CAS Applied Data Science
Module 4: Ethics

Best Practices

Ahmad Alhineidi

University of Bern - DSL

Module 4: Ethics & Best Practices

2025-10-24, 13:15-17:00 Best Practices and Documentation (Ahmad)

2025-10-31, 13:15-17:00 Introduction to IT Security for Data Scientists (Matthias and Darja)

2025-11-07, 13:15-17:00 Introduction to Git and GitHub I (Ahmad)

2025-11-14, 08:15-12:00 Legal Aspects Session

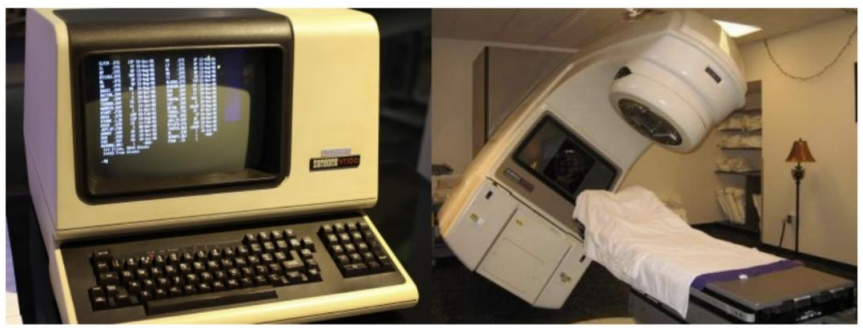
2025-11-14, 13:15-17:00 Introduction to Git and GitHub II (Ahmad)

Therac-25

UNIVERSITÄT BERN

1. From 1985 to 1987 a computer controlled radiation therapy machine massively overdosed about six people. Some died.

2. Software controlled interlock failed due to a race condition (high dose was possible without appropriate shielding)



https://de.wikipedia.org/wiki/Therac-25

Slide by PD. DR. Sigve Haug, UniBe

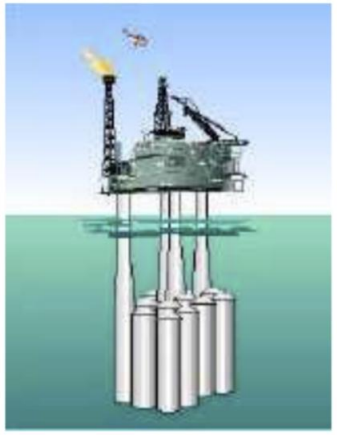
1991 Sinking of Norwegian Sleipner

UNIVERSITÄT BERN

1. 1991-08-23 The oil and gas platform Sleipner sinks

2. Economic loss about 700 MUSD

The post accident investigation traced the error to inaccurate finite element approximation of the linear elastic model of the tricell (using the popular finite element program NASTRAN). The shear stresses were underestimated by 47%, leading to insufficient design. In particular, certain concrete walls were not thick enough. More careful finite element analysis, made after the accident, predicted that failure would occur with this design at a depth of 62m, which matches well with the actual occurrence at 65m.



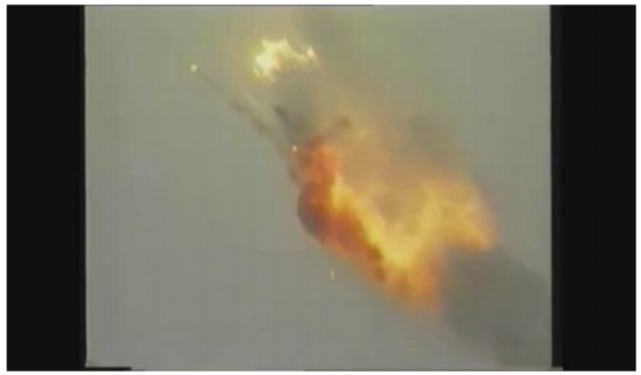
http://www-users.math.umn.edu/~arnold/disasters/sleipner.html

