Rust - Day 1

Why Rust?

- Low-level code is prone to various subtle bugs, which in most other languages can be caught only through extensive testing and reviews. Rust compiler plays a gate keeper role by refusing to compile with these bugs. Hence, devs have to spend less time on bugs.
- Cargo is the included dependency manager and build tool, used for adding, compiling and managing dependencies.
- Rustfmt ensures consistent coding style
- Helps understand system concepts like OS development.
- **Hundreds of companies, large and small, use Rust in production for a variety of tasks, including command line tools, web services, DevOps tooling, embedded devices, audio and video analysis and transcoding, cryptocurrencies, bioinformatics, search engines, Internet of Things applications, machine learning, and even major parts of the Firefox web browser.**
- Rust code can run quickly and provides speed to write programs.

Hello World

```
fn main() {
    // ! is used to define a macro while functions are called without !
    println!("Hello, World!");
}
```

- `println!` calls a Rust macro. If it had called a function instead, it would be entered as `println` (without the `!`).
- **Rust is an *ahead-of-time compiled* language, meaning you can compile a program and give the executable to someone else, and they can run it even without having Rust installed.**
- **Everything is a trade-off in language design.**

Cargo

- Cargo is Rust's build system and package manager.
- Cargo handles a lot of tasks for you, such as building your code, downloading the libraries your code depends on, and building those libraries.
- Creating a project with Cargo

```
cargo new hello_cargo
```

- It has also initialized a new Git repository along with a *.gitignore* file. Git files won't be generated if you run `cargo new` within an existing Git repository; you can override this behavior by using `cargo new --vcs=git`.

- **TOML - Tom's Obvious, Minimal Language**
- Packages of code are referred to as *crates*.
- Cargo expects your source files to live inside the *src* directory.
- The top-level project directory is just for README files, license information, configuration files, and anything else not related to your code.

- If you started a project that doesn't use Cargo, as we did with the "Hello, world!" project, you can convert it to a project that does use Cargo. Move the project code into the *src* directory and create an appropriate *Cargo.toml* file. One easy way to get that *Cargo.toml* file is to run `cargo init`, which will create it for you automatically.
- Build the project

```
cargo build
```

- This command creates an executable file in *target/debug/hello_cargo* (or *target\debug\hello_cargo.exe* on Windows) rather than in your current directory. Because the default build is a debug build, Cargo puts the binary in a directory named *debug*.
- Running the project after build

```
./target/debug/hello_cargo
```

- Running `cargo build` for the first time also causes Cargo to create a new file at the top level: *Cargo.lock*. This file keeps track of the exact versions of dependencies in your project. This project doesn't have dependencies, so the file is a bit sparse. You won't ever need to change this file manually; Cargo manages its contents for you.
- Build and run the code in single command - **Note: Cargo would not rebuild the files if no changes are made between `cargo run`**

```
$ cargo run
  Compiling hello_cargo v0.1.0 (file:///projects/hello_cargo)
   Finished dev [unoptimized + debuginfo] target(s) in 0.33 secs
    Running `target/debug/hello_cargo`
Hello, world!
```

- Cargo also provides a command called `cargo check`. This command quickly checks your code to make sure it compiles but doesn't produce an executable

```
$ cargo check
  Checking hello_cargo v0.1.0 (file:///projects/hello_cargo)
   Finished dev [unoptimized + debuginfo] target(s) in 0.32 secs
```

- Instead of saving the result of the build in the same directory as our code, Cargo stores it in the *target/debug* directory.

Building for Release

- When your project is finally ready for release, you can use `cargo build --release` to compile it with optimizations. This command will create an executable in *target/release* instead of *target/debug*.