Lab report

Mark: 15

---

1. Inheritance

Scenario: A company has two types of employees: *PermanentEmployee* and *ContractEmployee*. Both employee types have some shared details, such as name and id, but differ in how their salaries are calculated. Permanent employees have a fixed monthly salary, while contract employees are paid based on the number of hours worked.

Question:
Define a base class Employee with attributes name and id. Create two derived classes, PermanentEmployee and ContractEmployee, that inherit from Employee. The PermanentEmployee class should have a method calculate_salary() that returns a fixed monthly salary, while ContractEmployee should have a method calculate_salary() that calculates salary based on hourly rate and hours worked. Demonstrate this setup with appropriate test data.

---

2. Polymorphism

Scenario: A delivery company uses different modes of transportation for various packages, such as *Truck*, *Ship*, and *Plane*. Each transport type has a different way of calculating the delivery cost, which includes parameters like weight and distance.

Question:
Define a base class Transport with a method calculate_cost(weight, distance). Create three derived classes Truck, Ship, and Plane, each implementing calculate_cost() differently based on the transport mode. Demonstrate polymorphism by writing a function that calculates delivery costs for each mode of transportation using a single list of transport objects.

---

3. Built-in Exception Handling

Scenario: You are working on a function that processes a list of values by dividing each by a divisor provided by the user. If the divisor is zero, an error occurs. If the divisor is a non-integer or non-numeric type, it should also raise an error.

Question:
Write a function divide_elements(values, divisor) that accepts a list of numbers and a divisor. Use built-in exception handling to catch a ZeroDivisionError when dividing by zero and a TypeError if the divisor is not numeric. Ensure the program provides meaningful error messages to the user and gracefully continues processing if possible.

---

4. Custom Exception Handling

Scenario: You are designing a banking application that processes withdrawals from an account. Each account has a minimum balance that must be maintained. If a withdrawal would reduce the balance below this minimum, it should raise a custom exception called InsufficientFundsError.

Question:
Create a custom exception InsufficientFundsError with an appropriate message. Define a class BankAccount with attributes balance and min_balance. Write a method withdraw(amount) in BankAccount that raises an InsufficientFundsError if the withdrawal amount would bring the balance below the minimum allowed. Test the class and custom exception with a sample account.

---

5. Numpy Functions

Scenario: You are given a dataset of students' scores in different subjects stored in a 2D NumPy array, where each row represents a student and each column represents a subject. You need to calculate the average score per student and identify the student with the highest average.

Question:
Write a Python function using NumPy to calculate the average score for each student. Then, identify the student with the highest average score and print their score. Use appropriate NumPy functions to solve this problem efficiently.

6. Indexing and Slicing

Scenario: You have a 2D NumPy array representing sales data, where each row is a product, and columns represent sales over several months. You need to analyze sales for specific months and products.

Question:
Given a 2D array of sales data, write code to extract:

- Sales data for the first three products.
- Sales data for all products in the last two months.
- Sales data for a specific product and month (e.g., 2nd product in the 4th month).

Demonstrate these operations on a sample 2D array.

---

7. Type Casting

Scenario: You're working with data that was initially stored as strings in a CSV file and imported into a NumPy array. Now, you need to convert specific columns to integers or floats to perform calculations.

Question:
Given a 2D NumPy array with string values, write code to:

- Convert an entire column to integer type if it represents whole numbers (e.g., age data).
- Convert a column to float type if it represents decimal values (e.g., salary or price).

Demonstrate the type casting on a sample NumPy array.

---

8. Copy and View

Scenario: You have a large dataset in a NumPy array and want to perform operations on a subset without modifying the original data. You're unsure whether to use a copy or view.

Question:
Given a 2D array of data, write code to:

- Create a view of a specific row.
- Create a deep copy of a specific column.
- Modify both the view and the copy, and observe which one affects the original array.

Explain the difference in behavior between the copy and view based on your code.

---

9. Shape and Reshape

Scenario: You have a 1D array representing data from multiple sensors over a period. To perform matrix operations, you need to restructure the data into a 2D format where each row represents a sensor and each column represents a timestamp.

Question:
Given a 1D NumPy array, write code to:

- Reshape it into a 2D array with an appropriate number of rows and columns.
- If the reshaping is not possible due to mismatched elements, explain the error and provide a solution by padding/truncating the data.

