Questions:

- [x] ~~Remove Directory when file says open: step back one folder in cmd and then rmdir~~
- [ ] Cmd rustc main.rs only compiles individual files thats why it isn't working.
- [ ] Every executable Rust program must contain a function with the name: main
- [x]

  Let's say you have the following program in a file hello.rs:

  ```
  fn main() {
    println!("Hello world!");
  }
  ```

  Say you then run the command rustc hello.rs from the command-line. Which statement best describes what happens next? ___ rustc generates a binary executable named hello
- [x]

  Say you just downloaded a Cargo project, and then you run cargo run at the command-line. Which statement is NOT true about what happens next?
  (ANS) Cargo watches for file changes and re-executes the binary on a change
- [ ] Leaned about next repository
- [ ] Cargo clean and cargo build

		Cmd
	Check Version Control	rustc --version
	What is the name of the command-line tool for managing the version of Rust on your machine?	rustup
Creating a Project Directory	Creating Project directory	> mkdir "file link\projects"
	Navigate to Project Directory	> cd /d "%USERPROFILE%\projects"
	Create a New Rust Project	> mkdir hello_world
	Move Into the Project Folder	> cd hello_world
	Create source file	mkdir src
	Add code to main.rs	echo fn “code here”> src\main.rs
	Verify file exits	dir src→ main.rs
	compile and	rustc main.rs

	run the file	.\main.exe
Creating a Project with Cargo		
	check whether Cargo is installed	cargo --version
	create a new project using Cargo	cargo new hello_cargo
		cd hello_cargo
		dir
		dir src
		echo fn main() { println!("Hello, Cargo!"); } > src\main.rs
		cargo run
		Cargo build
		cargo check
		cargo add dependency_name
		cargo build --release

Chapter 2
Topics Learned
Using Cargo to Create a New Project

1. Setting up new project:

Rust projects are typically managed using Cargo, Rust's package manager and build system.

```
cargo new guessing_game
cd guessing_game
```

Project Structure

```
guessing_game/
├── Cargo.toml  # Project configuration and dependencies
└── src/
        └── main.rs  # The main Rust file where you write your code
```

2. Writing a Rust Program

Open src/main.rs and edit the existing code with:

```
fn main() {
   println!("Guess the number!");
}
```

`fn main() {}` → Defines the `main` function (Rust's entry point).
`println!()` → A macro that prints text to the console.

To compile and run the program:

```
cargo run
```

3. Guessing Game Code:

```rust
use std::io; // Import the `io` module from the standard library to handle user input.

fn main() {
    println!("Guess the number!");    // Print a welcome message to the console.

    println!("Please input your guess.");  // Prompt the user to input their guess.
    // `let` declares a variable.
    // `mut` makes the variable mutable (modifiable).
    // `String::new()` creates a new empty String instance.
    let mut guess = String::new();  // Create a mutable variable `guess` to store user input as a String.
```

```
    // Read user input from standard input (keyboard) and store it in `guess`.
    io::stdin()
        // Calls `read_line`, passing a mutable reference to `guess`.
        .read_line(&mut guess)// Reads the user's input and appends it to
`guess`
         // `expect` is used for error handling; if `read_line` fails, it will
terminate the program and print the given error message.
        .expect("Failed to read line");// If reading fails, the program will
crash with this message.

    // Uses `{}` as a placeholder for the `guess` variable in the formatted
string.
    println!("You guessed: {}", guess);// Print the user's guess back to them.
}
```

4. Accepting User Input (std::io)

`use std::io;` → Imports the io module to handle input/output.
`let mut guess = String::new();` → Creates an empty mutable String.
`io::stdin() .read_line(&mut guess)` → Reads user input from the keyboard.
`.expect("Failed to read line");` → Handles errors if input reading fails.

5. Random Number Generator exercise

```
use std::io; // Import the `io` module from the standard library to handle user
input.
use rand::Rng;// Import the `Rng` trait from the `rand` crate, enabling random
number generation.

fn main() {  // Defines the main function, the entry point of the program.
    println!("Guess the number!");    // Print a welcome message to the console.
    // `let` declares an immutable variable `secret_number`.
    // `rand::thread_rng()` creates a random number generator instance.
    // `.gen_range(1..=100)` generates a random number between 1 and 100
(inclusive).
    let secret_number = rand::thread_rng().gen_range(1..=100);
    println!("The secret number is: {secret_number}");  // Uses
`{secret_number}` to print the randomly generated number.

    println!("Please input your guess.");  // Prompt the user to input their
guess.
```

```
    // `let` declares a variable.
    // `mut` makes the variable mutable (modifiable).
    // `String::new()` creates a new empty String instance.
    let mut guess = String::new();  // Create a mutable variable `guess` to
store user input as a String.

    // Read user input from standard input (keyboard) and store it in `guess`.
    io::stdin()
        // Calls `read_line`, passing a mutable reference to `guess`.
        .read_line(&mut guess)// Reads the user's input and appends it to
`guess`
         // `expect` is used for error handling; if `read_line` fails, it will
terminate the program and print the given error message.
        .expect("Failed to read line");// If reading fails, the program will
crash with this message.

    // Uses `{}` as a placeholder for the `guess` variable in the formatted
string.
    println!("You guessed: {}", guess);// Print the user's guess back to them.
}
```

6.
7. Generating a Random Number (rand Crate)

Rust does not include random number generation in its standard library, so we need to add an external crate. So we Add rand to Cargo.toml

`cargo add rand`

So now, the toml file will have:

```
[dependencies]
rand = "0.8.5"
```

Modify main.rs to Generate a Random Number by:

`rand::thread_rng()` → Creates a random number generator.
`.gen_range(1..=100)` → Generates a number between 1 and 100.

