Day 01

⚠ Important Note

We use `clone()` in this example, but we are not claiming it is the optimal approach. The goal here is to present a solution that helps illustrate Rust's ownership system in a simple way. As the story unfolds in the next lesson, we'll dive into more efficient patterns. This example is intentionally simplified for the sake of education and storytelling.

Enjoy!

The Story

It's 1st December and the countdown has just begun. The elves are busy preparing for the Christmas and Santa is busy checking the list of children who have been good this year. It was supposed to be a smooth day until all of a sudden two of santa's elves burst into Santa's office with a problem.

The Story

"Santa!" one of the elves shouted. "The code won't compile! We've hit a wall, and it's all Rust's fault!"

Santa, sipping his triple-shot peppermint latte, raised an eyebrow. "Rust's fault? Or your fault?"

"It's the ownership rules!" the other elve blurted. "I think we violated them, we're used to Python, where variables just... work. Look at this!"

The Story

The elves tried their best, here is what they've written so far:

```
pub fn main() {
    let gift_message = String::from("Merry Christmas! Enjoy your gift!");
    attach_message_to_present(gift_message);

    println!("{}", gift_message);
}

pub fn attach_message_to_present(message: String) {
    println!("The present now has this message: {}", message);
}
```

The Story

However, the code won't compile. Can you help the elves attach the message to the present and print the message too?

Maybe... if there was only a way to get a clone of the message and pass it to the function. 🤔

Hints

- The elves want you to only use `clone()` to fix the code.