ETH zürich

IT Services

Towards reproducibility in data analysis and programming

About me

- Dr. Uwe Schmitt
- Work for Scientific IT Services (SIS)
- Scientific programmer
- I also work as tutor and consultant.

Our Goal: always produce same results from same data

Our Goal: always produce same results from same data

At any time

Our Goal: always produce same results from same data

At any time

At any place

Our Goal: always produce same results from same data

At any time

At any place

By any person

What can go wrong?

1. Software / tools are not available (anymore).

What can go wrong?

1. Software / tools are not available (anymore).
2. Used software is fragile.

What can go wrong?

1. Software / tools are not available (anymore).
2. Used software is fragile.
3. Processing steps are not documented.

What can go wrong?

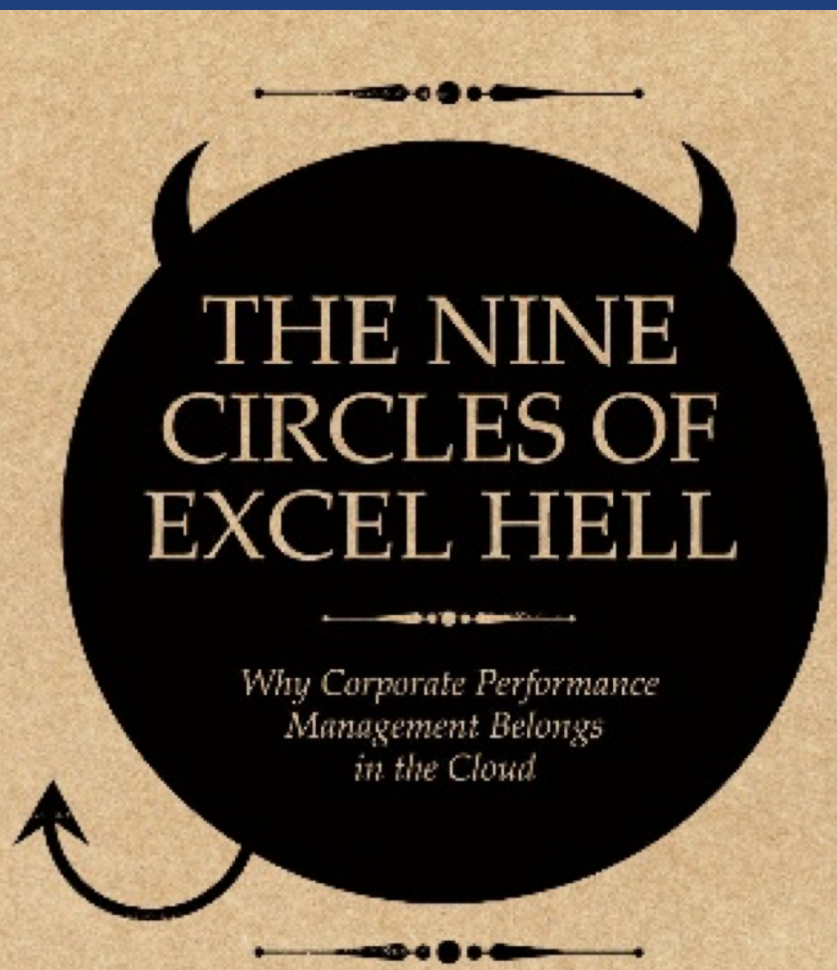
1. Software / tools are not available (anymore).
2. Used software is fragile.
3. Processing steps are not documented.
4. Human mistakes during processing.

1. Not available software / tools

- Use open source software / programming languages.
- Publish your code using an open source license.

2. Software is fragile

- Google for "excel hell"!


Adaptive Insights
THE NINE CIRCLES OF EXCEL HELL
Why Corporate Performance Management Belongs in the Cloud

2. Software is fragile

- Excel: incorrect leap year calculations 1900-02-29
- `7 Worst Excel Mistakes of All Time`

5. Harvard Professors Publish Wrong Result in Paper

Two Harvard professors incorrectly concluded that economic growth slows significantly if the debt of the country exceeds 90% of the country's Gross Domestic Product (GDP). However, it was later discovered by none other than a student that the average formula used didn't cover the complete range of values in the column.

