# Project 4

*Due Sunday, April 11 by 11:59pm*

For this project you will create a single file that, when run in Python, will define the following classes and all their attributes and methods. Remember that a class is like a template for representing some type of object which may then be used as part of a larger program. We will focus on just defining the classes. Please read **all instructions** before beginning the project.

First, you will define the `Bubble` class. The `Bubble` class is basically meant to represent 2-dimensional circles in the $xy$-plane that be combined to create larger circles (similar to how soap bubbles sometimes merge to form larger bubbles in real life). For this project, the words "bubble" and "circle" are interchangeable. Remember that a circle in the $xy$-plane can always be represented by the equation

$$(x-a)^2 + (y-b)^2 = r^2$$

where $(a, b)$ is the point in space that the circle is centered at, and $r$ is the radius of the circle. Your `Bubble` class definition must include the following attributes and methods. Remember that all methods in object-oriented programming must include `self` as the first (and possibly only) argument.

### Attributes and Methods

- Every class needs a constructor (or "initialization") method. The constructor method must accept 3 arguments from the user: $a$, $b$, and $r$ (as per the equation above). All 3 of these should be assigned as attributes.

  For instance, `bub = Bubble(0, 0, 1)` should create the unit circle.
  Meanwhile, `big_bub = Bubble(2, -1, 5)` should create a circle of radius 5 centered at $(2, -1)$.

- Define an `.area()` method that doesn't take any input from the user and simply returns the area of the circle.

  For instance, `bub.area()` should return $\pi \cdot 1^2 = 3.1415....$

- Define a method called `contains_point` that takes two numbers $x$ and $y$ as input from the user and checks to see if the point $(x, y)$ is contained in the bubble. It should return `True` if it is and `False` if not. Special note: if $(x, y)$ is on the boundary, the method should return `True`.

  For instance, `bub.contains_point(0.1, -0.2)` should return `True`.
  However, `bub.contains_point(5, 0)` should return `False`.

- Define a `__repr__` magic method that returns the equation of the circle in the format seen above. It doesn't have to look pretty, but it must be mathematically correct in all cases.

  For instance, `print(bub)` could print `(x - 0)^2 + (y - 0)^2 = 1^2`, which isn't simplified but is mathematically correct, so its fine.

- Define a `__sum__` magic method that takes two inputs `self` and `other`, both representing bubbles. This method should combine the two bubbles as follows. Create a new bubble (using the `Bubble` class constructor method) such that the new bubble is centered at the midpoint between the centers of the original two bubbles. The radius of the new bubble must be such that the new bubble's area is equal to the combined areas of the original two bubbles. Your magic method should return the new `Bubble` object.

  For instance, `new_bub = bub + big_bub` should create a new bubble object `new_bub` centered at $(1, -0.5)$ and with area equal to $\pi + \pi \cdot 5^2$.

  Note that we are simplifying reality by ignoring the technicalities of overlapping bubbles (in real life, if bubbles overlap then the total volume between them is not the sum of their volumes).

Second, you will create a class called `BubbleCollection`, representing a collection of bubbles. This class will be shorter, and will make use of the `Bubble` class, so make sure to write the `Bubble` class first.

**Attributes and Methods**

- The constructor method must accept 1 argument from the user: `bubble_list`, which should represent a list of `Bubble` objects. This argument should be assigned as the only attribute for this class.

  For instance, `bub_col = BubbleCollection([bub, big_bub])` should create a bubble collection consisting of 2 bubbles (the ones defined on the previous page).

- Define a method called `contains_point`. This method should be similar to and should make use of the method of the same name for the `Bubble` class, except instead of checking to see if the point $(x, y)$ is contained in a single bubble you should check to see if the point is contained in ANY of the bubbles in the `BubbleCollection` object.

  For instance, `bub_col.contains_point(5, 0)` should check if $(5, 0)$ is contained in either `bub` or `big_bub`.

- Define a method `.combine()` which adds all of the bubbles in the bubble collection together in order to get a new (and very large) `Bubble` object. The new `Bubble` object should be returned. Keep in mind there may be more than two bubbles in the collection.

  For instance, `bub_col.combine()` should return the same thing as `bub + big_bub`.

  Bonus question: does the order in which you combine the bubbles matter? Do you get a different result if you combine them in reverse order, or random order? You do not have to answer this question, but its something you can explore once your code is working.

Lastly, a few rules that apply to all projects.

- Your project must be submitted to Gradescope as a `.py` file (for instance, `Proj1.py`)
- **Name, UIC email, and Signature:** Please include your name and UIC email at the top of your submission as a comment. You must also copy and paste the following statement as a comment at the top of your file (make sure to put your name at the end):

  > I hereby attest that I have adhered to the rules for quizzes and projects as well as UIC's Academic Integrity standards. Signed: [[your full official name here]]

- ADD COMMENTS to clarify your choice of variables and to indicate what is happening at different steps of the program. The rule of thumb you should use in this course is 1 line of comments for every 3 lines of code. Your comments will be checked to see that you understand how your code is structured and what it is doing in the big picture. Code lacking in comments will be penalized!
- Even without comments, your code should be readable and clear. Avoid unneeded or slow or overly convoluted calculations. For instance, in Python it is often unnecessary to use `range(len(...))`.
- While this won't be strictly enforced, the PEP 8 style guide for Python code gives really really good advice for following proper coding conventions. Take a look at it!