A Practical Introduction to Machine Learning in Python
Day 1 - Monday Morning
»Introduction«

Damian Trilling
Anne Kroon

d.c.trilling@uva.nl, @damian0604
a.c.kroon@uva.nl, @annekroon

Today

Introducing. . .

. . . the people

Setting the stage

Defining "Big Data"

Defining Computational (Social|Communication) Science

The toolbox

The role of software in CSS

Python: A language, not a program

CSS project workflow

step-by-step

All course materials can be found at...

https://github.com/annekroon/gesis-machine-learning

Introducing. . .

Introducing. . .

. . . the people

Introducing...

...the people

Introducing...

dr. Damian Trilling
Associate Professor Political Communication & Journalism

- 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 |www.damiantrilling.net

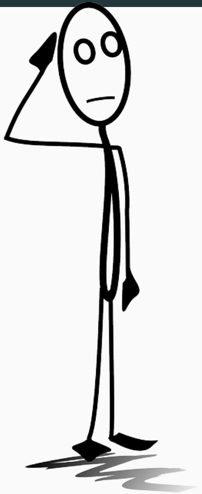
Introducing...

dr. Anne Kroon
Assistant Professor Corporate Communication

- Research focus on biased AI in recruitment, and media bias regarding minorities
- text analysis using automated approaches, word embeddings

@annekroon |a.c.kroon@uva.nl |http://www.uva.nl/profiel/k/r/a.c.kroon/a.c.kroon.html

Introducing...



Your name?

Your background?

Your reason to follow this course?

Do you have a dataset you are working on?

Short poll

Do you need

a an intro
b a brief refresher
c nothing

on

i datatypes (int, float, string, lists, dictionaries)
ii control flow statements (for, if, try/except)
iii ways to run your code (notebooks vs IDE's vs text editors)

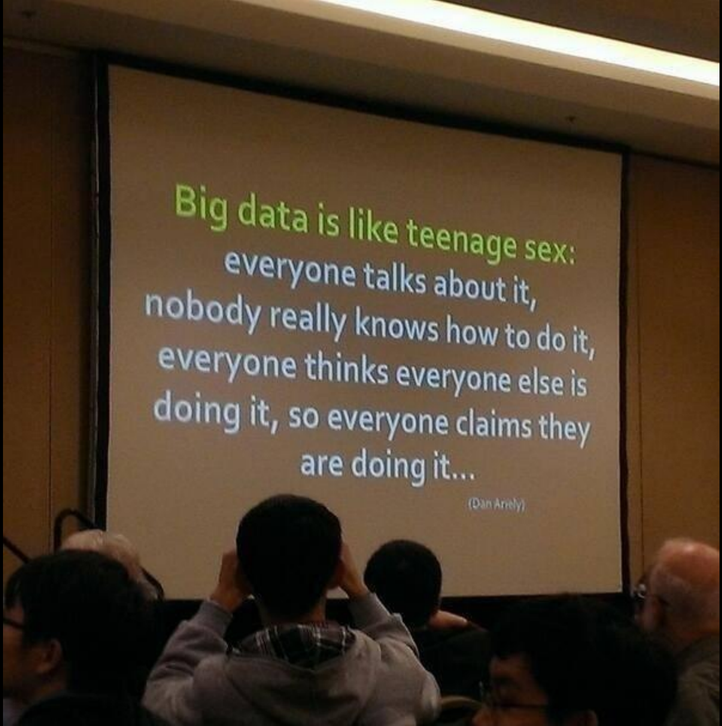
?

We will try do adapt today's programme to your needs!

Setting the stage

Setting the stage

Defining “Big Data”


Big data is like teenage sex:
everyone talks about it,
nobody really knows how to do it,
everyone thinks everyone else is
doing it, so everyone claims they
are doing it...
(Dan Ariely)

The “pragmatic” definition

Everything that needs so much computational power and/or storage that you cannot do it on a regular computer.

The “commercial” definition

gartner

“Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation.”

