Deadline: Sunday, Nov. 9th 11:59 PM

Lab 8 – Class (OOP) and Introduction to debugging

Summission

- Lab8_debug_exercise.pdf
- Lab8_<your_name>_<SID>_Q1.py
- Lab8_<your_name>_<SID>_Q2.py
- Lab8_<your_name>_<SID>_Q3.py

1. (2 Pts) Debug Practice

Debug and fix the programs in the Word file of "Lab8_Debug_Exercise", and submit pdf.

2. Program Assignment

1. (2pts) Employee Class

Write a Python class Employee with attributes like emp_id, name, salary, and department, and methods like calculate_salary, assign_department, and print_employee_details .

1. Use assign_department method to change the department of an employee.
2. Use print_employee_details method to print the details of an employee.
3. Use calculate_salary method takes one argument: hours_worked, which is the number of hours worked by the employee. If the number of hours worked is more than 50, the method computes overtime and adds it to the salary. Overtime is calculated as following:
    - overtime = hours_worked – 50
    - overtime amount = (overtime * (salary / 50))

Example Usage:

```
employee1 = Employee("ADAMS", "E7876", 50000, "ACCOUNTING")
employee2 = Employee("JONES", "E7499", 45000, "RESEARCH")

# Change the departments of employee1 and employee4
employee1.assign_department("OPERATIONS")
employee2.assign_department("SALES")

# Now calculate the overtime of the employees who are eligible:
employee1.calculate_salary(52)
employee2.calculate_salary(60)

print("Updated Employee Details:")
employee1.print_employee_details()
employee2.print_employee_details()
```

2. (2pts) ComplexNumber Class

Create a Python class called ComplexNumber to represent complex numbers and to perform basic arithmetic operations (addition and subtraction) on them without using inheritance. A complex number is defined as a + bi, where a is the real part and b is the imaginary part.

Your ComplexNumber class should have two instance variables: real for the real part and imag for the imaginary part. The class should also include the following methods:

1. An __init__ method that initializes the real and imaginary parts of the complex number.
2. A __str__ method that returns a string representation of the complex number in the form "a + bi" or "a - bi", handling the sign of the imaginary part properly.
3. An `add` method that takes another ComplexNumber object and returns a new `ComplexNumber representing the sum of the two complex numbers.
4. A subtract method that takes another ComplexNumber object and returns a new ComplexNumber representing the difference between the two complex numbers.

Note that you should not use the built-in complex number support in Python (i.e., complex type). Instead, implement the functionality from scratch.

Example Usage:

```
# Instantiate two ComplexNumber objects
c1 = ComplexNumber(3, 4)
c2 = ComplexNumber(1, -2)

# Add the two complex numbers
result_add = c1.add(c2)
print(result_add)  # Should display "4 + 2i"

# Subtract the second complex number from the first one
result_subtract = c1.subtract(c2)
print(result_subtract)  # Should display "2 + 6i
```

3. (4pts) Student Class

1) Write a class, Student, which has three properties, including a 6-digit student ID, a name, and a list of exam scores.

The constructor takes the student name as the first argument, followed by student ID, an optional number of additional arguments representing the examination scores. The Student class provides two methods, including `average`, which takes no argument and returns the average score of the student (if there is no scores input, return None), and __str__, which formats the student information as ID \t name \t scores, where scores is a tab-seperated list of floating-point numbers.

Example Usage:

```
s1=Student('John Smith', 100001)
```

```
print(s1)                 # 100001 John Smith
print(s1.average())       # None
s2=Student('Mary Harper', 100002, 100, 90.5, 75)
print(s2)                 # 100002  Mary Harper     100     90.5    75
print(s2.average())       # 88.5
```

2) Write a new class, Class, which has a single attribute recording a list of students. The two classes basically define a rudimentary student database. We can make it more useful be adding funtionalities.

1. Add a method enroll_student – add the pass-in student in the class
2. Add a method size – return how many students in the class
3. Add a method highest_score_student – return the student with the highest average score in class. (Don't consider the student if it no score included)
4. Add a method is_student_enrolled – take one argument of student id, return whether the student is in the class.

Example Usage:

```
s1=Student('John', 100001, 70)
s2=Student('Mary', 100002, 90.5, 75)
class1 = Class([s1, s2])
print(class1.size()).     # 2
print(class1.highest_score_student()).     # 100002  Mary Harper   90.5  75

s3=Student('Jack', 100005, 95, 80)
class1.enroll_student(s3)

print(class1.size()).     # 3
print(class1.is_student_enrolled(100005))       # True
print(class1.is_student_enrolled(100007))       # False
```