* Basic Python Programming *

Data Types & Operations

1. Sum of Positive Integers

Write a Python program that creates a list of integers and floats. Use a for loop to calculate the sum of all positive integers in the list.

2. Filtering and Summing Numbers

Write a Python program that creates a list of numbers, both integers and floats. Use list comprehension to create a new list that contains only the numbers greater than 10. Calculate the sum of the numbers in the new list.

3. Removing Duplicates

Given a list containing both integers and strings, write a Python program that removes duplicate elements from the list and prints the result.

4. Finding the Mean

Create a list of numbers (integers and floats) and write a Python program to compute the mean of the numbers in the list.

5. Count Occurrences of Elements

Write a Python program that counts how many times a particular number appears in a list of integers.

6. Multiplying Numbers

Create a list of numbers and use a for loop to multiply all the integers in the list together. If there are no integers, print a message indicating that.

String Manipulation

1. Case Count

Write a Python function `case_count(text: str) -> dict` that counts the number of uppercase and lowercase letters in a string.

Example:

Input: `"Hello World"`

Output: `{ "uppercase": 2, "lowercase": 8 }`

2. Removing Vowels

Write a Python function `remove_vowels(text: str) -> str` that takes a string and returns a new string with all vowels removed.

3. Palindrome Check

Write a Python function `is_palindrome(text: str) -> bool` that checks if a string is a palindrome. A string is a palindrome if it reads the same forward and backward (e.g., "madam").

4. String Rotation

Write a Python function `rotate_string(text: str, n: int) -> str` that rotates the string by `n` positions. For example, if the input is `"abcdef"` and `n = 2`, the output should be `"cdefab"`.

5. String Reversal

Write a Python function `reverse_words(sentence: str) -> str` that reverses the order of words in a given sentence.

Example:

Input: `"hello world"`

Output: `"world hello"`

6. Substring Check

Write a Python function `contains_substring(main: str, sub: str) -> bool` that checks if a substring exists within a string.

Example:

Input: `"hello world", "world"`

Output: `True`

Functions and Exception Handling

1. Safe Addition

Write a Python function `safe_add(x: float, y: float) -> float` that adds two numbers, and returns a custom error message if the numbers are not valid floats.

2. Safe Subtraction

Write a Python function `safe_subtract(x: float, y: float) -> float` that subtracts two numbers and handles the case where the subtraction results in a negative number.

3. Check for Division by Zero

Write a Python function `divide_numbers(x: float, y: float) -> float` that divides `x` by `y`. If `y` is zero, it should return `"Cannot divide by zero"`.

4. Handling Invalid Input

Write a Python function `parse_input(value: str) -> int` that takes a string input and converts it to an integer. If the conversion fails, return the message `"Invalid input"`.

5. Negative Numbers Handling

Write a Python function `add_positive_numbers(x: int, y: int) -> int` that adds two numbers but only if both are positive. If either number is negative, raise a custom error: `"Both numbers must be positive"`.

6. File Not Found Handling

Write a Python function `open_file(filename: str)` that attempts to open a file and prints an error message if the file is not found.

Working with Files

1. Write Multiple Lines to a File

Write a Python program that creates a file named `records.txt` and writes 4 lines of data (name and age) to it. Use a for loop to write the lines.

2. Count Words in a File

Write a Python program that opens a text file called `words.txt` and counts how many words are in the file.

3. Reading and Appending to a File

Write a Python program that opens a file, reads its contents, and appends a new line of text to it. Then, display the updated content of the file.

4. Extract Lines Containing a Specific Word

Write a Python program that reads a file called `log.txt` and prints all lines containing the word "error".

5. Copy Contents from One File to Another

Write a Python program that reads the content of a file called `source.txt` and writes it to a new file called `destination.txt`.

6. Sort and Write Data to a File

Write a Python program that creates a file, writes some random numbers to it, and then reads and sorts those numbers before writing the sorted list to a new file.

* Object-Oriented Programming *

Class Definition & Initialization

1. Student Class

Define a class `Student` with the attributes `name` (string), `age` (integer), and `grades` (list of floats). Create a constructor to initialize these attributes and a method `average_grade()` that returns the average grade of the student.

2. Library Class

Define a class `Library` with the following attributes: `book_title` (string), `author` (string), `published_year` (integer), and `is_available` (boolean). Create methods to check the availability of the book and borrow it.

3. Product Class

Define a class `Product` with the attributes `product_name` (string), `price` (float), and `quantity` (integer). Write methods to display the total value of the product in stock (`price * quantity`).

4. Car Class

Define a class `Car` with attributes `make`, `model`, `year`, and `color`. Create a method `display_car_info()` that prints out the car details in a readable format.

5. Rectangle Class

Define a class `Rectangle` with attributes `length` and `width`. Create methods to calculate the perimeter and area of the rectangle.

6. Student Database Class

Define a class `StudentDatabase` that holds a list of students. Each student is represented as a dictionary with their `name` and `age`. Provide methods to add a student, remove a student, and display all students.

Inheritance

1. Shape Inheritance

Define a base class `Shape` with a method `draw()`. Then create subclasses `Circle` and `Rectangle`, each implementing the `draw()` method. Demonstrate polymorphism.

