# Rust Programming Language

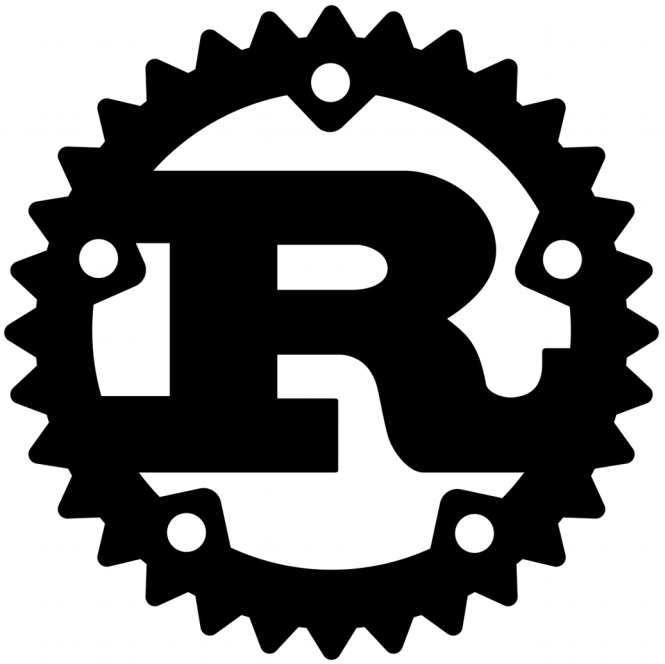

استاد:

**دکتر صادق سلیمانی**

ارائه دهندگان:

**یاسین امینی و مظهر زندسلیمی**


۱۳۹۷.۲.۲۳



# عناوین

- تاریخچه
- معرفی
- کاربرد
- جزئیات برنامه نویسی با RUST
- مقایسه با سایر زبان‌ها
- نمونه برنامه

# تاریخچه

- آغاز در سال ۲۰۰۶ توسط آقای Graydon Hoare
- مطرح شدن در شرکت موزیلا در سال ۲۰۰۹
- کامپایلر اولیه با زبان Ocaml
- انتشار نسخه اولیه آن در سال ۲۰۱۱

# معرفی

- توسعه توسط انجمن Open Source
- زبانی که توسط خود کامپایل می‌شود (Bootstrapping)
- حمایت‌کنندگان اصلی: موزیلا و سامسونگ
- انقلابی در زبان های سیستمی
- مکمل کمبود های C و C++

