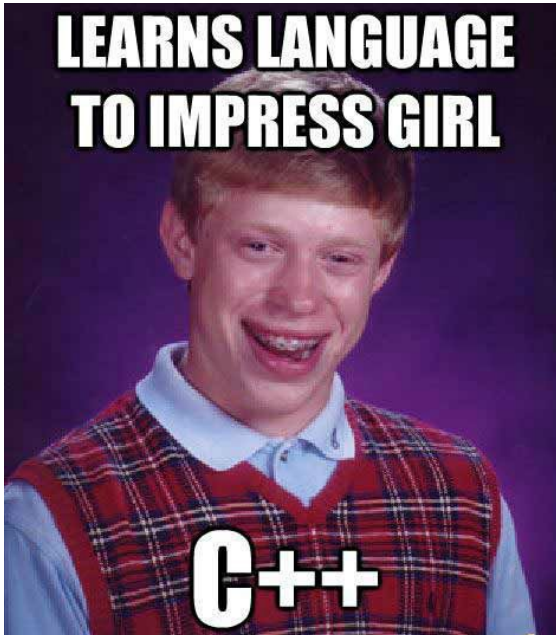
Teoría del Lenguaje (75.31)

Integrantes:

- Santiago Boselli
- Rocio Gallo Gosende
- Lucas Alexis Rack
- Federico Verstraeten
- Ezequiel Martín Zarza

¿Por Qué RUST?
¿Acaso porque somos **RUSTicos**?

Se usaba C++

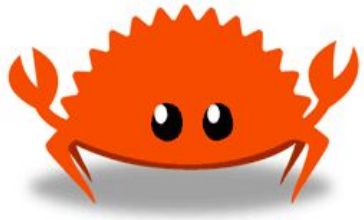


- Control total del hardware.
- Entornos limitados.
- Portabilidad.
- Mejores tiempos con menor consumo de memoria.


Como diría el tío Ben...

Filosofía

Hitos del Lenguaje



Graydon Hoare ……2006

Mozilla……2009

Lanzamiento……2010

Compilador Autocontenido……2011

Programación Orientada a Sistemas

- Servidores
- Juegos
- Sistemas operativos
- Sistemas embebidos
- Modelos científicos

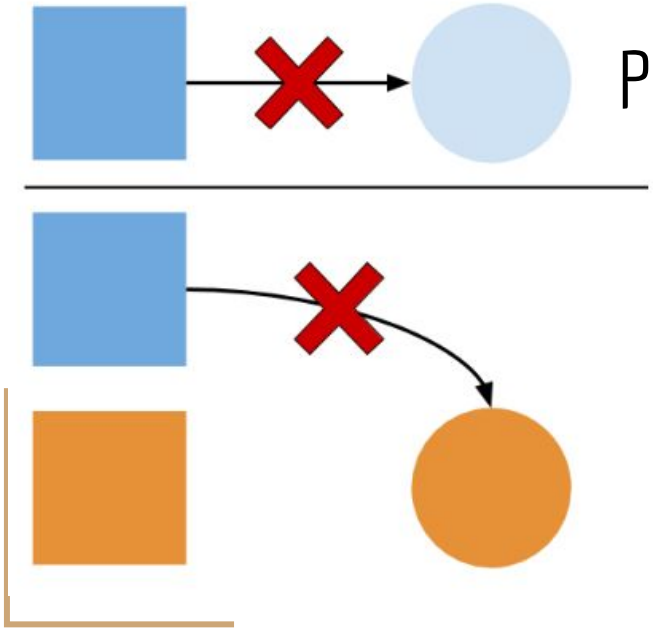
Características Principales

Seguridad en Memoria 1

Mutación: Cambiar el contenido de un puntero liberado.

Aliasing: Referencias a datos liberados.

Seguridad en Memoria 2



Para liberar/enviar un objeto de forma segura

- No le deben quedar referencias

Si se que puedo enviar un objeto,
se que puedo liberarlo.

Pertenencia

- Cada valor en tiene una variable “dueña”.
- Solo puede haber un dueño por dato.
- cuando el valor queda fuera de alcance, se tira.

Concurrencia

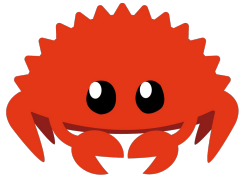
Soporte nativo.

Corrección de bugs.

Comunicación por canales.

Al día con las Tecnologías.

Traits

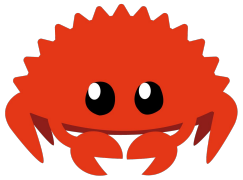


Abstraen comportamiento que los datos tienen en común.

Definen comportamiento compartido.

Zero Cost Abstraction

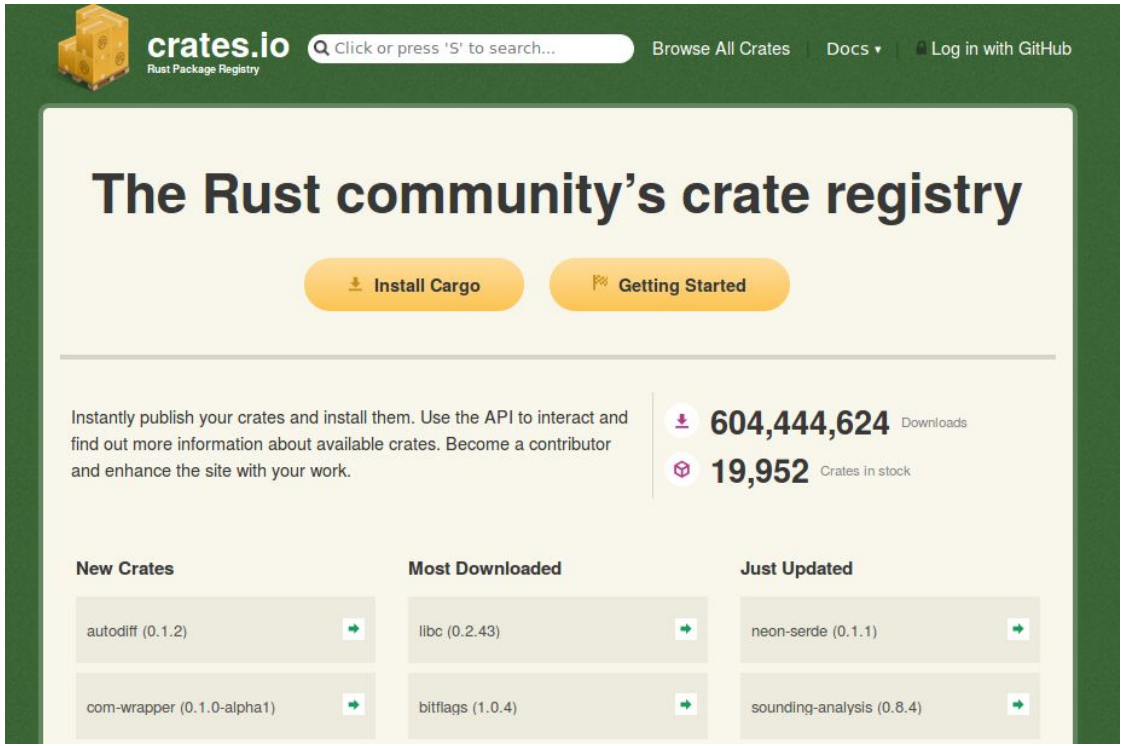
- Sin GC
- Sistema de tipos estático.
- Controles a tiempo de compilación



Rustc

- Restrictivo.
- Alienta a mejorar el código.
- Basado en patrones de diseño que evitan errores.

