Automating teaching about automation in Python
I heard you like automation, so I put some automation in your automation

Florian Bruhin

Swiss Python Summit 2022
September 22nd
About me

Florian Bruhin, @The-Compiler

2006  Started programming (QBASIC, bash)
2011  Started using Python
2013  Started developing qutebrowser
2015  Switched to pytest, ended up as a maintainer
2016 – 2019  BSc. in Computer Science at HSR OST

40% employed (OST): Teaching Automation with Python to first-semester students
60% open-source and freelancing (Bruhin Software): Python, pytest, Qt
About me

Florian Bruhin, @The-Compiler

2006  Started programming (QBASIC, bash)
2011  Started using Python
2013  Started developing qutebrowser
2015  Switched to pytest, ended up as a maintainer
2016 – 2019  BSc. in Computer Science at HSR OST

40% employed (OST): Teaching Automation with Python to first-semester students
60% open-source and freelancing (Bruhin Software): Python, pytest, Qt
About me

Florian Bruhin, @The-Compiler

2006  Started programming (QBASIC, bash)
2011  Started using Python
2013  Started developing qutebrowser
2015  Switched to pytest, ended up as a maintainer
2016 – 2019 BSc. in Computer Science at HSR OST

40% employed (OST): Teaching Automation with Python to first-semester students
60% open-source and freelancing (Bruhin Software): Python, pytest, Qt
About me

Florian Bruhin, @The-Compiler

- **2006** Started programming (QBasic, bash)
- **2011** Started using Python
- **2013** Started developing qutebrowser
- **2015** Switched to pytest, ended up as a maintainer
- **2016 – 2019** BSc. in Computer Science at HSR OST

**40% employed (OST):** Teaching Automation with Python to first-semester students

**60% open-source and freelancing (Bruhin Software):** Python, pytest, Qt
Florian Bruhin, @The-Compiler

2006 Started programming (QBASIC, bash)
2011 Started using Python
2013 Started developing qutebrowser
2015 Switched to pytest, ended up as a maintainer
2016 – 2019 BSc. in Computer Science at HSR OST

40% employed (OST): Teaching Automation with Python to first-semester students
60% open-source and freelancing (Bruhin Software): Python, pytest, Qt
About me

Florian Bruhin, @The-Compiler

2006  Started programming (QBAsic, bash)
2011  Started using Python
2013  Started developing qutebrowser
2015  Switched to pytest, ended up as a maintainer
2016 – 2019 BSc. in Computer Science at HSR OST

40% employed (OST): Teaching Automation with Python to first-semester students
60% open-source and freelancing (Bruhin Software): Python, pytest, Qt
The problem

- Pre-2021: Students learn Java as their primary programming language at OST
- Java can be a pain to deal with \(^\text{[citation needed]}\)
  ...especially if you want a tool to make your life easier rather than learn the fundamentals of programming
The problem

- Pre-2021: Students learn Java as their primary programming language at OST
- Java can be a pain to deal with [citation needed]
  ...especially if you want a tool to make your life easier rather than learn the fundamentals of programming
- More and more places where Python is used as a tool
  (to teach math, physics, AI, but also projects, final thesis, etc.)
- Students demand learning Python in their studies
  ...and lots of schools/universities have introduced/switched to it
The solution

- Fall semester 2021: New course **Automatisierung mit Python** *(Automation with Python)* for **all** first-semester IT students
- In **addition** to Java, but with a **different goal**: Solving real-life problems!
- “**Students will be able to use** the Python programming language [. . .] **for simple and complex automation tasks.**”
The solution

• Fall semester 2021: New course **Automatisierung mit Python** (Automation with Python) for **all** first-semester IT students
• In **addition** to Java, but with a **different goal**: Solving real-life problems!
• “Students will be able to use the Python programming language [...] for simple and complex automation tasks.”

• **Flipped classroom:** No lectures, no paper exam. Interactive graded labs and a small graded project!
• We have many newcomers studying IT, or people mostly doing support/network/..., without much programming experience.

You need to **get your hands dirty** to learn programming. It’s not just theory, but also a “craft”!
We want students to learn **both**: University of **Applied** sciences!
Interactive learning

Labs

Python Basics

Introduction

This notebook covers some basics of the Python programming language: Invoking the interpreter, operators, variables, types and calling functions.

This notebook covers the first (and parts of the second) chapter of the book.

