Python Mentorship Assignment: Advanced OOP & Logic

Part 1: Theoretical Questions

Q1. Mutable Default Arguments
- Explain what happens if you define a function like this: "def add_item(item, box=[]):".
- Why does the list "box" persist data between function calls if you don't provide a second argument?
- Challenge: Rewrite the function header to fix this common bug.

Q2. __str__ vs __repr__
- Both methods return string representations of an object. What is the strict technical difference in their intended audience?
- Which one is used as a fallback if the other is missing?

Q3. Class Variables vs. Instance Variables
- Explain the memory difference between a variable defined inside "__init__" (using self.var) versus a variable defined directly under the "class Name:" header.
- If you change a Class Variable using "ClassName.var = new_value", what happens to existing instances? What happens if you try to change it via "instance.var = new_value"?

Part 2: Programming Challenges

Q4. Complex Dictionary Parsing (Log Analysis)
Task: You have a string of server logs: "User1: Login; User2: Login; User1: Logout; User3: Login; User2: Logout".
Requirement: Write a function that parses this string.
Output: Return a dictionary that tracks the *current state* of each user. It should identify who is currently "Online" vs "Offline". (e.g., {'User1': 'Offline', 'User3': 'Online'}).
Constraint: Assume everyone starts "Offline". You must handle the logic of consecutive Logins or Logouts gracefully.

Q5. The "Safe" Calculator (Error Handling)
Task: Write a program that repeatedly asks the user for two numbers and an operator (+, -, /, *).
Requirement:
1. Perform the calculation.
2. Handle "ZeroDivisionError" specifically (print "Cannot divide by zero").
3. Handle "ValueError" if the user enters text instead of numbers.
4. Use an "else" block to print the result only if no errors occurred.
5. Use a "finally" block to print "Execution attempt complete" regardless of the outcome.

Part 3: Advanced OOP Challenges

Q6. Class Interaction & State Management (The Library System)
Task: Create two classes: "Book" and "Library".
- Book Class: Should have title, author, and a boolean is_checked_out.
- Library Class: Should hold a list of Book objects.

Methods Required:
- Library.add_book(book_obj): Adds a book to the library.
- Library.checkout_book(title): Searches for the book by title. If found and available, set is_checked_out to True. If already checked out, raise a custom Exception or print an error.
- Library.return_book(title): Sets is_checked_out to False.

Python Mentorship Assignment: Advanced OOP & Logic

Goal: Demonstrate that the Library object can modify the state of specific Book objects inside its internal list.

Q7. Encapsulation with Property Decorators
Task: Create a class "Employee".
Attributes: first, last, and salary.
Constraint 1: The "email" should not be a stored attribute. It should be a method accessible as a property (@property) that constructs "first.last@company.com" dynamically.
Constraint 2: Use a "@salary.setter" to ensure no one can set a negative salary. If a negative value is passed, raise a ValueError.
Constraint 3: Use a "@fullname.deleter" that sets the first and last names to None when the deleter is called.

Q8. Operator Overloading (Magic Methods)
Task: Create a class "TimeDuration" that accepts hours and minutes.
Requirements:
1. Normalization: If initialized with TimeDuration(hours=2, minutes=70), it should automatically convert to 3 hours, 10 minutes inside __init__.
2. Addition: Implement "__add__" so that "t1 + t2" returns a *new* TimeDuration object with the correct total time (handling minute rollover).
   Example: 2h 45m + 1h 30m should result in a new object 4h 15m.
3. String Rep: Implement "__str__" to print "XH:YM".
Test: Create two time objects, add them together using the "+" sign, and print the result.