Hello! Rust - An Introduction to safe systems Programming

PPT Credits: @dvigneshwer

About Me.

I am Mehul Patel

DevOps Engineer

moz://a

Reps Mentor, CAC

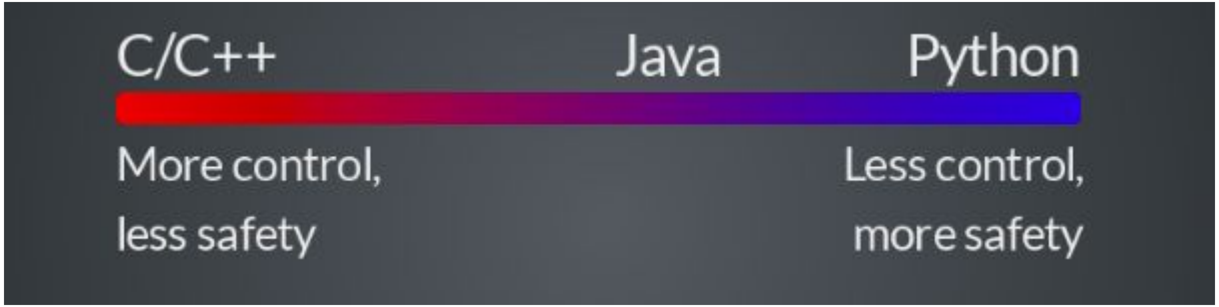
moz://a

moz://a

Let's hack into **Rust Programming**

What is Rust?

System programming language that has great **control** like C/C++, delivers **productivity** like in Python, and is super **safe**



What’s more about Rust?

(Baby don't hurt me, don't hurt me, no more)

- Rust is a new systems programming language designed for safety, concurrency, and speed.
- It was originally conceived by Graydon Hoare and is now developed by a team in Mozilla Research and the community.
- Multi-paradigm. Functional, imperative, object-oriented, whenever it makes sense.
- Low-level. Targets the same problem-space as C and C++
- Safe. Lovely, lovely types and pointer lifetimes guard against a lot of errors.

What is Rust?

“Systems programming
without fear

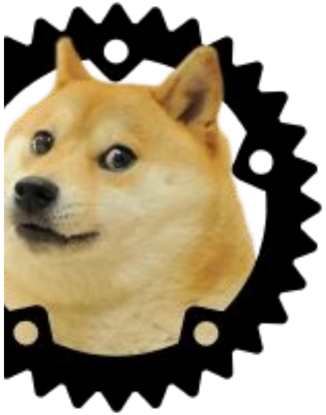
Firefox®

moz://a

Mission accomplished
Rust in Firefox 48

where are we with Rust?

R



2018 -19

Rust is the Most Loved Language by Developers

Friends of Rust

npm
coursera

Organizations running Rust in production.

(https://www.rust-lang.org/en-US/friends.html)

Common definition

Rust is a systems programming language that runs blazingly *fast*, prevents *segfaults*, and guarantees *thread safety*.

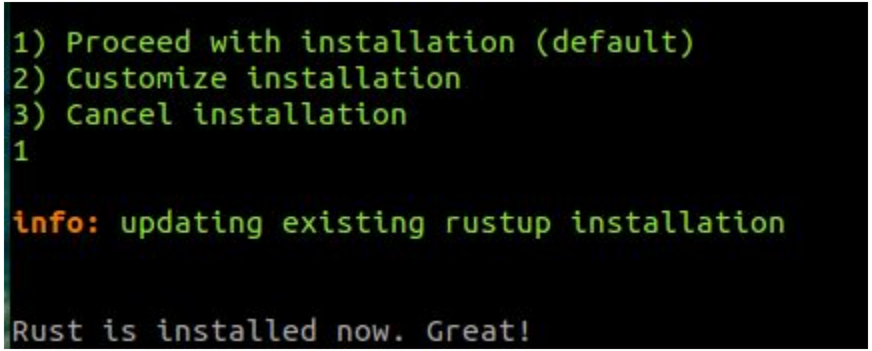
Why should I use Rust?

