Lab 02 Specification – A Hand-on Exercise to practice Object Oriented Programming
50 points

Due by: 03/24/2021 8:00 AM

Lab Goals

- Learn to develop reusable code.
- Practice fundamental techniques in object-oriented programming.
- Do a simple exercise on Inheritance.

Learning Assignment

If you have not done so already, please read all of the relevant ”GitHub Guides”, available at the following website:
`https://guides.github.com/`
that explains how to use many of the features that GitHub provides. This reading will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read:

- **GT chapter 02, 2.1,2.2,2.4**

Assignment Details

Now that we have discussed some basics of object-oriented progamming together in the last few lectures, it is your turn. In this lab, you will practice a variety of programs to retain the knowledge of object-oriented programming that had been covered so far. This includes modifying one or more code files to implement a series of functionalities. At any duration during and/or after the lab, students are recommended to team up with the Professor and the TL(s) to clarify if there is any confusion related to the lab and/or class materials.

Students are recommended to get started with this part in the laboratory session, by discussing ideas and clarifying with the Professor and the Technical Leader(s). It is acceptable to discuss high-level ideas with your peers, while all the work should be done individually. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other’s program(s) is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as skill test, exams, etc $\cdots$

Part 01 - A Simple Exercise to Implement Classes and Objects (20 points)

Write a Python program that implements the `Patient Database` program using a series of requirements outlined below.

1. Let us suppose that a hospital wants to create a list of their inpatients. The information to store include:
   (a) Patient ID
   (b) Patient Full Name
   (c) Admission Date
   (d) Disease
   (e) Discharge Date
2. For simplicity, let us make the `Patient Database` program work for ten patients and the patient details are shown below:

Patient ID	Patient Full Name	Admission Date	Disease	Discharge Date
10011001	Alex Crowe	12/30/2019	Tuberculosis	01/03/2020
10011002	Amelia Kaur	01/02/2020	Asthma	01/07/2020
10011003	Amanda Daya	01/15/2020	Heart Attack	01/23/2020
10011004	Ben Krish	01/20/2020	Stroke	01/23/2020
10011005	Brian Miller	12/25/2019	Urinary Tract Infection	12/30/2019
10011006	Chris Miller	12/12/2019	Asthma	12/15/2019
10011007	Drew Millan	01/12/2020	Hypertension	01/20/2020
10011008	Frank Derrick	01/12/2020	Sarcoidosis	01/26/2020
10011009	Robin Meade	01/13/2020	Lung Cancer	01/28/2020
10011010	Rosy David	01/15/2020	Diabetes	01/17/2020

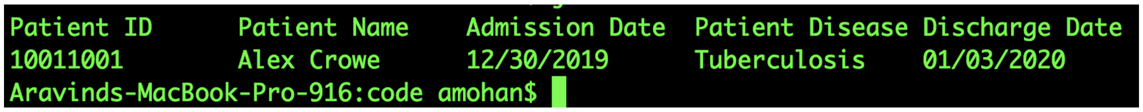
Table 1: Patient Database Table

3. The starter code is provided inside the lab repository in a file named, `patient.py` and `patientinfo.py`. The starter-code has very minimal lines of code and is required to be completed.
4. First, add the required constructor implementation to the `patient.py` file. The constructor should be used to initialize the members of the patient class. The members are patient_id, patient_fullname, admission_date, disease, discharge_date.
5. Next, add the lines of code to the function named `display` in the `patient.py` file. This function should take any patient object and print their ID, Name, Admission Date, Disease, and Discharge Date. It is not a requirement, but you may chose to print a header and display the output nicely formatted to read. You have complete freedom to think creatively on how to display the information effectively on screen?

6. Next, complete the `patientinfo.py` file by implementing an object for every patient based on the table provided above. Please note that you can either create 10 objects for 10 patients by including ten separate objects namely [p1, p2, p3, ..., p10] in your implementation. This may not be an efficient solution. An efficient solution will be to use a list to store the patient details and iterate through the list to create objects within your loop. Although the lab sheet does not set this as a requirement, the latter is recommended in your implementation. To avoid complexity, the patient details may be hard-coded in the program. That is, it is not required to get the patient details as user input. This will be recommended in the course project. Loading the data from an external file is more preferrable. We will look at such options in class in the next few weeks.

7. It is expected that in your implementation you invoke the `display` method in the `patientinfo.py` file. By invoking this method and passing the corresponding object, we can display the patient header and details as the output of the program on the console.

