Python (Fahrzeugtechnik)

Why should we use Python?

- one of the most popular programming language
- designed as teaching & scripting language in the 1990s by Guido van Rossum, hence it is easy to read and easy to learn
- becomes more and more popular due to "lack of 'real' programmes in industry
- easy to learn, read and maintain
- Python is interpreted → No need to compile the program and you can interact with the Python interpreter which is great for pre-development
- supports scripting, functional programming and object-oriented programming
- portable to almost any computer ~~program~~ platform
- lives from contributions of a large community and provides many great libraries
- Recent popular methods like deep learning, speech ~~reg~~ recognition and the like are ~~usally~~ usually programmed in Python
- If you want to process data on a computer, chances are very high that someone created a Python library for this

Zen of ~~Pe~~ Python

→ import this

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

~~03~~ - Software - development

Git

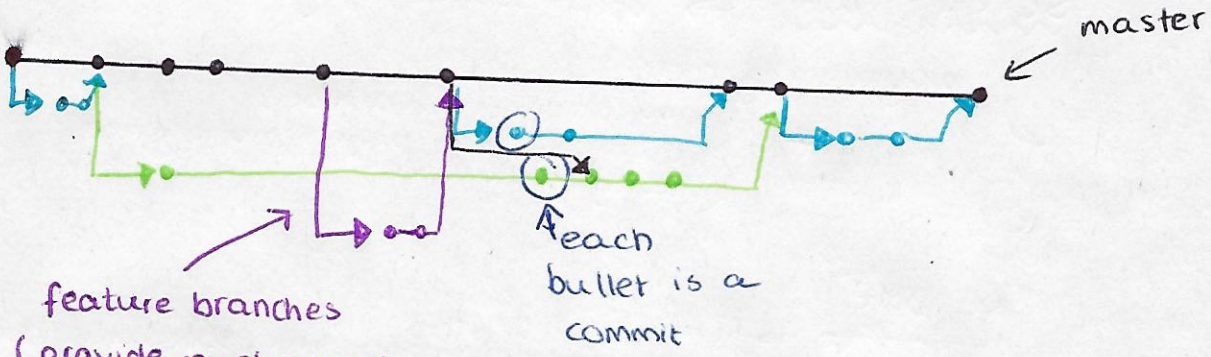
Version control (fundamental pillar in each software development project)

→ git

↳ powerful branch and merge concept

→ allows to develop software simultaneously and distributed

master - feature _branch _concept:



Pull _requests

merges of branches into master branch

= A pull request is a discussion between contributor (developer) and maintainer to include the changes of the developer in master version

- Practical implementation of code review and ensures code quality
- discussion is done on architecture level down to line of code level

GitHub → "provides a rich source of information and you can learn from giants in the software development!"

- development platform built specifically for git
- provides beside basic code hosting many features to development
- promote software products open source
- Features:
  - source code management (→ other options: bitbucket, gitlab)
  - continuous integration and delivery
  - code review
  - project management
  - team management
  - many more services

scikit-learn → open-source project

Open source project { programming code + many additional artifacts, documentation, project management, quality assurance

README
- file is the first anchor
- contains information what the software does and what is the aim of the project

good Readme { • information on how to install software, how to contribute, about the software health, links to wiki / other websites

Addd. files for developers
rules and guidelines

doc
usually a folder that contains the documentation of the software divided into:
- API
- user manual
- tutorial
- more theoretical explanation

commonly / documentation is automatically build through continuous integration and deployed to a web page

Source code folder stored on GitHub

Tests
vital to keep technical debt low and to keep the software soft

Issues and pull requests
bugs & feature requests discussed & tracked
actual change ⇒ organized, discussed & performed through pull requests

Pull requests
discussion between contributor and maintainer to change the code of the master or development branch

– Software-development

upsi

Continuous ~~development~~ integration

→ concept where changes from developers are integrated & tested in the main version ~~oft~~ of the software within short cycles

- often ~~twind~~ twinned w/ continuous delivery (CD) to be able to create a deliverable software product for customers in short cycles
- CI/CD fosters automation of manual tasks & provides foundation for distributed development on short cycles

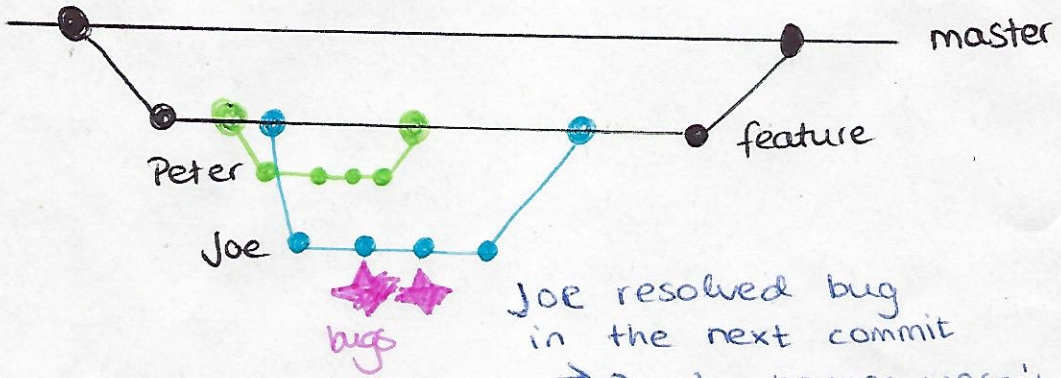
daily / many times a day

Why?

