Eric Bronner
Documentation for Moat Interview Follow-up Homework

INCR_DICT - PYTHON

- incr_dict
  - My implementation loops through the n-tuple argument and uses the value of the current iteration (usually a dictionary, which I'll refer to as *d)* to traverse the dict-tree argument, dct. The reference curr_dct is used to follow dct down its levels. If *d* is a dictionary in curr_dct, then curr_dct is replaced with *d* for future iterations. If *d* is not found in curr_dct, then *d* is added as a new dictionary to that level of dct.
  - If *d* is the last element of the n-tuple and is not found in curr_dct, instead of creating a new dictionary, *d* is added to curr_dct as an integer, and incremented to 1. If *d* is the last element of the n-tuple but is found in curr_dct, then the value is incremented.
  - If curr_dct ever becomes an integer during the traversal, an exception is thrown: the n-tuple passed is improperly formatted, as it asks incr_dict to follow the dictionary path down *through* an integer, instead of *to* one.

HASHMAP - JAVA

- Structure
  - The structure of my hashmap is a table that utilizes separate chaining to avoid collisions, and a hash function. Any data type can be used for key or value using generics, but all keys and all values must be consistent.
  - Table – The table in my implementation is an array of Linked Lists. I originally intended to use an ArrayList or Vector, but testing and Java's documentation revealed that ArrayLists and Vectors cannot be accessed at any location, unless the number of elements in the list is greater than or equal to the location I try to access (e.g., if the only element entered into an ArrayList is at index 9, if I try to access it using get(9), Java will throw an exception because size() still only returns 1).
  - To avoid collisions, every time the number of elements becomes greater than half the capacity, the table capacity is doubled and elements are reinserted.
  - Java's built-in Linked List and HashMap entries (nodes) are used, along with the built-in hashCode() method that each object in Java has. This assumes that any keys that are not built-in types will have implemented an acceptable hashCode() method, or the hash code of each key will depend on its memory address.
- Methods
  - put(key,value) – the key is hashed and converted to an index. The list at the index is extracted: it is created if it did not already exist, else it is iterated over. If the iterator finds that this key already has a value present in the table, it returns the old value and replaces it. If the key is not already in the table, it is added to the end of the list. The list is resized if the number of elements is too large.
  - get(key) – the key is hashed and converted to an index. If no list is found, then null is returned. Else, the list is iterated over. If the key is found, then its value is returned. Else, null is returned.
  - remove(key) – works almost exactly like get, but if the key is found in the table, the element is removed.
  - resize() - the list is copied into new array, and the original array is replaced with an empty array of twice the size. All elements are placed into the expanded array.