

# Lecture 4

## Structuring Python projects



**Intro questions:**

**[pwa.klicker.uzh.ch/join/asaeub](https://pwa.klicker.uzh.ch/join/asaeub)**

# Learning objectives

By the end of this lecture, you should:

- Understand what a “package” is in Python and how to create one
- Know how to manage dependencies in Python
- Know how to collaborate using platforms like GitHub or GitLab
- Know about different tools for maintaining Python code
- Understand what containerization is and why it is useful

# Teaser

Documentation

Code

Tests

What does all of  
this mean?! 🤔

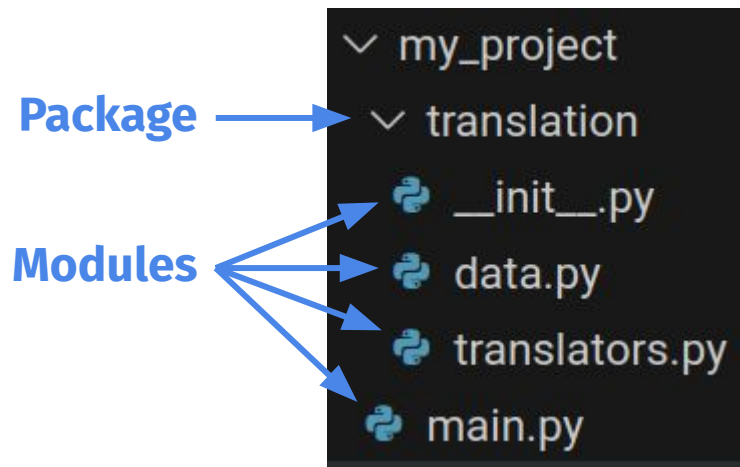
pymovements Public			Watch 2
main	16 Branches	30 Tags	Go to file t Add file <> Code
theDebbister feat: parse eyelink metadata (#674) ✓			b097c93 · 5 days ago 462 Commits
📁 .github	ci: update github action versions (#685)	5 days ago	
📁 docs	ci: pre-commit autoupdate (#672)	3 weeks ago	
📁 src/pymovements	feat: parse eyelink metadata (#674)	5 days ago	
📁 tests	feat: parse eyelink metadata (#674)	5 days ago	
📄 .gitattributes	feat: Add __version__ attribute (#469)	6 months ago	
📄 .gitignore	Git: Ignore notebook checkpoints (#287)	last year	
📄 .pre-commit-config.yaml	ci: pre-commit autoupdate (#681)	last week	
📄 .readthedocs.yaml	Remove requirements_doc.txt and add .readthedocs.ya...	2 years ago	
📄 CONTRIBUTING.md	update contributing.md for new tox command (#684)	last week	
📄 LICENSE.txt	License: Update year (#223)	last year	
📄 README.md	docs: include teaser in readme (#639)	2 months ago	
📄 pylintrc	build: drop support for python 3.8 (#572)	5 months ago	
📄 pyproject.toml	add typing_extensions to be able to install pymovements...	5 days ago	
📄 requirements.txt	docs: Update requirements.txt to reflect location of depe...	6 months ago	
📄 tox.ini	ci: improve tox integration command (#693)	5 days ago	

# Packaging

# What is a package?

A **package** is a collection of modules.

- A **module** is a **file** ending in `.py` containing Python code.
- A **package** is a **directory** containing a file `__init__.py` and any number of modules and packages.



# Import statements

```
# main.py:
```

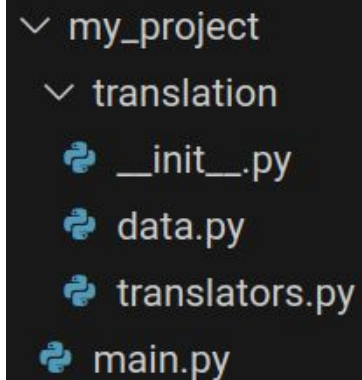
```
import translation.data
```

```
translation.data.ParallelDataset(["hello"], ["bonjour"])
```

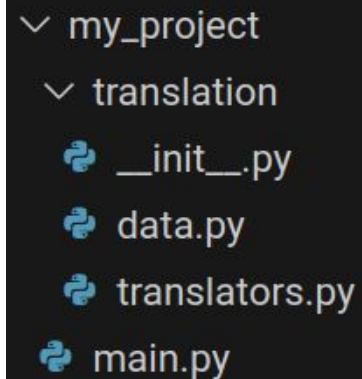
```
from translation.data import ParallelDataset
```

```
ParallelDataset(["hello"], ["bonjour"])
```

```
import translation    # Imports __init__.py
```



# Incorrect import statements



```
import translation.data.ParallelDataset ❌
```

→ **ModuleNotFoundError: No module named 'translation.data.ParallelDataset'**

```
import translation  
translation.data.ParallelDataset(["hello"], ["bonjour"]) ❌
```

→ **AttributeError: module 'translation' has no attribute 'data'**

## `__init__.py`

- When we import a package (instead of a module), the package's **`__init__.py`** is called and imported.
- `__init__.py` can be empty (e.g., if the user is not supposed to import the package, but only the modules inside it).
- Often, `__init__.py` will import the most commonly used modules and objects to provide easier access.



