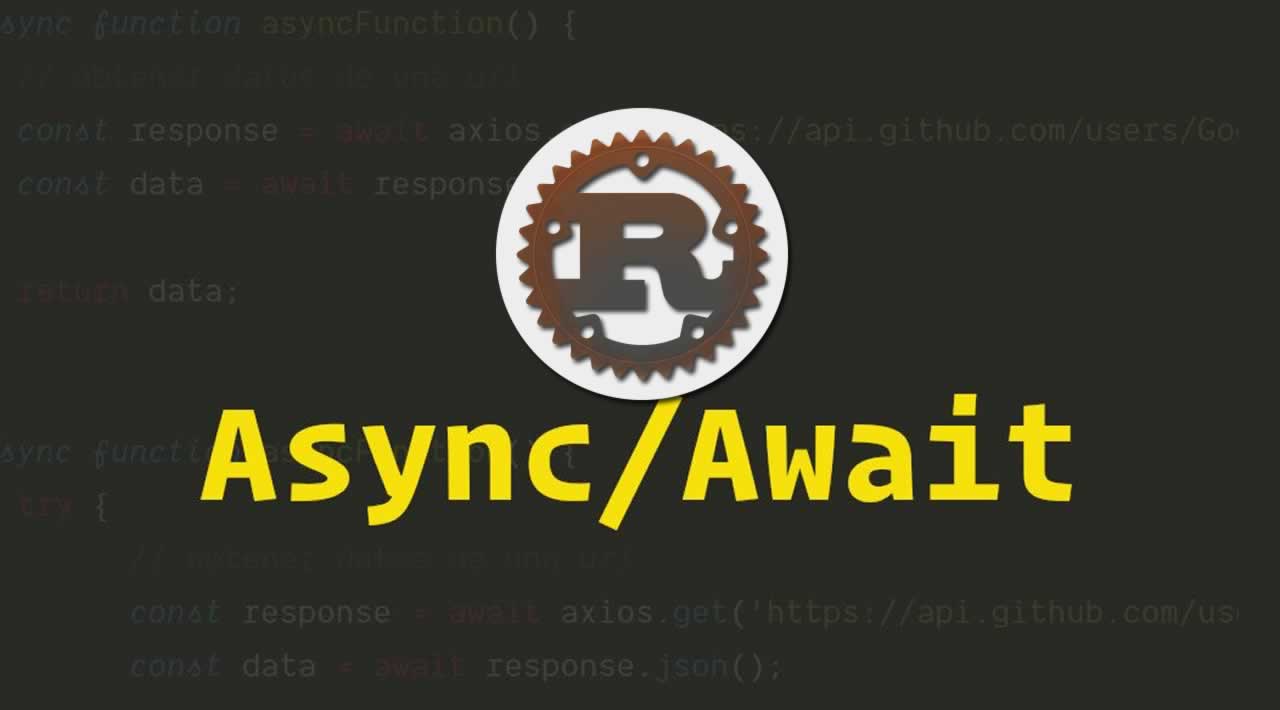
**An Article on Async Rust**



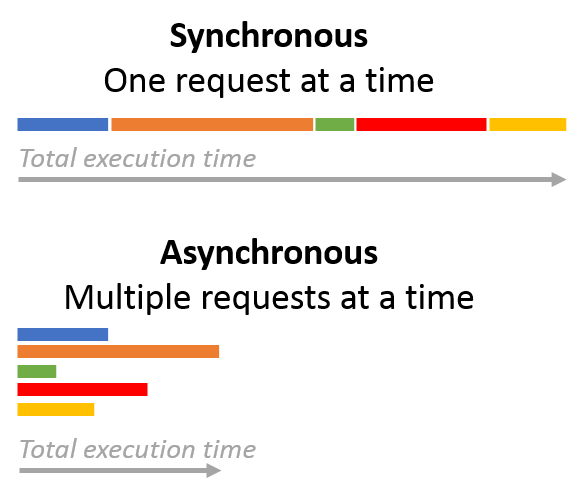
**What is async?**

Async is short for “asynchronous”. Async is a mean of running code concurrently. Also, it is meant to be multiple operations running in a time on a same OS thread.