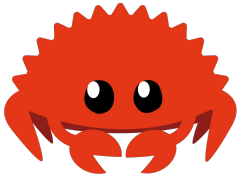
Cargo



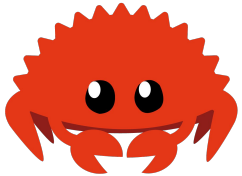
- Construir Proyectos.
- TOML.
- Genera documentación.
- Ejecuta las pruebas.
- Subir librerías.
- Comunidad.

Curiosidades Sintácticas

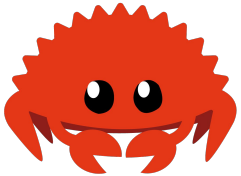
Comp. condicional



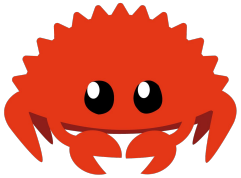
Macro_rules!



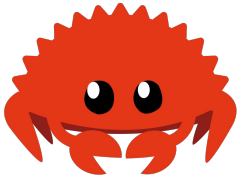
Unsafe



Closures



Testing

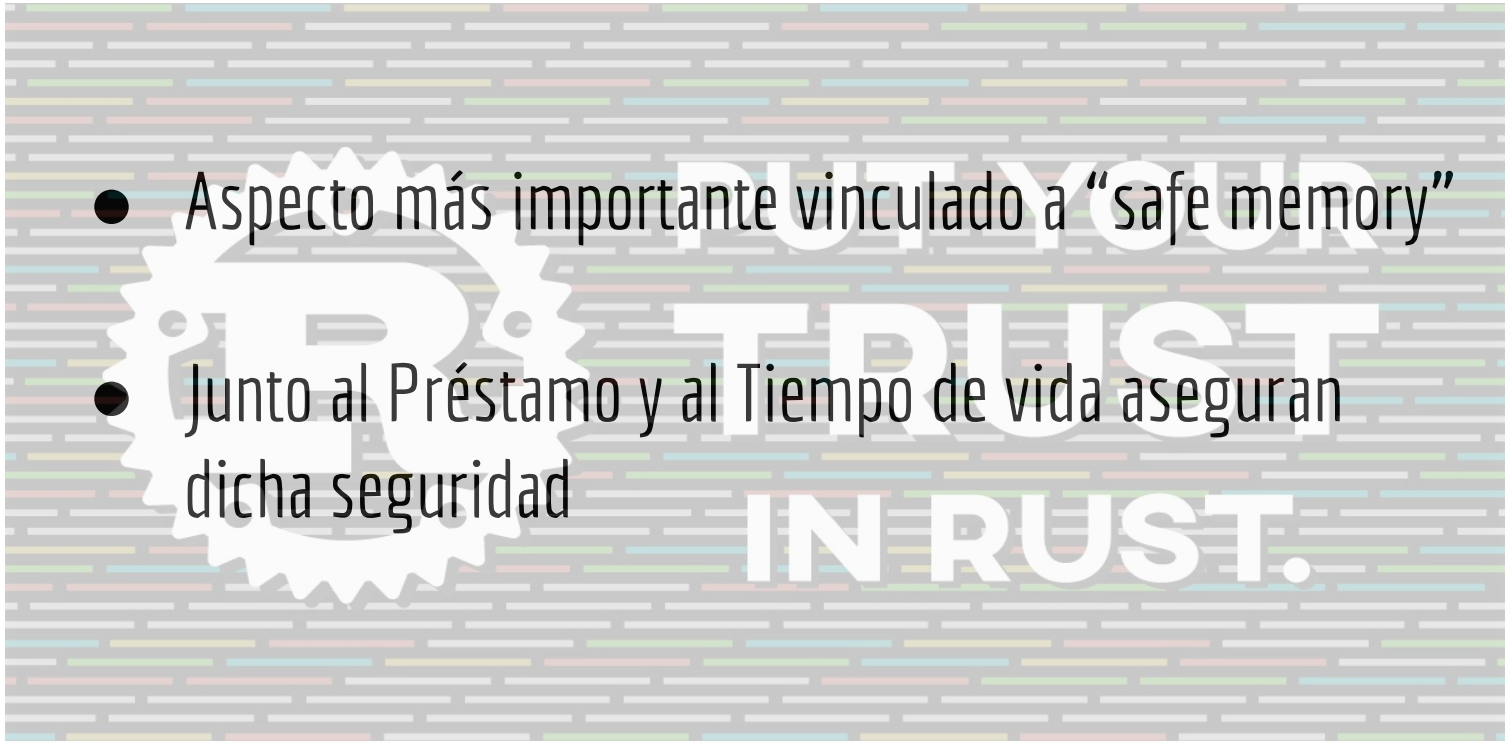


Conceptos a Explicar de Rust

- Pertenencia
- Préstamo
- Tiempos de vida
- Mutabilidad
- Manejo de errores
- Smart pointers

- Objetos en Rust
- Rust “Inseguro”
- Traits
- Concurrencia

Pertenencia



- Aspecto más importante vinculado a “safe memory”
- Junto al Préstamo y al Tiempo de vida aseguran dicha seguridad

Pertenencia

1. Cada valor en Rust tiene una variable que es su *owner*.

2. Solo puede haber un *owner* a la vez.

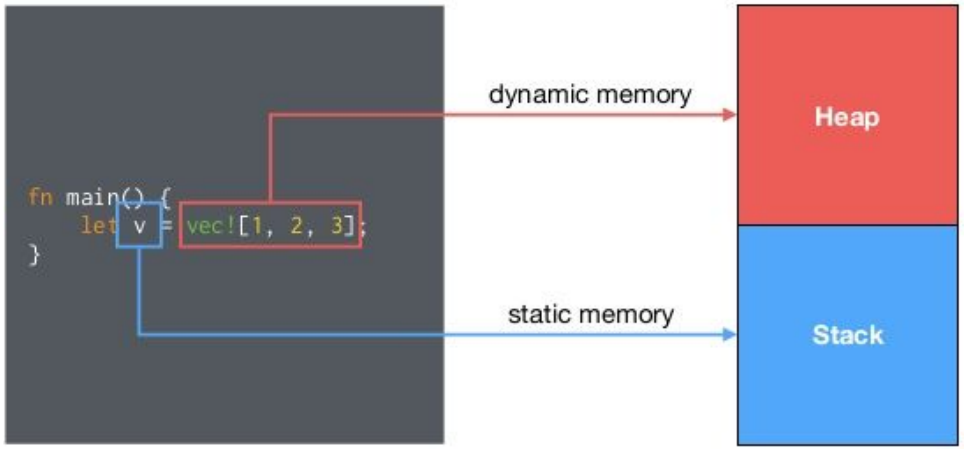
3. Cuando el *owner* sale del alcance, el valor se descarta.