# Shorter import statements

```
# __init__.py
```

```
from translation.data import ParallelDataset  
from translation.translators import WordByWordTranslator, GoogleTranslator
```

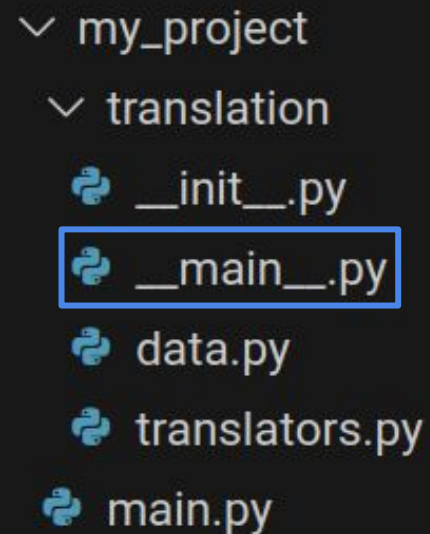
```
# main.py
```

```
from translation import ParallelDataset  
from translation import WordByWordTranslator  
from translation import GoogleTranslator
```

```
# Instead of: from translation.data import ...
```

## \_\_main\_\_.py

- Remember: Modules are executable by default.  
(using `if __name__ == "__main__":` is good practice)
- Packages can also be executable, if they contain a **\_\_main\_\_.py** module.
- Running **`python -m translation`** will execute `translation/__main__.py`.
- Many packages you already know are executable:  
`python -m pip`  
`python -m venv`  
`python -m pytest`



# Making packages installable with pip

Why would you want to make your own packages installable?

- To be able to **import your packages** from anywhere on your computer.
- To be able to **run your scripts** from anywhere on your computer.
- To make your code **easier to use for other people**.

→ We want to do

```
$ pip install translation
```

and then use

```
>>> import translation
```

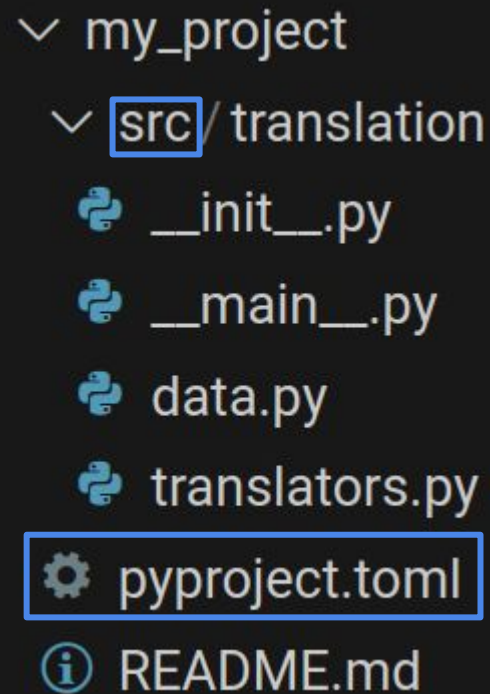
from anywhere on our machine

# pyproject.toml

```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[project]
name = "my_project"
version = "0.0.1"
authors = [
    { name="Andreas Säuberli", email="andreas@cl.uzh.ch" },
    { name="Lukas Fischer", email="fischerl@cl.uzh.ch" },
]
description = "A project to demonstrate packaging in Python"
readme = "README.md"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]
requires-python = "≥3.9"
dependencies = [
    "googletrans==4.0.0rc1",
    "sacrebleu ≥ 2.4.0, <3",
]

[tool.hatch.build.targets.wheel]
packages = ["src/translation"]
```




# pyproject.toml


```
[build-system]  
requires = ["hatchling"]  
build-backend = "hatchling.build"
```


```
[project]  
name = "my_project"  
version = "0.0.1"  
authors = [  
    { name="Andreas Säuberli", email="andreas@cl.uzh.ch" },  
    { name="Lukas Fischer", email="fischerl@cl.uzh.ch" },  
]  
description = "A project to demonstrate packaging in Python"  
readme = "README.md"  
classifiers = [  
    "Programming Language :: Python :: 3",  
    "License :: OSI Approved :: MIT License",  
    "Operating System :: OS Independent",  
]  
requires-python = "≥3.9"  
dependencies = [  
    "googletrans==4.0.0rc1",  
    "sacrebleu ≥ 2.4.0, < 3",  
]  
  
[tool.hatch.build.targets.wheel]  
packages = ["src/translation"]
```

✓ my\_project


✓ src / translation

 \_\_init\_\_.py

 \_\_main\_\_.py

 data.py

 translators.py

 pyproject.toml

 README.md

# pyproject.toml


```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"


[project]
name = "my_project"
version = "0.0.1"
authors = [
    { name="Andreas Säuberli", email="andreas@cl.uzh.ch" },
    { name="Lukas Fischer", email="fischerl@cl.uzh.ch" },
]
description = "A project to demonstrate packaging in Python"
readme = "README.md"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]
requires-python = "≥ 3.9"
dependencies = [
    "googletrans==4.0.0rc1",
    "sacrebleu ≥ 2.4.0, < 3",
]


[tool.hatch.build.targets.wheel]
packages = ["src/translation"]
```

