Student Name: _____ Student ID: _____ TA: _____

Rust Lab 04 23/7/2025

1. Vector Practice

1.1) Create and Display - Write code that:

- Creates a vector with these test scores: 85, 92, 78, 96, 88, 73, 91, 84
- Prints all the scores
- Prints how many scores there are

TA Checking: _____

1.2) Find Information - Using the same vector of scores:

- Find and print the highest score
- Find and print the lowest score
- Check if there's a score of 90 in the list
- Print the first score and the last score

TA Checking: _____

1.3) Modify the Vector - Starting with your original vector:

- Add a new score of 87 to the end
- Remove the last score
- Sort all scores from lowest to highest
- Print the updated vector

TA Checking: _____

1.4) Filter and Count - Work with your vector to:

- Count how many scores are 85 or higher
- Create a new vector containing only scores above 80
- Remove all scores below 75 from your original vector

TA Checking: _____

Hints - You might need these vector methods:

- `push()` - adds an element - `pop()` - removes last element - `len()` - gets the size
- `sort()` - sorts the vector - `iter().max()` - finds largest value - `iter().min()` - finds smallest value
- `contains()` - checks if value exists - `retain()` - keeps only elements that match a condition

2. Warehouse Inventory Management

Develop a Rust program to manage inventory in a warehouse. The system should track products using tuples (ID, name, quantity) and store them in a vector.

Requirements

1.1) Product Representation: Each product is represented by a tuple: (u32, String, u32) for (ID, name, quantity).

1.2) Inventory Management:

- Use a vector to store the list of products in the warehouse.
- Implement the following functionalities:
  1) Add a new product (ensure unique ID).
  2) Update the stock quantity of an existing product.
  3) Remove a product by its ID.
  4) List all products in the inventory
  5) Exit program

1.3) User Interaction: Create a menu-driven interface where users can select operations (e.g., 1 for add, 2 for update, etc.).

1.4) Bonus Points (Optional): Input Validation: Implement robust error handling for invalid user inputs (e.g., non-numeric input, invalid product IDs).

1.5) Example Menu:

Warehouse Inventory Management:
1. Add New Product
2. Update Stock Quantity
3. Remove Product
4. List All Products
5. Exit
Enter your choice:

Hints: vector of tuple: `let mut warehouse: Vec<(u32, String, u32)> = Vec::new();`

Id check and adding:

```
let mut id_exists = false;
for (existing_id, _, _) in &warehouse { // Iterate through the warehouse
   if *existing_id == id { // Check if the ID already exists
      id_exists = true;
      break; // Exit the loop early if found
   }
}
```

TA Comment: _____ O Check if error handling validation