2. Animal and Dog Classes

Create a class `Animal` with attributes `name` and `species`. Then, create a class `Dog` that inherits from `Animal`, with an additional attribute `breed`. Demonstrate usage by creating instances of both classes and printing their information.

3. Person and Employee Classes

Define a `Person` class with attributes `name`, `age`, and `address`. Then define an `Employee` class that inherits from `Person` and adds an attribute `salary`. Print out the details of an employee.

4. Vehicle and Electric Car Classes

Create a class `Vehicle` with attributes `make` and `model`. Then create a class `ElectricCar` that inherits from `Vehicle`, adding the attribute `battery_capacity`. Override the `display_info()` method to include the battery capacity.

5. Shape and Triangle Classes

Create an abstract class `Shape` with an abstract method `draw()`. Then create a subclass `Triangle` that implements the `draw()` method, printing a message when a triangle is drawn.

6. Fruit and Apple Classes

Define a class `Fruit` with an attribute `name` and a method `taste()`. Then define a subclass `Apple` that inherits from `Fruit` and overrides the `taste()` method to print a specific message about the apple's taste.

Encapsulation

1. Bank Account Class

Define a class `BankAccount` with private attributes `account_number` and `balance`. Provide public methods `deposit()` and `withdraw()` to handle deposits and withdrawals. Ensure that the `withdraw()` method checks that the withdrawal amount is not greater than the balance.

2. Employee Class with Salary

Define a class `Employee` with a private attribute `salary`. Write a method `get_salary()` that allows access to the salary, but only if the employee's salary is greater than 0.

3. Account Holder Class

Create a class `AccountHolder` with private attributes `name` and `account_balance`. Provide public methods `deposit(amount)` and `withdraw(amount)` to handle money transactions, while ensuring that negative values are not accepted.

4. Product Stock Class

Define a class `ProductStock` with private attributes `product_name` and `quantity_in_stock`. Write a public method `check_availability()` to check if a product is in stock and return a message accordingly.

5. Gradebook Class

Create a `Gradebook` class with private attributes `student_name` and `grades` (a list). Provide a method `add_grade()` to add grades, and a method `average_grade()` that returns the average of all grades.

6. Movie Class

Create a class `Movie` with a private attribute `rating` and a public method `get_rating()` that allows access to the rating only if it is above a certain threshold (e.g., 3).

Abstraction & Interfaces

1. Shape Area Calculation

Define an abstract class `Shape` with an abstract method `calculate_area()`. Then create subclasses `Circle` and `Rectangle` that implement this method, calculating the area for each shape.

2. Payment System

Create an abstract class `Payment` with an abstract method `process_payment()`. Then create two subclasses: `CreditCardPayment` and `PayPalPayment`. Implement the `process_payment()` method in both subclasses.

3. Transportation System

Create an abstract class `Transportation` with an abstract method `move()`. Then create subclasses `Car` and `Bicycle` that implement the `move()` method in different ways.

4. Appliance Interface

Define an abstract class `Appliance` with an abstract method `turn_on()`. Create subclasses `WashingMachine` and `Refrigerator`, each implementing the `turn_on()` method.

5. Shape Drawing Interface

Define an abstract class `Shape` with an abstract method `draw()`. Then create subclasses `Circle` and `Square` that each implement `draw()` to display the shape in a console.

6. Employee Task Management

Create an abstract class `Employee` with an abstract method `perform_task()`. Then create two subclasses, `Manager` and `Developer`, and implement `perform_task()` for each subclass with appropriate tasks.

Multiple Inheritance

1. Artist and Writer Classes

Define a class `Artist` with the attribute `art_style` and a method `create_art()`. Define another class `Writer` with the attribute `writing_style` and a method `write()`. Then define a class `CreativePerson` that inherits from both `Artist` and `Writer` and has a method `display_info()` that displays both art and writing styles.

2. Student and Sportsman Classes

Define a class `Student` with attributes `name` and `age`, and a method `study()`. Define another class `Sportsman` with an attribute `sport` and a method `play_sport()`. Then define a class `StudentAthlete` that inherits from both `Student` and `Sportsman` and implements a method `display_info()` to show all details.

3. Person and Vehicle Classes

Define a class `Person` with attributes `name` and `age`. Define another class `Vehicle` with an attribute `model`. Create a class `Driver` that inherits from both `Person` and `Vehicle` and displays a message that includes the driver's name, age, and the vehicle model.

4. Teacher and Researcher Classes

Define a class `Teacher` with the attribute `subject`. Define another class `Researcher` with the attribute `research_area`. Create a class `Professor` that inherits from both `Teacher` and `Researcher` and prints out the details of both the subject and research area.

5. Chef and Server Classes

Define a class `Chef` with the attribute `specialty`. Define another class `Server` with an attribute `restaurant_name`. Create a class `RestaurantEmployee` that inherits from both `Chef` and `Server` and displays the employee's specialty and restaurant name.

6. Product and Category Classes

Define a class `Product` with attributes `product_name` and `price`. Define another class `Category` with the attribute `category_name`. Then create a class `ProductCategory` that inherits from both `Product` and `Category`, and displays the product name along with its category.