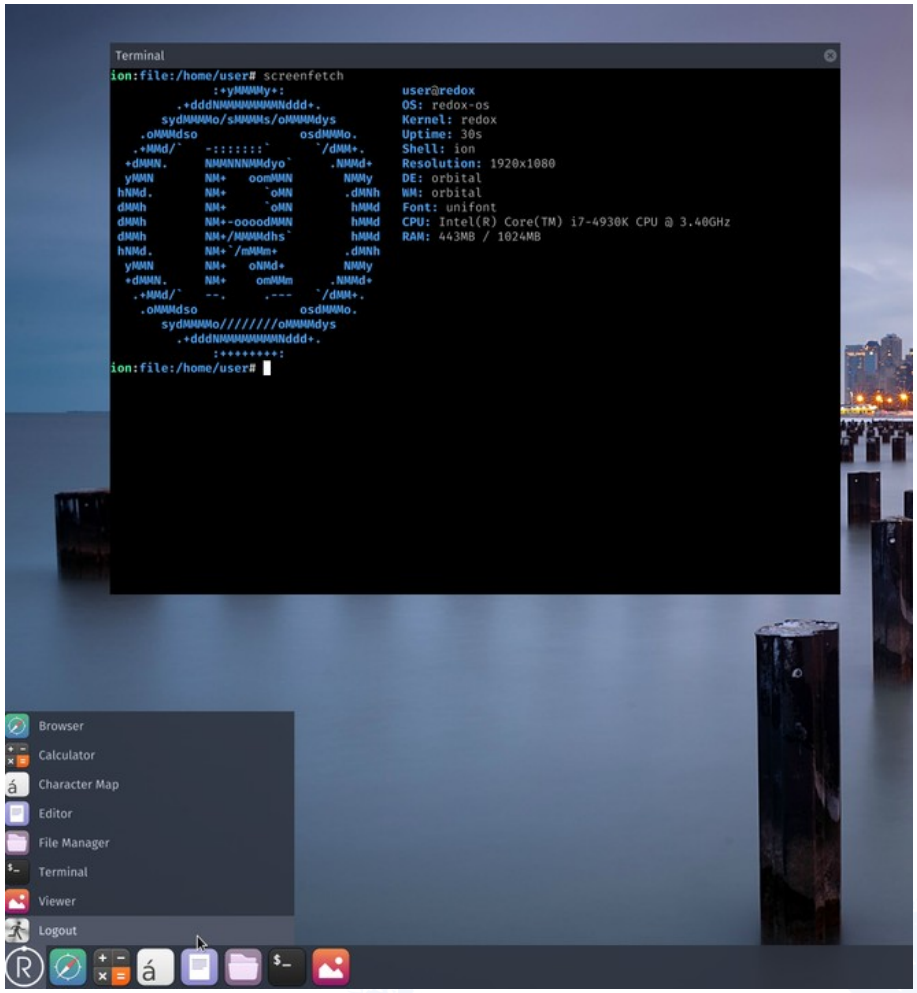
# معرفی

- محبوب ترین زبان برای یادگیری طبق نظرسنجی Stackoverflow
- مدیریت حافظه‌ی نو
- سرعت بالا
- مناسب برای برنامه نویسی Concurrent
- مناسب برای پردازش موازی در اینترنت اشیاء
- امنیت بالا

# کاربرد

- مرورگر:
  - « Servo
  - « Quantum (firefox)
- Cargo
  - « مدیر بسته‌ی زبان برنامه نویسی RUST
- Habitat
  - « توضیح اینجا قرار می‌گیرد
- Grin
  - « یک کریپتوکارنسی بر بستر بلاکچین

# کاربرد




- Magic Packet

  - « سیستم فایل pbox

- Redox OS
  - « یک سیستم عامل ریز هسته
- Piston
  - « موتور بازی
- Stratis
  - « یک سیستم فایل برای Fedora 28 طراحی شده است

# کاربرد

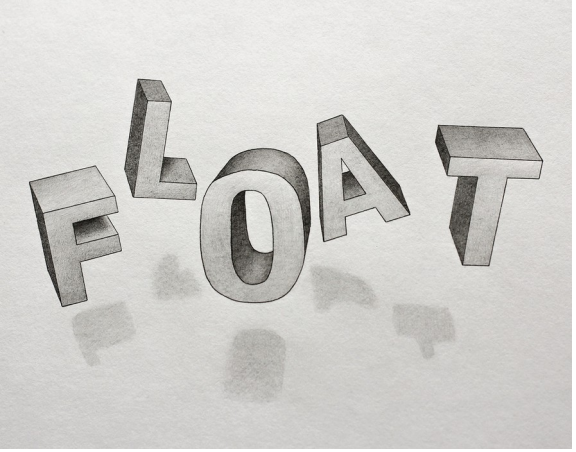
- Tock
  - یک سیستم عامل تعبیه شده
- XI
  - یک ویرایشگر متن پیشرفته از شرکت گوگل
- Lucidscape Mesh
  - یک موتور شبیه سازی شده در زمان واقعی برای واقعیت مجازی
- WitchBrook
  - بازی کامپیوتری

# جزئیات برنامه نویسی RUST

- داده
- عملیات اصلی
- کنترل ترتیب
- کنترل داده
- مدیریت حافظه
- محیط عملیاتی


# داده (اولیه)

- نوع اسکالر
  - Integer , floating point , boolean , character
- اعداد صحیح
  - Attribute: i8 , i16 , i32 , i64 , u8 , u16 , u32 , u64
  - Operation: + , - , * , / , % , = , < , <= , == , !=
    - let mut x:i8 = 256;
  - Value: **arch ->** isize , usize
    - وابسته به معماری سخت افزار ( ۶۴ یا ۳۲ بیت بودن )
  - Storage representation
  - Algorithm




