Rust as a Second Language

Joey Bernard

Land Acknowledgement

Introduction

- Systems programming language developed by Mozilla

Introduction

- Systems programming language developed by Mozilla
- Focuses: safety, speed, concurrency

Introduction

- Systems programming language developed by Mozilla
- Focuses: safety, speed, concurrency
- Memory-safe without garbage collection

Introduction

- Systems programming language developed by Mozilla
- Focuses: safety, speed, concurrency
- Memory-safe without garbage collection
- Popular in performance-critical applications

Setup

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

Setup

- `rustup` tool: `curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`
- Components: Cargo (build tool), rustc (compiler), crates.io (package registry)

Setup

- `rustup` tool: `curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`
- Components: Cargo (build tool), rustc (compiler), crates.io (package registry)
- Pick a version of rust - `rustup default stable`

Setup

- `rustup` tool: `curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`
- Components: Cargo (build tool), rustc (compiler), crates.io (package registry)
- Pick a version of rust - `rustup default stable`
- Verify with `rustc --version`

Exercise 1

Setup a new project with `cargo new hello_rust`

Hello World

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

- `fn` keyword, `main` function

Hello World

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

- `fn` keyword, `main` function
- `println!` is a macro

Exercise 2

Change the message to see what else you can print out

Variables and Mutability

- `let` for bindings

Variables and Mutability

- `let` for bindings
- Immutable by default

Variables and Mutability

- `let` for bindings
- Immutable by default
- Type annotations optional but encouraged

Variables and Mutability

- `let` for bindings
- Immutable by default
- Type annotations optional but encouraged
- Use the `mut` annotation to make a variable mutable

Variables and Mutability - Examples

```
fn main() {
    let x = 5;
    let mut y = 10;
    y += 1;
    println!("x: {}, y: {}", x, y);
}
```

- let, mut

Variables and Mutability - Examples

```
fn main() {
    let x = 5;
    let mut y = 10;
    y += 1;
    println!("x: {}, y: {}", x, y);
}
```

- let, mut
- Type inference

Data Types

```
let tup = (500, 6.4, 1);
let arr = [1, 2, 3, 4, 5];
```

- Scalar: `i32`, `f64`, `bool`, `char`

Data Types

```
let tup = (500, 6.4, 1);
let arr = [1, 2, 3, 4, 5];
```

- Scalar: `i32`, `f64`, `bool`, `char`
- Compound: Tuples, Arrays

Exercise 3

Create variables of type `bool`, `char`, and `f64` and print them

Answer 3

```
fn main() {
    let x:f64 = 42;
    ptintln!("x: {}", x);
}
```

Exercise 4

Access and print specific values from the tuple and array

Answer 4

```
fn main() {
    let arr = [ 1, 2, 3, 4, 5 ];
    let x = arr[2];
    println!("2nd item {}", x);
}
```

Functions

```
fn add(x: i32, y: i32) -> i32 {
    x + y
}
fn main() {
    let result = add(2, 3);
    if result > 3 {
        println!("Greater than 3");
    } else {
        println!("Less or equal to 3");
    }
}
```

Control Flow

```
if x > 0 {
    println!("Positive");
}
```

- if, else

Control Flow

```
if x > 0 {
    println!("Positive");
}
```

- if, else
- match

Control Flow

```
if x > 0 {
    println!("Positive");
}
```

- if, else
- match
- loop, while, for

Exercise 5

Write a function to return the square of a number

Answer 5

```
fn sq(x: i32) -> i32 {
    x * x;
}
fn main() {
    println!("42 squared is ", sq(42));
}
```

```
## Break 10 minutes
```

Answer 5

```
fn sq(x: i32) -> i32 {
    x * x;
}
fn main() {
    println!("42 squared is ", sq(42));
}
```

```
## Break 10 minutes
```

Answer 5

```
fn sq(x: i32) -> i32 {
    x * x;
}
fn main() {
    println!("42 squared is ", sq(42));
}
```

```
## Break 10 minutes
```

References

```
fn print_len(s: &String) {
    println!("{}", s.len());
}
```

- Borrow with &

References

```
fn print_len(s: &String) {
    println!("{}", s.len());
}
```

- Borrow with `&`
- Mutable with `&mut`

References

```
fn print_len(s: &String) {
    println!("{}", s.len());
}
```

- Borrow with `&`
- Mutable with `&mut`
- Prevents data races

Exercise 6

How can you access `s1`?

Answer 6

```
fn main() {
    let s1 = String::from("hello");
    let s2 = &s1;
    println!("{}", s1); // Error!
    println!("{}", s2);
}
```

Borrowing

```
fn main() {
    let s1 = String::from("hello");
    let len = calculate_length(&s1);
    println!("Length: {}", len);
}
fn calculate_length(s: &String) -> usize {
    s.len()
}
```

- & means reference

Borrowing

```
fn main() {
    let s1 = String::from("hello");
    let len = calculate_length(&s1);
    println!("Length: {}", len);
}
fn calculate_length(s: &String) -> usize {
    s.len()
}
```

- & means reference
- Immutable and mutable borrowing

Mutable Borrowing

```
fn modify(s: &mut String) {
    s.push_str(" world"## Mutable);
}
```

Exercise 7

Create a mutable string and pass it to `modify`

```
fn modify(s: &mut String) {
    s.push_str(" world"## Mutable);
}
fn main() {
    let mut s1 = String::from("hello");
    modify(&mut s1);
}
```

Structs and Enums

```
struct User {
    username: String,
    active: bool,
}
enum Message {
    Quit,
    Move { x: i32, y: i32 },
}
```