Pertenencia

Variables y memoria

- Cuando una variable es declarada, Rust asigna memoria en el ***stack*** y el ***heap***

- Cuando el dueño de los recursos es destruido, todos estos recursos serán liberados



Pertenencia

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

    //let v2 = v;

    println!("v[0] es : {}", v[0]);
}
```

Pertenencia

El trait Copy y la Pertenencia

```
fn main(){
    let   v   =   1;
    let   v2   =   v;
    println!("v   es:   {}",  v);
}
```

Préstamo

- Se pasan referencias en vez de argumentos, para prestar la pertenencia
- No se toma la referencia al recurso, se la toma prestada

Préstamo

Reglas

1- Cualquier préstamo debe vivir en un ámbito no mayor al del owner

2- Se puede tener un solo tipo tipo de préstamo a la vez:
- una o más referencias (&T) a un recurso
- exactamente una referencia mutable (&mut T)

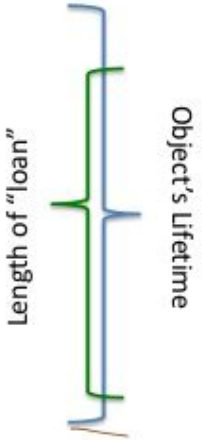
Préstamo

El préstamo previene:

- Invalidación de iteradores
- *Use after free*

Tiempo de Vida

We cannot **borrow** an object for longer than that object may live!



El tiempo de vida describe el ámbito en el cual una referencia es válida

Tiempo de Vida

```
fn main(){
        //Implícito
        fn foo1(x: &i32){...}

        //Explicito
        fn foo2<'y>(x: &'y i32){...}

        //Explicito y con referencia
        fn foo3<'z>(x: &'z mut i32){...}
}
```

Mutabilidad

- Enlace a variable mutable
- Enlace a variable inmutable, a una referencia mutable

Mutabilidad

Mutabilidad Interior vs Mutabilidad Exterior

```
use std::cell::RefCell;
```

```
use std::sync::Arc;
```

Mutabilidad

Mutabilidad a nivel de campos

La mutabilidad es una propiedad de un préstamo (&mut) o un enlace a variable (let mut).

Manejo de Errores

- Falla vs Pánico
- Try!

Smart Pointers

Son estructuras de datos que no solo hacen de puntero, sinó también tiene metadata adicional y capacidades adicionales

Los más comunes:

- Box<T>
- Rc<T>
- Ref<T> y RefMut<T>

Objetos en Rust

¿Es RUST Orientado a Objetos?

Objetos en Rust: Definición

"Object-oriented programs are made up of objects. An *object* packages both data and the procedures that operate on that data. The procedures are typically called *methods* or *operations*."

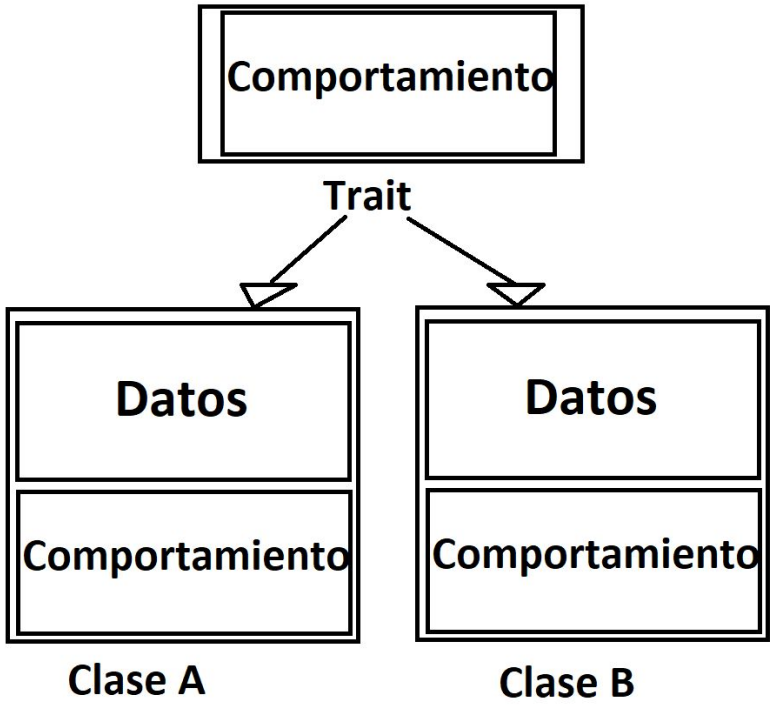
Design Patterns: Elements of Reusable Object-Oriented Software by Enoch Gamma, Richard Helm, Ralph Johnson, and John Vlissides

Objetos en Rust: Implementación

Objetos en Rust: Encapsulamiento

```
impl Button {
    pub fn draw(&self) {
    }
}
```

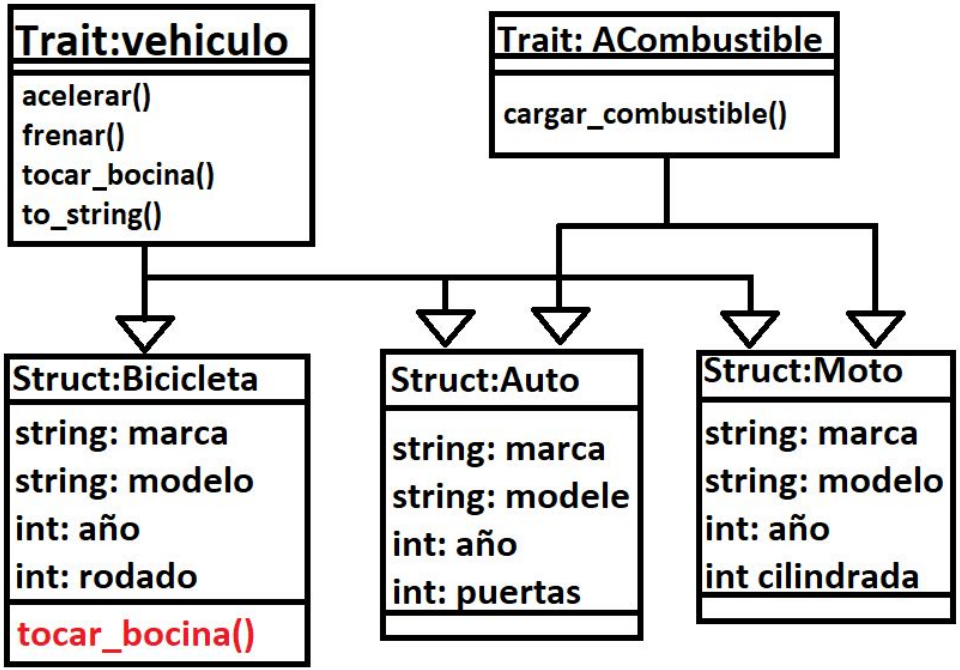
Objetos en Rust: Herencia



Objetos en Rust: Polimorfismo

Rust permite polimorfismo siempre que los objetos a los que se hace referencia implementen el mismo trait.

Objetos en Rust: Ejemplo



Características Avanzadas

Rust “Inseguro”

Rust Inseguro: Características

- Se desactivan garantías del compilador
- Segundo lenguaje con “superpoderes”
- Acceder a las capas bajas del hardware

Rust Inseguro: Superpoderes

- Acceder o modificar una variable global
- Desreferenciar “raw pointers”
- Llamar a una función o método inseguro
- Implementar un trait inseguro

Rust Inseguro: Variables Static (Globales)

