
BETTER CODE FOR
DATA SCIENCE
ALEXANDER CS HENDORF
@ PYDATA GLOBAL 2020

KŌNIGSWEG

ALEXANDER C. S. HENDORF

MANAGING PARTNER & PRINCIPAL CONSULTANT
DATA SCIENCE & AI AT KÖNIGSWEG.

PYTHON SOFTWARE FOUNDATION FELLOW, PYTHON SOFTWAREVERBAND CHAIR,
PYCONDE & PYDATA BERLIN CHAIR, LOCAL PYDATA COMMUNITY ORGANIZER

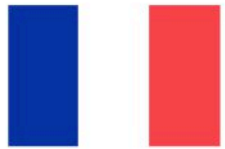
@HENDORF

AH@KOENIGSWEG.COM

PYDATA FRANKFURT

PyData Frankfurt

WEDNESDAY MAY 22 18:00
TECHQUARTIER FRANKFURT



— SciKit-Learn Keynote

Olivier Griesel
INRIA

KPMG

— Jupyter Keynote

Sylvain Corlay
Quantstack

KŌNIGSWEG

PYDATA SÜDWEST

THU, NOV 21 18:00 - 21:30
X-HOUSE, HEIDELBERG (@MAIN STATION)

PyData Südwest

– A.I. Caramba: How to Maintain Sanity

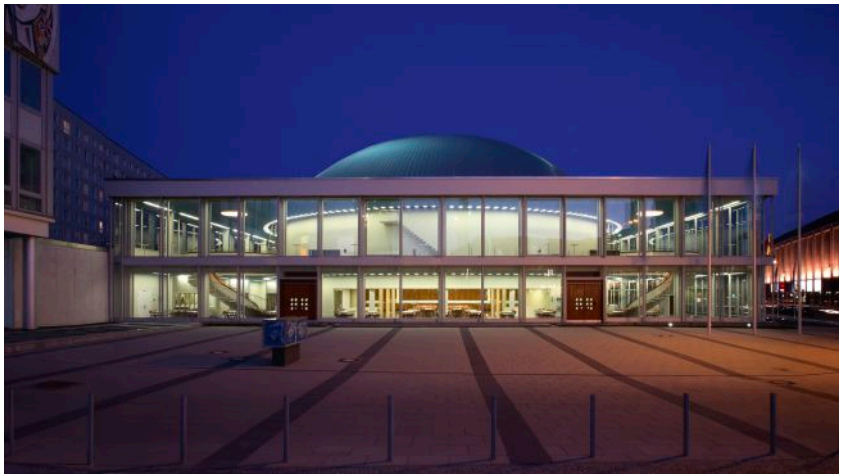
Vincent D. Warmerdam
GoDataDriven

HEIDELBERGCEMENT

– 2119: A Data Science Escape Room

Konrad Heimpel
Getsafe

KÖNIGSWEG



PyCon.DE

PyData Berlin

PyConDE & PyData Berlin 2021

Oct 13–15 2021
bcc Berlin Congress Center

KŌNIGSWEG

KŌNIGSWEG


KÖNIGSWEG
presents
WHAT'S MY LINE?

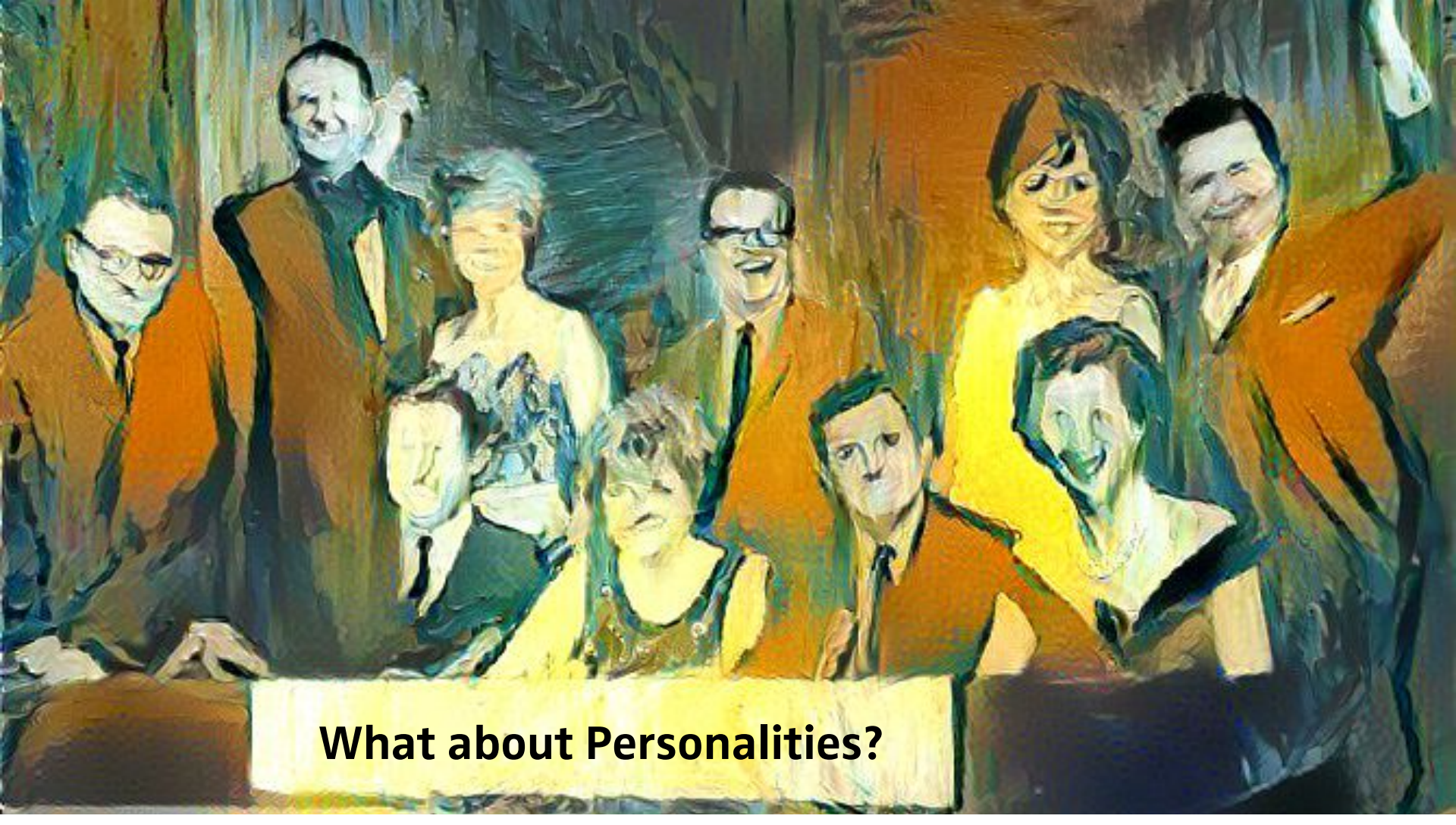
Why do Programming Skills Matter?

- Quality Assurance
- Focus on value, not bugs

- productivity my vary by 10x*

- Maintainability
- Reusability

- * https://www.construx.com/blog/the-origins-of-10x-how-valid-is-the-underlying-research/