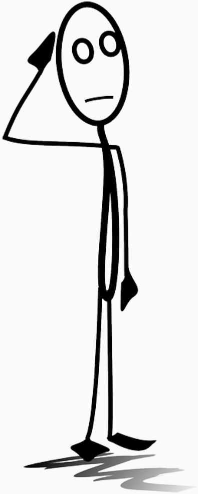
The "critical" definition

Boyd and Crawford (2012)

"

1. Technology: maximizing computation power and algorithmic accuracy to gather, analyze, link, and compare large data sets.
2. Analysis: drawing on large data sets to identify patterns in order to make economic, social, technical, and legal claims.
3. Mythology: the widespread belief that large data sets offer a higher form of intelligence and knowledge that can generate insights that were previously impossible, with the aura of truth, objectivity, and accuracy.

"



Do you think we are doing Big Data analysis?

Setting the stage

Defining Computational (Social|Communication) Science

A very young field

Lazer et al. (2009)

"The capacity to collect and analyze massive amounts of data has transformed such fields as biology and physics. But the emergence of a data-driven 'computational social science' has been much slower."

Epistemologies and paradigm shifts

Kitchin2014

- (Reborn) empiricism: purely inductive, correlation is enough
- Data-driven science: knowledge discovery guided by theory
- Computational social science and digital humanities: employ Big Data research within existing epistemologies
  - DH: descriptive statistics, visualizations
  - CSS: prediction and simulation

CCS as a subset of CSS

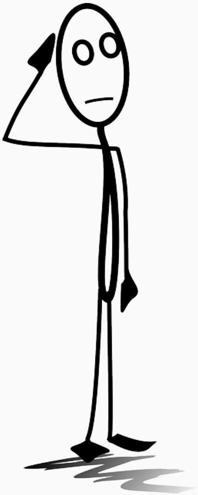
Hilbert et al. (2019)

"...our definition of computational communication science as an application of computational science to questions of human and social communication. As such, it is a natural subfield of computational social science" (followed by references to CSS definitions)

Data, analysis, theory

van Atteveldt and Peng (2018)

“. . . computational communication science studies generally involve: (1) large and complex data sets; (2) consisting of digital traces and other “naturally occurring” data; (3) requiring algorithmic solutions to analyze; and (4) allowing the study of human communication by applying and testing communication theory.”



1. *What do you think? What is the essence of Big Data/CSS/CCS?*
2. *How will what we do here relate to theories and methods from other courses?*

The toolbox

The toolbox

The role of software in CSS

Why program your own tool?

Vis (2013)

"Moreover, the tools we use can limit the range of questions that might be imagined, simply because they do not fit the affordances of the tool. Not many researchers themselves have the ability or access to other researchers who can build the required tools in line with any preferred enquiry. This then introduces serious limitations in terms of the scope of research that can be done."

Some considerations regarding the use of software in science

Assuming that science should be *transparent* and *reproducible by anyone*, we should

use tools that are

- platform-independent
- free (as in beer and as in speech, gratis and libre)
- which implies: open source

This ensures it can our research (a) can be reproduced by anyone, and that there is (b) no black box that no one can look inside. ⇒ ongoing open-science debate! (van Atteveldt et al., 2019)

Why program your own tool?

Vis (2013)

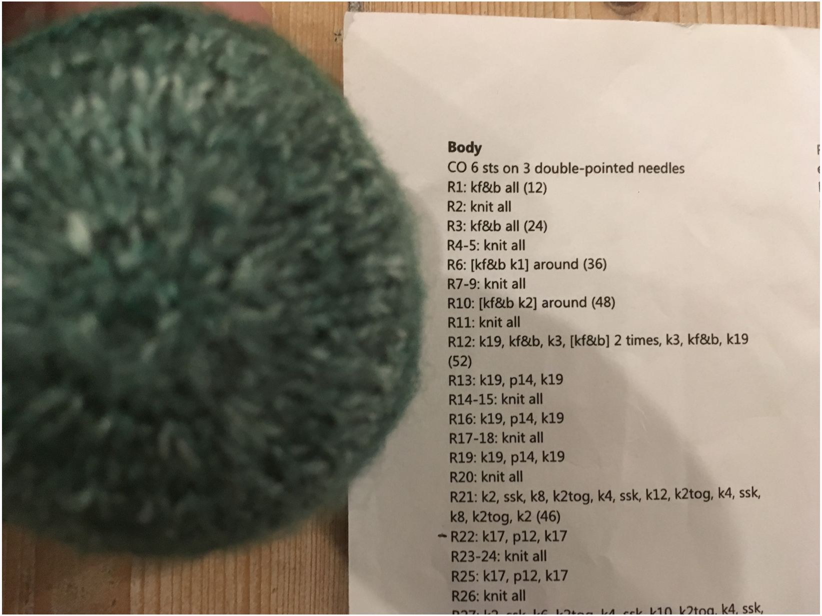
"[. . .] these [commercial] tools are often unsuitable for academic purposes because of their cost, along with the problematic 'black box' nature of many of these tools."