8. A screenshot below shows the output of the program. Please note, to minimize space, the screenshot only includes the details connected to one patient. The lab submission should include the details connected to all 10 patients.



```
Patient ID      Patient Name    Admission Date  Patient Disease Discharge Date
10011001        Alex Crowe      12/30/2019      Tuberculosis    01/03/2020
Aravinds-MacBook-Pro-916:code amohan$
```

Part 02 - An exercise to practice Single Inheritance (20 points)

Write a Python program to implement the concept of **Inheritance**.

1. The starter code is provided inside the lab repository in the files named, `room.py`. The starter code has a minimal amount of code. Add the following members to the `room` class.

   - A member variable named `length` of the integer data type.
   - A member variable named `breadth` of the integer data type.
   - A constructor with two formal parameters (x and y) of the integer data type. The constructor should initialize the value of the variables length and breadth to the value of x and y respectively.
   - A member method named `area` with no formal parameters. The method should compute the area by multiplying the length and breadth. The method should return the area as a float output. Round the float output to 2 decimal places. An example of using round built-in function is shown below for your reference.

```
no = 3.4567899
print(str(round(no, 2)))
```

2. The starter code is provided inside the lab repository in a file named, `bedroom.py`. The starter code has a minimal amount of code. Add the following members to the `bedroom` class.

   - Create a parent-child relationship between room and bedroom class. The bedroom class should be a child to the room class. So how do you connect two classes and create an edge between them? We discussed this in detail during class discussion (please refer the oop folder in Week04 folder of the course repo).
   - A member variable named `height`.
   - A constructor with three formal parameters (x, y, and z) of the integer data type. The constructor should invoke the parent constructor in the`room` class with the actual parameters x and y respectively. So how do you call the parent constructor from the child constructor? A parent constructor can be invoked by using the super keyword. An example is provided below for your reference. It is worth noting that here b is the child class name and a is the parent class. PS how the constructor in parent class (a) is invoked from the child class (b). Please try this code separately to understand the flow of execution.

```
class a(object):
    def __init__(self):
        print("parent a")
class b(a):
    def __init__(self):
        print("child b")
        super(b, self).__init__()
obj = b()
```

   - Next, the constructor should initialize the value of the variables height to the value of z. Please recall the difference between actual and formal parameters from class discussion. A quick look at our slides and notes will be helpful to refresh our understanding.
   - A member function named `volume` with no formal parameters. The method should compute the volume by multiplying the length, breadth, and height. The method should return the volume as a float output. Round the float output to 2 decimal places.

3. The starter code is provided inside the lab repository in a file named, `roomstub.py`. The starter code has a minimal amount of code. Add the following members to the `roomstub` class.

   - Prompt the user to type in the room specifications, such as the length, breadth, and height. The user prompt may be done in a similar manner as we did in the previous labs. The user prompt may be implemented by greeting the user with a welcome message, prompting the user to type in the values, and using input() builtin method to parse the user input and store it in the variables. Type convert the inputs to an integer. That is, length, breadth, and heights are assumed to be integers.
   - Instantiate the class bedroom by creating an object named `br`.
   - Invoke the `area` method using the object `br` and print the return value from the function on the console.
   - Invoke the `volume` method using the object `br` and print the return value from the function on the console.
   - Once all the requirements listed above are set up correctly, the output of the area and volume is expected to be printed as an output of the program. Note: The area and volume should be computed based on the user input for length, breadth, and height.
   - It is worth reiterating that the `area` method is inside the room class. But we are accessing the method through the bedroom class, which is the child of the room class. That's the magic of Inheritance.

Part 03 - An exercise to practice Multilevel Inheritance (20 points)

There are several classic programming examples used to show inheritance, including an `Animal` hierarchy, a `Vehicle` hierarchy, and a `Clock` hierarchy. In this lab, we will be implementing a fourth classic example, the `Shape` hierarchy.

The hierarchy of classes that you will create is shown at the top of the next page in Figure 1. Detailed descriptions for each of the classes follow, and a final UML class diagram for each class is shown in Figure 2.