Please note: There are several ways how to print strings and variable values together. The book introduces so-called “f-strings” rather late, but we prefer to use them much sooner (as in: from the beginning and in this lesson). They are described in the summary below.

Optional resources

You can find more information in the Python documentation:

- Invoking the shell
- Types and operators
- Expressions

Relevant Real Python tutorials:

- Basic Data Types in Python
- Variables in Python
- Operators and Expressions in Python

Summary

Invoking the Shell

Entering the interactive interpreter is done by running python (or python3) in the command line:

```
$ python
Python 3.7.3 (default, Jun 17 2019, 15:02:19)
```
e) Input and output

- Ask the user for their favourite color and save the result in `color`. Note that it's possible to pass an additional text to be shown to the user (a "prompt") to the function you'll need to use, so you don't need a separate `print` for this.
- Show the text `So you like the color red? Great choice!`, but with `red` replaced by the user's input.

```
[2]: color = input("What's your favourite color?")
print(f"So you like the color {color}? Great choice!")
```

What's your favourite color? purple
So you like the color purple? Great choice!
Interactive learning

Tests with testbook

Last change: 6 seconds ago

: Testing...0m

Failed

- test_18 failed
- test_26 failed

Passed

- test_01 passed
- test_02 passed
- ...
- test_106 passed
- test_107 passed

Submission successful! (2022-09-20 14:35:04, 7fc0904)
Interactive learning

Grading

• 1 ungraded lab (setup and getting started)
• 5 graded labs, 1/3 of final grade, **automated tests**
• Final project, **2/3 of final grade, graded manually**
  • Python basics, flow control, data structures, . . .
  • Writing a CLI
  • Using web APIs
The... problem?

Over **120 students**, a total of 9 slots (4 hours each) every 2 weeks.
Slightly less this year: 110 or so, and “only” 7 slots.

That’s **a lot** given that I’m doing this the first time!
Thanks, Stefan Richter, for trusting that we could pull it off.

In addition:

- **I love** writing opensource (qutebrowser/pytest), and giving company trainings
- Thus, this needs to stay a **40%** occupation (averaged over a year, I don’t teach in spring)
- Other people are busy too! But I got some help.
  Thanks, Marco, Méline and Urs!
Focusing my attention

- With $> 100$ students, any kind of manual work with $\mathcal{O}(n)$ is almost certainly **worth automating**!
- I teach students how to make their studies easier. Might as well make my job easier!
- I want to focus on the **interesting** part: Creating an environment to help people learn, helping people who are stuck, the beauty of **teaching**.
Focusing my attention

- With > 100 students, any kind of manual work with $O(n)$ is almost certainly **worth automating**!
- I teach students how to make their studies easier. Might as well make my job easier!
- I want to focus on the **interesting** part: Creating an environment to help people learn, helping people who are stuck, the beauty of **teaching**.
- Let Python do the **boring** part. Bonus points: It gets easier every year, because more is automated!
- A word of caution: Automation is **not** a substitute for teaching. **Know where to stop!**
- I’m not lazy (…sometimes) – but I want to **focus my attention** on things which benefit students most.
The danger of automation

"I SPEND A LOT OF TIME ON THIS TASK. I SHOULD WRITE A PROGRAM AUTOMATING IT!"

**Theory:**
- **Work on original task** decreases over time.
- **Automation takes over**.
- **Free time**.

**Reality:**
- **Writing code**.
- **Debugging**.
- **Rethinking**.
- **Ongoing development**.
- **No time for original task anymore**.
Eh, I just need a handful of very simple scripts.

I won’t bother setting up...

- ...a proper Python package
- ...type annotations
- ...linters / formatters
- ...tests (Yes, I’m a pytest maintainer. Yes, I’m ashamed.)

