Two-day workshop
Automated Content Analysis with Python

Day 1 – Morning

Damian Trilling

d.c.trilling@uva.nl
@damian0604
www.damiantrilling.net

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

1–5–2017

Introducing. . .

. . . the people

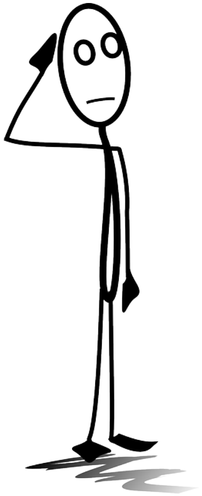
Introducing...
Damian

dr. Damian Trilling
Assistant Professor Political Communication & Journalism

- studied Communication Science in Münster and at the VU 2003–2009
- PhD candidate @ ASCoR 2009–2012
- interested in political communication and journalism in a changing media environment and in innovative (digital, large-scale, computational) research methods

@damian0604 d.c.trilling@uva.nl
REC-C 8th floor www.damiantrilling.net

Introducing. . .
You



Your name?
Your background?
Your reason to follow this course?

This morning

- Why Python?
- How does Python relate to software you might be familiar with, such as SPSS, STATA, or R?
- An introduction to data types – or why a "variable" is not what you might think
- Structure of a program
- Play around!

Why Python?

What is Python?

What?

- A language, not a specific program
- Huge advantage: flexibility, portability
- One of *the* languages for data analysis.

What is Python?

What?

- A language, not a specific program
- Huge advantage: flexibility, portability
- One of *the* languages for data analysis.

What is Python?

What?

- A language, not a specific program
- Huge advantage: flexibility, portability
- One of *the* languages for data analysis.

Which version?

We use Python 3.

`http://www.google.com` or `http://www.stackexchange.com` still offer a lot of Python2-code, but that can easily be adapted. Most notable difference: In Python 2, you write `print "Hi"`, this has changed to `print ("Hi")`

Comparing Python to things you are familiar with

A slightly un-nuanced list:

- SPSS, STATA, and R come from a statistics background, Python from a computer science background
  $\Rightarrow$ In R, things are often thought of as vectors and matrices.
  $\Rightarrow$ But you don't have to know anything about statistics to work with Python (but you *can* if you want to)
- SPSS and STATA (and to a lesser extend also R) assume that your data are a *table*, while Python regularly uses other data structures (like *lists* or *dictionaries*)
- SPSS and STATA are really bad in dealing with *text*
- Many things you can do in Python you can also do in R and vice versa
- *Huge* community of people using Python for processing text, and many great packages

If it's not a program, how do you work with it?

Interactive mode

- Just type `python3` on the command line, and you can start entering Python commands (You can leave again by entering `quit()`)
- Great for quick try-outs, but you cannot even save your code

If it’s not a program, how do you work with it?

Interactive mode

- Just type `python3` on the command line, and you can start entering Python commands (You can leave again by entering `quit()`)
- Great for quick try-outs, but you cannot even save your code

An editor of your choice

- Write your program in any text editor, save it as `myprog.py`
- and run it from the command line with `./myprog.py` or `python3 myprog.py`

If it's not a program, how do you start it?

An IDE (Integrated Development Environment)

- Provides an interface
- Both quick interactive try-outs and writing larger programs
- We use spyder, which looks a bit like RStudio (and to some extent like Stata)

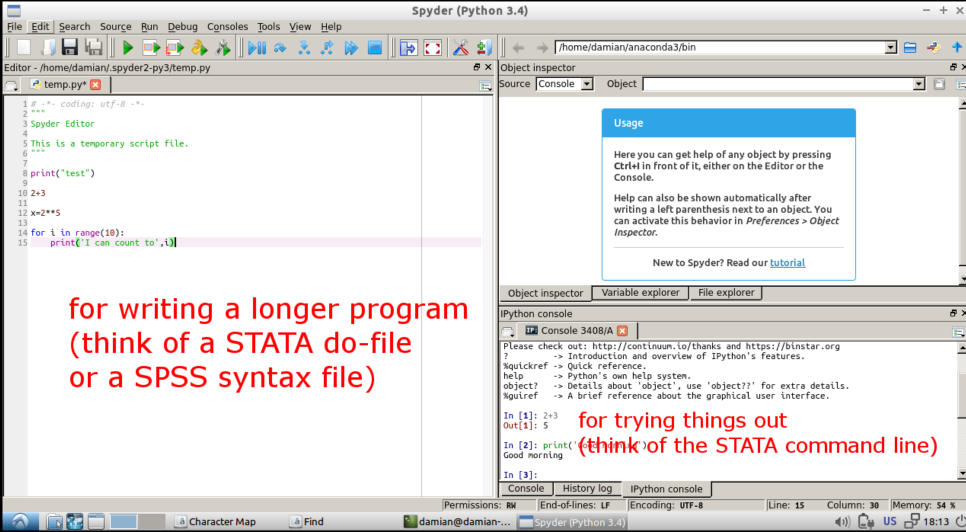
If it's not a program, how do you start it?

