

APT Session 5: Rust



Laurence
Tratt



Software Development Team
2017-11-17

What to expect from this session

1 Rust.

Prerequisites

- 1 Install Rust from <https://www.rust-lang.org/en-US/install.html>

- Programming languages involve trade-offs e.g.: Python is easy to program in, but can hide trivial mistakes, and is slow; C is fast and powerful, but is hard to program in reliably.

Rust

- Programming languages involve trade-offs e.g.: Python is easy to program in, but can hide trivial mistakes, and is slow; C is fast and powerful, but is hard to program in reliably.
- Rust is a new design point: it is safe (unlike C), fast (unlike Python), doesn't require garbage collection (like C, but unlike Python or Java), but doesn't require explicit `malloc/free` (unlike C).

Rust

- Programming languages involve trade-offs e.g.: Python is easy to program in, but can hide trivial mistakes, and is slow; C is fast and powerful, but is hard to program in reliably.
- Rust is a new design point: it is safe (unlike C), fast (unlike Python), doesn't require garbage collection (like C, but unlike Python or Java), but doesn't require explicit `malloc/free` (unlike C).
- Many people think of it is a plausible 'replacement' for C/C++.

Rust

- Programming languages involve trade-offs e.g.: Python is easy to program in, but can hide trivial mistakes, and is slow; C is fast and powerful, but is hard to program in reliably.
- Rust is a new design point: it is safe (unlike C), fast (unlike Python), doesn't require garbage collection (like C, but unlike Python or Java), but doesn't require explicit `malloc/free` (unlike C).
- Many people think of it as a plausible 'replacement' for C/C++.
- The [library documentation](#) is (mostly) very good. You'll need it this week.

The basics (1)

Rust has some syntactic similarities to Java, but there are no classes or objects. At first we can just consider top-level functions. Other useful things to know:

- The main method in Rust is `fn main()` which neither takes nor returns any arguments.
- Can print to the console with `println!("msg");`

The basics (1)

Rust has some syntactic similarities to Java, but there are no classes or objects. At first we can just consider top-level functions. Other useful things to know:

- The main method in Rust is `fn main()` which neither takes nor returns any arguments.
- Can print to the console with `println!("msg");`

Exercises:

- 1 Write a program which prints out `Hello world!` in Rust. Put it in a file `hello.rs` and compile it with `rustc hello.rs`. This will produce a file called `hello` which can then be run.

The basics (2)

- Format strings can be used e.g. `println!("{}", ...)`.
- Immutable variables (i.e. “assign only”) are introduced with `let x = ...;`
- Mutable variables (i.e. “can update their values”) are introduced with `let mut y = ...;`
- Iterate over numbers with `for i in num1..num2 {...}`.

The basics (2)

- Format strings can be used e.g. `println!("{}", ...)`.
- Immutable variables (i.e. “assign only”) are introduced with `let x = ...;`
- Mutable variables (i.e. “can update their values”) are introduced with `let mut y = ...;`
- Iterate over numbers with `for i in num1..num2 {...}`.

Exercises:

- 1 Write a program which assigns the string "hello world" to a variable `s` and then prints it out.
- 2 Write a program which sums all numbers between 0 and 100.

Vectors

- Vectors (i.e. growable arrays) are created with `vec! [a, b, c]`.
- Elements can be indexed via `v[0]`.
- A vectors length can be found with `v.len()`. Other methods can be found in the [documentation for Vecs](#).

Vectors

- Vectors (i.e. growable arrays) are created with `vec! [a, b, c]`.
 - Elements can be indexed via `v[0]`.
 - A vectors length can be found with `v.len()`. Other methods can be found in the [documentation for Vecs](#).
-

Exercises:

- 1 Create a vector with the 3 strings (in order) `c`, `b`, `a` and assign it to a variable.
- 2 Iterate over the vector and print out all its elements.
- 3 Sort the vector and print the elements out in a for loop.
- 4 What happens if you try to `println!` the vector? Try using the `{:?}` formatter instead of `{}`.

