Python II

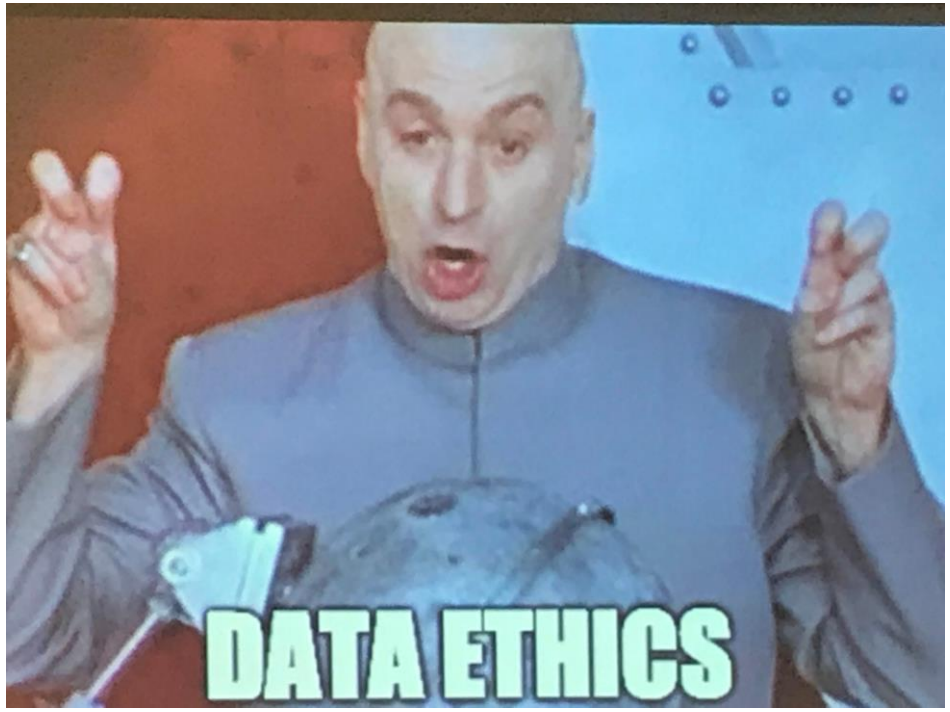
Programmierkurs 2 Data Science WS24/25

Leonard Traeger
M. Sc. Information Systems
leonard.traeger@fh-dortmund.de

Learning Goals Python II

- **Explain** the key concepts of Data Literacy and **demonstrate** its application.
- **List** components/steps of a data science life cycle framework and **outline** characteristics in which models differ in comparison.
- **Outline** the differences between the four container types in Python and **argue** which to use given an example.
- **Demonstrate** how index and values can be accessed, sliced, and iterated over respective containers.
- **Write** and define simple functions with single, multiple, and arbitrary numbers of variables as input.
- **Explain** Map, Filter, and Reduce functionalities and **provide examplarly** use-cases for each.
- **Apply** simple functions on containers via loop and list comprehensions with filtering.

Ethics: should you care?



Ethics (cont.)

A US art installation that will let people control a paintballing robot in a mock art gallery has been condemned by the firm that made the robo-dog.

Boston Dynamics criticised the project, calling it a "provocative use" of its quadruped robot, Spot.

It warned that if the "spectacle" goes ahead, Spot's warranty might be voided, meaning it could not be updated.

The group behind it, MSCHF, argues that Spot or robots like it will probably be used for military applications.

The group is known for creating viral stunts, stories and products.

The project, entitled Spot's Rampage, is due to start at 13:00 EST (18:00 GMT) on Wednesday, and will let people "remotely control a Spot robot" via a website.

https://www.bbc.com/news/technology-56182268

Forbes / Tech

FEB 16, 2012 @ 11:02 AM 2,998,353 VIEWS

How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did

Kashmir Hill
FORBES STAFF
Welcome to The Not-So Private Parts where technology & privacy collide
FULL BIO >

Every time you go shopping, you share intimate details about your consumption patterns with retailers. And many of those retailers are studying those details to figure out what you like, what you need, and which coupons are most likely to make you happy. Target TGT -0.43%, for example, has figured out how to data-mine its way into your womb, to figure out whether you have a baby on the way long before you need to start buying diapers.

Charles Duhigg outlines in the New York Times how Target tries to hook parents-to-be at that crucial moment before they turn into rampant — and loyal — buyers of all things pastel, plastic, and miniature. He talked to Target statistician Andrew Pole — before Target freaked out and cut off all communications — about the clues to a

Target has got you in its aim

http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/#b228dae34c62, Retrieved 6/16/2016

Ethics (cont.)

Ethical issues arise in practical applications, particularly when using **personal data.**

Data Science can lead to **impactful decisions** for **human beings**.

There are many important concerns about **fairness, privacy, security, abuse**...

