Lesson 3.1: Scalar Types

Rust utilizes five scalar types: the unit type, various integers, floats, booleans, and characters, which represent a single Unicode scalar value. Integers include signed (i) and unsigned (u) types, where i32 is the default if the type is not explicitly annotated.

Key Takeaways

- The unit type (()) is a zero-bit value used when a function returns nothing.
- usize and isize are platform-dependent integer types corresponding to the size of a pointer, and usize is necessary for indexing into collections. ◦ Floating point types default to f64 for greater precision, but f32 is available; literals must contain a digit before the decimal point.
- The char type is always four bytes (a Unicode scalar), and users will usually deal with UTF-8 strings rather than characters internally.

Key Concepts

- **Unit Type (()):** Represents the absence of data, serving as the return type for functions that produce no value.
- **usize/isize:** Platform-specific integers essential for addressing memory and indexing arrays or vectors.
- **Character (char):** Represents a single Unicode scalar value (4 bytes), defined by single quotes.
- **Type Suffixing:** Optionally adding a type suffix to a literal (e.g., 128u8) when a specific type is required, especially when passing values to generic functions.

Lesson 3.2: Compound Types

Compound types group multiple values together, starting with tuples and arrays. Tuples can store fixed collections of values of different types, and their members can be accessed using zero-based dot syntax or destructuring.

Key Takeaways

- Tuples are limited to a maximum of 12 elements (arity) for full functionality.
- Arrays are compound types that store fixed collections of values, all of which must be of the same type.
- Arrays are fixed-size; their size must be determined at compile time, which often leads developers to use vectors instead.
- Array type annotations include the type and the size separated by a semicolon (e.g., [f64; 2]).

Key Concepts

- **Tuple:** A fixed-size collection that can hold multiple values of disparate types, defined by parentheses.
- **Destructuring (Tuple):** Using a pattern in a let statement to assign individual tuple members to new variables simultaneously.
- **Array (Fixed Size):** A fixed-length sequence of elements where all elements share the same type.

Lesson 3.3: Exercise D - Simple Types

Exercise D provided hands-on experience using tuple indexing (.0, .1) to access members and creating fixed-size arrays with explicit type annotations. It also demonstrated accessing deeply nested data structures, requiring a chain of tuple and array indexing.

Key Takeaways

- Explicit array type annotation helps clarify code, specifying the type and count (e.g., [f64; 2]).
- Complex nested data requires careful use of both tuple indexing (dot syntax) and array indexing (square brackets) in sequence.
- Function arguments can be destructured; replacing a single tuple argument z with (x, y) in the parameter list improves function readability.

Key Concepts

- **Tuple Indexing (Access):** Using the dot operator followed by the index number (starting at 0) to retrieve specific elements from a tuple.
- **Destructuring Function Arguments:** A feature allowing a tuple passed into a function to be immediately broken down into named variables in the parameter list.

Lesson 3.4: Control Flow

Rust uses **if** as an expression (returning a value), not a statement, which replaces the need for the conditional ternary operator found in C. All blocks in an **if** expression must evaluate to the same type, and conditions must strictly evaluate to a Boolean, as Rust avoids type coercion.

Key Takeaways

- When an expression (like if) is used in a statement, a semicolon is required after the closing brace.
- The **loop** keyword creates an unconditional loop that requires a break statement for termination.
- Nested loops can be terminated using a **loop label** (e.g., 'label: loop) provided to the break command. ◦ The for loop iterates over any iterable value and often uses ranges, where .. is exclusive and ..= is inclusive of the end point.

Key Concepts

- **If Expression:** A control flow structure that returns a value; the braces are never optional, preventing common C pitfalls.
- **Loop Label:** An identifier (starting with a single apostrophe) used to target a specific outer loop for break or continue operations.
- **For Loop (Iterable):** A structure similar to scripting languages that can iterate over any iterable value, and supports destructuring the items received.
- **Range Syntax:** Defining an iteration sequence using start..end (exclusive) or start..=end (inclusive).

