Rust 2025

Bienvenid@s!

Metodología

Clases teóricas y prácticas

Comunicación y consultas por discord

2 entregas de ejercicios prácticos de manera individual (8/5 y 12/6)

Trabajo final grupal (10/7)

comunicación: discord

¿Qué es Rust?

Es un lenguaje de programación multiparadigma compilado de código abierto que se centra en la seguridad, la concurrencia y el rendimiento. Diseñado para ayudar a los desarrolladores a escribir código seguro y eficiente, ofrece características únicas que permiten un manejo de memoria seguro en tiempo de compilación sin la necesidad de un recolector de basura.

¿Qué es Rust? Algunas características

- **Sistema de tipos:** tiene un sistema de tipos estático y fuertemente tipado, lo que significa que el tipo de cada variable debe ser conocido en tiempo de compilación y no puede cambiar durante la ejecución del programa.

- **Seguridad de memoria:** garantiza la seguridad de memoria mediante un sistema de propiedad y préstamos, que ayuda a prevenir errores comunes como el uso después de liberar, doble liberación y corrupción de memoria.

- **Rendimiento:** compila el código directamente a código de máquina nativo y optimizado, lo que generalmente proporciona un alto rendimiento.

¿Qué es Rust? Algunas características cont..

- **Calidad y ayuda en los mensajes de errores:** Los mensajes de error que arroja el compilador de rust son muy descriptivos, detallados y en muchas oportunidades explican como subsanar el error.

- **Gestión de errores:** utiliza el tipo Result para manejar errores de manera explícita y segura. Esto puede resultar en un código más seguro y fácil de razonar.

- **Macros (metaprogramación):** admite macros para la generación de código en tiempo de compilación, lo que permite la creación de abstracciones personalizadas y la generación de código eficiente.

- **Empaquetado y administración de dependencias:** incluye cargo, una herramienta de construcción y administración de paquetes que facilita la gestión de dependencias y la construcción de proyectos.

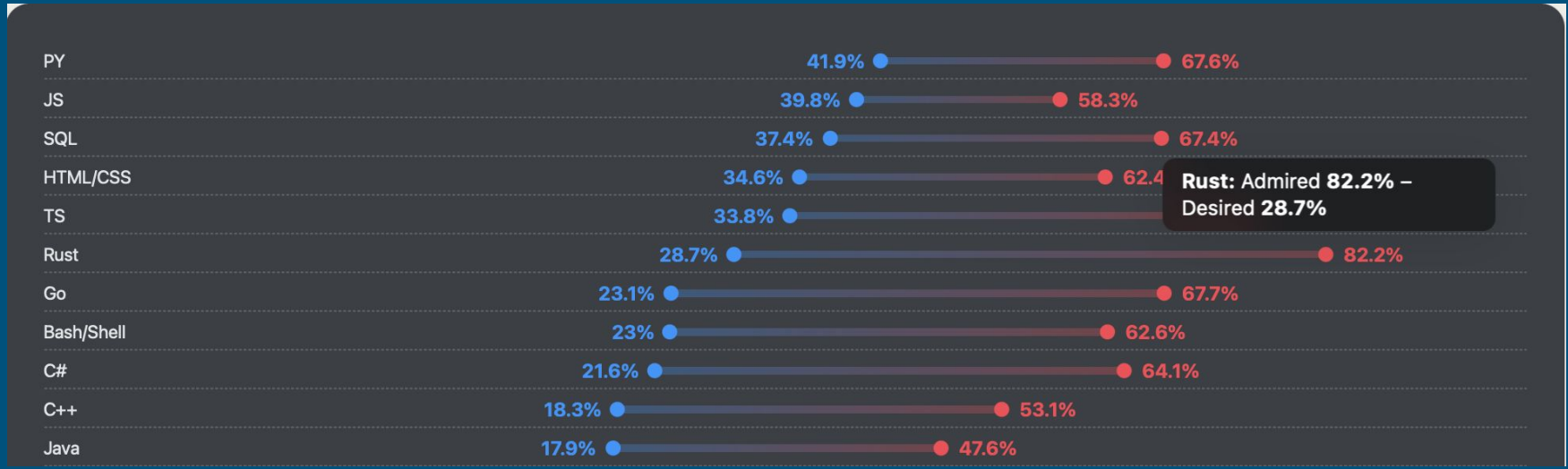
Un poco de historia

- Se comenzó a trabajar en rust por el 2006 y la versión 1.0 fue liberada en el 2015 por eso es un lenguaje muy reciente.

