Async Article Summary

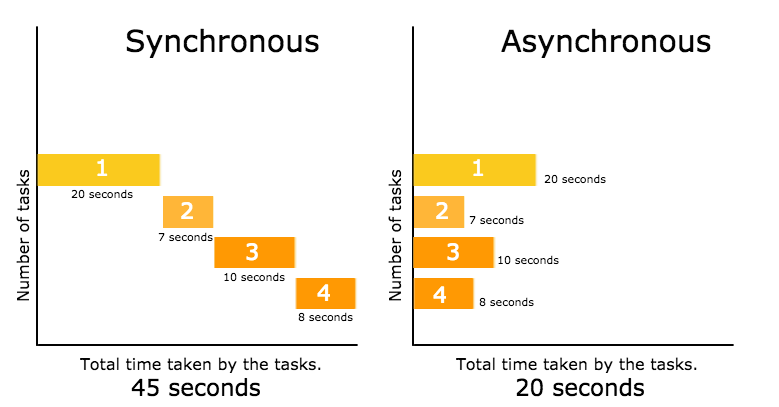
# Introduction:

This article focuses on the async feature added in Rust language and give a simple and brief explanation of how this feature works. There is also a tutorial on how to use async to fetch data from a web url.

# What does async mean?

In simple words async allows multiple tasks to run concurrently within a single thread. Meanwhile, in synchronous, no tasks overlap in processing (i.e. one task cannot start until there are no tasks currently running). This should not be confused with multithreading, as it is suitable for CPU-bound tasks, while async is suitable for IO-bound tasks.

Moreover, in async we reach a time when we need to wait for the result of an asynchronous process. This is where we use .await function in Rust. The values we get from these awaited functions are called **Futures**.

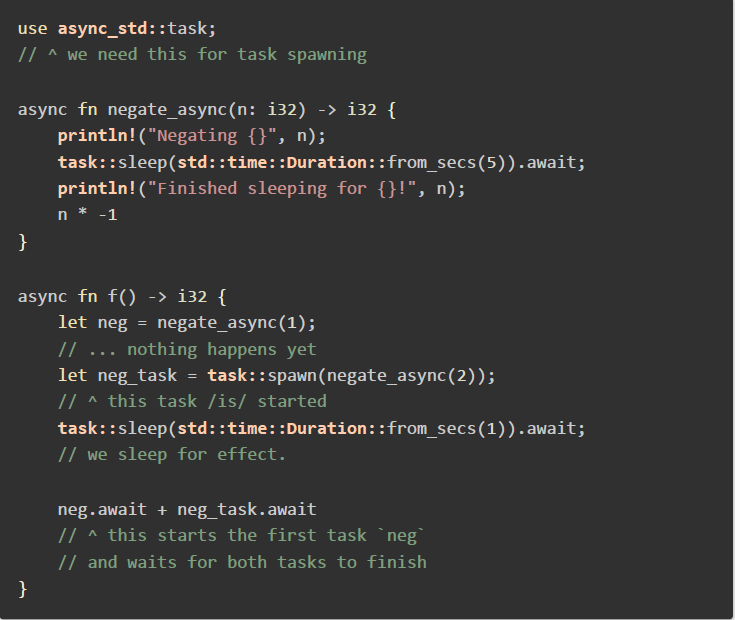
Figure below shows the comparison between Synchronous and Asynchronous processes. 

# Rust’s async is different than other languages:

## Async functions do not start right away:

We need an executor to start an async function in Rust. This means that the future will not get any value as the function has not started.

An example is given below to show how the async functions work in Rust.



In this example, the first ‘negate\_async’ function called will not start right away, therefore the future ‘neg’ will not have any value in it. On the other hand, because of the use of ‘task::spawn’ method, the future ‘neg\_task’ will get its value. In the end, the program use ‘.await’ to wait for both the async processes to finish, bringing a value to the future ‘neg’.

## You need external library for async/.await:

The standard library in Rust does not come with an executor, for this reason we need an external library. The most common as mentioned by the author are ‘async-std’ and ‘tokio’.