



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG



CHAI

Humanities-Centered AI

# Understanding Data vs. Machine Training

**Malte Luttermann – Institute for Humanities-Centered  
Artificial Intelligence (CHAI)**

October 24, 2025

# Overview of Contents

- (1) Programming language Python
  - (a) Introduction and first steps
  - (b) Basics
  - (c) Advanced
- (2) Markup languages
  - (a)  $\text{\LaTeX}$
  - (b) Markdown
- (3) Development environments
  - (a) Jupyter notebooks
- (4) Version control
  - (a) Git and GitHub
- (5) Scientific computing
  - (a) NumPy and SciPy
- (6) Data processing and visualisation
  - (a) Pandas, matplotlib, and NLTK
- (7) Machine learning (scikit-learn)
  - (a) Basics (datasets, analysis)
  - (b) Simple methods (clustering, ...)
- (8) Deep learning
  - (a) PyTorch

# Topics for Today

## (1) Python basics

- Packages and import
- Virtual environments
- Object-orientated programming



# Acknowledgement

- The upcoming slides are taken from the following lecture and have been translated and partially modified:
  - Dr. Magnus Bender: »[Python für Machine Learning und Data Science](#)« as part of the German course »Werkzeuge für das wissenschaftliche Arbeiten«

## Recap: Example from Last Lecture

```
1 def extract_numbers(l):
2     l = l.strip()
3     numbers = []
4     for p in l.split(","):
5         if p.strip().isnumeric():
6             numbers.append(int(p))
7     return numbers
8
9 def build_csv(nl):
10    csv = ""
11    for line in nl:
12        csv += ', '.join([
13            str(n) for n in line
14        ]) + "\n"
15    return csv
```

```
1 f = open("01-Python-Introduction-14.csv", "r")
2 lines = f.readlines()
3 f.close()
4
5 new_lines = []
6 for line in lines:
7     numbers = extract_numbers(line)
8     new_lines.append([n ** 2 for n in numbers])
9
10 print(build_csv(new_lines))
```

```
1 A, Carol, 12, 2045
2 B, Dave, 13, 5689
3 C, Alice, 89, 38594
4 D, Bob, 09, 2830
```

01-Python-Introduction-14.csv

```
$> python3 script.py
```

```
144,4182025
169,32364721
7921,1489496836
81,8008900
```

# Packages and Import

- Some functions are always available
  - E.g., `len()`, `str()`, `int()`, `range()`, `print()`
- Further functions or classes can be loaded via `import`
  - E.g., `import time`, `import sys`
- Custom functions and classes can also be loaded from another file (which is then called a *module*)
- *Packages* contain modules, classes, and functions from third parties, i.e., modules that are neither part of Python nor written by yourself



# Import

File name loaded as a module

```
1 import example_import
2
3 print(example_import)
4 print(example_import.bye, example_import.hello)
5
6 print(example_import.VARIABLE)
7 example_import.bye()
8 example_import.hello()
```

```
def hello():
    print("Hello World!")
3
def bye():
    print("Bye!")
6
7 VARIABLE = False
```

example\_import.py

Access via

`<module-name>.<function-name>`

The module is just a file (assuming files are in the same folder)

# Import

```
1 import example_import
2
3 print(example_import)
4 print(example_import.bye, example_import.hello)
5
6 print(example_import.VARIABLE)
7 example_import.bye()
8 example_import.hello()
```

```
1 def hello():
2     print("Hello World!")
3
4 def bye():
5     print("Bye!")
6
7 VARIABLE = False
```

example\_import.py

```
$> python3 script.py
```

```
<module 'example_import' from './example_import.py'>
<function bye at 0x7fb748078dc0> <function hello at 0x7fb748078d30>
False
Bye!
Hello World!
```

## Import with Custom Module Names

```
1 import example_import as example
2
3 print(example)
4 print(example.bye, example.hello)
5
6 print(example.VARIABLE)
7 example.bye()
8 example.hello()
```

```
1 def hello():
2     print("Hello World!")
3
4 def bye():
5     print("Bye!")
6
7 VARIABLE = False
```

example\_import.py

```
$> python3 script.py
```

```
<module 'example_import' from './example_import.py'>
<function bye at 0x7fb748078dc0> <function hello at 0x7fb748078d30>
False
Bye!
Hello World!
```

## Import with From

```
1 from example_import import hello
2
3 print(hello)
4 hello()
```

Only the function `hello` is imported

`hello` becomes globally available in the script

```
1 def hello():
2     print("Hello World!")
3
4 def bye():
5     print("Bye!")
6
7 VARIABLE = False
```