Mahrt and Scharkow (2013)

"[. . .] we should resist the temptation to let the opportunities and constraints of an application or platform determine the research question [. . .]"

The toolbox

Python: A language, not a program



An algorithm in a language that's a bit harder (I think) than Python

Python

What?

- A language, not a specific program
- Huge advantage: flexibility, portability
- One of *the* languages for data analysis. (The other one is R.)
  But Python is more flexible—the original version of Dropbox was written in Python. Some people say: R for numbers, Python for text and messy stuff.

Which version?

We use Python 3.

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

CSS project workflow

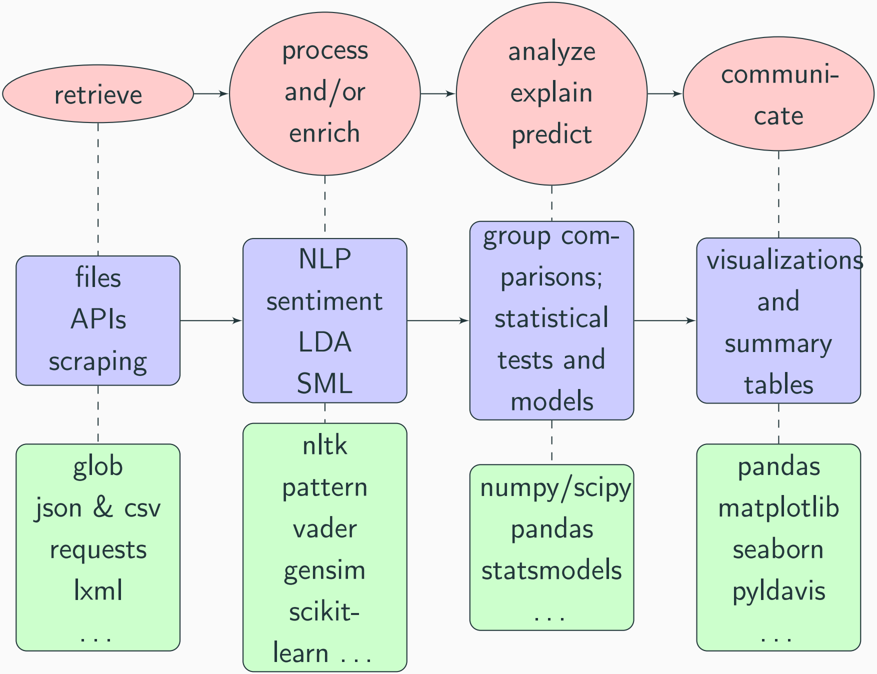
CSS project workflow

step-by-step

Steps of a CSS project

Different techniques for:

- retrieving data (previous week)
- processing data (previous week)
- analyzing data (main part of this week)
- visualising data (a bit on Friday)


retrieve
process and/or enrich
analyze explain predict
communi- cate
files APIs scraping
NLP sentiment LDA SML
group com- parisons; statistical tests and models
visualizations and summary tables
glob json & csv requests lxml . . .
nltk pattern vader gensim scikit- learn . . .
numpy/scipy pandas statsmodels . . .
pandas matplotlib seaborn pyldavis . . .