```
static mut COUNTER: u32 = 0;

fn main() {
    unsafe {
        COUNTER += 3;

        println!("COUNTER: {}", COUNTER);
    }
}
```

Rust Inseguro: Raw Pointers

```
let mut num = 5;

let r1 = &num as *const i32;
let r2 = &mut num as *mut i32;

unsafe {
    println!("r1 is: {}", *r1);
    println!("r2 is: {}", *r2);
}
```

Rust Inseguro: Funciones Inseguras

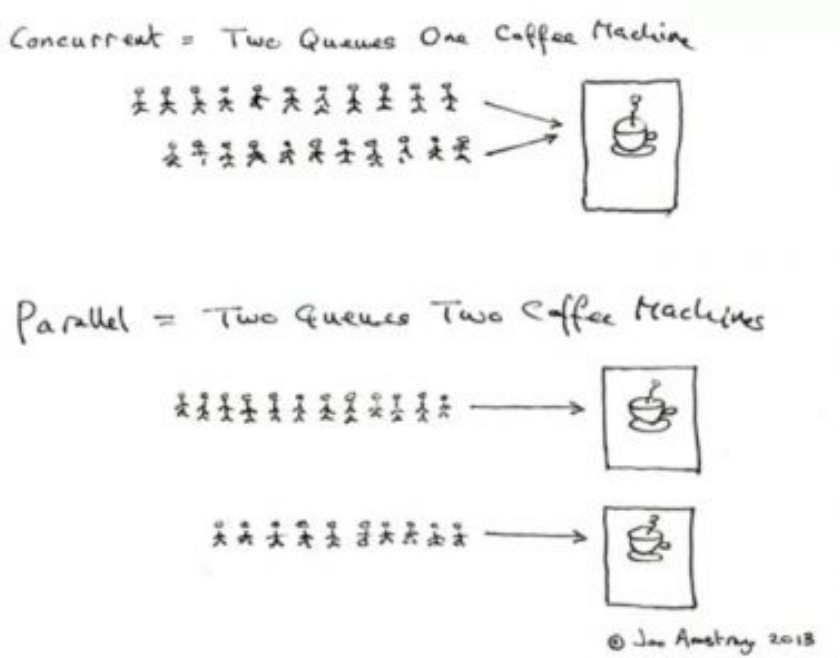
```
extern {
    fn c_function();
}
fn main() {
    println!("Calling c function");
    unsafe {
        c_function();
    }
}
```

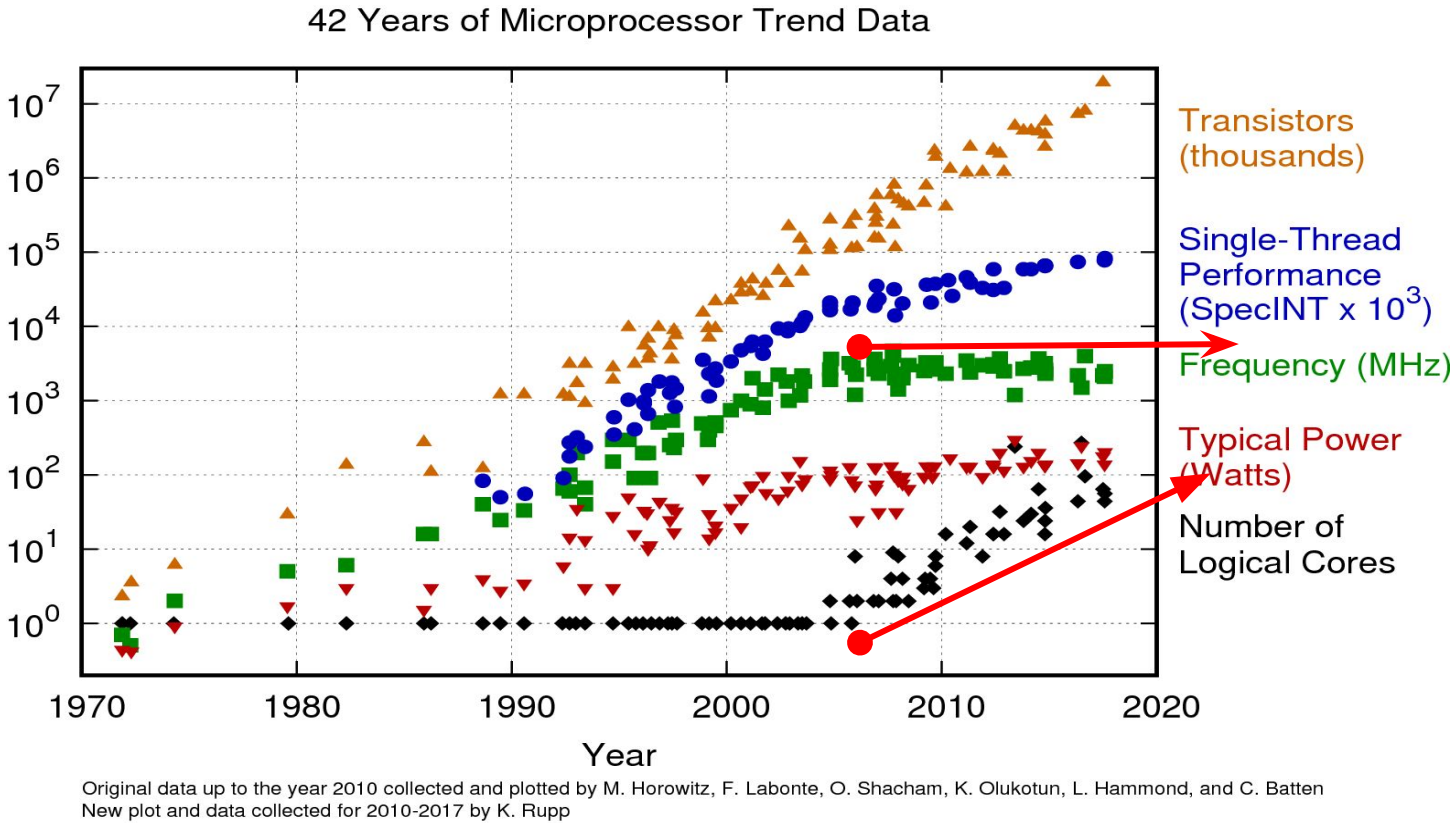
Rust Inseguro: Traits Inseguros

