Clone the repository

github.com/terhechte/appbuilders2022

Feel invited to work in pairs!

Language Introduction

Cargo

```
cargo new
cargo check
cargo run
cargo build
cargo test
cargo build --release
cargo run --release
```

cargo --help

```
cargo search graphql
cargo doc --open
cargo bench
cargo update
cargo clean
```

First steps

```
sh$ cargo new my-fancy-project
sh$ ls .
my-fancy-project
sh$
```

Structure of a binary Rust project

```
Cargo.toml
Cargo.lock
src/
  main.rs
target/
  debug/
  release/
```

Structure of a library Rust project

```
Cargo.toml
Cargo.lock
src/
  lib.rs
```

Cargo.toml

```
[package]
name = "my-project"
version = "0.1.0"
edition = "2021"

[dependencies]
serde_json = "1.0"
serde = { version = "1.0.137", features = ["derive"]}
eyere = "*"
```

Rust	Swift
String	String
usize	UInt
isize	Int
i8	Int8
i32	Int32
u8	UInt8
u32	UInt32
bool	Bool
f32	Float
f64	Double

Functions

Swift

```swift
func example(content: String) -> String {
    content
}
```

Rust

```rust
fn example(content: String) -> String {
    content
}
```

Swift

```swift
func example(content: Int) -> Int {
    let output = content * 2
    return output
}

example(content: 42)
```

Rust

```rust
fn example(content: usize) -> usize {
    let output = content * 2;
    output
}

example(42)
```

Swift

```swift
func example(content: String) -> String {
    var output = content * 2
    output += 50
    return output
}
```

Rust

```rust
fn example(content: usize) -> usize {
    let mut output = content * 2;
    output += 50;
    output
}
```

Swift

```swift
func example<T>(content: T) -> T {
    content
}
```

Rust

```rust
fn example<T>(content: T) -> T {
    content
}
```

Swift

```swift
func compareTwo<T: Equatable>(a: T, b: T
 ↳ ) -> Bool {
    a == b
}
```

Rust

```rust
fn compare_two<T: Eq>(a: T, b: T) -> bool
 ↳  {
    a == b
}
```

Swift

```swift
func compareTwo<T: Equatable>(a: T, b: T
 ↳ ) -> Bool
where T: Equatable {
    a == b
}
```

Rust

```rust
fn compare_two<T>(a: T, b: T) -> bool
where T: Eq {
    a == b
}
```

Closures

Swift

```swift
callAction({ a, b in
    a * b
})

callAction {
    $0 * $1
}

func takesClosure(action: (Int) -> bool)
```

Rust

```rust
call_action(|a, b| {
    a * b
})

call_action(|a, b| a * b)

fn takes_closure(action impl Fn(usize) ->
 ↳ bool)
```

Printing

Swift

```swift
let o = 42
print(o)
```

Rust

```rust
let o = 42;
println!("{}", o);
```

Swift

```swift
let o = 42
print("\(o)")
```

Rust

```rust
let o = 42;
println!("{o}");
```

Swift

```swift
let o = 42
print("\(o)")
```

Rust

```rust
let o = make_complex_type();
println!("{:?}", &o);
```

Arrays / Vectors

Swift

```swift
let array = [1, 2, 3]
```

Rust

```rust
let array = [1, 2, 3];
```

Swift

```swift
// Dynamic array
var array = [1, 2, 3]
```

Rust

```rust
// Static array, can't be extended
let mut array = [1, 2, 3];
```

Swift

```swift
// Dynamic array
var array = [1, 2, 3]
```

Rust

```rust
// Dynamic array
let mut array = vec![1, 2, 3];
let mut array = vec![];

array.push(7);
array.push(8);
```

Swift

```swift
let map = [
    5: "My favorite book."
]
```

Rust

```rust
let mut map = HashMap::new();
map.insert(5, "My favorite book.");
);
```