3. Processing steps are not documented.

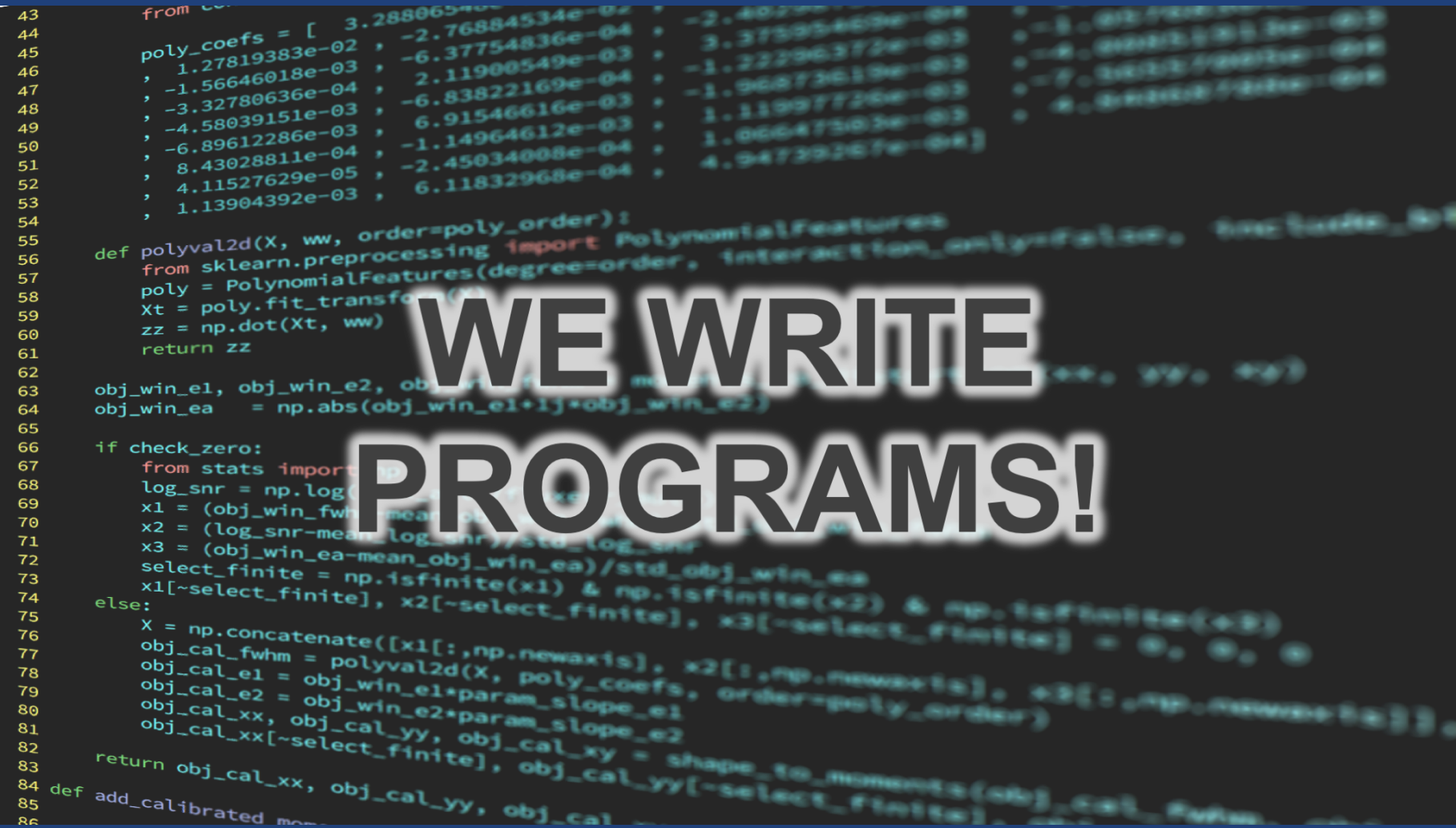
4. How to avoid human mistakes?


WE USE
RECIPES!


WE USE
PROTOCOLS!

Recipes / lab protocols:

- List of simple steps
- More or less exact instructions
- Executed by humans


WE WRITE
PROGRAMS!

Programs

```
numbers = read_txt("numbers.txt")
average = sum(numbers) / len(numbers)
print("average is", average)
```

```
average is 12.34
```

Programs

```
numbers = read_txt("numbers.txt")
average = sum(numbers) / len(numbers)
print("average is", average)
```

```
average is 12.34
```

- List of simple steps
- Exact instructions
- Executed by unforgiving computers

Why to program?

- Reduce / no manual steps in your analysis
- Automate as much as possible
- **Good code** is implicit documentation how you produced results
- Others can build upon your work


