Core complexity and Data structures

Exercise 1: Lifetime

Objective: Practice using lifetime annotations to ensure references remain valid.

Task: Write a function longest_common_suffix that takes two string slices and returns the longest common suffix. If there is no common suffix, return an empty string slice.

Requirements:

- Use lifetime annotations.
- Handle edge cases (e.g., empty strings, no common suffix). **Method/Function Information:**
- Use str::chars() to iterate over characters.
- Use str::rev() to reverse the string for easier suffix comparison.

Exercise 2: Structs

Objective: Implement a struct and associated methods.

Task: Define a struct Product with fields name, price, and stock. Implement a method apply_discount that reduces the price by a given percentage.

Requirements:

- Use an associated function to create a new Product instance.
- Implement the apply_discount method. **Method/Function Information:**
- Use f64 for the price to handle decimal values.
- Use self.price *= (1.0 - discount) to apply the discount.

Exercise 3: Generics

Objective: Use generic types to create a flexible function.

Task: Write a generic function swap that swaps the values of two variables of the same type.

Requirements:

- Use generic type parameters. **Method/Function Information:**
- Use a tuple to return the swapped values.

Exercise 4: Option

Objective: Practice handling optional values.

Task: Write a function find_index that takes a vector of integers and a target value. Return the index of the target value as Some(index) if found, otherwise return None.
Requirements:

- Use Option<usize> as the return type. **Method/Function Information:**
- Use iter().enumerate() to iterate over the vector with indices.

Exercise 5: Result

Objective: Practice error handling with Result.

Task: Write a function safe_sqrt that takes a f64 and returns the square root as Ok(f64) if the input is non-negative, otherwise return an Err(String).

Requirements:

- Use Result<f64, String> as the return type. **Method/Function Information:**
- Use f64::sqrt() to compute the square root.
- Return Err with a message if the input is negative.

Exercise: Financial Portfolio Analysis

Objective: Use iterators and data structures to analyze a financial portfolio. Calculate the total value, diversification score, and identify high-risk assets.

Task: You are given a dataset of financial assets in a portfolio. Each asset includes a ticker symbol, quantity, price per unit, and risk level (low, medium, high). Your task is to:

1. Calculate the total value of the portfolio.
2. Calculate a diversification score (number of unique asset classes).
3. Identify assets with a "high" risk level.
4. Filter out assets with a value below a specified threshold (e.g., 1000.0).

Requirements:

- Use HashMap to aggregate the total value by asset class.
- Use iterators (map, filter, fold) to process the data.
- Choose between HashMap or BTreeMap for aggregation.

Output :

```
Total portfolio value: 12700.00
Diversification score (unique assets): 4
High-risk assets: [Asset { ticker: "TSLA", quantity: 5, price: 700.0, risk: High }, Asset {
ticker: "NVDA", quantity: 4, price: 200.0, risk: High }]
Assets with value >= 1000.0: [Asset { ticker: "TSLA", quantity: 5, price: 700.0, risk: High },
Asset { ticker: "AMZN", quantity: 2, price: 3000.0, risk: Medium }, Asset { ticker: "GOOGL",
quantity: 3, price: 2500.0, risk: Medium }]
```