# داده (اولیه)

- **نوع اسکالر**
- اعداد اعشاری

» Attribute: f32 , f64

» Value:

« دقت وابسته به نوع ، سرعت برابر

» Operation: + , - , * , / , = , < , <= , == , !=

› let y:f32 = 3.14;

» Storage representation → IEEE-754

» Algorithm → software


# داده (اولیه)

- نوع اسکالر
- بولی

» Attribute: bool

» Value: true , false

» Operation: && , || , & , | , ! , == , != , =

› let z:bool = false;

» Storage representation → one Byte (bit cannot be addressed)

» Algorithm → hardware

# داده (اولیه)

- نوع اسکالر
- کراکتر

» Attribute: char

» Value:

› U+000 – U+D7FF (Exclusive)

› U+E000 – U+10FFFF (NonExclusive – chinese)

» Operation: = , is_digit() , is_alphabetic() , is_alphanumeric()

› let ch:char = '☺';

» Storage representation → 1 to 4 Byte (unicode)

» Algorithm → hardware

# داده (اولیه)

## Enumeration –

- Attribute: enum
- Value:
  - توسط برنامه نویس تعیین می‌شود.
- Operation: = , :: (accessing) , successor , predecessor , comparison
  - **Enum IpVersion {**
    **v4,**
    **v6,**
    **}**
  - let four = IpVersion::V4;


# داده (ساختار یافته)

- آرایه
  - » متجانس – اندازه‌ی ثابت – سازمان خطی بین اجزاء
  - » استفاده از حافظه‌ی stack به جای heap
- » let a = [1,2,3,4,5];
- » Accessing and Selection: A[0] = 2;
- » Storage representation → sequential
- » Algorithm → software
- آرایه با اندازه‌ی متغیر –> vector


# داده (ساختار یافته)

- تاپل (Tuple)
  - نامتجانس – اندازه متغیر – سازمان خطی بین اجزاء
- let tup: (i32, f64, u8, char) = (-12, 3.14 ,2, '☺');
- Accessing and Selection: tup.0 → -12
  - امکان return کردن چندین مقدار در توابع
  - مقدار دهی چندگانه
- let tup = (500, 6.4, 1);

  let (x, y, z) = tup;

# عملیات اصلی

- + → add , concat , or
- - → negation , subtraction
- * → multiplication , pointer reference
- /
- %
- ! → not , macro definition
- && , ||
- & → Borrow , bitwise and  &= → bit AND assign
- | → bitwise or , pattern alternatives  |= → bit OR assign

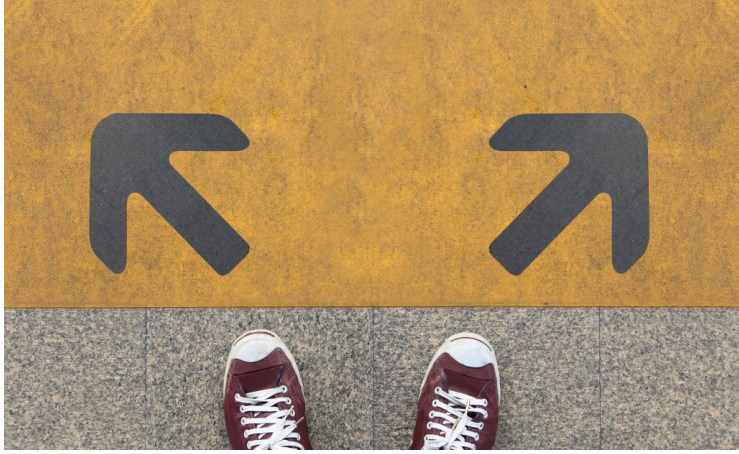
# عملیات اصلی

- : → loop label , variable type assign
- :: → object definition
- → → function return
- => → separate condition and execute
- _ → default in pattern , (literal integer separator 1_000)
- . → member access
- .. → subrange
- ... → subrange ( includes the higher bound )

# عملیات اصلی

- > , < , >= , <= , != , == , += , -= , *= , /= , %= , ;
- << → shift left  <<= → shift left and assign
- @ → pattern binding
- ^ → bit XOR  ^= → bit XOR and assign
- ? → error propogation