✓ my\_project

✓ src / translation


 \_\_init\_\_.py

 \_\_main\_\_.py

 data.py

 translators.py

 pyproject.toml

 README.md

# pyproject.toml


```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"


[project]
name = "my_project"
version = "0.0.1"
authors = [
    { name="Andreas Säuberli", email="andreas@cl.uzh.ch" },
    { name="Lukas Fischer", email="fischerl@cl.uzh.ch" },
]
description = "A project to demonstrate packaging in Python"
readme = "README.md"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]
requires-python = " ≥ 3.9"
dependencies = [
    "googletrans=4.0.0rc1",
    "sacrebleu ≥ 2.4.0, <3",
]


[tool.hatch.build.targets.wheel]
packages = ["src/translation"]
```

✓ my\_project


✓ src / translation

 \_\_init\_\_.py

 \_\_main\_\_.py

 data.py

 translators.py

 **pyproject.toml**

 README.md

# pyproject.toml


```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"


[project]
name = "my_project"
version = "0.0.1"
authors = [
    { name="Andreas Säuberli", email="andreas@cl.uzh.ch" },
    { name="Lukas Fischer", email="fischerl@cl.uzh.ch" },
]
description = "A project to demonstrate packaging in Python"
readme = "README.md"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]
requires-python = "≥3.9"
dependencies = [
    "googletrans==4.0.0rc1",
    "sacrebleu≥2.4.0,<3",
]
```


**[tool.hatch.build.targets.wheel]**  
**packages = ["src/translation"]**


✓ my\_project


✓ src / translation

 \_\_init\_\_.py

 \_\_main\_\_.py

 data.py

 translators.py

 **pyproject.toml**

 README.md



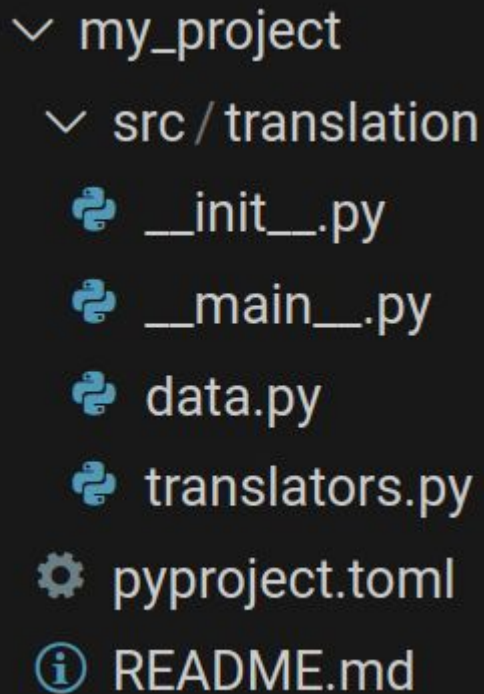
# Installing with pip

Install your package as it looks right now (“**frozen**”):

```
$ cd my_project  
$ pip install .
```

Install your package and keep the installation up-to-date as you change the code (“**editable**”; useful during development):

```
$ cd my_project  
$ pip install -e .
```



# Installing with pip

After installing, we can use:

- `$ python -m translation`
- `>>> import translation`
- `pip uninstall my_project`



# The Python Package Index (PyPI)

- [pypi.org](https://pypi.org) is the platform where Python packages are usually published.
- By default, `pip install <package-name>` will search for `<package-name>` on PyPI.
- Publishing packages on PyPI is free and easy:  

```
$ pip install build twine  
$ python -m build  
$ twine upload dist/*
```
- For testing purposes, there is a separate index called TestPyPI ([test.pypi.org](https://test.pypi.org); try it out in Exercise 3!):  

```
$ twine upload --repository testpypi dist/*
```

# Further reading on packaging

User guide: [packaging.python.org](https://packaging.python.org)

- [Packaging Python Projects](#) ← highly recommended!
- [The Packaging Flow](#)
- [Writing your pyproject.toml](#) ([full example](#))
- [Using TestPyPI](#) (for publishing your package)

# Versioning and dependency management

# Semantic Versioning

Given a version number **MAJOR.MINOR.PATCH**, increment the:

- **MAJOR** version when you make **incompatible** API changes
- **MINOR** version when you add functionality in a **backward compatible** manner
- **PATCH** version when you make **backward compatible** bug fixes

([semver.org](https://semver.org))

Example: version **1.3.4** (major = **1**, minor = **3**, patch = **4**)

# Semantic Versioning: backward compatibility

If I write some code using **spaCy version 3.7.4**, ...

- ... my code *should* still work with spaCy versions **3.7.5**, **3.8.0**, etc.  
→ backward compatibility
- ... my code *may* stop working with spaCy version **4.0.0**.  
→ breaking changes
- ... my code should also work with spaCy version 3.7.0, but there may be bugs or security vulnerabilities.

# Specifying dependency versions

Dependency specification	Meaning
spacy	"I don't care which version." ⚠ DANGEROUS! ⚠
spacy == 3.7.4	"I need <i>exactly</i> version 3.7.4."
spacy >= 3.7.0	"I need <i>at least</i> version 3.7.0."
spacy >= 3.7, < 4	"I need <i>at least</i> version 3.7.0, but lower than 4.0.0."
spacy ~= 3.7.4	"I need any version that is compatible with 3.7.4."

- If the specified range is not compatible with all other dependencies, installation will fail.
- If there are multiple compatible options, the most recent version will be installed.