An IDE (Integrated Development Environment)

- Provides an interface
- Both quick interactive try-outs and writing larger programs
- We use spyder, which looks a bit like RStudio (and to some extent like Stata)

Jupyter Notebook

- Runs in your browser
- Stores results and text along with code
- Great for *interactive* playing with data and for sharing results



Spyder (Python 3.4)
File Edit Search Source Run Debug Consoles Tools View Help
/home/damian/anaconda3/bin
Editor - /home/damian/.spyder2-py3/temp.py
temp.py*
Object inspector
Source Console Object
Object inspector Variable explorer File explorer
IPython console
Console 3408/A
Console History log IPython console
Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 15 Column: 30 Memory: 54 %
Character Map Find damian@damian-... Spyder (Python 3.4) US 18:13

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

print("test")

2+3

x=2**5

for i in range(10):
    print('I can count to',i)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

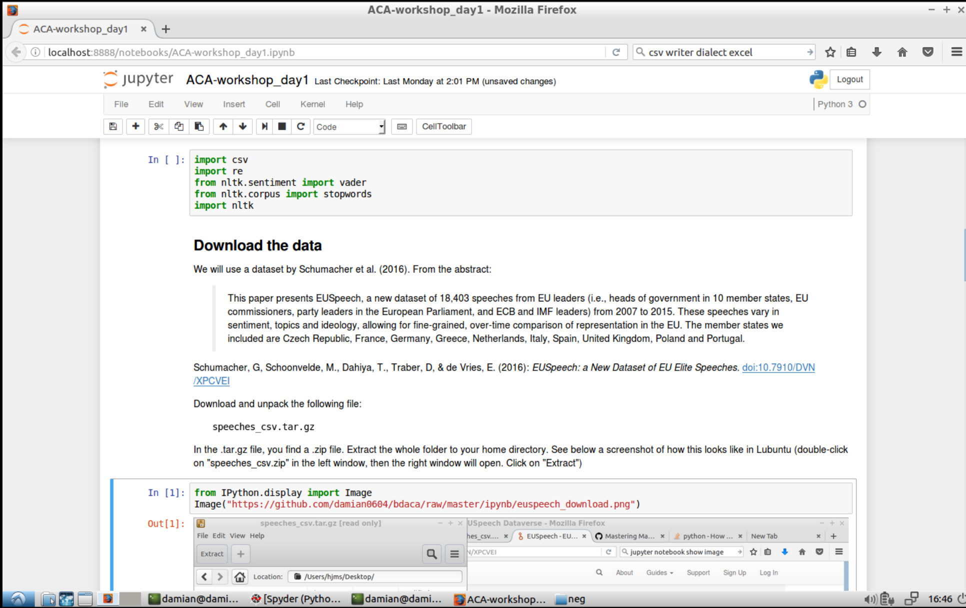
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Object Inspector.*

New to Spyder? Read our tutorial

```
Please check out: http://continuum.io/thanks and https://binstar.org
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
%guiref   -> A brief reference about the graphical user interface.

In [1]: 2+3
Out[1]: 5

In [2]: print('Good morning')
Good morning

In [3]:
```

for writing a longer program (think of a STATA do-file or a SPSS syntax file)

for trying things out (think of the STATA command line)


ACA-workshop_day1 - Mozilla Firefox
localhost:8888/notebooks/ACA-workshop_day1.ipynb
csv writer dialect excel
jupyter ACA-workshop_day1 Last Checkpoint: Last Monday at 2:01 PM (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Python 3
Code
CellToolbar

In [ ]:
```
import csv
import re
from nltk.sentiment import vader
from nltk.corpus import stopwords
import nltk
```

Download the data

We will use a dataset by Schumacher et al. (2016). From the abstract:

> This paper presents EUSpeech, a new dataset of 18,403 speeches from EU leaders (i.e., heads of government in 10 member states, EU commissioners, party leaders in the European Parliament, and ECB and IMF leaders) from 2007 to 2015. These speeches vary in sentiment, topics and ideology, allowing for fine-grained, over-time comparison of representation in the EU. The member states we included are Czech Republic, France, Germany, Greece, Netherlands, Italy, Spain, United Kingdom, Poland and Portugal.