`example_import.py`

## Import with From

```
1 from example_import import hello
2
3 print(hello)
4 hello()
```

```
$> python3 script.py
```

```
<function hello at 0x7fcdf0078b80>
Hello World!
```

```
1 def hello():
2     print("Hello World!")
3
4 def bye():
5     print("Bye!")
6
7 VARIABLE = False
```

example\_import.py

## Import with From

```
1 import example_import
2 from example_import import hello
3
4 print(example_import.hello)
5 print(hello)
6 hello()
```

```
$> python3 script.py
```

```
<function hello at 0x7ff6900a8b80>
<function hello at 0x7ff6900a8b80>
Hello World!
```

The same function!

```
1 def hello():
2     print("Hello World!")
3
4 def bye():
5     print("Bye!")
6
7 VARIABLE = False
```

example\_import.py

## Import \*

```
1 from example_import import *
2
3 hello()
4 bye()
5 print(VARIABLE)
```

```
$> python3 script.py
```

```
Hello World!
Bye!
False
```

```
1 def hello():
2     print("Hello World!")
3
4 def bye():
5     print("Bye!")
6
7 VARIABLE = False
```

example\_import.py

## Modules in the Standard Library

```
1 import html
2 print(html.escape("<div></div>"))
3
4 import html.entities
5 print(type(html.entities.html5))
```

Why does `import html` work without having a file `html.py` in the same folder?

- `html` is part of Python and contained in the *standard library*
- `entities` is a submodule of `html`

## Modules in the Standard Library

```
1 import html
2 print(html.escape("<div></div>"))
3
4 import html.entities
5 print(type(html.entities.html5))
```

- Python searches for imports in the included standard library in the installation directory
- Modules can be not only files but also folders, which contain files or folders (submodules)
- Submodules are separated by . during import

```
./python3.10
|-- LICENSE.txt
|-- abc.py
...
|-- copy.py
|-- datetime.py
|-- html
|   |-- __init__.py
|   |-- entities.py
|   |-- parser.py
|-- http
|   |-- __init__.py
...
```

## Import and Packages

- Python searches for modules and packages in the *import path* (`sys.path`) during import
  - Standard library
  - Installed packages, e.g., via `pip` (Python package installer)
  - Custom (manually installed, self-written) packages
  - Current directory



### Note

Especially due to many third-party packages, Python often is the first choice for applications in the field of Data Science.

# pip: Python Package Installer

- Installs packages from PyPi (<https://pypi.org/>)
- Use `python3 -m pip` or `pip3` to ensure the correct Python installation is used

```
$> pip3 install numpy
```

Install a single package

```
$> pip3 install numpy==1.23.2
```

Install a specific version

```
$> pip3 install -r requirements.txt
```

Install a list of packages

```
$> pip3 list
```

List installed packages

```
$> pip3 freeze > requirements.txt
```

Write installed packages with version to a file

```
1 gensim
2 nltk
3 numpy==1.23.2
4 pandas==1.4.4
5 scikit-learn==1.1.2
6 scipy==1.9.1
7 sklearn==0.0
```

`requirements.txt`

## Package Versions

```
1 gensim
2 nltk==3.7
3 numpy==1.23.2
4 pandas==1.4.4
5 pdoc==12.1.0
```

requirements1.txt

```
1 gensim==4.2.0
2 nltk==3.7
3 numpy
4 pandas==1.4.4
5 pdoc==11.2.0
```

requirements2.txt

```
1 gensim==4.2.0
2 nltk
3 numpy==1.23.2
4 pandas==1.4.4
5 pdoc==7.1.0
```

requirements3.txt

- Different projects with different dependencies
- Problem: Installing the correct version of each package for every project

# Virtual Environments

- Example for Linux/macOS using bash/zsh
  - More examples at <https://docs.python.org/3/library/venv.html>

```
$> cd ./Project1/  
$> python3 -m venv ./br/>$> source ./bin/activate  
(Project1) $> pip3 install -r requirements.txt  
(Project1) $> python script.py  
(Project1) $> ...  
(Project1) $> deactivate
```

```
./Project1/  
|-- bin  
|   |-- activate  
|   |-- pip3  
|   |-- python  
|-- lib  
|   |-- python3.10  
|       |-- site-packages  
|           |-- numpy  
|           |-- pip  
|-- pyvenv.cfg
```