In other words, Asynchronous programming is a parallel programming in which a unit work of an application run separately from the main application, and notifies to the calling thread after completion.

**Why Async?**

Async programming allows us to run multiple of these IO bound computations at a time on a single thread. Or we can say that, an asynchronous code let us to run several tasks simultaneously on the same OS thread.



**Async Rust**

Asynchronous coding in Rust Programming Language may be little bit different as it done in other programming languages like C# or JavaScript. In Async Rust, there is fearless concurrency while running multiple operations.

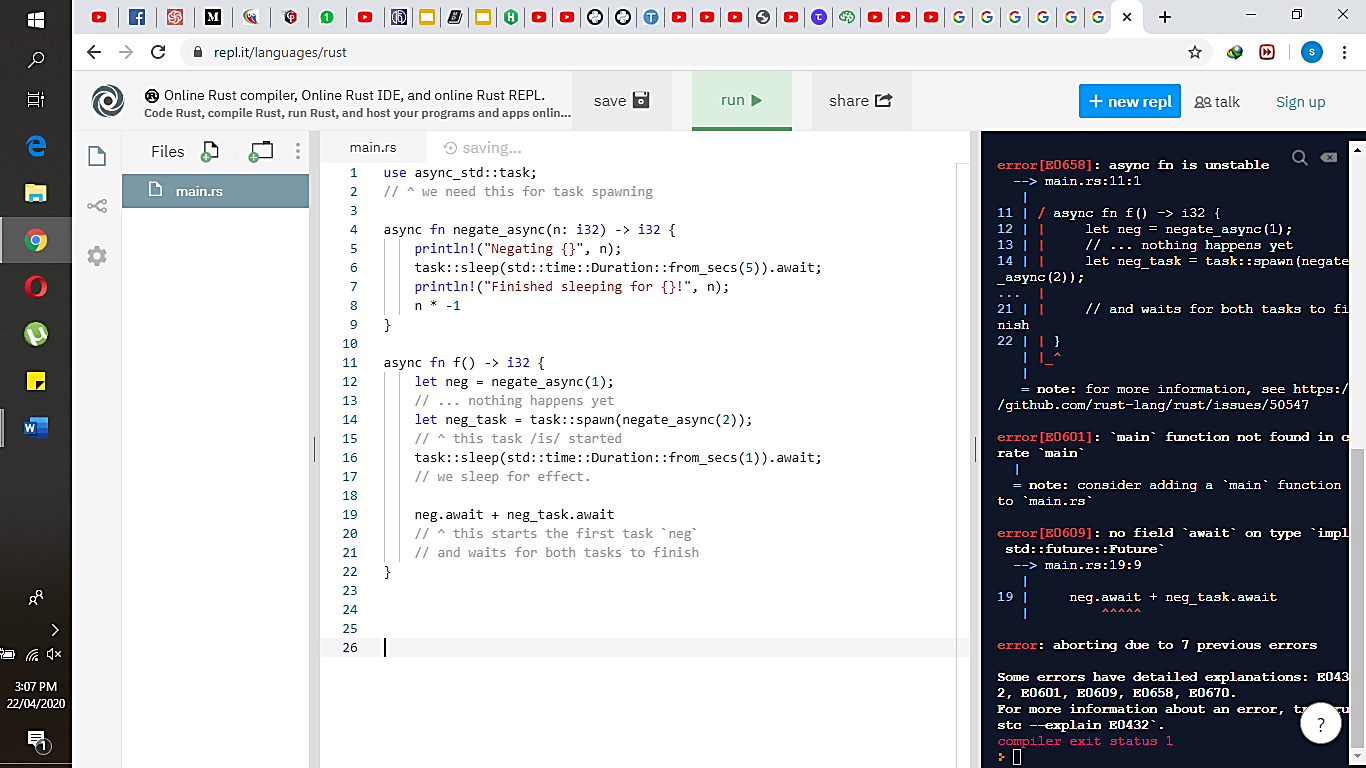
In Rust an Asynchronous function starts by .await or by launching a task using an executor.