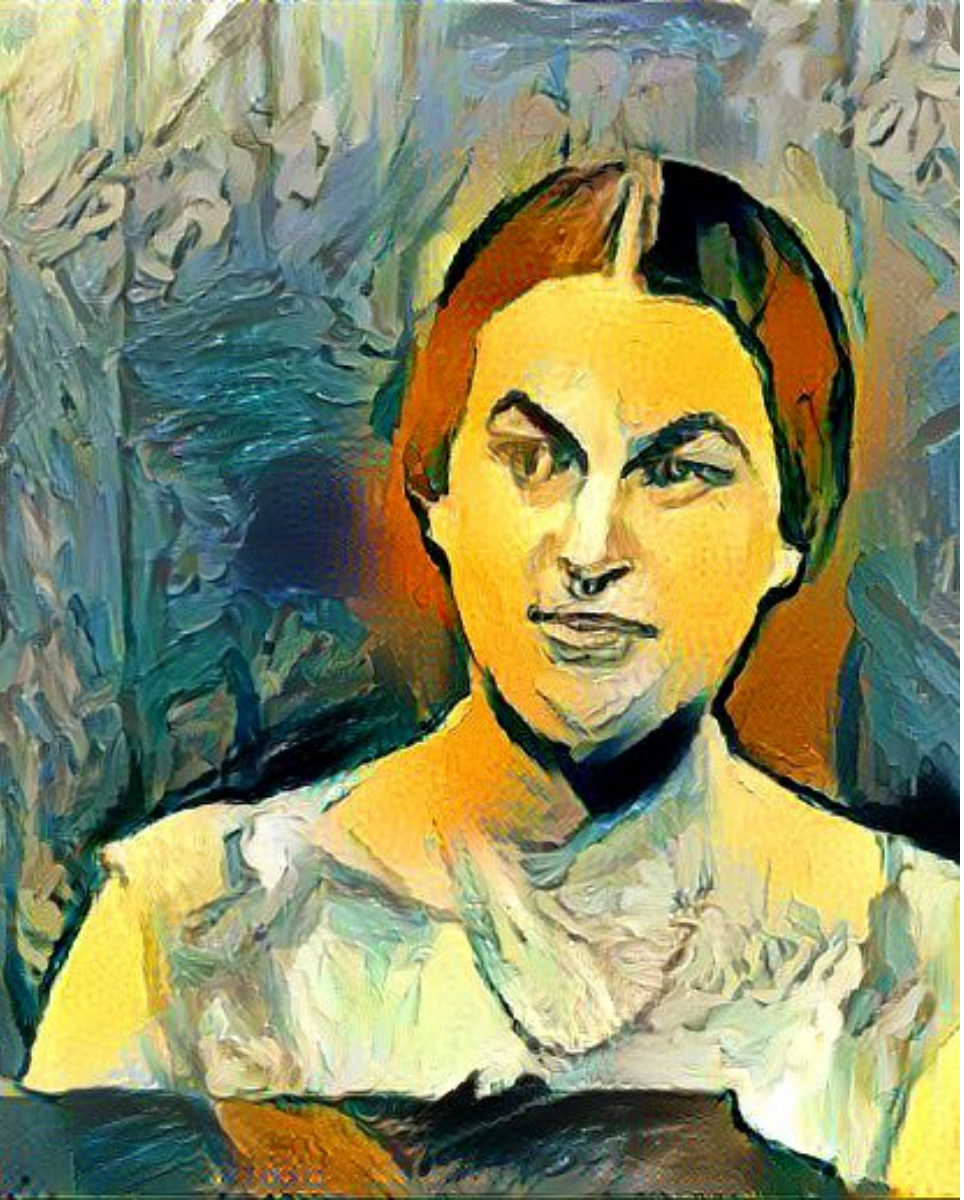

What about Personalities?

Some Personas

Stormy Simone	Coding to get things done studying, constantly overexcited about the latest paper
Abstract Alec	Just left academia, high level of problem abstraction, lacks production experience
Steady Saša	Skilled in another programming language, looking forward to learning new skills
Sceptical Stéphane	Skilled in another programming language, sceptical to new things, constantly comparing
Determined Dylan	Self-taught programmer with strong domain expertise, feels constantly not good enough as programmer
Leading Lian	Superior who used to program back in the days tends to throw too many good ideas in the room
Master Michi	Experienced coder and architect, knows personal limits und unknowns

What Workflows Do We Need?

- Clear
- Simple
- Minimal
- Stable



How Should We Set-Up Our Code?

- Have Principles!

- Coding
- Architecture
- Stack

Principles I

PEP20 The Zen of Python, by Tim Peters

```
>>> import this
```

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!

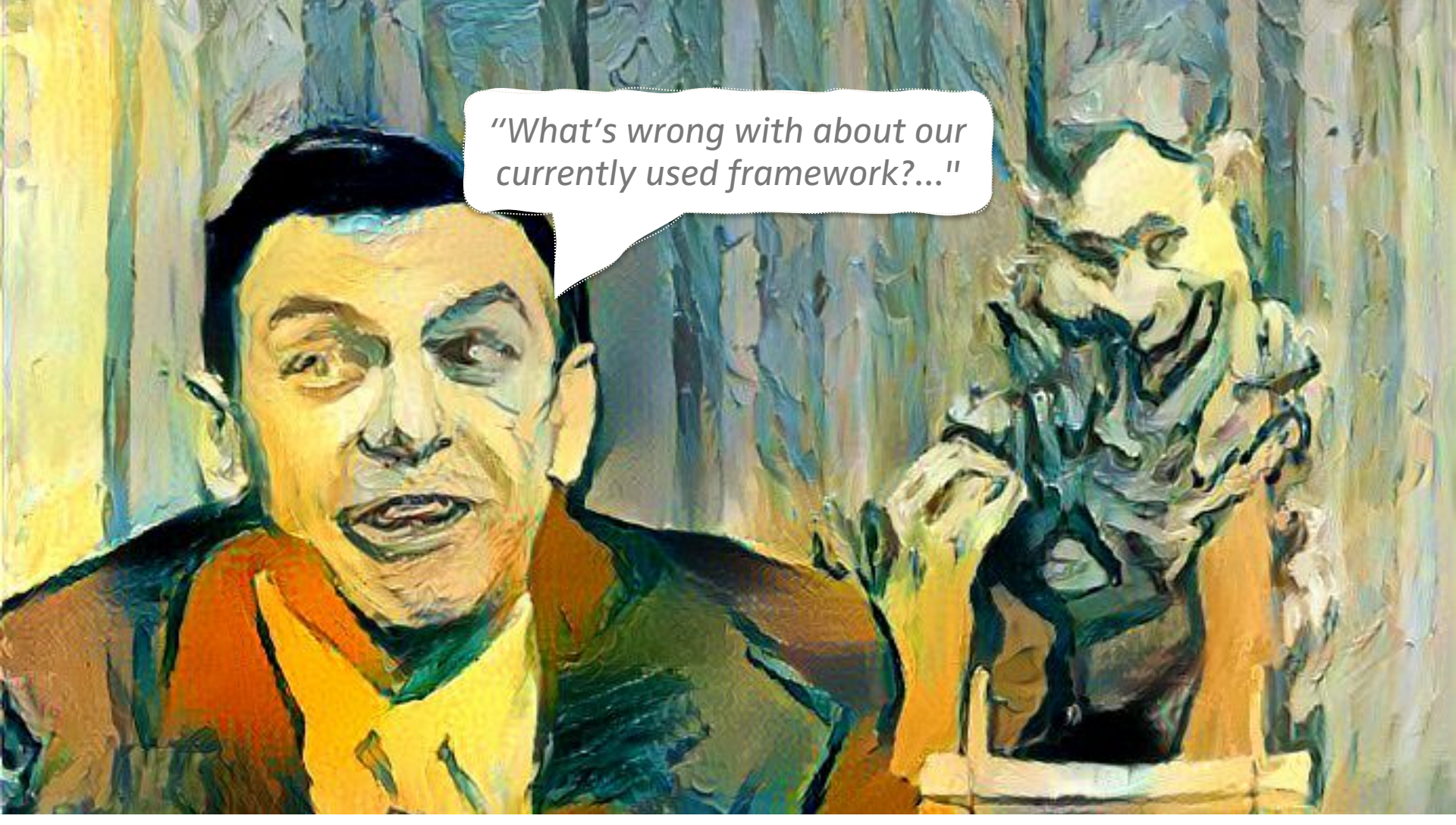
Principles II

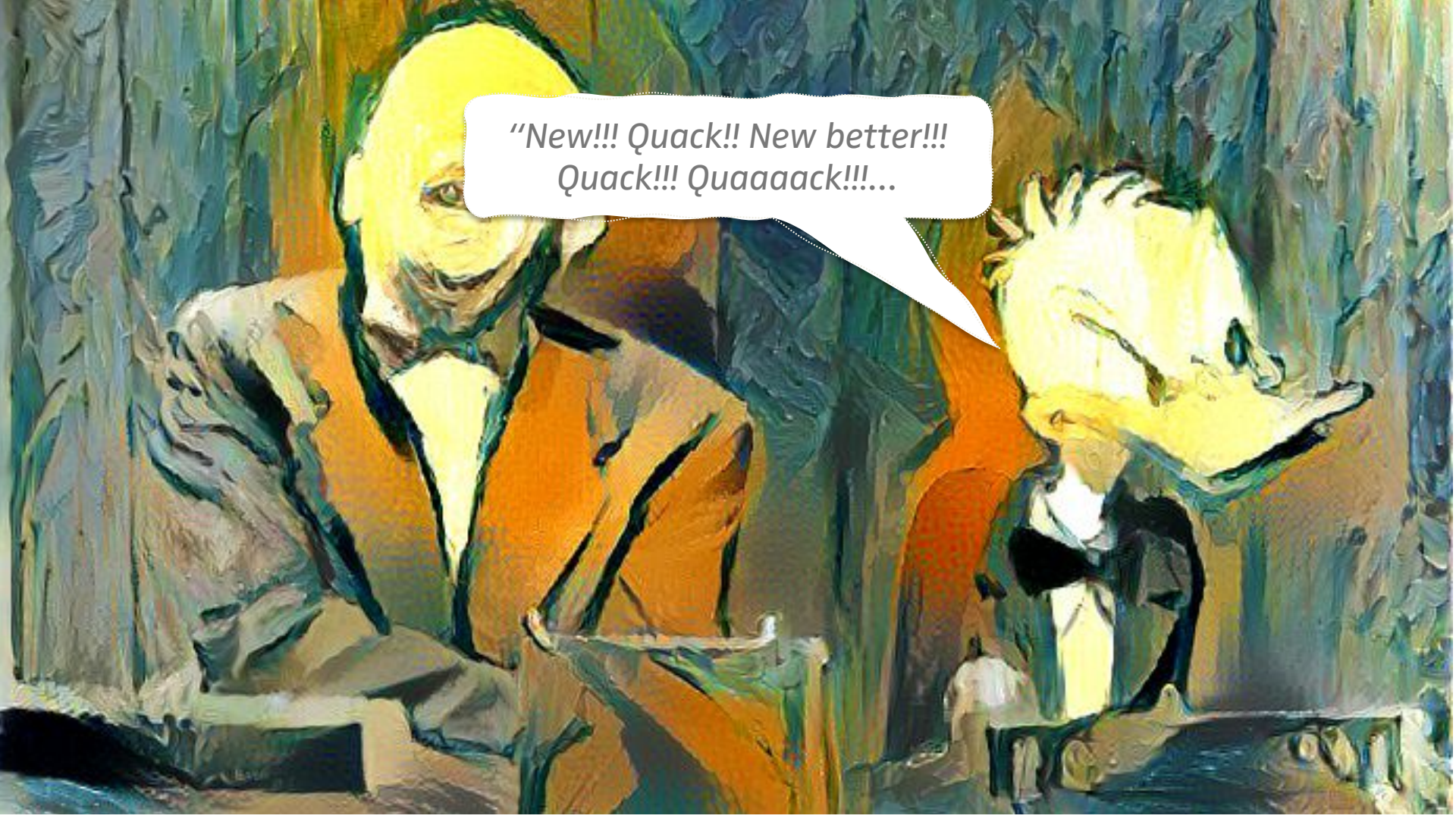
Architecture

- Clear
- Simple
- Minimal
- Stable
- Scalable
- Independent


“Saw a post on LinkedIn about that framework..."


