Async Await Notes

`async`

- Defining a function as `async`

```
async fn foo1() -> usize{
    0
}
```

results as

```
fn foo1() -> Future<Output = usize> {
    async {
            0
    }
}
```

- The `Future` return type can be thought as `promises` in javascript.

`await`

If you are awaiting for something, then the desugaring is as follows

```
fn foo1() -> Future<Output = usize> {
    async {
        let fut = read_to_string("file1").await;
    }
}
```

results to

```
fn foo1() -> Future<Output = usize> {
    async {
        let fut = read_to_string("file1");
        while !fut.is_ready() {
            std::thread::yeild_now();
            fut.try_complete();
        }
    }
}
```

Until the string with the contents of file1 isn't ready, give up a timeslice to the OS scheduler which handles the removal of the runnig process from the CPU and the selection of another process on particular strategy.

Another rustic way of represent the same thing can be

```
let fut = read_to_string("file1");
loop {
if let Some(result) = fut.try_check_completed()  {
```

```
        break result;
    } else {
        fut.try_make_progress();
        yield;
    }
}
```

First lets understand the model of the program . Lets say that each block on the figure is awating for the block downwards to do something. There is a call chain. whenever you yeild you return to top of the call stack.