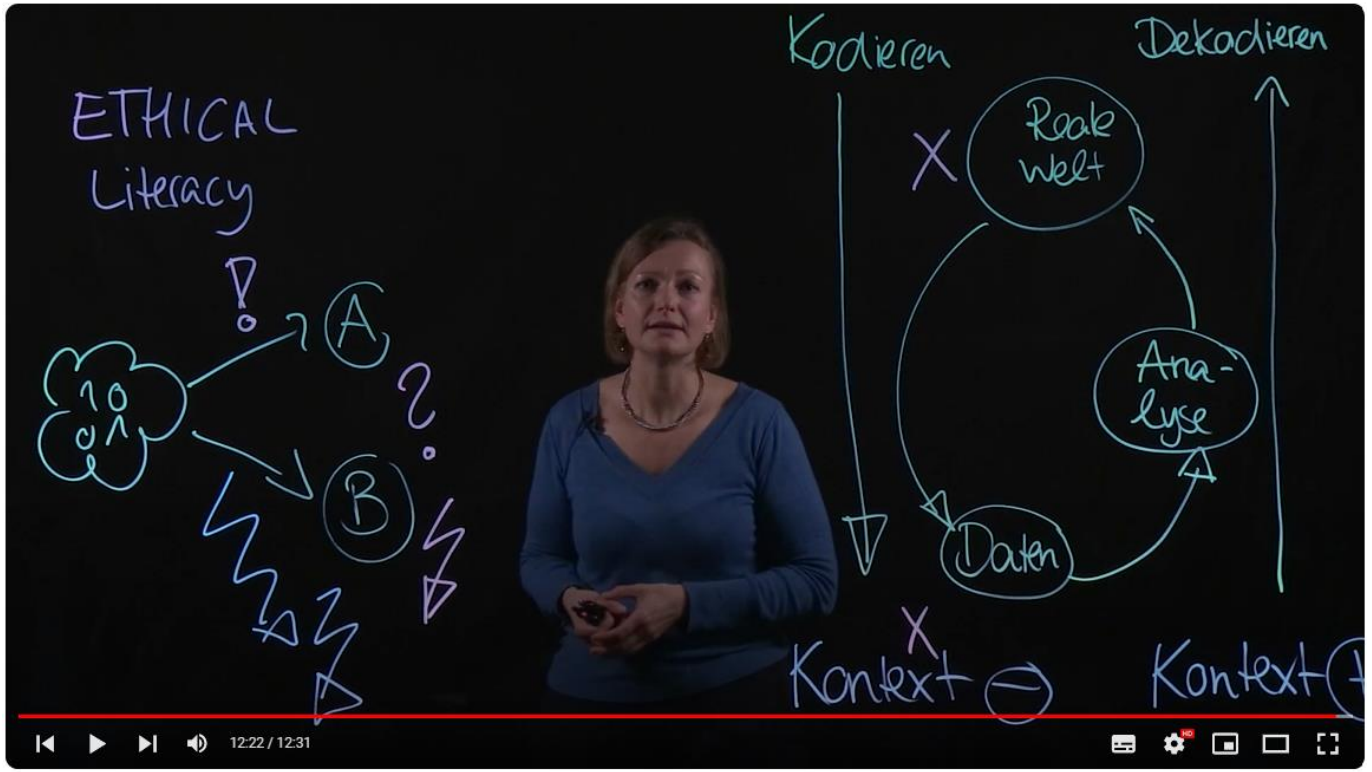
Can‘t we just anonymize the data?

- Anonymization is difficult.
- 85% of Americans can be identified from zip code, birth date and sex.
- The more dimensions / attributes / columns, the quicker you become unique!

Some Common Principles in DS/AI Ethics

- Ensure safety
- Ensure fairness
- Respect privacy
- Promote collaboration
- Provide transparency
- Limit harmful uses
- Establish accountability
- Uphold human rights and values
- Reflect diversity / inclusion
- Avoid concentration of power
- Achknolwedge legal/policy implications

Ethical and Data Literacy



Data Literacy - was ist das? Ein historischer Rückblick mit Katharina Schüller.

Hochschulforum Digitalisierung (HFD)
1370 Abonnenten

Abonnieren

39 Teilen Herunterladen Clip ...

3599 Aufrufe vor 3 Jahren
Das erste Video handelt von der Datenexplosion, die Anfang der 1980er begonnen hat. In einem historischen Rückblick über die letzten 40 Jahre werden die Entwicklungen im Bereich der Daten zusammenfassend dargestellt und aufbereitet. Im Fokus steht vor allem die Entstehung der zahlreichen Literacies, mit besonderem Blick auf die sogenannte "Ethical Literacy". ...mehr

Data Literacy

…is the ability to **collect**, **manage**, **evaluate**, and **apply** data in a **critical way**.

- Competency of the 21st century which enables to act in a digitized world.
- *Systematical approach to turn data into knowledge.*
- *Data Science needs to be **planned** to consciously use and question it in the **respective context**.*

Skill categorization: conceptional (blue), core (green), advanced (red)

Conceptual Framework	Introduction to Data
Data Collection	Data Discovery and Collection
	Evaluating and Ensuring Quality of Data and Sources
Data Management	Data Organization
	Data Manipulation
	Data Conversion
	Metadata Creation and Use
	Data Curation, Security and Re-Use
	Data Preservation
Data Evaluation	Data Tools
	Basic Data Analytics
	Data Interpretation (Understanding Data)
	Identifying Problems Using
	Data Visualization
	Presenting Data (Verbally)
	Data Driven Decisions Making (DDDM)
Data Application	Critical Thinking
	Data Culture
	Data Ethics
	Data Citation
	Data Sharing
	Evaluating Decisions based on Data

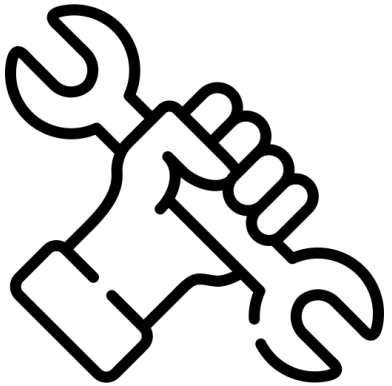
Abbildung 1: Data-Literacy-Kompetenzen nach Ridsdale et al. (2015)

Data Literacy (cont.)