"I have this new framework here..."


“What’s wrong with about our currently used framework?..."



KŌNIGSWEG

Principles III
Software Stack

- Defined
- Minimal
- Stable

- Set up for experimentation
- Extendable

Stack:
Python Standard Library

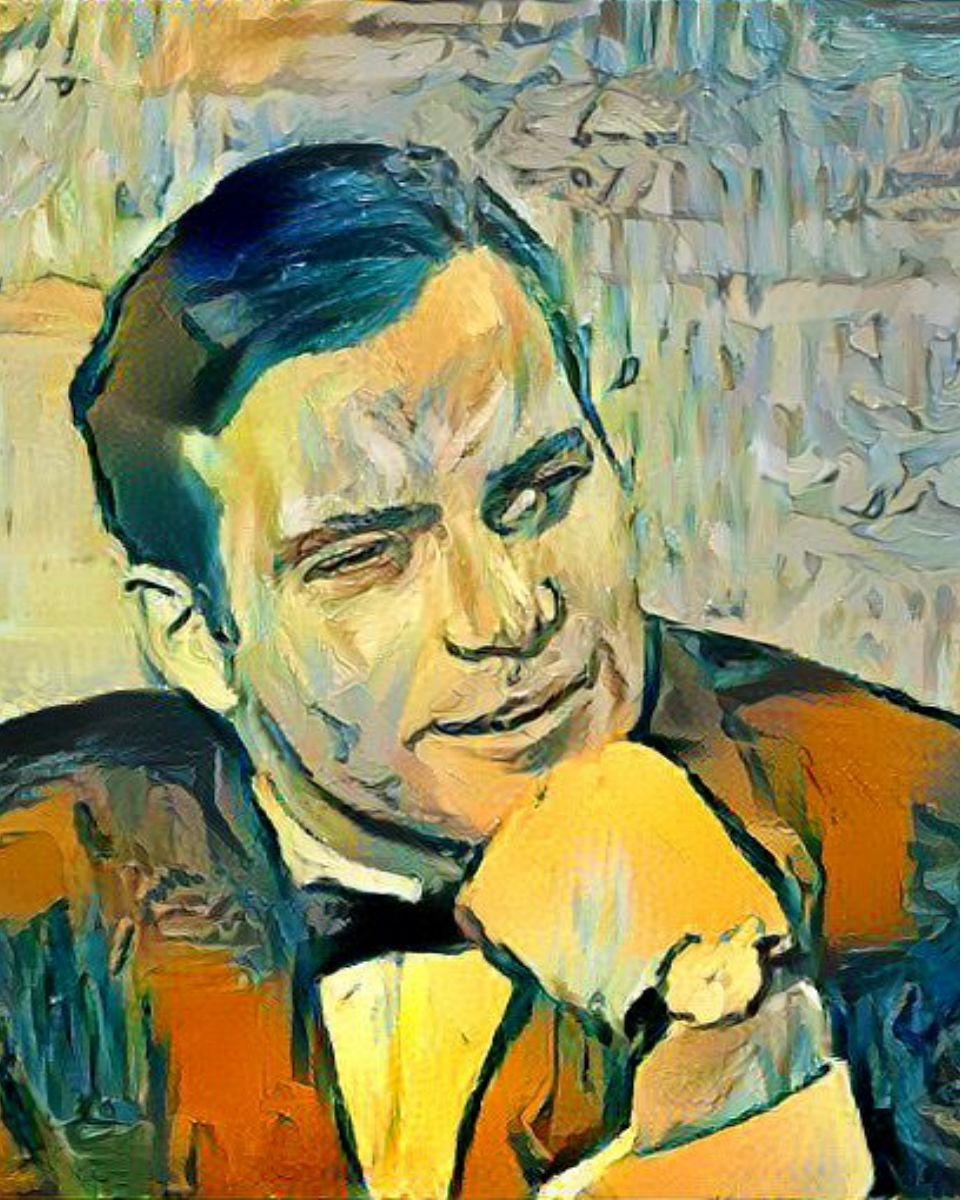
- Comes with Python
- Many useful libraries
- Look at standard lib first!

David Beazley | Keynote: Built in Super Heroes at PyData Chicago
Video: https://youtu.be/lyDLAutA88s .

What About Environments?

- Spaces to program within
- Environment variables
- Config files in many places
- Bash, zsh vars setups on load
- Management with conda, pip,...
- Reproducibility
- IDE
- Environment clones to try new stuff

Menu of the Day: Spaghetti Code



What Makes Code Good or Bad?

- Readable
- Maintainable
- Shareable
- Reliable
- Documented

What's Code Readability?

- Concise, descriptive names
- Explicit code
- Split long lines, 2 pages on one screen
- Break complex or long
  into smaller functions
- No fancy stuff because you can
- Add line comments to provide context
- Linted with autopep8/black et al.

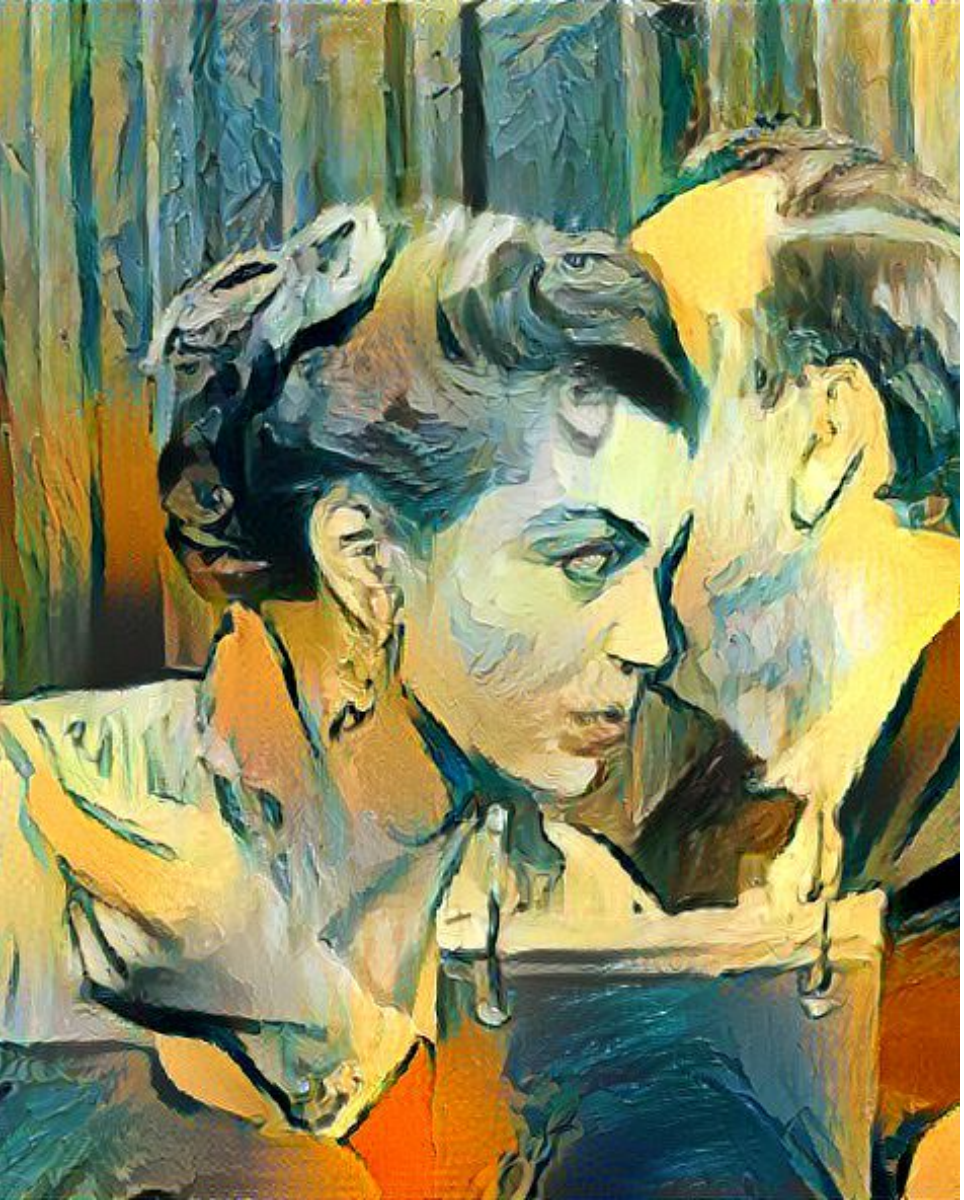
- Define your own style guide for stability

Tip: Dataclasses

- Be explicit if more than one return value
- Dataclasses provide useful interfaces with benefits
-

What Makes Code Maintainable?