CSS project workflow

A good workflow

A good workflow

The big picture

Start with pen and paper

1. Draw the Big Picture
2. Then work out what components you need

Best practices

Best practices

Open science

Maximize transparency

Maximizing transparency of code and data

- Use openly accessible repository (e.g., Github)
- Store and preserve (pseudonymised) data at a secure environment (e.g., OSF)
- Create reusable workflows

Advantages

- Reusable data and code
- Efficiency and credibility
- Recognition of tools and data

Best practices

Clean, high-quality code

Develop components separately

One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

Start small, then scale up

- Take your plan and solve *one* problem at a time (e.g., parsing a review page; or getting the URLs of all review pages)
- (for instance, by using functions [next slides])

Develop components separately

If you copy-paste code, you are doing something wrong

- Write loops!
- If something takes more than a couple of lines, write a function!

Copy-paste approach
(ugly, error-prone, hard to scale up)

```
allreviews = []

response = requests.get('http://xxxxx')
tree = fromstring(response.text)
reviewelements = tree.xpath('//div[@class="review"]')
reviews = [e.text for e in reviewelements]
allreviews.extend(reviews)

response = requests.get('http://yyyyy')
tree = fromstring(response.text)
reviewelements = tree.xpath('//div[@class="review"]')
reviews = [e.text for e in reviewelements]
allreviews.extend(reviews)
```

Better: for-loop

(easier to read, less error-prone, easier to scale up (e.g., more URLs, read URLs from a file or existing list)

```
allreviews = []

urls = [‘http://xxxxx’, ‘http://yyyyy’]

for url in urls:
    response = requests.get(url)
    tree = fromstring(response.text)
    reviewelements = tree.xpath(‘//div[@class="review"]’)
    reviews = [e.text for e in reviewelements]
    allreviews.extend(reviews)
```

Even better: for-loop with functions
(main loop is easier to read, function can be re-used in multiple contexts)

```
def getreviews(url):
    response = requests.get(url)
    tree = fromstring(response.text)
    reviewelements = tree.xpath(‘//div[@class="review"]’)
    return [e.text for e in reviewelements]


urls = [‘http://xxxxx’, ‘http://yyyyy’]

allreviews = []

for url in urls:
    allreviews.extend(getreviews(url))
```

Best practices

Exercise

Exercises

Did you pass?

- Think of a way to determine for a list of grades whether they are a pass (>5.5) or fail.
- Can you make that program robust enough to handle invalid input (e.g., a grade as 'ewghjieh')?
- How does your program deal with impossible grades (e.g., 12 or -3)?
- ...

Best practices

datatypes

Datatypes

Low-level: Native python datatypes

- Booleans, integers, floats, strings, bytes, byte arrays
- Lists, typles, sets, dictionaries

Advantages

- fast, flexible
- allows for nested, unstructured data

Disadvantages

- can be more cumbersome: e.g., inserting a column
- less consistency checks

Datatypes

Higher-level: importing modules

- e.g., numpy, pandas, seaborn

Advantages

- useful convenience functionality, works very intuitively (for tabular data)
- easy, allows for pretty visualization

Disadvantages

- not suited for one-dimensional or messy / deeply nested data
- when your data is very large (machine learning!!)

Datatypes in this course

In this week, we will mainly work with lower-level datatypes (as opposed to, for instance, pandas dataframes)

- Often, ML algorithms require native data types as input (i.e., lists, genertors)
- We have to seriously consider memory:
- Maybe size does not apply to your project yet, but in the future you might want to scale up.

Best practices

Generators

Generators

Generators

- We will work with *generators* to deal with memory issues
- Generators behave like iterators: loops through elements of an object.

Behavior of a generator

- Does not hold results in memory
- Only computes results at the moment you need them (i.e. lazy')
- You can only loop over your object ONCE.

Creating generators: Example 1

```
def my_generator(my_list):
    for i in my_list:
        yield i
example_list = [1, 2, 3, 4]
gen1 = my_generator(example_list)
next(gen1)
```

This will return:

```
1
```

Creating generators: Example 2 (shorter)

```
my_list = [1,2,3,4]
gen = (i for i in my_list)
```

Best practices

Scaling up

Scaling up

When considering datatypes, consider re-usability, scalability

- Use functions and classes to make code more readable and re-usable
- Avoid re-calculating values
- Think about how to minimize memory usage (e.g., Generators)
- Do not hard-code values, file names, etc., but take them as arguments

Make it robust

You cannot foresee every possible problem.
Most important: Make sure your program does not fail and loose all data just because something goes wrong at case 997/1000.

