RUST CHEAT SHEET

Jaideep Ganguly

Last updated on August 24, 2022

Contents

Listings

1 Introduction

1.1 Definitions

1. Packages - A Cargo feature that lets you build, test, and share crates.
2. Crates - A tree of modules that produces a library or executable.
3. Modules and use - Let you control the organization, scope, and privacy of paths.
4. Paths - A way of naming an item, such as a struct, function, or module.

1.2 Installation

```
curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh  # Install
rustup update            # Update
rustup self uninstall    # Uninstall
rustc --version          # Version
```

Listing 1: Installation

1.3 Cargo - Rust Package Manager

```
cargo new my-project        # Creating a new package with a binary crate
cargo new my-project --lib  # Creating a new package with a library crate
```

Listing 2: Creating new package with Cargo

```
[package]
name = "tpl"
version = "0.1.0"
authors = ["Jaideep Ganguly <ganguly.jaideep@gmail.com>"]
edition = "2021"

# See more keys and their definitions at
    ↪ https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
futures = "0.3.0"
tokio = {version = "0.2.*", features = ["full"]}
reqwest = {version = "0.10.0-alpha.1", features = ["json"]}
serde_json = "1.0"
rand = "0.5.5"
mysql = "*"
```

Listing 3: Example Cargo Listing

1.4 Data Types

```
bool                        // Boolean
u8, u16, u32, u64, u128 // Unsigned integers
i8, i16, i32, i64, i128 // Signed integers
f32, f64                    // Floating point numbers
usize                       // Unsiged integer, platform specific
isize                       // Signed integer, platform specific
char                        // Unicode scalar value
&str    // String literal, aka string slice, is a sequence of Unicode
    ↪ characters, value is known at compile time, static by default,
    ↪ guaranteed to be valid for the duration of the entire program.
String // Growable Mutable Collection. String object is used to represent
    ↪ string values that are provided at runtime, allocated in the heap.
```

Listing 4: Data Types

1.5 Program Structure

```
#![allow(unused_assignments)]
#![allow(unused_imports)]
#![allow(unused_variables)]
#![allow(dead_code)]
```

Listing 5: Suppress Compiler Warnings

```
pub fn fn_ex(x:i32) -> i32 {     // returns i32.
                                 // The keyword pub makes any module,
    println!("{}",x);            // function, or data structure
    return x+1;                  // accessible from inside of external
    ↪ modules.
}
```

Listing 6: Function Structure

```
mod mod_tpl;                // functions in mod_tpl.rs ; Alternately, functions
mod_tpl::typ_fn(33);        // in mod.rs under directory mod_tpl
```

Listing 7: Function Invocation

2 Ownership, Shadowing, Referencing & Lifetime

2.1 Ownership & Shadowing

```rust
pub fn variable_ex() {
    let b:bool = true;              // let introduces a variable into the
    println!("b={}",b);             // current scope
    let i:i32 = 300;

    let x = 101.25;                 // type inference

    let mut f:f64 = 3.14;           // Variables are immutable by default,
    f = 3.14159;                    // mut keyword makes it mutable.

    type NanoSecond = u64;          // type alias
    const MAX_PTS: u32 = 100_000; // constant
    static mut COUNTER: u32 = 0;   // stored in dedicated memory location

    let b = "Hello";      // Shadowing allows you to re-declare
    println!("b={}",b); // a variable in the same scope, using the same
    ↪ name. The re-declared variable differs from the original by having a
    ↪ different type. This is especially useful for casting data from one
    ↪ type into another.

    let mut s:&str = "Jaideep";
    s = "Ganguly";
    let s = String::from("Hi Jaideep!");
}
```

Listing 8: Ownership & Shadowing

2.2 Lifetime

```rust
pub fn lifetime_ex<'t>(x: &'t str, y: &'t str) -> &'t str {
    if x.bytes().len() > y.bytes().len() {
        x
    } else {
        y
    }
}
```