Slide by PD. DR. Sigve Haug, UniBe

1996 Ariane 5 Explosion

UNIVERSITÄT BERN

- June 4, 1996, unmanned rocket launched by ESA explodes
- Value about 500 MUSD
- Cause: Integer Overflow (conversion from 64 bit to 16 bit)



https://www.youtube.com/watch?v=kYUrqdUyEpI

Slide by PD. DR. Sigve Haug, UniBe

1983-09-23 World War III (almost)

UNIVERSITÄT BERN

- Soviet early warning satellite reports 5 US missiles coming towards Soviet
- S. Petrov reports it as a false alarm (luckily)

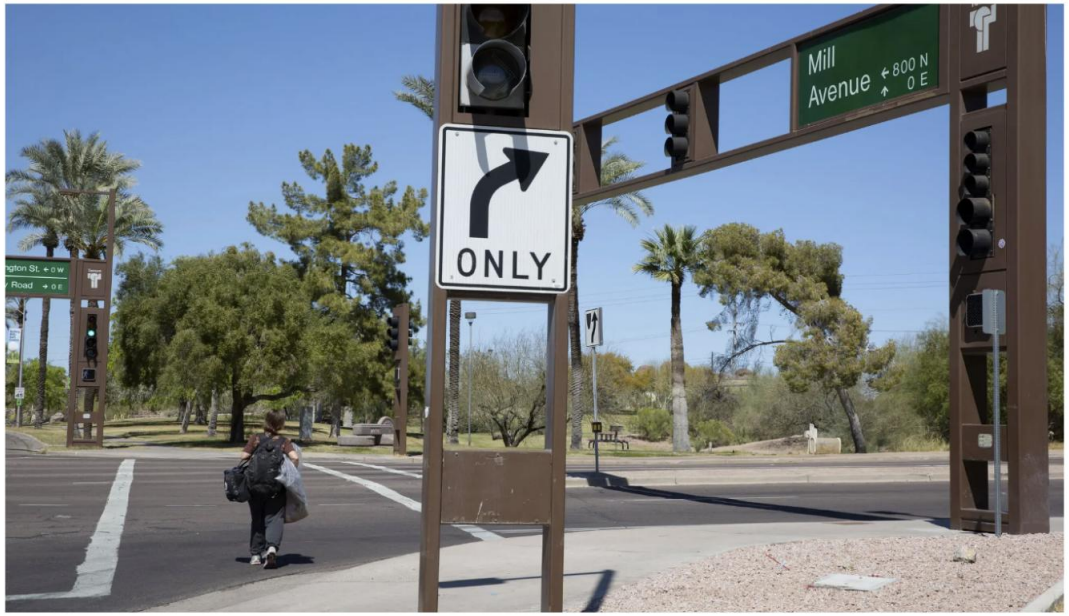
- Could have caused massive attack from Soviet
- Was a misinterpretation of reflecting sun light from cloud tops

Stanislav Petrov :"I had a funny feeling in my gut"

Slide by PD. DR. Sigve Haug, UniBe

AI kills woman March 2019

UNIVERSITÄT BERN



A woman crossing Mill Avenue at its intersection with Curry Road in Tempe, Ariz., on Monday. A pedestrian was struck and killed by a self-driving Uber vehicle at the intersection a night earlier. Caitlin O'Hara for The New York Times

Slide by PD. DR. Sigve Haug, UniBe

Racial Bias in AI

UNIVERSITÄT BERN

nature.com/articles/d41586-019-03228-6

nature > news > article

NEWS | 24 October 2019 | Update 26 October 2019

Millions of black people affected by racial bias in health-care algorithms

Study reveals rampant racism in decision-making software used by US hospitals – and highlights ways to correct it.