- Use `try/except` to explicitly tell the program how to handle errors
- Write data to files (or database) in between
- Use `assert len(x) == len(y)` for sanity checks

Best practices

data storage

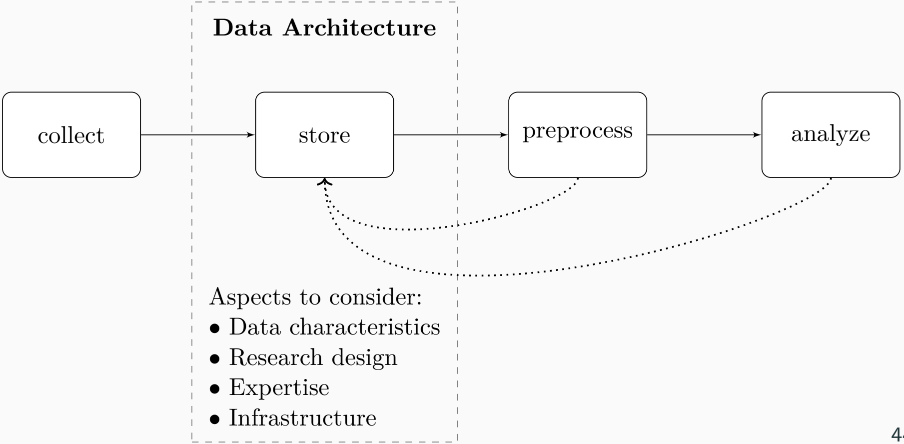
Storing data

Use of databases

Storing data

- We can store our data in files (often, one CSV or JSON file)
- But that's not very efficient if we have large datasets; especially if we want to select subsets later on
- SQL-databases to store tables (e.g., MySQL)
- NoSQL-databases to store less structured data (e.g., JSON with unknown keys) (e.g., MongoDB, ElasticSearch)
- ⇒ Günther, E., Trilling, D., & Van de Velde, R.N. (2018). But how do we store it? (Big) data architecture in the social-scientific research process. In: *Stuetzer, C.M., Welker, M., & Egger, M. (eds.): Computational Social Science in the Age of Big Data. Concepts, Methodologies, Tools, and Applications*. Cologne, Germany: Herbert von Halem.

Storing data



From retrieved data to enriched data

1. Retrieved →

```
<div>
<span id='author'
class='big'>
Author: John
Doe </span>
<span id='viewed'>
seen 42 times
</span>
</div>
```

2. Structured →

```
{
"author":
" Author: John Doe ",
"viewed":
" seen 42 times "
}
```

3. Cleaned →

```
{
"author":
" john doe ",
"viewed":
42 ,
}
```

4. Enriched

```
{
"author":
" john doe ",
"author_gender":
" M ",
"viewed":
42 ,
}
```

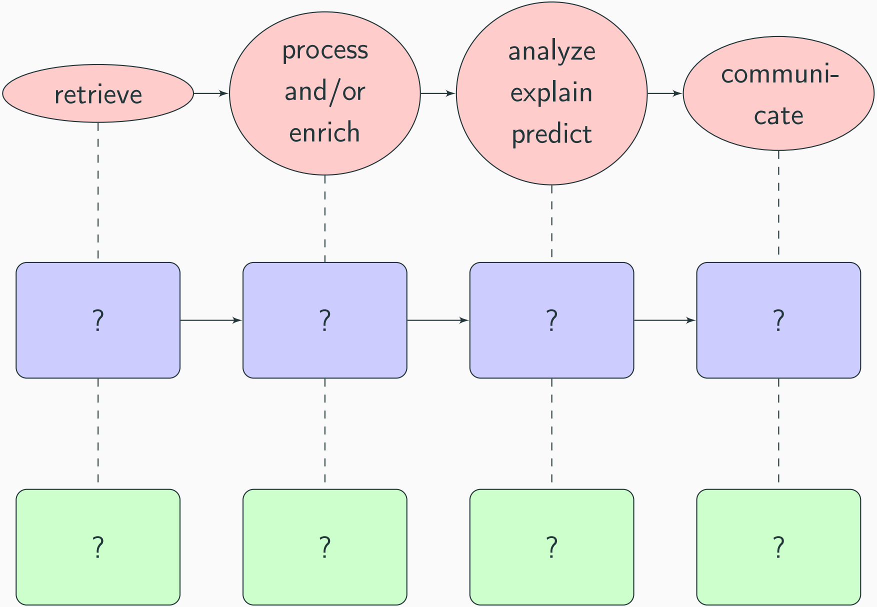
Looking forward

Looking forward

And now you...

