Article Review

**Async/.await Rust**

Asynchronous programming might be a bit different from the basic Rust programming. Lets just try to understand what is

async programming.

Actually there are 2 basic type of programming modules:

1. Synchronous programming
2. asynchronous programming

Synchronous programming can be defined as sequential flow pf program, like what we did in the Waterfall model (if you’re a software engineering student, you might know about this particular process model) in the beginning. In synchronous programming, the tasks are performed one after another. But how it happens, its non of our concern. Lets jump to the main concept of asynchronous programming.

In rust, async means running the code simultaneous/ concurrent/ parallel, or computations are running on a single thread. Multi-threading is a bit related to this, but its a completely separate concept. When the tasks are highly CPU intensive; we go for Multi-threading. When the waiting time increases in the task, concurrent programming is ideal. These tasks are called **IO bound** tasks. **IO Bound** refers to a condition in which the time it takes to complete a computation is determined principally by the period spent waiting for input-output operations to be completed. So we can run these IO bound computations at a same time by using asynchronous programming. Its kinda OS concept that when a task is waiting, the computer is idle and you can run another task at the same time so your computer doesn’t remains idle. When all these computations are being made and we are requiring the result, we can use a function: **.await**. In rust, values that are awaitable are known as futures. Futures are single eventual values produced by asynchronous computations. Some programming languages like JavaScript call this Promises.

Rust’s async is different from what we used in other languages like c# etc. An async function does not starts executing immediately, to start an asynchronous function, you must either .await it or launch a task using an executor. Until this happens, all you have is a Future that has not started.

Basically an external library is required to do asynchronous programming in rust. You also need a dedicated executer that takes care of executing the futures (eventual values produced by aynchronous programming), giving priorities to them and taking our the results after being executed. Standard lib doesn’t includes a executer so we require a external crate for this. The two most effective cratres are:

1. async-std
2. tokio

The application which the writer is developing is a simple app that fetches some Pokemon data and prints it on the console.

* *Step 1:*

creating a new project named as async basics

* *Step 2:*

adding dependencies in the cargo.toml file (I.e: async-std, surf). **Async-Std** provides an async version of std. It provides all the interfaces we are used to, but only in an async version and ready for Rust's async/await syntax. **Surf** is a friendly HTTP client built for casual Rustaceans. It's completely modular, and built directly for async/await. Whether it's a quick script, or a cross-platform SDK, Surf will make it work.

* *Step 3:*

the final step is to make a main.rs file that fetches all the data (code is given in the article).

The writer conveys that is type of stabilized now and more work can be done in this particular feature of Rust.

Article by: Thomas Heartman

Link: <https://bit.ly/3by0GpL>

Student Name: Muhammad Ahmed Anis

Roll Number: IOT049117