The *real* danger
The *real* danger: Whooops

- Development / Deployment scripts: 730 lines
- Scraping participants: 600 lines
- Sending welcome mails / other mail code: 370 lines
The real danger: Whooops

- Development / Deployment scripts: 730 lines
- Scraping participants: 600 lines
- Sending welcome mails / other mail code: 370 lines
- “taas”: Testing As A Service: 430 lines
- “AutPy Commander” GUI: 1300 lines
- Checking lab submission progress: 480 lines
- Misc. utilities for handling data: 390 lines

45 Python files
6500 lines
5000 LOC (tokei)

Not including:
- 950 lines of solutions
- 3000 lines tests for labs
The *real* danger: Whooops

Development / Deployment scripts 730 lines
Scraping participants 600 lines
Sending welcome mails / other mail code 370 lines

“taas”: Testing As A Service 430 lines
“AutPy Commander” GUI 1300 lines
Checking lab submission progress 480 lines
Misc. utilities for handling data 390 lines

Getting progress about final project hand-ins 350 lines
Grading final projects, sending feedback 680 lines
<table>
<thead>
<tr>
<th>Task</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development / Deployment scripts</td>
<td>730</td>
</tr>
<tr>
<td>Scraping participants</td>
<td>600</td>
</tr>
<tr>
<td>Sending welcome mails / other mail code</td>
<td>370</td>
</tr>
<tr>
<td>“taas”: Testing As A Service</td>
<td>430</td>
</tr>
<tr>
<td>“AutPy Commander” GUI</td>
<td>1300</td>
</tr>
<tr>
<td>Checking lab submission progress</td>
<td>480</td>
</tr>
<tr>
<td>Misc. utilities for handling data</td>
<td>390</td>
</tr>
<tr>
<td>Getting progress about final project hand-ins</td>
<td>350</td>
</tr>
<tr>
<td>Grading final projects, sending feedback</td>
<td>680</td>
</tr>
<tr>
<td>Checking / resetting to deadlines</td>
<td>180</td>
</tr>
<tr>
<td>“pseudotaas”: Rerunning tests locally</td>
<td>300</td>
</tr>
<tr>
<td>Calculating final grades</td>
<td>550</td>
</tr>
</tbody>
</table>

45 Python files
6500 lines
5000 LOC (tokei)
not including:
950 lines of solutions
3000 lines tests for labs
The real danger: Whooops

Development / Deployment scripts 730 lines
Scraping participants 600 lines
Sending welcome mails / other mail code 370 lines

“taas”: Testing As A Service 430 lines
“AutPy Commander” GUI 1300 lines
Checking lab submission progress 480 lines
Misc. utilities for handling data 390 lines

Getting progress about final project hand-ins 350 lines
Grading final projects, sending feedback 680 lines

Checking / resetting to deadlines 180 lines
“pseudotaas”: Rerunning tests locally 300 lines
Calculating final grades 550 lines

45 Python files

5000 LOC (tokei) not including:
950 lines of solutions
3000 lines tests for labs
The *real* danger: Whooops

- Development / Deployment scripts: 730 lines
- Scraping participants: 600 lines
- Sending welcome mails / other mail code: 370 lines
- “taas”: Testing As A Service: 430 lines
- “AutPy Commander” GUI: 1300 lines
- Checking lab submission progress: 480 lines
- Misc. utilities for handling data: 390 lines
- Getting progress about final project hand-ins: 350 lines
- Grading final projects, sending feedback: 680 lines
- Checking / resetting to deadlines: 180 lines
- “pseudotaas”: Rerunning tests locally: 300 lines
- Calculating final grades: 550 lines

**45 Python files**

**6500 lines**

**5000 LOC** (tokei)

*not including:*
- 950 lines of solutions
- 3000 lines tests for labs
The real danger: Whooops

Development / Deployment scripts: 730 lines
Scraping participants: 600 lines
Sending welcome mails / other mail code: 370 lines
“taas”: Testing As A Service: 430 lines
“AutPy Commander” GUI: 1300 lines
Checking lab submission progress: 480 lines
Misc. utilities for handling data: 390 lines
Getting progress about final project hand-ins: 350 lines
Grading final projects, sending feedback: 680 lines
Checking / resetting to deadlines: 180 lines
“pseudotaas”: Rerunning tests locally: 300 lines
Calculating final grades: 550 lines

45 Python files

6500 lines
5000 LOC (tokei)

not including:
950 lines of solutions
3000 lines tests for labs
1. I was there
2. Contents important
3. Contents interesting
4. Needed time is high
5. Content matches desc.
6. Useful material
7. Well structured
8. Understandable
9. Speed good
10. Extra material / media
11. Link between lect./ex.
12. Room for questions
13. Checking progress
14. Lecturer competent
15. Link between theory/practice
16. Didactics
17. Lecturer engaged
18. Mutual respect
19. Room / environment
20. Overall
Success!