Looking forward

Try to fill in the blanks for your personal CSS project


retrieve
process and/or enrich
analyze explain predict
communi-cate
?
?
?
?
?
?
?
?

Long story short:

Don't forget to plan the bigger picture

We will focus on machine learning this week. But for each technique we cover, think about how it fits in *your* workflow.

...and now lets get started!

The ACA toolkit

Types of Automated Content Analysis

The ACA toolkit

Top-down vs. bottom-up

Methodological approach

	Counting and Dictionary	*Supervised Machine Learning*	*Unsupervised Machine Learning*
Typical research interests and content features	visibility analysis sentiment analysis subjectivity analysis	frames topics gender bias	frames topics
Common statistical procedures	string comparisons counting	support vector machines naive Bayes	principal component analysis cluster analysis latent dirichlet allocation semantic network analysis

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure `y`)

Again, you already know some techniques to find out how `x1`, `x2,...x_i` co-occur from other courses:

- Principal Component Analysis (PCA)
- Cluster analysis
-

Final

This afternoon

Getting started
Getting started with the IMBD dataset

Backupslides basics

Backup slides in case we need to do more fundamentals

Datatypes

Python lingo

Basic datatypes (variables)

int `37`

float `1.75`

bool `True`, `False`

string `"Alice"`

(**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"` (In fact, you can. It's `"555"`).

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

Python lingo

More advanced datatypes

list `firstnames = ['Alice','Bob','Cecile']`
`lastnames = ['Garcia','Lee','Miller']`

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

dict `agedict = {'Alice': 18, 'Bob': 22, 'Cecile': 45}`

Note that the elements of a list, the keys of a dict, and the values of a dict can have any* datatype! (You can even mix them, but it's better to be consistent!)

*Well, keys cannot be mutable → see book

Python lingo

Retrieving specific items

list `firstnames[0]` gives you the first entry

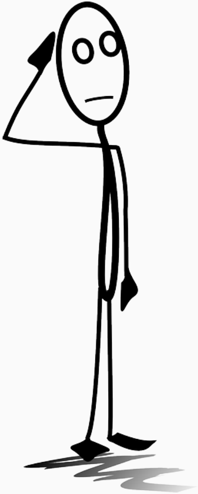
`firstnames[-2]` gives you the one-but-last entry

`firstnames[:2]` gives you entries 0 and 1

`firstnames[1:3]` gives you entries 1 and 2

`firstnames[1:]` gives you entries 1 until the end

dict `agedict["Alice"]` gives you 18



Think of at least two different ways of storing data about some fictious persons (first name, last name, age, phone number, . . .) using lists and/or dictionaries. What are the pros and cons?

Python lingo

Less frequent, but still useful datatypes

set A collection in which each item is unique: `{1,2,3}`

tuple Like a list, but *immutable*: `(1,2,2,2,3)`

defaultdict A dict that does not raise an error but returns the "empty" value of its datatype (0 for int, `""` for str) if you try access a non-existing key (great for storing results and counting things!)

np.array A list-like datatype provided by the `numpy` package optimized for efficient mathematical operations.

... ...

Functions and methods

Python lingo

Functions

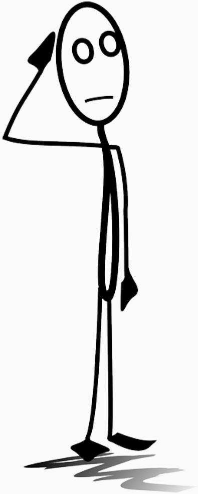
functions Take an input and return something else
`int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

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.

Some functions

```
len(x)        # returns the length of x
y = len(x)    # assign the value returned by len(x) to y
print(len(x)) # print the value returned by len(x)
print(y)      # print y
int(x)        # convert x to an integer
str(x)        # convert x to a string
sum(x)        # get the sum of x
```



How could you print the mean (average) of a list of integers using the functions on the previous slide?