Programming can be fun


… well, not always

EFFECTS OF COMPUTER PROGRAMMING ON COGNITIVE OUTCOMES: A META-ANALYSIS

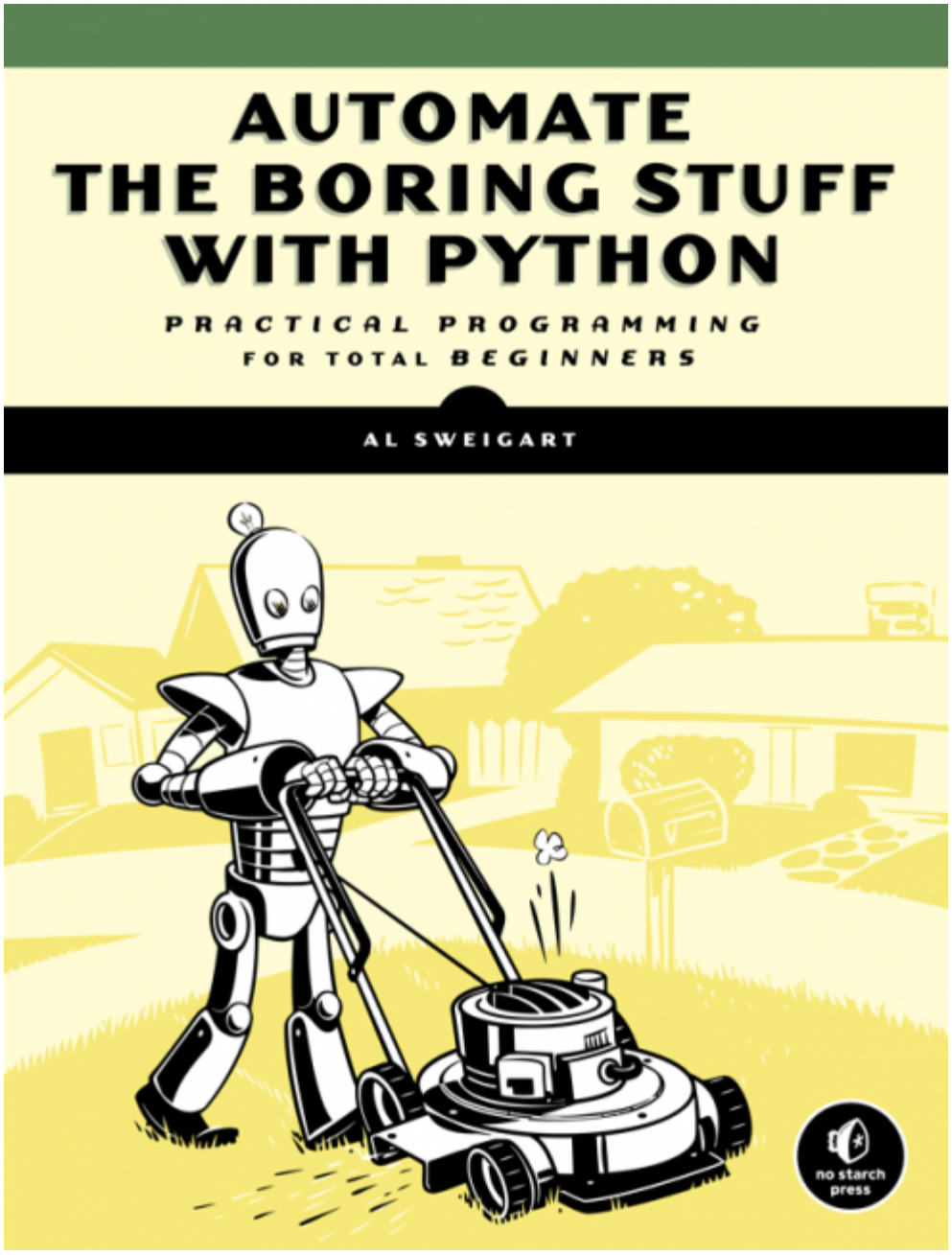
YUEN-KUANG CLIFF LIAO
University of Houston