## Virtual Environments

- Example for Linux/macOS using bash/zsh
  - More examples at <https://docs.python.org/3/library/venv.html>

```
$> cd ./Project1/
```

Change directory to project folder

```
$> python3 -m venv ./
```

Create new virtual environment in current directory

```
$> source ./bin/activate
```

```
(Project1) $> pip3 install -r requirements.txt
```

```
(Project1) $> python script.py
```

```
(Project1) $> ...
```

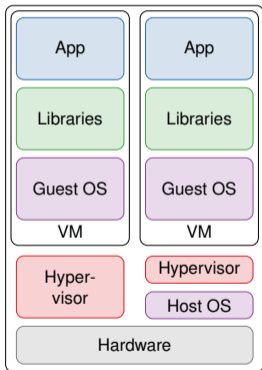
Installation of packages only within the environment

```
(Project1) $> deactivate
```

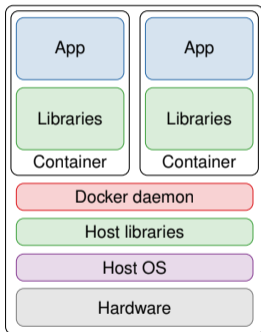
# Virtual Environments

- Virtual environments allow:
  - Different versions of the same (Python) packages
  - Different (Python) dependencies on the same computer
- Problems:
  - Libraries and dependencies of the operating system remain
  - No safeguarding (access to file system, resources, etc.)

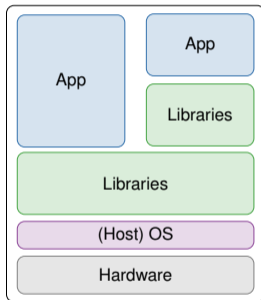
# Excursion: Virtual Machines (VMs) and Containers



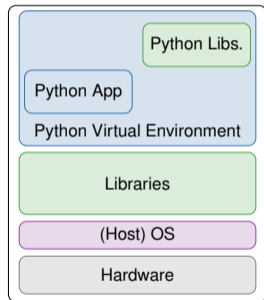
Virtual Machine (VM)



Container (e.g., Docker)



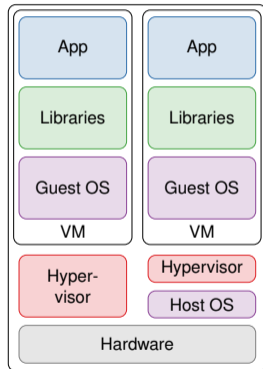
No Isolation



Virtual Environment

## Excursion: Virtual Machines (VMs) and Containers

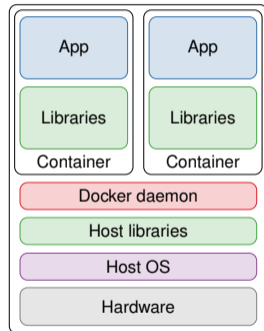
- Programs run in a virtual machine with libraries and its own operating system (OS)
- Virtual machines are fully isolated



Virtual Machine (VM)

## Excursion: Virtual Machines (VMs) and Containers

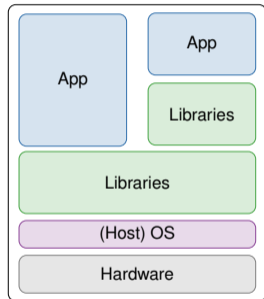
- Programs run in containers, along with their libraries
- Containers are externally isolated



Container (e.g., Docker)

## Excursion: Virtual Machines (VMs) and Containers

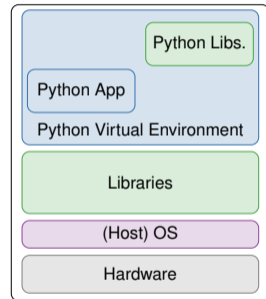
- Programs run directly on the operating system and use own or shared libraries



No Isolation

# Excursion: Virtual Machines (VMs) and Containers

- Within Python, virtual environments isolate installed libraries



Virtual Environment

## Excursion: Virtual Machines (VMs) and Containers

### Virtual machines

- ... provide virtual hardware
- ... require a separate operating system for each VM
- ... induce resource overhead
- ... isolate each VM from host system and other VMs

### Container

- ... offer flexibility of virtual machines
- ... offer almost native performance
- ... isolate each container from host system and other containers