Listing 9: Lifetime

2.3 String Functions Struct std::string::String

```rust
pub fn string_ex(mut s:String) {
    s = String::from("Jaideep Ganguly");

    let mut litstr = "Hello Jaideep!";  // convert literal string to String
    s = litstr.to_string();
    s = s.replace("Jaideep","Mone");    // not in place, returns new String
    litstr = s.as_str();                // convert to literal

    s.push('G');                // append a char
    s.push_str("anguly");       // append a slice
    println!("{}",s.len());     // length
    println!("{}",s.trim());    // remove leading and trailing spaces
    let s = s.clear();          // clear

    let s = "Jaideep Ganguly";
    let v: Vec<&str> = s.split_whitespace().collect(); // split by whitesp.
    for token in v {
        println!("{}",token);
    }
    let s = "Jaideep.Ganguly";  // split
    let x = s.split(".");
    for token in x {
        println!("{}",token);
    }
    let x = s.chars();          // chars
    for token in x {
        println!("{}",token);
    }
    let s1 = String::from("Jaideep");   // concatenation
    let s2 = String::from("Ganguly");
    let mut s = s1 + &s2;
    let s1 = String::from("Jaideep");   // concatenation
    let s2 = String::from("Ganguly");
    s = format!("{} - {}",s1,s2);
    s = s.to_uppercase();               // uppercase
    s = s.to_lowercase();               // lowercase
}
```

Listing 10: String Functions

3 Flow

```rust
pub fn flow(x:i32) {
    if x < 10 {
        println!("{:?}", "Less than 10");
    }
    else if x > 10 {
        println!("{:?}", "Greater than 10");
    }
    else {
        println!("{:?}", "equal to 10");
    }
}
```

Listing 11: Flow

4 Loop

```rust
pub fn loop_ex() {
    for n in (1..11).step_by(2) {     // excludes 11
      println!("{}", n);
    }

    let names = vec!["Jaideep", "Ganguly"];
    for name in names.iter() {
        println!("{}", name);
    }

    let mut n = 0;
    while n < 11 {
      n += 2;
      print!("{} ", n);
    }
}
```

Listing 12: Loop

5 Data Structures

5.1 Tuple, Array, Slice

```rust
pub fn tup_arr_sli_ex() {
    let tup1: (i32,f64,String) = (10,200.32,String::from("Jai")); // tuple
    let tup2: (i32,f64,&str) = (10,200.32,"Ganguly");
    println!("{:?} {:?}",tup1, tup2);   // ? implements std::fmt::Display

    let arr1 = [1, 2, 3, 4, 5]; // // array. Must have a known length,
    println!("{:?}",arr1);      // all elements must be initialized.
    let arr2:[i32; 3] = [0; 3]; // Length determined at compile time.
    println!("{:?}",arr2);
    for item in arr2.iter().enumerate() {
        let (i,x):(usize,&i32) = item;
        println!("array[{i}] = {x}");
    }

    let slice = &arr1[1 .. 3];  // slice; length determined at runtime;
    println!("{:?}",slice);     // excludes arr1[3]
}
```

Listing 13: Tuple, Array, Slice

5.2 Struct

```rust
#[derive(Debug)]
pub struct Rect<'a> {
    pub id: &'a str,
    pub width: i32,
    pub length: i32
}

impl<'a> Rect<'a> {
    pub fn area(&self) -> i32 {
        self.width * self.length
    }

    pub fn volume(&self, height: i32) -> i32 {
        self.area()*height
    }
}

pub fn struct_ex() {
    let r = Rect {id:"id100",width:10, length:20};
    println!("{:?}",r );
    println!("Area = {}",r.area());
    println!("Volume = {}",r.volume(10));
}
```

Listing 14: Struct

6 Trait

```rust
pub trait TrAnimal {
    fn eat(&self) {
        println!("I eat grass");
    }
}

pub struct Herbivore;

impl TrAnimal for Herbivore{
    fn eat(&self) {
        println!("I eat plants");
    }
}

pub struct Carnivore;

impl TrAnimal for Carnivore {
    fn eat(&self) {
        println!("I eat meat");
    }
}
```

Listing 15: Trait

6.1 Traitbound

```rust
pub trait TrActivity {
    fn fly(&self);
}

#[derive(Debug)]
pub struct Eagle;

impl TrActivity for Eagle {
    fn fly(&self) {
        println!("{:?} is flying",&self);
    }
}

pub fn activity<T: TrActivity + std::fmt::Debug>(bird: T) {
    println!("I fly as {:?}",bird);
}
```

