Generics

Learn to Code with Rust / Section Review

Generics

- A **generic** is a type argument.
- A **generic** is a placeholder type for a future concrete type (or multiple types).
- A **generic** is to a type what a parameter is to an argument. It's a stand-in for a future value.
- Generics enable reusability by not coupling a function/struct/enum/etc to a hardcoded type.

Generic Syntax

- Declare a generic with a pair of angle brackets and a name. **T** is a common choice **(<T>)**.
- Separate multiple generics with a comma and a space.
- Multiple generics enable multiple future types. They do not *mandate* that the types must be different; they *enable* the types to be different.

Usecases

- Use the type **T** where you want **T** to play the role of the future type.
- In a function definition, **T** can represent the type of the parameters and/or return value.
- In a struct definition, **T** can represent the type of a field.
- In an enum definition, **T** can represent the type of the associated data belonging to a variant.

impl Blocks and Generics

- The **generic** is coupled to the definition of a type.
- The **impl** keyword requires that we provide a concrete type or a generic type for the main type.
- We can hardcode a specific type to declare methods for only *that* specific type.
- We can use **impl<T> SomeType<T>** syntax to declare methods for any type **T**.