Software Engineering, University of Oulu

**Lab 3: Linting**

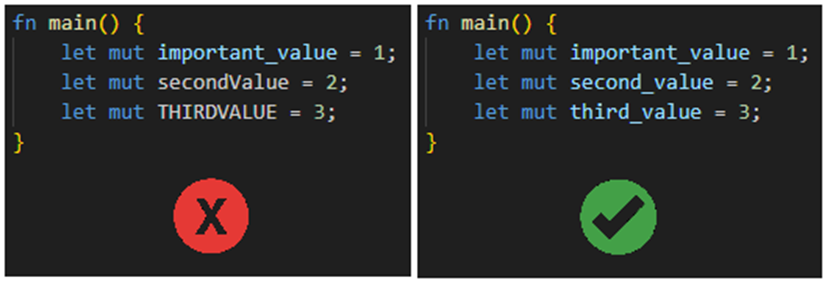
*Disclosure: The material in this paper has not been reviewed, endorsed, or approved of by the Rust Foundation. For more information on the Rust Foundation Trademark Policy, click* [*here*](https://docs.google.com/document/d/1ErZlwz9bbSI43dNo-rgQdkovm2h5ycuW220mWSOAuok/edit) *.*

1. **Introduction**

Imagine that a large team of developers are working on the same code base. However, with varying coding styles and preferences, potential issues may arise, leading to confusion and inefficiencies. Every developer has their own style and preferences when it comes to creating code, but everyone in the team should be able to seamlessly understand every section regardless of who originally wrote it.

This is why organizations have their own [coding conventions](https://en.wikipedia.org/wiki/Coding_conventions). The way of enforcing them is knows as [linting](https://en.wikipedia.org/wiki/Lint_(software)). During the linting process, the source code is analyzed to identify any style errors that deviate from the established guidelines. Linting plays a vital role in maintaining code consistency and readability.

By integrating linting into the continuous integration (CI) pipeline, every code change goes under automatic evaluation against the coding conventions, ensuring that every commit maintains the consistency of the code base. This early integration is crucial as it saves significant time and effort that would otherwise be spent refactoring code later in the development process.



For example, consistent variable naming improves readability.

1. **Linting in Rust**

Rustc comes with a built-in linter that includes a collection of common lint checks.

The basic lints can be run simply by typing:

cargo check

To see the list of all the inbuilt lints, type:

rustc -W help

Different lints have different levels of severity:

|  |  |
| --- | --- |
| allow |  |
| warn | throws a warning |
| force-warn | throws a warning that cannot be overwritten |
| deny | throws an error |
| forbid | throws an error that cannot be |

1. **Exercise**
2. Create a GitHub classroom environment: <link here>.
3. Lint the project and examine the output.
4. Fix all linting issues in main.rs until the linter gives no errors or warnings.
5. Complete the CI/CD-pipeline to automatically check for linting issues.
6. Commit your changes and push themto the remote repository.
7. Return a link to your repository in Moodle.