1. What do I want to do with data?
   - Data and its analysis are not an end in themselves (Selbstzweck).
   - *Target a concrete use case or application.*
2. What can I do with data?
   - The technical and methodological possibilities play a crucial role.
   - *Become aware of your capabilities.*
3. What am I allowed to do with data?
   - Legal regulations governing the use of data.
   - *Consider what you are allowed or at least not allowed to do.*
4. What should I do with data?
   - Data is a valuable resource which can create, beyond legally permitted actions, something good for society.
   - *Consider your personal and societal benefit of your application.*

https://ki-campus.org/sites/default/files/2021-10/data-literacy-charta_v1_2.pdf

Think-Pair-Share #1



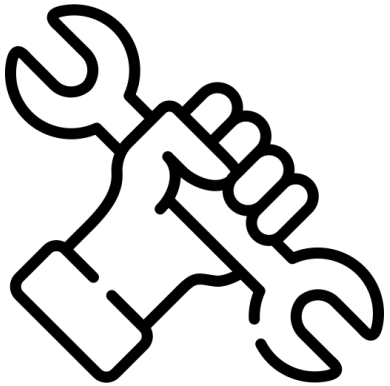
Suppose you have access to a data set from a **large multi-million-$$$ fast-food chain**.

The data set provides information about the personal information of
customers (age, weight, eating behaviour, ...)
purchases (date, price, meals, ...)
meal information (calories, sugar, fat, ...).

Discuss with your neighbouring peers (group of two to three):

1) What do I want to do with data?
2) What can I do with data?
   Imagine your technical and methodological capabilities as a multi-million-$$$ company.
3) What am I allowed to do with data?
4) What should I do with data?

Think-Pair-Share #1



1) What do I want to do with data?
   -
2) What can I do with data?
   -
3) What am I allowed to do with data?
   -
4) What should I do with data?
   -

Comparison and Logical Operators

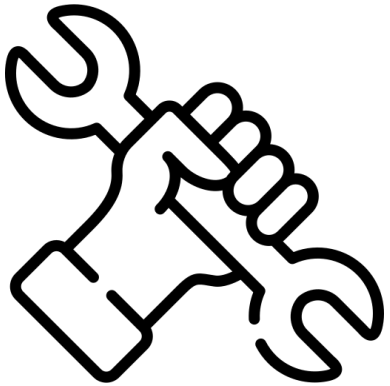
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal
!=	not equal
and	both must be true
or	one or both must be true
not	reverses the truth value

