Rust: Seguridad, eficiencia y comodidad

Por Pablo Cabrera Vara

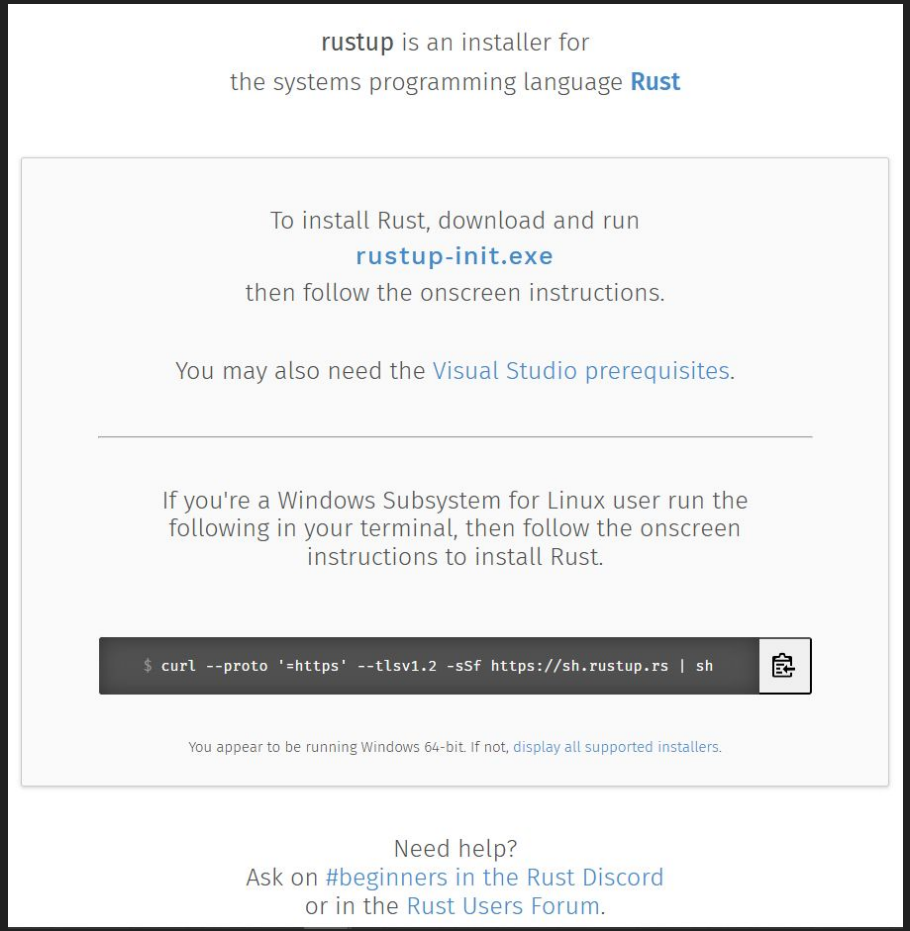
Google Developer Groups
On Campus

Advertencia Disclaimer

¿Por qué aprender rust?

Instalar Rust

https://rustup.rs/



Hay mucho contenido pero luego les comparto la presentación

Toda buena historia empieza con un hello world

```
Símbolo del sistema

C:\Users\pablo\Desktop\CositasUVa\tallerRust>cargo new hello_world
    Creating binary (application) `hello_world` package
note: see more `Cargo.toml` keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

C:\Users\pablo\Desktop\CositasUVa\tallerRust>
```

```
EXPLORER                    main.rs U
HELLO_...                   src > main.rs > main
  src                           Run | Debug
    main.rs              U    1  fn main() {
  target                     2      println!("Hello, world!");
  .gitignore             U   3  }
  Cargo.lock             U   4
  Cargo.toml             U
```

```
PS C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world> cargo run
   Compiling hello_world v0.1.0 (C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.27s
     Running `target\debug\hello_world.exe`
Hello, world!
```

```
PS C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world> cargo build --release
   Compiling hello_world v0.1.0 (C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world)
    Finished `release` profile [optimized] target(s) in 0.32s
```