Some methods

Some string methods

```
mystring = "Hi! How are you?"
mystring.lower() # return lowercased string (doesn't change original!)
mylowercasedstring = mystring.lower() # save to a new variable
mystring = mystring.lower() # or override the old one
mystring.upper() # uppercase
mystring.split() # Splits on spaces and returns a list ['Hi!', 'How', '
    are', 'you?']
```

We'll look into some list methods later.

⇒ **You can use TAB-completion in Jupyter to see all methods (and properties) of an object!**

Writing own functions

You can write an own function:

```
def addone(x):
    y = x + 1
    return y
```

Functions take some input ("argument") (in this example, we called it `x`) and *return* some result.

Thus, running

```
addone(5)
```

returns `6`.

Writing own functions

Attention, R users! (maybe obvious for others?)

You *cannot** apply the function that we just created on a whole list – after all, it takes an int, not a list as input.

(wait a sec foruntil we cover for loops later today, but this is how you'd do it (by calling the function for each element in the list separately):):

```
mynumbers = [5, 3, 2, 4]
results = [addone(e) for e in mynumbers]
```

* Technically speaking, you could do this by wrapping the `map` function around your own function, but that's not considered "pythonic". Don't do it ;-)

Modifying lists & dicts

Modifying lists

Let's use one of our first methods! Each *list* has a method `.append()`:

Appending to a list

```
mijnlijst = ["element 1", "element 2"]
anotherone = "element 3" # note that this is a string, not a list!
mijnlijst.append(anotherone)
print(mijnlijst)
```

gives you:

```
["element 1", "element 2", "element 3"]
```

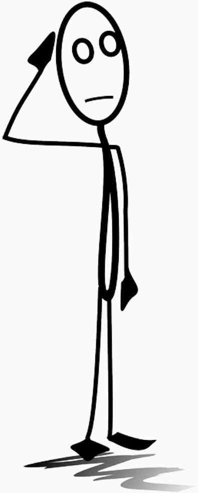
Modifying lists

Merging two lists (= extending)

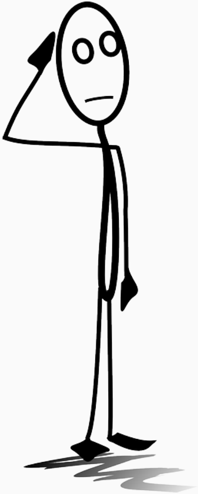
```
mijnlijst = ["element 1", "element 2"]
anotherone = ["element 3", "element 4"]
mijnlist.extend(anotherone)
print(mijnlijst)
```

gives you:

```
["element 1", "element 2", "element 3", "element 4"]
```



What would have happened if we had used `.append()` instead of `.extend()`?



Why do you think that the Python developers implemented `.append()` and `.extend()` as methods of a list and not as functions?

Modifying dicts

Adding a key to a dict (or changing the value of an existing key)

```
mydict = {"whatever": 42, "something": 11}
mydict["somethingelse"] = 76
print(mydict)
```

gives you:

```
{'whatever': 42, 'somethingelse': 76, 'something': 11}
```

If a key already exists, its value is simply replaced.

for, if/elif/else, try/except

How can we structure our program?

If we want to *repeat* a block of code, exectute a block of code only *under specific conditions*, or more generally want to structure our code, we use *indention*.

Indention: The Python way of structuring your program

- Your program is structured by TABs or SPACEs.
- Jupyter (or your IDE) handles (guesses) this for you, but make sure to not interfere and not to mix TABs or SPACEs!
- Default: four spaces per level of indention.

