Rust Quick Reference

1. **Overview**

1.1 Characteristics

* Precompiled just like C.
* zero-cost abstractions: higher-level features that compile to lower-level code as fast as code written manually.
* Includes the official building system & packet manager **Cargo**, allowing the user to control and build dependencies.

1.2 Developer tools in Rust

* Cargo: the included dependency manager and build tool, makes adding, compiling, and managing dependencies painless and consistent across the Rust ecosystem.

Cargo has a mechanism that ensures you can rebuild the same artifact every build. Cargo will use only the versions of the dependencies you specified until you indicate otherwise. To ignore manual versions, use cargo update which will update dependencies to latest versions.

* Rustfmt: ensures a consistent coding style across developers.
* The Rust Language Server: powers Integrated Development Environment (IDE) integration for code completion and inline error messages.

1.3 Fields on Rust

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.

Crate:collection of Rust source code files(Basically a library).

The *Prelude* is a list of functionalities that Rust imports into every program, it includes: traits of fundamental types, destructors and overloading, heap allocation, ownership, clone, comparison traits, generic conversions, iterators, heap allocated strings and vectors.

1. **Rust Fundamentals**
   1. Variables

let -> Creates a variable.

**{}**  -> Curly brackets are the format specifiers(called % in C) of Rust. In Rust they are just a place holder.

Eg:println!("x = {} and y = {}", x, y);

**Shadowing:** Shadowing lets us reuse variable with same name rather than forcing us to create two unique variables,if the variables have different type.

* 1. Traits

--mut: Assigns mutable(modifiable, non-static content) attribute. In Rust variables are **immutable** by **default**. Eg: let mut guess = 5;

* 1. Compiling

|  |  |
| --- | --- |
| Instruction | Description |
| Cargo new {name} | Generates the packet manager folder for Cargo to manage your rust project |
| Cargo build  --release | Compiles rust program.  Compiles with optimizations(superfast code however is slower compilation time) |
| ./target/debug/{CargoFile} | Creates executable of the cargo project on the target |
| Cargo run |  |
| Cargo check | Check correct compiling without producing an executable(speed-up the process) |
| Use | Import library |

Figure 2.3.1 Compiling your project

* 1. Input/Output

println!() -> Macro to print a string on screen.

* 1. Strings

The String type is the most common type that has ownership over the contents of the string. Growable and UTF-8 encoded.

* 1. Methods

**instance::method**() -> ´´method´´ is an associated function of ´´instance´´ type (static method).

Eg: let guess = String::new();

instance::method.submethod() -> Calls submethod on method handle. Eg: io::stdin().read\_line()

* 1. Failures

Result types are enumerations. For Result, the variants are Ok or Err. The Ok variant indicates the operation was successful, and inside Ok is the successfully generated value. The Err variant means the operation failed, and Err contains information about how or why the operation failed. The purpose of these Result types is to encode error-handling information.

Eg: .expect(“failed”) -> Expect Unwraps a result, yielding the content of an [Ok](https://doc.rust-lang.org/std/result/enum.Result.html#variant.Ok). Otherwise panics and includes message and content of Err.

* 1. Scope **{}**

1. **Reference (&)**

**& ->** Get memory address: operator that gets the memory address(in hexadecimal) of a piece of data.

Result Enumumerator: is a type that represents either success ([Ok](https://doc.rust-lang.org/std/result/enum.Result.html#variant.Ok)) or failure ([Err](https://doc.rust-lang.org/std/result/enum.Result.html#variant.Err)). It helps to propagate errors.

pub enum Result<T, E> {

Ok(T),

Err(E),

}