Learning Rust: A Beginner's Guide

From Zero to Rustacean

Rust Tutorial Series

December 14, 2025

Table of Contents

What is Rust?

Definition

Rust is a systems programming language that focuses on:

- **Safety**: Memory safety without garbage collection
- **Speed**: Performance comparable to C/C++
- **Concurrency**: Fearless concurrent programming

Fun Fact

Created by Mozilla Research, Rust has been voted the "most loved programming language" in Stack Overflow surveys for several years.

Why Learn Rust?

1. **Memory Safety**: No null pointers, no data races
2. **Zero-Cost Abstractions**: High-level features without runtime overhead
3. **Growing Ecosystem**: Excellent package manager (Cargo) and rich libraries
4. **Career Opportunities**: High demand in systems programming, web services, blockchain
5. **Great Tooling**: Excellent compiler errors, built-in testing, documentation tools

Hello, World!

```
fn main() {
    println!("Hello, World!");
}
```

Key Elements

- `fn main()` - Entry point of every Rust program
- `println!` - A macro (note the `!`)
- Statements end with semicolons `;`

Installing Rust

Installation

Visit `rustup.rs` and run:

```
curl --proto '=https' --tlsv1.2 -sSf
  https://sh.rustup.rs | sh
```

Verify Installation

```
rustc --version
cargo --version
```

Variables and Mutability

Immutable by Default

```
let x = 5;
// x = 6;  // ERROR! Cannot assign twice to immutable
```

Mutable Variables

```
let mut y = 5;
y = 6;  // OK!
println!("y = {}", y);
```

Data Types

Scalar Types:

```
// Integers
let a: i32 = 42;
let b: u64 = 100;

// Floats
let c: f64 = 3.14;

// Boolean
let d: bool = true;

// Character
let e: char = 'R';
```

Compound Types:

```
// Tuple
let tup = (500, 6.4, 'x');
let (x, y, z) = tup;

// Array
let arr = [1, 2, 3, 4, 5];
let first = arr[0];
```

Functions

```rust
fn add(a: i32, b: i32) -> i32 {
    a + b  // Expression (no semicolon)
}

fn subtract(a: i32, b: i32) -> i32 {
    return a - b;  // Explicit return
}

fn main() {
    println!("5 + 3 = {}", add(5, 3));
    println!("10 - 4 = {}", subtract(10, 4));
}
```

Control Flow: if/else

```rust
fn main() {
    let number = 7;

    if number < 5 {
        println!("Less than 5");
    } else if number < 10 {
        println!("Between 5 and 10");
    } else {
        println!("10 or greater");
    }

    // if is an expression
    let x = if number < 5 { 1 } else { 2 };
}
```

Loops

loop:

```rust
let mut count = 0;
loop {
    count += 1;
    if count == 10 {
        break;
    }
}
```

while:

```rust
let mut n = 3;
while n != 0 {
    println!("{}!", n);
    n -= 1;
}
```

for:

```rust
let arr = [10, 20, 30];
for element in arr.iter() {
    println!("{}", element)
        ;
}

// Range
for n in 1..4 {
    println!("{}!", n);
}
```

Ownership Rules

The Three Rules of Ownership

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

Why Ownership?

Ownership enables memory safety without garbage collection!

Move Semantics

```rust
fn main() {
    let s1 = String::from("hello");
    let s2 = s1;  // s1 is moved to s2

    // println!("{}", s1);  // ERROR! s1 no longer
        valid
    println!("{}", s2);      // OK
}
```

Key Point

When we assign s1 to s2, the String data is **moved**, not copied. s1 is no longer valid!

Clone and Copy

Clone (Deep Copy):

```
let s1 = String::from("hello");
let s2 = s1.clone();

// Both valid
println!("{}", s1);
println!("{}", s2);
```

Copy (Stack Only):

```
let x = 5;
let y = x;  // Copy, not move

// Both valid
println!("{}", x);
println!("{}", y);
```

Copy Types

Simple types on the stack: integers, floats, booleans, characters

References

```rust
fn main() {
    let s1 = String::from("hello");
    let len = calculate_length(&s1);

    println!("Length of '{}' is {}", s1, len);
    // s1 is still valid!
}

fn calculate_length(s: &String) -> usize {
    s.len()
}  // s goes out of scope but doesn't drop the String
```

Borrowing

Creating a reference is called **borrowing**. References don't take ownership.

The Borrowing Rules

Rules Enforced at Compile Time

At any given time, you can have **either**:

- One mutable reference, **OR**
- Any number of immutable references

Prevents Data Races!

Rust prevents data races at compile time through these rules.

Mutable References

```
fn main() {
    let mut s = String::from("hello");
    change(&mut s);
    println!("{}", s);
}

fn change(s: &mut String) {
    s.push_str(", world");
}
```

Restriction

You can only have ONE mutable reference to a value in a scope!

Slices

```
fn main() {
    let s = String::from("hello world");

    let hello = &s[0..5];    // or &s[..5]
    let world = &s[6..11];   // or &s[6..]

    println!("{} {}", hello, world);
}
```

String Slice Type: &str

Slices reference a contiguous sequence without taking ownership.