```
unsafe trait Foo {
    // methods go here
}

unsafe impl Foo for i32 {
    // method implementations go here
}
```

Concurrencia

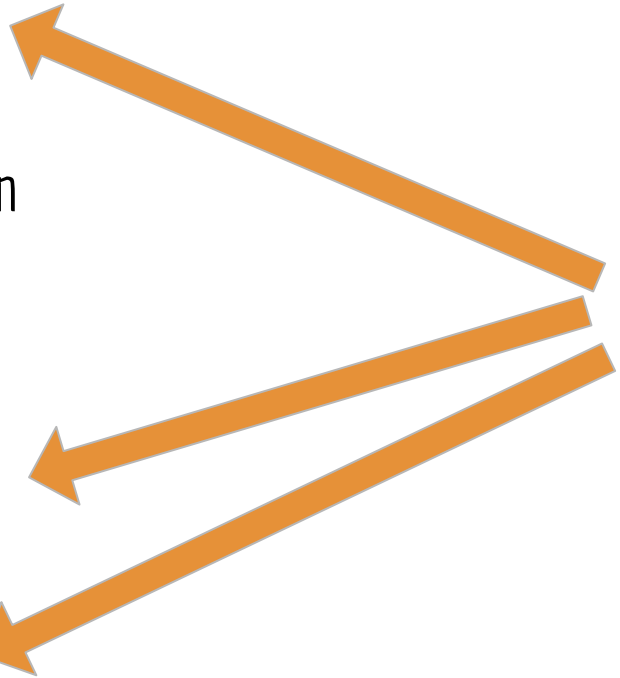




https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/

Usar Concurrencia es difícil...

- Data races
- Race condition
- Deadlocks
- Use after free
- Double free



Explotables!

The Rust Programming Language Blog

Fearless Concurrency with Rust

Apr 10, 2015 • Aaron Turon

The Rust project was initiated to solve two thorny problems:

- How do you do safe systems programming?
- How do you make concurrency painless?

Initially these problems seemed orthogonal, but to our amazement, the solution turned out to be identical: **the same tools that make Rust safe also help you tackle concurrency head-on**.

https://blog.rust-lang.org/2015/04/10/Fearless-Concurrency.html

Threads

- Rust usa modelo **1:1**, menos overhead que el modelo **M:N**.
- Crear con **thread::spawn** pasando como parámetro un *closure* con el código a ejecutar.
- **move** ⟶ pasar ownership de los datos del thread principal al nuevo thread.

Channels

- Pasar mensajes entre threads.
- *Multiple Producer, Single Consumer*
- **mpsc::channel** devuelve una tupla (**sender**,**receiver**). El **sender** puede clonarse.
- **receiver** recibe valores hasta que se cierre el canal.
- Datos enviados pasan a ser propiedad del **receiver**.

Memoria Compartida

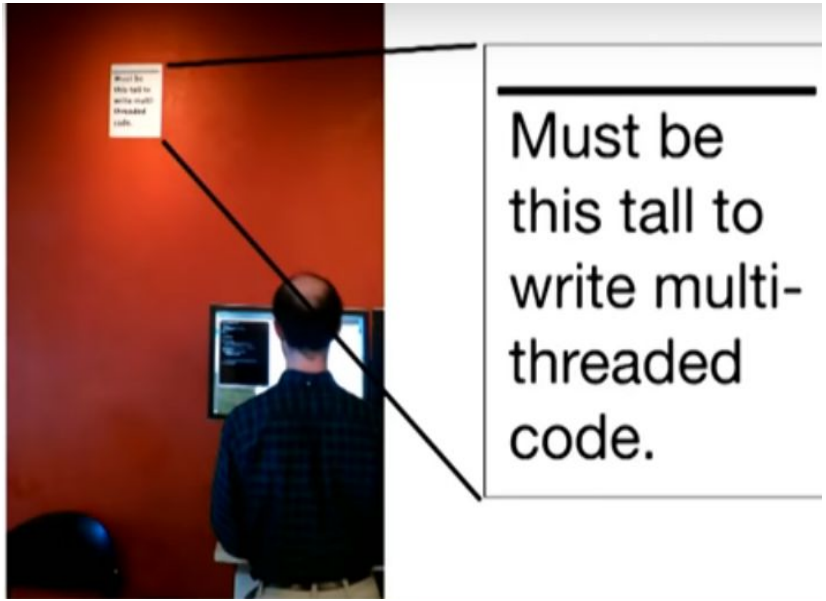
Shared-State Concurrency

- **Mutex** ⟶ acceso a datos por un sólo thread a la vez.
- Acceder a los datos ⟶ **lock**.
- Al salir de scope, el **mutex** se libera automáticamente.
- **Atomic Referenc counter (ARC)** ⟶ compartir un **mutex** entre varios threads (thread safe)

Proyecto Integrador

Multithreaded Web Server