# How to define Python dependencies?

## **pyproject.toml:**

```
requires-python = ">= 3.9"

dependencies = [
    "googletrans == 4.0.0rc1",
    "sacrebleu >= 2.4.0, < 3",
    "spacy ~= 3.7.4",
]
```

## **requirements.txt:**

```
googletrans == 4.0.0rc1
sacrebleu >= 2.4.0, < 3
spacy ~= 3.7.4
```

# How to define Python dependencies?

- If you're writing a library that other people will use, ...
- If reproducibility isn't important, ...
- If you want to avoid compatibility issues with other libraries, ...
- If you want to restrict the Python version, ...

... use **pyproject.toml** and define dependencies with a **range of compatible versions**, for example: `spacy>=3.7.0,<4`

To install the dependencies, install your package with **`pip install .`**

- If your code is for one-time use (e.g., data analysis or ML experiments), ...
- If you're deploying your code (e.g., as a web app), ...
- If reproducibility is important, ...
- If there is no risk of compatibility issues, ...

... use **requirements.txt** and define dependencies with exact versions, for example: `spacy==3.7.4`

To install the dependencies, use **`pip install -r requirements.txt`**

# Quiz

[pwa.klicker.uzh.ch/join/asaeub](https://pwa.klicker.uzh.ch/join/asaeub)



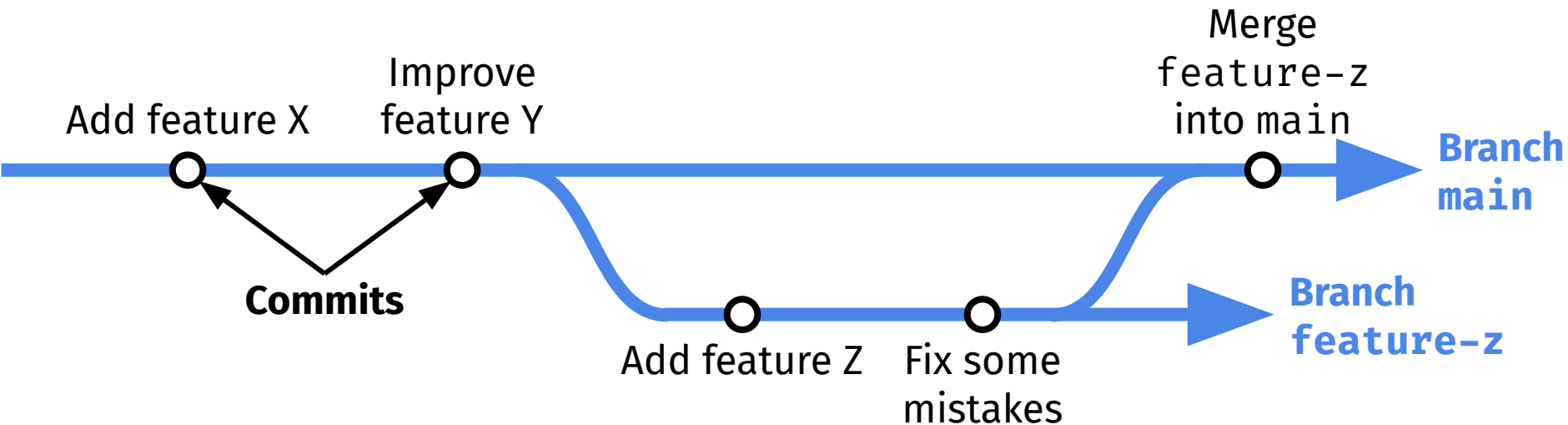
# Open source

# Git repositories for collaboration

Remember:

- **Git** is a version control software running on your computer.
- **GitHub** and **GitLab** are popular platforms for hosting open-source projects online.

# Git repositories for collaboration



# How to contribute to open-source projects

1. **Fork** the repository you want to contribute to.
2. **Clone** your forked repository.
3. Make and test your changes locally. **Commit** as frequently as you wish.
4. When you are done, **push** your changes to your forked repository.
5. Create a **pull request** (sometimes called “merge request”) explaining your changes and asking the maintainers to accept (“pull”) your changes into the main repository.
6. The maintainers will **review** and (hopefully) **merge** your changes into the main repository.

# CI/CD

## **Continuous Integration (CI):**

- Continuously and frequently **merge small changes** into the main branch.
- Developers can work independently and without complex merge conflicts.

## **Continuous delivery (CD):**

- Continuously and frequently **release new versions**.
- More immediate testing and delivery of updates, less risk in releasing.



# CI/CD and DevOps

- For CI/CD to work efficiently in large software projects, we need to continuously and automatically **check and improve code quality**.
- **DevOps** (development+operations) attempts to optimize CI/CD using automation workflows.
- Automated **CI/CD pipelines** can be used to ...
  - ... run unit tests and report coverage after each commit
  - ... prevent type errors
  - ... check code style and detect [“code smells”](#)
  - ... standardize code formatting
  - ... package and deploy code at each release
  - ... generate release notes
  - ...