Tuples

Swift

```swift
let tuple = ("a", 5, 42.0, true)
print(tuple.0)
```

Rust

```rust
let tuple = ("a", 5, 42.0, true);
println!(tuple.0);
```

Control Structures

Swift

```swift
if 5 > 10 {
    return 20
} else {
    return 30
}
```

Rust

```rust
if 5 > 10 {
    return 20;
} else {
    return 30;
}
```

Rust

```rust
fn test() -> usize {
    // The last expression is returned
    let a = 4 + 2;
    let b = a * 3
    b
}
```

Rust

```
let hello = if a > b {
    23
} else {
    24
};

let what = loop {
    if a > 3 {
        break 23;
    }
};
```

Swift

```swift
for x in array {
    print(x)
}
```

Rust

```rust
for x in array {
    println!(x);
}
```

First task

Tip: Debug Printing

You can use `dbg!(...)` for debug printing

```
// It will just pipe the data through
let my_thing = dbg!(call_thing());
```

Tip: Debugger

- If you installed the debugger VSCode extension, you have a proper debugger
- Just press F5 to run the app in the debugger

Strings

- Rust has two String types. This will be explained later.
- The initial examples try to use `String`. You might see `&str` though.
- A `String` is composed out of `char`s. You can get them via `string.chars()`

Demo

- Docs
- Crates.io
- VSCode

Project 1

- Read characters from a file
- Figure out how many symbols are math symbols
- Some code has already been written
- See `Readme.md`

Enums

Swift

```swift
enum Gender {
    case male
    case female
    case neutral
}
```

Rust

```rust
enum Gender {
    Male,
    Female
    Neutral
}
```

Swift

```swift
enum Example {
    case first(String)
    case second(Float)
    case third(Int, String)
}
```

Rust

```rust
enum Example {
    First(String),
    Second(f32),
    Third(i32, String),
}
```

Rust

```
enum Example {
    First { value: String },
    Second { value: f32 },
    Third { value: i32, content: String }
}
```

Swift

```swift
enum Gender { case male, female }
let x = Gender.female
```

Rust

```rust
enum Gender { Male, Female }
let x = Gender::Male;
```

Optionals

Swift

```
enum Optional<T> {
    case some(T)
    case none
}
```

Rust

```
enum Option<T> {
    Some(T)
    None
}
```

Rust

```rust
if let Some(unwrapped) = optional_value {
}

if let Option::Some(unwrapped) = optional_value {
}
```

Pattern Matching

Swift

```swift
let o = Optional.some(Value.number(42))
switch readUser() {
    case .some(.number(let n)) where n >
 ↳ 10: print(n),
    case .some(let o): print(o),
    default: ()
}
```

Rust

```rust
let o = Some(Value::Number(42))
match read_user() {
    Some(Value::Number(n)) if n > 10
 ↳  ⇒ println!(n),
    Some(o) ⇒ println!("{o}"),
    _ ⇒ ()
}
```

Swift

```swift
guard case let .number(n) = readUser, n >
 ↳ 10 else { return }
```

Rust

```rust
let nr = match read_user() {
    Some(Value::Number(n)) if n > 10
 ↳  ⇒ n,
    _ ⇒ return
}
```

Rust

```
fn ex() → Option<()> {
    let lastChar = user?.name?.chars().first?.int;
}
```

Structures

Swift

```swift
struct Hello {
    let first: String
    var second: Int
}
```

Rust

```rust
struct Hello {
    first: String,
    second: i32
}
```

Rust

```rust
struct Hello {
    first: String,
}
impl Hello {
    // self.print_intro();
    fn print_intro(&self) {
        // In rust, `self.` is required
        println!("Hello {}", self.first)
    }

    // static fn: Hello.print_intro();
    fn print_intro() {
       ...
    }
}
```

Swift

```swift
struct Hello: Codable {
    let first: String
    var second: Int
}
```

Rust

```rust
// Requires `serde` dependency
#[derive(Serialize, Deserialize)]
struct Hello {
    first: String,
    second: i32
}
```

Rust

```rust
#[derive(Debug, Clone, Default)]
struct Hello {
    first: String,
    second: i32
}

let x = Hello::default();
let y = x.clone();
println!("{:?}", x);
```