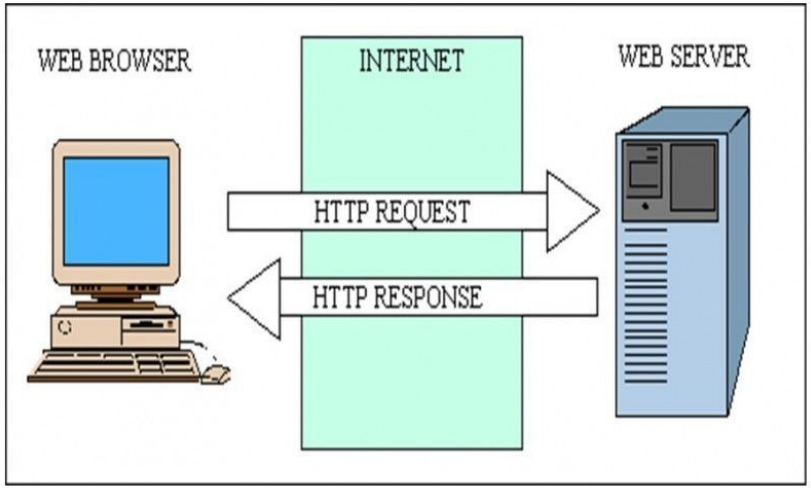
Mulithreaded Web Server



RUST-PLAN:

1. Iniciar un proyecto en Rust.
2. Escuchar conexiones TCP de un socket.
3. Parsear algunas HTTP request.
4. Crear un HTTP response.
5. Mejorar el rendimiento del server.

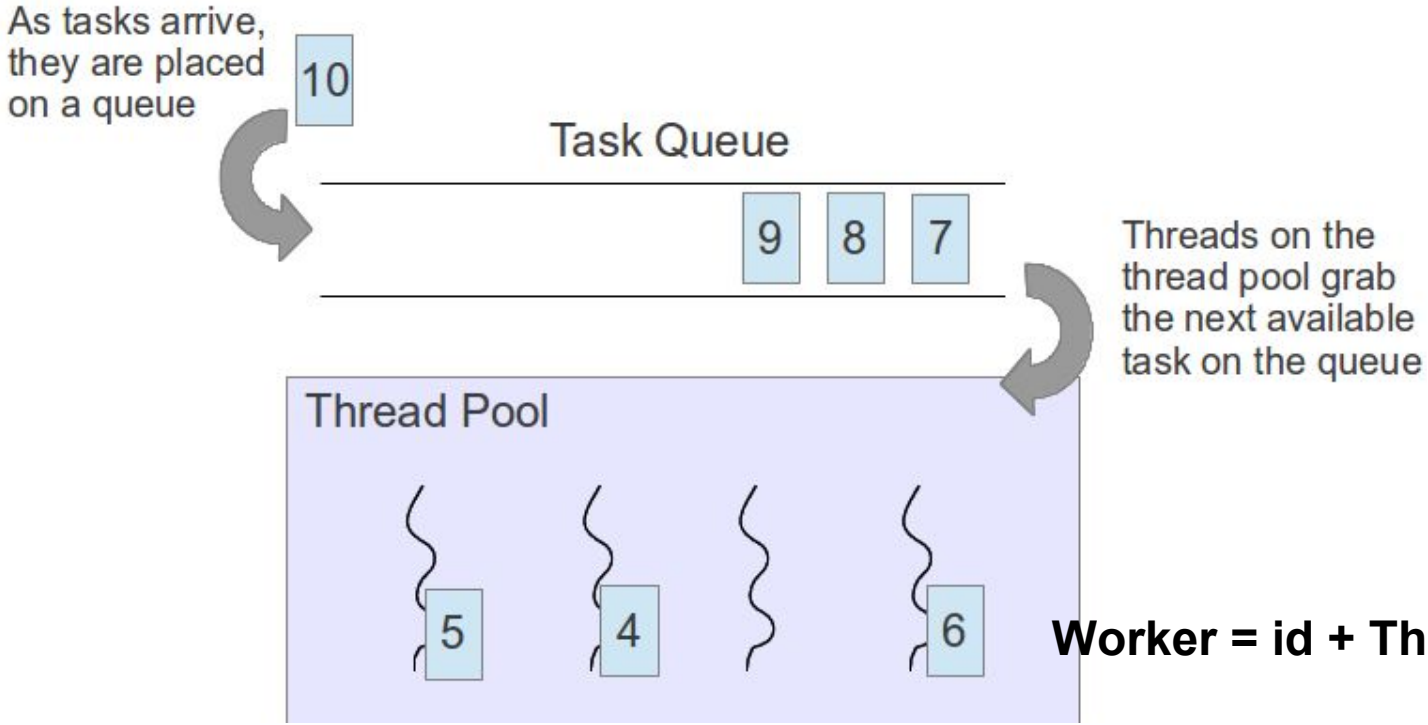
Single Thread Web Server



1. Establecer la conexión **TCP.**
2. Procesar Single **request** por vez.
3. Crear un **Thread** por **request recibido.**
4. Thread Pool?


WHO THE FUCK
IS THREADPOOL?
memegenerator.net

Thread Pool



Worker = id + Thread

RUST en la Actualidad

Rust: en qué es bueno

- Concurrencia/paralelismo
- Sistemas operativos
- Software a bajo nivel
- Interoperabilidad con C

Rust: en qué no es bueno

- Interfaces gráficas
- Interconexión con librerías de C++
- POO

Rust en Empresas I

- **Amazon** - Creando herramientas en Rust.
- **Atlassian** - Usan Rust de backend.
- **Dropbox** - Usan Rust en frontend y backend.
- **Facebook** - Herramientas para source control.
- **Google** - Parte del Fuchsia project.
- **Microsoft** - Usan Rust en Azure IoT.

Rust en Empresas II

- **npm** - Crearon npm core services.
- **Red Hat** - Crearon sistema de almacenamiento.
- **Reddit** - Usan rust en los comentarios
- **Twitter** - Como parte del build team support.
- y más

Rust en los Jueguito'

- **Chucklefish Games** - Usan Rust en varios juegos en desarrollo.
- **Electronic Arts** - Usan Rust en SEED.
- **Frostbite Engine** - Usan Rust en procesamiento de backend.
- **Ready at Dawn Studios** -Todos sus desarrollos serán en Rust.
- **Unity** - Usan Rust en data engineering.
- **Emuladores** - Rustation y Rustendo64

Negocios & RUST

CLOUDFLARE

Dropbox

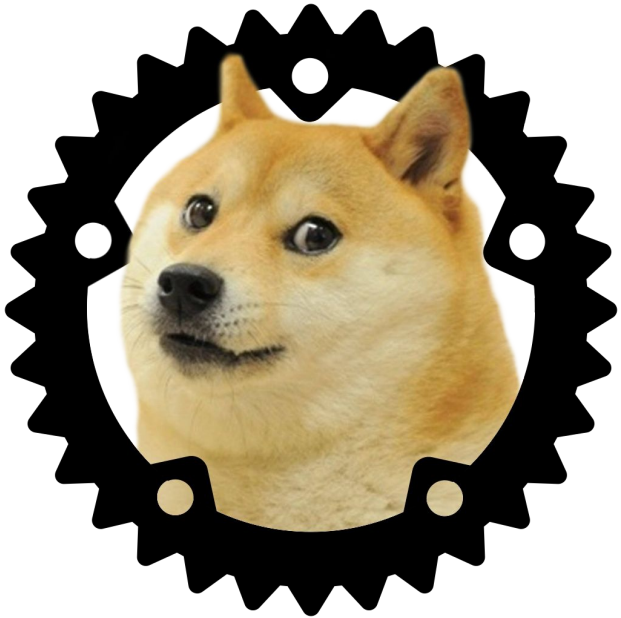
CHEF

SAMSUNG SmartThings™

ATLASSIAN

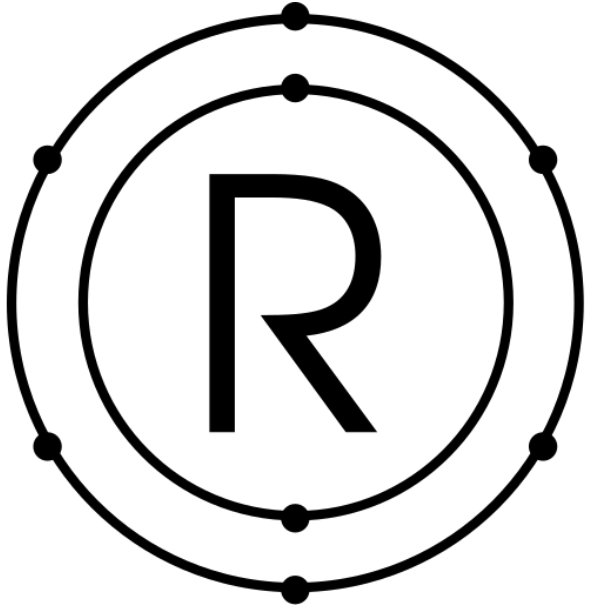