- Concept of “Interactive learning” / “flipped classroom” as a whole. Daniele Procida / EvilDMP of Diátaxis\(^1\):
  “I hardly believe in teaching anymore. The best thing you can do is creating an environment where people learn.”
- Using git as a “database” for student submissions, with one branch per student
- Including test logs (HTML + JSON) in the commits
- Having a custom GUI tool to view a student’s submission and test report
- Project grading based on parsing Markdown checklists
- …all the other automation really, not regretting any of the time spent on it!

\(^1\)diataxis.fr, structuring docs into tutorial/how-to/explanation/reference
Issues

• Students accidentally deleting Jupyter cell tags
  ⇒ Tooling to notify us, protecting cells

• OST GitLab admins migrating storage without making it read-only
  ⇒ Thankfully I had the lost commits locally... please don’t do that again!

• Various smaller issues with Jupyter cluster
  ⇒ Ample deadline extension for affected students, we’re humans!
Issues

• Students accidentally deleting Jupyter cell tags
  ⇒ Tooling to notify us, protecting cells

• OST GitLab admins migrating storage without making it read-only
  ⇒ Thankfully I had the lost commits locally... please don’t do that again!

• Various smaller issues with Jupyter cluster
  ⇒ Ample deadline extension for affected students, we’re humans!

• Me forgetting to check some checkboxes in the project grading checklists
  ⇒ Human mistakes bound to happen with so many students and days of grading.
  Caught thanks to detailed feedback mails being sent, additional sanity checks

• Grading system’s API is “Download a template .xlsx, add grades, upload”
  ...but openpyxl somehow corrupts template.
  ⇒ Needs further debugging, until then, copy-paste all grades once in Libreoffice
Things nobody can prepare you for...

Disclaimer: I don’t like calling students out for their mistakes, making mistakes is normal. But those occurrences are just too strange to not tell you about...

- Ctrl+C, Ctrl+V
- “Allgemeiner Verpeiltheitsfaktor”
- Student was in military (“WK”) for weeks, “didn’t know” they had to hand in stuff
- `r = requests.get("https://random.dog/woof.json")
data = eval(r.content)`
- `git --config user.name "student.email@example.com"
git --config user.email "password-for-said-email`
⇒ With >100 students, prepare to see every corner case you can think of, and some you’d never think of. Automation won’t help you take difficult decisions.
Issues

Things nobody can prepare you for...

Disclaimer: I don’t like calling students out for their mistakes, making mistakes is normal. But those occurrences are just too strange to not tell you about...

- `Ctrl` + `C`, `Ctrl` + `V`

```python
r = requests.get("https://random.dog/woof.json")
data = eval(r.content)
```

```bash
git --config user.name "student.email@example.com"
git --config user.email "password-for-said-email"
```

⇒ With >100 students, prepare to see every corner case you can think of, and some you’d never think of. Automation won’t help you take difficult decisions.
Issues

Things nobody can prepare you for...

Disclaimer: I don’t like calling students out for their mistakes, making mistakes is normal. But those occurrences are just too strange to not tell you about...

- `Ctrl` + `C`, `Ctrl` + `V`

- “Allgemeiner Verpeiltheitsfaktor”
  - Student was in military (“WK”) for weeks, “didn’t know” they had to hand in stuff
Issues

Things nobody can prepare you for...

Disclaimer: I don’t like calling students out for their mistakes, making mistakes is normal. But those occurrences are just too strange to not tell you about...

• \[\text{Ctrl} + \text{C}, \quad \text{Ctrl} + \text{V}\]

• “Allgemeiner Verpeiltheitsfaktor”
  Student was in military (“WK”) for weeks, “didn’t know” they had to hand in stuff

• \[r = \text{requests.get("https://random.dog/woof.json")}\]
  \[\text{data} = \text{eval}(r.\text{content})\]
Things nobody can prepare you for...

Disclaimer: I don’t like calling students out for their mistakes, making mistakes is normal. But those occurrences are just too strange to not tell you about...

- `Ctrl+C`, `Ctrl+V`

- “Allgemeiner Verpeiltheitsfaktor”
  Student was in military (“WK”) for weeks, “didn’t know” they had to hand in stuff

- `r = requests.get("https://random.dog/woof.json")`
  `data = eval(r.content)`

- `git --config user.name "student.email@example.com"`
  `git --config user.email "password-for-said-email"`
Issues

Things nobody can prepare you for...