GEORGE W. BRIGHT
University of North Carolina at Greensboro

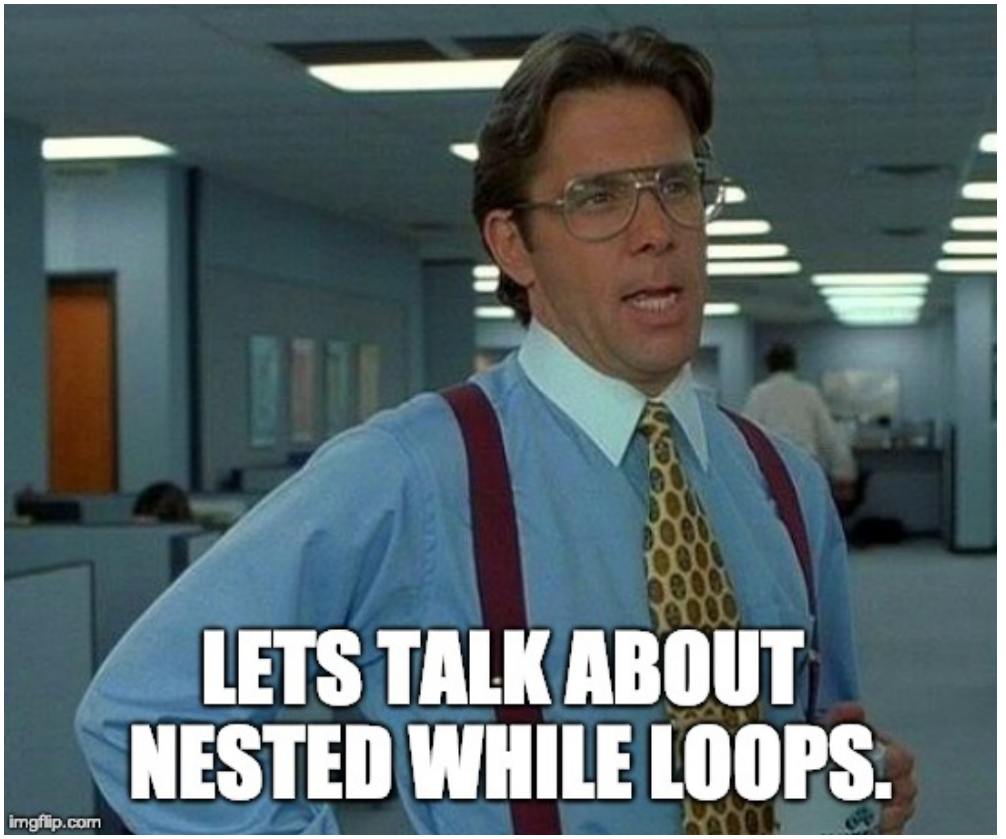
ABSTRACT

Although claims regarding the cognitive benefits of computer programming have been made, results from existing empirical studies are conflicting. To make a more reliable conclusion on this issue, a meta-analysis was performed to synthesize existing research concerning the effects of computer programming on cognitive outcomes. Sixty-five studies were located from three sources, and their quantitative data were transformed into a common scale—Effect Size. The analysis showed that 58 or 89 percent of the study-weighted effect sizes were positive and favored the computer programming group over the control groups. The overall grand mean of the study-weighted effect size for all 432 comparisons was 0.41; this suggests that students having computer programming experiences scored about sixteen percentile points higher on various cognitive-ability tests than students who did not have programming experiences. In addition, four of the seven coded variables selected for this study (i.e., type of publication, grade level, language studied, and duration of treatment) had a statistically significant impact on the mean study-weighted effect sizes. The findings suggest that the outcomes of learning a computer language go beyond the content of that specific computer language. The results also suggest to teachers a mildly effective approach for teaching cognitive skills in a classroom setting.

... the findings suggest that the outcomes of learning a computer language go beyond the content of that specific computer language.


AUTOMATE
THE BORING STUFF
WITH PYTHON
PRACTICAL PROGRAMMING
FOR TOTAL BEGINNERS
AL SWEIGART
no starch
press

Eases talking to the IT people.



How do I learn to program?

- Choose easy-to-learn and open source language like Python or R.

How do I learn to program?

- Choose easy-to-learn and open source language like Python or R.
- R preferable for advanced statistics and elaborate plotting.

How do I learn to program?

- Choose easy-to-learn and open source language like Python or R.
- R preferable for advanced statistics and elaborate plotting.
- Python preferable for data science and machine learning.

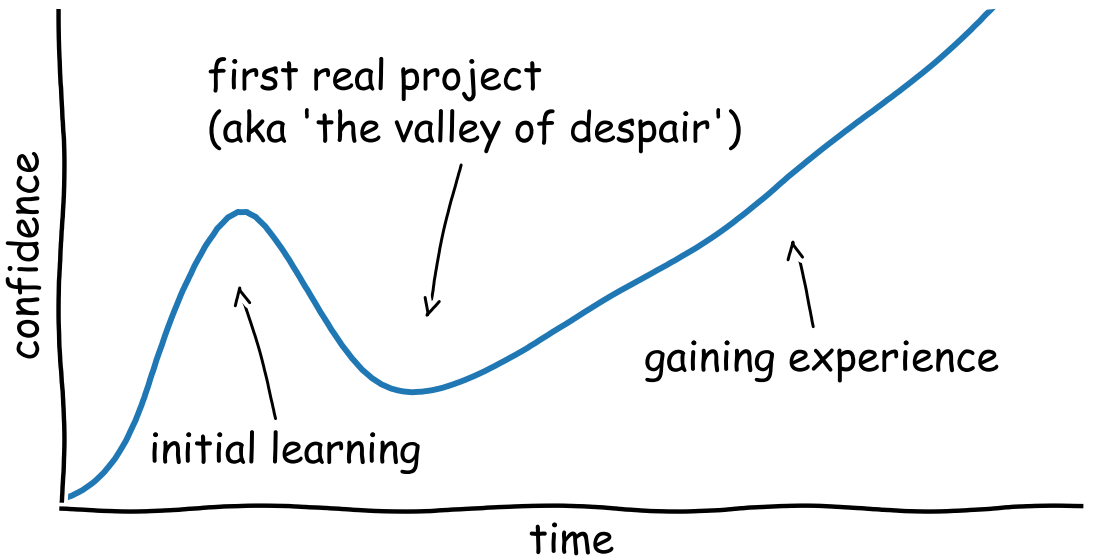
How do I learn to program?

- Choose easy-to-learn and open source language like Python or R.
- R preferable for advanced statistics and elaborate plotting.
- Python preferable for data science and machine learning.
- I consider Python as the clearer and more versatile programming language.

How do I learn to program?

- Choose easy-to-learn and open source language like Python or R.
- R preferable for advanced statistics and elaborate plotting.
- Python preferable for data science and machine learning.
- I consider Python as the clearer and more versatile programming language.
- There are many books and online courses!

Typical learning curve



Now I know programming, what can go wrong?

Now I know programming, what can go wrong?

Actually a lot!

What can go wrong?

1. Programs change over time.

What can go wrong?