Schumacher, G, Schoonvelde, M., Dahiya, T., Traber, D, & de Vries, E. (2016): *EUSpeech: a New Dataset of EU Elite Speeches.* doi:10.7910/DVN/XPCVEI

Download and unpack the following file:

```
speeches_csv.tar.gz
```

In the .tar.gz file, you find a .zip file. Extract the whole folder to your home directory. See below a screenshot of how this looks like in Lubuntu (double-click on "speeches_csv.zip" in the left window, then the right window will open. Click on "Extract")

In [1]:
```
from IPython.display import Image
Image("https://github.com/damian0604/bdaca/raw/master/ipynb/euspeech_download.png")
```

Out[1]:
speeches_csv.tar.gz [read only]
File Edit View Help
Extract
Location: /Users/hjms/Desktop/
USpeech Dataverse - Mozilla Firefox
EUSpeech - EU...
Mastering Ma...
python - How ...
New Tab
jupyter notebook show image
About Guides Support Sign Up Log In
damian@dami...
[Spyder (Pytho...
damian@dami...
ACA-workshop...
neg
16:46

Let's start up a Python environment and write a Hello-world-program!

Start playing!

Exercises **1. Run a program that greets you.**

The code for this is

```
print("Hello world")
```

After that, do some calculations. You can do that in a similar way:

```
a=2
print(a*3)
```

Just play around.

Additional ressources
Codecademy course on Python
`https://www.codecademy.com/learn/python`

The very, very, basics of programming

You can read all this back in Chapter 4.

Python lingo

Basic datatypes (variables)

int	`32`
float	`1.75`
bool	`True`, `False`
string	`"Damian"`

Python lingo

Basic datatypes (variables)

int	`32`
float	`1.75`
bool	`True`, `False`
string	`"Damian"`
(variable name	`firstname`)

"firstname" and firstname is not the same.

Python lingo

Basic datatypes (variables)

int	`32`
float	`1.75`
bool	`True`, `False`
string	`"Damian"`
(variable name	`firstname`)

"firstname" and firstname is not the same.

"5" and 5 is not the same.

But you can transform it: `int("5")` will return 5.

You cannot calculate 3 `* "5"` Actually, you can. It gives you "555".

But you can calculate 3 `* int("5")`

Python lingo

More advanced datatypes

Python lingo

More advanced datatypes

list
```
firstnames = ['Damian','Lori','Bjoern']
lastnames =
['Trilling','Meester','Burscher']
```

Python lingo

More advanced datatypes

list firstnames = ['Damian','Lori','Bjoern']
lastnames =
['Trilling','Meester','Burscher']

list ages = [18,22,45,23]

Python lingo

More advanced datatypes

list `firstnames = ['Damian','Lori','Bjoern']`
`lastnames = ['Trilling','Meester','Burscher']`

list `ages = [18,22,45,23]`

dict `familynames= {'Bjoern': 'Burscher', 'Damian': 'Trilling', 'Lori': 'Meester'}`

dict `{'Bjoern': 26, 'Damian': 31, 'Lori': 25}`

Python lingo

Functions

Python lingo

Functions

functions	Take an input and return something else `int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

Python lingo

Functions

functions	Take an input and return something else `int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.
methods	are similar to functions, but directly associated with an object. `"SCREAM".lower()` returns the string "scream"

Python lingo

Functions

functions	Take an input and return something else `int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.
methods	are similar to functions, but directly associated with an object. `"SCREAM".lower()` returns the string "scream"

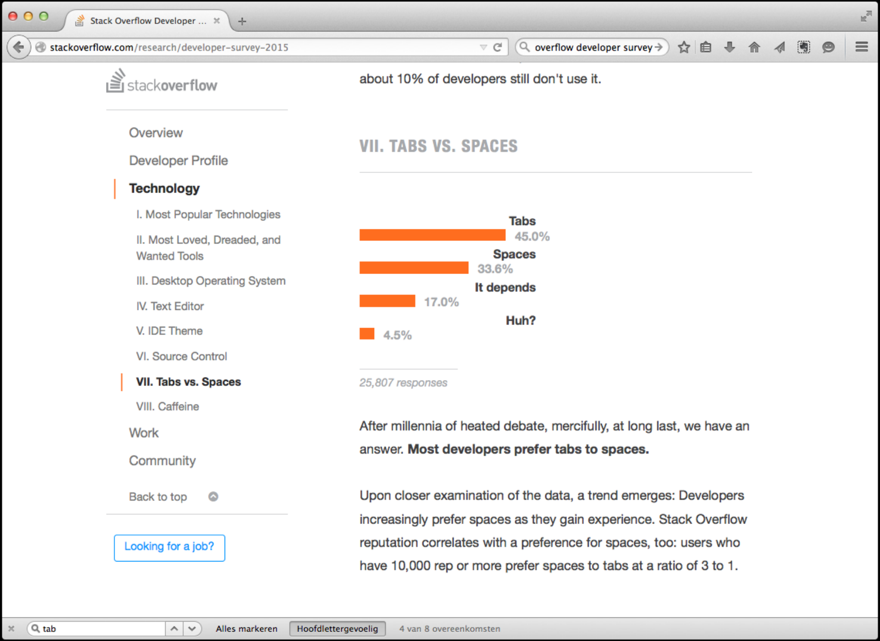
Both functions and methods end with `()`. Between the `()`, *arguments* can (sometimes have to) be supplied.

