



Async a Minimal Example

Rusty weirdness





Rust Async Programming

- We're going to be using `async-std` for spawning tasks, and `surf` to fetch data from the API. Let's add them to the `Cargo.toml` file. Your whole file should look something like this:
- `async-std = "1"`
- `surf = "1"`





Rust Async Programming

```
use async_std::task;  
  
use surf;  
  
// fetch data from a url and return the results as a string.  
// if an error occurs, return the error.  
  
async fn fetch(url: &str) -> Result<String, surf::Exception> {  
    surf::get(url).recv_string().await  
}
```





Rust Async Programming

```
// execute the fetch function and print the results
async fn execute() {
    match fetch("https://pokeapi.co/api/v2/move/surf").await {
        Ok(s) => println!("Fetched results: {:?}", s),
        Err(e) => println!("Got an error: {:?}", e),
    };
}
```





Rust Async Programming

```
fn main() {  
    task::block_on(execute());  
    // ^ start the future and wait for it to finish  
}
```





Rust Async Programming

`use` statements

Nothing exciting here. Just importing the crates we declared in the `Cargo.toml` file: `surf` and `async_std`.

`fetch`

This is simply a thin wrapper around the `surf::get` function which returns either the payload as a `String` or an `Exception` if something went wrong.





Rust Async Programming

execute

This function calls `fetch` with the endpoint for the move Surf, waits for the result to return, and then matches on the result. If everything went well: print the output. Else: print the error.





Rust Async Programming

main

main simply kicks off execute and waits for it to finish.

task::block_on is a synchronous counterpart to task::spawn that starts an asynchronous operation, but blocks until it has finished. Because the main function can't itself be async (at least not at the time of writing), we can't use .await in it, but we can block on asynchronous operations.





Try_join in Async Rust

- Polls multiple futures simultaneously, resolving to a Result containing either a tuple of the successful outputs or an error.
- **try_join!** is similar to [join!], but completes immediately if any of the futures return an error.
- This macro is only usable inside of async functions, closures, and blocks.





Try_join in Async Rust

```
use surf;
use futures::try_join;
use async_std::task;

fn main() -> Result<(), Box
```



Resources

Book : <https://rust-lang.github.io/async-book/>

Link to the article : <https://thomashartmann.dev/blog/async-rust/>

Source code repository : <https://github.com/PIAIC-IOT/Quarter3-Online.git>

Summary