# Container: Docker

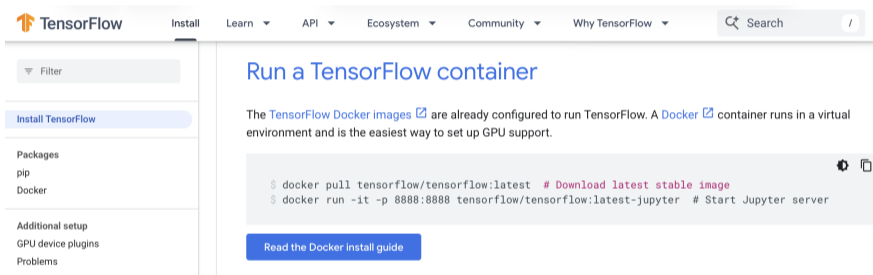


The name »Docker« describes several things:

- **Dockerfile**: Instruction file for creating Docker-Images
- **Docker-Image**: Executable »file system« of a container
- **Docker-Container**: Currently executed Docker-Image
- **Docker-Runtime**: Execution environment for Docker-Containers/-Images
- **Docker-Daemon**: Manages Docker-Containers and -Images, also assigns permissions for network access, file access, and resources
- **Docker-Hub**: External repository for Docker-Images from »Docker Inc.«

# Python and Docker

- (1) Download
- (2) Start
- (3) Try out
- (4) Delete



The screenshot shows the TensorFlow website's 'Install' page. The main heading is 'Run a TensorFlow container'. Below the heading, there is a paragraph explaining that TensorFlow Docker images are pre-configured for GPU support. A code block contains two terminal commands: `$ docker pull tensorflow/tensorflow:latest # Download latest stable image` and `$ docker run -it -p 8888:8888 tensorflow/tensorflow:latest-jupyter # Start Jupyter server`. A blue button labeled 'Read the Docker install guide' is positioned below the code block. The left sidebar of the website lists navigation options like 'Packages', 'Additional setup', and 'Problems'.

## Note

Not only works for a quick start, but also when collaborating with others: Everyone works in the same environment.

## Example: Python with Docker

### ■ Quickly try something out

```
$> docker run -it --rm python:3 python
Unable to find image 'python:3' locally
3: Pulling from library/python
28aec8b14b3e: Pull complete
003e6ed58c0c: Pull complete
390c9631087e: Pull complete
fa24803fc7a7: Pull complete
2e62d2176063: Pull complete
cf6a2b986b53: Pull complete
b7347f592462: Pull complete
Digest: sha256:e3a6ccbe44d9cbfa4f107f238a0e95fa70e0d084e87689222e951d062ac89854
Status: Downloaded newer image for python:3
Python 3.14.0 (main, Oct 8 2025, 23:08:26) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

-it	Interactive
--rm	Remove container afterwards
python:3	Python version 3
python	Start Python

## Example: Python with Docker

### ■ Run a script

```
$> docker run -it --rm  
-v "$PWD":/usr/src/myapp  
-w /usr/src/myapp python:3 python script.py
```

<code>-it</code>	Interactive
<code>--rm</code>	Remove container afterwards
<code>-v \$PWD:/usr/src/myapp</code>	Link current folder into the container (to <code>/usr/src/myapp</code> )
<code>-w /usr/src/myapp</code>	Start container in the folder <code>/usr/src/myapp</code>
<code>python script.py</code>	Run the script <code>script.py</code>

## Example: Dockerfile

- `$> docker build --tag python-docker .`
- `$> docker run --publish 8000:5000 python-docker`

```
1 FROM python:3.8-slim-buster
2
3 WORKDIR /app
4
5 COPY requirements.txt requirements.txt
6 RUN pip3 install -r requirements.txt
7
8 COPY . .
9
10 CMD [ "python3", "-m", "flask", "run", "--host=0.0.0.0" ]
```

Use Debian Buster as a basis

Install packages

Copy code into the image

Define how the code should be started

Dockerfile

# Example: Docker-Compose

