Assignment # 2

***Async Rust***

What does ***async*** mean?

In Rust, when we talk about async, we’re talking about running code concurrently, or having multiple overlapping (in time) computations run on a single thread. Multithreading is a related, but distinct concept. Multithreading is ideal for when you’ve got computationally intensive tasks (so-called *CPU-bound* tasks) that can be spread across multiple, separated cores. Concurrent programming is better suited for when the task spends a lot of time waiting, such as for a response from a server. These tasks are called *IO-bound*.

So asynchronous programming lets us run multiple of these IO-bound computations at the same time on a single thread. They can run at the same time because when they’re waiting for a response, they’re just idle, so we can let the computer keep working on something that isn’t waiting. When we reach a point where we need the result of an asynchronous computation, we must .await it. In Rust, values that are ‘awaitable’ are known as ‘futures’.

An async function does not (necessarily) start executing immediately

To start an asynchronous function, you must either .await it or launch a task using an executor (we’ll get to that in a moment). Until this happens, all you have is a Future that has not started. Let’s look at an example to make it clearer:

use **async\_std::**task;

// ^ we need this for task spawning

async fn negate\_async(n: i32) -> i32 {

**println!**("Negating {}", n);

**task::**sleep(**std::time::Duration::**from\_secs(5)).await;

**println!**("Finished sleeping for {}!", n);

n \* -1

}

async fn f() -> i32 {

let neg = negate\_async(1);

// ... nothing happens yet

let neg\_task = **task::**spawn(negate\_async(2));

// ^ this task /is/ started

**task::**sleep(**std::time::Duration::**from\_secs(1)).await;

// we sleep for effect.

neg.await + neg\_task.await

// ^ this starts the first task `neg`

// and waits for both tasks to finish

}