Variables

```
fn main() {
    let constante_pequeña: u8 = 8;
    let mut variable_grande: i64 = 8;
    let mut variable_muy_grande: u128 = 8;
    let constante_flotante: f64 = 1.0;
    let mut variable_con_inferencia_de_tipos = 33;
    let caracter_unicode: char = '川';
    println!("Imprimimos constantes y variables: {constante_pequeña}, {variable_grande},
{variable_muy_grande}, {constante_flotante}");
}
```

```
PS C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world> cargo run
   Compiling hello_world v0.1.0 (C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world)
warning: variable does not need to be mutable
 --> src/main.rs:3:9
  |
3 |     let mut variable_grande: i64 = 8;
  |         ----^^^^^^^^^^^^^^^^
  |         |
  |         help: remove this `mut`
  |
  = note: `#[warn(unused_mut)]` on by default
```

if, else, if else

```
let a = 3;
if a == 1 {
    println!("a es 1");
} else if a == 2 {
    println!("a es 2");
} else {
    println!("a tiene un valor no esperado");
}
```

match

```
let a = 3;
match a {
    1 => println!("a es 1"),
    2 => println!("a es 2"),
    _ => println!("a tiene un valor no esperado"),
}
```

Ownership(Propiedad)

1. Cada valor tiene un único propietario.
2. Cuando el propietario de un valor sale de su alcance (scope), Rust libera automáticamente la memoria asociada a ese valor.
3. Si asignas el valor a otra variable, la propiedad se transfiere, y la variable original deja de ser válida.

Ejemplo 1(Ownership)

```
fn main() {
    let s1 = String::from("Hola, Rust");
    let s2 = s1; // s1 transfiere su propiedad a s2

    // println!("{}", s1); // Esto causaría un error, ya que s1 ha perdido su propiedad
    println!("{}", s2); // Salida: Hola, Rust
}
```

Ejemplo 2(Ownership)

```rust
fn main() {
    let s1 = String::from("Hola, Rust");
    let s2 = s1.clone(); // Clona el valor de s1 a s2

    println!("{}", s1); // Salida: Hola, Rust
    println!("{}", s2); // Salida: Hola, Rust
}
```

Borrowing(Préstamo)

1. Puedes tener múltiples referencias inmutables (**&**) a la vez.
2. Solo puedes tener una referencia mutable (**&mut**) en un momento dado.
3. No puedes mezclar referencias inmutables y mutables activas a la vez.

Ejemplo 1(Borrowing)

```rust
fn imprimir(s: &String) {
    println!("{}", s);
}

fn main() {
    let s = String::from("Hola, Rust");
    imprimir(&s); // Pasamos una referencia a la función
    println!("{}", s); // Salida: Hola, Rust, s sigue siendo válido
    // Con & se mira pero no se toca
}
```

Ejemplo 2(Borrowing)

```rust
fn modificar(s: &mut String) {
    s.push_str(", ¡aprendamos Rust!");
}

fn main() {
    let mut s = String::from("Hola");
    modificar(&mut s); // Pasamos una referencia mutable

    println!("{}", s); // Salida: Hola, ¡aprendamos Rust!
}
```

Ejemplo 3(Borrowing)

```rust
fn main() {
    let s = String::from("Rust");
    let r1 = &s;
    let r2 = &s;

    println!("r1: {}, r2: {}", r1, r2); // Ambas referencias inmutables funcionan
}
```

Ejemplo 4(Borrowing)

```rust
fn main() {
    let mut s = String::from("Rust");
    let r1 = &s;
    let r2 = &s;
    let r3 = &mut s; // Error: No se permite una referencia
                     mutable junto a inmutables

    println!("{} {} {}", r1, r2, r3);
}
```

&str

```rust
fn main() {
    let saludo: &str = "Hola, Rust!"; // Un string slice, inmutable y almacenado en el stack
    // Se almacena en el stack.
    // Como es una referencia, &str es ligero en memoria y rápido,
    // ideal para cadenas que no se modificarán.
    println!("{}", saludo);
}
```