Disclaimer: I don’t like calling students out for their mistakes, making mistakes is normal. But those occurrences are just too strange to not tell you about...

• \texttt{Ctrl} + \texttt{C}, \texttt{Ctrl} + \texttt{V}

• "Allgemeiner Verpeiltheitsfaktor"
  Student was in military ("WK") for weeks, “didn’t know” they had to hand in stuff

• \texttt{r = requests.get("https://random.dog/woof.json")}
  \texttt{data = eval(r.content)}

• \texttt{git --config user.name "student.email@example.com"}
  \texttt{git --config user.email "password-for-said-email"}

⇒ With >100 students, prepare to see every corner case you can think of, and some you’d never think of. Automation won’t help you take difficult decisions.
Automation examples

Before the semester
### Automation examples

#### Before the semester

**How long can you work on making a routine task more efficient before you’re spending more time than you save? (Across five years)**

<table>
<thead>
<tr>
<th>How Much Time You Shave Off</th>
<th>How Often You Do the Task</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50/day</td>
</tr>
<tr>
<td>1 second</td>
<td>1 day</td>
</tr>
<tr>
<td>5 seconds</td>
<td>5 days</td>
</tr>
<tr>
<td>30 seconds</td>
<td>4 weeks</td>
</tr>
<tr>
<td>1 minute</td>
<td>6 weeks</td>
</tr>
<tr>
<td>5 minutes</td>
<td>9 months</td>
</tr>
<tr>
<td>30 minutes</td>
<td>6 months</td>
</tr>
<tr>
<td>1 hour</td>
<td>10 months</td>
</tr>
<tr>
<td>6 hours</td>
<td>2 months</td>
</tr>
<tr>
<td>1 day</td>
<td>8 weeks</td>
</tr>
</tbody>
</table>

XKCD 1205, Randall Munroe / xkcd.com
Before the semester

No automation needed?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SECOND</td>
<td></td>
<td>1 DAY</td>
<td>2 HOURS</td>
<td>30 MINUTES</td>
<td>4 MINUTES</td>
<td>1 MINUTE</td>
<td>5 SECONDS</td>
</tr>
<tr>
<td>5 SECONDS</td>
<td></td>
<td>5 DAYS</td>
<td>12 HOURS</td>
<td>2 HOURS</td>
<td>21 MINUTES</td>
<td>5 MINUTES</td>
<td>25 SECONDS</td>
</tr>
<tr>
<td>30 SECONDS</td>
<td></td>
<td>4 WEEKS</td>
<td>3 DAYS</td>
<td>12 HOURS</td>
<td>2 HOURS</td>
<td>30 MINUTES</td>
<td>2 MINUTES</td>
</tr>
<tr>
<td>1 MINUTE</td>
<td></td>
<td>8 WEEKS</td>
<td>6 DAYS</td>
<td>1 DAY</td>
<td>4 HOURS</td>
<td>1 HOUR</td>
<td>5 MINUTES</td>
</tr>
<tr>
<td>5 MINUTES</td>
<td></td>
<td>9 MONTHS</td>
<td>4 WEEKS</td>
<td>6 DAYS</td>
<td>21 HOURS</td>
<td>5 HOURS</td>
<td>25 MINUTES</td>
</tr>
<tr>
<td>30 MINUTES</td>
<td></td>
<td>6 MONTHS</td>
<td>5 WEEKS</td>
<td>5 DAYS</td>
<td>1 DAY</td>
<td>2 HOURS</td>
<td></td>
</tr>
<tr>
<td>1 HOUR</td>
<td></td>
<td>10 MONTHS</td>
<td>2 MONTHS</td>
<td>10 DAYS</td>
<td>2 DAYS</td>
<td>5 HOURS</td>
<td></td>
</tr>
<tr>
<td>6 HOURS</td>
<td></td>
<td>2 MONTHS</td>
<td>2 WEEKS</td>
<td>1 DAY</td>
<td>8 WEEKS</td>
<td>5 DAYS</td>
<td></td>
</tr>
<tr>
<td>1 DAY</td>
<td></td>
<td>2 MONTHS</td>
<td>2 WEEKS</td>
<td>1 DAY</td>
<td>8 WEEKS</td>
<td>5 DAYS</td>
<td></td>
</tr>
</tbody>
</table>

XKCD 1205, Randall Munroe / xkcd.com
Automation examples

Before the semester

No automation needed?