Slide by PD. DR. Sigve Haug, UniBe

Learning objectives

Write better/re-usable code

Better document your code

Deal with large data (big data)

Agenda

13:15 – 14:30: Intro & writing better code

14:30 – 15:00: Dealing with large data

15:00 – 15:30: Break

15:30 – 17:00: Documentation (self study & presentations)

Github repo

- Github repo for in-class tasks
- Link

What is ‘better code’?

Readable: Easy to understand for others (and your future self)

Maintainable: Simple to update, extend, or fix

Reproducible: Produces the same results across runs and environments

Testable: Can be verified for correctness through automated tests

Efficient: Uses time and resources wisely without over-complication

Collaborative: Written with team workflows, version control, and documentation in mind

Best Practices in Writing **Python** Code for Data Science

Python: frequently used in Data Science

Data Science packages: pandas, numpy, matplotlib, etc

Large eco-system and support

Easy syntax, beginner friendly

PEP 20 – The Zen of Python

Abstract

Long time Pythoneer Tim Peters succinctly channels the BDFL's guiding principles for Python's design into 20 aphorisms, only 19 of which have been written down.

The Zen of Python

```
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!
```

Source: link

PEP 8: **Style Guide for Python Code**

Written by Guido van Rossum and others (~2001)

Pep 8 & PEP 257 (Docstring Conventions) were adapted as standard

document that provides style conventions for writing Python code.

Why It Matters: Promotes readability, consistency, and collaboration. Makes it easier for different developers to work on the same codebase.

Goal: To make Python code more consistent and readable, so it "looks like it was written by the same person."

Follow pep 8 guidelines

- Use white spaces for better readability
- Use 4 spaces indentation
- Naming convention
- **Task**: Find out what are the naming convention in pep 8 for class, functions and variable names

```
# Bad
def myFunc(x,y): return x+y

# Good (PEP 8 compliant)
def add_numbers(x, y):
    return x + y
```

Write Clear and Descriptive Variable/Function Names

```
# Bad
a = [1,2,3,4]
def f(x): return sum(x)/len(x)

# Good
scores = [1, 2, 3, 4]
def calculate_mean(values):
    return sum(values) / len(values)
```

Use Docstrings for Documentation

- **Reference**: PEP 257 – Docstring Conventions.
- **Why**: Helps others (and yourself) understand code months later.
- Useful in MCP systems and automatic documentation

```
def calculate_mean(values):
    """
    Calculate the mean of a list of numeric values.

    Parameters
    ----------
    values : list of float
        A list of numbers.

    Returns
    -------
    float
        The arithmetic mean.
    """
    return sum(values) / len(values)
```

Use Modular Code (Functions & Classes)

- **Why**: Modular code avoids duplication and promotes reusability.
- Do not rewrite existing functions and classes

```
# Bad
data = [1,2,3,4,5]
mean = sum(data) / len(data)
std = (sum((x-mean)**2 for x in data) / len(data))**0.5

# Good
import statistics
mean = statistics.mean(data)
std = statistics.stdev(data)
```

Write Tests for Your Code

- Tests catches bugs early on
- Allows you to think of all kind of inputs
- Test-driven development is a thing
- Pytest or unittest in python

Handle exception on top of tests

- Why: avoid code breaking in production
- Control the output when an error occur
- In python [try, except, error name]
- Avoid empty except

```
try:
    value = int("abc")
except ValueError as e:
    print(f"Error: {e}")
```

Use Version control system (git)

- Allows you to save progress in your project
- Easy to collaborate with others
- Have better idea when things go wrong, when, and why
- 2 sessions on git & github this module

Avoid Hardcoding and Use Configuration

- More flexibility in editing your script
- Avoid issues with different os [path]

```
# Bad
file_path = "/Users/ahmad/data/input.csv"

# Good
import os
file_path = os.getenv("DATA_FILE", "default.csv")
```

Ensure Reproducibility (Random Seeds & Environment Tracking)

- Very important in science / data science
- **Why**: Results should be reproducible across runs and machines.
- Example: requirements.txt, seed

```
import numpy as np
np.random.seed(42)
print(np.random.rand(3))
```