String

```rust
fn main() {
    let mut mensaje = String::from("Hola"); // Crear un String mutable
    // Se almacena en el heap, permitiendo flexibilidad en la gestión de memoria.
    mensaje.push_str(", Rust!"); // Agrega ", Rust!" al final del String
    println!("{}", mensaje);
}
```

Conversiones entre &str y String

```
fn main() {
    let saludo = "Hola, mundo"; // &str
    // Convertir &str a String
    let saludo_string = saludo.to_string();
    // Convertir String a &str
    let referencia_str: &str = saludo_string.as_str();
    println!("{}", referencia_str);
}
```

Funciones

```rust
fn es_par(n: i32) -> bool {
    if n % 2 == 0 {
        true //return true;
    } else {
        false //return true;
    }
}

fn main() {
    let numero = 7;
    if es_par(numero) {
        println!("{} es par", numero);
    } else {
        println!("{} es impar", numero);
    }
}
```

Ejemplo 2(Funciones)

```rust
/// Función que suma 1 al valor de una variable entera mutable
fn suma_1(numero: &mut i32) -> i32 {
    *numero += 1; // Usa el operador de desreferencia `*` para modificar el valor de la referencia
    *numero       // Retorna el nuevo valor de `numero` después de la suma
}

fn main() {
    let mut valor = 5; // Definimos una variable mutable con el valor inicial de 5
    let nuevo_valor = suma_1(&mut valor); // Llamamos a `suma_1` pasando una referencia mutable

    println!("Valor después de sumar 1: {}", valor); // Imprime el valor modificado (6)
    println!("Nuevo valor retornado: {}", nuevo_valor); // Imprime el valor retornado (6)
}
```

Ejemplo 3(Funciones)

```rust
fn agregar_texto(s: &mut String) -> usize {
    s.push_str(", bienvenido a Rust!"); // Modifica el contenido del String
    s.len() // Retorna el tamaño actualizado del String
}

fn main() {
    let mut mensaje = String::from("Hola"); // Crear un String mutable
    let longitud = agregar_texto(&mut mensaje); // Pasar una referencia mutable a la función

    println!("Mensaje modificado: {}", mensaje); // Salida: "Hola, bienvenido a Rust!"
    println!("Longitud del mensaje: {}", longitud); // Salida: longitud actualizada del mensaje
}
```

Ejemplo 4(Funciones)

```rust
fn clasificar_numero(numero: i32) -> String {
    let mut descripcion = match numero {
        n if n > 0 => return String::from("Positivo"), // Retorna inmediatamente si es positivo
        n if n < 0 => "Negativo".to_string(),          // Asigna "Negativo" si es menor a cero
        _ => "Cero".to_string(),                       // Asigna "Cero" si es exactamente cero
    };

    // La descripción del número se retorna aquí si no se retornó directamente desde el match
    descripcion.push_str(" por fuera del match");

    descripcion
}

fn main() {
    let numero1 = clasificar_numero(10); // Positivo
    let numero2 = clasificar_numero(-5); // Negativo
    let numero3 = clasificar_numero(0);  // Cero

    println!("Número 1: {}", numero1);
    println!("Número 2: {}", numero2);
    println!("Número 3: {}", numero3);
}
```

```
PS C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world> cargo run
   Compiling hello_world v0.1.0 (C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.24s
     Running `target\debug\hello_world.exe`
Número 1: Positivo
Número 2: Negativo por fuera del match
Número 3: Cero por fuera del match
```

array

```
fn main() {
    let mut array: [i32; 3] = [1, 2 ,4];
    let mut array_inferido = [1, 2 ,4];
    let matriz: [[i32; 2]; 2] = [[1, 2], [2, 1]];
    println!("{}, {}", array[1], matriz[0][1]);
}
```