1. Programs change over time.
2. Programs can break.

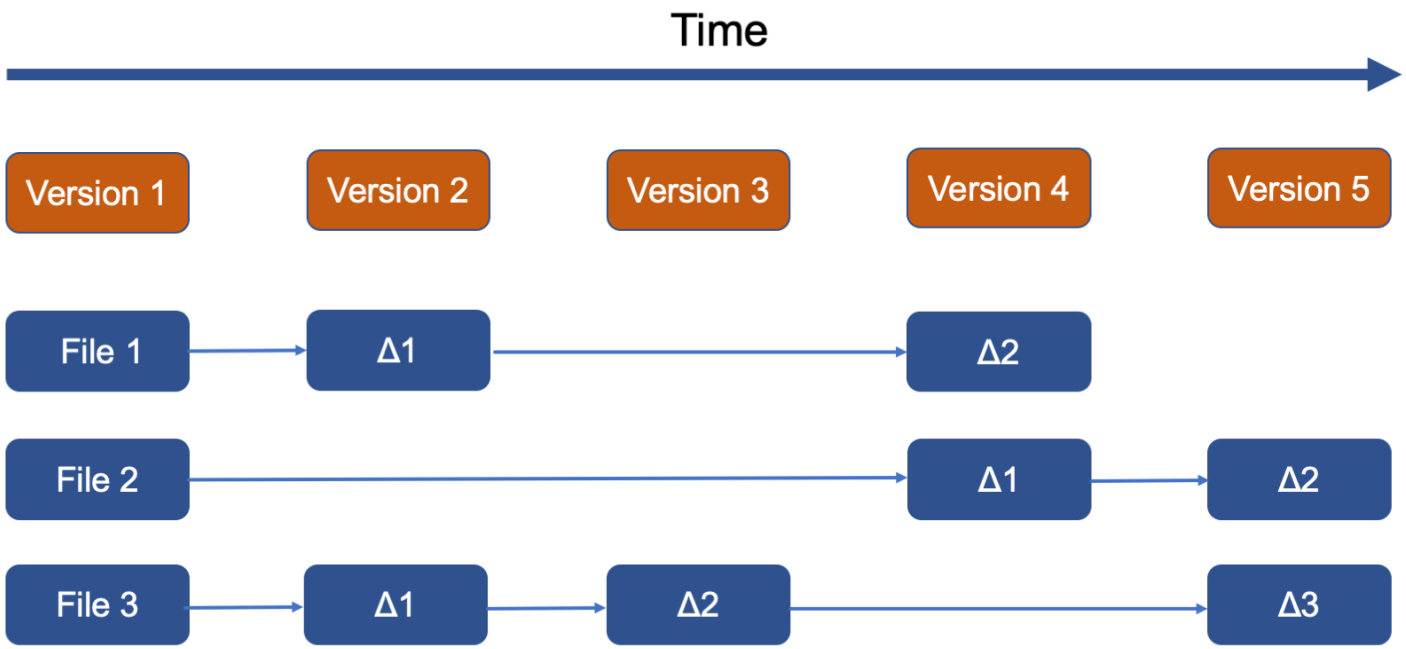
What can go wrong?

1. Programs change over time.
2. Programs can break.
3. Code can be complex.

What can go wrong?

1. Programs change over time.
2. Programs can break.
3. Code can be complex.
4. Programs will run on other computers.

1. Managing changes


Time
Version 1
Version 2
Version 3
Version 4
Version 5
File 1
Δ1
Δ2
File 2
Δ1
Δ2
File 3
Δ1
Δ2
Δ3

Version control systems (VCS)

- **time machines** for your source code and textual data.
- `git` is the most common tool for tracking changes over time.
- `git` ≠ `github`!
- `github`, `gitlab`: web frontends for managing git repositories.
- ETH has its own instance `gitlab.ethz.ch` for hosting code.

git benefits

- No version numbers in file names any more!
- No comments to keep old and outdated code.
- Undo changes.
- Supports collaborative development.

Version your software

- Learn to write "packages" instead of emailing code.
- Use semantic versioning `x.y.z`.
  - `x` for major updates (python2 and python3)
  - `y` for new features which don't crash existing results.
  - `z` is incremented for bug fixes.
- "freeze" dependencies: document versions of external code.

2. Programs can be incorrect

Why?

- You make mistakes during development.
- Software complexity grows during development.
- Others use your software not as intended.

Techniques

- Defensive programming.

```
def average(data):
    assert len(data) > 0
    ...
```

Techniques

- Defensive programming.

```
def average(data):
    assert len(data) > 0
    ...
```

- Automated code tests: unit tests vs. regression tests.

```
def test_average():
    assert average([1]) == 1
    assert average([1, 2]) == 1.5
    assert average([1, 2, 3]) == 2
```

- A collection of unit tests is a *test suite*.

3. Code can complex.

Clean code ("you read code more often than you write it")

- Choose good names for variables and functions.
- Write many functions.
- **DRY** (don't repeat yourself): Avoid duplications.
- Write generic code: e.g. don't hard code file names.
- Document your program incl. the underlying concepts.
- unit tests enforce better code structure.
- Read about "clean code".