Control Statement

- Example: Test if a is greater than b
- Python‘s `elif` is C‘s `else if`

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
elif a > 201 or not( b > 33):
  print("something else")
else:
  print("a is greater than b")
```

Training #1



1. Generate a random number representing the number of points a student scored in an exam between zero and 116.
   You can import the random int generator with
   ```
   from random import randint as ri
   ```
   and generate random numbers with
   ```
   ri(start_range, end_range).
   ```
2. Write a control statement that outputs a final grade given the randomly generated number. Distinct only between round grades (1,2,3,4, and failed) in this exercise.

Punkte	Note	
116 - 94,9 %	1,0	1
<94,9 - 89,5 %	1,3	
<89,5 - 84,3 %	1,7	
<84,3 - 79,0 %	2,0	2
<79,0 - 73,7 %	2,3	
<73,7 - 68,2 %	2,7	
<68,2 - 63,1 %	3,0	3
<63,1 - 57,9 %	3,3	
<57,9 - 52,6 %	3,7	
<52,6 - 50,0 %	4,0	4
< 50,0 %	n.b.	F

Containers / Collections

- How can we efficiently store data elements in Python?
- How can we efficiently access these?

		Ordered	Changeable	Indexed	Duplicates
List	**[ ]**	Yes	Yes	Yes	Yes
Tuple	**( )**	Yes	No	Yes	Yes
Set	**{ }**	No	Yes	No	No
Dictionary	**{"_:_"}**	No	Yes	Yes	No

player_score[0,1,3,3]

tweet = {'#cat', '#cute', '#lol'}

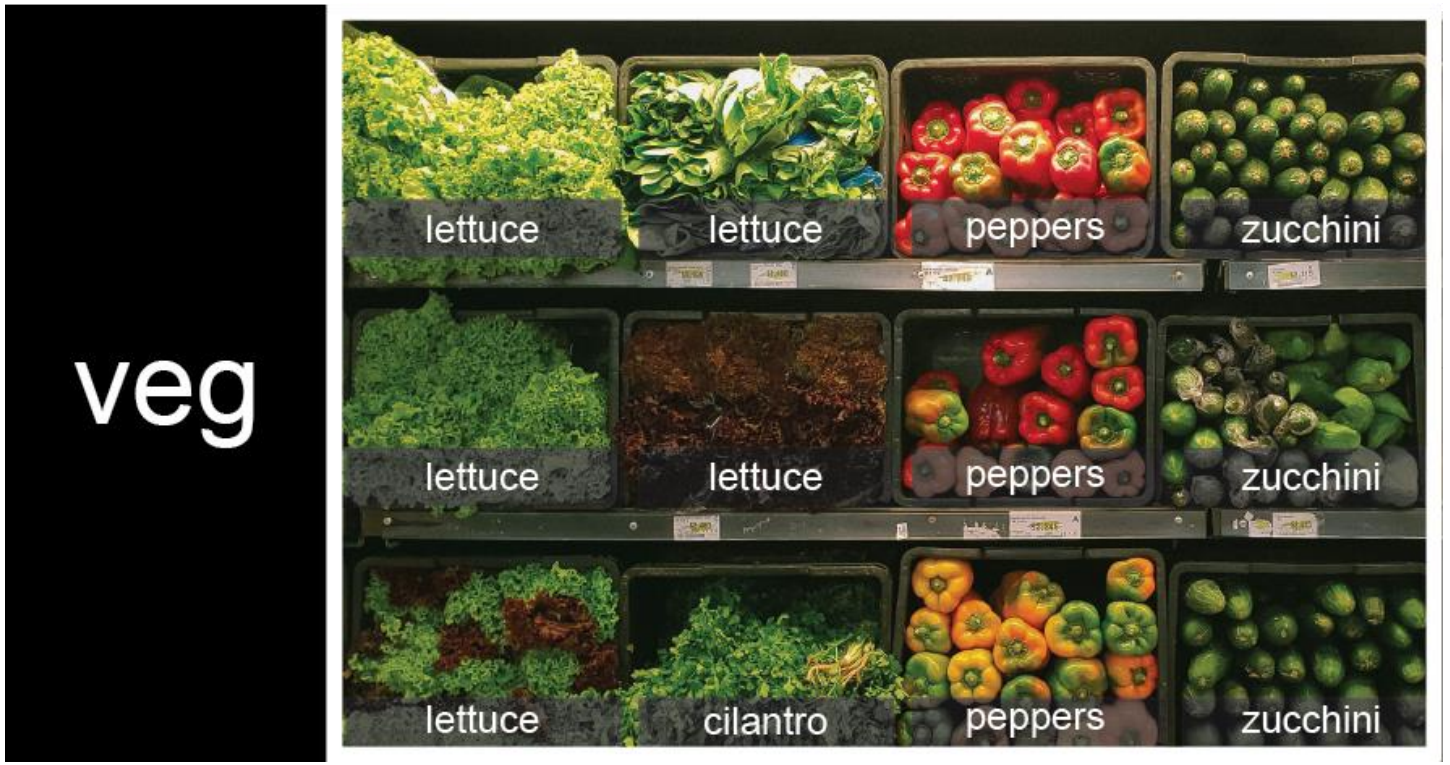
player_score(0,1,3,3)

eng_ger = {'I':'Ich', 'learn':'lernen'}

Python List Example

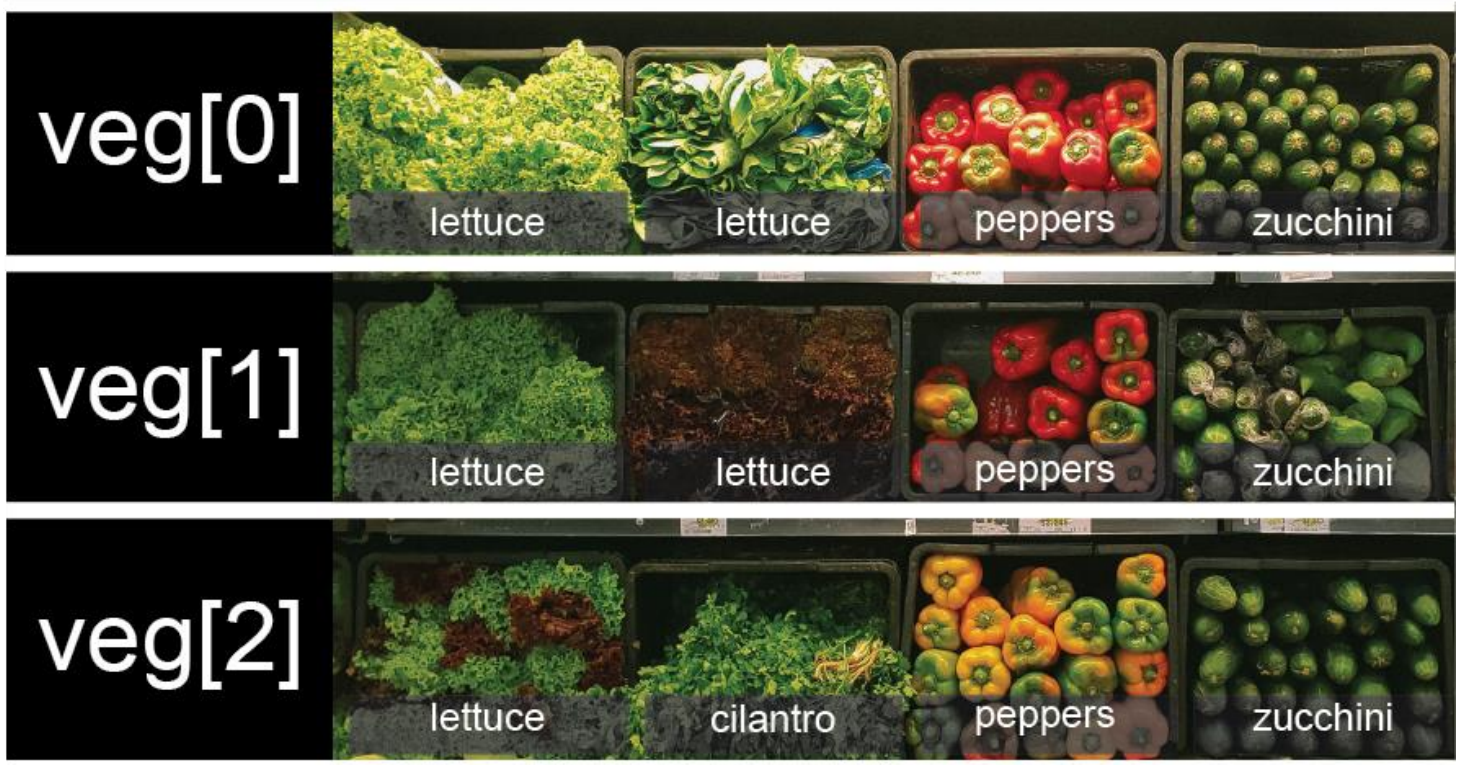
https://swcarpentry.github.io/python-novice-inflammation/04-lists.html