Error Handling

Rust

```rust
fn parse_me(content: String) -> Result<i32> {
    let parsed: i32 = content.parse()?;
    parsed *= 100;
    Ok(parsed)
}
```

Rust

```rust
fn parse_me(content: String) -> Option<i32> {
    let parsed: i32 = content.optional_fn().ok()?;
    parsed *= 100;
    Some(parsed)
}
```

Importing

```
std
  ::result
        ::Result
  ::collections
            ::HashMap
            ::Vec
            ::BTreeSet

use std::result::Result;
use std::collections::{HashMap, Vec};
```

```
src/
  lib.rs
  utils.rs
  project.rs
```

```rust
mod utils;
mod project;

use std::result::Result;
```

Unit Tests

```
fn my_func(u: usize) -> bool { ... }

// At the bottom of source file
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_my_func() {
      assert_eq!(my_func(3), true);
    }
}
```

Second Task

Tip: Unwrap

- Instead of using `match` or `if let` use `.unwrap()` which force unwraps

```
let x: Option<usize> = None;

use_me(x.unwrap());
```

Tip: Clone

> `error[E0382]: borrow of moved value: name-of-variable`

Try `.clone()`. This will be explained later

```
let u = MyStruct::default();
my_func(u.clone());
```

Project 2

- Detect language of Tweet
- Remove stop words depending on the tweet language
- Find most used words in a set of tweets
- Some code has already been written
- Checkout `utils.rs`.
- `Readme.md`

Remember this? So verbose

```
let mut o = vec![];
for x in items {
   if something(x) {
     o.push(x);
   }
}
```

What about.. `map` & friends?

```
collection
 .map(|e| e * 2)
 .filter(|e| e > 5)
```

Stay Tuned!

Traits / Protocols

Interfaces

Swift

```swift
protocol MyProtocol {
    func partyBoy() -> Int;
}

extension Hello: MyProtocol {
    func partyBoy() -> Int {
        42
    }
}
```

Rust

```rust
trait MyTrait {
    fn party_boy() -> i32;
}

impl MyTrait for Hello {
    func party_boy() -> i32 {
        42
    }
}
```

Swift

```swift
protocol MyProtocol {
    associatedType Output;
    func partyBoy() -> Self.Output;
}

extension Hello: MyProtocol {
    func partyBoy() -> Int {
        42
    }
}
```

Rust

```rust
trait MyTrait: PartialEq {
    type Output;
    fn party_boy(&self) -> Self::Output;
}

impl MyTrait for Hello {
    type Output = i32;
    fn party_boy(&self) -> i32 {
        42
    }
}
```

Swift

```swift
struct Hello {
    let first: String
    var second: Int
}

extension Hello: Equatable {
    static func == (lhs: Self, rhs: Self
 ↳ ) -> Bool {
        lhs.second == rhs.second
    }
}
```

Rust

```rust
struct Hello {
    first: String,
    second: i32
}

impl PartialEq for Hello {
    fn eq(&self, other: &Self)
      -> bool {
        self.second == other.second
    }
}
```

Swift

```swift
struct Hello: Equatable {
    let first: String
    var second: Int
}
```

Rust

```rust
#[derive(Eq, PartialEq)]
struct Hello {
    first: String,
    second: i32
}
```

Rust

```rust
// std::iter::Iterator Trait
pub trait Iterator {
    ...
    fn count(self) -> usize
    fn filter<P>(self, predicate: P) -> Filter<Self, P>
    fn take_while<P>(self, predicate: P) -> TakeWhile<Self, P>
    fn flat_map<U, F>(self, f: F) -> FlatMap<Self, U, F>
    fn enumerate(self) -> Enumerate<Self>
    ...
}
```

Swift

```swift
data.map { $0 * 2 }
    .filter { $0 > 10 }
    .prefix(3)
    .reduce(0, +)
```

Rust

```rust
data.iter()
    .map(|a| a * 2)
    .filter(|a| a > 10)
    .take(3)
    .sum();
```

Rust

```
// Getting a iterator
collection.iter().map(...)

collection.into_iter().map(...)
```

Memory Management

Rust distinguishes between Stack and Heap.