Demonstrate the reshaping and necessary checks on a sample array.

---

10. Join

Scenario: You have two arrays representing data from two different branches of a store. To analyze combined sales, you need to merge the data by joining arrays along different axes.

Question:
Given two 1D NumPy arrays representing sales, write code to:

- Join the arrays horizontally to represent two branches as columns.
- Join the arrays vertically to represent two branches as additional rows.

Demonstrate both types of joins and explain in which scenario each type would be useful.

---

11. np.where

Scenario: You're analyzing temperatures recorded over several days. You want to identify days when the temperature exceeded a certain threshold and replace any low temperatures with a specified minimum value.

Question:
Given a NumPy array of temperatures, write code to:

- Use `np.where` to find indices where the temperature exceeds a certain threshold.
- Use `np.where` to replace temperatures below a certain threshold with a minimum value.

Demonstrate this operation with a sample temperature array.

---

7. Search and Sort

Scenario: You have a dataset representing student scores in various exams stored in an array. You need to locate specific scores and arrange the data in ascending or descending order.

Question:
Given a 1D NumPy array of scores, write code to:

- Search for specific scores (e.g., 75 and 90) and return their indices.
- Sort the array in ascending and descending order.

Demonstrate both operations and provide examples.

---

12. Filter

Scenario: You have an array representing product prices, and you need to filter out products that fall within a certain price range for a sale promotion.

Question:
Given a 1D array of product prices, write code to:

- Use boolean indexing to filter products within a specified price range (e.g., between $20 and $50).
- Display only the filtered prices.

Demonstrate the filtering with a sample price array.

---

13. Flatten

Scenario: You have a 3D array representing data collected from multiple devices over multiple days, with each device's daily data stored in a 2D matrix format. You need to process this data as a single list.

Question:
Given a 3D NumPy array, write code to:

- Flatten the array into a 1D array.
- Explain why flattening might be necessary and what kind of operations it enables.

Demonstrate this operation with a sample 3D array.

---

13. Encapsulation

Scenario: Imagine you are developing a software application for a bank that includes a `BankAccount` class. The class should manage customers' account balances, allowing deposits, withdrawals, and balance checks. For security and integrity, it's important to restrict direct access to the account balance, ensuring only authorized methods can modify or view it.

Question:

1. **Create** a `BankAccount` class that includes the following:
    - A private variable for the account balance.
    - Public methods for:
        - **Deposit:** Adds a specified amount to the account balance.
        - **Withdraw:** Subtracts a specified amount from the account balance, ensuring the balance does not go below zero.
        - **Check Balance:** Returns the current balance without allowing direct access to the balance variable.
2. **Perform** the following actions:
    - Deposit money into an account.
    - Attempt a withdrawal larger than the current balance to ensure the account does not go negative.
    - Check the balance.

---

14. Encapsulation

Scenario: You are working on a library management system and need to create a `LibraryBook` class to manage details about each book in the library. To prevent unauthorized changes to critical information, you should use encapsulation principles, specifically by employing private and protected attributes and methods.

Question:

1. **Create** a `LibraryBook` class with the following attributes and methods:

- A **private attribute** `ISBN` that stores the book's ISBN number, which should not be accessible directly or modified from outside the class.
- A **protected attribute** `title` to store the book's title, which can be accessible within subclasses but should not be modified directly from outside.
- A **protected method** `display_basic_info()` that prints the title and author of the book, intended to be used by subclasses for detailed display.
- A **public method** `get_ISBN()` that returns the book's ISBN in a secure way (e.g., masked format).
- A **public method** `borrow_book(borrower_name)` that:
  - Changes the book's status to "borrowed".
  - Prints a message indicating the book has been borrowed by `borrower_name`.

2. **Create** a subclass called `DigitalLibraryBook` that inherits from `LibraryBook`. This subclass should:
   - Use the protected `display_basic_info()` method to print the book's basic info along with additional information specific to digital books (e.g., file format).
3. **Perform** the following actions:
   - Create an instance of `LibraryBook` with a specific title, author, and ISBN.
   - Display the masked ISBN using `get_ISBN()`.
   - Borrow the book using `borrow_book()`.
   - Create an instance of `DigitalLibraryBook` and display the book's basic and digital information using `display_basic_info()`.