- cost of change ~~time often~~ roughly exponential function of time
  → the earlier an error is found the cheaper and easier they can be resolved
- software development in teams requires to split larger chunks* of work into smaller ones** → developer can work simultaneously w/o CI, tests can be only applied after integration of all stories
  → if these fail, the entire feature is incomplete

* "feature"

** "story"



Joe resolved bug in the next commit
→ Peter's changes weren't ~~interrupted~~ corrupted by the bugs & integration went well through

Requirements for CI

1) software is under vision control

... → Karteikarte

CI Solutions

- Jenkins:
  - very flexible and rich on features on the price of not being easy to learn and maintain ~~from exper~~

    → rather expert knowledge needed
- Travis CI:
  - more lightweight solution
  - easy to use and directly integrated as service in GitHub
- GitHub Actions:
  - very easy to use
  - most lightweight solution for GitHub repos currently (professors opinion lmao)

bad & good smelling code

Clean Code (book & coding principles by Robert C. Martin)

↳ Uncle Bob

Goal:
→ beautiful code
↳ 1) good and easy to read
2) easy to test
3) easy to reuse
4) easy to modify

→ clean code to reduce technical debt

use meaningful names!

for all layers of code!

Functions should:

- be small
- do one thing
- use descriptive names
- have no side effects
- have not more than 3 arguments (best is zero)

(methods in classes can be treated similarly to functions)
[classes should be small as well]

Do not report yourself!

- avoid redundant code
- reuse code instead

(avoid "Spaghetti code")

Comments do not heal bad code!

try to express yourself in the code

Good comments
- informative
- explain the intention why something has been made
- warn of consequences
- list To-Dos

Bad comments
- redundant
- misleading, outdated, simply wrong
- comments that comment out code

eww this is so cringe... guilty

Use good readable formatting!

vertical and horizontal space

→ Python was designed with formatting in mind

Why clean code and not just Continuous integration?

Clean code is more than linted code

note: roughly 80% of time is required for developers to read code

→ linter can tell you to use lowercase letters but cannot suggest meaningful names

→ If you automate a mess, you get automated mess

Beyond clean code

- Clean code not enough ensure high software quality
  - → clean architecture,
    clean tests,
    clean design,
    clean organization
    - → organizational mess will cause a messy product

Refactoring

= small code improvements to transform the code into clean code w/o changing functionality

- → improves the code smell and reduces technical debt

can be source of conflict:

if quantity of delivered feature is more important than quality

Software development - git

`git --version`	returns version of git
`git init <directory>`	initializes a folder as git repository
`git clone <repo>`	clones a repository for instance from GitHub on your computer
`git add <files or A for all>`	stages specific files or all for next commit
`git commit -m "<message>"`	commits all staged files w/ a commit message

Software development - git

`git status`	lists staged or modified files
`git log`	displays git history
`git merge <branch>`	merges <branch> into your current branch
`git revert <commit id>`	reverts a specific commit

COMMIT MESSAGES — add meaningful ones!

`git commit -m "Your message goes here"`

Software development - git

CONFIGURATION FILES

- **.gitignore** — which defines which fiied should be ignored by git
- **.gitattributes** — defines for instance how merges should be handled
- **setup.cfg** — which defines options of Python code checkers like pytest and flake8
- **setup.py** — which organizes development version installation

git fetch	fetches changes from remote, for instance new branches
git pull	pulls changes from current branch on your computer
git push	pushes your local changes to remote on same branch
git check-out -b <branch>	checkout <branch> ~~into your current~~ from remote on your computer

Software Development – Continuous integration

REQUIREMENTS

1) Software is under version control (git)
2) Tests are either together w/ code or before coding written (test driven development)
3) The test and building steps are automated and this automation is also under version control
4) A CI software triggers tests and builds on events like commits or pull requests and stores and returns results
5) Build slaves are available (hardware PCs or cloud services) to test and build the software
6) Tests and builds are fast.
7) Cultural environment that fosters failing fast and prioritizes bug fixes

Continuous Integration

MINDSET behind CI

- Automate the boring stuff
  → no check list / manual quality tests!
  No one wants / is going to go through checklist & run tests

- Test often, test fast

- use all weapons you have (static analysis, dynamic tests, integration tests, ...)