```
veg = [['lettuce', 'lettuce', 'peppers', 'zucchini'],
       ['lettuce', 'lettuce', 'peppers', 'zucchini'],
       ['lettuce', 'cilantro', 'peppers', 'zucchini']]
```



Python List Example (cont.)

```
veg = [['lettuce', 'lettuce', 'peppers', 'zucchini'],
       ['lettuce', 'lettuce', 'peppers', 'zucchini'],
       ['lettuce', 'cilantro', 'peppers', 'zucchini']]
```



Python List Example (cont.)

```
veg = [['lettuce', 'lettuce', 'peppers', 'zucchini'],
       ['lettuce', 'lettuce', 'peppers', 'zucchini'],
       ['lettuce', 'cilantro', 'peppers', 'zucchini']]
```

List [ ]

https://docs.python.org/3.5/tutorial/datastructures.html#more-on-lists

A list is the Python equivalent of an array, but is resizable and can contain elements of different types:

```
x_list = ["Zuckerberg", "Musk", 42]    # Create a list
print(x_list, x_list[2])  # Prints "['Zuckerberg', 'Musk', 42] 42"
print(x_list[-1])     # Negative indices count from the end of the list;
                      # prints "42"
x_list[2] = 'Gates'   # Lists can contain elements of different types
print(x_list)         # Prints "['Zuckerberg', 'Musk', 'Gates']"
x_list.append('Bezos')# Add a new element to the end of the list
print(x_list)         # Prints "['Zuckerberg', 'Musk', 'Gates', 'Bezos']"
x = x_list.pop()      # Remove and return the last element of the list
print(x, x_list)      # Prints "Bezos ['Zuckerberg', 'Musk', 'Gates']"
```

More list methods append(), clear(), copy(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort() to find in documentation.

List [ ] Slicing

Similar use in NumPy arrays

Python provides concise syntax to access sublists (instead one element at the time):

```
nums = list(range(5))     # range is a function creating a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive);
                          # prints "[2, 3]"
print(nums[2:])           # Get a slice from index 2 to the end;
                          # prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive);
                          #  prints "[0, 1]"
print(nums[:])            # Get a slice of the whole list;
                          # prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative;
                          # prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]
```

List [ ] Loop

You can loop over the elements of a list like this:

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
# Prints "cat", "dog", "monkey", each on its own line.
```

If you want access to the **index** of **each element** within the body of a loop, use the built-in **enumerate** function:

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: cat", "#2: dog", "#3: monkey", each on its own line
```

List Comprehensions

- Frequently, we want to transform one type of data into another.

Example: compute price per square apartment (12€ / square)

```
square_meters = [30, 40, 50, 60, 70]
square_prices = []
for square_meter in square_meters:
    square_prices.append(square_meter * 12)
print(square_prices)   # Prints [360, 480, 600, 720, 840]
```

- Make this code simpler using a list comprehension:

```
square_meters = [30, 40, 50, 60, 70]
square_prices = [x * 12 for x in square_meters]
print(square_prices)   # Prints [360, 480, 600, 720, 840]
```

List Comprehensions (cont.)

- Can also contain conditions.

Example: compute price per square apartment for **small** apartments (15€ / square)

```
square_meters = [30, 40, 50, 60, 70]
square_prices = [x * 15 for x in square_meters if x < 51]
print(square_prices)  # Prints "[450, 600, 750]"
```

Tuples ( )

https://docs.python.org/3.5/tutorial/datastructures.html#tuples-and-sequences

- Consists of **heterogenous sequences** and values separated by commas.
- **Unchangeable** ordered list of values.

```
t = ('Zuckerberg', 42, 'Gates', 'Bezos') # Create a tuple
print(type(t))   # Prints "<class 'tuple'>"
print(t.count('Zuckerberg')) # Prints "1"
print(t.index('Zuckerberg')) # Prints "0"
print(t[0])                  # Prints "Zuckerberg"
# Tuples may be nested with heterogeneous sequences:
t2 = t, ('Musk','Cooper')
print(t2)        # Prints "(('Zuckerberg', 42, 'Gates', 'Bezos'),
                 #          ('Musk', 'Cooper'))"
t[1] = 'Musk'    #  TypeError: 'tuple' object does not support item assignment
```

Set { }

https://docs.python.org/3.5/library/stdtypes.html#set

- Unordered collection of distinct elements.

