**2023 Spring OOP II**

**HW\_3**

**Q.1**

Create a class named “Player” that initiates a player with default name as “player”, default level as 0, and default character type as “default”. Create a function that compares two players' levels and prints the players name, level, and character type that has the higher level. If both characters are the same level print “tie”. Write a main program that creates and compares two instances created with random levels 1-99.

**Q.2**

Assume there are ***N*** students, *S1*, *S2*, …, *SN* with student IDs 1, 2, …, ***N***, and ***M*** seats available in a classroom with ***N*** <= ***M***. Generate a random seating assignment with the **random** module to spread the students out as much as possible. For simplicity, you can assume the layout for all the seats is one-dimensional (or linear).

A sample output looks like

**Number of students: 7**

**Number of seats: 20**

**[1, '\_', '\_', 3, '\_', '\_', '\_', 4, '\_', '\_', 2, '\_', '\_', 6, '\_', '\_', 7, '\_', '\_', 5]**

but you can provide the output in your desired format.

Do not use pseudo random number. Also provide a function named **findSeat** which takes a student ID as input (thus an integer between 1 and ***N***) and output the seat number (an integer between 1 and ***M***) assigned to the student.

**Q.3**

File Concatenation: Many operating systems allow one to concatenate files. Design a class called **TextDocument** that reads the contents of a text file only. Create a constructor that takes the path of a text file. Create an overloaded addition operator for this class that adds (concatenates) two **TextDocument** objects (appends the second one at the end of the first one) and creates a new text file. Handle all possible error cases, such as for file existence.

**Q.4**

Design a class called **Color**. The fields of the class are three decimals for Red, Green, and Blue components in the range 0 to 1, inclusive (0 indicates Black and 1 indicates White). Add checks to ensure that the values are always in the given range. Provide addition and subtraction operators for the color class. Include saturation in the addition and subtraction: if any component goes less than 0 or greater than 1, assign them 0 or 1, respectively.

**Q.5**

Design a **Logarithm** class. Fields should be the base and number. Provide addition and subtraction operators—remember to adjust bases appropriately. *Hint*: Use base 2 or 10 as a canonical base for operations.

**Q.6**

Augment the Rational number class in the textbook to include multiplication and division. Include the ability to accommodate operands of type int.

**Q.7**

Write an **Odometer** class. An odometer is the gauge on your car that measures distance traveled. In the United States, an odometer measures miles; elsewhere, it measures kilometers. Many vehicles with electronic odometer interfaces have the ability to switch between miles and kilometers. The accuracy is 1/10 of a mile (kilometer).

Something to consider: if an odometer gets replaced, a new one must be able to be set to some specified mileage.

Include the following methods:

(a) \_\_init\_\_, \_\_str\_\_, \_\_repr\_\_.

(b) The constructor must take two arguments that both have default values: one is mileage, and the other  
 specifies units.

(c) Addition and subtraction both have one odometer operand and one numeric operand, where the  
 numeric operand represents the miles being added/subtracted (not two odometer operands).

(d) Addition should be commutative (but not subtraction).

(e) Output should always be rounded to a 1/10 of a mile (kilometer), but the odometer itself should maintain  
 full floating-point accuracy.

Include sample code that uses your class and demonstrates the use of all methods as well as demonstrating error handling.

**Q.8**

Write a **Clock** class that measures hours, minutes, and seconds.

Include the following methods:

(a) \_\_init\_\_, \_\_str\_\_, \_\_repr\_\_.

(b) Addition allows for both clock-to-clock operations and clock-to-integer operations. If one argument   
 is an integer, the integer is assumed to represent hours. Addition should be commutative!

(c) Output should always be rounded to the second.

(d) If inappropriate values are passed to the constructor, the created clock instance should get a value of 0 hours,   
 0 minutes, and 0 seconds.

Include sample code that uses your class and demonstrates the use of all methods as well as error handling.