But the standard library doesn’t come with an executer so, we need an external library to run futures. The executor takes care of executing the futures, polling them and returning the results after completion.

**What is meant by Future?**

In Rust, async fn creates an asynchronous function which returns a Future. To execute the body of the function, the returned Future must be run to completion. They aim to break code into small, composable actions that can be executed by a part of our system.

To understand it, these are few things that must be understood, so consider the following example.



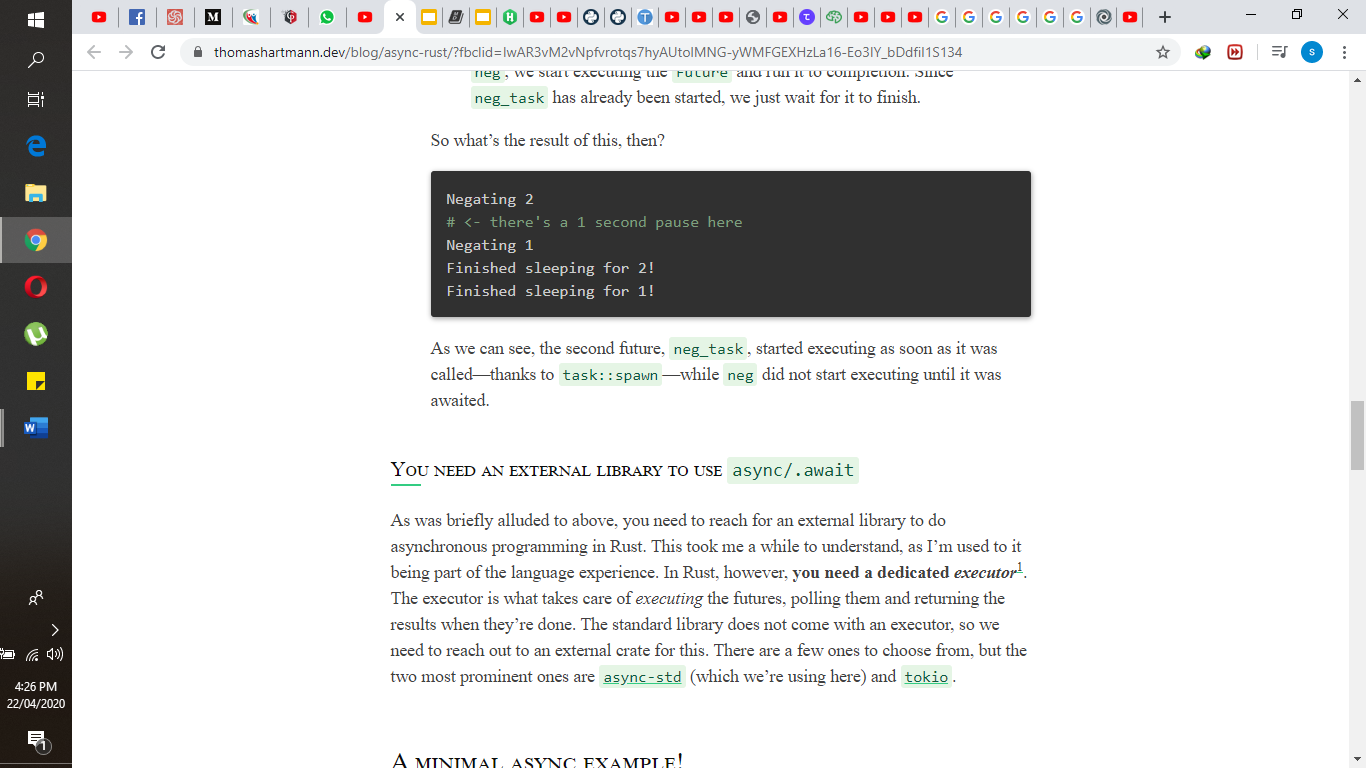
In above Code:

The first line, async\_std::task .

