Homework 4: Objects, Classes and Methods

Due February 14, 11:59 pm

Worth 15 points

Read this first. A few things to bring to your attention:

1. **Important:** If you have not already done so, please request a Flux Hadoop account. Instructions for doing this can be found on Canvas.
2. Start early! If you run into trouble installing things or importing packages, it's best to find those problems well in advance, not the night before your assignment is due when we cannot help you!
3. **Make sure you back up your work!** I recommend, at a minimum, doing your work in a Dropbox folder or, better yet, using `git`, which is well worth your time and effort to learn.

Instructions on writing and submitting your homework.
Failure to follow these instructions will result in lost points. Your homework should be written in a jupyter notebook file. I have made a template available on Canvas, and on the course website at `http://www-personal.umich.edu/~klevin/teaching/Winter2018/STATS701/hw_template.ipynb`. You will submit, via Canvas, a `.zip` file called `yourUniqueName_hwX.zip`, where `X` is the homework number. So, if I were to hand in a file for homework 1, it would be called `klevin_hw1.zip`. Contact the instructor or your GSI if you have trouble creating such a file.

When I extract your compressed file, the result should be a directory, also called `yourUniqueName_hwX`. In that directory, at a minimum, should be a jupyter notebook file, called `yourUniqueName.hwX.ipynb`, where again `X` is the number of the current homework. You should feel free to define supplementary functions in other Python scripts, which you should include in your compressed directory. So, for example, if the code in your notebook file imports a function from a Python file called `supplementary.py`, then the file `supplementary.py` should be included in your submission. In short, I should be able to extract your archived file and run your notebook file on my own machine. Please include all of your code for all problems in the homework in a single Python notebook unless instructed otherwise, and please include in your notebook file a list of any and all people with whom you discussed this homework assignment. Please also include an estimate of how many hours you spent on each of the three sections of this homework assignment.

These instructions can also be found on the course webpage at `http://www-personal.umich.edu/~klevin/teaching/Winter2018/STATS701/hw_instructions.html`. Please direct any questions to either the instructor or your GSI.

1 Still More Fun with Vectors (5 points)

In this exercise, we'll encounter our old friend the vector yet again, this time taking an object-oriented approach.

1. Define a class `Vector`. Every vector should have a dimension (a non-negative integer) and a list or tuple of its entries. The initializer for your class should take the dimension as its first argument and a list or tuple of numbers (ints or floats), representing the vector's entries, as its second argument. Choose sensible default behavior for the case where the user applies only a dimension and no entries. The initializer should raise a sensible error in the case where the dimension is invalid (i.e., wrong type or a negative number), and should also raise an error in the event that the dimension and the number of supplied entries disagree.

2. Did you choose to make the vector's entries a tuple or a list (there is no wrong answer here, although I would say one is better than the other in this context)? Defend your choice.

3. Are the dimension and entries class attributes or instance attributes? Why is this the right design choice?

4. Implement a method `Vector.dot` that takes a single `Vector` as its argument and returns the inner product of the caller with the given `Vector` object. Your method should raise an appropriate error in the event that the argument is not of the correct type or in the event that the dimensions of the two vectors do not agree.

5. We would also like our `Vector` class to support scalar multiplication. Left- or right-multiplication by a scalar, e.g., `2*v` or `v*2`, where `v` is a `Vector` object, should result in a new `Vector` object with its entries all scaled by the given scalar. We will also follow R and numpy (which you will learn in a few weeks), and use `*` to denote entrywise vector-vector multiplication, so that for `Vector` objects `v` and `w`, `v*w` results in a new `Vector` object, with the i-th entry of `v*w` equal to the i-th entry of `v` multiplied by the i-th entry of `w`. Implement the appropriate operators to support this multiplication operation. Many languages have a convention for dealing with multiplication of vectors that differ in their dimension, but we will punt on this matter. Your method should raise an appropriate error in the event that `v` and `w` disagree in their dimensions.

6. For a real number $0 \le p \le \infty$, and a vector $v \in \mathbb{R}^d$, the p-norm of v, written $\|v\|_p$, is given by

$$\|v\|_p = \begin{cases} \sum_{i=1}^d 1_{v_i \neq 0} & \text{if } p = 0 \\ (\sum_{i=1}^d |v_i|^p)^{1/p} & \text{if } 0 < p < \infty, . \\ \max_{i=1,2,\dots,d} |v_i| & \text{if } p = \infty \end{cases}$$

   Strictly speaking, this is only a norm for $p \ge 1$, but that's beside the point. Implement a method `Vector.norm` that takes a single int or float `p` as an argument and returns the `p`-norm of the calling `Vector` object. Your method should work whether `p` is an integer or float. Your method should raise a sensible error in the event that p is negative. **Hint:** see `https://docs.python.org/3/library/functions.html#float` for documentation on representing positive infinity in Python.