Listing 16: Traitbound

```rust
pub fn trait_ex() {
    use TrAnimal;
    let h = Herbivore;
    h.eat();

    let c = Carnivore;
    c.eat();

    use TrActivity;
    let eagle = Eagle;
    eagle.fly();
    activity(eagle);

    /**********************************************************
    let hen = mod_tpl::Hen;  // Compile Error because hen does
    mod_tpl::activity(hen);  // not implement TrActivity trait
     **********************************************************/
}
```

Listing 17: Trait Example

7 Generic

```rust
pub fn generic_ex() {
    struct Data<T> {
        value:T,
    }

    let t:Data<i32> = Data{value:350};                        // i32
    println!("value is :{}",t.value);

    let t2:Data<String> = Data{value:"Tom".to_string()};    // String
    println!("value is :{}",t2.value);
    // end main_generic
}
```

Listing 18: Generic

8 Enum

```rust
#[derive(Debug)]
pub enum Command {
  Quit,
  Move { x: i32, y: i32 },
  Speak(String),
  ChangeBGColor(i32, i32, i32),
}
```

Listing 19: Enum

```rust
pub fn enum_ex() {
    let msg = Command::ChangeBGColor(10, 20, 30);

    match msg {
        Command::Quit => {
            println!("{:?}",msg);
        },
        Command::ChangeBGColor(r,g,b) => {
            println!("{} {} {} ",r,g,b);
        },
        _ => {
            println!("{:?}","Non ChangeBGColor");
        }
        _ => {
            println!("{:?}","Non Quit");
        }
    }

    if let msg = Command::ChangeBGColor(10,20,30) {
        println!("{:?}", "ok");
    }
}
```

Listing 20: Enum Match

```rust
enum Option<T> {
    Some(T), // used to return a value
    None     // used to indicate null, Rust does not support null
}
```

Listing 21: Some

```rust
enum Result<T,E> {
    OK(T),
    Err(E)
}
```

Listing 22: Result

9 Collections

9.1 Vec

```rust
pub fn vec_ex() {

    let mut v = vec!["Hello","how","are","you"]; // create a vector
    v.push("today");     // push
    v.pop();             // pop
    v.insert(4, "sir"); // insert a value in a particular position
    v.remove(0);         // remove a value by its index

    let index = v.iter().position(|x| x == &"sir" ).unwrap(); // rm value
    v.remove(index);

    for i in &v {        // iterate, & required because v is moved due to
    ↪ implicit call to .into_iter()
        println!("{}", i);
    }

    for (i, elem) in v.iter().enumerate() {
        println!("Element at position {}: {:?}", i, elem);
    }
}
```

Listing 23: Vec

9.2 HashMap

```rust
use std::collections::HashMap;
pub fn hashmap_ex() {
   let mut hm: HashMap<String,String> = HashMap::new();
   hm.insert("MA".to_string(),"Massachusetts".to_string());
   hm.insert("NY".to_string(),"New York".to_string());
   hm.insert("CA".to_string(),"California".to_string());

   for (key, val) in hm.iter() {
      println!("key: {} val: {}", key, val);
   }

   *hm.get_mut("MA").unwrap() = "MASSACHUSETTS".to_string();

   hm.remove("CA");
   for (key, val) in hm.iter() {
      println!("key: {} val: {}", key, val);
   }
   println!("{:?}", hm.len());
}
```

Listing 24: HashMap

10 Closure

```rust
pub fn closure_ex1() {
    use std::thread;
    use std::time::Duration;
    let some_closure = |number: u32| -> u32 {
        println!("calculating ...");
        thread::sleep(Duration::from_secs(3));
        number + 1
    };
}

pub fn closure_ex2(x:i32) -> i32 {
    let y = 3;
    let add = |x| {
        x + y
    };

    let result = receive_closure(add, x);
    result
}

fn receive_closure<F>(f: F, x: i32) -> i32
    where F: Fn(i32) -> i32 {
        f(x) // as i32
    }
```

