Ownership

Learn to Code with Rust / Section Review

Ownership

- **Ownership** is a compiler feature to prevent memory errors found in other languages.
- Every value in a program has one owner at a time.
- The owner is responsible for cleaning up a value's data.
- A value's owner can change over the course of the program.
- A **move** is the transfer of ownership from one owner to another.

The Stack

- The **stack** and the **heap** are two different parts or regions of the computer's memory that are available to a Rust program when it runs.
- The stack is generally faster, but it only supports data of a fixed size that is known at compile time.
- The stack follows a last in, first out (LIFO) strategy for cleanup.

The Heap

- The heap stores data of a dynamic size at runtime. Dynamic means "unpredictable" or "not fixed".
- The memory allocator finds an empty spot in the heap that is large enough to store the data.
- The memory allocator returns a **reference**, a memory address to the heap value.
- An address is a different idea from the value itself. An address is what we can follow to get to the value.

Basics of Ownership

- A **scope** is the boundary within a program where a name is valid.
- A scope ends when a block concludes. A **block** is a region of code created by a pair of curly braces.
- The owner cleans up its data when its name goes out of scope.

The Copy Trait

- A **trait** is a contract that requires that a type support a functionality.
- A type implements the **Copy** trait if it can produce a duplicate of itself automatically in specific situations.
- When we assign one variable to another, Rust will make a full copy of the value if its type implements the **Copy** trait.
- Primitive values like integers, floats, and booleans all implement the **Copy** trait.

The **str** Type

- Rust has 2 core string types.
- The first type is a string literal, also called a string slice. This is the **str** version.
- Create a string slice with a pair of double quotes and a hardcoded value.
- **String literals** are embedded directly into the binary, the executable file that the Rust compiler produces from our source code.

The **String** Type I

- The **String** type lives on the heap and supports dynamic text. It can grow and shrink in size.
- The **String** type does not implement the **Copy** trait.
- Therefore, ownership moves from one owner to another when we assign a **String** to another variable or pass it into a function.

The **String** Type II

- The **clone** method creates a duplicate of a value.
- Cloning creates a separate, independent copy of the value, so ownership does not move.
- The **drop** function invalidates a name and deallocates the corresponding heap memory.
- Rust calls **drop** automatically at the end of the scope.

References

- A **reference** is an address of a value in memory. In other languages, it's called a pointer.
- A **reference** allows the program to use a value without transferring ownership.
- We describe this action of creating a reference as "borrowing".
- Create a reference with the **borrow operator**, the **&** symbol.

The Dereference Operator

- The **dereference operator** (*) follows a reference to the original value.
- In common operations like printing out a value or invoking a method, Rust will automatically dereference the address.
- Immutable references implement the **Copy** trait. Rust will create a copy of the reference and the original one will remain valid.

Function Parameters

- Function parameters follow the same rules of ownership (**copying** vs. **moving**).
- If a type does not implement the **Copy** trait, ownership moves to the parameter.
- If the type implements the **Copy** trait, Rust creates a copy of the value for the parameter.
- If ownership moves, the function has to return the value to the caller to prevent the memory from being deallocated.