Borrowing

- Rust's most distinctive (and odd!) feature: no garbage collection, but no `malloc/free` either.
- High-level idea: whoever creates a 'thing' (e.g. a vector) owns it, but can temporarily borrow/lend it someone else, or permanently move it to someone else.
- Move is the default; lending happens with `&` (immutable borrow) or `&mut` (mutable borrow).

Borrowing

- Rust's most distinctive (and odd!) feature: no garbage collection, but no `malloc/free` either.
- High-level idea: whoever creates a 'thing' (e.g. a vector) owns it, but can temporarily borrow/lend it someone else, or permanently move it to someone else.
- Move is the default; lending happens with `&` (immutable borrow) or `&mut` (mutable borrow).

Exercises:

- 1 Create a function which takes a vector of integers (`Vec<u64>`), sorts it, and prints it out.
- 2 Create a function which takes a mutable borrow of integers (`&mut Vec<u64>`) and sorts it in place (i.e. without copying).

- `File::open` (`p`) opens the file `p` for reading. It returns a `Result` type denoting success or failure.
- Via the `match` statement, we can use *pattern matching* to uncover the outcome:

```
match File::open(...) {  
  Ok(f) => { /* read file contents */ },  
  Err(e) => { panic!("Couldn't open file: {}", e); }  
}
```

Note that `match` is exhaustive: you *have* to cater for all cases. `'_'` is a wildcard which means “match all the unstated cases” and can be useful to avoid writing out all cases.

- [File::open](#) (p) opens the file p for reading. It returns a [Result](#) type denoting success or failure.
- Via the `match` statement, we can use *pattern matching* to uncover the outcome:

```
match File::open(...) {  
  Ok(f) => { /* read file contents */ },  
  Err(e) => { panic!("Couldn't open file: {}", e); }  
}
```

Note that `match` is exhaustive: you *have* to cater for all cases. `'_'` is a wildcard which means “match all the unstated cases” and can be useful to avoid writing out all cases.

Exercises:

- 1 Create a file `sort.rs` which reads the contents of a file, [splits it into lines](#), sorts them, and then prints them out.

cargo

- `cargo` is a package manager distributed with Rust. It makes using external libraries from crates.io easy.
- The `Cargo.toml` file tells `cargo` how a program is laid out and which libraries it requires.

cargo

- `cargo` is a package manager distributed with Rust. It makes using external libraries from crates.io easy.
 - The `Cargo.toml` file tells `cargo` how a program is laid out and which libraries it requires.
-

Exercises:

- 1 Make a new directory `htvld`, `cd` into it, and run `cargo init --bin`. What happens if you run `cargo run`?
- 2 Edit `Cargo.toml`: under the `[dependencies]` heading, tell Cargo you want the [url](#) package with the line `url="1.6.0"`.
- 3 In `src/main.rs` you'll have to add `extern crate url`. After that you can `use` (i.e. `import`) the module's contents as normal.
- 4 [Parse URLs](#) given [on the command line](#) and report nicely to the user if they're valid or not (e.g. `cargo run http://127.0.0.1/ x` should complain about `x`).

A web server

Write a web server that can return simple HTTP requests. Some hints:

- [Bind](#) to `127.0.0.1:9090`
- Reading from network sockets is painful (e.g. has the other end really finished sending data or just paused?). [Read](#) into a fixed sized buffer (use the `[0; 1024]` syntax which means 'an array of length 1024').
- You will need to 'use' the [Read](#) and [Write](#) traits. These bring methods you'll need into scope.
- A simple HTTP response is `HTTP/1.1 200 OK\r\n\r\n` followed by content (e.g. HTML).
- [Flush](#) the stream after you've written to it (otherwise your content may be buffered indefinitely).

Post-session exercises

Try these (no particular order):

- 'Exceptions' (i.e. functions which return a `Result` type) are made easy to use through the '?' operator.
- `cargo check` is very fast to tell you if there are type errors, which is useful when you're prototyping.
- `cargo test` knows about Rust's `#[test]` framework.
- crates.io has many useful and interesting libraries.
- Look at the [full range of documentation](#).