8. Comparing the User Guess to the Secret Number

```
use rand::Rng; // Import the `Rng` trait from the `rand` crate, enabling random
number generation.
use std::cmp::Ordering; // Allows comparison between numbers.
use std::io; // Import the `io` module from the standard library to handle user
input.
```

```rust
fn main() {  // Defines the main function, the entry point of the program.
    println!("Guess the number!"); // Print a welcome message to the console.

    // Generate a secret random number between 1 and 100.
    let secret_number = rand::thread_rng().gen_range(1..=100);

    println!("The secret number is: {}", secret_number); // Print the secret number for debugging.

    loop {
        println!("Please input your guess.");  // Prompt the user to input their guess.

        let mut guess = String::new();  // Create a mutable variable `guess` to store user input as a String.

        // Read user input from standard input (keyboard) and store it in `guess`.
        io::stdin()
            .read_line(&mut guess) // Reads the user's input and appends it to `guess`
            .expect("Failed to read line"); // If reading fails, the program will crash with this message.

        // Convert input string into a number (trim whitespace and parse as u32).
        let guess: u32 = match guess.trim().parse() {
            Ok(num) => num, // If parsing is successful, use the number.
            Err(_) => {
                println!("Please enter a valid number!"); // Handle invalid input.
                continue; // Restart the loop.
            }
        };

        println!("You guessed: {}", guess); // Display the user's guess.

        // Compare the user's guess with the secret number.
```

```
        match guess.cmp(&secret_number) {
            Ordering::Less => println!("Too small!"), // If the guess is lower.
            Ordering::Greater => println!("Too big!"), // If the guess is
higher.
            Ordering::Equal => {
                println!("You win!"); // If the guess is correct.
                break; // Exit the loop.
            }
        }
    }
}
```

use std::cmp::Ordering; → Imports Ordering which has:
Ordering::Less (guess too low)
Ordering::Greater (guess too high)
Ordering::Equal (guess correct)
Looping Until Correct
loop {} creates an infinite loop.
continue; restarts the loop if input is invalid.
break; exits the loop when the guess is correct.
Converting Input to a Number
trim().parse() converts the string into a u32 integer.
match guess.trim().parse() handles invalid input gracefully.

9. Handling Invalid Input Gracefully

Right now, if the user types non-numeric input, the game crashes. Instead, we handle it using:

```
et guess: u32 = match guess.trim().parse() {
            Ok(num) => num, // If parsing is successful, use the number.
            Err(_) => {
                println!("Please enter a valid number!"); // Handle invalid
input.
                continue; // Restart the loop.
    }
        };
```

So if a User types in a different data type program will not crash

Cargo commands:

```
    add                Add dependencies to a Cargo.toml manifest file
    b                  alias: build
    bench              Execute all benchmarks of a local package
    build              Compile a local package and all of its dependencies
    c                  alias: check
    check              Check a local package and all of its dependencies for errors
    clean              Remove artifacts that cargo has generated in the past
    clippy             Checks a package to catch common mistakes and improve your Rust code.
    config             Inspect configuration values
    d                  alias: doc
    doc                Build a package's documentation
    fetch              Fetch dependencies of a package from the network
    fix                Automatically fix lint warnings reported by rustc
    fmt                Formats all bin and lib files of the current crate using rustfmt.
    generate-lockfile  Generate the lockfile for a package
    git-checkout       REMOVED: This command has been removed
    help               Displays help for a cargo subcommand
    info               Display information about a package in the registry
    init               Create a new cargo package in an existing directory
    install            Install a Rust binary
    locate-project     Print a JSON representation of a Cargo.toml file's location
    login              Log in to a registry.
    logout             Remove an API token from the registry locally
    metadata           Output the resolved dependencies of a package, the concrete used versions
including overrides, in machine-readable format
    miri
    new                Create a new cargo package at <path>
    owner              Manage the owners of a crate on the registry
    package            Assemble the local package into a distributable tarball
    pkgid              Print a fully qualified package specification
    publish            Upload a package to the registry
    r                  alias: run
    read-manifest      DEPRECATED: Print a JSON representation of a Cargo.toml manifest.
    remove             Remove dependencies from a Cargo.toml manifest file
    report             Generate and display various kinds of reports
    rm                 alias: remove
    run                Run a binary or example of the local package
    rustc              Compile a package, and pass extra options to the compiler
    rustdoc            Build a package's documentation, using specified custom flags.
    search             Search packages in the registry. Default registry is crates.io
    t                  alias: test
```

test Execute all unit and integration tests and build examples of a local package
tree Display a tree visualization of a dependency graph
uninstall Remove a Rust binary
update Update dependencies as recorded in the local lock file
vendor Vendor all dependencies for a project locally
verify-project DEPRECATED: Check correctness of crate manifest.
version Show version information
yank Remove a pushed crate from the index