Consuming Rust bite by byte

Bite 0 - Introduction

Jim Fawcett

https://JimFawcett.github.io

Data

Why Rust?

- Memory and Data Race safety
  - Enforced data ownership rules insure Memory and Data Race safety.
- Error Handling
  - Any function that can fail returns a result indicating success or failure. Code has to handle errors in well defined ways.
- Performance
  - Rust compiles to native code and does not need garbage collection, so it is as fast as C and C++.
- Simple Value Behavior
  - Rust supports value behavior without the need to define copy and move constructors and assignment operators.
- Extraordinarily effective tool chain

What is this?

- This is the first in a series of bites - brief presentations - about the Rust programming language:
  - Each presentation will be brief – a few slides
  - Each will focus on one part of the Rust language
  - The series will build in bite sized chunks: easy to grasp, quick to consume.

Exercises

1. Each Bite ends with a few exercises.
2. Each one is simple, taking only a few minutes to complete.
   - Use Visual Studio Code
   - Create a package using Cargo:
     - Cargo new bite_5_exercises.
   - If you build each exercise as a function, then you can call any of them from the main function.

References

Link	Description
ConsumingRustBite2 - UDB	Undefined behavior – example from C++ code
Rust Story - Data	Expanded discussion in Rust Story

That’s all until Bite #1

Bite #1 introduces Bind, Copy, Move, and Clone.