Closures

Learn to Code with Rust / Section Review

Intro to Closures

- A **closure** is an anonymous function. It's a sequence of steps that can accept inputs and return an output.
- The **functional programming** paradigm treats a function like any other value.
- We can pass a closure to a function, return it out of a function, assign it to a variable, store it in a collection type, etc.

Closure Syntax

- Declare a closure with a pair of vertical pipes and curly braces. Omit the braces for a one-line expression.
- Define parameters inside the pipes.
- Use arrow syntax to designate a return value.
- The compiler can infer the types of closure parameters and return values based on the closure's logic and its invocation.

The Fn Traits

- The **Fn** traits represent an invocable procedure (function or closure) that can be called with parentheses.
- There are three traits in the hierarchy: **FnOnce**, **FnMut**, and **Fn**. **Fn** is the most restrictive one.
- When evaluating a closure argument, the compiler will validate the closure's trait, its parameter types and its return type.

Capturing Values I

- Closures can capture values. To capture means "to save a piece of data" for later use.
- An **FnOnce** closure captures values by ownership. The closure can only be invoked once.
- An **FnMut** closure captures values by mutable reference. The closure can be invoked multiple times.

Capturing Values II

- An **Fn** closure captures values by immutable reference or captures no values at all. The closure can be invoked multiple times.
- The **move** keyword forces a closure to take ownership of its values even when it normally would borrow a reference.
- Even with the **move** keyword, some closures can still be invoked multiple times. One example is if the closure doesn't return the moved value.

Closure Parameters I

- To define a parameter that accepts a closure, declare a generic and apply it to the parameter.
- Add a trait bound to mandate that the generic implements one of the three **Fn** traits.
- The most common syntax is adding the **where** keyword after the parameter list.

Closure Parameters II

- Alternatively, we can force the trait constraint on the generic in its declaration inside in the angle brackets.
- We can also apply the constraint directly to a parameter with the **impl** keyword.
- Plain functions can be passed in for "closure" parameters. Functions are also invocable procedures, but closures are more common.

Closures in Rust

- The **unwrap_or_else** method on an **Option** unwraps the **Some** variant's data or invokes the closure argument in case of **None**.
- The **retain** method on a **String** accepts a closure with the Boolean logic that determines if a character should be kept.