- Consistent style and approaches
- Self-explanatory
- Single points of failure / DRY Principle
- Well origanised
- Tests



How Can I Share My Code?

- Use git
- Commit often
- Use a repository (service)
- .gitignore secrets
- Guidelines how to make pull requests
- Team reviews are good

- CI/CD with auto tests and publishing

What Makes Code Reliable?

- Testing
- Systems are not reliable
  use explicit Exceptions to handle
- Type hints
- Simplicity

What Does Documented Mean?

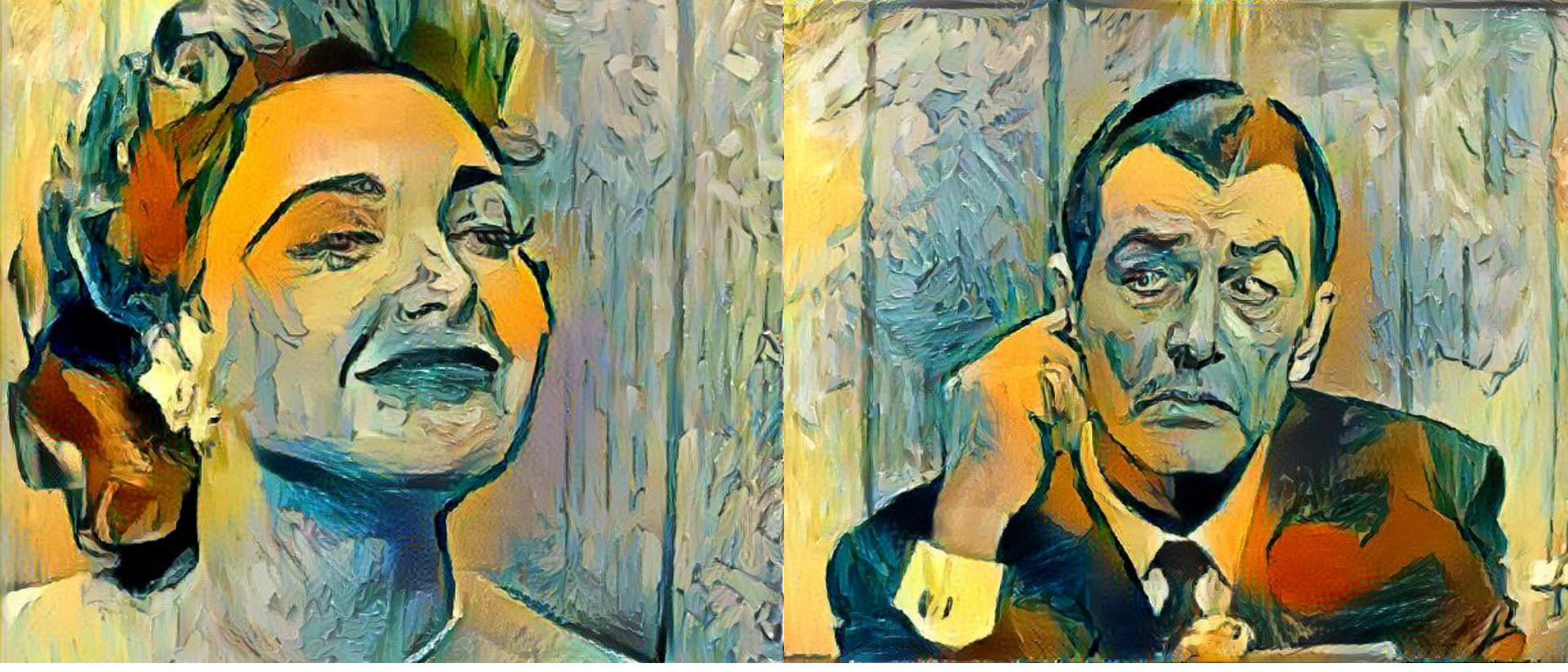
- READMEs
- Docstrings
- Line Comments
- Working Example Code

- Do not state the obvious
- Concise, not chatty
- Constantly update and refactor

Breaking the Rules!

- Know when to break the rules
- Have a good reason
- Be explicit about it

KŌNIGSWEG



Jupyter Notebooks to Production

KŌNIGSWEG

What About Jupyterlab?

- Exploratory Data Analysis
- Exploratory Code Execution
- Open Source
- Visualisation
- Formats / MD-Texts
- Ecosystem

Director of Artificial Intelligence and Autopilot Vision at Tesla

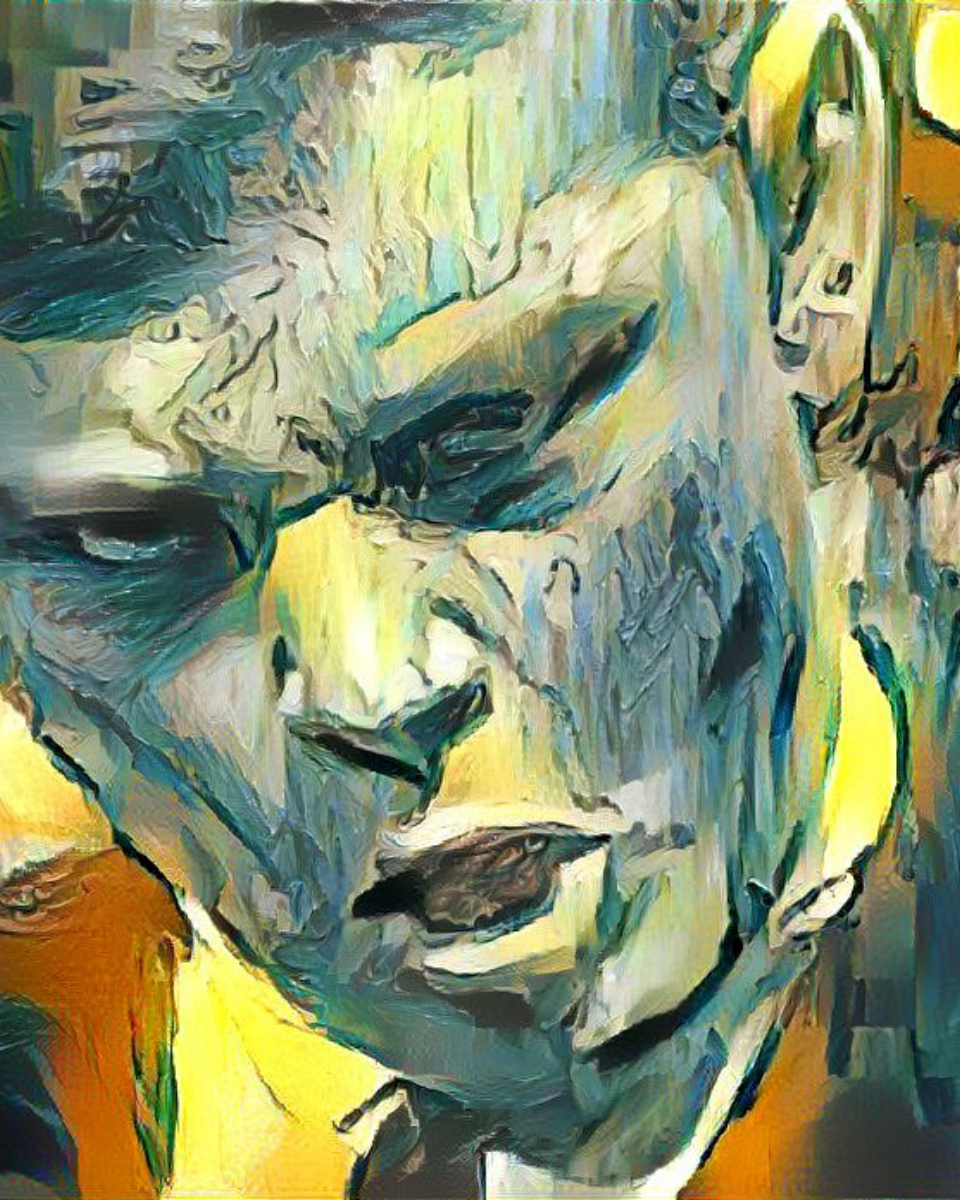
Andrej Karpathy @karpathy · 28 Min.
so I accidentally held down something and deleted all cells in this jupyter notebook I've been building for ~2 months, and the "undo delete cell" isn't bringing them back. Lol.
52 23 319

Andrej Karpathy @karpathy · 23 Min.
I'm still shook. Some jupyter hotkey, somehow held down with my left palm, just iteratively deletes everything and undo doesn't bring them back (it creates an empty cell only). Hug your favorite notebooks and keep them safe ❤️
24 6 226

Andrej Karpathy @karpathy · 5 Min.
thanks everyone, I was luckily able to find a snapshot in the .ipynb_checkpoints/ folder. You know that annoying thing you always add to the top of your .gitignore? turns out it can actually be useful :)
3 1 75

Aug 30 2020, https://twitter.com/karpathy/status/1299972064426651650

No Comment.