Rust: Servo

Motor de renderizado experimental de mozilla que sucederá al actual Gecko



Rust: Redox

Sistema operativo basado en unix con enfoque en la seguridad y en el rendimiento



Benchmarking

The Computer Language Benchmarks Game (originalmente llamado *The Great Computer Language Shootout*)

binary-trees, chameneos-redux, fannkuch-redux, fasta, k-nucleotide, mandelbrot, meteor-contest, n-body, pidigts, etc.

RUST Vs. C

reverse-complement

source	secs	mem	gz	cpu	cpu load
Rust	**1.60**	995,212	1376	2.73	24% 25% 96% 30%
C gcc	1.76	994,524	1438	3.98	46% 96% 43% 44%

k-nucleotide

source	secs	mem	gz	cpu	cpu load
Rust	**5.98**	137,956	1648	18.00	78% 49% 90% 85%
C gcc	6.27	130,024	1506	17.46	68% 47% 65% 100%

RUST Vs. C++

reverse-complement

source	secs	mem	gz	cpu	cpu load
Rust	**1.60**	995,212	1376	2.73	24% 25% 96% 30%
C++ g++	2.94	980,524	2280	4.53	15% 50% 52% 40%

pidigits

source	secs	mem	gz	cpu	cpu load
Rust	**1.74**	4,520	1366	1.74	1% 3% 0% 99%
C++ g++	1.89	4,312	513	1.88	1% 1% 99% 1%

RUST Vs. Go

regex-redux

source	secs	mem	gz	cpu	cpu load
Rust	**2.44**	194,804	765	3.87	85% 41% 20% 16%
Go	28.89	338,812	802	60.70	41% 51% 48% 70%

binary-trees

source	secs	mem	gz	cpu	cpu load
Rust	**4.14**	175,692	721	15.18	90% 90% 91% 100%
Go	28.56	466,636	654	109.05	96% 95% 96% 95%

RUST Vs. Java

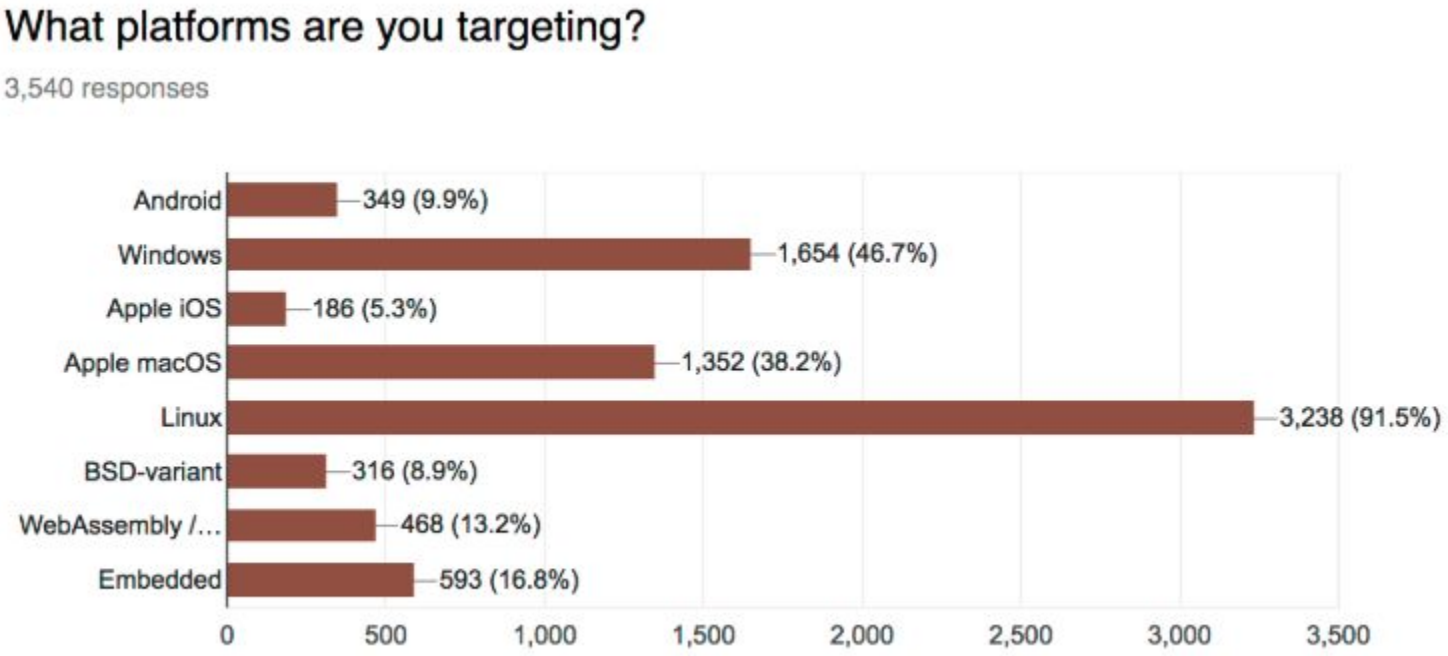
regex-redux

source	secs	mem	gz	cpu	cpu load
Rust	**2.44**	194,804	765	3.87	85% 41% 20% 16%
Java	10.52	637,380	929	31.89	75% 80% 77% 72%

mandelbrot

source	secs	mem	gz	cpu	cpu load
Rust	**1.74**	33,712	1332	6.86	98% 100% 98% 98%
Java	6.96	76,748	796	27.07	98% 98% 96% 98%

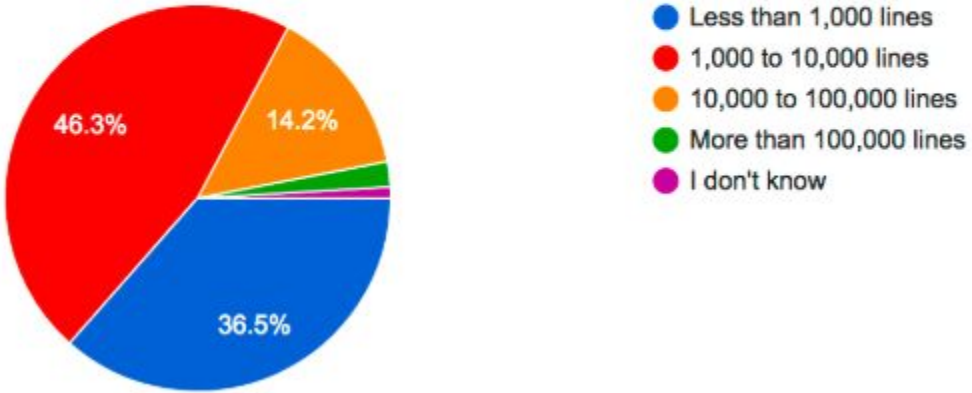
Estadística I



Estadística II

If you summed the size of all Rust projects you work on, how big would it be?