Listing 25: Closure

11 Error Handling

```rust
pub fn error_ex() {
    let x = 5;
    if (x > 10) {
        panic!("I am panicking, can't proceed any further");
    }
    println!("I won't print this");

    // let f = File::open("/xyz/file.txt").expect("File not found");
    let f = File::open("/Users/jaideep.ganguly/rust/src/inp.txt");
    match f {
       Ok(f) => {
          println!("file: {:?}",f);
       },
       Err(e) => {
          println!("file not found{:?}",e);    // handled error
       }
    }
    println!("I will print this");
 }
```

Listing 26: Error Handling

12 Smart Pointers

12.1 Deref

```rust
pub fn deref_ex() {
    let x = 5;
    let y = Box::new(x);
    println!("{:?}", "Checking");
    assert_eq!(5,x);    // will panic if false
    assert_eq!(5,*y);


    #[derive(Debug)]
    struct MyBox<T> { // same as: struct MyBox<T>(T);
        a: T
    }

    use std::ops::Deref;
    impl<T> Deref for MyBox<T> {
        type Target = T;

        fn deref(&self) -> &T {
            &self.a
        }
    }

    let x = MyBox{a:100};
    println!("{:?}",x);              // output: MyBox { a: 100 }
    println!("{}",*(x.deref()));     // output: 100
}
```

Listing 27: Deref

12.2 Drop

```rust
pub fn drop_ex() {
    let x = mysmaptr{ data : String::from("Hello") };
    println!("struct mysmaptr with data {}", x.data);

    struct mysmaptr {
        data: String
    }

    impl Drop for mysmaptr {
        fn drop(&mut self) {
            println!("Dropping struct mysmaptr with data {}", self.data);
        }
    }
}
```

Listing 28: Drop

13 Concurrency

13.1 Thread

```rust
use std::thread;
use std::sync::{Arc,Mutex};
use std::time::{Duration, Instant};
use std::process;
use std::sync::mpsc;
use futures::future;
use futures::join;
use futures::try_join;
use tokio::macros::support::Future;

pub fn thread_ex() {
    let handle = thread::spawn( || {
    for i in 1..10 {
    println!("Hello # {} from the spawned thread!", i);
        thread::sleep(Duration::from_millis(1));
    } });

    for i in 1..5 {
    println!("Hi # {} from the main thread!", i);
    ↪ thread::sleep(Duration::from_millis(1));
    }

    handle.join().unwrap();
}
```

Listing 29: Thread

```rust
pub fn mutex_ex() {
    let counter = Arc::new(Mutex::new(100));  // atomic ref count
    let mut handles = vec![];                 // stores refs to threads

    for _ in 0..10 {                          // spawn 10 threads
        let counter = Arc::clone(&counter);   // clone the arc
        let handle = thread::spawn( move || { // move closure
            let mut num = counter.lock().unwrap();
            *num += 1;
        });
        handles.push(handle);
    }

    for handle in handles {                   // join the threads
        handle.join().unwrap();
    }

    println!("Result: {}", *counter.lock().unwrap());
}
```

Listing 30: Mutex

```rust
pub fn msgpass_ex() {
    // Channel to send and receive messages between concurrent sections of
    ↪  code; has two halves, a transmitter and a receiver.

    let (tx, rx) = mpsc::channel();      // multiple producer, 1 consumer
    let tx2 = mpsc::Sender::clone(&tx); // clone a second producer

    // spawn a thread, move the transmitter into the closure
    // spawned thread will now own the transmitter
    thread::spawn( move || {
        let vals = vec![
            String::from("Hello"),
            String::from("from"),
            String::from("thread-1"),
        ];

        for val in vals {
            tx.send(val).unwrap();
            thread::sleep(Duration::from_secs(1));
        }
    });

    thread::spawn( move || { // same comments as above
        let vals = vec![
            String::from("Hi"),
            String::from("there"),
            String::from("thread-2"),
        ];

        for val in vals {
            tx2.send(val).unwrap();
            thread::sleep(Duration::from_secs(1));
        }
    });

    // receive the result, timeout beyond 1 sec
    let result = rx.recv_timeout(Duration::from_millis(1000));

    match result {
        Err(e) => {
            println!("{:?}",e);
            process::exit(0);
        },
        Ok(x) => {
            for received in rx {
                println!("Got: {}", received);
            }
        }
    }
}
```