## Example: CI/CD pipeline definition (from Exercise 1)

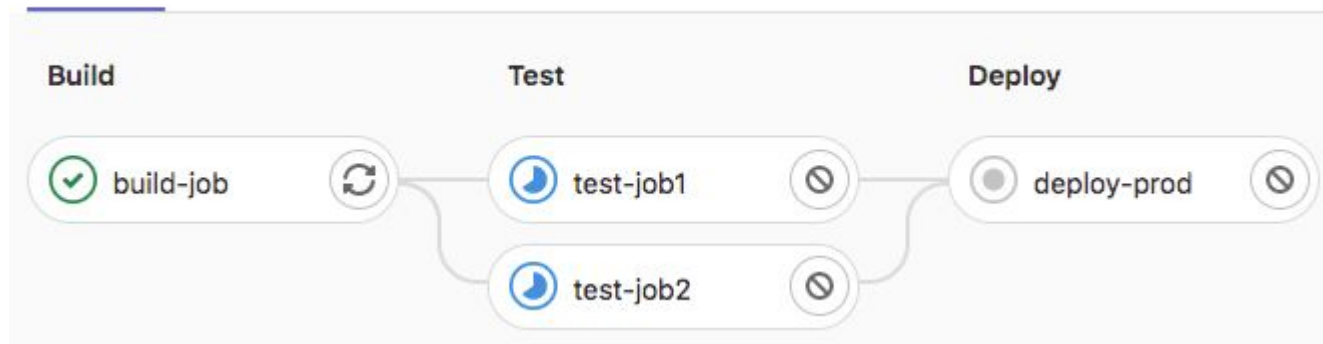
```
stages:  
  - test
```

```
pytest:  
  stage: test  
  image: python:3.12  
  script:  
    - pip install nltk  
    - python -m nltk.downloader stopwords  
    - pip install pytest=7.4.3  
    - pip install coverage pytest-cov  
    - coverage run --source=. --omit='*/test_*.py,*_test.py,*__init__.py' -m pytest  
    - coverage report -m
```

```
coverage: /(?i)total.*? (100(?:\.\d+)?\%|[1-9]?[0-9]\d(?:\.\d+)?\%)\$/
```

Status	Pipeline	Triggerer	Commit	Stages
 running	#217408060 latest		 master → 271ad0cd  Update .gitlab-ci.yml	  

Pipeline Needs Jobs 4 Tests 0



# Code licenses

Licenses define what other people are allowed to do with your code (copy, use, change, redistribute, ...).

Common licenses for code:

- [MIT License](#): no limitations
- [Apache License](#): minor limitations
- [GNU General Public License](#) (GPL): requires derivative works to use the same license

# Code quality

# High quality code ...

- ... works (and you can prove it!)
- ... will keep working after the next update
- ... is secure
- ... is readable
- ... is maintainable
- ... is consistent
- ... is well documented

# Linters

A **linter** is a program that checks your code for syntax errors, undefined variables, formatting errors, etc. Some linters also do **type checking**.

- [Pylint](#)
- [Flake8](#) (enforces PEP 8 formatting conventions)
- [Mypy](#) (very strict type checking)

```
$ mypy .  
translators.py:3: error: Skipping analyzing "googletrans": module is installed, but missing library stubs c  
translators.py:3: note: See https://mypy.readthedocs.io/en/stable/running\_mypy.html#missing-imports  
translators.py:27: error: "WordByWordTranslator" has no attribute "dictionary"; maybe "_dictionary"? [attr  
Found 2 errors in 1 file (checked 4 source files)
```

# Code formatters

A code formatter is a program that automatically adapts your code to a standardized formatting style.

- [Black](#)
- [autopep8](#)
- [yapf](#)

*Black*



```
class Translator(ABC) :
    requires_network = False
    @ abstractmethod

    def translate(self, text: str) -> str:
        pass
    def evaluate(self, dataset: "ParallelDataset") -> float:
        translations = [
            self.translate(source) for source, _ in dataset]
        targets = [
            target for _, target in dataset]
        return sacrebleu.\
            corpus_bleu(translations, [targets]).score
```

```
class Translator(ABC):
    requires_network = False

    @abstractmethod
    def translate(self, text: str) -> str:
        pass

    def evaluate(self, dataset: "ParallelDataset") -> float:
        translations = [self.translate(source) for source, _ in dataset]
        targets = [target for _, target in dataset]
        return sacrebleu.corpus_bleu(translations, [targets]).score
```



# AI-Tools

More advanced tools can check your code for potential **security vulnerabilities** or **bad programming patterns**.

- [Codiga](#)
- [Snyk](#)
- ...

# Using tools for code quality

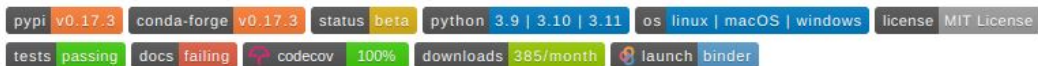
You can use tools ...

- ... **in your IDE** while you're coding
- ... in pre-commit hooks each time you commit
- ... **in CI/CD pipelines** each time you push

# Real-world example

# Example open-source project

[pymovements](#) is a Python package for analyzing eye-tracking data developed by the Universities of Potsdam and Zurich.



pymovements is an open-source python package for processing eye movement data. It provides a simple interface to download publicly available datasets, preprocess gaze data, detect oculomotoric events and render plots to visually analyze your results.

# Example open-source project

Look at the **pymovements** repository on GitHub:

[github.com/aeeye-lab/pymovements](https://github.com/aeeye-lab/pymovements)

- What is the latest version?
- Do they publish the package on PyPI?
- Do they have tests? Are the tests passing?
- What tools and CI/CD pipelines do they use?
- Is there documentation?
- Is there a license?