Other best practices

- **KISS**: Keep it simple and stupid: Keep your solutions as simple as possible.
- **YAGN**: You ain't gonna need it: Don't overdesign your programs.
- *In the face of ambiguity, refuse the temptation to guess*:
  - Don't try to fix invalid input.
  - Complain instead!
- Understand your programs vs *programming by coincidence*.
- Be brave to trash your code and start again.

4. Programs will run in different environments

Problem:

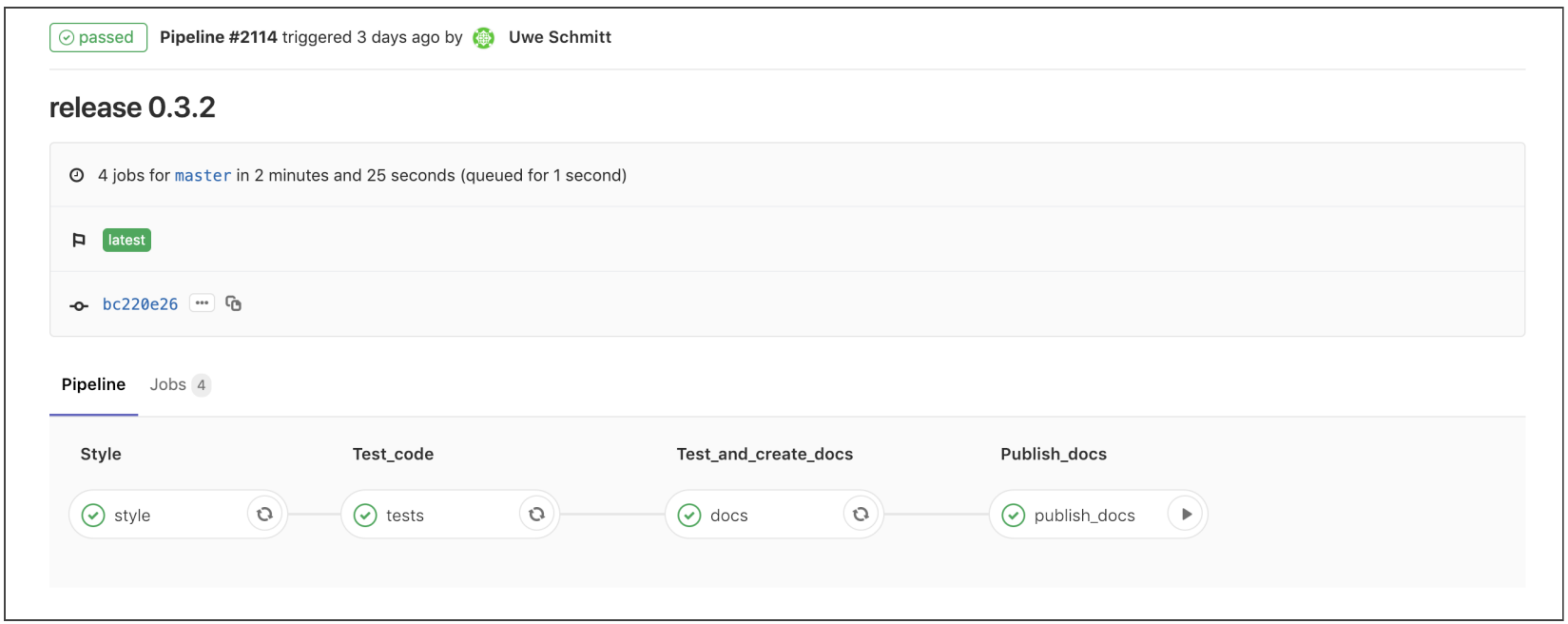
Your program depends on other software

Like: Python 3.6 or libraries

How to check if my code works on different computers?

- CI tests = *continuous integration tests*
- Automates installation on pristine computer and running tests.
- Can be integrated in `github.com`, `gitlab.com` or `gitlab.ethz.ch`.

CI Pipeline in `gitlab`.



Virtual environments

Virtual environments try to isolate programs and their dependencies from the rest of the computer.

- Python has the concept of so called "virtual environments".

