Async programming with Tokio

Replacing sync versions with async versions.

Std -> Tokio (or Async-std)

Sync version

```
use std::net::TcpListener;


let listener = TcpListener::bind("127.0.0.1:8080").unwrap();
match listener.accept() {
    Ok((_socket, addr)) => println!("new client: {:?}", addr),
    Err(e) => println!("couldn't get client: {:?}", e),
}
```

Returning Errors on the command line

```
fn main() -> Result<(), Box<dyn std::error::Error>> {

    let content = std::fs::read_to_string("test.txt")?;

    println!("file content: {}", content);

    Ok(())
}
```

Expanding out ?

```
fn main() -> Result<(), Box<dyn std::error::Error>> {
    let result = std::fs::read_to_string("test.txt");
    let content = match result {
        Ok(content) => { content },
        Err(error) => { return Err(error.into()); }
    };
    println!("file content: {}", content);
    Ok(())
}
```

Introducing async functions

```
async fn say_world() {
    println!("world");
}

#[tokio::main]
async fn main() {
    // Calling `say_world()` does not execute the body of `say_world()`.
    let op = say_world();

    // This println! comes first
    println!("hello");

    // Calling `.await` on `op` starts executing `say_world`.
    op.await;
}
```

Writing a green thread -- part 1

```
use tokio::net::{TcpListener, TcpStream};

async fn process(socket: TcpStream) {
    // ...
}

#[tokio::main]
async fn main() ->  Result<(), Box<dyn std::error::Error>>  {
    let mut listener = TcpListener::bind("127.0.0.1:8080").await?;
```

Green thread – part 2

```
loop {
    let (mut socket, _) = listener. accept().await?;

// Spawns a new asynchronous task. Spawning a task enables the task to execute concurrently to other tasks. The spawned task may execute on the current thread, or it may be sent to a different thread to be executed.

    tokio::spawn(async move {
        // Process each socket concurrently.
        process(socket).await
    });
  }
}
```

Okay, lets write the complete TCP server, in fact lets write an echo server

```
use tokio::net::TcpListener;
use tokio::prelude::*;

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let mut listener = TcpListener::bind("127.0.0.1:8080").await?;

    loop {
        let (mut socket, _) = listener.accept().await?;
```

```
tokio::spawn(async move {
        let mut buf = [0; 1024];

        // In a loop, read data from the socket and write the data back.
        loop {
            let n = match socket.read(&mut buf).await {
                // socket closed
                Ok(n) if n == 0 => return,
                Ok(n) => n,
                Err(e) => {
                    eprintln!("failed to read from socket; err = {:?}", e);
                    return;
                }
            };
```

```
// Write the data back
                if let Err(e) = socket.write_all(&buf[0..n]).await {
                    eprintln!("failed to write to socket; err = {:?}", e);
                    return;
                }
            }
        });
    }
}
```