Async Await

- When creating real world software, there are many scenarios where we need to allow the application to run multiple "tasks" concurrently
- Software often uses multiple execution "threads" to function. For example an instance of Facebook is running in a datacenter in the USA and is receiving millions of requests to send messages from one user to another. These threads are executing code independently of each other and should not become blocked if one or more other threads fail
- How does Spotify download a stream of binary from an API, process it into discrete wave-packets to be send to your AirPods, and allow you to swipe across the screen to look at other artists.

```
// Getting the thread, and putting it to sleep for that time, wait is then allowing it to run synchronically...
Task.Delay(3000).wait();

// additionally we need to add the keyword async to the function, and return a task
private static async Task<Toast> ToastBread(int slice){
        Task.Delay().wait();
        return new Toast();
}

// additionally we need to adjust the assignment, if the function is alled...
Task<Toast> newToast = ToastBread(1);

// Additionally will await the function to return
Toast newToast = await ToastBread(2);

// or to wait for the previous call, if we do want to wait for previous call
Toast newToast = await newToast;

// additionally if we want to start tasks at the same time we use, this allows us to await multiple things at the same time
Task cpuTask = Task.Run((/*...*/)=>{
        for(int i = 0; i < /*variable*/; i++){
                Thread.Sleep(1000);
                Console.WriteLine($"{i} seconds so far");
        }
});
// ...
Task cpuTask = await cpuTask;
```

Note

Mark the "Main" function, the calling function, in async as well. This will only affect it when the await keyword is used