- Inicialmente fue pensado para programación de sistemas (sistemas operativos, browsers, engine de videojuegos)

- Fue creado por un grupo de desarrolladores de mozilla

El lenguaje más querido por devs por 9no año consecutivo en 2024



fuente

¿Para que se usa?

Aplicaciones de línea de comando

Sistemas operativos

Browser engine

Backend/Api

Escribir aplicaciones de bajo nivel para mejorar performance

Webassembly (wasm)

Sistemas embebidos, microcontroladores, iot

Blockchain

¿Donde se usa?

Firefox

Redox Os

Substrate

Fucsia Os

Android

Meta

Amazon (AWS)

¿Por que usarlo?

Seguridad de tipos: el compilador asegura operaciones correctas.

Seguridad de memoria: todas las referencias apuntarán a una dirección válida.(no existen los null pointer exception por ej)

Sin condiciones de carrera: asegura que distintas partes de un programa no pueden modificar un espacio de memoria al mismo tiempo.

Abstracciones de costo cero: permite usar conceptos de alto nivel con un costo nulo o muy pequeño en comparación a otros lenguajes.

Runtime mínimo: lo más optimizado posible similar a c , c++, sin overhead extra.

Mercado laboral creciente

Instalación

Siempre utilizaremos para todo la doc oficial

Para linux y/o mac os ejecutar lo siguiente en una terminal:

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

Para windows descargar e instalar el instalador de la versión stable según arquitectura desde acá

Compilar y ejecutar ejemplo sencillo

crearemos un archivo con extensión finalizada en .rs

dentro de él colocaremos el siguiente código:

```
fn main() {
    println!("Seminario Rust 2024!");
}
```

Compilar y ejecutar ejemplo sencillo

Abriremos una terminal y posicionados en el directorio donde creamos el archivo anterior compilaremos dicho archivo con el siguiente comando:

```
rustc nombre_de_archivo.rs
```

si observamos generó un ejecutable, para ejecutarlo haremos lo siguiente:

```
./nombre_de_archivo
```

Como hacer comentarios

Los comentarios los podremos hacer de 2 maneras:

usando:

```
//comentario de una linea
```

o usando:

```
/*
comentado un bloque
de varias lineas
*/
```

Como definir una variable

para poder definir una variable se utiliza la palabra clave let seguida del nombre de la variable:

```
let mi_variable = 5;
```

como dijimos Rust es fuertemente tipado, pero no es obligatorio declarar el tipo al declarar una variable si al hacerlo le asignamos un valor ya que el compilador hace inferencia de tipos en tiempo de compilación.

Inmutabilidad vs mutabilidad

Supongamos que tenemos el siguiente programa:

```
fn main() {
    let numero= 5;
    println!("{}", numero);
}
```

La variable numero es inmutable, esto quiere decir que durante la ejecución del programa no podremos modificar su valor. Rust nos obliga a que seamos explícitos siempre. Al intentar modificarla nos arrojaría un error cuando compilamos.

Inmutabilidad vs mutabilidad

```
fn main() {
    let numero= 5;
    numero = numero + 1;
    println!("{}", numero);
}
```

```
error[E0384]: cannot assign twice to immutable variable `numero`
 --> in.rs:3:5
  |
2 |     let numero= 5;
  |         ------
  |         |
  |         first assignment to `numero`
  |         help: consider making this binding mutable: `mut numero`
3 |     numero = numero + 1;
  |     ^^^^^^^^^^^^^^^^^^^^ cannot assign twice to immutable variable

error: aborting due to previous error

For more information about this error, try `rustc --explain E0384`.
```

Inmutabilidad vs mutabilidad

Pero si podríamos hacer lo que se llama shadowing:

```
fn main() {
    let numero= 5;
    println!("{}", numero);
    let numero= numero + 1;
    println!("{}", numero);
}
```

Inmutabilidad vs mutabilidad

ejemplo haciéndola mutable que es lo que queremos:

```
fn main() {
    let mut numero= 5;
    numero = numero + 1;
    println!("{}", numero);
}
```

Constantes

se definen con la palabra reservada const por ej si quiero definir una constante llamada MI_CONSTANTE con el valor de 10, sería de la siguiente manera:

```
const MI_CONSTANTE:u8 = 10;
```

a las constantes hay que indicarles obligatoriamente el tipo.

Son inmutables. A diferencia de las variables inmutables, siempre van a tener el mismo valor durante toda la ejecución del programa.

Tipos de datos

Tipado estático: el chequeo de los tipos de datos se hace en tiempo de compilación.