- Custom data types

Structs and Enums

```
struct User {
    username: String,
    active: bool,
}
enum Message {
    Quit,
    Move { x: i32, y: i32 },
}
```

- Custom data types
- `impl` blocks for methods

impl

```
impl User {
    fn print(&self) {
        println!("User: {}, active: {}", self.name, self.ac
    }
}
```

Exercise 8

Create and print a `User`; create an enum `Status` with at least 2 variants

Answer 8

```
struct User {
    username: String,
    active: bool,
}
impl User {
    fn print(&self) {
        println!("User: {}, active: {}", self.name, self.ac
    }
}
fn main() {
    let user1 = User {
        username: String::from("Joey"),
        active: true,
    };
    println!("Username: {}", user1.username);
```

Matching

```
match direction {
    Direction::Up => println!("Up"),
    _ => println!("Other")
}
```

- Powerful control Structure

Option and Result

```
fn divide(a: i32, b: i32) -> Option<i32> {
    if b != 0 { Some(a / b) } else { None }
}
```

- Avoid nulls and unchecked errors

Option and Result

```
fn divide(a: i32, b: i32) -> Option<i32> {
    if b != 0 { Some(a / b) } else { None }
}
```

- Avoid nulls and unchecked errors
- ***Option*** allows you to return either ***Some()***, or return ***None***

Option and Result

```
fn divide(a: i32, b: i32) -> Option<i32> {
    if b != 0 { Some(a / b) } else { None }
}
```

- Avoid nulls and unchecked errors
- ***Option*** allows you to return either ***Some()***, or return ***None***
- ***Result*** allows you to return either ***Ok()*** or ***Err()***

Option and Result

```
fn divide(a: i32, b: i32) -> Option<i32> {
    if b != 0 { Some(a / b) } else { None }
}
```

- Avoid nulls and unchecked errors
- ***Option*** allows you to return either ***Some()***, or return ***None***
- ***Result*** allows you to return either ***Ok()*** or ***Err()***
- Allows for more robust return values

Break 10 minutes

Vectors and Hashmaps

```rust
let mut v = vec![1, 2, 3];
v.push(4);

use std::collections::HashMap;
let mut scores = HashMap::new();
scores.insert(String::from("Blue"), 10);
```

Exercise 9

Iterate over a `Vec` and print entries

Answer 9

```
fn main() {
    let x = vec![1, 2, 3];
    for num in x.iter() {
        println!("Number: {}", num);
    }
}
```

Traits and Generics - 1

```
trait Drawable {
    fn draw(&self);
}
```

- Like interfaces

Traits and Generics - 1

```
trait Drawable {
    fn draw(&self);
}
```

- Like interfaces
- Enable polymorphism

Traits and Generics - 2

```
trait Summary {
    fn summarize(&self) -> String;
}
struct Article {
    title: String,
}
impl Summary for Article {
    fn summarize(&self) -> String {
        format!("Read more: {}", self.title)
    }
}
```

- Polymorphism with traits

Traits and Generics - 2

```
trait Summary {
    fn summarize(&self) -> String;
}
struct Article {
    title: String,
}
impl Summary for Article {
    fn summarize(&self) -> String {
        format!("Read more: {}", self.title)
    }
}
```

- Polymorphism with traits
- Generic types: `fn largest<T: PartialOrd>(list: &[T]) -> T { ... }`

Threads

```rust
use std::thread;

let handle = thread::spawn(|| {
    for i in 1..5 {
        println!("Thread count: {}", i);
    }
});
handle.join().unwrap();
```

- Threads, channels, async/await

Threads

```rust
use std::thread;

let handle = thread::spawn(|| {
    for i in 1..5 {
        println!("Thread count: {}", i);
    }
});
handle.join().unwrap();
```

- Threads, channels, `async/await`
- Safe and ergonomic concurrency

Exercise 10

Create two threads and have each print different messages

Answer 10

```
use std::thread;

let handle = thread::spawn(|| {
    for i in 1..5 {
        println!("Thread id: {}", thread::current().id());
    }
});
handle.join().unwrap();
```

Unsafe

- Dereference a raw pointer

Unsafe

- Dereference a raw pointer
- Call an unsafe function or method

Unsafe

- Dereference a raw pointer
- Call an unsafe function or method
- Access or modify a mutable static variable

Unsafe

- Dereference a raw pointer
- Call an unsafe function or method
- Access or modify a mutable static variable
- Implement an unsafe trait

Unsafe

- Dereference a raw pointer
- Call an unsafe function or method
- Access or modify a mutable static variable
- Implement an unsafe trait
- Access fields of a union

Unsafe - Example

```
let mut num = 5;

    let r1 = &raw const num;
    let r2 = &raw mut num;

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

Further Resources

- https://rustlings.rust-lang.org/

Further Resources

- https://rustlings.rust-lang.org/
- https://doc.rust-lang.org/book/

Further Resources

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

Further Resources

- https://rustlings.rust-lang.org/
- https://doc.rust-lang.org/book/
- https://doc.rust-lang.org/rust-by-example/
- https://crates.io

Further Resources

- https://rustlings.rust-lang.org/
- https://doc.rust-lang.org/book/
- https://doc.rust-lang.org/rust-by-example/
- https://crates.io
- https://docs.rs

Further Resources

- https://rustlings.rust-lang.org/
- https://doc.rust-lang.org/book/
- https://doc.rust-lang.org/rust-by-example/
- https://crates.io
- https://docs.rs
- https://play.rust-lang.org/

Further Resources

- https://rustlings.rust-lang.org/
- https://doc.rust-lang.org/book/
- https://doc.rust-lang.org/rust-by-example/
- https://crates.io
- https://docs.rs
- https://play.rust-lang.org/
- https://exercism.org/tracks/rust