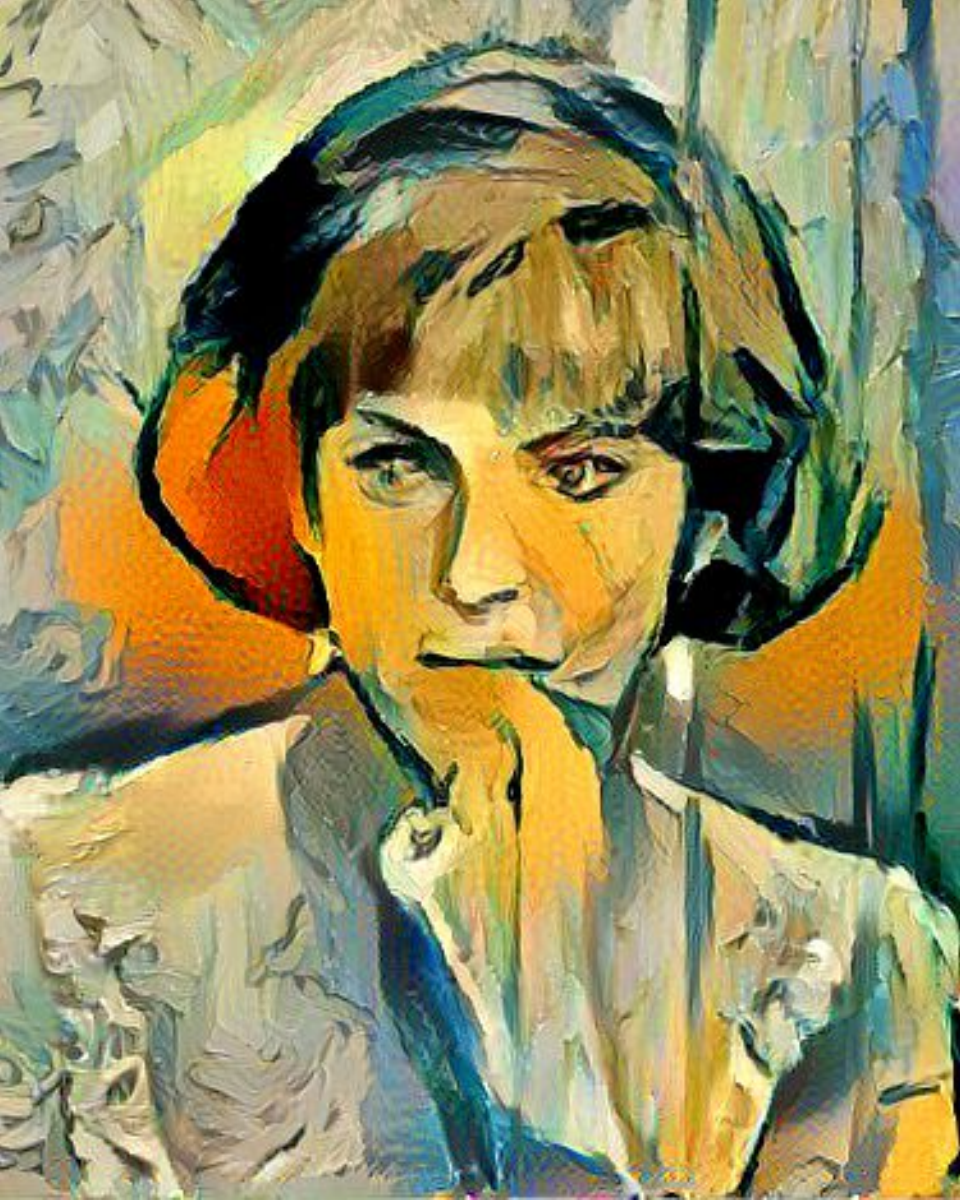


Any Downsides Using Notebooks?

- Code Completion
- Pointing Code Smells
- Code Versioning

- Notebooks are scripts
- Execution order
- Introspection
- Debugging

Guessing the Occupation



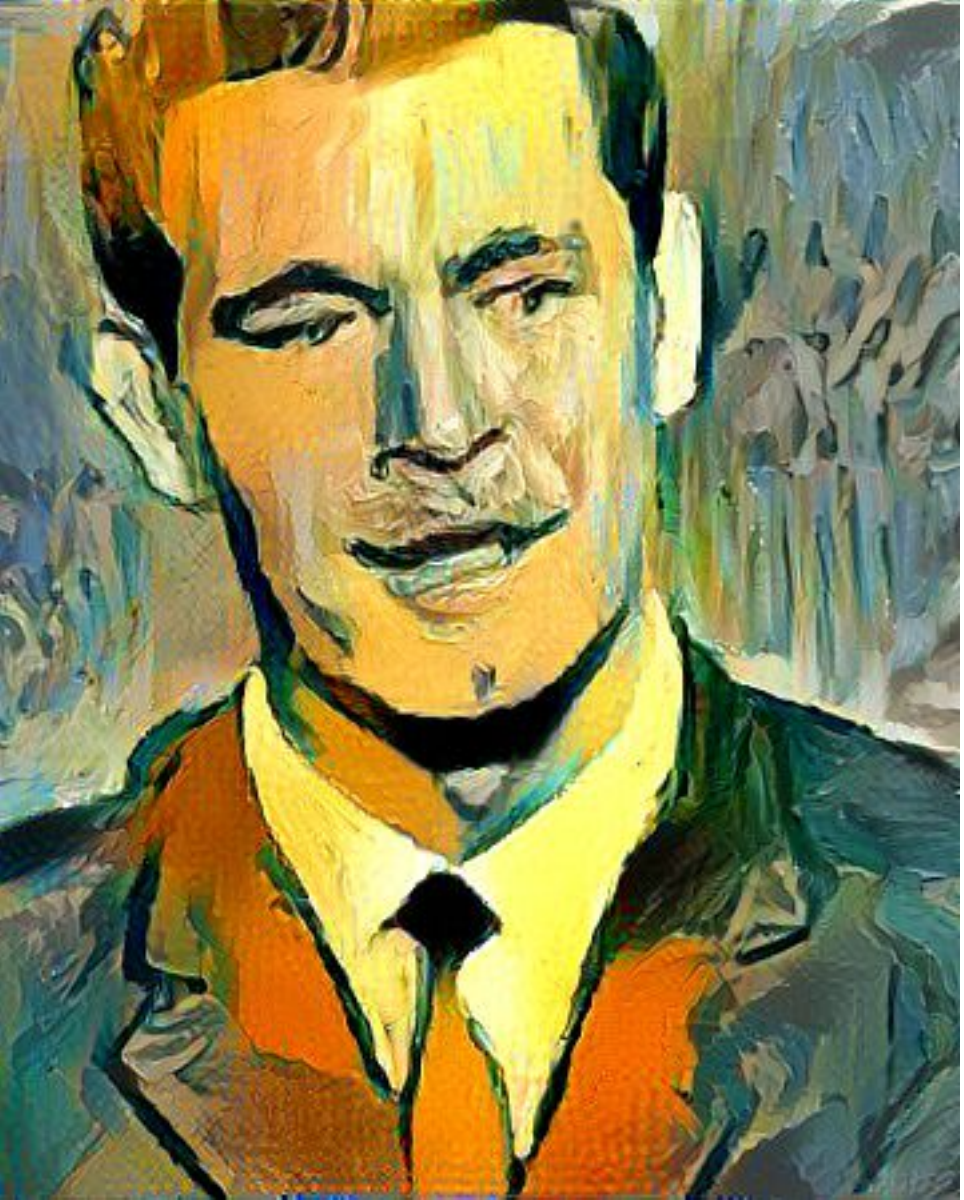
What Common Antipatterns in Notebooks?

- Wall of code
- Non-local variables
- Redundancies

- Lack of versioning
- Missing documentation

What Can I Do About Wall of Code?

- Use multiple cells
- Break down into function
- Break down into multiple functions
- Classes
- Use modules



What About Non-local Variables?