# Programming Project



- `pymovements` is an open-source Python package developed by a team of the UZH and the University of Potsdam for analyzing and processing eye-tracking data
- Contributions to the package count towards a **programming project (3-9 credits)**
- The package is available in a public GitHub repository (<https://github.com/aeeye-lab/pymovements>) and maintains an “Issue” list → check out the list to find tasks you are interested in
- The package follows standard software development practices (git workflow with automated tests, contribution via PR followed by a review process, ...)
  - If this does not mean anything to you, it might be an excellent opportunity to get to know such practices which are also relevant for industry software engineering jobs

Are you interested?

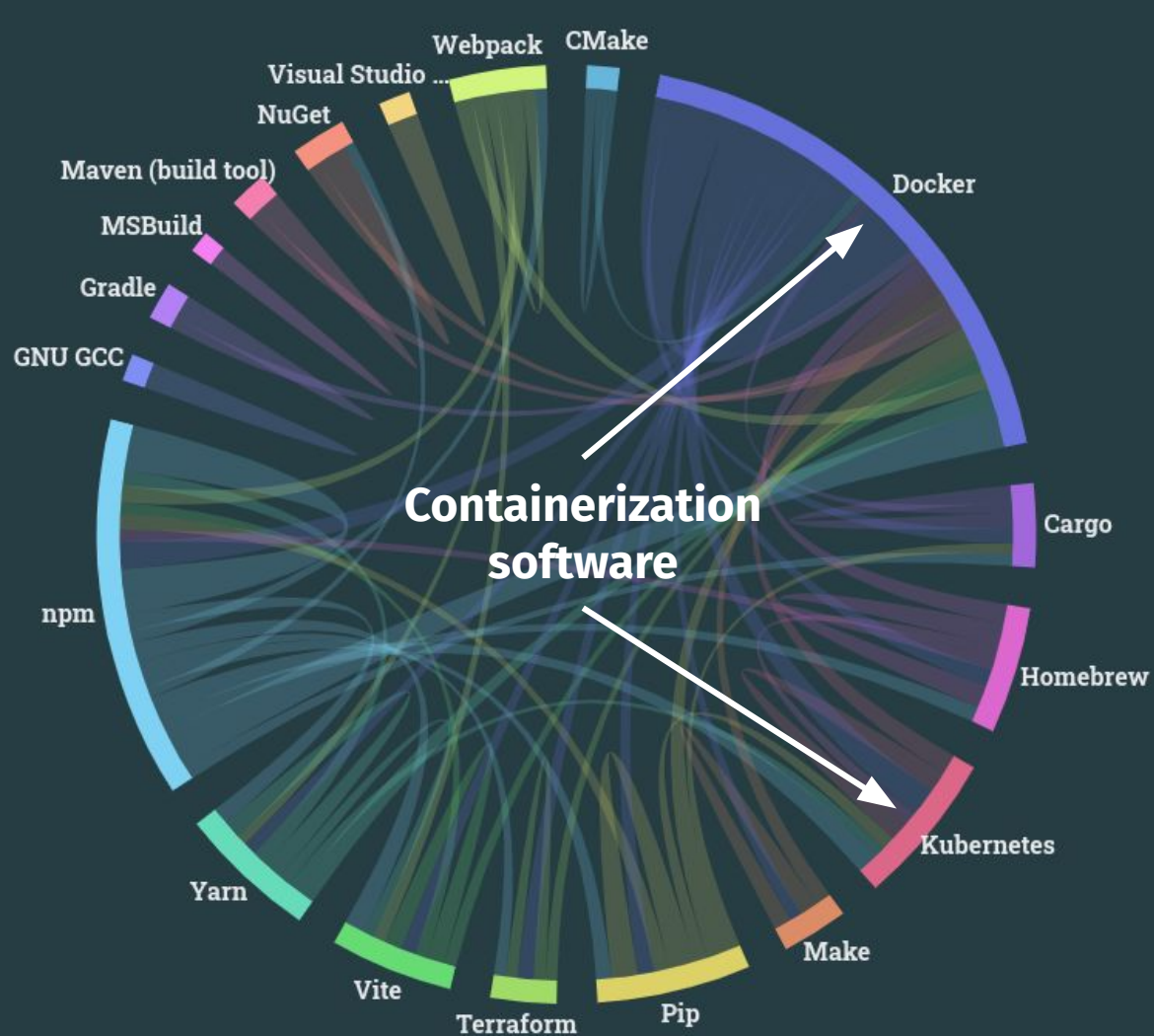
Contact **Deborah Jakobi** ([jakobi@cl.uzh.ch](mailto:jakobi@cl.uzh.ch), CC to [pymovements-list@uni-potsdam.de](mailto:pymovements-list@uni-potsdam.de))

# Containerization

# What tools do developers use to build software?

(apart from programming languages, editors, databases, and libraries)

