Summary of Asynchronous Rust

In start Rust language do not support the Asynchronous programming, that feature was added recently, which means Asynch/.await now stabilized and we can use them.

Asynchronous programming is different from multi-threading. In Asynchronous multiple computation run on a single thread means when the task is on wait state to get a result (called futures in rust environment), thread can perform some other task that is not in waiting state. As per this functionality of Asynchronous programming it is best suited for IO-bound tasks. IO-bound are those tasks that includes the input/output functionalities and waiting of results from those tasks. Multi-threading is best suited for that tasks which are CPU-bound and spread all over the cores to perform internal functionalities. In Rust we need an external library to perform Asynch programming and two most prominent external crates are asynch-std and tokio.

Syntax for writing Asynchronous function is **asynch fn.** It impliments the Future trait and also returns the Future. **.await** is used to put the future into halt and let the thread to perform other tasks in **async fn**. Calling a blocking function writen as block_on(thread name) in a synchronous method, blocks the whole thread and wait for the future value to return.

For instance:

```
fn main () {
        block_on(async_main());
}

async fn async_main() {
        let f1 = talk();
        let f2 = laugh();
        let f3 = drive_usephone();
        futures::join!(f1, f2, f3);
}

async fn drive_usephone() {
        let cellphone = usephone().await;
        drive(cellphone).await;
}
```

In this example main function will wait for the async_main thread to execute due to block_on executor. In async_main Asynchronous function a person can talk and laugh while driving or using cellphone but can’t use the cellphone while driving. So, we have created another Asynchronous funtion called drive_usephone and await them to use the cellphone then drive.