```
1 services:
2   app:
3     image: python-docker
4     ports:
5       - "5000:8000"
6     depends_on:
7       - db
8       - redis
9     environment:
10      - ...
11  db:
12    image: mariadb:latest
13    restart: unless-stopped
14    volumes:
15      - /var/lib/mysql
16    environment:
17      - MYSQL_PASSWORD=secret
18      - MYSQL_DATABASE=app
19      - MYSQL_USER=user
20  redis:
21    image: redis:alpine
22    restart: unless-stopped
23    volumes:
24      - ./redis:/data
```

docker-compose.yml

All required components can be specified with their configuration

\$> docker-compose up -d

Pulling db

Pulling redis

Docker-Images are downloaded from DockerHub ...

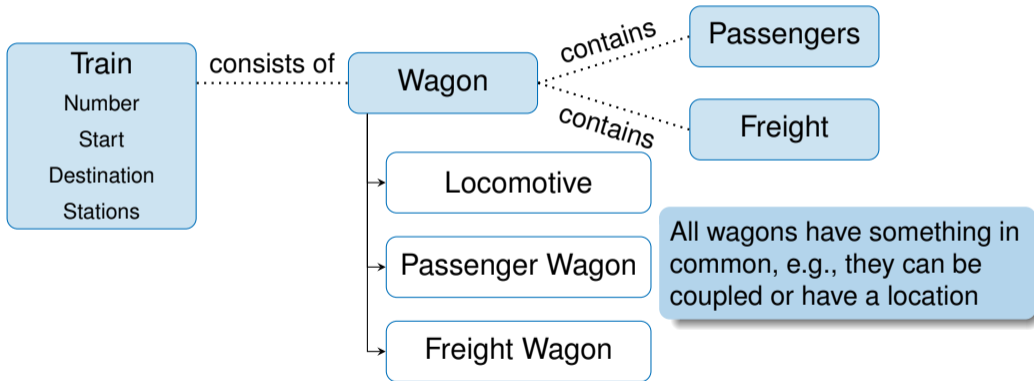
Starting db ... done

Starting redis ... done

Starting app ... done

... and started afterwards

# Object-Orientated Programming (OOP)



# Classes and Objects

Train  
Number  
Start  
Destination  
Stations

```
1 class Train():
2     def __init__(self, n, s, d):
3         self.number = n
4         self.start = s
5         self.destination = d
6         self.stations = []
7
8     def add_station(self, s):
9         self.stations.append(s)
10
11 hl_hh = Train("RE 8", "Lubeck Hbf", "Hamburg Hbf")
12 hl_hh.add_station("Lubeck Hbf")
13 hl_hh.add_station("Reinfeld (Holstein)")
14 hl_hh.add_station("Bad Oldesloe")
15 hl_hh.add_station("Hamburg Hbf")
16
17 print(hl_hh)
18 print(hl_hh.stations)
```

```
$> python3 script.py
```

```
<__main__.Train object at
0x7fd3200f6e50>
['Lubeck Hbf', 'Reinfeld (Holstein)',
'Bad Oldesloe', 'Hamburg Hbf']
```

# Classes and Objects

```
1 class Train():
2     def __init__(self, n, s, d):
3         self.number = n
4         self.start = s
5         self.destination = d
6         self.stations = []
7
8     def add_station(self, s):
9         self.stations.append(s)
10
11 hl_hh = Train("RE 8", "Lubeck Hbf", "Hamburg Hbf")
12 hl_hh.add_station("Lubeck Hbf")
13 hl_hh.add_station("Reinfeld (Holstein)")
14 hl_hh.add_station("Bad Oldesloe")
15 hl_hh.add_station("Hamburg Hbf")
16
17 print(hl_hh)
18 print(hl_hh.stations)
```

Class with attributes  
and methods

Create an object of the  
class `Train`

Call a method (a function of the object)

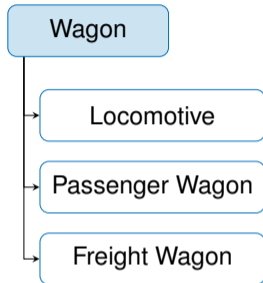
## Classes and Objects

- A class bundles functionality (methods) and values (attributes) of a »type«
- An object is an instance of a class
  - Attributes (variables of the class) are individual for each object
  - Methods (functions of the class) are the same for all objects
- There is no encapsulation via `private`, `protected`, ... as in, e.g., Java
- Everything is internally accessible via `self` and externally via the object
- Convention: Names of internal attributes and functions start with `_`

# Superclasses and Subclasses

```
1 class Wagon():  
2     def __init__(self, p, n):  
3         self.previous_wagon = p  
4         self.next_wagon = n
```

```
1 class FreightWagon(Wagon):  
2     SELF_POWERED = False  
  
3  
4     def __init__(self, *args):  
5         super().__init__(*args)  
6         self.freight = []  
7  
8     def add_freight(self, g):  
9         self.freight.append(g)  
10  
11 class PersonWagon(Wagon):  
12     SELF_POWERED = True  
13  
14     def enter_person(self, p):  
15         pass  
16  
17 class Locomotive(Wagon):  
18     SELF_POWERED = True
```



# Superclasses and Subclasses

