Homework 5: Functional Programming

Due February 21, 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 Iterators and Generators (4 points)

In this exercise, you'll get some practice with creating iterators and generators. **Note:** in this problem, the word *enumerate* below is meant the sense of returning elements, not in the sense of the Python function `enumerate`. So, if I say that an iterator enumerates a sequence $a_0, a_1, a_2, \dots$, I mean that these are the elements that it returns upon calls to the `__next__` method, *not* that it returns pairs (i, a_i) like the `enumerate` function.

1. Define a class `Fibo` of iterators that enumerate the Fibonacci numbers. For the purposes of this problem, the Fibonacci sequence begins $0, 1, 1, 2, 3, \dots$, with the n-th Fibonacci number f_n given by the recursive formula $f_n = f_{n-1} + f_{n-2}$. Your solution should not make use of any function aside from addition (i.e., you should not to use the function `fibo()` defined in lecture a few weeks ago). Your class should support, at a minimum, an initialization method, a `__iter__` method (so that we can get an iterator) and a `__next__` method. **Note:** there is an especially simple solution to this problem that can be expressed in just a few lines using tuple assignment.

2. Define a generator `integers` that enumerates the nonnegative integers, in increasing order, starting with 0.

3. Define a generator `primes` that enumerates the prime numbers. Recall that a prime number is any integer $p > 1$ whose only divisors are p and 1. **Note:** you may use the function `is_prime` that we defined in class (or something similar to it), but such solutions will not receive full credit, as there is a more graceful solution that avoids declaring a separate function or method for directly checking primality. **Hint:** consider a pattern similar to the one seen in lecture using the `any` and/or `all` functions.

2 List Comprehensions and Generator Expressions (3 points)

In this exercise you'll write a few simple list comprehensions and generator expressions. Again in this problem I use the term *enumerate* to mean that a list comprehension or generator expression returns certain elements, rather than in the sense of the Python function `enumerate`.

1. Write a list comprehension that enumerates the odd squares of the integers 1 through 20 inclusive.

2. Write a generator expression that enumerates the perfect cubes, starting from 1.

3. Write a generator expression that enumerates the tetrahedral numbers. The n-th tetrahedral number is given by $T_n = \binom{n+2}{3}$, where $\binom{x}{y}$ is the binomial coefficient

$$\binom{x}{y} = \frac{x!}{y!(x-y)!}.$$

   Hint: use the generator `integers` that you defined in the previous problem.

3 Map, Filter and Reduce (4 points)

In this exercise, you'll learn a bit about map, filter and reduce operations. We will come back to these operations in a few weeks when we discuss MapReduce and related frameworks in distributed computing. In this problem, I expect that you will use only the functions `map`, `filter` and functions from the `functools` and `itertools` modules, along with the `range` function (and similar list-related functions) and a sprinkling of lambda expressions.

1. Write a one-line expression that computes the sum of the first 10 odd square numbers (starting with 1).
2. Write a one-line expression that computes the product of the first 17 primes. You may use the `primes` generator that you defined above.
3. Write a one-line expression that computes a list of the first ten harmonic numbers. Recall that the n-th harmonic number is given by $H_n = \sum_{k=1}^{n} 1/k$.
4. Write a one-line expression that computes the geometric mean of the first 10 tetrahedral numbers. You may use the generator that you wrote in the previous problem. Recall that the geometric mean of a collection of n numbers $a_1, a_2, \ldots, a_n$ is given by $(\prod_{i=1}^{n} a_i)^{1/n}$.

4 Fun with Polynomials (4 points)

In this exercise you'll get a bit of experience writing higher-order functions. You may ignore error checking in this problem.

1. Write a function `eval_poly` that takes two arguments: a number (int or float) `x`, and a list of numbers (again, ints and/or floats) `coeffs`. The elements of `coeffs` encode the coefficients of a polynomial $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$, with a_i given by `coeffs[i]`. `eval_poly` should return the value of this polynomial, evaluated at `x`. You should be able to express your solution in a single line (not counting the function definition header) by using functional programming patterns discussed in lecture.
2. Write a function `make_poly` that takes a list of numbers `coeffs` as its only argument and returns another function `p`. Here again the list `coeffs` represents the coefficients of a polynomial as in the previous question. The function `p` should take a single number (int or float) `x` as its argument, and return the value of the polynomial represented by `coeffs` evaluated at `x`. Of course, one way to solve this problem is to simply copy-paste your function definition for `eval_poly` inside the function `make_poly`, or to call `eval_poly` using the `functools.partial` function. Neither of these solutions will receive credit, because there is a more graceful solution. You should again be able to express the solution to this problem in a single line (not including the definition header).