Task

- Work in groups (2-3)
- Go through pep 8 (https://peps.python.org/pep-0008/)
- Discuss 1 or 2 useful practice(s)/style(s) and why is it useful?
- Post your findings as slides (link)
- Discussion & preparation 15 min
- Presentation (2 min each group)

Dealing with large data

- Big data: could indicate large or complex data
- Usual data processing pipeline might fail
- Structured [json, sql, csv], semi-structured [xml], raw data [txt, pdfs]
- Large data & limited resources require special technique
- Python [generators, pandas, multiprocessing]

Dealing with large data (generators)

- Special type of iterator
- Lazy (one item at a time)
- Doesn’t load all to memory
- Very useful in NLP or ML (Create generators pipelines)
- Can use it to store infinite sets
- Create with a function
- Create with generator comprehension

Dealing with large data (generators)

```
def count_up_to(n):
    count = 1
    while count <= n:
        yield count
        count += 1

counter = count_up_to(1000000000)
for number in counter:
    print(number)
```

Dealing with large data (generators)

Generator comprehension

```
generator_example = (number for number in range(100000000))
generator_example = (token.lower() for token in line if token not in stopwords)
```

Task

- Create large text file with the script “large_data/create_large_file.py”
- Will create txt file with the size of ~1.2 GB
- Output: how many times the word “the/The” occurs in the file

Code documentation

Why Documentation matters:

- Makes code understandable for *others* (and your future self)
- Bridges the gap between code and human understanding
- Essential for collaboration and long-term maintenance

Code documentation

Goals of documentation:

- Explain *what* the code does
- Describe *how* to use it (functions, classes, modules)
- Provide context: *why* certain decisions were made

Code documentation

Levels of documentation

- variables, function, and classes names
- Inline comments
- Doc strings
- Readme
- Website with full docs, examples & tutorials
- Video tutorials, seminars

Code documentation

Levels of documentation: variables, function, and classes names

- The first layer of documentation is choosing meaningful names
- Good name replaces the need for comments

```python
# Bad
def f(x):
    return x * x

# Good
def square(number: int) -> int:
    """Return the square of a number."""
    return number * number
```

Code documentation

Levels of documentation: Inline Comments

- Comments should explain *why* something is done
- not *what* the code already says
- Keep them short

```
# Bad
i = i + 1  # increment i by 1


# Good
# Prevent infinite loop by ensuring counter progresses
i = i + 1


# Bad
# sort list
data.sort()


# Good
# Sort data alphabetically for consistent display
data.sort()
```

Code documentation

Levels of documentation: Doc string

- Docstrings describe what a function/class/module does, its inputs, outputs, and edge cases. Use triple quotes """

```python
# Bad
def calc(x, y):
    return x / y

# Good
def divide(numerator: float, denominator: float) -> float:
    """
    Divide one number by another.

    Parameters
    ----------
    numerator : float
        The number to be divided.
    denominator : float
        The number to divide by (must not be zero).

    Returns
    -------
    float
        The result of the division.

    Raises
    ------
    ZeroDivisionError
        If denominator is zero.
    """
    return numerator / denominator
```

Code documentation

Levels of documentation: Readme

- Every project should have read me
- Entry point for users
- Example: read me pandas project

Code documentation

Levels of documentation:
wiki/website/tutorials

- Further introduce the software
- Give real use cases with examples
- Forums and github discussion to further improve
- **scikit-learn** example: link

Code documentation

Documentation examples

- Automatic documentation generation with pdoc
- Generate docs into html based on the comments, doc strings, variale types
- Forces you into writing good comments / doc strings

Workshop & presentations

- Split into groups of 2-4
- A section will be assigned to you from the paper: link
- Read the section (10 min), discuss (10 min), prepare 1-2 slides to present (10 min)
- Appen your slides here: link