```
1 class Wagon():  
2     def __init__(self, p, n):  
3         self.previous_wagon = p  
4         self.next_wagon = n
```

Each wagon has a predecessor and a successor

`pass` is used as a placeholder for a method that is not yet implemented

```
1 class FreightWagon(Wagon):  
2     SELF_POWERED = False  
  
3  
4     def __init__(self, *args):  
5         super().__init__(*args)  
6         self.freight = []  
  
7  
8     def add_freight(self, g):  
9         self.freight.append(g)  
  
10  
11 class PersonWagon(Wagon):  
12     SELF_POWERED = True  
  
13  
14     def enter_person(self, p):  
15         pass  
  
16  
17 class Locomotive(Wagon):  
18     SELF_POWERED = True
```

A freight wagon is a wagon and has all attributes and methods of a wagon

A freight wagon is not self-powered (which holds for all freight wagon objects, i.e., `SELF_POWERED` is a static attribute)

`super()` references the superclass and `__init__()` calls its constructor

`*args` takes all arguments as a tuple

# Superclasses and Subclasses

```
1 w = Wagon("P", "N")
2 print(w.previous_wagon, w.next_wagon)
3
4 fw = FreightWagon("a", "c")
5 print(fw.previous_wagon,
6       fw.next_wagon, fw.freight)
7 fw.add_freight("Metal")
8 fw.add_freight("Silver")
9 print(fw.previous_wagon,
10       fw.next_wagon, fw.freight)
11
12 pw = PersonWagon("b", "d")
13 pw.enter_person("Malte")
14
15 l = Locomotive(None, "a")
16 print(l.previous_wagon, l.next_wagon)
17 print(l.SELF_POWERED,
18       Locomotive.SELF_POWERED)
```

```
$> python3 script.py
```

```
P N
a c []
a c ['Metal', 'Silver']
None a
True True
```

## Back to the Train

Train

consists of

Wagon

contains

Freight

```
1 l = Locomotive(None)
2 fw1 = FreightWagon(l)
3 fw1.add_freight("Grain")
4 fw2 = FreightWagon(fw1)
5 fw2.add_freight("Oats")
6 fw2.add_freight("Oatmeal")
7 fw3 = FreightWagon(fw2)
8
9 t = Train("FW 12", "Luebeck", "Muenchen")
10 t.add_station("Hannover")
11 t.add_station("Kassel")
12 t.add_station("Regensburg")
13 t.set_wagons([l, fw1, fw2, fw3])
14
15 print("Train length", len(t))
16 print(t)
17 print("Wagon 2 with", fw2.freight)
```

Train »FW 12«

Wagon: Locomotive

Wagon: Freight Wagon 1  
Grain

Wagon: Freight Wagon 2  
Oats  
Oatmeal

Wagon: Freight Wagon 3

## Back to the Train

Train

consists of

Wagon

contains

Freight

```
1 l = Locomotive(None)
2 fw1 = FreightWagon(l)
3 fw1.add_freight("Grain")
4 fw2 = FreightWagon(fw1)
5 fw2.add_freight("Oats")
6 fw2.add_freight("Oatmeal")
7 fw3 = FreightWagon(fw2)
8
9 t = Train("FW 12", "Luebeck", "Muenchen")
10 t.add_station("Hannover")
11 t.add_station("Kassel")
12 t.add_station("Regensburg")
13 t.set_wagons([l, fw1, fw2, fw3])
14
15 print("Train length", len(t))
16 print(t)
17 print("Wagon 2 with", fw2.freight)
```

```
$> python3 script.py
```

Train length 4

Train FW 12 from Luebeck to Muenchen

via Hannover, Kassel, Regensburg

with 4 wagons [Locomotive, FreightWagon,

FreightWagon, FreightWagon]

Wagon 2 with ['Oats', 'Oatmeal']

Formatting the output of  
an object will be part of  
the next lecture!

# Object-Oriented Programming with Python

- Python supports multiple inheritance (multiple superclasses)
  - `class MyClass(Class1, Class2)`
- Static methods (analogous to static attributes) have no `self` in their definition
  - `def my_static_method(a, b):`
  - `MyClass.my_static_method(a, b)`
- Objects and classes have some internal attributes
  - `my_object.__class__` (class of the object)
  - `my_object.__class__.__name__` (name of the class as `str`)
  - `my_object.__dict__` (dictionary with all attributes)
  - `my_object.__class__.__dict__` (dictionary with methods and static attributes)

# Summary

- Packages and import
- Virtual environments and Docker
- Object-oriented programming