Se dividen en scalar types y compound types.

Tipos de datos (scalar types)

- Integer: con y sin signo
- Floating-Point(32 y 64 bits)
- Boolean
- Character

Tipos de datos (scalar types)

Integer:

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

ej: `let numero: u32 = 42;`

Tipos de datos (scalar types)

Integer literals:

Number literals	Example
Decimal	`98_222`
Hex	`0xff`
Octal	`0o77`
Binary	`0b1111_0000`
Byte (`u8` only)	`b'A'`

ej: `let num = 32_500;`

Tipos de datos (scalar types)

Floating-Point: como se comentó son de 32 y 64 bits y la definición es de la siguiente manera:

```
fn main() {
    let x = 3.0; // f64
    let y: f32 = 3.0; // f32
}
```

Tipos de datos (scalar types)

Operaciones:

```
fn main() {
    let suma = 7 + 3;
    let resta = 95.5 - 4.3;
    let producto = 4 * 30;
    let division = 56.7 / 32.2;
    let division2 = -5 / 3; // Resultado -1
    let resto = 43 % 5;
}
```

Tipos de datos (scalar types)

Boolean

```
fn main() {
    let t = true;
    let f: bool = false; // con explicito tipo
}
```

operaciones: `&, &&, |, || , == , ^ , > , < , >= , <= , ! , !=`

Tipos de datos (scalar types)

Character:

```
fn main() {
    let c = 'z';
    let z: char = 'ℤ'; // con tipo explicito
    let gatito = '😻';
}
```

El tipo de char de Rust tiene un tamaño de cuatro bytes y representa un valor Unicode, lo que significa que puede representar mucho más que solo ASCII. Las letras acentuadas; los caracteres chinos, japoneses, coreanos, emojis, etc.

Tipos de datos (compound types)

- String
- Tuple
- Array

Tipos de datos (compound types): strings

-str: Es un tipo de cadena de caracteres inmutable y de longitud fija.

-String: Es un tipo de cadena de caracteres que es mutable y de longitud variable.

Tipos de datos (compound types): strings

ejemplo:

```
fn main() {
    let str_fijo:&str = "Soy un string inmutable";
    let mut str_mutable:String = "Soy mutable".to_string();
    str_mutable += " concateno" ;
    println!("{}", str_mutable);
}
```

Tipos de datos (compound types): tuple

Es una forma de agrupar distintos valores y pueden tener distintos tipos.

```
fn main() {
    let mut tupla:(String, f32, u8) = ("hola".to_string(), 3.0, 3);
    tupla.0 = "cambio valor".to_string();
    println!("{}", tupla.0);
    println!("{:?}", tupla);
    let(hola, flotante, entero) = tupla;
    println!("{}", hola);
}
```

Tipos de datos (compound types): arrays

Son de tamaño fijo y tienen el mismo tipo de dato. ej:

```
fn main() {
    let arreglo = [1,2,3,5];
    println!("el tercer el elemento es: {}", arreglo[2]);
    let arreglo2 :[char ;2]= ['1', '2'];
    println!("el ultimo elemento es: {}", arreglo2.last().unwrap());
}
```

Uso de libs

En rust podremos hacer uso de libs o módulos para determinada acción y hacer reutilización de código, observemos el siguiente fragmento de código:

```
use std::io::stdin;
fn main() {
    println!("Ingrese su nombre: ");
    let mut nombre = String::new();
    stdin().read_line(&mut nombre).expect("Error al leer el nombre.");
    println!("Hola, {}!", nombre);
}
```

En la línea 1 importamos stdin de la lib standar de rust para poder leer desde teclado información.

stdin().read_line no devuelve un Result, ya lo veremos más adelante

el expect nos sirve para indicar en caso de que el Result tenga error

que mensaje arrojar al disparar un Panic!

¿Qué es cargo?

Es el administrador de paquetes de Rust como npm o pip por ej. y también nos facilita la creación de proyectos.

¿Como usar cargo?