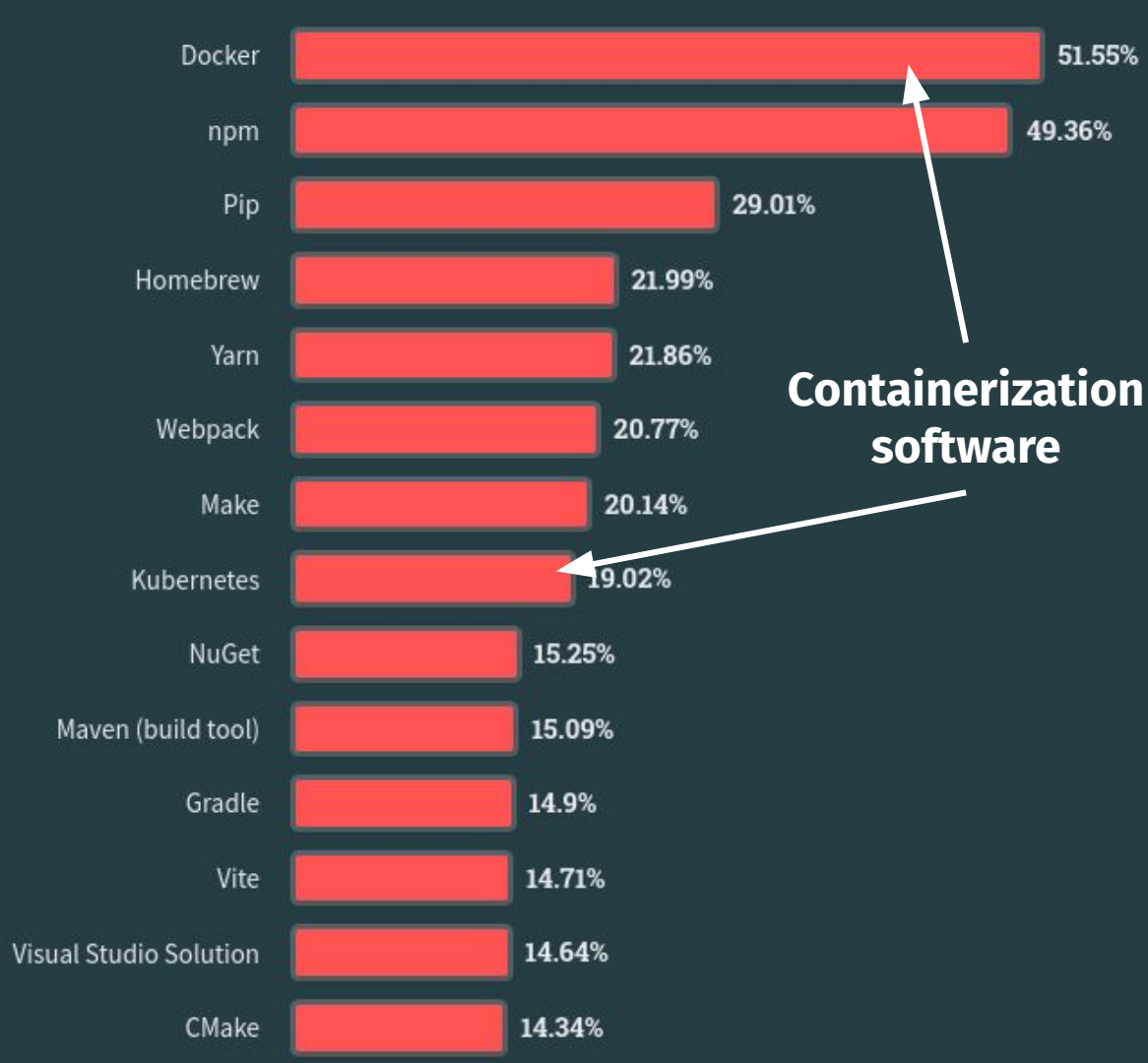
Source: [StackOverflow developer survey 2023](#)





# What tools do developers **like** using?

Source: [StackOverflow developer survey 2023](#)



