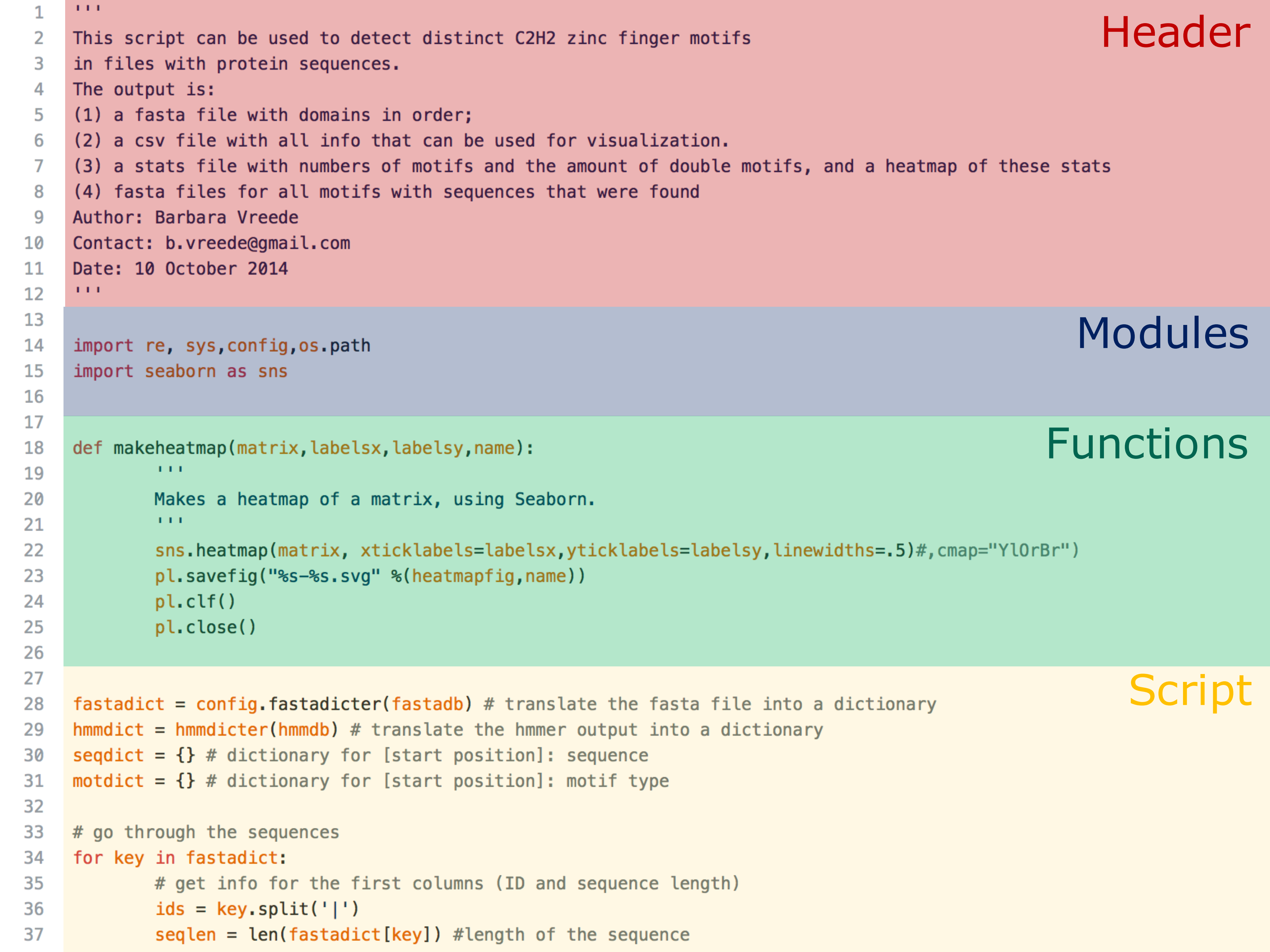
How to set up your python script

1. Header with information on the script:
    - Date
    - Context
    - Author
    - Purpose
    - Dependencies *(what is needed to use the script)*

2. Load modules *(existing code you will use in your script)*

3. Functions *(reusable code that you have written yourself)*

4. …the rest of the script.