```
s = {'Zuckerberg', 42, 'Gates', 'Bezos'}
print('Zuckerberg' in s)  # Check if an element is in a set; prints "True"
print('Musk' in s)        # prints "False"
s.add('Musk')           # Add an element to a set
print('Musk' in s)      # Prints "True"
print(len(s))           # Number of elements in a set; prints "5"
s.add('Musk')           # Adding an existing element does nothing
print(len(s))           # Prints "5"
s.remove('Musk')        # Remove an element from a set
print(len(s))           # Prints "4"
```

- More set methods add(), clear(), copy(), difference_update(), discard(), issubset(), issuperset(), remove(), union(), update() to find in documentation.

Set { } Loop

- Iterating over a set has the same syntax as iterating over a list (loop, comprehensions).
- Since sets are unordered, you **cannot make assumptions** about the **order** in which you visit the elements of the set.

```
s = {'Zuckerberg', 42, 'Gates', 'Bezos', 'Musk'}
for idx, answer in enumerate(s):
    print('#%d: %s' % (idx + 1, answer))
# Prints
#1: Bezos
#2: Gates
#3: Zuckerberg
#4: Musk
#5: 42
```

Dictionary {"_":"_"}

https://docs.python.org/3.5/library/stdtypes.html#dict

- A dictionary stores (key, value) pairs.
- Duplicate keys will overwrite existing values!

```
d = {'Tsd.': '000', 'Mil.': '000000'}  # Create a new dictionary with some data
print(d['Tsd.'])       # Get an entry from a dictionary; prints "000"
print('Tsd.' in d)     # Check if a dictionary has a given key; prints "True"
d['Bil.'] = '000000000'      # Set an entry in a dictionary
print(d['Bil.'])             # Prints "000000000"
#print(d['Tril.'])           # KeyError: 'Tril.' not a key of d
print(d.get('Tril.', 'N/A')) # Get an element with a default; prints "N/A"
print(d.get('Bil.', 'N/A'))  # Get an element with a default; prints "000000000"
del d['Bil.']                # Remove an element from a dictionary
print(d.get('Bil.', 'N/A'))  # "Bil." is no longer a key; prints "N/A"
```

- More dictionary methods clear(), copy(), fromkeys(), get(), items(), keys(), pop(), popitem(), setdefault(), update(), values() to find in documentation.

Dictionary {"_":"_"} Loop

- You can loop over the keys in a dictionary.

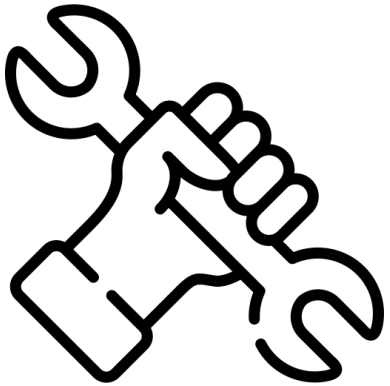
```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

- If you want access to **keys and** their corresponding **values**, use the **items method**:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

Think-Pair-Share #2

		Ordered	Changeable	Indexed	Duplicates
List	**[ ]**	Yes	Yes	Yes	Yes
Tuple	**( )**	Yes	No	Yes	Yes
Set	**{ }**	No	Yes	No	No
Dictionary	**{"_:_"}**	No	Yes	Yes	No



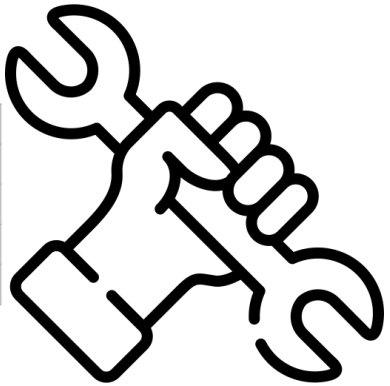
Discuss with your neighbouring peers (groups of two to three):

What container / collection type would suite

- 100 famous dinosaur species?
- All best movie grammy award winners until today?
- Shirt numbers and corresponding player names in a football league?
- Abreviations of all airports and their full name?

Think-Pair-Share #2

		Ordered	Changeable	Indexed	Duplicates
List	**[ ]**	Yes	Yes	Yes	Yes
Tuple	**( )**	Yes	No	Yes	Yes
Set	**{ }**	No	Yes	No	No
Dictionary	**{"_:_"}**	No	Yes	Yes	No



- 100 famous dinosaur species?
  -
- All best movie grammy award winners?
  -
- Shirt numbers and corresponding player names in a large tennis league?
  -
- Abreviations of all airports and their full name?
  -

Functions

Both NumPy and Pandas become very powerful with functions!

- One of the most useful programming constructions.
- Recognize function calls by **( )** after the function name.
- Functions take **0** or **more arguments**.
- Some arguments are required, others have default values and can be omitted.
- Use the `def` keyword to **def**ine a **new function**.

```
def isClassPassed(numberOfPoints):
    if numberOfPoints >= 50:
        return 'passed'
    else:
        return 'failed'
```

Functions (cont.)

- Allow you to name and reuse blocks of code.
- Help you to **break complex problems** into **simpler parts**.
- Make your code more readable.
- Rule of thumb: if you copy-and-paste the same code more than once, it's probably better to encapsulate that code into a function.

```
def isClassPassed(numberOfPoints):
    if numberOfPoints >= 50:
        return 'passed'
    else:
        return 'failed'
```

```
for pointsScored in [55, 12, 66, 1, 99]:
    print(isClassPassed(pointsScored))
#Prints "passed, failed, passed, failed, passed"
```

Functions (cont.)