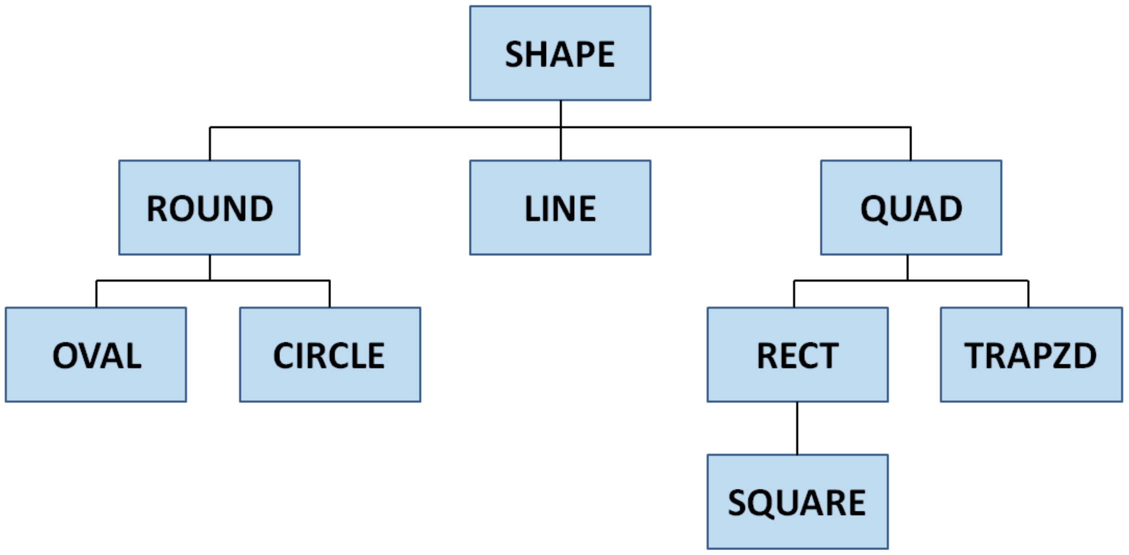


Figure 1: A summary view of the classes involved in this lab and the inheritance relationships between them.

To begin, we need to create a `Shape` class. For each of the shapes that we create, we will want to have the ability to calculate their perimeter and area; however, each function will have a different way to calculate perimeter and area, so we can't put the functions that calculate here. But we can create `float` variables for `perimeter` and `area`, as well as the constructor to initialize those variables. You may initialize `perimeter` and `area` to 0 here. For simplicity, this step is completed in the starter code.

Also included in this section is the ability for your classes to work with a `driver` function called `stub` that is provided in the `starter-code` directory. Please look carefully at this code, as it is already working for the circle shape. I recommend starting this lab slowly, implementing one branch of the hierarchy at a time, testing with the `driver` function. A printout of the output for `stub` is included after all of the shape descriptions.

The overall goal of the `stub.py` code is expecting the `getArea()` and `getPerimeter()` functions to exist, however. Figure 1 shows that we need to create three child classes that will inherit from `Shape`: `Round`, `Line`, and `Quad`. The `Round` and `Quad` classes have children of their own, which will actually implement the `calculateArea()` and `calculatePerimeter()` functions. The details for each of these implementations follow.

1. **Round Class:** The `round` class is an intermediate class in the hierarchy, located above the `oval` and `circle` classes but below the `shape` class. We still calculate the area and perimeter of these shapes differently, so that behavior doesn't belong at this level. We can add one member variable here though. Both child shapes are round (hence the name of the class), so both will make use of π in calculating the perimeter and area. In this class, we can declare a `float` for `pi` equal to 3.1415926535 (you can initialize it here) using a constructor.

2. **Line Class:** The `line` class is at the bottom of its hierarchy chain, so it will actually implement the two functions `calculateArea()` and `calculatePerimeter()`. We know that the area of a line is 0, so this one is trivial. We will think of a line as a rectangle with a height of 0, so the perimeter of the line is equal to double its length, or $2 * \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. We will also need to create a `line` constructor to get those x_1, y_1, x_2, and y_2 variables in that order, as well as instance variables to store them (because these will belong to the `line` class.

3. **Quad Class:** The `quad` class is another intermediate class located between the `shape` class and its actual shapes. Much like the `round` class, we will calculate the area and perimeter differently for each of its children, so that behavior doesn't belong at this level. We can add variables for `width` and `height` at this level though, as well as the constructor to initialize the members.

