**Exercise 1: Advanced Payroll System with Overtime Calculation**

**Objectives:**

* Understand object-oriented programming (OOP) in Python.
* Implement classes and methods.
* Use inheritance to extend functionality.

**Business Scenario:**

You are tasked with enhancing the payroll system to handle overtime pay. Employees should receive 1.5 times their hourly rate for any hours worked beyond 40 in a week.

**Tasks:**

1. Create a new Python file named **advanced\_payroll\_system.py**.
2. Define a class **Employee** with attributes **name**, **hours\_worked**, and **hourly\_rate**.
3. Implement a method **calculate\_pay()** that calculates the total pay, including overtime pay.
4. Create a subclass **Manager** that has an additional attribute **bonus**.
5. Override the **calculate\_pay()** method in the **Manager** class to include the bonus in the total pay.
6. Instantiate objects for both **Employee** and **Manager** and calculate their pay.
7. Print the total pay for both.

**Exercise 2: Dynamic Order Processing System with Discount Strategy**

**Objectives:**

* Use design patterns (Strategy Pattern) to implement dynamic behavior.
* Apply polymorphism and interfaces.

**Business Scenario:**

You need to implement a dynamic order processing system where different discount strategies can be applied based on the type of customer (e.g., regular, premium, VIP).

**Tasks:**

1. Create a new Python file named **dynamic\_order\_processing.py**.
2. Define an interface **DiscountStrategy** with a method **apply\_discount(order\_amount)**.
3. Implement different discount strategies (**RegularDiscount, PremiumDiscount, VIPDiscount**) that implement the **DiscountStrategy** interface.
4. Define a class **Order** that has attributes **customer\_type** and **order\_amount**.
5. Implement a method **final\_price()** in the Order class that applies the appropriate discount strategy based on **customer\_type**.
6. Create instances of Order for different customer types and calculate the final price.
7. Print the final prices after applying discounts.

**Exercise 3: Inventory Management with Restocking Alerts**

**Objectives:**

* Use threading to handle asynchronous tasks.
* Implement file I/O operations for data persistence.
* Use exception handling for robust error management.

**Business Scenario:**

Enhance the inventory management system to include restocking alerts and save inventory data to a file for persistence.

**Tasks:**

1. Create a new Python file named enhanced\_inventory\_management.py.
2. Define a class **Inventory** with methods to add, remove, and check stock levels of items.
3. Implement a method **save\_to\_file()** that saves the current inventory state to a file.
4. Implement a method **load\_from\_file()** that loads the inventory state from a file.
5. Use threading to periodically check stock levels and print restocking alerts for items that are low in stock.
6. Add exception handling to manage file I/O errors.
7. Create an inventory instance, perform some operations, and save the state to a file.
8. Load the inventory state from the file and print it.

**Exercise 4: Organizing Programs and Using Modules**

**Objectives:**

* Organize a Python program using functions and modules.
* Understand how to create and import custom modules.

**Business Scenario:**

You are tasked with creating a utility tool for text processing that includes several functions like counting words, finding unique words, and converting text to uppercase. The tool should be organized into separate modules to ensure clean and maintainable code.

**Tasks:**

1. **Create a New Project Directory:**
   * Create a new directory named **text\_processing\_tool**.
2. **Create Functions:**
   * Inside the **text\_processing\_tool** directory, create separate Python files for each text processing task:
     + **count\_words.py**: Define a function **count\_words(text)** that returns the number of words in a given text.
     + **find\_unique\_words.py**: Define a function **find\_unique\_words(text)** that returns a set of unique words in the text.
     + **convert\_to\_uppercase.py**: Define a function **convert\_to\_uppercase(text)** that converts all text to uppercase.
3. **Organize Code:**
   * In the text\_processing\_tool directory, create an **\_\_init\_\_.py** file to make it a package.
   * Ensure each module is correctly imported in the **\_\_init\_\_.py** file.
4. **Import and Use Modules:**
   * Create a new Python file named **main.py** outside the **text\_processing\_tool** directory.
   * In **main.py**, import the custom modules.
   * Write a script that:
     + Prompts the user to enter a text string.
     + Uses the imported functions to process the text based on user input.
     + Prints the results of the text processing tasks.

**Example Directory Structure:**

text\_processing\_tool/

│

├── \_\_init\_\_.py

├── count\_words.py

├── find\_unique\_words.py

└── convert\_to\_uppercase.py

main.py

**Exercise 5: Python Lists, Dictionaries, Tuples, and Sets (Collection Objects)**

**Objectives:**

* Understand and use Python collections: lists, dictionaries, tuples, and sets.
* Perform various operations on these collections.

**Business Scenario:**

You are creating an inventory management system that keeps track of products, their quantities, and prices. You need to use different collections to store and manipulate this data.

**Tasks:**

1. **Lists:**
   * Create a new Python file named **inventory\_management.py**.
   * Create a list of product names.
   * Write functions to add, remove, and update products in the list.
2. **Dictionaries:**
   * Use a dictionary to store product details (name, quantity, price).
   * Write functions to add, update, and delete product details.
3. **Tuples:**
   * Use tuples to represent immutable product data.
   * Demonstrate how to use tuples in a product catalog.
4. **Sets:**
   * Use sets to track unique product categories.
   * Write functions to add and remove categories from the set.
5. **Combining Collections:**
   * Write a function to generate a report of products sorted by price.
   * Write a function to find products that fall within a certain price range using set operations.