- Functions can become very complex and nested.
- Documentation essential.

```
#Here's a template from Spyder - use this template every time.
def func(x, y = 0, *z, **j):
    """
    Describe your function's purpose concisely.
    Parameters
    ----------
    x : TYPE  DESCRIPTION.
    y : TYPE, optional DESCRIPTION. The default is 0.
    z : TYPE, arbitrary tuple of arguments.
    (If unkown/changing number of arguments)
    j : TYPE, arbitrary keyword arguments.

    Returns
    ----------

    """
```

Functions & List Comprehensions

```
def isClassPassed(numberOfPoints):
    if numberOfPoints >= 50:
        return 'passed'
    else:
        return 'failed'
```

- Both NumPy and Pandas become powerful with list comprehensions and functions.

```
scoredPoints = [55, 12, 66, 1, 99];
classPassed = [isClassPassed(x) for x in scoredPoints]
print(classPassed)
#Prints "['passed', 'failed', 'passed', 'failed', 'passed']"
```

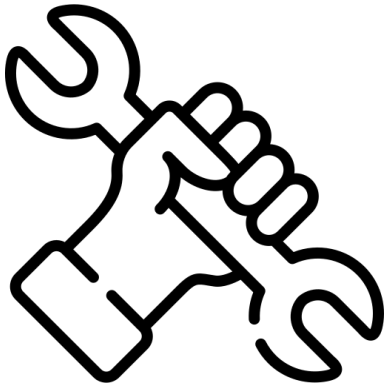
Functions & List Comprehensions & Filtering

- Most “pythonic” element of Python syntax after *Lex Fridman*. https://www.youtube.com/watch?v=belS2Ek4-ow
- Syntax: `[f(x) for x in container if g(x)]`

```
def isClassPassed(numberOfPoints):
    if numberOfPoints >= 50:
        return 'passed'
    else:
        return 'failed'
```

```
scoredPoints = [55, 12, 66, 1, 99, 300, -1];
classPassed = [isClassPassed(x) for x in scoredPoints if x in range(100)]
print(classPassed)
#Prints "['passed', 'failed', 'passed', 'failed', 'passed']"
```

Training #2



1. Generate an array with 100 random numbers representing the number of points students scored between zero and 116 and store these in a suitable container.
   You can import the random int generator with
   ```
   from random import randint as ri
   ```
   and generate random numbers with
   ```
   ri(start_range, end_range).
   ```
2. Write a function that outputs a final grade given scored points as input. Distinct only between round grades (1,2,3,4, and failed) in this exercise.
3. Apply this function to the container storing all points (the 100 students have scored) and store these in a new container.

Punkte	Note	
116 - 94,9 %	1,0	1
<94,9 - 89,5 %	1,3	
<89,5 - 84,3 %	1,7	
<84,3 - 79,0 %	2,0	2
<79,0 - 73,7 %	2,3	
<73,7 - 68,2 %	2,7	
<68,2 - 63,1 %	3,0	3
<63,1 - 57,9 %	3,3	
<57,9 - 52,6 %	3,7	
<52,6 - 50,0 %	4,0	4
< 50,0 %	n.b.	F

Map, Filter, and Reduce

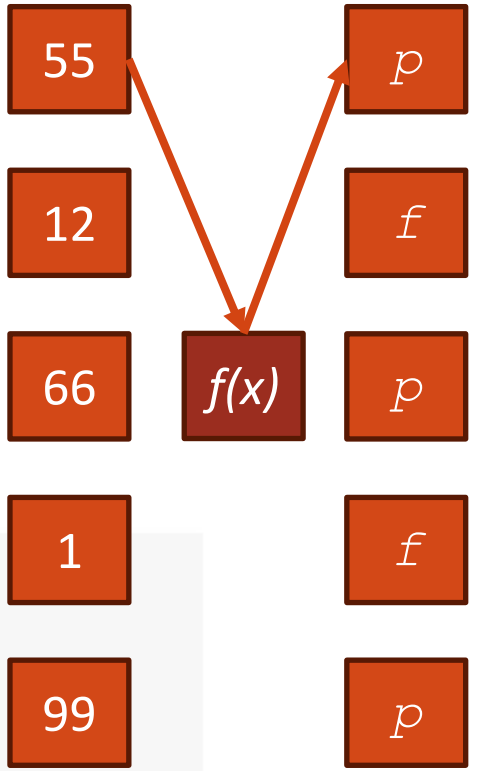
- List comprehensions can fulfill almost all data transformation tasks with functions.
- But, in high degrees of
  - Data volume
  - Distributed storage
  - Parallel computation
  - Fault tolerance
- ...a different parallel computing topology has been invented.

Choose a preference between comprehension list and map+filter+reduce style but understand both!

MapReduce: Simplified Data Processing on Large Clusters by Jeffrey Dean and Sanjay Ghemawat, 2004, Google Inc.

Map `map(func, *iterables)`

- `func` is the function on which each element in `iterables` is applied on.
- The output is a `map object` containing a list.
- To get the result as a list, use the built-in `list( )` function.




```
def isClassPassed(numberOfPoints):
    if numberOfPoints >= 50:
        return 'passed'
    else:
        return 'failed'
```

```
scoredPoints = [55, 12, 66, 1, 99];
classPassed = list(map(isClassPassed, scoredPoints))
print(classPassed)
#Prints "['passed', 'failed', 'passed', 'failed', 'passed']"
```

Map `map(func, *iterables)` (Cont.)