I have my own definition

Rust is a good choice when you'd choose C++. You can also say, "**Rust** is a systems programming language that pursuing the trifecta: *safe, concurrent, and fast*." I would say, Rust is an *ownership-oriented* programming language.

Installing Rust

Ubuntu / MacOS

- Open your terminal (Ctrl + Alt +T)
- **curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh**
- 

```
1) Proceed with installation (default)
2) Customize installation
3) Cancel installation
1

info: updating existing rustup installation


Rust is installed now. Great!
```

Installing Rust

rustc --version

cargo --version

Windows :

- Go to **https://win.rustup.rs/**
  - This will download **rustup-init.exe**
- Double click and start the installation

Features of rustup tool

-> Update to latest version:

```
rustup update stable
```

-> Update the rustup tool to the latest version

```
rustup self update
```

-> Install the nightly toolkit version of the Rust compiler:

```
rustup install nightly
```

-> Change the default version of the Rust compiler to nightly version:

```
rustup default nightly
```

Firstly, the reason that I’ve looked into Rust at first.

- Rust is new enough that you can write useful stuff that would have already existed in other languages
- It gives a relatively familiar tool to the modern C++ developers, but in the much more consistent and reliable ways.
- It is low-level enough that you take account of most resources.
- It's more like C++ and Go, less like Node and Ruby
- cargo is awesome. Managing crates just works as intended, which makes a whole lot of troubles you may have in other languages just vanish with a satisfying *poof*.

Why should one consider Rust?

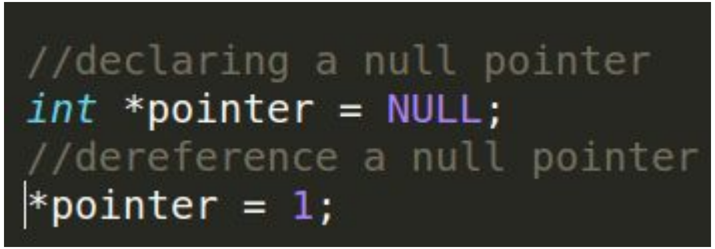
- State of art programming language
- Solves a lot of common system programming bugs
- Cargo : Rust Package manager
- Improving your toolkit
- Self learning
- It's FUN ...

Basic Terminologies

- ➔ Low and high level language
- ➔ System programming
- ➔ Stack and heap
- ➔ Concurrency and parallelism
- ➔ Compile time and run time
- ➔ Type system
- ➔ Garbage collector
- ➔ Mutability
- ➔ Scope

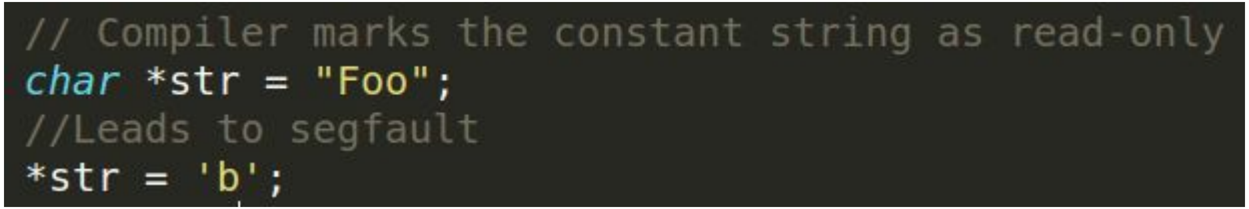
Segmentation Fault

- ➔ Dereference a null pointer



```
//declaring a null pointer
int *pointer = NULL;
//dereference a null pointer
*pointer = 1;
```