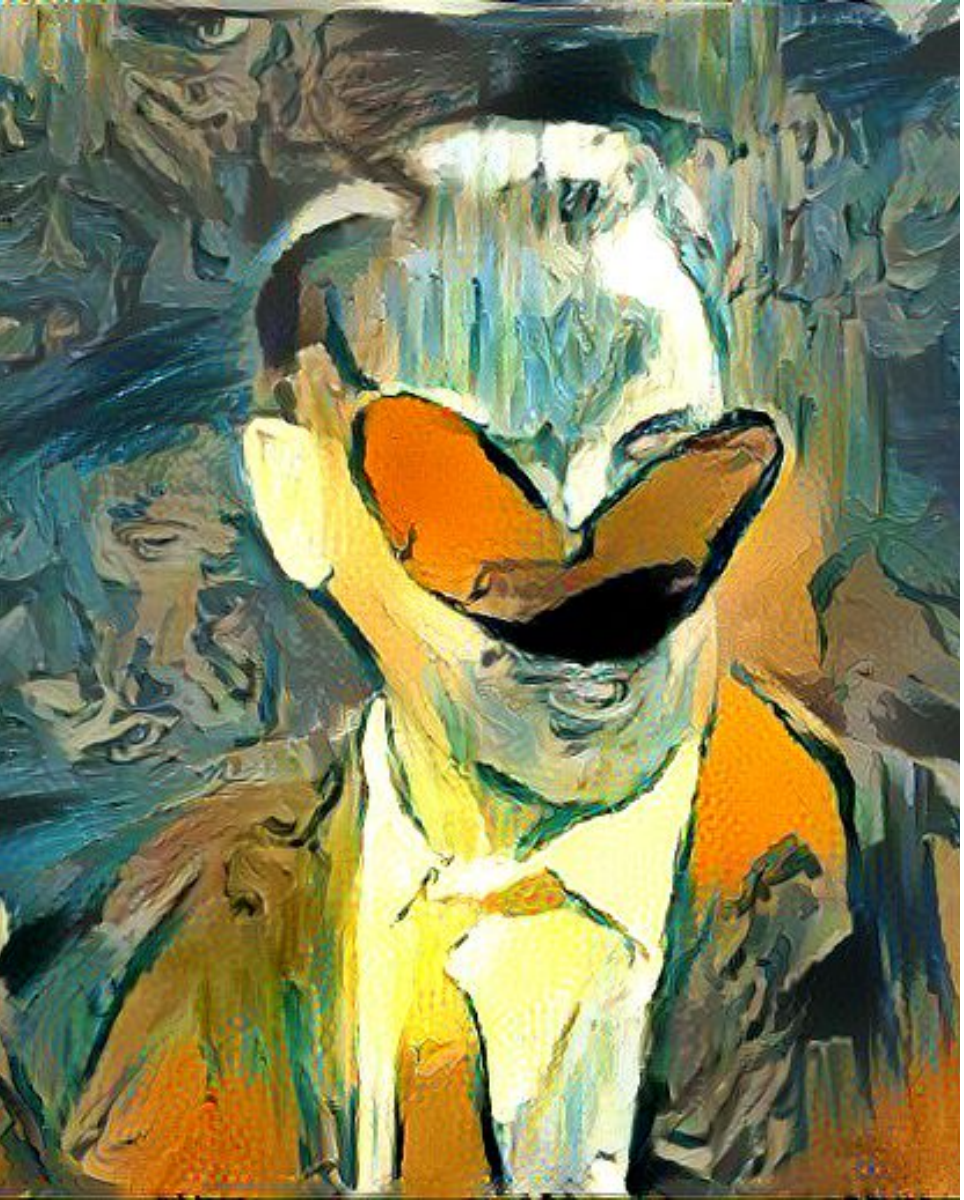
- Hard to refactor
- Hard to move code with non-local variables

- Use function parameters
- Use classes and attributes

What Can I Do About Redundancies?

- Avoid them ;)

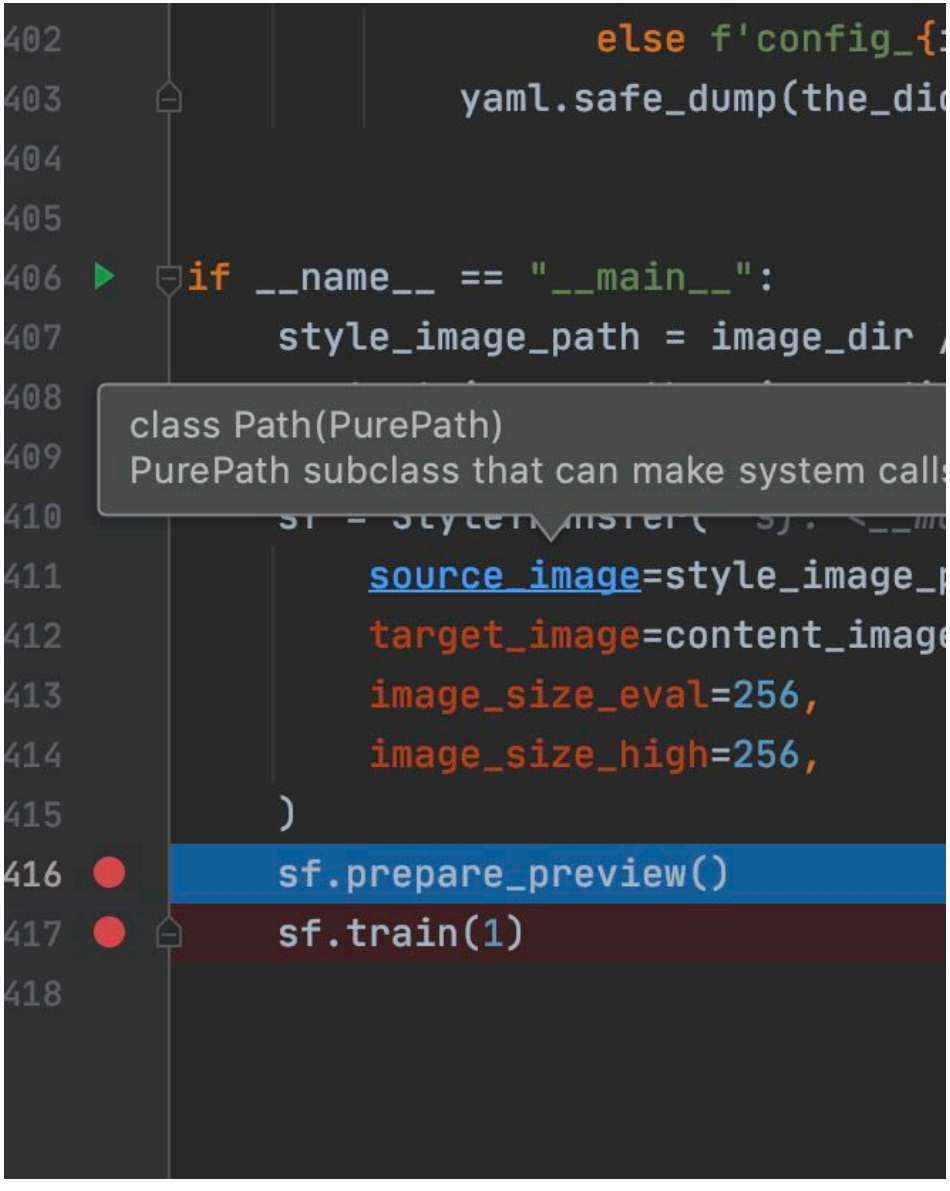
- Use functions
- DRY-principle
- Constant Refactoring

