HW1: Technology Template Repository

Overview:

Create an advanced template repository for a specific technology (like a programming language and environment). This template serves as a foundation for designing and developing projects, fully equipped for immediate use. It includes features for build management, unit testing, continuous integration, static analysis, code style adherence, and component specification.

Objectives:

Create a git repository which meets the following specifications (Pick either C/C++ OR Python):

Programming Language	**C/C++**	**Python**
Compiler/Interpreter	Clang	CPython
Testing Framework	Google Test	Pytest
Dependency Manager	vcpkg	uv
Code Formatting	ClangFormat	ruff
Static Analysis Tool	ClangTidy	mypy and ruff
Code Coverage	gcov	coverage.py

Use CircleCI as the CI/CD solution.

Deliverables:

- A template Github repository meeting the above criteria.
- The repository should include a pre-configured, restrictive setup for static analysis and code formatting.
    - All available checks in `ruff` and `mypy` need to be enabled and then any that needed to be disabled should be explained “why?”

- **Ensure the template includes the concept of a component, clearly defined and documented, in a file called component.md**
- Set up templates for issues and pull requests to standardize submissions.
    - `pull_request_template.md` in the root of the repo.
    - A bug report and feature request issue template (use the template builder)
- A set of 3 components with accompanying unit tests, integration tests and end to end tests.
    - **Components**
        - Calculator - performs basic arithmetic operations
        - Logger - records operations performed by calculator
        - Notifier - sends an alert when the result exceeds a given threshold.
    - **Unit Tests**
        - Calculator - Test addition, subtraction, and multiplication.
        - Logger - Test that operations are logged correctly.
        - Notifier - Test that notifications are sent when threshold is exceeded
    - **Integration Tests**
        - Calculator ↔ Logger (Mock Notifier)
        - Logger ↔ Notifier (Mock Calculator)
    - **End-to-End Test**
        - Perform a calculation, log it, and send a notification when the threshold is exceeded.

- A CI pipeline executing the test. Tests should be visible within the “Tests” section of the CircleCI job dashboard. (You might need to use the predefined step store-test-results for this).
    - Make sure you include links from CircleCI runs that have both test passes and test failures in your HW submission.
- A CI pipeline generating test coverage report (perhaps as part of the test one)... or separate. Report should be accessible / browsable from the UI.
    - Include a link to the report in your HW submission.
- A comprehensive README.md explaining repository usage.
- Proper .gitignore file tailored to the language/environment.
- An appropriate open-source license.
- **(OPTIONAL)** Configure a pre-commit hook that runs code formatting, linting, and unit tests checks, ensuring only correct code is committed to the GitHub repository.
-

Example repositories:
Go
Java
C++

Python

Measures of Success:

The repository functions “out of the box” for its intended technology stack.
All configurations and templates are correctly set up and documented.
The repository demonstrates best practices in software development, including project organization and workflow management.

Extra Credit

1. Implement nose2 in the Python repository as an exploratory alternative to pytest.
2. Extend the repository to demonstrate advanced CI/CD workflows beyond basic testing. These may include things like parallel test execution, integration with other tools, etc.
3. If you are in a mixed Python/C++ team, put in a demo for cross-language modules using C++ from Python using `nanobind`.

FAQ

1. How to setup CI/CD?
    a. Head over to https://circleci.com/ and read their getting started guide. Play with the examples and come back with questions about what is not clear.
2. Does everybody need their own branch?
    a. Each team will submit their own repository with a single `main` branch
3. What if this seems too easy and I want to do more?
    a. If you have already done the Python -> C++ bridge extra credit…
    b. You can try to do C++ modules!
    Standard C++ Modules — Clang 21.0.0git documentation
    Note that his works fairly differently in gcc and msvc so there is a lot of work here…
    c. You can try to build a C++ -> Python bridge (this is where you can host an interpreter in your C++ application and have some C++ context exposed to it)
4. Sanity check: For cpp/python teams, we have to add all specs (tests, dependency manager, linter, etc…) just like we did for python, for our nanobind to be considered correct. It’s not enough to just build a c++ module and call it a day without testing it / running it through a build process.