Congratulations! You passed!

Grade received 100% **Latest Submission Grade** 100% **To pass** 80% or higher

Quiz

Go to next item

Rust Components and Patterns

Review Learning Objectives

1. What is the purpose of using doctest in Rust? **1 / 1 point**

- To generate automatic documentation for Rust code
- Verifying the correctness of code by writing comprehensive and accurate test cases
- To automate the deployment process of Rust applications.
- To validate and enforce coding style and formatting in Rust.

Correct
Correct! The purpose of using doctest in Rust is to verify the correctness of code by writing comprehensive and accurate test cases.

Submit your assignment

Due Jan 21, 11:59 PM PST

Try again

Receive grade

To Pass 80% or higher

Your grade

100%

View Feedback

We keep your highest score

2. What is the benefit of organizing Rust code using public and private modules? **1 / 1 point**

- To reduce the size of a release binary
- Ensuring proper encapsulation and structuring for efficient project management
- Improving the organization of code with modules and tests
- To prevent modification of public fields

Correct
Correct! Organizing Rust code using public and private modules ensures proper encapsulation and structuring for efficient project management.

Like Dislike Report an issue

3. How do you distinguish between public and private fields in Rust structs? **1 / 1 point**

- By using the visibility modifiers (pub and private) for fields
- By organizing and structuring modules with private headers
- By using test failure messages that demonstrate private vs. public fields
- By utilizing doctest to verify code visibility

Correct
Correct! In Rust, you distinguish between public and private fields in structs by using the visibility modifiers (pub and private) for fields.

4. Why are integration tests in Rust external to the library being tested? **1 / 1 point**

- To measure test coverage of a Rust program
- Because it is only possible to test public code outside of the module
- To ensure testing of the library's functionality as it would be used by any other code, using only the public API
- Because integration tests require building a release binary external to the library being tested

Correct
Correct! Integration tests in Rust are external to the library being tested to ensure testing of the library's functionality as it would be used by any other code, using only the public API.

5. Why are the assert_eq! and assert_ne! macros commonly used in Rust tests? **1 / 1 point**

- To expose values that are not exposed in certain fields
- To conveniently compare two values for equality or inequality and print the values if the assertion fails
- To expose private functions and other code that is not exposed from the same module
- To increase the visibility on all String and str types in failure

Correct
Correct! The assert_eq! and assert_ne! macros in Rust tests are commonly used to conveniently compare two values for equality or inequality and print the values if the assertion fails.

6. Why can private functions be tested directly in Rust? **1 / 1 point**

- It isn't possible to expose private functions in Rust for testing while keeping them private in the runtime
- It isn't possible to test private functions in Rust with Cargo
- Because Rust's private rules allow testing of private function in separate modules
- Because Rust's privacy rules allow testing of private functions and their visibility within the same module

Correct
Correct! Private functions can be tested directly in Rust because Rust's privacy rules allow testing of private functions and their visibility within the same module.

7. What is one of the main purposes of using a Makefile in Rust? **1 / 1 point**

- To control the dependency workflow
- To support and integrate with CI/CD systems like Jenkins or GitHub Actions
- To support advanced features flags of Cargo
- Streamlining and automating the build process of the Rust project

Correct
Correct! The purpose of using a Makefile in Rust is to streamline and automate the build process of the Rust project.

8. What is the benefit of extending functionality with modules in Rust? **1 / 1 point**

- Verifying the correctness of code by writing comprehensive and accurate test cases
- Enhancing code organization and promoting code reusability and modularity
- To ensure modules are included when building a release binary
- Modularized code is easier to test

Correct
Correct! The benefit of extending functionality with modules in Rust is to enhance code organization and promote code reusability and modularity.

9. What is the purpose of using Cargo for managing dependencies in Rust? **1 / 1 point**

- To control allow and deny lists in dependencies
- To easily manage feature flags from dependencies
- Simplifying and automating the process of handling external dependencies in Rust projects
- To download multiple dependencies locally

Correct
Correct! The purpose of using Cargo for managing dependencies in Rust is to simplify and automate the process of handling external dependencies in Rust projects.

10. Why is the #[cfg(test)] annotation used in Rust test modules? **1 / 1 point**

- To include tests when building a Rust binary
- To enable running tests in the CLI when building
- To compile and run test code only when running cargo test, excluding it from cargo build
- To allow loading test modules

Correct
Correct! The #[cfg(test)] annotation in Rust test modules is used to compile and run test code only when running cargo test, excluding it from cargo build.