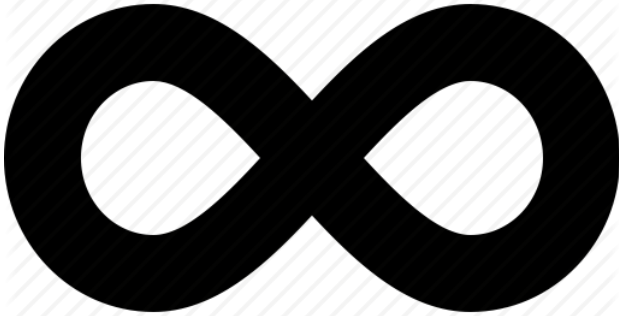
# کنترل ترتیب



- If expression

```
fn main() {
    let number = 3;
    if number < 5 {
        println!("condition was true");
    } else {
        println!("condition was false");
    }
}
```

# کنترل ترتیب



- Loop expression

```
fn main() {
    loop {
        println!("again!");
    }
} →  while True  ( exit by break )
```

# کنترل ترتیب


- While expression

```
fn main() {
    let mut number = 3;
    while number != 0 {
        println!("{}!" , number);
        number = number – 1;
    }
}
```

# کنترل ترتیب

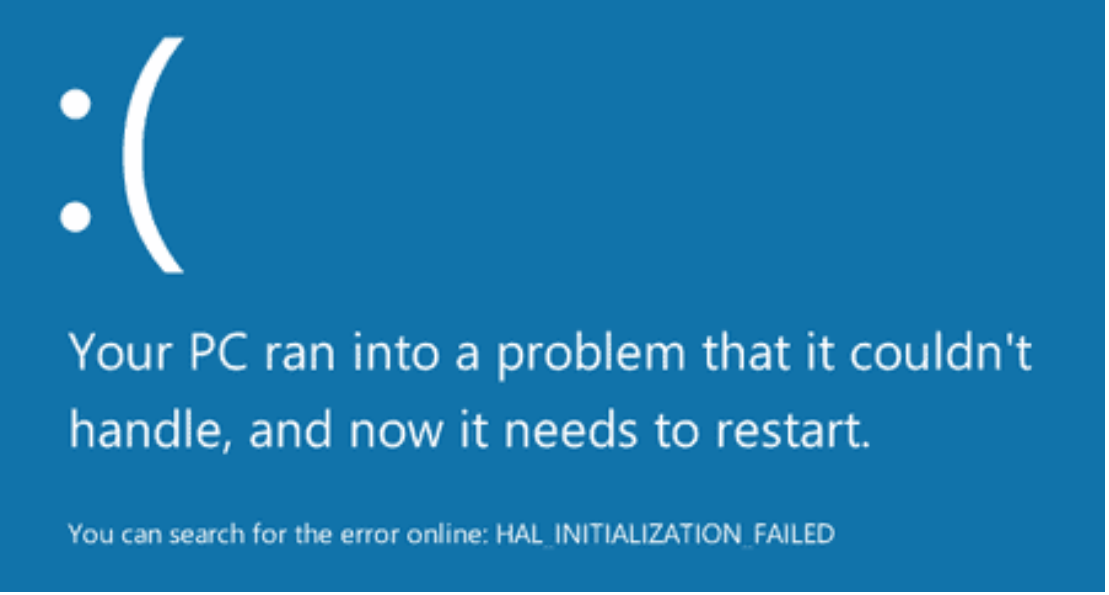
» For expression

```
fn main() {
    for number in (1..4).rev() {
        println!("{}!", number);
    }
    println!("LIFTOFF!!!");
}
```

# کنترل ترتیب

- Match case expression

```
enum Coin {
    Penny,
    Nickel,
    Dime,
    Quarter,
}
fn value_in_cents(coin: Coin) -> u32 {
    match coin {
        Coin::Penny => 1,
        Coin::Nickel => 5,
        Coin::Dime => 10,
        Coin::Quarter => 25,
        _ => 0,
    }
}
```




