

# Referencia Rápida de Rust

Propiedad, tipos, traits, coincidencia de patrones

## Fundamentos

### Hola Mundo

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

### Comandos de Cargo

```
cargo new my_project # create new project
cargo build          # compile (debug)
cargo build --release # compile (optimized)
cargo run            # build and run
cargo test           # run tests
```

### Estructura del Proyecto

<b>Cargo.toml</b>	Manifiesto del proyecto (dependencias, metadatos)
<b>src/main.rs</b>	Punto de entrada del crate binario
<b>src/lib.rs</b>	Raíz del crate de biblioteca
<b>tests/</b>	Directorio de pruebas de integración

## Variables y Mutabilidad

### Entace y Mutabilidad

```
let x = 5; // immutable by default
let mut y = 10; // mutable
y += 1;
const MAX: u32 = 100; // compile-time constant
```

### Ocultamiento

```
let x = 5;
let x = x + 1; // shadows previous x
let x = "now a string"; // can change type
```

### Tipos Escalares

<b>i8..i128, isize</b>	Enteros con signo
<b>u8..u128, usize</b>	Enteros sin signo
<b>f32, f64</b>	Punto flotante (f64 por defecto)
<b>bool</b>	<b>true / false</b>
<b>char</b>	Valor escalar Unicode (4 bytes)

### Tipos Compuestos

```
let tup: (i32, f64, char) = (42, 6.4, 'z');
let (a, b, c) = tup; // destructure
let arr: [i32; 3] = [1, 2, 3];
let first = arr[0];
```

## Funciones

### Definición

```
fn add(a: i32, b: i32) -> i32 {
    a + b // no semicolon = return expression
}
```

### Closures

```
let double = |x: i32| x * 2;
let sum: i32 = vec![1, 2, 3]
    .iter()
    .map(|x| x * 2)
    .sum();
```

### Punteros de Función y Traits

<b>fn(T) -&gt; U</b>	Tipo puntero de función
<b>Fn(T) -&gt; U</b>	Closure que toma prestado
<b>FnMut(T) -&gt; U</b>	Closure que toma prestado mutablemente
<b>FnOnce(T) -&gt; U</b>	Closure que toma propiedad

## Flujo de Control

### If / Else

```
let status = if score >= 90 { "A" }
              else if score >= 80 { "B" }
              else { "C" }; // if is an expression
```

### Bucles

```
loop { break; } // infinite
while condition { } // while
for item in &vec { } // iterator
for i in 0..10 { } // range
for (i, v) in vec.iter().enumerate() { }
```

### Etiquetas de Bucle

```
'outer: for i in 0..5 {
    for j in 0..5 {
        if i + j > 6 { break 'outer; }
    }
}
```

## Propiedad y Préstamo

### Reglas de Propiedad

1. Each value has exactly one owner.
2. When the owner goes out of scope, the value is dropped.
3. Values can be moved or cloned.

### Mover y Clonar

```
let s1 = String::from("hello");
let s2 = s1; // s1 is moved, no longer valid
let s3 = s2.clone(); // deep copy, both valid
```

### Préstamo

```
fn len(s: &String) -> usize { s.len() } // shared ref
fn push(s: &mut String) { s.push('!'); } // mutable ref
// Rule: many &T OR one &mut T, never both
```

### Tiempos de Vida

```
fn longest<'a>(a: &'a str, b: &'a str) -> &'a str {
    if a.len() > b.len() { a } else { b }
}
```

## Structs y Enums

### Struct

```
struct User {
    name: String,
    age: u32,
    active: bool,
}
let u = User { name: String::from("Alice"), age: 30, active: true };
```

### Bloque Impl

```
impl User {
    fn new(name: &str, age: u32) -> Self {
        Self { name: name.to_string(), age, active: true }
    }
    fn greeting(&self) -> String {
        format!("Hi, {}", self.name)
    }
}
```

## Enums

```
enum Shape {
    Circle(f64),
    Rect { w: f64, h: f64 },
    Point,
}
let s = Shape::Circle(5.0);
```

## Coincidencia de Patrones

### Expresión Match

```
match shape {
    Shape::Circle(r) => std::f64::consts::PI * r * r,
    Shape::Rect { w, h } => w * h,
    Shape::Point => 0.0,
}
```

### If Let y While Let

```
if let Some(val) = optional {
    println!("{val}");
}
while let Some(top) = stack.pop() {
    println!("{top}");
}
```

## Sintaxis de Patrones

<b>_</b>	Comodín, coincide con todo
<b>x @ 1..=5</b>	Enlazar rango coincidente a <b>x</b>
<b>(a, b, ..)</b>	Desestructurar tupla, ignorar el resto
<b>Some(x) if x &gt; 0</b>	Guardia de coincidencia
<b>Foo { x, .. }</b>	Struct, ignorar otros campos