Indention: The Python way of structuring your program

Indention

Structure

The program is structured by TABs or SPACEs



Stack Overflow Developer ...
stackoverflow.com/research/developer-survey-2015
overflow developer survey
stackoverflow

Back to top

Looking for a job?

about 10% of developers still don't use it.

VII. TABS VS. SPACES

Tabs
45.0%
Spaces
33.6%
It depends
17.0%
Huh?
4.5%

25,807 responses

After millennia of heated debate, mercifully, at long last, we have an answer. **Most developers prefer tabs to spaces.**

Upon closer examination of the data, a trend emerges: Developers increasingly prefer spaces as they gain experience. Stack Overflow reputation correlates with a preference for spaces, too: users who have 10,000 rep or more prefer spaces to tabs at a ratio of 3 to 1.

Indention

Structure

The program is structured by TABs or SPACEs

```
firstnames=['Damian','Lori','Bjoern']
age={'Bjoern': 27, 'Damian': 32, 'Lori': 26}
print ("The names and ages of all BigData people:")
for naam in firstnames:
    print (naam,age[naam])
```

Indention

Structure

The program is structured by TABs or SPACEs

```
firstnames=['Damian','Lori','Bjoern']
age={'Bjoern': 27, 'Damian': 32, 'Lori': 26}
print ("The names and ages of all BigData people:")
for naam in firstnames:
    print (naam,age[naam])
```

Don't mix up TABs and spaces! Both are valid, but you have to be consequent!!! Best: always use 4 spaces!

Indention

Structure

The program is structured by TABs or SPACEs

```
print ("The names and ages of all BigData people:")
for naam in firstnames:
    print (naam,age[naam])
    if naam=="Damian":
        print ("He teaches this course")
    elif naam=="Lori":
        print ("She was an assistant last year")
    elif naam=="Bjoern":
        print ("He helps on Wednesdays")
    else:
        print ("No idea who this is")
```

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a `:`

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (`if`, `elif`, and `else` statements)

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a `:`

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (`if`, `elif`, and `else` statements)
- an alternative block should be executed if an error occurs (`try` and `except` statements)

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (`if`, `elif`, and `else` statements)
- an alternative block should be executed if an error occurs (`try` and `except` statements)
- a file is opened, but should be closed again after the block has been executed (`with` statement)

Let’s explore for-loops and the like interactively

– start up a Python interpreter –

Exercise

Exercise

Write a program that loops over a list of strings and checks whether they contain a substring of your choice.
Things you need:

- for loops
- if statements
- the print() function
- the in operator:

```
In [39]: "a" in "morning"
Out[39]: False

In [40]: "o" in "morning"
Out[40]: True

In [41]: "orn" in "morning"
Out[41]: True
```

Loading and saving data

Working with files

Using pandas

There is a R-like data structure (a pandas dataframe) in which can directly read CSV, Excel, Stata, ...

```
 import pandas as pd
mydataframe = pd.read_csv("test.csv")
```

But often, we don't necessarily want a *table*. In fact, we might not even want to have the whole file in memory.

Working with files

Reading a file line by line

```
with open('test.csv') as fi:
    for line in fi:
        print(line) # or do something more useful
```

We thus loop over the file line-by-line, taking one line at a time, do something with it, and take the next one.
This means that we can process files of arbitrary size, as we never have more than one single line in memory. (⇒ very different approach to the concept of "opening" a file than in programs you are familiar with)

Working with files

Writing a file line by line

```
with open('test.txt',mode='w') as fo:
    fo.writeline('test test test
n')
```

`.writeline()` takes a single string as argument, `.writelines()` a list of strings

The csv module

There is a useful package for dealing with csv files. It splits every row in a list of strings (for each column):

```
import csv
name=[]
age=[]
height=[]
with open('/home/damian/mensen.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for row in reader:
        name.append(row[0])
        age.append(row[1])
        height.append(row[2])
print("Done!")
```