The async function negate\_async takes a signed integer as an input, delays for five seconds and returns the number multiplied by -1.

In async function f, the first line makes a future of negate\_async function and assign this to the variable neg. The things to be noted that it does not start executing yet. Then in the next line, the task::spawn function to start the execution of Future returned by negate\_async and assign to the neg\_task variable. Then to make it obvious before start running the task, it sleeps for a second. In the last, await both the futures and return them after add them together. From neg.await it start executing the future and by neg\_task.await it just wait for it to finish because it has already been started.

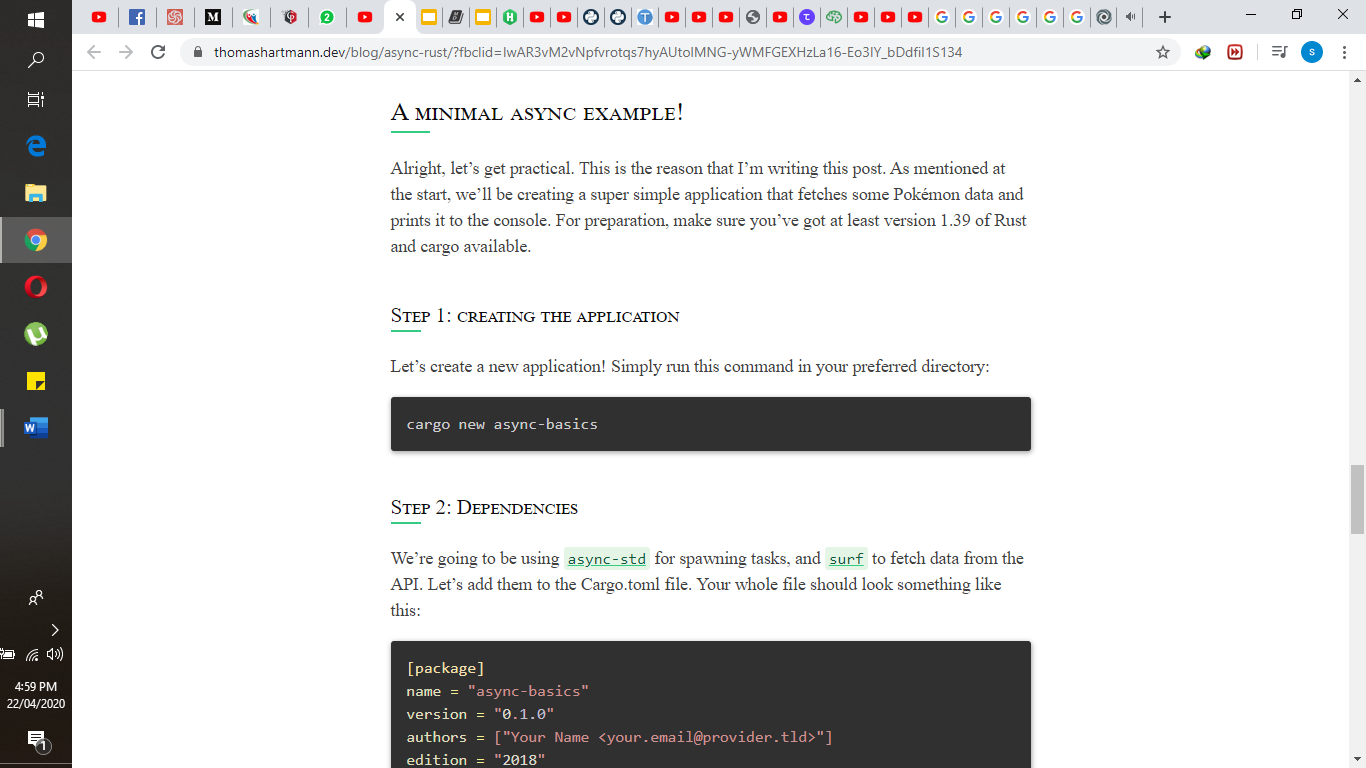
The output will be:



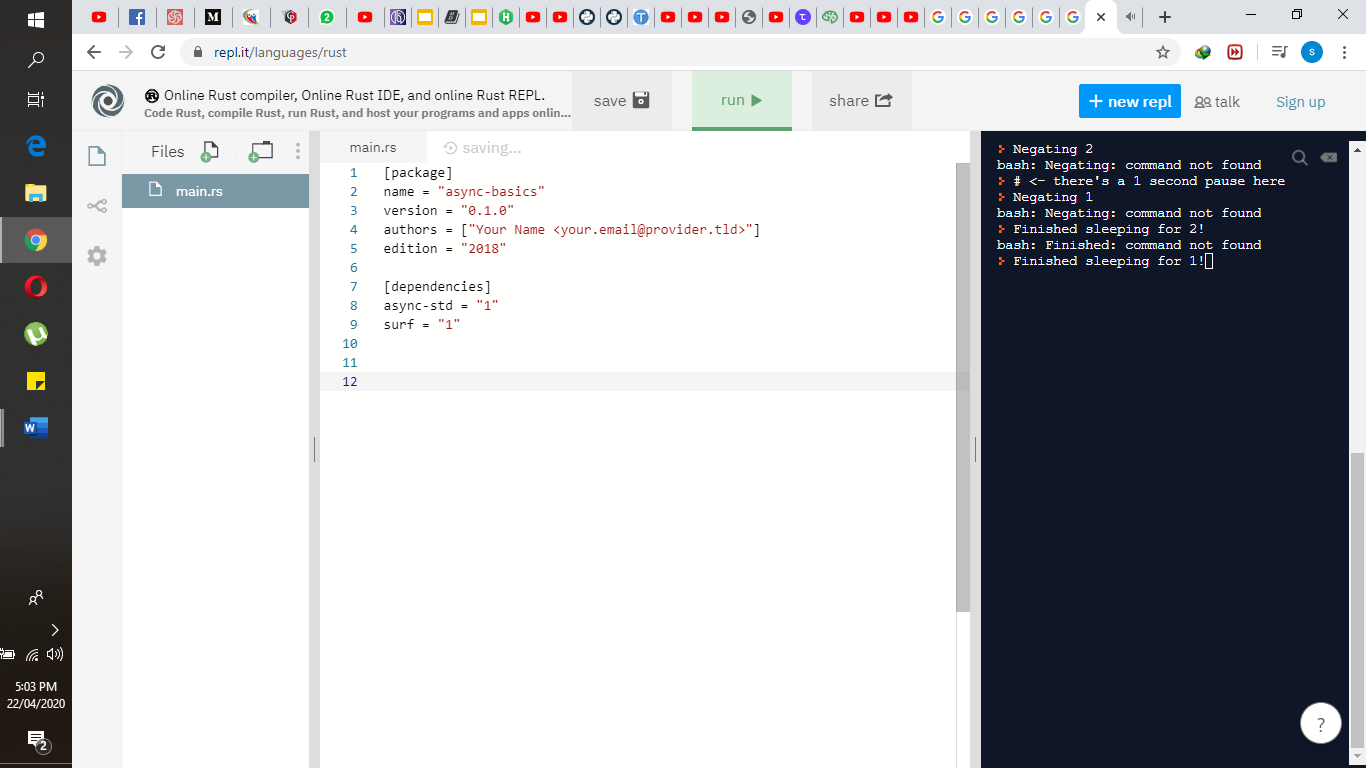
As it shows that, the second future, neg\_task, started executing as it is called because of task::spawn, while neg didn’t start executing until it was awaited.

To create a simple application that fetches some data and prints it to the console, the steps are as under:

**STEP:1** Create an application by simple running this command.



**STEP:2** Write async\_std for spawning task and surf to fetch data from the API in cargo.toml file. It should look like this.



**STEP:3** Finally, modify the main.rs file by using the following code.