Listing 31: Message Passing

```rust
pub async fn long_running_fn_1(x: &mut i32) -> i32 {
    thread::sleep(Duration::from_secs(1));
    *x = *x + 1;
    thread::sleep(Duration::from_secs(1));
    *x
}
```

Listing 32: Long running fh 1

```rust
pub async fn long_running_fn_2() -> i32 {
    thread::sleep(Duration::from_secs(4));
    42
}
```

Listing 33: Long running fh 2

```rust
    let t1 = Instant::now();
    let mut x1 = 100;
    let r1 = mod_tpl::long_running_fn_1(&mut x1).await; // Sequential exec.
    let r2 = mod_tpl::long_running_fn_2().await;
    let t2 = Instant::now();
    println!("{} {} {:?}",r1,r2,t2-t1);

    let tasks = vec![   // Concurrent execution
        tokio::spawn(async move { mod_tpl::long_running_fn_1(&mut
    ↪ x1).await}),
        tokio::spawn(async move { mod_tpl::long_running_fn_2().await }),
    ];

    let t1 = Instant::now();
    let r = futures::future::join_all(tasks).await;      // join the tasks
    let t2 = Instant::now();
    println!("{:?} {:?}",r,t2-t1);
```

Listing 34: Invoking Future

```rust
use std::fs::File;
use std::io::Write;
use std::io::Read;
use std::fs::OpenOptions;
use std::fs;

pub fn std_inp() {
    let mut line = String::new();
    println!("Please enter your name:");
    let nb = std::io::stdin().read_line(&mut line).unwrap();
    println!("Hi {}", line);
    println!("# of bytes read , {}", nb);
}

pub fn std_out() {
    let b1 = std::io::stdout()
        .write("Hi ".as_bytes()).unwrap();
    let b2 = std::io::stdout()
        .write(String::from("There\n").as_bytes()).unwrap();
    std::io::stdout().
        write(format!("#bytes written {}",(b1+b2))
            .as_bytes()).unwrap();
}
pub fn cl_arg() {
    let cmd_line = std::env::args();
    println!("# of command line arguments:{}",cmd_line.len());
    for arg in cmd_line {
        println!("{}",arg);
    }
}

pub fn file_read(filename: &str){
    let mut file = std::fs::File::open(filename).unwrap();
    let mut contents = String::new();
    file.read_to_string(&mut contents).unwrap();
    print!("{}", contents);
}

pub fn file_write(filename: &str, s: &str) {
    let mut file = std::fs::File::create(filename)
        .expect("Create failed");
    file.write_all(s.as_bytes())
        .expect("write failed");
    println!("Write completed" );
}

pub fn file_append(filename: &str, s: &str) {
    let mut file = OpenOptions::new()
```

```rust
        .append(true).open(filename)
        .expect("Failed to open file");
    file.write_all(s.as_bytes()).expect("write failure");
    println!("Appended file {}",filename);
}

pub fn file_copy(src: &str, des: &str) {
    let mut file_inp = std::fs::File::open(src).unwrap();
    let mut file_out = std::fs::File::create(des).unwrap();
    let mut buffer = [0u8; 4096];
    loop {
        let nbytes = file_inp.read(&mut buffer).unwrap();
        file_out.write(&buffer[..nbytes]).unwrap();
        if nbytes < buffer.len() {
            break;
        }
    }
}

pub fn file_delete(filename: &str) {
    fs::remove_file(filename).expect("Unable to delete file");
    println!("Deleted file {}",filename);
}
```

Listing 35: IO

15 JSON

```rust
use serde::{Deserialize, Serialize};

#[derive(Debug, Deserialize, Serialize)]
struct Person {
    name: String,
    age: usize,
    verified: bool,
}
pub fn json_ex() {
    let json = r#"
            {
                "name": "George",
                "age": 27,
                "verified": false
            }
        "#;
    let p: Person = serde_json::from_str(json).unwrap(); // JSON to Struct
    println!("{:?}", p);
    let j = serde_json::to_string(&p);                 // Struct to JSON
    println!("{:?}", j.unwrap());

}
```