Header

```
'''
This script can be used to detect distinct C2H2 zinc finger motifs
in files with protein sequences.
The output is:
(1) a fasta file with domains in order;
(2) a csv file with all info that can be used for visualization.
(3) a stats file with numbers of motifs and the amount of double motifs, and a heatmap of these stats
(4) fasta files for all motifs with sequences that were found
Author: Barbara Vreede
Contact: b.vreede@gmail.com
Date: 10 October 2014
'''
```

Modules

```

import re, sys,config,os.path
import seaborn as sns

```

Functions

```

def makeheatmap(matrix,labelsx,labelsy,name):
        '''
        Makes a heatmap of a matrix, using Seaborn.
        '''
        sns.heatmap(matrix, xticklabels=labelsx,yticklabels=labelsy,linewidths=.5)#,cmap="YlOrBr")
        pl.savefig("%s-%s.svg" %(heatmapfig,name))
        pl.clf()
        pl.close()

```

Script

```

fastadict = config.fastadicter(fastadb) # translate the fasta file into a dictionary
hmmdict = hmmdicter(hmmdb) # translate the hmmer output into a dictionary
seqdict = {} # dictionary for [start position]: sequence
motdict = {} # dictionary for [start position]: motif type

# go through the sequences
for key in fastadict:
        # get info for the first columns (ID and sequence length)
        ids = key.split('|')
        seqlen = len(fastadict[key]) #length of the sequence
```

Python formatting conventions and best practices

- Humans should be able to read your code
- Indentation is a functional part of your code
- Variable names: avoid capitalization, punctuation, ...
- Write robust code: specify constants and variables
- Lines should be short (less than 80 characters)
- Use blank lines sparingly, but wisely

Style guide: https://www.python.org/dev/peps/pep-0008

Humans should be able to read your code

Bad

```
from pandas import to_datetime, read_csv
x =
read_csv(“/home/student3/Desktop/backupsSaturdayDesktopHomeC
omputer/courses/2010/August/summercoursePython/data/PatientD
ATA2.txt”)
to_datetime(x[5],format=”%d-%m-%Y”)
x[5].diff()
```

Good

```
import pandas as pd

# import the data
df_pd2 = pd.read_csv(“PatientDATA2.txt”)

# print the time between each subsequent visit
df_pd2[“VISIT”] = pd.to_datetime(
    df_pd2[“VISIT”],
    format=“%d-%m-%Y”
)
print(df_pd2[“VISIT”].diff())
```

Indentation is a functional part of your code

Bad

```
for date in df_pd2[“VISIT”]:
print(date)
```

```
 File "<stdin>", line 2
  print(date)
    ^
IndentationError: expected an indented block
```

Good

```
for date in df_pd2[“VISIT”]:
    print(date)
```

```
2-3-2016
9-8-2015
17-4-2016
...
```

! Do not mix tabs and spaces! (1 indent = 4 spaces)

Variable names: avoid capitalization, punctuation…

Bad

```
# forbidden: never start with numbers or use .&*%()$#@-...
2df = pandas.read_csv(“PatientDATA2.txt”)
df.2 = pandas.read_csv(“PatientDATA2.txt”)

# bad manners: capitalization or obscure names
Df2 = pandas.read_csv(“PatientDATA2.txt”)
a = pandas.read_csv(“PatientDATA2.txt”)
```

Good

```
df_pd2 = pandas.read_csv(“PatientDATA2.txt”)
patientdata2 = pandas.read_csv(“PatientDATA2.txt”)

# even more explicit
patientdata_visits = pandas.read_csv(“PatientDATA2.txt”)
```

Write robust code

Bad

```
# print the time between each subsequent visit
print(df_pd2[5].diff())

# we want to divide the patients into 5 groups
# define group size
print(100 / 5)
```

Good

```
# print the time between each subsequent visit
print(df_pd2[“VISITS”].diff())

# we want to divide the patients into 5 groups
# define group size
npatients = df_pd2.shape[1]
ngroups = 5
groupsize = npatients / ngroups
```

Lines should be short

Bad

```
# this is a very long comment, and you will have to scroll endlessly to re
print(pandas.to_datetime(df_pd2[“VISIT”], dayfirst=False, format=“%d-%m-%Y
```

Good

```
# keep your comments short, too!
df_pd2[“VISIT”] = pandas.to_datetime(
    df_pd2[“VISIT”],
    format=“%d-%m-%Y”
)
print(df_pd2[“VISIT”].diff())
```