- People get added last-minute
- …after preparing
- …even after semester started
### Automation examples

#### Before the semester

**How long can you work on making a routine task more efficient before you’re spending more time than you save? (Across five years)**

<table>
<thead>
<tr>
<th>How often you do the task</th>
<th>50/day</th>
<th>5/day</th>
<th>Daily</th>
<th>Weekly</th>
<th>Monthly</th>
<th>Yearly</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 second</td>
<td>1 day</td>
<td>2 hours</td>
<td>30 minutes</td>
<td>4 minutes</td>
<td>1 minute</td>
<td>5 seconds</td>
</tr>
<tr>
<td>5 seconds</td>
<td>5 days</td>
<td>12 hours</td>
<td>2 hours</td>
<td>21 minutes</td>
<td>5 minutes</td>
<td>25 seconds</td>
</tr>
<tr>
<td>30 seconds</td>
<td>4 weeks</td>
<td>3 days</td>
<td>12 hours</td>
<td>2 hours</td>
<td>30 minutes</td>
<td>2 minutes</td>
</tr>
<tr>
<td>1 minute</td>
<td>6 weeks</td>
<td>1 day</td>
<td>4 hours</td>
<td>1 hour</td>
<td>5 minutes</td>
<td>25 seconds</td>
</tr>
<tr>
<td>5 minutes</td>
<td>9 months</td>
<td>4 weeks</td>
<td>16 days</td>
<td>21 hours</td>
<td>5 hours</td>
<td>25 seconds</td>
</tr>
<tr>
<td>30 minutes</td>
<td>6 months</td>
<td>5 weeks</td>
<td>5 days</td>
<td>1 day</td>
<td>2 hours</td>
<td>5 hours</td>
</tr>
<tr>
<td>1 hour</td>
<td>10 months</td>
<td>2 months</td>
<td>10 days</td>
<td>2 days</td>
<td>5 hours</td>
<td>25 seconds</td>
</tr>
<tr>
<td>6 hours</td>
<td>2 months</td>
<td>2 weeks</td>
<td>2 weeks</td>
<td>1 day</td>
<td>8 weeks</td>
<td>5 days</td>
</tr>
<tr>
<td>1 day</td>
<td>8 weeks</td>
<td>5 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### No automation needed?

- People get added last-minute
- ...after preparing
- ...even after semester started
- People leave in the middle of the semester
- ...and nobody tells you
Automation examples

Before the semester

• No API (as far as I know)
• Lots of data (200 KB of JSON) in `window.adunisModel = ...`
• Not what we need, however...
• But HTML is structured enough. `requests` and `bs4` to the rescue!
Automation examples
Before the semester

- No API (as far as I know)
- Lots of data (200 KB of JSON) in `window.adunisModel = ...`
- Not what we need, however...
- But HTML is structured enough. `requests` and `bs4` to the rescue!

- Weird Microsoft-based login flow
- Couldn’t figure out how it works, libraries seem to be for APIs only
Automation examples
Before the semester

• No API (as far as I know)
• Lots of data (200 KB of JSON) in `window.adunisModel = ...`
• Not what we need, however...
• But HTML is structured enough. `requests` and `bs4` to the rescue!

• Weird Microsoft-based login flow
• Couldn’t figure out how it works, libraries seem to be for APIs only

• Whatever, I write a browser since 2013, and I can access cookies
• Login via QtWebEngine browser (injected JS to fill values)
• Grab session cookie, feed it to `requests` 😎
Automation examples

During the semester: Commander

```python
a = -2.7
b = 3.4

a_abs = abs(a)
b_abs = abs(b)

a_rounded = round(a)
b_rounded = round(b)

bigger = min(a, b)
smaller = min(a, b)
```

b) Type-related Functions

There are also functions to get the type of an object:

```python
tb = <conftest.TBWrapper object at 0x7ff66e336d90>
expected = 3.4

@pytest.mark.parametrize("var, expected",
    [(
        "a", -2.7),
        ("b", 3.4),
        ("a_abs", 2.7),
        ("b_abs", 3.4),
        ("a_rounded", -3),
        ("b_rounded", 3),
        ("bigger", 3.4),
        ("smaller", -2.7)])
```