2 Objects and Classes: Geometry Edition (5 points)

In this problem, you'll get more practice with designing classes and their methods.

1. Here's a freebie. Implement a class `Point` that represents a point in 2-dimensional Euclidean space. Your object should include an initialization method that takes two arguments (with sensibly-chosen default values).

2. Implement the necessary operator(s) to support comparison (equality, less than, less or equal to, greater than, etc) of `Point` objects. We will say that two `Point` objects are equivalent if they have the same x- and y-coordinates. Otherwise, comparison should be analogous to tuples in Python, so that comparison is done on x-coordinates first, and then on y-coordinates. So, for example, the point $(2, 4)$ is ordered before (less than) $(2, 5)$.

3. Implement the operator to allow addition of points. That is, if `p1` and `p2` are `Point` objects, then `p1 + p2` should be supported, and should return the `Point` corresponding to adding `p1` and `p2` entrywise (i.e., vector addition).

4. Implement a class `Line` that represents a line in the 2-dimensional Euclidean plane. Implement an initialization method that takes a slope and a y-intercept as its two arguments. There are, of course, any number of ways to represent a line, and you are free to choose among them as you like, though of course the slope-intercept representation is most natural given this initializer.

5. Implement a method `Line.project` that takes a `Point` object as its only argument and returns a `Point` object representing the projection of the argument `Point` object onto the line represented by the caller. **Note:** this method should create and return a new `Point` object rather than modifying the argument.

3 Objects and Inheritance (5 points)

In this exercise, you'll get some practice with objects and inheritance by building some simple objects that might arise when dealing with a bibliography management program like BibTeX or OneNote.

1. For starters, we can't have a bibliography without authors. Define a class `Author`, with the following attributes:

   (a) `given_name`, a string representing the given name (i.e., first name in English) of the author.

   (b) `family_name`, a string representing the family name (i.e., last name in English) of the author.

   (c) `author_id`, an integer that will uniquely identify this author (and thus avoid the problem of being unable to tell apart two or more authors named John Smith.

   Write an initializer for this class that takes a first and last name as its arguments. These should both default to `None`. Your class should include a class attribute called `next_id`, which is an integer, and is initially zero. Upon initialization of a new `Author` object, the new object's `author_id` attribute should be taken to be the current value of `next_id`, and then the class attribute should be incremented so that the next ID we give out is distinct.

2. Implement the `__str__` operator for the `Author` method. The string representation of an author with first name `John` and last name `Smith` should be `'Smith, J.'`. That is, the string format should be the family name, followed by first initial. You may assume that all authors have only a first and last name, so that for our purposes, Richard W. Hamming is simply named Richard Hamming.

3. The basic unit of a bibliography is a document. For our purposes, we will assume that every document has an author, a title and a year of publication. Define a class `Document`, with the following attributes:

   (a) `author`, a list of `Author` objects.

   (b) `title`, a string.

   (c) `year`, an integer representing a year in the Gregorian calendar.

   Implement an initializer for this object, which takes a list of authors, title and year as its three arguments, in that order. The author list should default to the empty list, and the other two arguments should default to `None`.

4. Implement the `__str__` operator for the `Document` class. The string representation for a document titled *Principia* by author named Isaac Newton in the year 1687 should be: `'Newton, I. (1687). Principia.'` If there is more than one author in a document, they should be listed in the order that they are listed in the document, separated by the word "and". So the string representation for a document titled *Principia Mathematica* by authors Alfred Whitehead and Bertrand Russell in 1910 should be `'Whitehead, A. and Russell, B. (1910). Principia Mathematica.'`. Authors should be separated by the word "and" no matter how many authors are in the list `Document.author`. If any of the attributes of a document is not specified (i.e., is `None`), the `__str__` operator should raise a `ValueError`.

5. Implement a class `Book` that inherits from `Document`, but has an additional attribute `publisher`, a string naming a publishing house. Override the initializer to now take four arguments, the fourth being the string representing the publisher, and the first three being in the same order as for the `Document` initializer. Override the `__str__` method to print the same formatted string, but with the publisher name added to the end. So, in the example above, if *Principia Mathematica* were published by Cambridge University Press, then the string representation would be `'Whitehead, A. and Russell, B. (1910). Principia Mathematica. Cambridge University Press.'` .