- What did we win? Flexibility and fault-tolerance.
- As the output of the map function is a `map object`.

```
scoredPoints = [55, 12, 66, 1, 99];
classPassed = list(map(isClassPassed, scoredPoints))
print(classPassed)
#Prints "['passed', 'failed', 'passed', 'failed', 'passed']"
```

- We can reuse it cascadingly.

```
scoredPoints = [55, 12, 66, 1, 99];
classPassed2 = list(map(str.upper, map(isClassPassed, scoredPoints)))
print(classPassed2)
#Prints "['PASSED', 'FAILED', 'PASSED', 'FAILED', 'PASSED']"
```

Filter `filter(func, iterable)`

- `func` is the function on which each element in `iterable` is filtered through.
- The output is a `map object` containing a list with only `boolean = true` filtered elements.
- To get the result as a list, use the built-in `list( )` function.

```
def onlyClassPassed(scoredPoints):
  return scoredPoints > 50
```

```
scoredPoints = [55, 12, 66, 1, 99];
over50scoredPoints = list(filter(onlyClassPassed, scoredPoints))
print(over50scoredPoints)
#Prints "[55, 66, 99]"
```

55

12

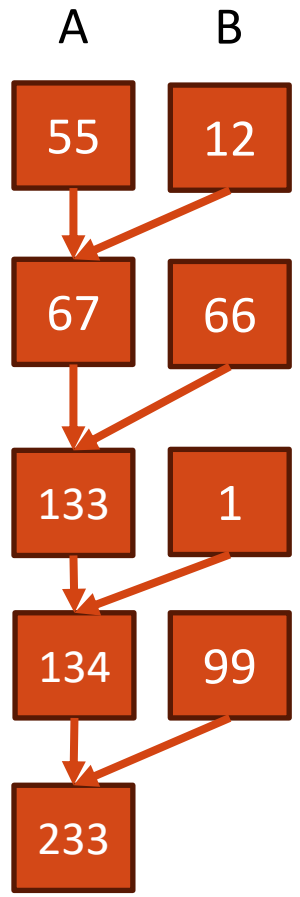
66

1

99

Reduce `reduce(func, iterable[,initial])`

- `func` is the function on which each element in the iterable gets **cumulatively applied.**
- `initial` is the optional value that gets placed before the elements of the iterable in the calculation and serves as a default when the iterable is empty.
- The output is a single value (name source: many values “reduced” to single).



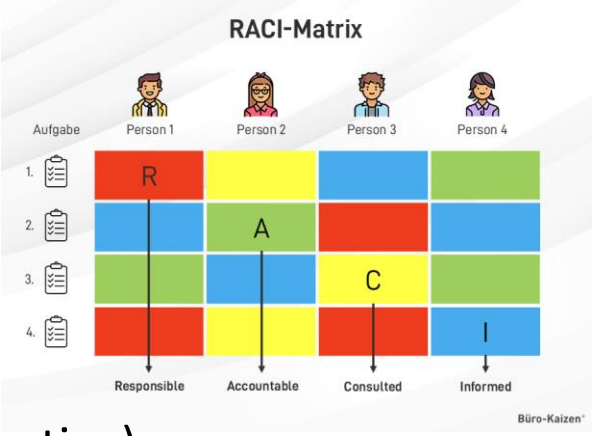

```
def sumScoredPoints(scoredPointsA, scoredPointsB):
  return scoredPointsA + scoredPointsB
```

```
scoredPoints = [55, 12, 66, 1, 99];
print(reduce(sumScoredPoints, scoredPoints))
#Prints "233"
```

Takeaways

- Adapting your project to an existing data science framework leads to a comprehensible and reproducible workflow.
- Follow ethical principles ☺
- Python's readability, easily defined functions, diverse containers, and container loops including list comprehensions make Python as a programming language loved by developers… and hopefully you too.

Project Milestone #1



- Formed teams and a teamname (e.g., HealthyFoodies, Sportineers, ...be creative).
- Declared a domain problem everyone finds interesting in your team and found an official source to justify and motivate others.
- Chosen and summarized a Data Science Life Cycle Framework which you believe would fit your data science project. You roughly estimated in distribution of total 100% which part you want to emphasize on.
- Chosen and summarized data literacy competencies after Ridsdale (2015) you believe will become part in your project. You have planned as a team who is going to be responsible, accountable, consulted, and informed for which skill set.
- Defined collaborative tools and a rough meeting structure so everyone within the team is able to code Python, share, and document files.
- **Planning usually goes wrong but it is important to start somewhere!**

Outlook

- In next week 3 we will dive deep into object-oriented programming with Python.
- Even though we can do a lot with functions and containers, repetitive tasks can be easily outsourced to constructors, destructors, decorator annotated and regular class methods with inheritance.

```
## let's define player1, player2, player3
player1 = Player(player_number = 1, player_name = 'Gregor Kobel',
                 player_value = "35,00 Mio. €")

player2 = Player(player_number = 35, player_name = 'Silas Ostrzinski',
                 player_value = "150 Tsd. €")

player3 = Player(player_number = 1, player_name = 'Marc-André ter Stegen',
                 player_value = "35,00 Mio. €", club_name="FC Barcelona")
```

- We will also peak into common data formats CSV, JSON, and XML and how some of them can help us to generate datasets by ourselves.

See you again next week **online**!

Questions?