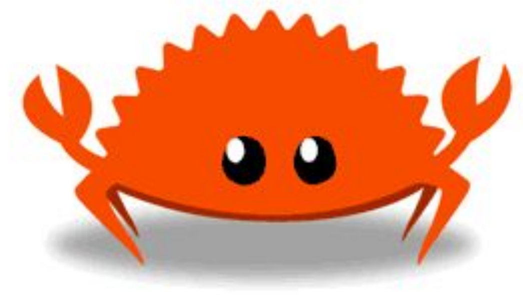
Introduction to Rust

By Pradip Hudekar

What is Rust?

A language empowering everyone to build reliable and efficient software.



Why another language?

Performance

Reliability

Productivity

Performance

- Compiled
- No Garbage Collector
- Does not need virtual machine
- Small runtime size
- Zero cost abstractions

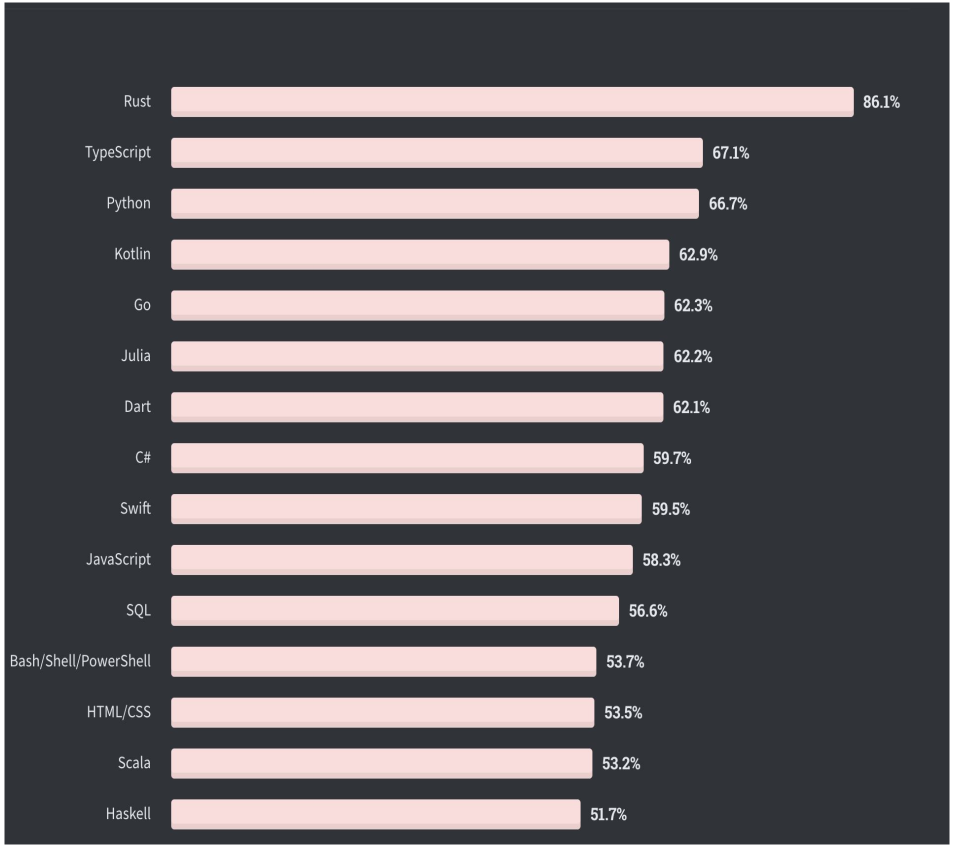
Reliability

If it compiles, it works !!

- Type checking
- Ownership model
- Memory safety
- Thread safety

Productivity

- Rich tooling for building large scale projects
- Native support for testing
- Cargo: Dependency and build manager
- Amazing community



Most Loved Language

Rust has been voted most loved language since last 5 years consistently on Stackoverflow survey

Hello Rust

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

Primitives

```
let name = "Ferris";
let age = 7;
let is_teen = age < 18;
let weight = 24.8;
let classes = [1, 2, 3];
```

Conditionals

```rust
if age > 12 && age < 18 {
    println!("Teenager")
} else if age <= 12 {
    println!("Kid")
} else {
    println!("Adult")
}
```

Loops

```
for month in 1..=12 {
    if (month % 2 != 0) {
        println!("31 days")
    } else if (month == 2) {
        println!("Depends")
    } else {
        println!("30 days")
    }
}
```

```
let mut n = 1;
while n < 5 {
    println!("{}", n);
    n = n + 1;
}
```

```
loop {
    println!("{}", n);
    n = n + 1;
    if n > 5 {
        break;
    }
}
```