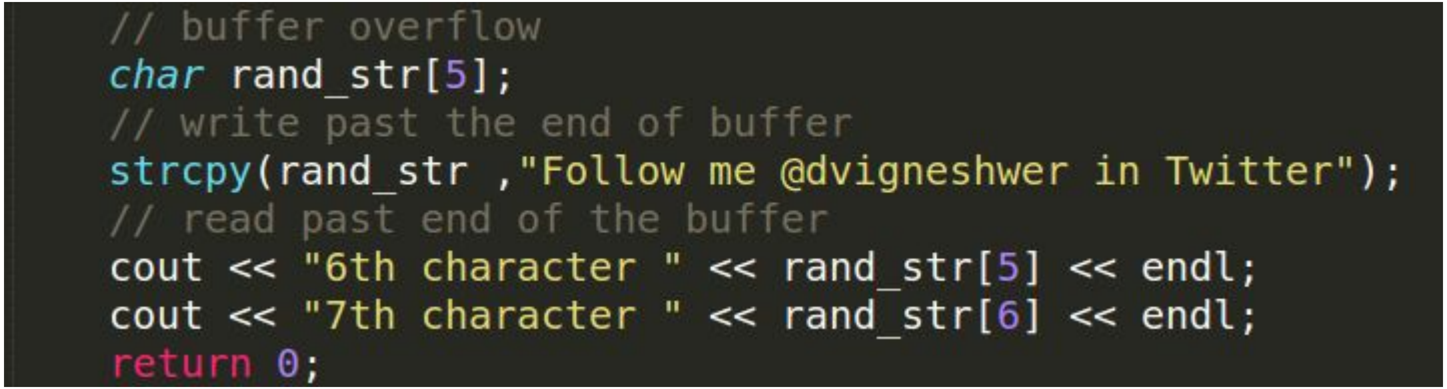
- ➔ Try to write to a portion of memory that was marked as read-only



```
// Compiler marks the constant string as read-only
char *str = "Foo";
//Leads to segfault
*str = 'b';
```

Buffer OverFlow

➔ Writing and reading the past end of buffer



```
// buffer overflow
char rand_str[5];
// write past the end of buffer
strcpy(rand_str ,"Follow me @dvigneshwer in Twitter");
// read past end of the buffer
cout << "6th character " << rand_str[5] << endl;
cout << "7th character " << rand_str[6] << endl;
return 0;
```

Hack without Fear

- Strong type system
  - Reduces a lot of common bugs
- Borrowing and Ownership
  - Memory safety
  - Freedom from data races
- Abstraction without overhead
- Stability without stagnation
- Libraries & tools ecosystem

Type System

Hello World

```
// Execution starts here

fn main() {

    let greet = “world”;

    println!("Hello {}!”, greet);

}
```

Variable Bindings

```
let x = 5;

let (x, y) = (1, 2); // patterns

let x: i32 = 5; // Type annotations

let x = 5; // By default, bindings are immutable.

x = 10;

let mut x = 5; // mut x: i32

x = 10;
```

Function in Rust

```rust
fn main() {
    print_sum(5, 6);
}
fn print_sum(x: i32, y: i32) {
    println!("sum is: {}", x + y);
}
```

Identify the error

```
fn main() {
    print_sum(5, 6);
}
fn print_sum(x , y ) {
    println!("sum is: {}", x + y);
}
```

Returning a value

```
fn add_one(x: i32) -> i32 {

    x + 1

}

fn add_one(x: i32) -> i32 {

    x + 1;

}
```

Expressions vs Statements

```
x = y = 5

let x = (let y = 5); // Expected identifier, found keyword `let`.
```

Rust : Expression -based language

Primitive Types

bool

```
let bool_val: bool = true;

println!("Bool value is {}", bool_val);

let bool_val: bool = false;
```

char

```
let x_char: char = 'a';

// Printing the character

println!("x char is {}", x_char);
```

i8/i16/i32/i64/isize

```
let num =10;
println!("Num is {}", num);
let age: i32 =40;
println!("Age is {}", age);
println!("Max i32 {}",i32::MAX);
println!("Max i32 {}",i32::MIN);
```

Arrays

```
let name: [type; size] = [elem1, elem2, elem3, elem4];

let array: [i32; 5] = [0, 1, 2, 3, 4];

let rand_array = [1,2,3]; // Defining an array

println!("random array {:?}",rand_array );

println!("random array 1st element {}",rand_array[0] ); // indexing starts with 0

println!("random array length {}",rand_array.len() );
```

Tuples

```
// Declaring a tuple
let rand_tuple = ("DevFest Siberia", 2017);
let rand_tuple2 : (&str, i8) = ("Viki",4);
// tuple operations
println!(" Name : {}", rand_tuple2.0);
println!(" Lucky no : {}", rand_tuple2.1);
```

slice

```
let array: [i32; 5] = [0, 1, 2, 3, 4];

println!("random array {:?}",&rand_array[0..3] ); // last three elements
```