Stack

- Values on the stack are fast to access
- They're oftentimes values such as `usize` or `enum` types

Heap

- Slower to access because they work by allocating data via a pointer
- References to data
- Oftentimes bigger data like `Vec` or `HashMap`

The developer can decide where to put the data:

```
let x = 5; // Stack
let y = Box::new(5) // Heap
let x = MyStruct::new(); // Stack
let x = Box::new(MyStruct::new()) // Heap
```

Ownership

```rust
let container = vec![1, 2, 3];
let other_container = container;

println!("{}", other_container);

// Error: borrow of moved value: `container`
println!("{}", container);
```

```rust
let container = vec![1, 2, 3];

// Don't move, *borrow*
let other_container = &container;

println!("{}", &other_container);

println!("{}", &container);
```

`&something` means borrow something

Don't own it, just borrow it, somebody else still owns it.

```rust
fn ppp(value: &Vec<usize>) {
    println!("{value}");
}

let x = vec![1, 2, 3];
ppp(&x);

// Works!
println!("{:?}", &x);
```

Some types change shape as references [1]

Owned	Borrowed
String	&str
Vec	&[T]
PathBuf	&Path

[1] `&String` or `&Vec<T>` is not wrong, but will emit a warning

```rust
let first: &str = "Hello";
let second: String = "Hello".to_owned();

fn test(input: &str) -> { ... }

test(first);
test(second); // Fails
test(&second); // works
```

Because

- `String` is just a `&str` allocated on the heap
- `Vec<T>` is just a `&[T]` allocated on the heap

```
// Get a &T Iterator (Borrow the value)
collection.iter().map(...)
// Get a T Iterator (Own the value)
collection.into_iter().map(...)
```

Tip 1

`clone` instead of moving

```
#[derive(Clone)]
struct MyStruct;

let p = MyStruct;

fn test(input: MyStruct) { ... }
test(p.clone());
```

Tip 2

Use Reference-Counting

```
use std::rc::Rc;

let p = Rc::new(vec![1]);

fn test(input: Rc<Vec<usize>>) { ... }
test(p.clone());

// Atomic Reference Counting: For Multi-Threading
use std::sync::Arc;
let p = Arc::new(5);
```

Building Mobile Libraries

- `cbindgen` for C / C++
- Mozilla `Uniffi` for Android, Swift, Python, Ruby
- Define interface in `udl` file

```
dictionary EmojiCount {
    string emoji;
    u32 count;
};

interface EmojiStream {
    sequence<EmojiCount> next_items(u32 limit);
};
```

```
# Android
uniffi-bindgen generate src/libemojistream.udl --language kotlin --out-dir generated/

# iOS
xcode_uniffi create
```

Demo

- Show how the final project runs in Xcode
- Show the uniffi Bindings

Final Project

- Implement the `grouped_charts` and `grouped_emoji` functions
- It will find all emoji in a set of Tweets and group them
- Then it will calculate a tree map for these emoji groups
- You can run the binary (`cargo run`)
- Or you can run it from Xcode by opening the Xcode project

Outlook

Macros in Rust

- Build-In Macro system
- Expand the language at compile time

```
format!("{}", hello);
```

Macros in Rust

- Build-In Macro system
- Expand the language at compile time

```
html! {
        <body>
            <h1>{ content.title }</h1>
            { content.items.map(|item| html!(
                <h2>{ format!("{}", item.name) }</h2>
            }
        </body>
    }
```

Lots of crazy stuff

```
let n = 5;
python! {
    for i in range('n):
        print(i, "Hello", 'who)
    print("Goodbye")
}
```

Rayon

One additional line to run your code on all cores

```
data
    .par_iter()
    .map(|a| a * 2)
    .filter(|a| a > 10)
    .take(3)
    .sum();
```

Other Targets

- C: cbindgen
- C++: cxx
- Swift: swift-bridge
- Flutter: flutter_rust_bridge
- Dart: membrane
- Python: Pyo3
- Ruby: rutie
- Javascript / Wasm: Trunk
- Example: https://github.com/imWildCat/uniffi-rs-fullstack-examples