<table>
<thead>
<tr>
<th>nodeid</th>
<th>tag</th>
<th>when</th>
<th>outcome</th>
<th>details</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_builtin_math.py::test_values[bigger-3.4]</td>
<td>built-in-functions-math-functions</td>
<td>call</td>
<td>failed</td>
<td>assert -2.7 == 3.4</td>
</tr>
<tr>
<td>test_builtin_help.py::test_output</td>
<td>built-in-functions-help</td>
<td>call</td>
<td>passed</td>
<td></td>
</tr>
<tr>
<td>test_builtin_help.py::test_ast</td>
<td>built-in-functions-help</td>
<td>call</td>
<td>passed</td>
<td></td>
</tr>
</tbody>
</table>

Focus (Ctrl+1/2/3/4/5): Test (Ctrl+T): Folder (Ctrl+F): Branch (Ctrl+B): Zapper (Ctrl+J): 105 passed 2 failed 0 error = 5.91
### Automation examples

#### During the semester: Overview

<table>
<thead>
<tr>
<th>Name</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>16</th>
<th>17</th>
<th>project</th>
</tr>
</thead>
<tbody>
<tr>
<td>florian.bruhin</td>
<td>5.91</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>urs.baumann</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>test.testerli</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

*pmisc*  
100% 0:00:00
Automation examples

During the semester: Grep

$ python3 submission-grep.py reports 2.7

1-getting-started/01-python-basic test_builtin_math.py::test_values
   assert -2.7 == 3.4

florian.bruhin

100% 0:00:00

$ python3 submission-grep.py nodeid values

florian.bruhin

1-getting-started/01-python-basic test_builtin_math.py::test_values
   assert -2.7 == 3.4

florian.bruhin

100% 0:00:00
## Automation examples

Towards end of semester: Project overview

<table>
<thead>
<tr>
<th>Name</th>
<th>Project</th>
<th>Files</th>
<th>Git</th>
<th>types</th>
<th>cli</th>
<th>black</th>
<th>rich</th>
<th>pillow</th>
<th>poetry</th>
<th>pytest</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>florian.bruhin urs.baumann</td>
<td>✔️ ✔️</td>
<td>test.txt AutPy_CLI/.gitignore AutPy_CLI/README.md AutPy_CLI/wuff.py ./.git (Inhalte übersprungen) project/Python_Projekt_St...</td>
<td>❌</td>
<td>❌</td>
<td>✔️</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>0</td>
</tr>
<tr>
<td>test.testerli</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

100% 0:00:00
Automation examples

After end of semester: Project grading

Project grading workflow:

- Pick random student
- Get zip from submissions repository
- Unpack zip in “grading-area” folder
Automation examples
After end of semester: Project grading

Project grading workflow:

- Pick random student
- Get zip from submissions repository
- Unpack zip in “grading-area” folder
- Show overview (file list, detected features)
- Prepare checklist and open in editor
- Wait until editor closed
Automation examples

After end of semester: Project grading

Project grading workflow:

• Pick random student
• Get zip from submissions repository
• Unpack zip in “grading-area” folder
• Show overview (file list, detected features)
• Prepare checklist and open in editor
• Wait until editor closed

• Parse checklist
• Show parsed points and grade, wait for confirmation
• Commit grading file to submissions repository
• ...and during the whole process, only show names as rot13
Automation examples
After end of semester: Project grading

Project grading workflow:

• Pick random student
• Get zip from submissions repository
• Unpack zip in “grading-area” folder
• Show overview (file list, detected features)
• Prepare checklist and open in editor
• Wait until editor closed

• Parse checklist
• Show parsed points and grade, wait for confirmation
• Commit grading file to submissions repository

• ...and during the whole process, only show names as rot13
Automation examples

After end of semester: Project grading

Project grading workflow:

- Pick random student
- Get zip from submissions repository
- Unpack zip in “grading-area” folder
- Show overview (file list, detected features)
- Prepare checklist and open in editor
- Wait until editor closed
- Parse checklist
- Show parsed points and grade, wait for confirmation
- Commit grading file to submissions repository
- ...and during the whole process, only show names as rot13

\[
\text{florian.bruhin}
\]

\[
\text{sybevna.oehuva}
\]
Automation examples
After end of semester: Project grading

# Functionality (24P)

- Data download / reading (7P)
  - [ ] Download URL is obtained via API (2P)
  - [ ] Latest available data set used by default (1P)
  - ...