# Why containerization?

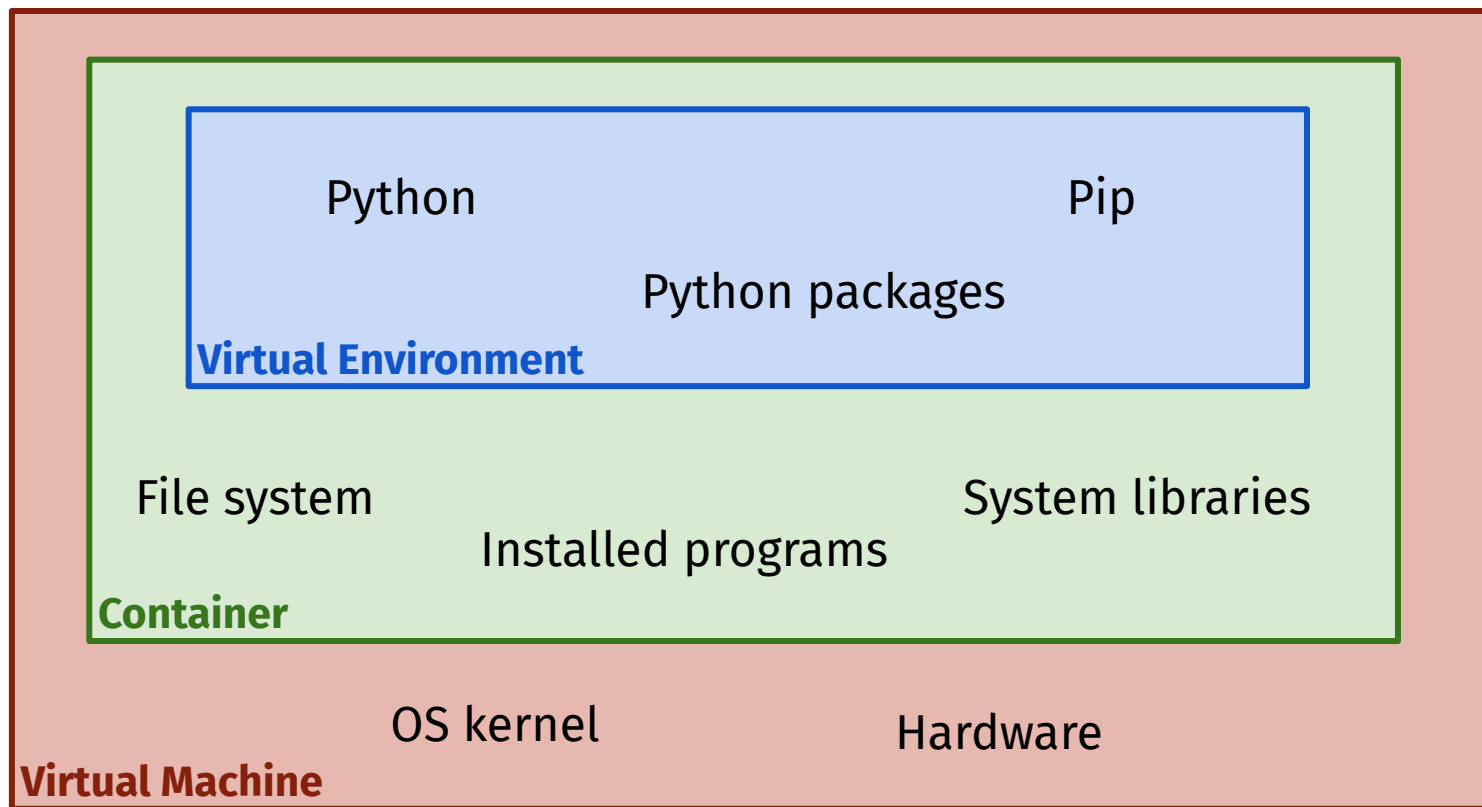
Hey, your code is broken! When I try running it, I get this error: [...]  
Please fix your code, thanks! 🙄

Sorry, I have no idea what you're talking about. It works perfectly fine on my machine! 🤔

# Why your code might not work on someone else's computer

- Different Python version
- Different package versions
- Different system library versions
- Different file paths
- Different operating system
- Different default text encoding (we'll talk about this in April)
- Different system language or time zone
- ...

Using **virtual environments** and proper **dependency specification** only address some of these problems!



# Docker

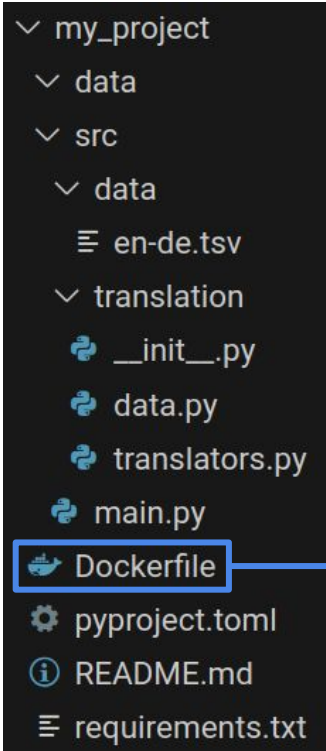


**Docker** is a **containerization software**. You can use it to ...

- ... build containers
- ... start and stop containers
- ... run commands inside containers

**Docker Hub** is a **platform** for hosting container images online.

# Dockerfiles



A Dockerfile contains instructions for building a **container image**.

```
FROM python:3.11

# Install dependencies
COPY requirements.txt /tmp/
RUN pip install -r /tmp/requirements.txt

# Set the working directory in the container
WORKDIR /src

# Copy the code into the container
COPY src /src

# Run main.py when the container launches
CMD ["python", "main.py"]
```

# Building and running containers

1. Build your image:

```
docker build -t 'my-image-name'
```

2. Run a container from your image:

```
docker run --rm 'my-image-name'
```

(--rm means that the container will be removed after it has terminated)

3. Share your image with others via Docker Hub:

```
docker push 'my-image-name'
```

# Try out my crappy translator!

1. Install Docker: [docker.com/products/docker-desktop](https://docker.com/products/docker-desktop)
2. `docker run --rm -ti saeub/pcl2-translation`  
(-ti means that the container will accept input via the terminal)
3. Type some English text



# When to use containers?

- To make your scientific experiments reproducible
- To deploy applications in the cloud
- To make sure all developers use the same environment
- When using outdated software that no longer works on your operating system

# Quiz

[pwa.klicker.uzh.ch/join/asaeub](https://pwa.klicker.uzh.ch/join/asaeub)



# Conclusion

# Take-home messages

- A **package** is a folder containing a `__init__.py` module.
- To make packages installable with `pip`, add a **`pyproject.toml`** file.
- Packages can be published on the Python package index (**PyPI**).
- **Semantic versioning** tells us which versions are **backward compatible**.
- **CI/CD** makes quick collaborative development and deployment possible.
- **Linters, code formatters**, and other tools can keep your code consistent.
- **Containerization** bundles your code together with the required environment to make sure your code will also work on other machines.