# کنترل ترتیب

- کنترل خطا
  - قابل بازیابی
    - منطقی است که خطا را به کاربر گزارش دهیم و دوباره اجرا کنیم
    - عدم پیدا کردن فایل
  - غیرقابل بازیابی
    - خطاهای غیرقابل برگشت مثل دسترسی به یک مکان در انتهای آرایه

# کنترل ترتیب

- قابل بازیابی

```
use std::fs::File;

fn main() {
    let f = File::open("hello.txt").expect("Error:
File not found!");
    let f = File::open("hello.txt").unwrap();
    let f = File::open("hello.txt")?;
}
```

# کنترل ترتیب

- غیرقابل بازیابی

```
fn main() {
      panic!("crash and burn");
} // unrecoverable and stop programm execution
```

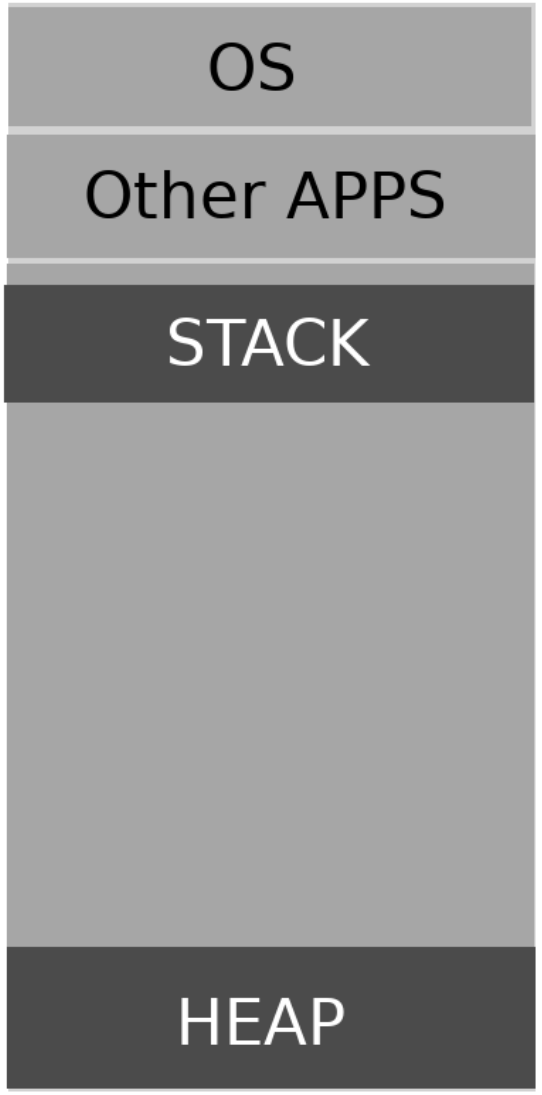
```
fn main() {
    let f = File::open("hello.txt");
    let f = match f {
        Ok(file) => file,
        Err(error) => {
            panic!("There was a problem opening the file: {:?}", error)
        },
    };
}
```

# مدیریت حافظه

- **Stack**
  - » Fast
  - » Local variables' home
  - » Function lifetime
  - » Starts from the first available block

# مدیریت حافظه

- **Heap**
  - » Slow
  - » Global variables' home
  - » Big in size
  - » Elastic
  - » Starts from the end of memory



# مدیریت حافظه

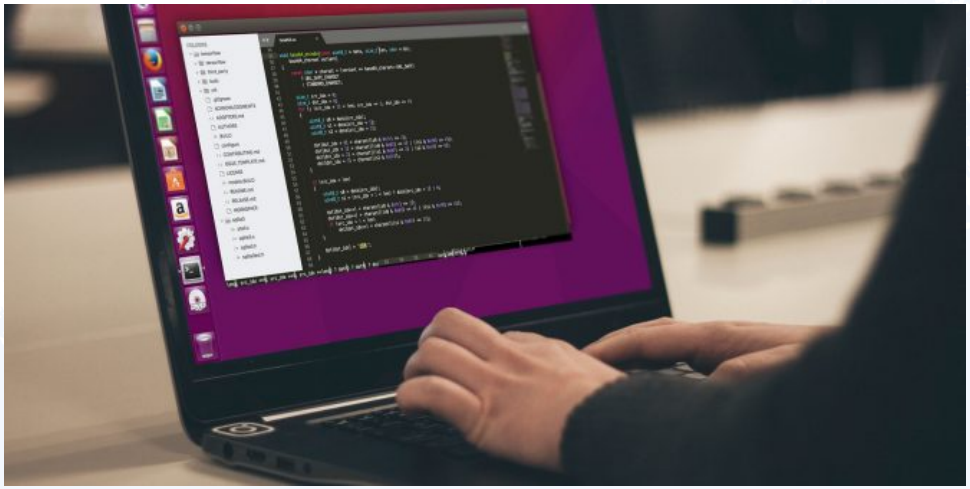
- Examples:

```
fn foo(i: &i32) {
    let z = 42;
}

fn main() {
    let x = 5;
    let y = &x;

    foo(y);
}
```