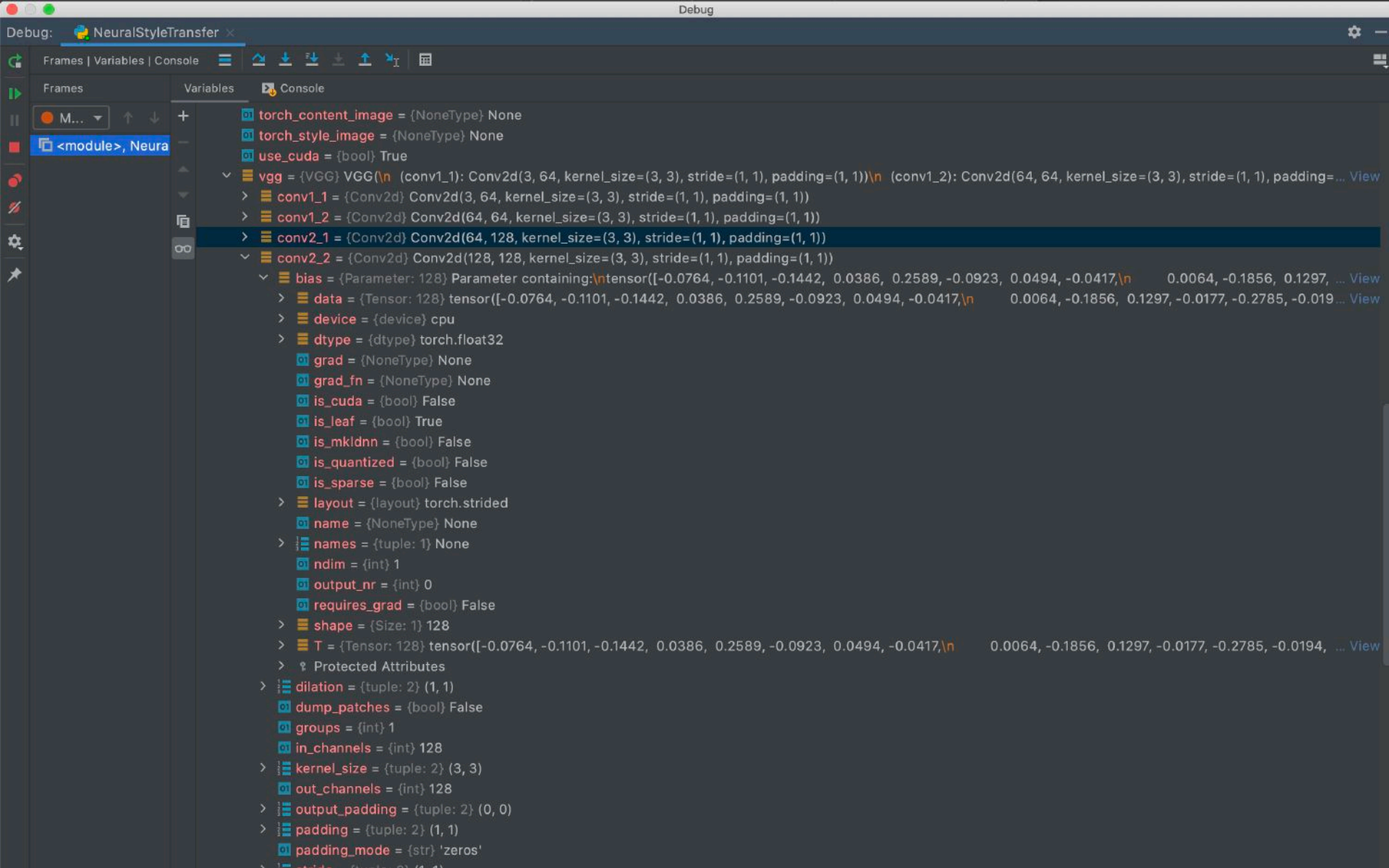


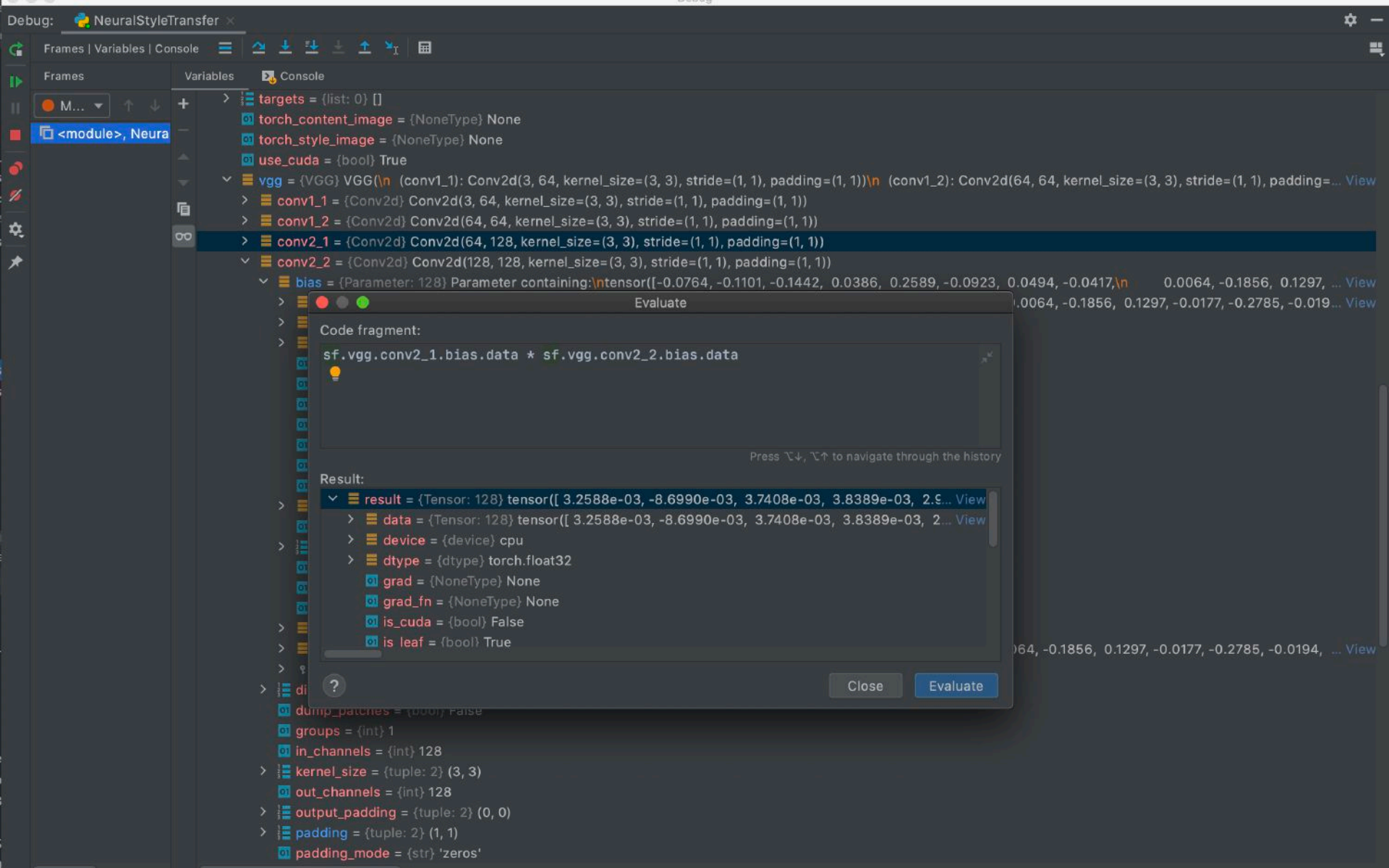
How Can I Debug?

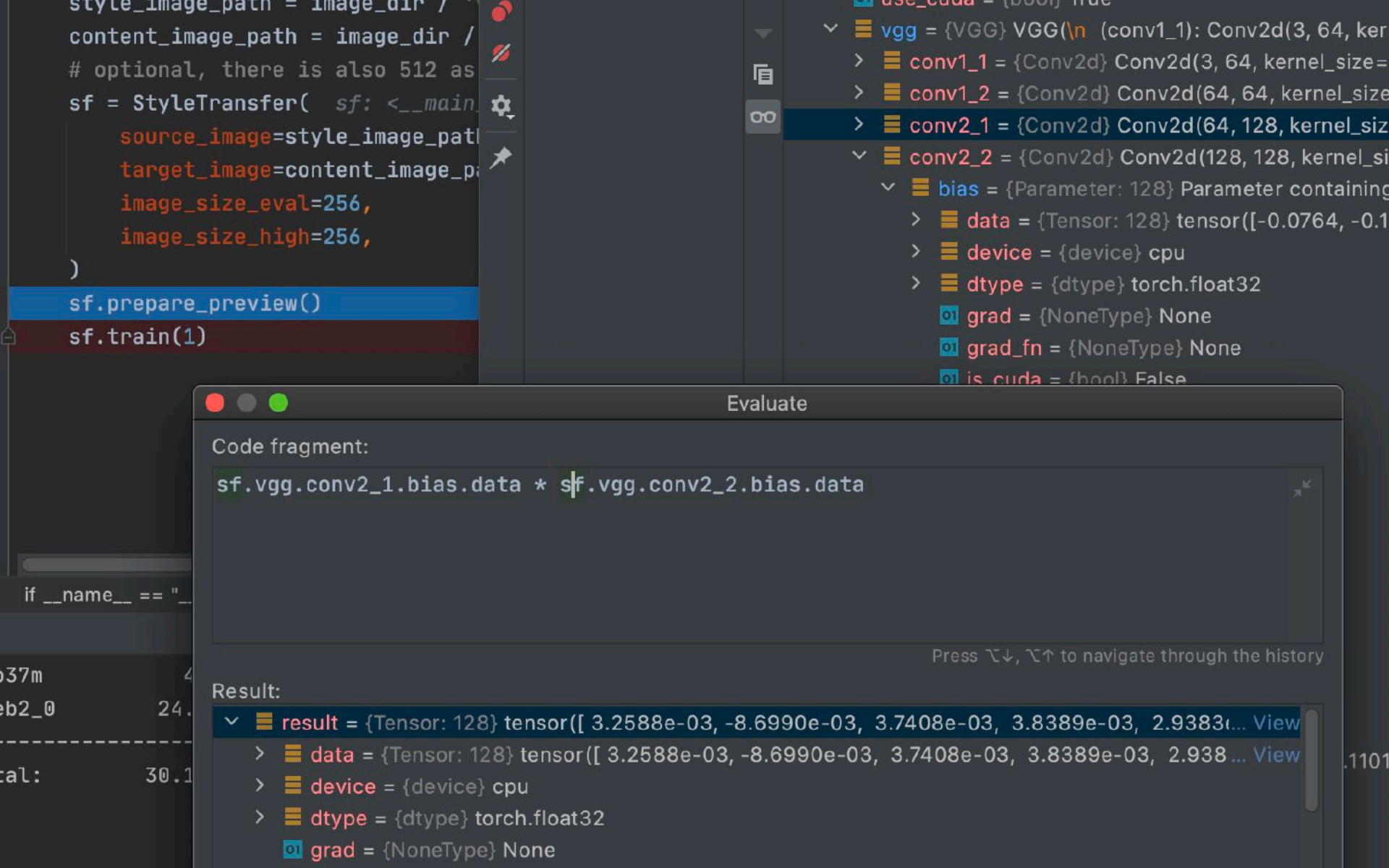
- Use Introspection
- IDE provide good debuggers

- Print statements are ok
- Logging is better


else f'config_{
yaml.safe_dump(the_di
if __name__ == "__main__":
style_image_path = image_dir
class Path(PurePath)
PurePath subclass that can make system call
source_image=style_image_
target_image=content_image
image_size_eval=256,
image_size_high=256,
)
sf.prepare_preview()
sf.train(1)