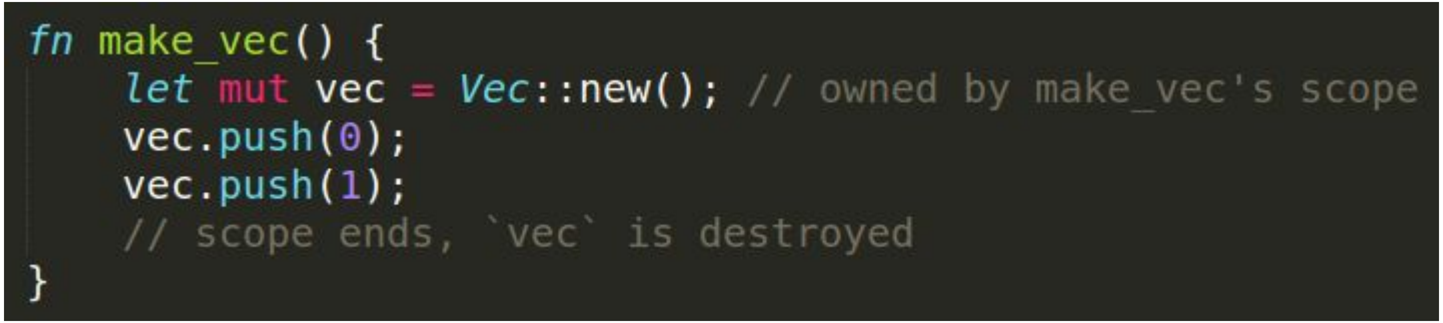
String

```
let rand_string = "Devfest Siberia 2017"; // declaring a random string

println!("length of the string is {}",rand_string.len() ); // printing the length of the string

let (first,second) = rand_string.split_at(7); // Splits in string

let count = rand_string.chars().count(); // Count using iterator count
```

```
println!(rand_string)
```

Ownership

In Rust, every value has an “**owning scope**” and passing or returning a value means transferring ownership (“moving” it) to a new scope



```
fn make_vec() {
    let mut vec = Vec::new(); // owned by make_vec's scope
    vec.push(0);
    vec.push(1);
    // scope ends, `vec` is destroyed
}
```

