Traits

Learn to Code with Rust

Contracts in the Real World

- A contract is a document that people sign that states their obligations.
- Imagine a non-specific contract with the following requirement: "You promise to arrive at 9am at a location"
  - A college student can promise to arrive at class at 9am
  - A software engineer can promise to arrive at work at 9am
  - A flight can promise to arrive at an airport at 9am
  - A package can promise to arrive at a house at 9am
- The *situations* are different but they honor the same promise.

Traits

- A **trait** is a contract that requires that a type support one or more methods.
- Traits establish consistency between types; methods that represent the same *behavior* have the same name.
- When a type opts in to honoring a trait's requirements, we say the type **implements** the trait.
- Types can vary in their implementation but still implement the same trait.

Traits II

- A type can choose to opting in to implementing a trait.
- A type can implement multiple traits. There are hundreds of traits available in Rust.
- A trait is called an interface or protocol in other programming languages.

The Display Trait I

- The **Display** trait requires that a type can be represented as a user-friendly, readable string.
- The **Display** trait mandates a **format** method that returns the string.
- When we use the **{ }** interpolation syntax, Rust relies on the **format** method.
- Integers, floats, and booleans all implement the Display trait so we are able to interpolate them with curly braces.

The Display Trait II

- It is not always clear how a complex type should be represented as a piece of text.
- Not all types implement the Display trait. One example is the array type.