Debug
Debug: NeuralStyleTransfer
Frames | Variables | Console
Frames
Variables
Console
M...
<module>, Neura
torch_content_image = {NoneType} None
torch_style_image = {NoneType} None
use_cuda = {bool} True
vgg = {VGG} VGG(\n (conv1_1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))\n (conv1_2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding= View
conv1_1 = {Conv2d} Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
conv1_2 = {Conv2d} Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
conv2_1 = {Conv2d} Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
conv2_2 = {Conv2d} Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
bias = {Parameter: 128} Parameter containing:\ntensor([-0.0764, -0.1101, -0.1442, 0.0386, 0.2589, -0.0923, 0.0494, -0.0417,\n 0.0064, -0.1856, 0.1297, View
data = {Tensor: 128} tensor([-0.0764, -0.1101, -0.1442, 0.0386, 0.2589, -0.0923, 0.0494, -0.0417,\n 0.0064, -0.1856, 0.1297, -0.0177, -0.2785, -0.019 View
device = {device} cpu
dtype = {dtype} torch.float32
grad = {NoneType} None
grad_fn = {NoneType} None
is_cuda = {bool} False
is_leaf = {bool} True
is_mkldnn = {bool} False
is_quantized = {bool} False
is_sparse = {bool} False
layout = {layout} torch.strided
name = {NoneType} None
names = {tuple: 1} None
ndim = {int} 1
output_nr = {int} 0
requires_grad = {bool} False
shape = {Size: 1} 128
T = {Tensor: 128} tensor([-0.0764, -0.1101, -0.1442, 0.0386, 0.2589, -0.0923, 0.0494, -0.0417,\n 0.0064, -0.1856, 0.1297, -0.0177, -0.2785, -0.0194, View
Protected Attributes
dilation = {tuple: 2} (1, 1)
dump_patches = {bool} False
groups = {int} 1
in_channels = {int} 128
kernel_size = {tuple: 2} (3, 3)
out_channels = {int} 128
output_padding = {tuple: 2} (0, 0)
padding = {tuple: 2} (1, 1)
padding_mode = {str} 'zeros'


Debug:
NeuralStyleTransfer
Frames | Variables | Console
Frames
Variables
Console
<module>, Neura
targets = {list: 0} []
torch_content_image = {NoneType} None
torch_style_image = {NoneType} None
use_cuda = {bool} True
vgg = {VGG} VGG(\n (conv1_1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))\n (conv1_2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=... View
conv1_1 = {Conv2d} Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
conv1_2 = {Conv2d} Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
conv2_1 = {Conv2d} Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
conv2_2 = {Conv2d} Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
bias = {Parameter: 128} Parameter containing:\ntensor([-0.0764, -0.1101, -0.1442, 0.0386, 0.2589, -0.0923, 0.0494, -0.0417,\n 0.0064, -0.1856, 0.1297, ... View
Evaluate
Code fragment:
sf.vgg.conv2_1.bias.data * sf.vgg.conv2_2.bias.data
Press ⌥↓, ⌥↑ to navigate through the history
Result:
result = {Tensor: 128} tensor([3.2588e-03, -8.6990e-03, 3.7408e-03, 3.8389e-03, 2.9... View
data = {Tensor: 128} tensor([3.2588e-03, -8.6990e-03, 3.7408e-03, 3.8389e-03, 2... View
device = {device} cpu
dtype = {dtype} torch.float32
grad = {NoneType} None
grad_fn = {NoneType} None
is_cuda = {bool} False
is_leaf = {bool} True
Close
Evaluate
groups = {int} 1
in_channels = {int} 128
kernel_size = {tuple: 2} (3, 3)
out_channels = {int} 128
output_padding = {tuple: 2} (0, 0)
padding = {tuple: 2} (1, 1)
padding_mode = {str} 'zeros'


style_image_path = image_dir /
content_image_path = image_dir /
# optional, there is also 512 as
sf = StyleTransfer(sf: <__main
source_image=style_image_pat
target_image=content_image_p
image_size_eval=256,
image_size_high=256,
)
sf.prepare_preview()
sf.train(1)
vgg = {VGG} VGG(\n (conv1_1): Conv2d(3, 64, ker
conv1_1 = {Conv2d} Conv2d(3, 64, kernel_size=
conv1_2 = {Conv2d} Conv2d(64, 64, kernel_size
conv2_1 = {Conv2d} Conv2d(64, 128, kernel_siz
conv2_2 = {Conv2d} Conv2d(128, 128, kernel_si
bias = {Parameter: 128} Parameter containing
data = {Tensor: 128} tensor([-0.0764, -0.1
device = {device} cpu
dtype = {dtype} torch.float32
grad = {NoneType} None
grad_fn = {NoneType} None
Evaluate
Code fragment:
sf.vgg.conv2_1.bias.data * sf.vgg.conv2_2.bias.data
Press ⌥↓, ⌥↑ to navigate through the history
Result:
result = {Tensor: 128} tensor([3.2588e-03, -8.6990e-03, 3.7408e-03, 3.8389e-03, 2.9383e... View
data = {Tensor: 128} tensor([3.2588e-03, -8.6990e-03, 3.7408e-03, 3.8389e-03, 2.938 ... View
device = {device} cpu
dtype = {dtype} torch.float32
grad = {NoneType} None
if __name__ == "__
37m
eb2_0
24.
tal:
30.1

Jupyter and an IDE!

- Best of both
- Very productive
- Focus on the strong suits

How Can I Have Shorter, Concise Notebooks?

- Functions
- Classes
- Modules
- Magics!

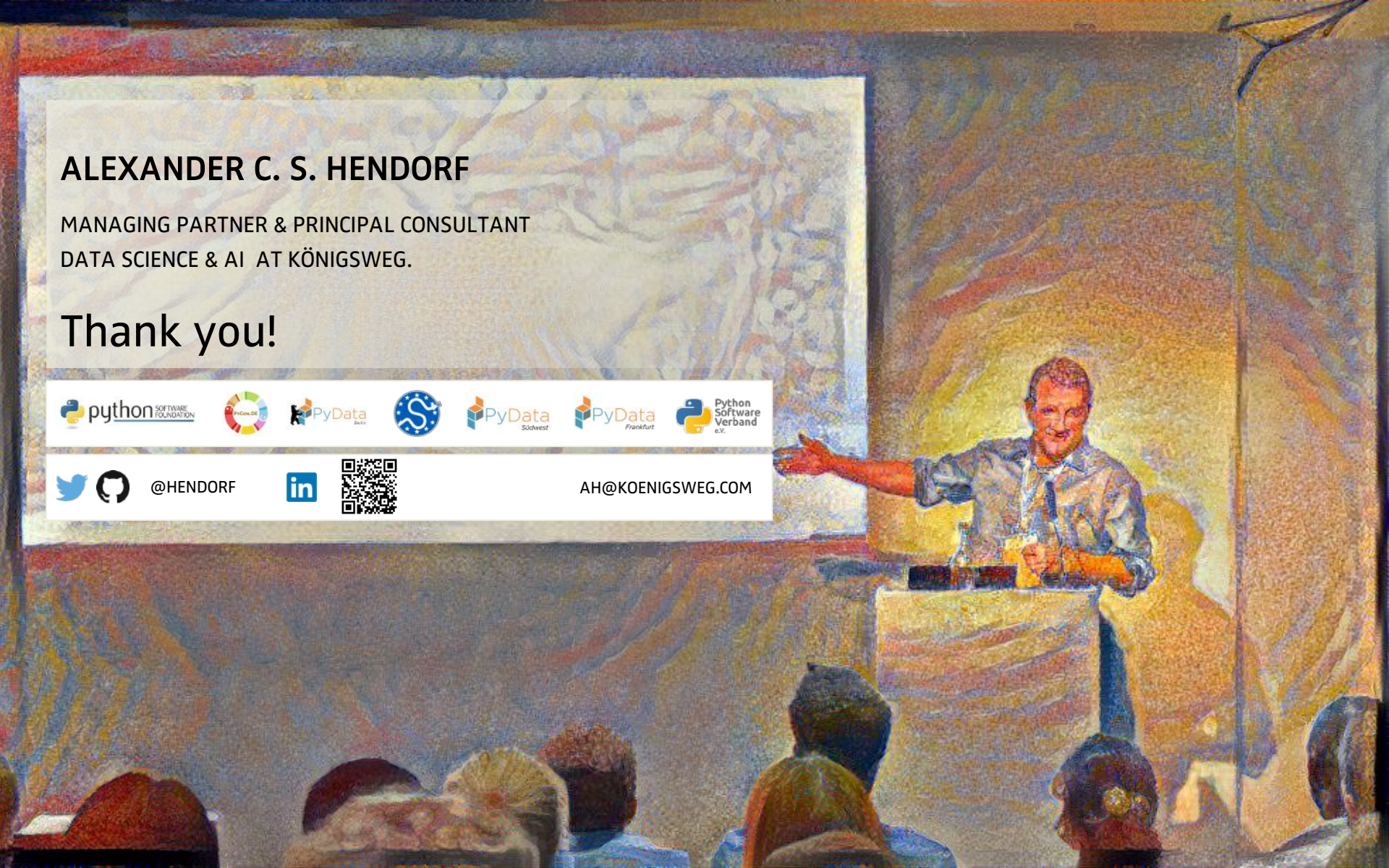
- Relevant code only notebooks

Anything Else?

- Type Hints
- Decomposition
- Parallelizing execution
- Parameters as config
- Sub-classing
- Multi modules
- Descriptors
- Decorators
- …

What Are the Main Take-Aways?

- Good coding is as good engineering an essential success factor
- Python offers several ways to achieve the required quality level for a successful implementation
- Beyond writing quality code, there are other aspects to consider such as personas, standardization and architecture
- Mentors will speed up the process a lot



ALEXANDER C. S. HENDORF

MANAGING PARTNER & PRINCIPAL CONSULTANT
DATA SCIENCE & AI AT KÖNIGSWEG.

Thank you!

@HENDORF

AH@KOENIGSWEG.COM