Day 1 - Function Exercise

Exercise 1: Basic Function

Task: Create a function called `greet` that:

1. Takes a name (&str) as parameter
2. Returns a String in the format "Hello, {name}!"
3. Call it with your name and print the result

```rust
fn main() {
    // Call your function here
}

```

Exercise 2: Multiple Parameters

Task: Write a function `calculate_area` that:

1. Takes width and height (both f64)
2. Returns the area (width * height)
3. Handle the case where either parameter is negative by returning 0.0

```rust
fn main() {
    println!("Area: {}", calculate_area(3.5, 4.2));
    println!("Area (negative): {}", calculate_area(-1.0, 5.0));
}

```

Exercise 3: Function with Conditional Logic

Task: Create a function `is_even` that:

1. Takes an integer (i32)
2. Returns a bool (true if even, false if odd)
3. Use an expression body (no return keyword)

```rust
fn main() {
    println!("Is 4 even? {}", is_even(4));
    println!("Is 7 even? {}", is_even(7));
}

```

Exercise 4: Recursive Function

Task: Implement a recursive `factorial` function that:

1. Takes an unsigned integer (u32)
2. Returns its factorial (n! = n × (n-1) × ... × 1)
3. Handle the base case (0! = 1)

```rust
fn main() {
    println!("5! = {}", factorial(5));
    println!("0! = {}", factorial(0));
}

```

Exercise 5: Generic Function

Task: Write a function `find_max` that:

1. Takes a slice of any type that implements PartialOrd
2. Returns an Option<&T> (Some with max value, or None if empty)
3. Test it with both integers and strings

```rust

```

```rust
fn main() {
    let numbers = vec![34, 12, 78, 3];
    let words = vec!["apple", "banana", "cherry"];

    println!("Max number: {:?}", find_max(&numbers));
    println!("Max word: {:?}", find_max(&words));
}

```

Bonus Exercise: Higher-Order Function

Task: Create a function `apply_twice` that:

1. Takes a function and a value
2. Applies the function to the value twice
3. Test it with a simple increment function

```rust
fn main() {
    let increment = |x| x + 1;
    println!("Apply twice: {}", apply_twice(increment, 5));
}

```

Each exercise focuses on different Rust function features:

1. Basic declaration and string handling
2. Parameter handling and simple validation
3. Expression-bodied functions
4. Recursion and base cases
5. Generics and trait bounds

Bonus: Function pointers/closures