The Async/Await

The Rust syntax features async/await, which allow you to watch the running thread rather than interrupt it and make progress in another program while waiting for the process to finish. Async/await works on the principle of letting programmers write what appears to be standard synchronous code, but the compiler converts it to asynchronous code. The two keywords are awaited, and the async foundation is used. The synchronous function in the signature function can be changed into an asynchronous function that returns the future by using the asynchronous keyword. In synchronous functions, the await keyword might be used to retrieve the asynchronous value of the future because the async/await keywords alone would not be helpful.

Async/ await are special pieces of rust syntax that make it possible to yield control of the current thread rather than blocking , allowing other code to make progress while waiting on an operation to complete.

There are two main ways to use async : async fn and async : async fn and async blocks. Each return a value that implements the future trait :

// `foo()` returns a type that implements `Future<Output = u8>`.

// `foo().await` will result in a value of type `u8`.

async fn foo() -> u8 { 5 }

fn bar() -> impl Future<Output = u8> {

// This `async` block results in a type that implements

// `Future<Output = u8>`.

async {

let x: u8 = foo().await;

x + 5

}

}