2020 - ESS 112 Programming I (Python)

Assignment 7

Instructions

- Answers to each question should be provided in a file whose name is mentioned against the respective question.
- Use appropriate function names and class names as specified in the questions. Please ensure that your code does not have any extraneous input/output code.
- Add comments wherever necessary and submit a clean, well-written code.

Questions

1. Create a class `Vehicle` as described below:

   (a) The constructor should take the `name` and `brand` as strings, and the `price` and the `mileage` as a floats.

   (b) Write a class method `checkLuxury` which checks if the `price` of a vehicle is greater than 1000000.0 and returns a boolean value.

   (c) Write a class method `checkEfficiency` which checks if the `mileage` of the vehicle is greater than 20.0 and returns a boolean value.

   (d) Write a function `efficientVehicles`, *that is not a part of the Vehicle class* which takes a list of vehicle objects and returns a list of all the *names* of the efficient vehicles.

   (e) Write a method `priceOfBrand`, *that is not a part of the Vehicle class*, which takes the brand and a list of vehicle objects as parameters and returns the sum of prices of all the vehicles belonging to a brand.

   Use the following as a test case to check your code (define an `__str__` function that returns a string as given):

```
def t1():
    v1 = Vehicle("Alto", "Suzuki", 100000.0, 50.0)
    v2 = Vehicle("SX4", "Suzuki", 200000.0, 35.5)
    v3 = Vehicle("R8", "Audi", 1000000.0, 15.7)
    v4 = Vehicle("Q3", "Audi", 1500000.0, 18.5)

    print(v1.checkLuxury())

    print(v2.checkEfficiency())

    print("Efficient Vehicle: ", efficientVehicles([v1, v2, v3, v4]))

    audiPrice = priceOfBrand("Audi", [v1, v2, v3, v4])
    print("Audi prices:", audiPrice)

    print(v1)
```

```
if __name__ == "__main__":
    t1()
```

Output –

```
False
True
Efficient Vehicles: ['Alto', 'SX4']
Audi price: 2500000.0
Vehicle Name: Alto, Brand: Suzuki, Price: 100000.0, Mileage: 50.0
```

(**file:** Q1.py)

2. Create a class `Date` whose constructor takes three attributes: `day`, `month` and `year`. Creating the object should print an error message "Invalid date" if they contain invalid values (eg. 29th February 2021 is an invalid date). *Let the upper limit on the year be 2021* (if the given date is (31, 12, 2021) then simply print (1, 1, 2022)).

   - Write a method `leap` to check if the given year is a leap year.
   - Write a method `valid` to check if the given date is valid, that calls the `leap` function inside it. This method can be called inside the constructor to initialize the class attributes as necessary.
   - The above two methods are provided to give a concrete format to the computation in the following `tomorrow` method and the test cases will *only* be run on the tomorrow method. Therefore, the `leap` and `valid` methods may return any value necessary for computation.

   Write a method `tomorrow` that returns a tuple containing the next day of the current date. If the date object was invalid, it should *return* (not print) the string "Cannot find next day for invalid date".

   Use the following as a test case to check your code:

```
def t2():
    d1 = Date(15, 8, 2002)
    print(d1.tomorrow())

    d2 = Date(29, 2, 2021)
    print(d2.tomorrow())

    d3 = Date(31, 6, 1842)
    print(d3.tomorrow())

    d4 = Date(25, 3, 2022)
    print(d4.tomorrow())

    d5 = Date(16, 13, 1257)
    print(d5.tomorrow())

    d6 = Date(-7, -3, -2001)
    print(d6.tomorrow())

    d7 = Date(28, 2, 2020)
    print(d7.tomorrow())

    d8 = Date(31, 12, 1999)
```

```
        print(d8.tomorrow())

    if __name__ == "__main__":
        t2()
```

Output:

```
(16, 8, 2002)
Invalid date
Cannot find next day for invalid date
Invalid date
Cannot find next day for invalid date
Invalid date
Cannot find next day for invalid date
Invalid date
Cannot find next day for invalid date
Invalid date
Cannot find next day for invalid date
(29, 2, 2020)
(1, 1, 2000)
```

(**file:** Q2.py)

3. Create an `Employee` class as described below:

   (a) The constructor should take the following two parameters: the employee name `emp_name` as a string and date of birth `dob` as a tuple of (dd, mm, yyyy).

   (b) The constructor should include an attribute `emp_id` which is computed automatically by maintaining an *emp_count* variable as a class variable (a class variable is a variable that is shared by all objects of the class).

   (c) Write a method `checkSpecialEligibility` that checks if the age of the employee ≥ 50 (as of the current date) and returns a boolean value. You may compute the age by using the *datetime* module and computing the difference in the year values.