4. **Oval Class:** The `oval` class will be similar to what we needed to do for the `line` class, as well as all subsequent classes in this section. We will need two variables for our oval, representing half of the major axis and minor axis of the oval (think of them as the radius of the oval at both its largest and smallest). We can call these variables `major` and `minor`. We will need a constructor to set these variables whenever a new `oval` is created. Finally, we will need to implement the `calculateArea()` and `calculatePerimeter()` functions. The area of an oval is $pi * major * minor$, and the perimeter of an oval is approximated by $2 * pi * \sqrt{\frac{major^2+minor^2}{2}}$. The `calculateArea()` and `calculatePerimeter()` functions should perform these calculations, and then set the values into the `area` and `perimeter` variables at the `shape` class level.

5. **Circle Class:** The `circle` class is similar to the `oval` class, in that we will be calculating the perimeter and area again. Instead of two variables, we only need one: `radius`, as well as a constructor to set it. Finally, the formula to calculate the perimeter of a circle is $2 * pi * radius$, and the formula to calculate to the area is $pi * radius^2$.

6. **Rect Class:** The `rect` class is a special case because it is a parent to the `square` class, but also has its own area and perimeter. We already have the `width` and `height` variables in the `quad` class, so the `rect` class only needs a constructor to set them and the functions to calculate perimeter ($2 * (width + height)$) and area ($width * height$).

7. **Trapzd Class:** The `trapzd` class will get its height and base width from the `quad` class as well, but it requires a second width (for the top) to calculate the area, and two side lengths to calculate the perimeter. Therefore, we need to create three additional variables at this level, in addition to the constructor and the area and perimeter functions. The area of a trapezoid is $height * (\frac{width+otherBase}{2})$, and the perimeter is $width + otherBase + side1 + side2$. The `trapzd` class will have three private variables, a constructor, and two public functions.

8. **Square Class:** Last but not least is the `square` class, which inherits from the `rect` class. A square is just a simple case of a rectangle where the height and width are equal. Therefore, you can set up your constructor to use either `height` or `width` for your area and perimeter calculations. Assuming that we use `width`, the formula for the area of a square is $width^2$, and the formula for the perimeter is $4 * width$. The `square` class will have a constructor and two functions.

   Note because the `square` class inherits from the `rect` class, and the `rect` constructor requires two parameters, the first line in our `square` constructor needs to convert from one to two parameters, something like `super(w, 0)` or `super(w, w)`.

9. **Sample Output with stub**

```
TESTING LINE SHAPE
------------------
Line area = 0.0
Line perimeter = 16.1245154965971

TESTING OVAL SHAPE
------------------
Oval area = 226.194671052
Oval perimeter = 59.607529593072904

TESTING CIRCLE SHAPE
--------------------
Circle area = 30.974846926448603
Circle perimeter = 19.729201863980002

TESTING RECT SHAPE
------------------
Rect area = 36.0
Rect perimeter = 30.0

TESTING TRAPZD SHAPE
--------------------
Trapzd area = 16.0
Trapzd perimeter = 11.0

TESTING SQUARE SHAPE
--------------------
Square area = 2.617924
Square perimeter = 6.472
```

Part 04 - Honor Code

Make sure to **Sign** the following statement in the `honor-code.txt` file in your repository. To sign your name, simply replace Student Name with your name. The lab work will not be graded unless the honor code file is signed by you.

This work is mine unless otherwise cited - Student Name

Submission Details

For this assignment, please submit the following to your GitHub repository by using the link shared to you by the Professor:

1. Commented source code from the "patient.py" and "patientinfo.py" programs.
2. Commented source code from the "room.py", "bedroom.py", and "stub.py"programs.
3. Commented source code from the "shape.py", "round.py", "line.py", "quad.py", "oval.py", "circle.py", "rect.py", "trapzd.py", and "sqaure.py" programs.
4. A signed honor code file, named `honor-code.txt`.
5. To reiterate, it is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus.

Grading Rubric

1. There will be full points awarded for the lab if all the requirements in the lab specification are correctly implemented. Partial credits will be awarded if deemed appropriate.
2. Failure to upload the lab assignment code to your git repo will lead you to receive no points given for the lab submission. In this case, there is no solid base to grade the work.
3. There will be no partial credit awarded if your code doesn't compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before the due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student's responsibility to let the professor know about it after the final submission in GitHub. In this way, an updated version of the student's submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then, in this case, there are no points awarded to the student.
4. If you need any clarification on your lab grade, talk to the Professor. The lab grade may be changed if deemed appropriate.