UNIVERSITÄT BERN

CAS Applied Data Science

Module 4

Best Practices for Data Science and Scientific Computing

2023-10-20

Group 1

UNIVERSITÄT BERN

1. Importance of Readable Code

- Code should be **understandable by others and your future self**.
- **Why?** Readable code is easier to maintain, debug, and confirm for correctness.

2. Guidelines for Writing Human-Friendly Code

- **Limit Memory Load**:
  - Break code into **small, focused functions** for better readability.
  - Reduces cognitive load by limiting details to remember at once.
- **Use Meaningful Names**:
  - **Clear, distinctive names** for variables and functions prevent confusion.
  - Avoid non-descriptive or similar names (e.g., `a` vs. `foo`, or `result1` vs. `result2`).
- **Consistent Style**:
  - Uniform formatting and naming conventions improve readability.
  - Reduces errors and makes the code easier to follow.

Conclusion

- Writing for people, not just computers, **builds reliable, maintainable software**—vital for scientific accuracy and collaboration.

Group 2

Let the Computer Do the Work

UNIVERSITÄT BERN

Repetitive Computational Tasks in Science

- Many scientific tasks require repeating similar computational steps (e.g., processing large data files and or regenerating figures)

Automating Repetitive Tasks

- Scientists are encouraged to use computers for task repetition: (reducing time wastage and errors.).
    - **Task Automation and Command Re-use :** Use tools (e.g., “history” feature in command-line interfaces) to save and reuse recent c

Scripts and Workflow Management

- A file containing a series of commands is known as a **script**. it can be managed with **workflow tools** (e.g., Make) for consistency and efficie
- Workflow tools help define dependencies between files and automatically update outputs if inputs change.

Using Build Tools for Efficiency

- Build tools can automate workflows by defining data dependencies and program relationships, enabling efficient regeneration of needed out

Ensuring Reproducibility

- **Provenance of Data:** Record all essential details to recreate outputs accurately.
- Essential details to capture include:
    - Unique identifiers and version numbers for raw data. and Versioning for programs and libraries.
    - Parameters used to generate each output.and Program names and version numbers used in the process.

Standardization Efforts

- Some initiatives aim to automate and standardize the recording of these reproducibility details for consistency across research.

Group 4

Don't Repeat Yourself (or Others) - DRY principles

- Every piece of data must have a single authoritative representation in the system
  - Constants are defined once throughout a project
  - Each defined constant is a unique constant

- Modularize code rather than copying and pasting
  - Create ways to repeat a step
  - Bug fixing counts for parts were that module is present
  - Easier to re-use for other projects

- Re-use code instead of rewriting it
  - Saves time and frustration
  - Action is always done in the same way

Group 5 - Plan for Mistakes

- Check for Errors within the Code (Error Messages)
- Automated Testing
  - Unit Tests, Integration Tests, Regression Tests
  - One line, whole Block of Code, different input data
- Use previous bugs for future testing
- Use a debugger

An *assertion* is a statement that checks whether a certain condition is true during the program's execution. If the condition is false, the assertion causes the program to produce an error, often stopping execution and displaying an error message. This helps programmers quickly identify and address issues in the code.

Group 6 - only Optimize code that works

UNIVERSITÄT BERN

- Use profiler to identify bottlenecks inside the code
- Use high level language (easy readability, less complex)
- Check result of unoptimized against optimized to see the difference

In this context, a *profiler* is a tool used to analyze a program's runtime behavior, specifically to identify parts of the code that consume the most time or resources. By focusing on functions or code segments with the highest resource usage (the "bottlenecks"), developers can target these areas for optimization, making the process of improving performance more efficient. Profilers generally provide detailed reports on which functions or lines of code are executed most frequently or take the longest to run, enabling developers to pinpoint where optimization would have the greatest impact.

Group 7 - Documentation

Good documentation helps people to understand code

BP1: Document design and purpose, not mechanics

- Focus on the why and what, rather than how it is implemented
- Helps to understand concept and goals of a code

BP2: Refactor your code, instead of explaining how it works

- Refactoring: Simplifying, organizing, renaming
- Improve clarity and readability of the code itself than relying on comments

BP3: Embed the documentation where it is used

- E.g., documentation of a software → embed the documentation in that software
- When others make changes, they can directly adapt the documentation
- Valuable to explain the functionality, typical use cases, and benefits of each feature

Group 8 - Collaborate

Why and when to collaborate?

To eliminate bugs, improve readability, increase efficiency and transparency.

ALWAYS: do the pre-merge code reviews

- for smaller teams: use pair programming when bringing someone new up to speed

  and when tackling particularly tricky problems. 1 Driver / 1 Navigator.

- for bigger teams and more complex projects use an issue tracking tool (e.g. JIRA), use VCS, consider applying SCRUM or other software development methodology.

Group 8 - BP3 Make Incremental Changes

UNIVERSITÄT BERN

- adopt to evolving requirements
- work incrementally with small frequent code steps to allow frequent feedback (agile development)
- use version control systems (VCS) to track and match changes in a group of developers
- ensure reproducibility by storing code, observations and meta-data by using VCS

b
UNIVERSITÄT
BERN