Vec

```rust
fn main() {
    let mut vec_vacio = Vec::new();
    vec_vacio.push(3); // [3]
    vec_vacio.push(5); // [3, 5]
    let vec_macro = vec![1, 2, 3, 4, 5]; // [1, 2, 3, 4, 5]
    let vec_repetido = vec![0; 5]; // [0, 0, 0, 0, 0]
    let mut vec_con_capacidad: Vec<i32> = Vec::with_capacity(10);
    vec_vacio.pop(); // [3]
}
```

for

```rust
fn main() {
    // Iterar sobre el rango (1..6 excluye el 6)
    for numero in 1..6 {
        println!("Número: {}", numero);
    }
}
```

Ejemplo 2(for)

```
fn main() {
    let numeros = vec![1, 2, 3, 4, 5]; // Crear un vector de enteros

    // Iterar sobre cada elemento del vector
    for numero in numeros.iter() {
        println!("Número: {}", numero);
    }
}
```

Ejemplo 3(for)

```rust
fn main() {
    let mut numeros = vec![1, 2, 3, 4, 5]; // Crear un vector mutable de enteros

    // Iterar sobre el vector con una referencia mutable para modificar cada elemento
    for numero in numeros.iter_mut() {
        *numero += 1; // Sumar 1 al valor al que apunta la referencia mutable
    }

    println!("Vector después de sumar 1 a cada elemento:  {:?}", numeros);
}
```

```
PS C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world> cargo run
   Compiling hello_world v0.1.0 (C:\Users\pablo\Desktop\CositasUVa\tallerRust\hello_world)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.29s
     Running `target\debug\hello_world.exe`
Vector después de sumar 1 a cada elemento: [2, 3, 4, 5, 6]
```

loop

```rust
fn main() {
    let mut contador = 0;

    loop {
        contador += 1;
        println!("Contador: {}", contador);

        if contador >= 5 {
            break; // Salir del bucle cuando el contador alcanza 5
        }
    }
}
```

while

```rust
fn main() {
    let mut contador = 0;

    while contador < 5 {
        contador += 1;
        println!("Contador: {}", contador);
    }
}
```

Enums

```
enum Color {
    Rojo,
    Verde,
    Azul,
}

fn main() {
    let color_favorito = Color::Verde;

    match color_favorito {
        Color::Rojo => println!("El color favorito es Rojo."),
        Color::Verde => println!("El color favorito es Verde."),
        Color::Azul => println!("El color favorito es Azul."),
    }
}
```

Enums como wrappers de tipos

```
enum Forma {
    Circulo(f64),           // radio
    Rectangulo(f64, f64),   // ancho y alto
}

fn calcular_area(forma: Forma) -> f64 {
    match forma {
        Forma::Circulo(r) => std::f64::consts::PI * r * r,
        Forma::Rectangulo(ancho, alto) => ancho * alto,
    }
}

fn main() {
    let mi_circulo = Forma::Circulo(2.0);
    let mi_rectangulo = Forma::Rectangulo(3.0, 4.0);

    println!("Área del círculo: {}", calcular_area(mi_circulo));
    println!("Área del rectángulo: {}", calcular_area(mi_rectangulo));
}
```

Gestión de errores

1. Option
2. Result
3. Unwrap()
4. Expect()

Option

```
enum Option<T> {
    Some(T), // Contiene un valor
    None, // No contiene un valor
}
```

Option

```rust
fn obtener_elemento(v: &Vec<i32>, indice: usize) -> Option<i32> {
    if indice < v.len() {
        Some(v[indice]) // Retorna el elemento si el índice es válido
    } else {
        None // Retorna None si el índice es inválido
    }
}

fn main() {
    let numeros = vec![1, 2, 3];

    match obtener_elemento(&numeros, 2) {
        Some(valor) => println!("Valor encontrado: {}", valor),
        None => println!("No se encontró ningún valor"),
    }
}
// Vec ya tiene este método solo que lo llama get()
```

Result

```
enum Result<T, E> {
    Ok(T), // Resultado exitoso
    Err(E), // Error
}
```