## Manejo de Errores

### Result y Option

```
enum Result<T, E> { Ok(T), Err(E) }
enum Option<T> { Some(T), None }
```

### El Operador ?

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

## Manejar Errores

```
match result {
    Ok(val) => println!("{val}"),
    Err(e) => eprintln!("Error: {e}"),
}
let val = result.unwrap_or(0);
let val = result.unwrap_or_else(|_| default());
```

## Métodos Comunes

<b>.unwrap()</b>	Obtener valor o entrar en pánico
<b>.expect(msg)</b>	Obtener valor o entrar en pánico con mensaje
<b>.unwrap_or(default)</b>	Obtener valor o usar predeterminado
<b>.map(f)</b>	Transformar el valor Ok/Some
<b>.and_then(f)</b>	Encadenar operaciones (flatMap)
<b>.is_ok() / .is_some()</b>	Verificación booleana

# Referencia Rápida de Rust

## Traits

### Definir e Implementar

```
trait Summary {
    fn summarize(&self) -> String;
    fn preview(&self) -> String { // default impl
        format!("{}", ..., &self.summarize()[..20])
    }
}
impl Summary for User {
    fn summarize(&self) -> String { self.name.clone() }
}
```

### Límites de Traits

```
fn notify(item: &impl Summary) { }
fn notify<T: Summary + Display>(item: &T) { }
fn notify(item: &(impl Summary + Display)) { }
```

### Traits Comunes

<b>Display</b>	Formato de cadena para el usuario
<b>Debug</b>	Formato de depuración ( <code>{:?}</code> )
<b>Clone, Copy</b>	Duplicación (profunda / a nivel de bits)
<b>PartialEq, Eq</b>	Comparación de igualdad
<b>PartialOrd, Ord</b>	Comparación de orden
<b>Iterator</b>	<code>next()</code> para iteración
<b>From, Into</b>	Conversiones de tipos
<b>Default</b>	Constructor de valor predeterminado

## Colecciones

### Vec

```
let mut v: Vec<i32> = vec![1, 2, 3];
v.push(4);
v.pop(); // returns Option<i32>
let first = &v[0]; // panics if empty
let first = v.get(0); // returns Option<&i32>
```

### HashMap

```
use std::collections::HashMap;
let mut m = HashMap::new();
m.insert("key", 42);
m.entry("key").or_insert(0);
if let Some(val) = m.get("key") { }
```

### String

```
let s = String::from("hello");
let s = "hello".to_string();
let combined = format!("{}", s, "world");
for c in s.chars() { } // iterate characters
```

### Iteradores

```
let sum: i32 = vec![1, 2, 3].iter().sum();
let doubled: Vec<_> = v.iter().map(|x| x * 2).collect();
let evens: Vec<_> = v.iter().filter(|x| *x % 2 == 0).collect();
```

## Concurrencia

### Hilos

```
use std::thread;
let handle = thread::spawn(|| {
    println!("from spawned thread");
});
handle.join().unwrap();
```

## Canales

```
use std::sync::mpsc;
let (tx, rx) = mpsc::channel();
tx.send(42).unwrap();
let val = rx.recv().unwrap();
```

## Estado Compartido

<b>Arc&lt;T&gt;</b>	Conteo de referencias atómico (Rc seguro para hilos)
<b>Mutex&lt;T&gt;</b>	Exclusión mutua, bloquear para acceder al valor
<b>RwLock&lt;T&gt;</b>	Múltiples lectores o un escritor
<b>Send</b>	Trait: seguro para transferir entre hilos
<b>Sync</b>	Trait: seguro para compartir referencias entre hilos

## Macros y Atributos

### Macros Comunes

<b>println!()</b>	Imprimir con nueva línea
<b>format!()</b>	Retornar String formateado
<b>vec![]</b>	Crear Vec desde literales
<b>todo!()</b>	Marcador de posición, entra en pánico en runtime
<b>assert!(expr)</b>	Entrar en pánico si expr es falso
<b>assert_eq!(a, b)</b>	Entrar en pánico si a != b

### Atributos Derive

```
#[derive(Debug, Clone, PartialEq)]
struct Point { x: f64, y: f64 }
// Auto-implements Debug, Clone, PartialEq
```

### Atributos de Prueba

```
#[cfg(test)]
mod tests {
    use super::*;
    #[test]
    fn it_works() { assert_eq!(add(2, 2), 4); }
    #[test]
    #[should_panic]
    fn it_panics() { panic!("boom"); }
}
```