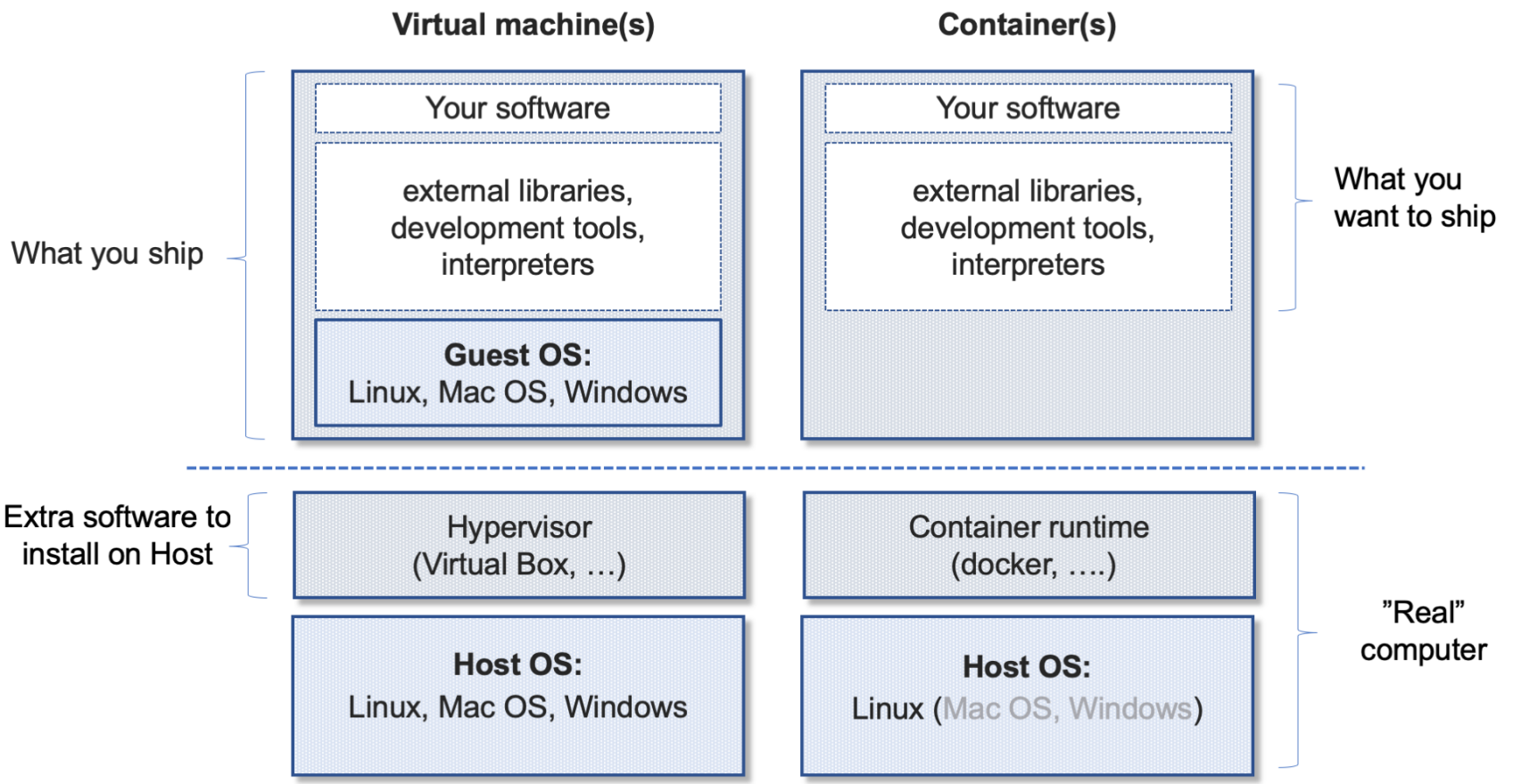
```
$ python3 -m venv ...
```

- *Anaconda* supports so called "conda environments" for Python and R.