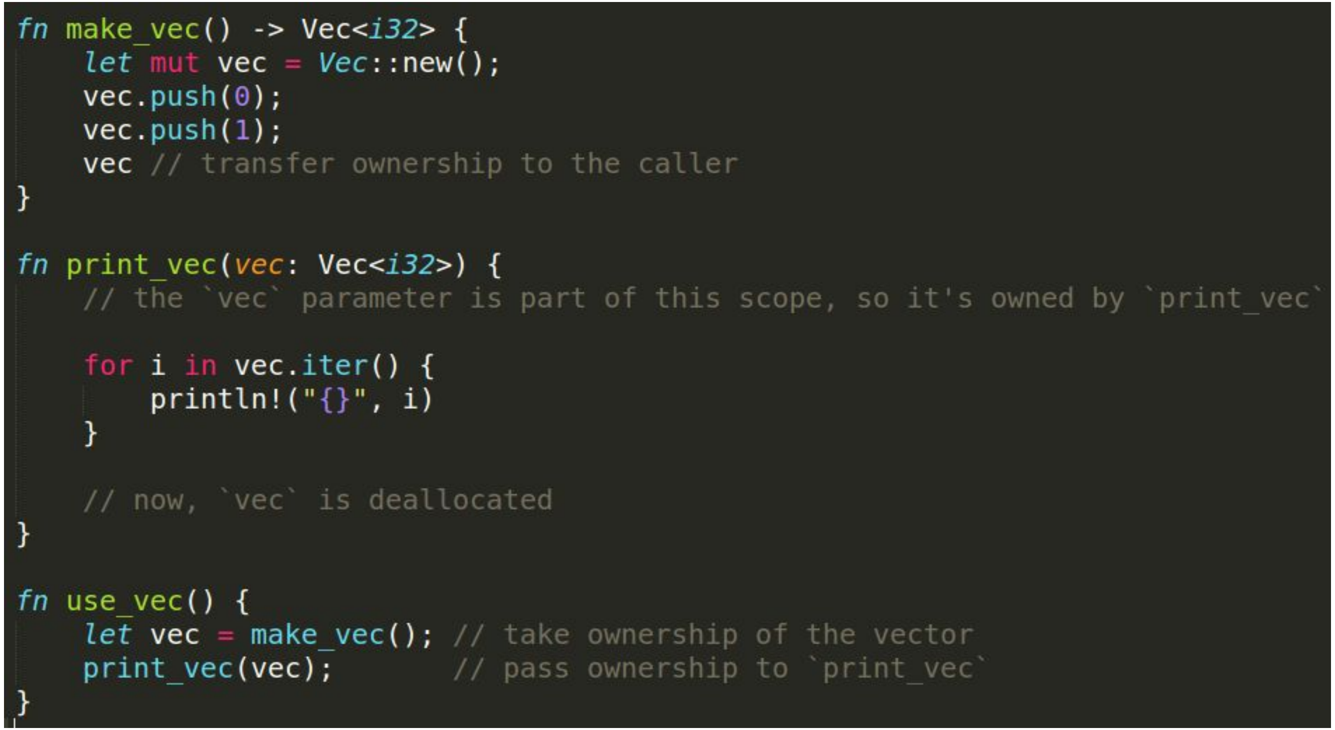
Example 1

```
fn foo{
    let v = vec![1,2,3];
    let x = v;
   println!("{:?}",v); // ERROR : use of moved value: "v"
}
```

Example 2

```
fn print(v : Vec<u32>) {
    println!(“{:?}”, v);
}
fn make_vec() {
    let v = vec![1,2,3];
    print(v);
    print(v); // ERROR : use of moved value: “v”
}
```

Example 3



```
fn make_vec() -> Vec<i32> {
    let mut vec = Vec::new();
    vec.push(0);
    vec.push(1);
    vec // transfer ownership to the caller
}

fn print_vec(vec: Vec<i32>) {
    // the `vec` parameter is part of this scope, so it's owned by `print_vec`

    for i in vec.iter() {
        println!("{}", i)
    }

    // now, `vec` is deallocated
}

fn use_vec() {
    let vec = make_vec(); // take ownership of the vector
    print_vec(vec);       // pass ownership to `print_vec`
}
```

Aliasing

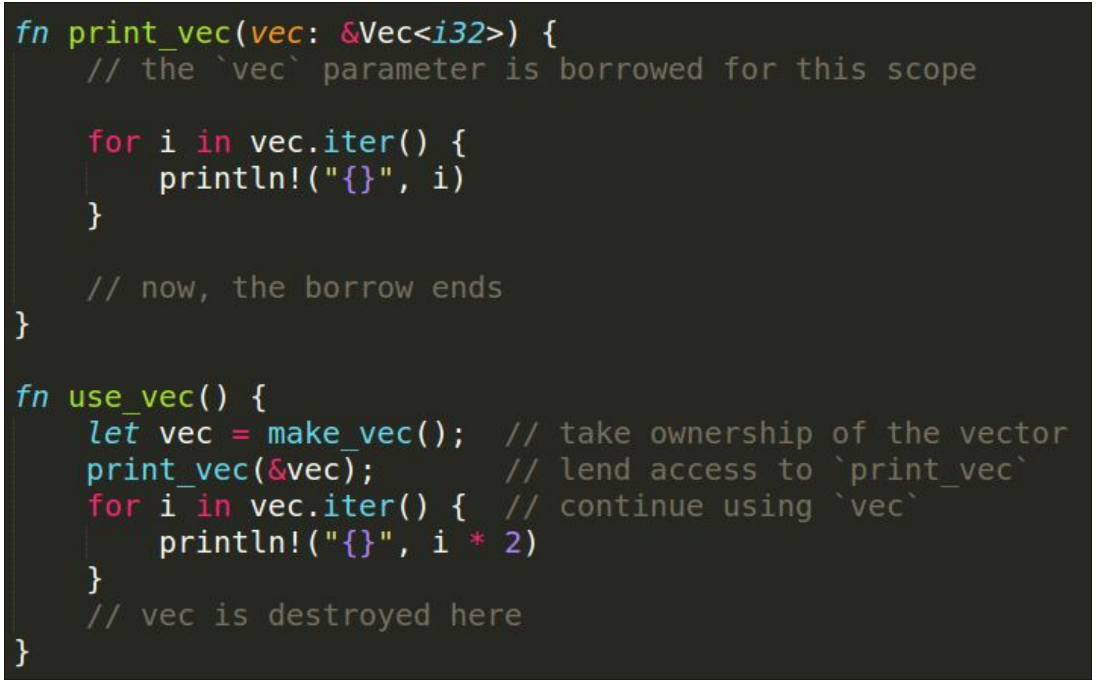
Aliasing -> More than one pointer to the same memory

