Ownership & Borrowing in Rust

Updated January 24, 2026 · 3 min read

Hi Hello!! This is my first blog and in this I would elaborate on the topic of how I struggled and learned the rust programming language as a fresher(0 experience) in the IT industry. When I started to learn concepts like borrowing and reference and error like "This variable no longer holds this value" makes me confused. When I found that the 3 rules were the rust follows ownership and based on this the variable can be reused, transferred, and copied. Ownership in rust is a critical topic to know and work on rust. It deals with memory management and how the variables are used with the reference. Ownership is a set of rules that govern how a rust program manages memory. All programs have to manage how they use a computer`s memory while running. Rust uses a third approach, memory is managed through a system of ownership with a set of rules that the compiler checks. If any of the rules are violated, the program won`t compile.

There are three important rules in ownership:

1. Each value in rust has a variable that`s its owner
2. There can be only one owner at a time
3. When the owner goes out of scope, the value will be dropped

Why Rust Ownership exists?
Although the programming languages such as python, java follows high level programming and a garbage collector to resolve memory management internally but rust doesn`t have a garbage collector to clear memory for free and use efficiently in memory it uses ownership

- C/C++ ⟶ memory leaks, dangling pointers
- Java/Python ⟶ garbage collector (runtime cost)
- Rust ⟶ safety at compile time

Ownership:

Ownership: One Value, One Owner

1. Each value in rust has a variable that`s its owner

```
let s = String::from(“hello”);    // s is the owner of hello
```

2. There can be only one owner at a time

```
{
let s = String::from("hi");
}                              // s goes out of scope ⟶ memory freed automatically
```

3. Ownership can be moved

```
let s1 = String::from("rust");
let s2 = s1;      // ownership moves to s2
// println!("{}", s1); ❌ error
```

COPY VS MOVE:

```
let a = 10;
let b = a;
println!("{}", a); // works
```

```
let s1 = String::from("hi");
let s2 = s1; // move
```

Ownership and Functions:

Passing value → ownership moves

```
fn take(s: String) {
println!("{}", s);
```

Ownership and Functions:

Passing value → ownership moves

```
fn take(s: String) {
println!("{}", s);
}
let s = String::from("hello");
take(s);
```

Returning ownership back:

```
fn give_back(s: String) -> String {
s
}
let s1 = String::from("rust");
let s2 = give_back(s1);
```

Reference:

Reference:

A reference lets you use a value without owning it.

```
let s = String::from("rust");
let r = &s; // reference          s⟶owner  & r⟶borrower
```

Immutable reference : Only reads

```
let s = String::from("hello");
let r1 = &s;
let r2 = &s;
println!("{}", r1);
println!("{}", r2);
```

Mutable reference: Read and write

```
let mut s = String::from("hello");
let r = &mut s;
r.push_str(" world");
println!("{}", r);
```

For deep understanding of rust : https://www.notion.so/RUST-2d0221a27852809a8903c1aca2101a3c

Conclusion:

Ownership and borrowing feel restrictive at first, but they slowly change the way you think about memory.