- [ ] Searching for dogs (2P)

- Statistics (9P)
  - [ ] Longest dog name is output correctly (0.5P)
  - [ ] Shortest dog name is output correctly (0.5P)
  - [ ] Top 10 is output correctly (1P)
  - ...

Automation examples
After end of semester: Project grading

- Parse Markdown checklist
- Calculate points
- Send HTML + plaintext mails

<table>
<thead>
<tr>
<th>Thema</th>
<th>Punkte Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Funktionalität (24P)</td>
<td>18.5</td>
</tr>
<tr>
<td>Error Handling (10P)</td>
<td>4</td>
</tr>
<tr>
<td>Best Practices (20P)</td>
<td>7.5</td>
</tr>
<tr>
<td>Nutzer-Sicht (6P)</td>
<td>6</td>
</tr>
<tr>
<td>git</td>
<td>8</td>
</tr>
<tr>
<td>rich</td>
<td>8</td>
</tr>
<tr>
<td>Zusatzpunkte</td>
<td>0</td>
</tr>
<tr>
<td><strong>Punkte total</strong></td>
<td><strong>52.0</strong></td>
</tr>
</tbody>
</table>

Funktionalität (24P)

- Daten-Download / Einlesen (7P)
  - Daten-URL wird via API bezogen (2P)
  - Standardmässig neuster verfügbare Datensatz (1P): 2021 hardcoded
Automation examples

After end of semester: Final grade

<table>
<thead>
<tr>
<th>Thema</th>
<th>Gewichtung</th>
<th>Punkte</th>
<th>Max</th>
<th>ca. %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab 01</td>
<td>107.0</td>
<td>107</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td><strong>Block 1</strong></td>
<td>107.0</td>
<td>107</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td>Lab 02</td>
<td>59.0</td>
<td>59</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td>Lab 03</td>
<td>47.0</td>
<td>47</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td><strong>Block 2</strong></td>
<td>1/6</td>
<td>106.0</td>
<td>106</td>
<td>100.0%</td>
</tr>
<tr>
<td>Lab 04</td>
<td>24.0</td>
<td>24</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td>Lab 05</td>
<td>55.0</td>
<td>55</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td>Lab 06</td>
<td>38.0</td>
<td>38</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td><strong>Block 3</strong></td>
<td>1/6</td>
<td>117.0</td>
<td>117</td>
<td>100.0%</td>
</tr>
<tr>
<td>Lab 07</td>
<td>60.0</td>
<td>60</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td>Lab 08</td>
<td>4.0</td>
<td>4</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td>Lab 09</td>
<td>18.0</td>
<td>18</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td>Lab 10</td>
<td>48.0</td>
<td>48</td>
<td>100.0%</td>
<td></td>
</tr>
<tr>
<td><strong>Block 4</strong></td>
<td>1/6</td>
<td>130.0</td>
<td>130</td>
<td>100.0%</td>
</tr>
<tr>
<td>Lab 16 (nicht abgegeben)</td>
<td>0.0</td>
<td>7</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Lab 17 (nicht abgegeben)</td>
<td>0.0</td>
<td>5</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td><strong>Block 5</strong></td>
<td>1/6</td>
<td>74.5</td>
<td>80</td>
<td>93.1%</td>
</tr>
</tbody>
</table>

Nach entsprechender Gewichtung führt das zu ca. 81.04%, bzw. zur Note: $5 \times 389 / 480 + 1 \approx 5.0521$

Eingetragene Note, auf Viertelsnoten gerundet: 5
Next steps

- Teaching another ≈ 100 students, with many small improvements!
- Grading another ≈ 100 student projects...
Next steps

- Teaching another ≈ 100 students, with many small improvements!
- Grading another ≈ 100 student projects...
- Turning this into a proper package (autpy? autopypy? pyautpy? autmetapy?)
- Type annotations and autoformatters (done!)
- Tests for all the automation logic...
- Using GitPython/pygitops/pygit2/Dulwich/Gittle/… instead of subprocess (nicer API and performance)
Next steps

- Teaching another $\approx 100$ students, with many small improvements!
- Grading another $\approx 100$ student projects...
- Turning this into a proper package (autpy? autpypy? pyautpy? autmetapy?)
- Type annotations and autoformatters (done!)
- Tests for all the automation logic...
- Using GitPython/pygitops/pygit2/Dulwich/Gittle/… instead of subprocess (nicer API and performance)
- Maybe: Generalizing and publishing?