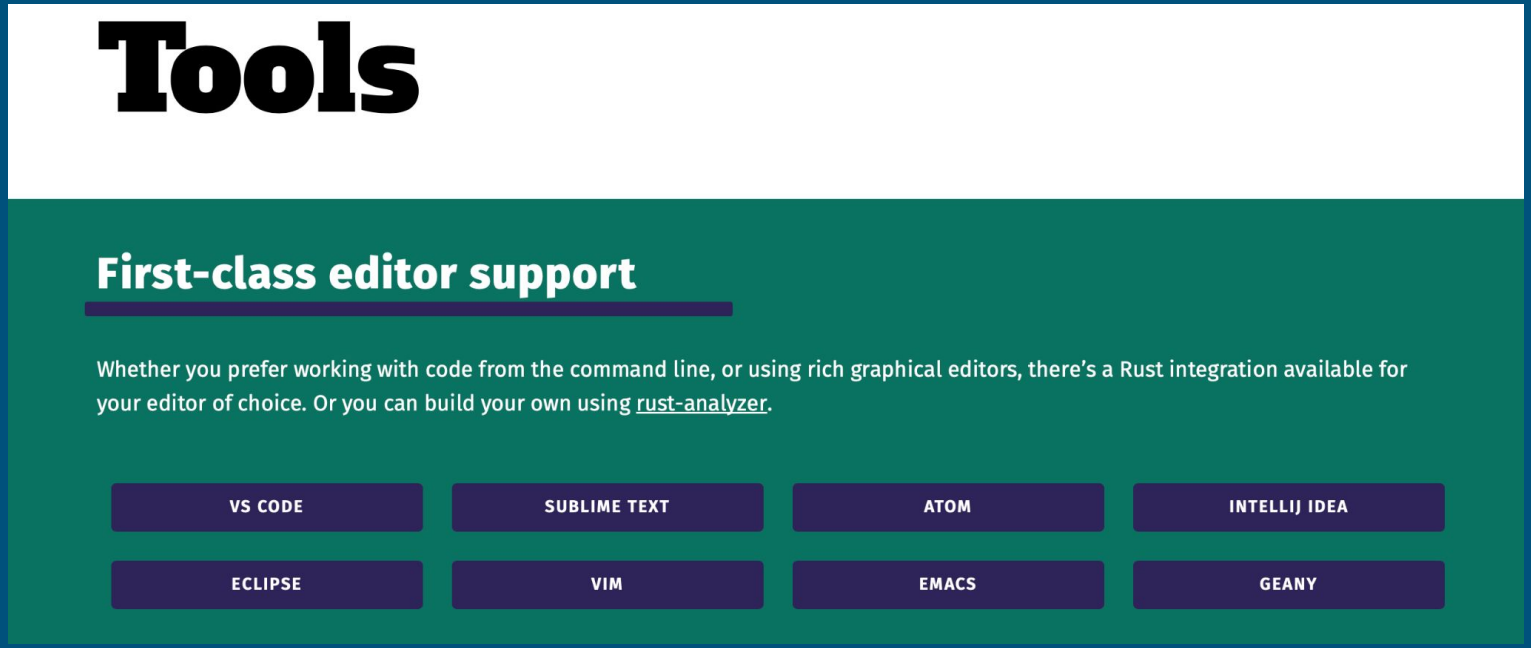
`cargo new nombre_del_proyecto`

Esto nos creará un proyecto con la siguiente estructura:
-un archivo llamado Cargo.toml
-un directorio src

otros comandos: `build, run, check`

crates: `https://crates.io`

¿Qué ide usar?


Tools
First-class editor support
Whether you prefer working with code from the command line, or using rich graphical editors, there's a Rust integration available for your editor of choice. Or you can build your own using rust-analyzer.
VS CODE
SUBLIME TEXT
ATOM
INTELLIJ IDEA
ECLIPSE
VIM
EMACS
GEANY

Are we (I)DE yet?

An overview about the state of Rust support by text editors and their integrated brethren.

Below you'll find a table listing the comparable features of editors, followed by specific information about single programs. The last part presents some more tooling of Rust's ecosystem.

	Syntax highlighting (.rs)	Syntax highlighting (.toml)	Snippets	Code Completion	Linting	Code Formatting	Go-to Definition	Debugging	Documentation Tooltips
Atom	✓	✓	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]		✓[1]
Emacs	✓[1]	✓	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]		✓[1]
Sublime	✓	✓[1]	✓	✓[1]	✓	✓[1]	✓[1]		
Vim/Neovim	✓	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]		✓[1]
VS Code	✓	✓[1]	✓	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]
Show more editors ⇩									
Eclipse	✓[1]		✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]
IntelliJ-based IDEs	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]
Visual Studio	✓			✓[1]			✓[1]	✓[1]	
GNOME Builder	✓		✓	✓	✓	✓[1]	✓		
Show more IDEs ⇩									

✓ = supported out-of-the-box, ✓[1] = supported via plugin

Vs Code extensiones recomendadas

rust-analyzer

better-toml

crates

error lens