Strings

Learn to Code with Rust / Section Review

Two Types of Strings

- A **string** is a sequence of text characters. Rust has 2 string types: **str** and **String**.
- The **str** type usually appears in its borrowed form: **&str**. We call this type a "string slice" or a "ref str".
- Rust embeds string literals in the executable. When the program runs, it receives a string slice (reference) to the memory that holds that text.
- The **String** type lives on the heap. It can grow and shrink in size.

Deref Coercion

- Rust can transform a **&String** type into a **&str** type.
- A **String** reference can always be represented as a string slice, a borrow of some portion of text.
- The operation does not work in reverse.
- A **&str** cannot always be represented as a **&String**. The original source of data may not be a heap **String**.

Concatenation

- The **push_str** method concatenates content to the end of a **String**.
- The **push** method appends a character to the end of a **String**.
- The **+** symbol calls the **add** method. It takes ownership from the first String and returns the mutated String.

Index Positions

- Rust prohibits accessing an individual byte position within a string, even if we use the borrow operator.
- There is the possibility we access a byte that is part of a larger byte sequence.
- Rust permits the range syntax to extract a byte sequence. Combine it with the borrow operator.

The **format!** Macro

- The **format!** macro is similar to **println!** but returns the **String** instead of printing it.
- The familiar syntax options (**{}**, **{0}**) all apply.

Common Methods

- The **trim** method removes whitespace from the beginning and end of a string. It returns a string slice.
- The **to_uppercase** and **to_lowercase** method return **Strings** with uppercase and lowercase characters.
- The **replace** method swaps all occurrences of one character sequence with another.
- The **split** method cuts a string at all occurrences of a delimiter. Call the **collect** method to gather the results in a vector.

Collecting User Input

- The **io::stdin** struct includes a **read_line** method to collect user input.
- The **read_line** method accepts a mutable reference to a String.
- The **read_line** method returns a **Result** enum. The operation has the potential to fail.
- The **Ok** variant stores the number of collected bytes. The original String will hold the user's entry.