Lesson 3.5: Exercise E - Control Flow

Exercise E focused on implementing different looping constructs: using an unconditional loop with an internal if/break condition, using a for loop with an inclusive range (..=) for summation, and using a while loop based on the vector's length. It also required chaining conditional logic using if, else if, and else inside an iteration.

Key Takeaways

- The **loop** construct is flexible, relying entirely on explicit break commands to halt execution.
- Ranges must be correctly specified (e.g., 7..=23 or 7..24) to achieve the desired inclusive iteration result.
- while loops are useful for scenarios where termination depends on the current state of a collection, such as checking fives.len().

Key Concepts

- **Unconditional Loop:** A loop construct designed for scenarios where the exit condition must be checked internally using conditional logic and a break statement.
- **Inclusive Range: Using the ..**= syntax ensures that the last value in the range is processed by the loop.

Lesson 3.6: Strings

Rust utilizes two main string types: the immutable borrowed string slice (&str) (used for literals) and the mutable, owned String (capital S), both guaranteed to be valid UTF-8. Due to the variable length of Unicode characters (scalers and graphemes), strings cannot be indexed by character position, as this would violate Rust's guarantee of constant-time indexing operations.

Key Takeaways

- A mutable String is typically created from an immutable &str using methods like .to_string().
- Graphemes (what a user perceives as a character) decompose into variable amounts of Unicode scalers, which in turn decompose into variable amounts of bytes.
- To safely access string content, users must choose an iterator: .bytes() (fixed size, good for ASCII overlap), .chars() (Unicode scalers), or external crates (like Unicode Segmentation for graphemes). ◦ Instead of indexing, iterators offer the .nth() method to retrieve elements after traversing the variable-length sequences.

Key Concepts

- **&str (Borrowed String Slice):** An immutable reference to string data, containing a pointer and a length.
- **String (Owned String):** A mutable string that owns its data, consisting of a pointer, length, and capacity.
- **UTF-8 Complexity:** Unicode characters require 1–4 bytes in UTF-8, making direct indexing by character position impossible for constant-time safety.
- **Graphemes vs. Scalers:** Graphemes are the desired user-facing characters, but they may be composed of multiple Unicode scalar values, further complicating indexing.

Lesson 3.7: String Literals

String literals support standard escape sequences (e.g., \n or \u{hex} for Unicode code points) and are multi-line by default. To format multi-line strings cleanly, a backslash at the end of a line removes the newline character and all leading whitespace on the next line.

Key Takeaways

- Raw strings, prefixed with an R (e.g., r"..."), prevent the evaluation of escape sequences, making them ideal for patterns like regular expressions.
- To embed quotation marks in a raw string, the literal must be enclosed using a balanced number of hash symbols outside the quotes (e.g., r#"..."#).
- Using an escaped newline followed by a backslash (\n\) allows trimming indentation while still preserving a newline character.

Key Concepts

- **Unicode Escape:** Specifying a Unicode code point using the escape sequence \u followed by the hexadecimal number in curly braces.
- **Raw String (r"..."):** A literal string where backslashes are not interpreted as escape sequences
- **Indentation Trimming:** A feature where a backslash at the line end within a literal removes subsequent newline and indentation whitespace.

Lesson 3.8: Exercise F (Strings)

Exercise F applied knowledge of string literals by using Unicode escape codes to print specific emojis and utilizing newline escape codes (\n) to format multi-line text correctly. It also required converting an immutable string literal (borrowed slice) into an owned String using the .to_string() method.

Key Takeaways

- The Unicode escape format \u{hex_code} is crucial for including specific Unicode characters.
- The .to_string() method creates a heap-allocated, owned String from a borrowed string slice.
- Newline characters must be explicitly handled via escapes (\n) or special indentation trimming techniques for formatted output.

Key Concepts

- **\u{hex}:** The syntax for including a Unicode scalar value directly into a string literal.
- **String Conversion:** The process of changing a borrowed string slice (&str) to an owned, mutable String using .to_string()..