Defining Structs

```rust
struct User {
    username: String,
    email: String,
    sign_in_count: u64,
    active: bool,
}

fn main() {
    let user1 = User {
        email: String::from("user@example.com"),
        username: String::from("user123"),
        active: true,
        sign_in_count: 1,
    };

    println!("User: {}", user1.username);
}
```

Methods

```
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }

    fn square(size: u32) -> Rectangle {
        Rectangle { width: size, height: size }
    }
}
```

Enums

```
enum IpAddr {
    V4(u8, u8, u8, u8),
    V6(String),
}

enum Message {
    Quit,
    Move { x: i32, y: i32 },
    Write(String),
    ChangeColor(i32, i32, i32),
}

let home = IpAddr::V4(127, 0, 0, 1);
let msg = Message::Write(String::from("hello"));
```

Pattern Matching

```
enum Coin {
    Penny,
    Nickel,
    Dime,
    Quarter,
}

fn value_in_cents(coin: Coin) -> u8 {
    match coin {
        Coin::Penny => 1,
        Coin::Nickel => 5,
        Coin::Dime => 10,
        Coin::Quarter => 25,
    }
}
```

Option Enum

```rust
fn divide(a: i32, b: i32) -> Option<i32> {
    if b == 0 {
        None
    } else {
        Some(a / b)
    }
}

fn main() {
    match divide(10, 2) {
        Some(result) => println!("Result: {}", result)
            ,
        None => println!("Cannot divide by zero"),
    }
}
```

No Null in Rust!

`Option<T>` replaces null values safely.

Vectors

```rust
fn main() {
    // Creating vectors
    let v1: Vec<i32> = Vec::new();
    let v2 = vec![1, 2, 3];

    // Adding elements
    let mut v3 = Vec::new();
    v3.push(5);
    v3.push(6);

    // Accessing elements
    let third = &v3[2];
    match v3.get(2) {
        Some(third) => println!("Third: {}", third),
        None => println!("No third element"),
    }
}
```

Hash Maps

```rust
use std::collections::HashMap;

fn main() {
    let mut scores = HashMap::new();

    scores.insert(String::from("Blue"), 10);
    scores.insert(String::from("Yellow"), 50);

    // Getting values
    let team = String::from("Blue");
    let score = scores.get(&team);

    // Iterating
    for (key, value) in &scores {
        println!("{}: {}", key, value);
    }
}
```

Result Type

```rust
use std::fs::File;

fn main() {
    let f = File::open("hello.txt");

    let f = match f {
        Ok(file) => file,
        Err(error) => {
            println!("Error: {:?}", error);
            return;
        }
    };
}
```

The ? Operator

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_file() -> Result<String, io::Error> {
    let mut f = File::open("hello.txt")?;
    let mut s = String::new();
    f.read_to_string(&mut s)?;
    Ok(s)
}

// Even more concise
fn read_file_short() -> Result<String, io::Error> {
    std::fs::read_to_string("hello.txt")
}
```

Generic Functions

```rust
fn largest<T: PartialOrd>(list: &[T]) -> &T {
    let mut largest = &list[0];

    for item in list {
        if item > largest {
            largest = item;
        }
    }

    largest
}

fn main() {
    let numbers = vec![34, 50, 25, 100, 65];
    println!("Largest: {}", largest(&numbers));
}
```

Traits

```rust
trait Summary {
    fn summarize(&self) -> String;
}

struct Article {
    headline: String,
    content: String,
}

impl Summary for Article {
    fn summarize(&self) -> String {
        format!("{}", self.headline)
    }
}
```

Threads

```rust
use std::thread;

fn main() {
    let handle = thread::spawn(|| {
        for i in 1..10 {
            println!("Thread: {}", i);
        }
    });

    for i in 1..5 {
        println!("Main: {}", i);
    }

    handle.join().unwrap();
}
```

Channels

```rust
use std::sync::mpsc;
use std::thread;

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        tx.send(String::from("hi")).unwrap();
    });

    let received = rx.recv().unwrap();
    println!("Got: {}", received);
}
```

Shared State with Mutex

```rust
use std::sync::{Arc, Mutex};
use std::thread;

fn main() {
    let counter = Arc::new(Mutex::new(0));
    let mut handles = vec![];

    for _ in 0..10 {
        let counter = Arc::clone(&counter);
        let handle = thread::spawn(move || {
            *counter.lock().unwrap() += 1;
        });
        handles.push(handle);
    }

    for handle in handles {
        handle.join().unwrap();
    }
```

What We've Learned

- **Basics**: Variables, types, functions, control flow
- **Ownership**: Rust's unique memory management system
- **Borrowing**: References and the borrow checker
- **Structs & Enums**: Custom data types
- **Collections**: Vectors, strings, hash maps
- **Error Handling**: Result and Option types
- **Generics & Traits**: Code reuse and polymorphism
- **Concurrency**: Fearless concurrent programming

Next Steps

1. **Practice Daily**: Write Rust code every day
2. **Build Projects**: Apply what you've learned
3. **Read Code**: Study well-written Rust projects
4. **Join Community**: Rust forum, Discord, Reddit
5. **Advanced Topics**: Macros, async/await, unsafe Rust
6. **Contribute**: Open source Rust projects

Happy Coding, Rustacean!

Resources

Official Resources

- The Rust Book: `doc.rust-lang.org/book/`
- Rust by Example: `doc.rust-lang.org/rust-by-example/`
- Rustlings: Interactive exercises

Community

- Forum: `users.rust-lang.org`
- Reddit: `r/rust`
- Discord: Rust Community Server

This Tutorial

- GitHub: `github.com/adiel2012/rust-beginner`