Result

```rust
fn dividir(dividendo: f64, divisor: f64) -> Result<f64, String> {
    if divisor == 0.0 {
        Err(String::from("No se puede dividir entre cero")) // Retorna un error
    } else {
        Ok(dividendo / divisor) // Retorna el resultado exitoso
    }
}

fn main() {
    match dividir(10.0, 0.0) {
        Ok(resultado) => println!("Resultado: {}", resultado),
        Err(e) => println!("Error: {}", e),
    }
}
```

unwrap() con Option

```
fn main() {
    let valor = Some(10);
    let numero = valor.unwrap();  // Devuelve el valor 10, ya que es Some(10)
    println!("Número: {}", numero);

    let sin_valor: Option<i32> = None;
    let numero_inexistente = sin_valor.unwrap();  // Causa un pánico porque es None
    println!("Número: {}", numero_inexistente);
}
```

unwrap() con Result

```rust
fn main() {
    let resultado: Result<i32, &str> =  Ok(20);
    let numero = resultado.unwrap();  // Devuelve el valor 20, ya que es Ok(20)
    println!("Número: {}", numero);

    let resultado_errado: Result<i32, &str> =  Err("Algo salió mal");
    let numero_fallido = resultado_errado.unwrap();  // Causa un pánico porque es Err
    println!("Número: {}", numero_fallido);
}
```

expect() con Option

```
fn main() {
    let valor = Some(15);
    let numero = valor.expect("No debería fallar aquí"); // Devuelve 15 porque es Some(15)
    println!("Número: {}", numero);

    let sin_valor: Option<i32> = None;
    let numero_inexistente = sin_valor.expect("Error: Se esperaba un número"); // Causa un pánico
con mensaje personalizado
    println!("Número: {}", numero_inexistente);
}
```

expect() con Result

```rust
fn main() {
    let resultado: Result<i32, &str> = Ok(30);
    let numero = resultado.expect("No debería fallar aquí"); // Devuelve 30 porque es Ok(30)
    println!("Número: {}", numero);

    let resultado_errado: Result<i32, &str> = Err("Algo salió mal");
    let numero_fallido = resultado_errado.expect("Error al intentar obtener el número"); // Causa un pánico con mensaje personalizado
    println!("Número: {}", numero_fallido);
}
```

Ejemplo 1(Entrada)

```rust
use std::io;

fn main() {
    let mut nombre = String::new(); // Variable donde se almacenará la entrada

    println!("Por favor, ingresa tu nombre:");

    io::stdin() // Lee desde la entrada estándar
        .read_line(&mut nombre) // Lee la línea y la guarda en `nombre`
        .expect("Error al leer la entrada"); // Maneja un posible error

    println!("Hola, {}!", nombre.trim()); // Usa trim para eliminar espacios o saltos de línea
}
```

Ejemplo 2(Entrada)

```rust
use std::io;

fn main() {
    let mut entrada = String::new();

    println!("Ingresa un número:");

    io::stdin()
        .read_line(&mut entrada)
        .expect("Error al leer la entrada");

    // Convertimos la entrada a un número i32
    let numero: i32 = entrada.trim().parse().expect("Por favor, ingresa un número válido");

    println!("Número ingresado: {}", numero);
}
```

Ejemplo 3(Entrada)

```rust
use std::io;

fn main() {
    let mut numero1 = String::new();
    let mut numero2 = String::new();

    println!("Ingresa el primer número:");
    io::stdin().read_line(&mut numero1).expect("Error al leer el primer número");

    println!("Ingresa el segundo número:");
    io::stdin().read_line(&mut numero2).expect("Error al leer el segundo número");

    let numero1: i32 = numero1.trim().parse().expect("Por favor, ingresa un número válido");
    let numero2: i32 = numero2.trim().parse().expect("Por favor, ingresa un número válido");

    let suma = numero1 + numero2;
    println!("La suma de los números es: {}", suma);
}
```

struct

```
struct Persona {
    nombre: String,
    edad: u32,
    ciudad: String,
}
```