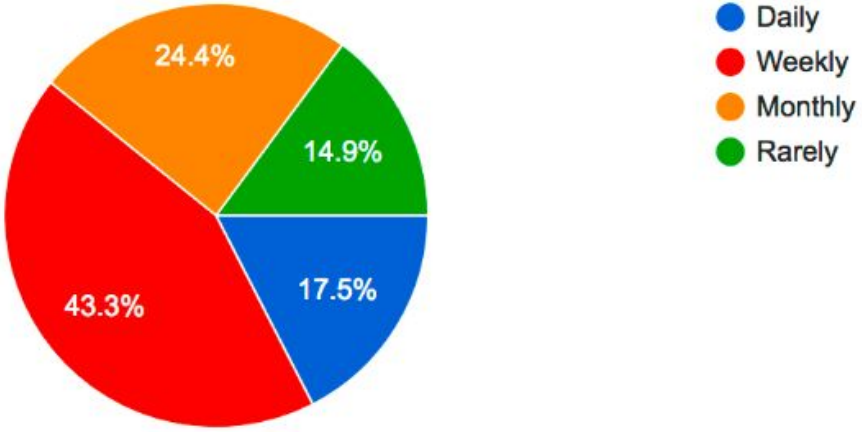
3,588 responses



Estadística III

How regularly do you work with Rust?

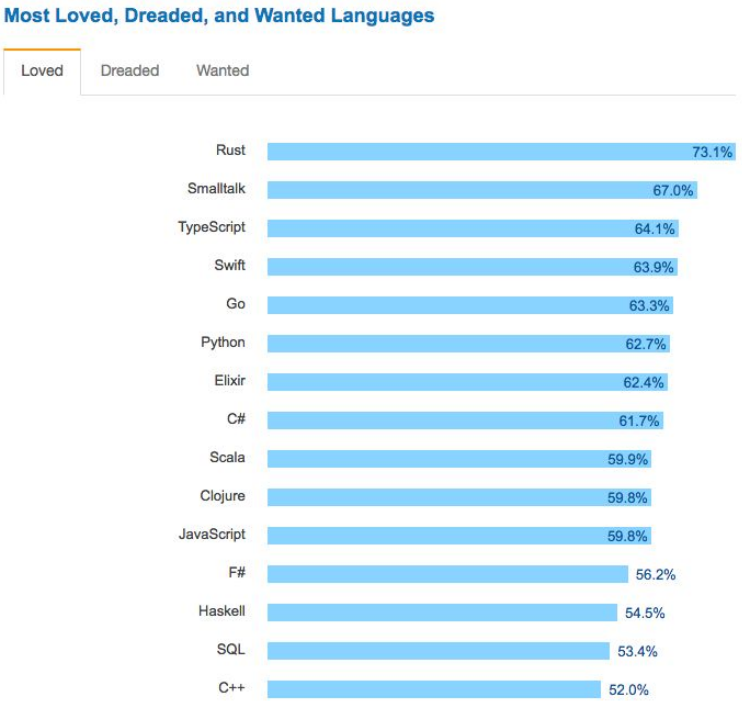
3,586 responses



Stack Overflow Survey

stackoverflow

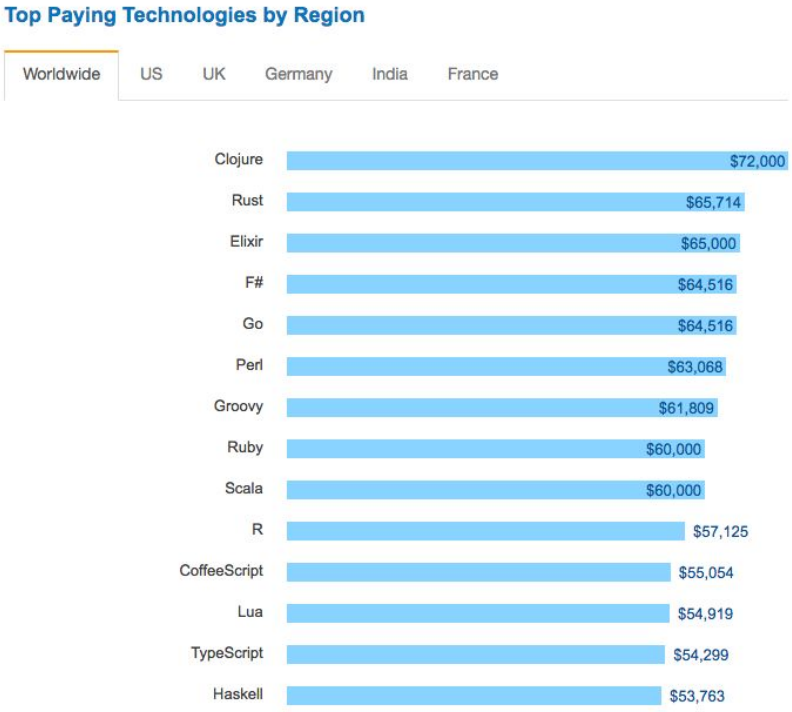
RUST resultó ser el lenguaje más “amado” por segundo año consecutivo...



Stack Overflow Survey

stackoverflow

Los programadores de RUST tienen el segundo salario más alto en promedio a nivel global...



This Week in Rust

06 NOV 2018

This Week in Rust 259

Hello and welcome to another issue of *This Week in Rust*! Rust is a systems language pursuing the trifecta: safety, concurrency, and speed. This is a weekly summary of its progress and community. Want something mentioned? Tweet us at @ThisWeekInRust or send us a pull request. Want to get involved? We love contributions.

This Week in Rust is openly developed on GitHub. If you find any errors in this week's issue, please submit a PR.

Updates from Rust Community

News & Blog Posts

- Rust now available on 14 Debian architectures.
- MIR-based borrowck is almost here.
- How to speed up the Rust compiler in 2018: NLL edition.

Summary

RUST

*An open-source systems language from Mozilla, emphasizing **safety**, **concurrency**, and **speed**.*

Bibliografía

http://intorust.com/
https://doc.rust-lang.org/rust-by-example/
https://doc.rust-lang.org/book/second-edition/
https://blog.rust-lang.org/
https://stevedonovan.github.io/rust-gentle-intro/
https://www.rust-lang.org/en-US/faq.html
https://this-week-in-rust.org/
https://insights.stackoverflow.com
https://www.ibm.com/developerworks
https://blog.rust-lang.org/2017/09/05/Rust-2017-Survey-Results.html

No es para ponernos tristes

Gracias Stan, pero con RUST esto no te pasaba.

¡Muchas Gracias!