References and Borrowing

Learn to Code with Rust / Section Review

References

- A **reference** stores the memory address of a value.
- **Borrowing** means creating a reference.
- References enable the reuse of data without moving ownership.

Immutable References

- References are immutable by default.
- An **immutable reference** does not have permission to modify the original value at the memory address.
- A value can have any number of immutable references. There is no risk.
- Immutable references implement the **Copy** trait. Rust will create a full copy in situations where one is needed (variable assignment, function parameters, variable inside array, etc).

Mutable References

- An **mutable reference** has permission to modify the original value at the memory address.
- A value can only have one mutable reference at a time.
- Mutable references do not implement the **Copy** trait. Ownership will move on variable reassignment.
- The compiler understands the reference's lifetime, which is the time it is being utilized in the program. A lifetime can end before the function's scope.

Dangling References

- A **dangling reference** is a pointer to a memory address that has been deallocated.
- Dangling references create bugs and unpredictable behaviors in other programming languages.
- The Rust compiler prevents dangling references. A reference is guaranteed to point to valid data.
- The referent (the original data) must outlive the reference.

Ownership with Composite Types

- Composite types like **arrays** and **tuples** have ownership over their elements.
- If an value implements the **Copy** trait, Rust will create a copy of it when we index into the type.
- If an value does not implement the **Copy** trait, ownership will move from the composite type to the new owner.