Ownership Model

Structs

```rust
fn main() {
   let name = String::from("Ferris");
   let age = 7;
   let ferris = Person { name, age, weight: 24.8 };

   if ferris.is_teen() {
       println!("{} can vote", ferris.name);
   } else {
       println!("{} is not eligible to vote",
ferris.name);
   }
}
```

```rust
struct Person {
   name: String,
   age: u8,
   weight: f32,
}

impl Person {
   pub fn is_teen(&self) -> bool {
       self.age < 18
   }
}
```

Enumerations

```rust
use std::time::Instant;

fn main() {
     let message = LogMessage::Warning(
                    Instant::now(),
                    String::from("Be careful")
                    );
}
enum LogMessage {
   Info(String),
   Warning(Instant, String),
   Error(u32, Instant, String),
}
```

Pattern Matching

Traits

```rust
fn main() {
    let lion = Lion {};
    lion.make_sound();

    let dog = Dog {};
    dog.make_sound();
}

trait Animal {
    fn make_sound(&self);
}
```

```rust
struct Lion;
impl Animal for Lion {
    fn make_sound(&self) {
        println!("Lion roars!")
    }
}

struct Dog;
impl Animal for Dog {
    fn make_sound(&self) {
        println!("Dog barks!")
    }
}
```

Generics

```
trait Purchase<T>
where
   T: PaymentMethod,
{
   fn payment_method(&self) -> &T;
   fn process_payment(&self) -> String {
       let payment_method: &T =
self.payment_method();
       payment_method.charge()
   }
}
```

```
struct Subscription<T>
where
   T: PaymentMethod,
{
   name: String,
   payment_method: T,
}

impl<T> Purchase<T> for Subscription<T>
where
   T: PaymentMethod,
{
   fn payment_method(&self) -> &T {
       &self.payment_method
   }
}
```

Object Oriented Programming

Encapsulating data and methods to operate on it together

Functional Programming

Programs are constructed by applying and composing deterministic functions without causing side effects

Rust is a multi paradigm language

Smart Pointers

What is a smart pointer?

- Points to some address in memory
- Can hold additional logic
- Must have Deref and Drop methods

Box<T>

```
struct BinaryTree<T>
where
    T: PartialOrd,
{
    value: T,
    left: Option<Box<BinaryTree<T>>>,
    right: Option<Box<BinaryTree<T>>>,
}
```

Multi Owner model

Hmmm! Does Rust allow that?

Reference Counted Types

`Rc<T>`

Runtime Borrow Checking

`RefCell<T>`

Concurrent Programming

Problem with multithreading

- Race conditions
- Deadlocks
- Non-reproducible bugs

Threads

```rust
use std::thread;

fn main() {
    let handle = thread::spawn(|| println!("Hello from a thread"));
    handle.join().unwrap();
}
```

Threads Cont..

```rust
use std::thread;

fn main() {
    let message = "Hello World";
    let handle = thread::spawn(move || println!("{}", message));

    handle.join().unwrap();
}
```

Sharing memory

```
use std::thread;
fn main() {
    let mut threads = vec![];
    let mut count = 0;
    for i in 1..=10 {
        threads.push(thread::spawn(move || {
            count += 1;
            println!("{} -> {}", i, count)
        }));
    }
    for t in threads {t.join().unwrap();}
}
```

Using Arc with Mutex

```
use std::sync::{Arc, Mutex};
use std::thread;
fn main() {
   let mut threads = vec![];
   let counter = Arc::new(Mutex::new(0));
   for i in 1..=10 {
       let count = Arc::clone(&counter);
       threads.push(thread::spawn(move || {
           thread::sleep(std::time::Duration::from_secs(1));
           if let Ok(mut current) = count.lock() {
               *current += 1;
               println!("{} -> {}", i, *current)
           }
       }));
   }
    for t in threads {t.join().unwrap();}
}
```

Passing messages between threads

```
use std::sync::mpsc;
use std::thread;

fn main() {
   let mut threads = vec![];
   let mut counter = 0;
   let (sender, receiver) = mpsc::channel();
   for i in 1..=10 {
       let tx = sender.clone();
       threads.push(thread::spawn(move || {
thread::sleep(std::time::Duration::from_secs(1));
           tx.send((i, 1)).unwrap();
       }));
   }
```

```
   for _ in 1..=threads.len() {
       let (thread, increment) = receiver.recv().unwrap();
       counter += increment;
       println!("{} -> {}", thread, counter);
   }

   for t in threads {
       t.join().unwrap();
   }
}
```

Where can I use it?

- Command line applications
- Systems programming
- Networking applications
- Web applications
- Game development
- GUI Applications
- Web Assembly
- Embedded devices

Thank You!

Hope you find the joy and peace while practicing Rust.

References:
https://www.rust-lang.org/
https://doc.rust-lang.org/book
https://doc.rust-lang.org/rust-by-example/