Listing 36: JSON

```rust
use mysql::*;
use mysql::prelude::*;
use chrono::prelude::*; //For date and time

#[derive(Debug, PartialEq, Eq)]
struct Tab {
    cat: String,
    tsk: String
}

pub fn dbs() {

    let url = "mysql://root:root@localhost:3306/pgm";
    let opts:Opts = Opts::from_url(url).unwrap();
    let pool = Pool::new(opts).unwrap();
    let mut conn = pool.get_conn().unwrap();


    let selected_tab = conn.query_map(  // Select
        "SELECT cat, tsk FROM pgm.pgm",
        | (cat, tsk) | {
            Tab { cat,tsk }
        },
    ).unwrap();

    for r in selected_tab.iter() {
        println!("{}: {}", r.cat, r.tsk);
    }


    // let rows = vec![
    //     Tab { co1: "hi".to_string(),    co2: 2, },
    //     Tab { co1: "hello".to_string(), co2: 4, },
    // ];

    // conn.exec_batch( // insert
    //     r"INSERT INTO tpl.tab (catt, tsk)
    //     VALUES (:cat, :tsk)",
    //     rows.iter().map(|p| params! {
    //         "cat" => String::from(&p.cat),
    //         "tsk" => p.tsk,
    //     })
    // ).unwrap();


}
```

Listing 37: DB Server

17 Tonic - Rust implementation of gRPC

```
cargo new tonic_ex # A tonic example
cd tonic_ex
mkdir proto
touch proto/helloworld.proto
```

Listing 38: Creating new package with Cargo

```
syntax = "proto3";
package helloworld;

service Greeter {
    rpc SayHello (HelloRequest) returns (HelloReply);
}

message HelloRequest {
   string name = 1;
}

message HelloReply {
    string message = 1;
}
```

Listing 39: proto/helloworld.proto

```
[package]
name = "tonic_ex"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at
   ↪ https://doc.rust-lang.org/cargo/reference/manifest.html

[[bin]] # Bin to run the HelloWorld gRPC server
name = "helloworld-server"
path = "src/server.rs"

[[bin]] # Bin to run the HelloWorld gRPC client
name = "helloworld-client"
path = "src/client.rs"

[dependencies]
tonic = "0.7"
prost = "0.10"
tokio = { version = "1.0", features = ["macros", "rt-multi-thread"] }

[build-dependencies]
tonic-build = "0.7"
```

Listing 40: Cargo.toml

```rust
fn main() -> Result<(), Box<dyn std::error::Error>> {
    tonic_build::compile_protos("proto/helloworld.proto")?;
    Ok(())
}
```

Listing 41: build.rs (at root of project)

```rust
use tonic::{transport::Server, Request, Response, Status};
use hello_world::greeter_server::{Greeter, GreeterServer};
use hello_world::{HelloReply, HelloRequest};

pub mod hello_world {
    tonic::include_proto!("helloworld");
}

#[derive(Debug, Default)]
pub struct MyGreeter {}

#[tonic::async_trait]
impl Greeter for MyGreeter {
    async fn say_hello(&self,request: Request<HelloRequest>,) ->
    ↪ Result<Response<HelloReply>, Status> {
        println!("Got a request: {:?}", request);

        let reply = hello_world::HelloReply {
            message: format!("Hello {}!",
                                        request.into_inner().name).into(),
        };

        Ok(Response::new(reply))
    }
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let addr = "[::1]:50051".parse()?;
    let greeter = MyGreeter::default();

    Server::builder()
        .add_service(GreeterServer::new(greeter))
        .serve(addr)
        .await?;

    Ok(())
}
```

Listing 42: server.rs

```rust
use hello_world::greeter_client::GreeterClient;
use hello_world::HelloRequest;

pub mod hello_world {
    tonic::include_proto!("helloworld");
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let mut client = GreeterClient::connect("http://[::1]:50051").await?;
    let request = tonic::Request::new(HelloRequest {
        name: "Tonic".into(),
    });
    let response = client.say_hello(request).await?;
    println!("RESPONSE={:?}", response);
    Ok(())
}
```

Listing 43: client.rs

17.1 Java Client for Rust Service

Java Client - Rust Server connectivity using Tonic

Bibliography

[1] A Book on Rust. Moving to the Rust Programming Language by Jaideep Ganguly

[2] A Slide Deck on Rust. A Slide Deck on Rust by Jaideep Ganguly