Data Types

Learn to Code with Rust / Section Review

Integers

- An integer is a **whole number**.
- A signed integer (**i** family) supports negative and positive values.
- An unsigned integer (**u** family) supports only zero and positive values. They can extend twice as far in the positive direction.
- The number after **i** or **u** is the number of bits that the value occupies.

Floats

- A floating-point (float) is a number with a decimal component.
- Rust supports two float types: **f32** and **f64**. A **f64** supports 15-17 digits of precision.
  A **f32** supports 6-9 digits of precision.
- Use the **:.n** format specifier to print a float with a custom precision.

The usize and isize Types

- **usize** and **isize** are aliases for an existing type.
- On a 32-bit system, **usize** will be a **u32** and **isize** will be an **i32**.
- On a 64-bit system, **usize** will be a **u64** and **isize** will be an **i64**.
- The advantage of **usize** and **isize** is the versatility in being able to run across different systems.

Strings and Characters

- A **string** is a piece of text.
- A **character** (**char** type) represents a single Unicode character.
- A **string literal** is a hardcoded string in our source code. We declare it with double quotes.
- The type of a string is **&str**. We'll talk more about this type later in the course.
- Special characters render different content in the string.
  - \n adds a new line.
  - \t adds a tab.
  - \" escapes a double quote.

Methods

- A **method** is a function attached to a value. A method is an action or behavior that the value can perform.
- Add a dot after the value, then the method name and a pair of parentheses.
- Methods may accept arguments. An argument is an input.
  - Separate multiple arguments with a space and a comma.

Math Operations

- The **+** operator performs addition.
- The **-** operator performs subtraction.
- The ***** operator performs multiplication.
- The **/** symbol performs floor division. Diving an integer by an integer produces an integer. Use floats for decimal division.
- Augmented assignment operators like **+=** perform an operation on a variable's value and assign the result back to the variable.

Booleans

- A **Boolean** is a type whose only two values are **true** and **false**.
- The **equality operator** (==) returns **true** if its operands are equal.
- The **inequality operator** (!=) returns **true** if its operands are not equal.
- The **&&** (AND) operator returns **true** if both of its operands are **true**.
- The **||** (OR) operator returns **true** if either of its operands are **true**.

Arrays and Tuples

- An **array** is an ordered collection of homogenous data. Declare it with a pair of square brackets.
- A **tuple** is an ordered collection of heterogenous data. Declare it with a pair of parentheses.
- Rust assigns an order in line (the **index position**) to each element. The index starts counting from 0.
- Use square brackets to access an array element by its index position.
- Use dot syntax to access a tuple element by its index position.

Traits

- A **trait** is a contract that requires that a type support one or more methods.
- A type can opt in to implementing a trait. A type can implement multiple traits. A trait can be implemented by multiple types.

Debug and Display Traits

- The **Display** trait mandates that a type can represent itself as a user-friendly string.
- The **Debug** trait mandates that a type can represent itself as a string for developer debugging.
- Use **:?** in curly braces to render the **Debug** representation of a type. Add a **#** for pretty-print formatting.

The **dbg!** Macro

- The **dbg!** macro prints a technical representation of its argument along with the file and line number.
- It's a good shortcut to print the **Debug** representation of a value.
- All macros end with an **!**.

Ranges

- A **range** is a sequence or interval of consecutive values.
- The **a..b** syntax creates a range from a to b where b is exclusive.
- The **a..=b** syntax creates a range from a to b where b is inclusive.
- Iterate over a range with the **for in** construct.

Generics

- A **generic** is a placeholder for a future type much like a parameter is a placeholder for a future value.
- A **generic** is a type argument.
- Generics enable reuse by enabling the design of a type without knowing all of the concrete types that will fill in for the generic.
- A Range supports a generic that represents the type of the range's elements.