**Rust**

Rust is a statically-typed programming language designed for performance and safety, and it's also provided code safety, memory efficient, High performance, effective compiler, support for concurrent programming, safe concurrency and memory management, memory ownership and borrowing concept, life time concept, preventing data races.

**Some other benefits:**

Pattern matching and algebraic data types (enums)

Task-based concurrency. Lightweight tasks can run in parallel without sharing any memory

Higher-order functions (closures)

Polymorphism, combining Java-like interfaces and Haskell-like type classes

Generics

No buffer overflows

Immutable by default

A non-blocking garbage collector

**data race:**

A data race occurs when at least two different instructions from different threads are trying to access the same memory location simultaneously, while at least one of them is trying to write something and there is no synchronization that could set any particular order among the various accesses. Access to the memory without synchronization is undefined.

**Concurrent programming:**

In programming terms, concurrent programming is a technique in which two or more processes start, run in an interleaved fashion through context switching and complete in an overlapping time period by managing access to shared resources e.g. on a single core of CPU.

**Asynchronous programming:**

Asynchronous programming is a method of programming that can allow multiple different things to be run concurrently (or in parallel). In Rust, it is accomplished using a high-level idea called a Future.

**Ownership, Scope and Borrowing**

Each value in Rust has a variable that is called its **owner**

There can only be **one owner** at a time

When the owner goes **out of scope**, the **value** will be **dropped**

**Variable's scope:** is range within a program for which that variable and the value are valid

**Clone:** (full copy) is one way to avoid the ownership and scope related issues

**Borrowing:** is passing the reference of that variable not pass the ownership and copy. Borrowing can be mutable but by default it’s immutable

At any given time, you can have either:

* One mutable reference, or
* Any number of immutable references

References must always be valid

* A value is no longer valid after it is dropped
* Any references to that value also become invalid when it is dropped

**String and &str**

In rust String is UTF-8 Encoded

Non-Null-Byte Terminated

String types in Rust

* String and &str
* CString and &CStr
* OsString and &OsStr

**String:** is an owned String and Data Freed after drop

Always store on heap

hold 3 parts: Length, Capasty, Pointer

**&str:** always immutable and

Hold 2 parts: Length and Pointer

**Mod and Crate**

Mod stands for module in rust we use module for organize and reuse of our code. Mod can have the nested mod and we access using :: like:

std::fs::File::write

if we write our custom mode than we can access that mod by using crate kay word like:

mod print\_things {pub fn print{}}

in main:

crate::print\_things::print()

child mode can access its parent mod data using:

super::\*

**Example code:**

mod (modules or namespace)

<https://replit.com/@abdullahshahi11/module#src/main.rs>

**Test**

we can write the test cases in rust using test macro and can run it using “cargo test”

we do not need any separate file we can write in any file at the end.

#[cfg(test)]

#[test]

#[]

#[test]

fn check\_some\_function(){

assert!(8==8);

}

we can write test in mod

#[cfg(test)]

mod tests {

use super::\*

#[test]

fn check\_some\_function(){

assert!(8==8);

}

}

**Example code:**

Test (test a calculator)

<https://replit.com/@abdullahshahi11/tsetcalculatorandtestallfunctions#src/main.rs>

**Macro**

In rust macros are special kind of function which write the code automatically and it’s work like a match statement. Macro have ‘!’ sign at the end.

**Tuesday, 22 March 2022**

**Topics:**

tuple, arrays, vectors, conditions, match, mutable, immutable, const, static

**Thursday, 24 March 2022**

**Topics:**

loops, conditions, range, result(ok, err), option( some, none), struct, enum, implementation of struct, genaric types, into, from

**Friday, 25 March 2022**

**Topics:**

HashMap, BTreeMap, HashSet, BTreeSet, BinaryHeap, VecDeque, "?" Operator, panic! macro, assert, assert\_eq, assert\_ne

**Links:**

<https://replit.com/@abdullahshahi11/panicassertasserteqassertne#src/main.rs>

<https://replit.com/@abdullahshahi11/questionmarkoperator#src/main.rs>

**Monday, 28 March 2022**

**Topics:**

traits, override buildin traits, trait bounds, chaining methods, iterators, closure, unwrap\_or with closure, foreach with closure, map with closure

**Links:**

<https://replit.com/@abdullahshahi11/matchandinput#src/main.rs>

<https://replit.com/@abdullahshahi11/simpletraits#src/main.rs>

<https://replit.com/@abdullahshahi11/implementdisplaytraitforstruct#src/main.rs>

<https://replit.com/@abdullahshahi11/traitsconstraintsselfdebugclone#src/main.rs>

<https://replit.com/@abdullahshahi11/traitbounds#src/main.rs>

<https://replit.com/@abdullahshahi11/traitsfrominvec#src/main.rs>

<https://replit.com/@abdullahshahi11/chainingmethods#src/main.rs>

<https://replit.com/@abdullahshahi11/iteratorsoverrideiteratortrait#src/main.rs>

<https://replit.com/@abdullahshahi11/closures#src/main.rs>

<https://replit.com/@abdullahshahi11/closuresunwraporandmapandforeach#src/main.rs>

**Tuesday, 29 March 2022**

**Topics:**

Make a tuple Struct, classic c struct ,unit struct , zip, char\_indices, filter, filter\_map, ok, ok\_or, ok\_or\_else, and, and\_then, any , all , rev, find, position, cycle, take, fold, chunk, window, match\_indices, peekable, peek