Indention

Structure

A first example of an indented block – in this case, we want to *repeat* this block:

```
agedict = {'Zeus': None, 'Denis': 96, 'Alice': 18, 'Rebecca': 20 , 'Bob
    ': 22, 'Cecile': 45}

myfriends = ['Alice','Bob','Cecile']

print ("The names and ages of my friends:")
for buddy in myfriends:
        print (f"My friend {buddy} is {agedict[buddy]} years old")
```

Output:

```
My friend Alice is 18 years old
My friend Bob is 22 years old
```

What happened here?

```
1 for buddy in myfriends:
2     print (f"My friend {buddy} is {agedict[buddy]} years old")
```

The for loop

1. Take the first element from `myfriends` and call it `buddy` (like `buddy = myfriends[0]`) (line 1)
2. Execute the indented block (line 2, but could be more lines)
3. Go back to line 1, take next element (like `buddy = myfriends[1]`)
4. Execture the indented block ...
5. ...repeat until no elements are left ...

The f-string (*formatted* string)

What happened here?

```
for buddy in myfriends:
    print (f"My friend {buddy} is {agedict[buddy]} years old")
```

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

More in general, the : + indention indicates that

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

Can we also loop over dicts?

Sure! But we need to indicate how exactly:

```
mydict = {"A":100, "B": 60, "C": 30}

for k in mydict: # or mydict.keys()
    print(k)

for v in mydict.values():
    print(v)

for k,v in mydict.items():
    print(f"{k} has the value {v}")
```

Can we also loop over dicts?

The result:

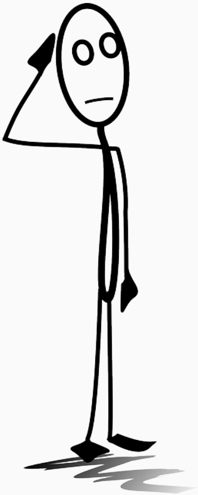
```
A
B
C

100
60
30

A has the value 100
B has the value 60
C has the value 30
```

if statements

Structure

Only execute block if condition is met

```
x = 5
if x <10:
  print(f"{x} is smaller than 10")
elif x > 20:
  print(f"{x} is greater than 20")
else:
  print("No previous condition is met, therefore 10<={x}<=20")
```



Can you see how such an if statement could be particularly useful when nested in a for loop?

try/except

Structure

If executed block fails, run another block instead

```
x = "5"
try:
  myint = int(x)
except:
  myint = 0
```

Again, more useful when executed repeatedly (in a loop or function):

```
mylist = ["5", 3, "whatever", 2.2]
myresults = []
for x in mylist:
    try:
        myresults.append(int(x))
    except:
```

Bonus: Python goodies

List comprehensions

Structure

A for loop that .append()s to an empty list can be replaced by a one-liner:

```
mynumbers = [2,1,6,5]
mysquarednumbers = []
for x in mynumbers:
    mysquarednumbers.append(x**2))
```

is equivalent to:

```
mynumbers = [2,1,6,5]
mysquarednumbers = [x**2 for x in mynumbers]
```

Optionally, we can have a condition:

```
mynumbers = [2,1,6,5]
```

List comprehensions

A very pythonic construct

- Every for loop can also be written as a for loop that appends to a new list to collect the results.
- For very complex operations (e.g., nested for loops), it can be easier to write out the full loops.
- But mostly, list comprehensions are really great! (and much more concise!)

$\Rightarrow$ **You really should learn this!**

Generators

Structure

A lazy for loop (or function) that only generates its next element when it is needed:

You can create a generator just like a list comprehension (but with `()` instead of `[]`):

```
mynumbers = [2,1,6,5]
squaregen = (x**2 for x in mynumbers) # these are NOT calculated yet
for e in squaregen:
    print(e)            # only here, we are calculating the NEXT item
```

Or like a function (but with `yield` instead of `return`):

```
def squaregen(listofnumbers):
    for x in listofnumbers:
        yield(x**2)
```

Generators

A very memory and time efficient construct

- Every function that *returns* a list can also be written as a generator that *yields* the elements of the list
- Especially useful if
    - it takes a long time to calculate the list
    - the list is very large and uses a lot of memory (hi big data!)
    - the elements in the list are fetched from a slow source (a file, a network connection)
    - you don't know whether you actually will need all elements

⇒ **You probably don't need this right now, but (a) it will come in very handy once you deal with web scraping or very**

Make sure you understood all of today's concepts.

Also read Chapter 4 and ask questions if needed. If you want do some exercises with basic python, please see here: https://github.com/annekroon/gesis-machine-learning/blob/main/day1/exercises-basicpython/exercises.md