Part 1
In the initial part, the Product class is created, which is simply a model for a product. It stores its ID, name, and price. Additionally, it has a function to apply a discount to the price and another function to display the product in an orderly way when printed.

Part 2
Next, the ProductRepository interface is defined, which outlines the functions that any product repository should have. It specifies three mandatory actions:

- saving a product,
- finding a product by its ID,
- finding products that are more expensive than a given minimum price.
  The implementation details of these actions are not provided here; only the fact that these functions must exist is defined. This way, we can later create different methods for storing products without changing the rest of the program.

Part 3
A real implementation of the repository is then created, called InMemoryProductRepository, which stores products in memory. It starts by loading four example products. It implements the three functions promised in the interface: save, find_by_id, and find_expensive_products to find products that cost more than a specified price.

Part 4
This part creates the PricingService, which is a business service responsible for applying discounts. It takes a product repository to work with the products. It has a function that finds all expensive products, applies a discount, and then saves the changes. This way, the service focuses only on the business logic and not on how or where the products are stored.

Result
Finally, this part shows how everything we've built works together.
First, a product repository in memory and a pricing service that uses that repository are created. Then, all products are printed before any discount is applied.

Next, a 10% discount is applied to the expensive products (those costing more than $500). Finally, the expensive products are printed again to verify that the discount has been correctly applied. This part ties everything together and demonstrates how the Repository Pattern works in practice.