The key problem to most memory problems out there is when mutation and aliasing both happens at the same time.

Ownership concepts avoids Aliasing

Borrowing

If you have access to a value in Rust, you can lend out that access to the functions you call



```rust
fn print_vec(vec: &Vec<i32>) {
    // the `vec` parameter is borrowed for this scope

    for i in vec.iter() {
        println!("{}", i)
    }

    // now, the borrow ends
}

fn use_vec() {
    let vec = make_vec();  // take ownership of the vector
    print_vec(&vec);       // lend access to `print_vec`
    for i in vec.iter() {  // continue using `vec`
        println!("{}", i * 2)
    }
    // vec is destroyed here
}
```

Types of Borrowing

There is two type of borrowing in Rust, both the cases aliasing and mutation do not happen simultaneously

- Shared Borrowing (&T)
- Mutable Borrow (&mut T)

&mut T

```
fn  add_one(v: &mut Vec<u32> ) {
        v.push(1)
}
fn foo() {
let mut v = Vec![1,2,3];
add_one(&mut v);
}
```

Lifetimes

```
let outer;
{
    let v = 1;
    outer = &v; // ERROR: ‘v’ doesn’t live long
}
println!(“{}”, outer);
```

Mutability Rules

All variables are immutable by default

Only one mutable reference at a time

But as many immutable &'s as you want

Mutable references block all other access

The &mut must go out of scope before using other &'s

A bit complex example

```
fn avg(list: &[f64]) -> f64 {
    let mut total = 0;
    for el in list{
        total += *el;
    }
    total/list.len() as f64
}
```

HLL version

```
fn avg(list: &[f64]) -> f64 {
    list.iter().sum::<f64>() / list.len() as f64
}
```

Parallel Version (Rayon)

```
fn avg(list: &[f64]) -> f64 {
        list.par_iter().sum::<f64>() / list.len() as f64
}
```

Demos

- ➔ Vectors, Pointers, closures, typecasting
- ➔ Complex Data Structures : Structs, enum, impl , trait
- ➔ Decision making and looping statements
- ➔ Crates and Modules
- ➔ Cargo features
- ➔ Introduction to Rust library ecosystem
- ➔ Error handling
- ➔ Understanding Macros

What Next ?

Rust Hacks

Rust Programming Language account for web developers, Community Evangelist.

Rust Hacks

Follow us on **Twitter**:

@rusthack

https://twitter.com/rusthack

Join the **Telegram** Channel:

@rusthacks

https://t.me/rusthacks

Contribute to #RustHacks on **GitHub** :

@rusthacks

https://github.com/rusthacks

Join the **Facebook** Group :

https://www.facebook.com/groups/rusthacks/

Rust Trailheads

Follow us on Twitter:

@rustlang

@ThisWeekInRust

Join Rust IRC Channels:

#rust

#rust-community

#rust-machine-learning

#tensorflow-rust

Join Rust Websites:

rust-lang.org

rustbyexample.com

rustaceans.org

reddit.com/r/rust/

Thanks!

Twitter @rowdymehul

Instagram @rowdymehul

Telegram @rowdymehul

Email iamrowdymehul@gmail.com

Website rowdymehul.com