**Links:**

<https://replit.com/@abdullahshahi11/structtask#src/main.rs>

<https://replit.com/@abdullahshahi11/zipcollectcharindicesforeachfilter#src/main.rs>

<https://replit.com/@abdullahshahi11/filterfiltermap#src/main.rs>

<https://replit.com/@abdullahshahi11/okokorokorelse#src/main.rs>

<https://replit.com/@abdullahshahi11/andandthen#src/main.rs>

<https://replit.com/@abdullahshahi11/anyallrev#src/main.rs>

<https://replit.com/@abdullahshahi11/anyfindposition#src/main.rs:19:46>

<https://replit.com/@abdullahshahi11/cycletakefoldchunkwindow#src/main.rs>

<https://replit.com/@abdullahshahi11/matchindicespeekablepeek#src/main.rs>

**Wednesday, 30 March 2022**

**Topics:**

dbg macro, inspect, Lifetimes, Interior mutability, cell, ref\_cell, mutex, rwlock (read, write), cow (clone on write), borrowed, owned, type aliases, aliases, todo macro, rc (reference counter), RefCell inside Rc

**Links:**

<https://replit.com/@abdullahshahi11/dbginspect#src/main.rs>

<https://replit.com/@abdullahshahi11/memorylifetime#src/main.rs>

<https://replit.com/@abdullahshahi11/interiormutabilitycellrefcellmutexrwlock#src/main.rs>

<https://replit.com/@abdullahshahi11/cowborrowedowned#src/main.rs>

<https://replit.com/@abdullahshahi11/aliases#src/main.rs>

<https://replit.com/@abdullahshahi11/todomacro#src/main.rs>

<https://replit.com/@abdullahshahi11/rcandrefcallinrc#src/main.rs>

**Thursday, 31 March 2022**

**Topics:**

multiple threads, arc (atomic reference counter), Closures in functions (Fn, FnMute, FnOnce), implement trait (return closure), box, channels (sender, receiver), attributes

**Links:**

<https://replit.com/@abdullahshahi11/multiplethread#src/main.rs>

<https://replit.com/@abdullahshahi11/arcmutexrwlock1#src/main.rs>

<https://replit.com/@abdullahshahi11/arcmutexrwlock2#src/main.rs>

<https://replit.com/@abdullahshahi11/closuresinfunction#src/main.rs>

<https://replit.com/@abdullahshahi11/impltraitreturnclosure#src/main.rs>

<https://replit.com/@abdullahshahi11/box#src/main.rs>

<https://replit.com/@abdullahshahi11/channelsendereceiver#src/main.rs>

**Friday, 01 April 2022**

**Topics:**

Default, builder pattern, deref, deref mut, dref implementation, mod (modules or namespace), test (test a calculator), external creates (create.io), rand (<https://crates.io/> ), rayon (use for iter in multithread), serde (use for json parsing, Serialization and Deserialization ), regex (use regular expration), standard library (integers, floats, bool)

**Links:**

<https://replit.com/@abdullahshahi11/defaultimplementdefaulttrait#src/main.rs>

<https://replit.com/@abdullahshahi11/defaultwithbuilderpattern#src/main.rs>

<https://replit.com/@abdullahshahi11/derefderefmut#src/main.rs>

<https://replit.com/@abdullahshahi11/module#src/main.rs>

<https://replit.com/@abdullahshahi11/tsetcalculatorandtestallfunctions#src/main.rs>

<https://replit.com/@abdullahshahi11/randexternalcreates#src/main.rs>

<https://replit.com/@abdullahshahi11/rayonexternalcreates#src/main.rs>

<https://replit.com/@abdullahshahi11/standardlibraryintegersfloatsbool#src/main.rs>

**Monday, 04 April 2022**

**Topics:**

standard library (vec, string, OsString, mem, prelude, time), function pointer, function pointer with Struct,

macro ([https://cheats.rs/#macros-attributes](https://cheats.rs/" \l "macros-attributes))

(expression (expr), stringify!, token tree (tt), type (ty), item($))

**Links:**

<https://replit.com/@abdullahshahi11/standardlibraryvecstringosstringmemprelude#src/main.rs>

<https://replit.com/@abdullahshahi11/functionpointerwithrandexample#src/main.rs>

<https://replit.com/@abdullahshahi11/functionpointerwithstruct#src/main.rs>

<https://replit.com/@abdullahshahi11/macro#src/main.rs>

**Assignment**

**Topics:**

pet store program

**Links:**

[https://replit.com/@abdullahshahid137/petstore#src/main.rs](https://replit.com/@abdullahshahid137/petstore" \l "src/main.rs)

**Cargo**

Cargo.toml [dependencies] name: “version” (~0.1.0, ~0.1, ~0 and ^, <, > , <=, >=, \* (in end) )

Cargo new [project name]

Cargo new [project name] --lib

Cargo build

Cargo build --release

Cargo run

Cargo check

Cargo test

Cargo doc

Cargo publish

Rustup (ructup install [version]), (rustup install beta) (rustup default beta)

Rustup uninstall [version]

Rustup –version

Cargo expand (use for inner micro’s rust code)

**Input Program**

Use std::io::{self};

Fn main(){

Let mut input = String::new();

Peintln!(“Enter Some Text:”);

Match io::stdin().read\_line(&mut input){

Ok(\_) => println!(“Your input is: {}”, input),

Err(error) => println!(“Error: {}”, error),

}

}