Ejemplo 1(struct)

```
struct Persona {
    nombre: String,
    edad: u32,
    ciudad: String,
}

fn main() {
    let persona1 = Persona {
        nombre: String::from("Alice"),
        edad: 30,
        ciudad: String::from("Ciudad Rust"),
    };

    println!("{} tiene {} años y vive en {}.", persona1.nombre, persona1.edad, persona1.ciudad);
}
```

Tuple Struct

```rust
struct Punto(f64, f64);

fn main() {
    let origen = Punto(0.0, 0.0);
    let destino = Punto(5.2, 3.8);

    println!("Punto de origen: ({}, {})", origen.0, origen.1);
    println!("Punto de destino: ({}, {})", destino.0, destino.1);
}
```

Bloque impl (new)

```
struct Persona {
    nombre: String,
    edad: u32,
}

impl Persona {
    fn new(nombre: String, edad: u32) -> Self {
        Persona { nombre, edad }
    }
}

fn main() {
    let persona = Persona::new(String::from("Alice"), 30);
    println!("{} tiene {} años.", persona.nombre, persona.edad);
}
```

Bloque impl (build)

```
struct Persona {
    nombre: String,
    edad: u32,
}

impl Persona {
    // Método `build` que crea una instancia de `Persona` solo si `edad` es válida
    fn build(nombre: String, edad: u32) -> Result< Self, String> {
        if edad >= 0 && edad <= 150 {
            Ok(Persona { nombre, edad })
        } else {
            Err(String::from( "La edad debe estar entre 0 y 150." ))
        }
    }
}

fn main() {
    match Persona::build(String::from( "Bob"), 200) {
        Ok(persona) => println! ("{} tiene {} años.", persona.nombre, persona.edad),
        Err(error) => println! ("Error al crear la persona: {}", error),
    }
}
```

Bloque impl(otros métodos)

```
struct Rectangulo {
    ancho: u32,
    alto: u32,
}

impl Rectangulo {
    // Método normal `area`: calcula el área del rectángulo
    fn area(&self) -> u32 {
        self.ancho * self.alto
    }

    // Método mutable `incrementar_dimensiones`: incrementa ancho y alto en una cantidad dada
    fn incrementar_dimensiones(&mut self, incremento: u32) {
        self.ancho += incremento;
        self.alto += incremento;
    }
}
```

Ejemplo de uso de struct con impl

```
fn main() {
    let mut rect = Rectangulo::new(10, 5);
    println!("Área del rectángulo: {}", rect.area());
    println!("¿Es un cuadrado? {}", rect.es_cuadrado());
    println!("Nuevo tamaño del rectángulo: ancho {}, alto {}", rect.ancho,
rect.alto);
}
```

Módulos

```rust
mod persona {
    //Todo lo que este dentro de este bloque es privado excepto que tenga pub delante
    pub struct Persona {
        nombre: String,
        edad: u32,
    }

    impl Persona {
        pub fn new(nombre: String, edad: u32) ->Self {
            Persona { nombre, edad }
        }
    }

}

fn main() {
    let persona1 = persona::Persona::new(String::from("Alice"), 30);
    println!("{} tiene {} años.", persona1.nombre, persona1.edad);
}
```