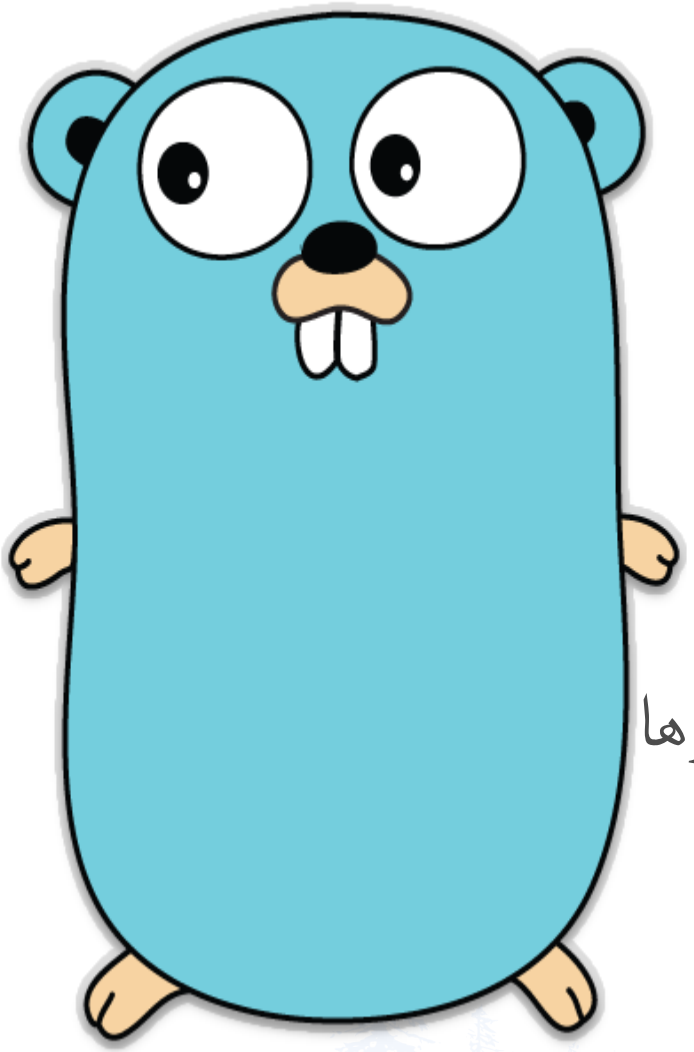
| Address | Name | Value |
|---|---|---|
| 3 | z | 42 |
| 2 | i | → 0 |
| 1 | y | → 0 |
| 0 | x | 5 |

# محیط عملیاتی

- PrintIn!("Rust is the best programming lang");
- input
  - stdin().read_line(&mut s).expect("Did not enter a correct string");
- let mut f = File::open("foo.txt")?;



# مقایسه با سایر زبان‌ها



- مقایسه با GO
  - Gc ندارد
  - نحو پیچیده‌تری دارد
  - ارتباط بهتر با زبان‌های دیگر
  - کنترل بهتر روی جزئیات (طول عمر متغییرها
  - سرعت توسعه پایین‌تر
  - همروندی بهتر

# مقایسه با سایر زبان‌ها


- مقایسه با JAVA
  - سرعت بالاتر
  - مصرف حافظه کمتر
  - GC ندارد
  - مناسب برای کارهای سطح پایین
  - سرعت توسعه پایینتر
  - سابقه کمتر