Use blank lines sparingly, but wisely

Bad

```
import pandas as pd
df_pd2 = pd.read_csv(“PatientDATA2.txt”)


df_pd2[“VISIT”] = pd.to_datetime(
     df_pd2[“VISIT”],
     format=“%d-%m-%Y”
)
print(df_pd2[“VISIT”])
```

Good

```
import pandas as pd

df_pd2 = pd.read_csv(“PatientDATA2.txt”)
df_pd2[“VISIT”] = pd.to_datetime(
     df_pd2[“VISIT”],
     format=“%d-%m-%Y”
)

print(df_pd2[“VISIT”])
```

Python commenting best practices

- Headers and comments and docstrings, oh my…
- Humans should be able to read your code!
- Comment your intent, not the code’s function
- Comments can be a great coding tool!

Headers and comments and docstrings, oh my…

Header

```
###########################################################
### Script for the workshop Data Handling in Python
### Date: 18 April 2018
### Author: Jonathan de Bruin, Barbara Vreede
### Contact: j.debruin1@uu.nl, b.m.i.vreede@uu.nl
###########################################################
```

Comment

```
# import the data
df_pd2 = pd.read_csv(“PatientDATA2.txt”)
```

```
df_pd2 = pd.read_csv(“PatientDATA2.txt”)   # import the data
```

Docstring

```
"""
This block of text is a docstring. It is a long(er) piece of
text, that may run over several lines.
Triple quotation marks are used to indicate the beginning
and the end of a docstring.
"""
```

Humans should be able to read your code

Bad

```
import pandas as pd
df_pd2 = pd.read_csv(“PatientDATA2.txt”)
df_pd2[“VISIT”] = pd.to_datetime(
    df_pd2[“VISIT”],
    format=“%d-%m-%Y”
)
print(df_pd2[“VISIT”].diff())
```

Good

```
import pandas as pd

# import the data
df_pd2 = pd.read_csv(“PatientDATA2.txt”)

# print the time between each subsequent visit
df_pd2[“VISIT”] = pd.to_datetime(
    df_pd2[“VISIT”],
    format=“%d-%m-%Y”
)
print(df_pd2[“VISIT”].diff())
```

Comment your intent, not the code’s function

Bad

```
df_pd2  = pd.read_csv(“PatientDATA2.txt”)  # read the csv
df_pd2[“VISIT”]  = pd.to_datetime(df_pd2[“VISIT”],format=“%d-
%m-%Y”) # the “VISIT” column is a date
print(df_pd2[“VISIT”].diff()) # print the difference
```

Good

```
# import the patient dataset that includes visit dates
df_pd2  = pd.read_csv(“PatientDATA2.txt”)

# print the time between each subsequent visit
df_pd2[“VISIT”]  = pd.to_datetime(
    df_pd2[“VISIT”],
    format=“%d-%m-%Y”
)
print(df_pd2[“VISIT”].diff())
```

Comments can be a great coding tool

Starting your script…

```
# import the patient data
# make sure the dataset includes visits
# ensure proper formatting of the visit column
# print the time between each subsequent visit
```

… write the code, revise your comments

```
# import the patient dataset that includes visit dates
df_pd2 = pd.read_csv(“PatientDATA2.txt”)

# print the time between each subsequent visit
df_pd2[“VISIT”] = pd.to_datetime(
    df_pd2[“VISIT”],
    format=“%d-%m-%Y”
)
print(df_pd2[“VISIT”].diff())
```

File naming conventions

Use fixed elements in your file name:
I.e. description of content, project number, name researcher/team.

DON’T →

- Use special characters (&%$#) or periods or spaces
- Start with numbers (e.g. 2018_patientanalysis.py)

DO →

- Keep names short, relevant, but descriptive (stand-alone!)
- Go from generic to specific (handy with sorting and finding)
- Use ‘_’ when combining elements to a filename

Practical Data Handling in Python: your turn!

Part 1:

- Go to the SURF environment at https://uu-its.jove.surfsara.nl
- Log in with the details on your handout
- Open Jupyter notebook

Part 2:

- Install Anaconda if necessary
- Open Anaconda Launcher
- The files (notebook, data1, data2, .py script) can be found in the SURF environment.
- Store these in a folder on your computer
- Open file ‘datahandling.py’ in Spyder on your computer

Contact us:

info.rdm@uu.nl