Final Project: Functional Programming
2025 - 2026

1. Description

This project focuses on modeling an e-commerce system, placing a strong emphasis on data type structure and the application of complex business rules through pure functions. The goal is to model and manage orders in an e-commerce system, including the catalog, customers, shopping carts, and final orders. Key concepts:

- Product: Has a unique ID, name, price, and category (Category).
- Category: Could be Electronics, Books, Clothing, Groceries.
- Customer: Has an ID, name, and a loyalty level (Loyalty Level).
- Loyalty Level: Can be Bronze (no discount), Silver (5% discount), Gold (10% discount).
- Cart Item: Contains a Product and a quantity (Int).
- Shopping Cart: A list of Cart Items.
- Stock: Contains a list of Products and their available quantity (Int).
- Order: Contains a Customer, a Shopping Cart, the calculated total price, and the status (Order Status).
- Order Status: Can be Pending, Processing, Shipped, Delivered, Cancelled.
- Search Criterion: orders can be searched by Customer ID, by Loyalty Level, by Product ID, by Category or by total price.

2. Work to Perform

1. Exhaustive Type Definition: Define exhaustively all the former concepts, using the most appropriate way (type, data, newtype) for each of them. Ensure that aliases are defined as needed to make arguments of the new types easier to understand.
2. Pretty Printing Functions: Functions to show the former concepts in a clear and readable way by implementing the `Show` Type Class.
3. Price Calculation: Implement `calculateProductPrice :: Product -> Float` that calculates the price of a Product.
4. Discount Application: Define a `Discountable` Type Class with a method `applyDiscount Discountable a => a -> Float -> Float` and make `LoyaltyLevel` and `Category` instances.
   - Implement `applyDiscount :: LoyaltyLevel -> Float -> Float` that receives the original price and returns the price with the customer's discount.
   - Implement an additional function `applyDiscount:: Category -> Float -> Float` to apply a 15% discount to products in the Books category.
   - The final function `calculateOrderTotal :: Order -> Float` must first apply the category discount (if applicable) and then the customer's loyalty discount.

5. Purchase: `Implement addToCart :: CartItem -> Cart -> Cart` that adds the given quantity of the Product to the Cart, returning the updated Cart. If the Product is already in the Cart, it will increase the quantity.
6. Cart Validation: Implement `checkStock :: Cart -> [Product]` which, given a global list of stock, returns the list of products in the cart for which there is insufficient stock.
7. Order Creation: Create a function `createOrder :: Customer -> Cart -> Either Error Order` that checks the stock. If stock is insufficient, returns an Error with the list of missing products. If stock is available, creates the Order with the Pending status and the calculated total.
8. Status Update: Implement `updateOrderStatus :: OrderStatus -> OrderStatus -> Maybe Order`. This function should only allow logical state transitions (e.g., Shipped to Delivered, but never Delivered to Pending). If the transition is invalid, return Nothing.
9. Polymorphic Search: Implement `searchOrders :: [SearchCriterion] -> [Order] -> [Order]` that filters the list of orders. The criteria should be a list of descriptors (e.g., `[ByStatus Processing, ByCustomerName "John"]`).
10. Customer management: Implement `highValueCustomers :: [Order] -> Float -> [Customer]` that returns the list of customers who have active orders (not Delivered or Cancelled) totaling more than the second parameter.
11. Functions to use I/O to allow the user to interact with the system. He/she will be able to search by Product, by Category or by product maximum price, to add elements to the Cart and to place an Order.
12. Functions to use I/O to manage the shop. The shop owner will be able to search by user ID, by Loyalty Level or by high value customers, and to process and ship all the pending orders.

    The former functions must work for any input of the right type, including empty lists where applicable. Any auxiliary function considered necessary can be created.

3. Delivery Rules

The following rules are **compulsory.** Not following them can result in the work not being considered for evaluation. The final project must be performed in groups of two students. Each group must upload to Aula Global an ipynb file containing the source code before the deadline. The name of the file must be "name-initials1-name-initials2.zip" (for example if the students are Lucía Pérez Gómez and Juan García Jiménez, the file will be named lpg-jgj.ipynb). Both students must upload the file.

In addition to the code, the file must include text cells that with at least:

- A brief summary of the document.
- Description of the types created.
- General description of the functions.
- Description of the work performed, functionalities included, parts not performed.
- Conclusions:

- Final summary of the work performed.
- Main problems found when implementing the program.
- Personal comments and feedback, including criticism of the final project.

4. Evaluation

The final project will be evaluated from 0 to 10 points.

The following items will be considered to mark the final project:

- Appropriate types design.
- Reimplementation of functions in the derived types.
- Correct execution of the program and compliance with the guidelines stated by this document.
- Quality of the implemented code.
- Use of the syllabus of the course (ask before using any function not presented during the course).
- Inclusion of code comments.
- General description of the implemented code in the documentation. (Section 3).
- Adhesion to the delivery rules (Section 3).

The following penalties will be also applied:

- 1 point if not having enough comments.
- 1 point if functions' types are not specified or are too generic.

Copies will be seriously penalized: both copying and copied groups will be excluded from continuous evaluation in addition to any other administrative measure that the University could take. Two final projects will be considered a copy even if the similarity between them is limited to a single function. Automatic tools will be used to check for plagiarisms.

The practical exercise will be corrected in an oral presentation. Both students will have to defend their work in front of the professor.