# مقایسه با سایر زبان‌ها

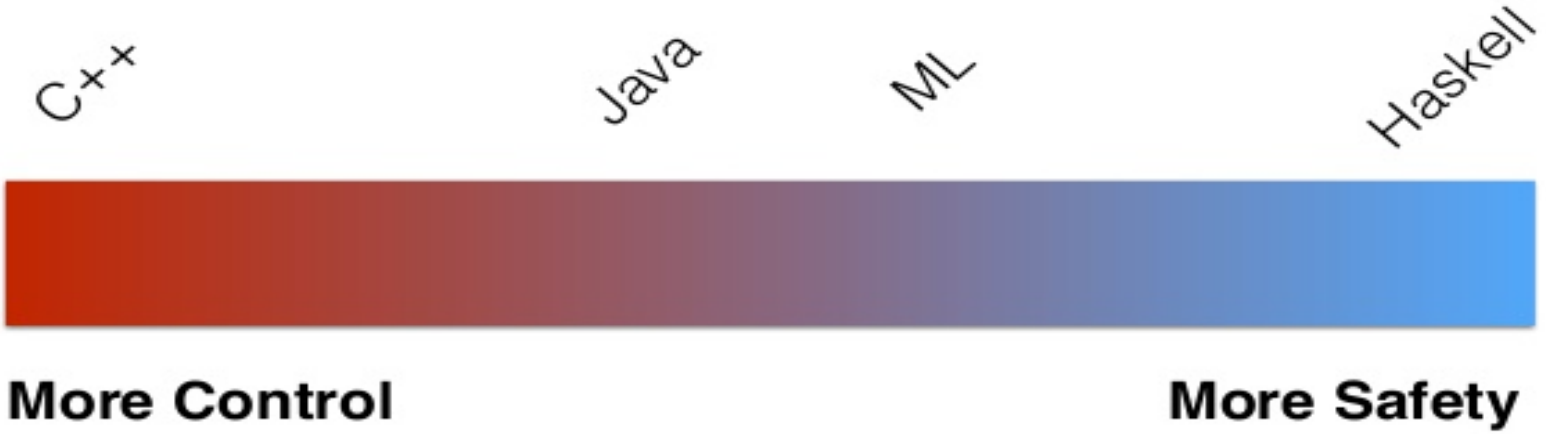
- مقایسه با PYTHON
  - سریعتر
  - سرعت توسعه پایین‌تر و نحو پیچده
  - همروندی بسیار بهتر
  - مشکل gil ندارد
  - مناسب‌تر برای کارهای سیستمی
  - جامعیت کمتر
  - افزایش عملکرد و سرعت با اجرای پایتون روی راست
  - انجمن کوچک‌تر

# مقایسه با سایر زبان‌ها

- مقایسه با C
  - امنیت بالاتر
  - سرعت کمی پایین‌تر
  - نحو ساده‌تر
  - مدیریت حافظه خودکار و بهتر
  - همروندی بهتر
  - شی گرایی و برنامه نویسی تابعی
  - سنگین‌تر ( سی برای سیستم های تعبیه شده بهتر است)
  - مستندات مسنجم تر

# مقایسه با سایر زبان‌ها

## What makes Rust different?




**Rust:** Control *and* safety

# نمونه برنامه

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

- Compile → rustc main.rs
- Run → ./main

# نمونه برنامه

```
fn iterative_factorial(n: int) -> int {
    let mut i = 1;
    let mut result = 1;
    while i <= n {
        result *= i;
        i += 1;
    }
    return result; /
}
```

```
fn recursive_factorial(n: int) -> int {
    if n <= 1 { 1 }
    else { n *recursive_factorial(n-1) } }
```

```
fn main() {
    println!("Recursive result: {:i}", recursive_factorial(10));
    println!("Iterative result: {:i}", iterative_factorial(10));
}
```

# نمونه برنامه

- cargo new project_name
  - » cargo.toml  src/  .git/
- cargo build
- cargo run
- cargo.toml → import new lib
- cargo install lib


Thank You