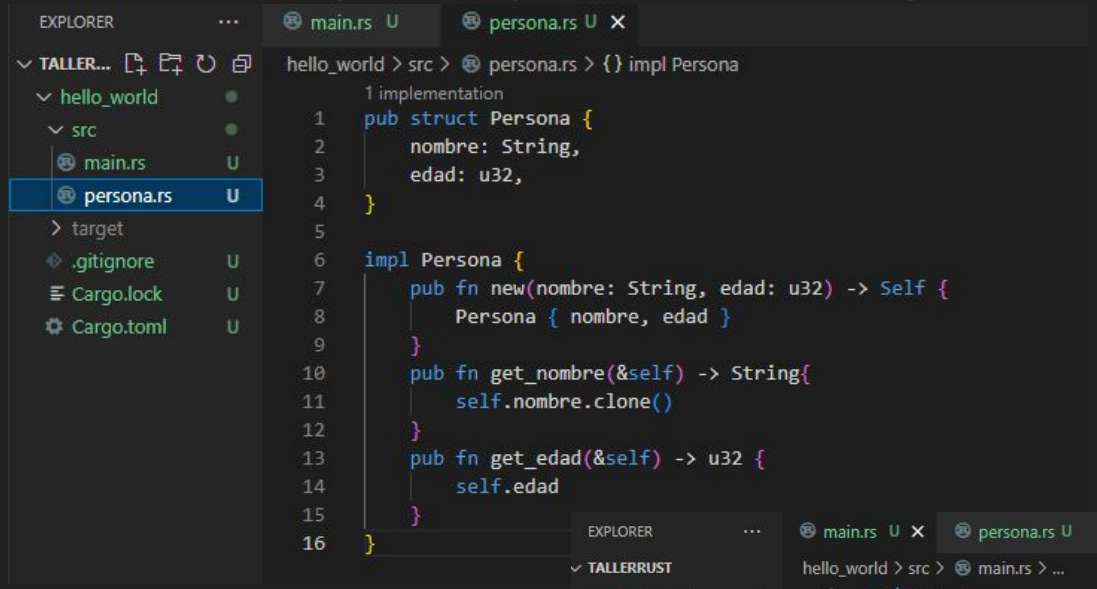
Módulos

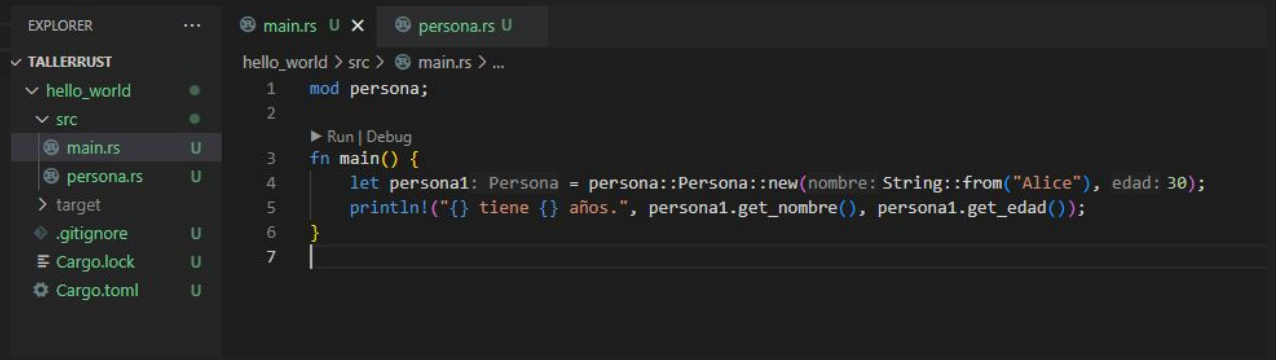
```rust
mod persona {
    //Todo lo que este dentro de este bloque es privado excepto que tenga pub delante
    pub struct Persona {
        nombre: String,
        edad: u32,
    }

    impl Persona {
        pub fn new(nombre: String, edad: u32) -> Self {
            Persona { nombre, edad }
        }
        pub fn get_nombre(&self) -> String{
            self.nombre.clone()
        }
        pub fn get_edad(&self) -> u32 {
            self.edad
        }
    }

}
fn main() {
    let persona1 = persona::Persona::new(String::from("Alice"), 30);
    println!("{} tiene {} años.", persona1.get_nombre(), persona1.get_edad());
}
```

Módulos(para gente de bien)





Ahora a trabajar

Small Rust projects

From sources across the web

Basic Calculator

Game engine

Operating system

Gracias por venir y espero que les haya gustado

QR con las diapositivas

Google Developer Groups
On Campus

Ejercicios con Rustlings ¡Vayan hasta donde se sientan valientes!

https://github.com/rust-lang/rustlings/tree/main

Google Developer Groups On Campus

QR con las diapositivas
(otra vez por si acaso)