This assignment is meant to “encourage” you to get a basic familiarity with Python. Python is one of the languages we’ll be using in the course; all of the code accompanying the textbook is in Python.

You should use the references on the course homepage to help you learn Python; if you’re familiar with a high-level language such as Java or Perl, much of Python will look pretty familiar.

What to turn in: Printouts of your source code. For this assignment, I don’t need to see any test output. (the programs are simple enough that I can tell from looking at them whether they’re right).

Task 1: Factorial:
Recall the definition of factorial: \( n! = n \times (n-1) \times (n-2) \times \ldots \times 1 \).

a) Write a recursive function to compute factorial in Python. It should take one input \( n \) and print out \( n! \). It should compute this by recursively calling itself.

b) Write an iterative function to compute factorial in Python. Again, it should take an integer as input as output the factorial of that integer. It should use \texttt{for}, \texttt{in}, and \texttt{range}.

Task 2: Dictionaries:
The Python dictionary is an extremely useful data structure. It is a generalization of a hash table; it allows you to insert a key, which is typically a string or an integer, and a value, which is a data member associated with that key.

In this assignment, we’ll make a simple phone book. The phone book will be stored in a dictionary named \texttt{phoneBook}. This should be passed in as an argument to each function you write. You can store both the name and number as a string.

1) Write functions to a) insert a new name and number b) find the number for a name c) print out all the entries (name and number) in the phone book. Your functions should handle the case where a user already has a number (print out an error message in this case) and the user is not in your phone book (also print out an error).

2) Often, people have more than one phone number. Extend your phone
book so that the value associated with each name is not a number, but a list, which is a collection of numbers. Fix your insert function so that if the user inserts a number for a user who’s already in the phone book, you store all their numbers in the tuple.

Task 3: Priority Queue:
Thinking back to your data structures class, remember that a priority queue is a special type of queue in which elements are enqueued according to priority, rather than order of arrival. A priority queue can be implemented using either a list or a heap.

1) Before starting your implementation, you’ll want to consider whether to use a heap or a list to store the queue. Give one reason for using each data structure in terms of $O()$ running time. (For example: operation X takes $O(1)$ time for a heap. Since priority queues use this operation a lot, a heap would make sense to use.)

2) Construct a queueElement class. It should have two elements: data (the thing we’re enqueueing) and priority (an integer). For now, the data field can be a string.

3) Construct a priorityQueue class. It should have methods to insert new elements, dequeue the highest priority element, return the number of elements in the queue, and return a boolean indicating whether the queue is empty. If you implement your priority queue as a list, you may need a reordering/resorting method, and if you do it as a heap, you’ll need a heapify method.

At this point, you can set the priorities for queue elements by hand.