   (d) Write a method to ensure that calling print() on an object of the employee class results in an output of the following example format:
   `Employee ID: 1, Employee Name:  Ajay, Employee Age = 29 years.`

   (e) Write a method `addWorkExperience` which takes a list of previous companies that the employee has been associated with (`previous_companies`) as a parameter and assigns it to an attribute `work_exp`. If the work_exp attribute already has some values, simply append the given list of companies to it.

   (f) Write a method `getWorkExperience()` which returns the attribute `work_exp`.

You can use the following test case to check your code:

```
def t3():
    e1 = Employee("Ajay", (21,3,1992))
    e2 = Employee("Rakesh", (31,12, 1990))
    e3 = Employee("Manoj", (2,2,1970))

    print(e1.checkSpecialEligibility())
    print(e3.checkSpecialEligibility())

    e2.addWorkExperience(["Amazon", "Morgan Stanley"])
    e2.addWorkExperience(["Microsoft", "Goldman Sachs"])
    print(e2.getWorkExperience())

    print(e1)
    print(e2)
    print(e3)

if __name__ == "__main__":
    t3()
```

Output:

```
False
True
['Amazon', 'Morgan Stanley', 'Microsoft', 'Goldman Sachs']
Employee ID: 1, Employee Name: Ajay, Employee Age = 29 years
Employee ID: 2, Employee Name: Rakesh, Employee Age = 31 years
Employee ID: 3, Employee Name: Manoj, Employee Age = 51 years
```

(**file:** Q3.py)

4. (a) Implement a `Student` class where the constructor takes the following parameters: `roll_num, stu_name, department, jee_rank`. Also include an attribute `courses_enrolled` which is initialized to an empty list.

   (b) Implement a `Professor` class with attributes: `prof_id, prof_name, department, courses_taught`

   (c) Implement an `Institution` class where the constructor takes the following parameters: `inst_name, location, department_list, profs_list, students_list`

      i. Implement a method `enrollStudent` which takes a student object `s` (corresponding updates have been made in the sample test case) and course name as parameters. It checks if the course is taught by a professor who is in the same department as that of the student and adds it to the list of `courses_enrolled` attribute of `s` and prints `Enrolled successfully`. If the departments are different, print `Not eligible to enroll in` *`course_name`*. On receiving an invalid course name, print `Invalid course name`.

      ii. Implement a method `findToppers` which takes an integer n and returns the list of top n student names based on their JEE ranks. Set the default value of n as 1.

      iii. Write a method to ensure that calling print() on an object of Institution class results in an output of the following example format:
      `Institution` *`inst_name`* `is located in` *`location`* `and has` *`x`* `professors and` *`y`* `students. It has` *`z`* `departments:` *`d1_name, d2_name, d3_name etc.`*

You can use the following test case to check your code:

```
def t4():
    s1 = Student(1, "Vikram", "CSE", 5500)
    s2 = Student(2, "Samrudhhi", "ECE", 2500)
    s3 = Student(3, "Apoorv", "ECE", 6300)
    s4 = Student(4, "Chaitanya", "CSE", 9500)
    s5 = Student(5, "Akanksha", "CSE", 3200)
    s6 = Student(6, "Akshita", "ECE", 5700)

    p1 = Professor(1, "Sanjay", "CSE", ["Java", "Computer Graphics"])
    p2 = Professor(2, "Ajeesh", "CSE", ["Programming Languages", "Compilers"])
    p3 = Professor(3, "Nirmal", "ECE", ["VLSI"])
    p4 = Professor(4, "Shantanu", "ECE", ["Processor Architecture", "RTOS"])
    p5 = Professor(5, "Rajesh", "CSE", ["ML", "Visual Recognition", "NLP"])
    p6 = Professor(6, "Geetha", "ECE", ["Digital Design"])
    p7 = Professor(7, "Anusha", "CSE", ["DSA", "Graph Theory"])

    inst1 = Institution("IIITB",  "Bangalore", ["CSE", "ECE"],
                        [p1, p2, p3, p6], [s1, s2, s3])
    inst2 = Institution("IITD", "Delhi", ["CSE", "ECE", "EEE"],
                        [p4, p5, p7], [s4, s5, s6])

    inst1.enrollStudent(s1, "Java")
    inst1.enrollStudent(s1, "C")
    inst1.enrollStudent(s1, "Compilers")
    inst1.enrollStudent(s2, "Computer Graphics")

    print(inst1.findToppers(2))
    print(inst2.findToppers())

    print(inst2)

if __name__ == "__main__":
    t4()
```

Output:

```
Enrolled successfully
Invalid course name
Enrolled successfully
Not eligible to enroll in Computer Graphics

["Samrudhhi", "Vikram"]
["Akanksha"]

Institution IITD is located in Delhi and has 3 professors and 3 students.
It has 3 departments: CSE, ECE, EEE
```

(**file:** Q4.py)