Sledge hammers for complex scenarios

Concepts

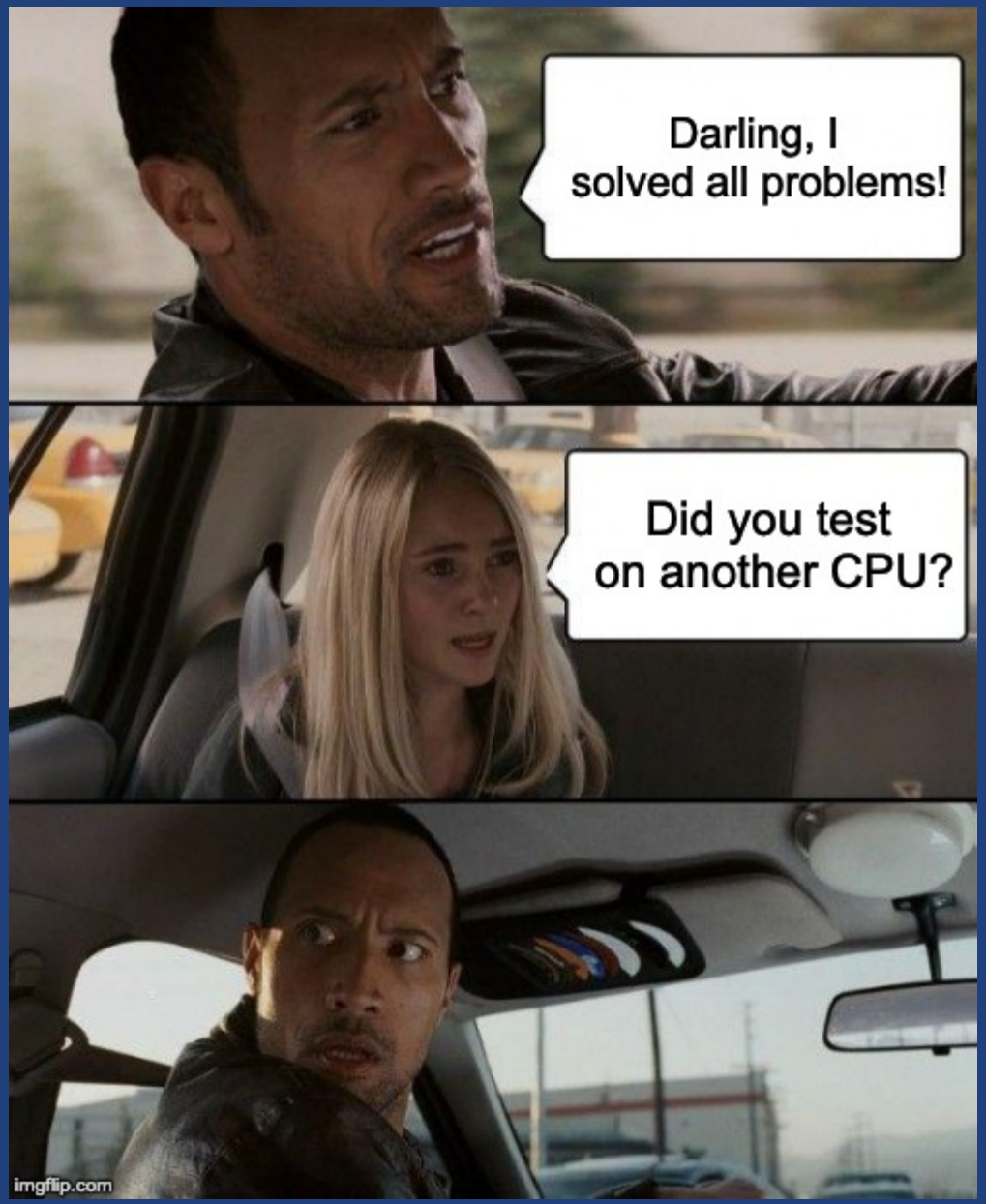
- Idea: bundle your software and all dependencies
- Virtual Machine (VM): bundle contains full operating system
- Container: does not bundle operating system
- `docker`: one way to manage and run containers.


Virtual machine(s)
Container(s)
Your software
external libraries, development tools, interpreters
Guest OS: Linux, Mac OS, Windows
Your software
external libraries, development tools, interpreters
What you ship
What you want to ship
Extra software to install on Host
Hypervisor (Virtual Box, …)
Container runtime (docker, ….)
Host OS: Linux, Mac OS, Windows
Host OS: Linux (Mac OS, Windows)
”Real” computer

Comparison VM vs Container

	Advantages	Disadvantages
Virtual Machine	Easy to setup	10s of GB at least to ship startup time: minutes reduced performance
Container	lightweight startup time: milliseconds native performance	Some learning involved, Linux guest only

All problems solved?


Darling, I
solved all problems!
Did you test
on another CPU?
imgflip.com

Computer arithmetic is not exact!

```
>>> from math import sin, pi

>>> sin(pi)
1.2246467991473532e-16

>>> 0.1 + 0.2 + 0.3
0.6000000000000001

>>> (0.1 + 0.2) + 0.3 == 0.1 + (0.2 + 0.3)
False
```

- $0.2_{10} = 0.0011\ 0011\ 0011\ 0011..._2$
- Numbers have to be truncated (usually 52 digits for 64 bit floats) as memory is limited.
- **This is not a problem for reproducibility!**

- Such behaviour for `+` ,`*`, `-` and `/` is standardized by *IEEE Standard for Floating-Point Arithmetic (IEEE 754)*.
- But `exp` and other analytical functions not!

I ran this on two computers with different CPUs

```
>>> "%.14e" % math.exp(-math.sin(431))
```

```
1.76144146064997e+00
```

```
>>> "%.14e" % math.exp(-math.sin(431))
```

```
1.76144146064998e+00
```

- This is very rare and its actual effect (error propagation) requires mathematical analysis.
- CI testing can help to detect such issues!

Randomized algorithms

E.g used in machine learning (cross valiadation, batch learning).

- Most random numbers are *pseudo random numbers*.
- Starting with a given "seed" the computer will always create the same random number sequence.
- Freeze the seed when archiving / publishing your code. Also when unit testing.

```
>>> import random
>>> random.seed(42)
>>> random.random()
0.6394267984578837
```

But this is so much to learn

But this is so much to learn

Learn incrementally

But this costs so much time

But this costs so much time

Think about actual costs and risks.

How can I continue after this presentation?

How can I continue after this presentation?

Don't hesitate to contact us
`https://sis.id.ethz.ch`

SIB Course best practices in programming.

Summary

Summary

Learn programming!

Summary

Learn programming!

Use `git`!

Summary

Learn programming!

Use `git`!

Write robust and clean code!

Summary

Learn programming!

Use `git`!

Write robust and clean code!

Implement automated code tests!

Summary

Learn programming!

Use `git`!

Write robust and clean code!

Implement automated code tests!

Use VM or containers!

Thanks for your attention!