1. What is the complexity of these uses of the pop method on a Python list:
   - `L.pop()` \( O(1) \)
   - `L.pop(0)` \( O(len(L)) \)
   - `L.pop(500)` \( O(len(L) - 500) \)

2. What is the complexity of `L.reverse()`? (\( L \) is a list in Python)
   \( O(len(L)) \)

3. What is the complexity of these uses of the "in" operator on Python data structures, where \( D \) is a
   dict, \( L \) is a list, \( S \) is a set, and \( T \) is a tuple. In the complexity notation, understand "len" to be the size of the data structure being used.
   - \( x \) in \( D \) \( O(1) \)
   - \( x \) in \( L \) \( O(len) \)
   - \( x \) in \( S \) \( O(1) \)
   - \( x \) in \( T \) \( O(len) \)
   - \( x \) in \( D.values() \) \( O(len) \)

4. You can roll your own associative data structure by using two lists. One list "myKeys" contains the keys,
   the other "myVals" contains the associated values. For example, if \( myKeys[17] \) is "cat" and \( myVals[17] \)
   is "fast", then we are associating the information "fast" with "cat". To lookup a key, you just use
   \( myVals[myKeys.indexOf("cat")] \).
   Another was is to use a list of tuples, like \([("cat", "fast"), ("dog", "friendly"), ...]\).
   We can lookup a key by:
   ```python
   for k, v in myKeysVals:
       if k == "cat":
           return v
   ```
   Lastly, we can use the builtin Python data structure, dict. With a dict named \( D \), we can lookup a key with
   \( D["cat"] \).

   What are the complexities of these three methods for lookup (two lists, list of tuples, and dict)?
   In complexity notation, think of "len" as the size of the data structure being operated on, i.e. the number of
   key-value pairs.
   \( O(len), O(len), O(1) \)

5. There are two basic ways of organizing information, hashing and sorting. Python's dicts and sets use
   hashing. What is the hash of the string "cat"?
   ```bash
   $ python3 -c "print(hash('cat'))"
   -5491434775337045571
   ```
   Great, apparently "cat" hashes to -549... But wait, try it again:
Now "cat" hashes to 481... Shouldn't the hash code be stable so that we can find "cat" by its hash later?

Ah, during the course of a particular Python session, the hash remains stable. But it changes from interpreter session to interpreter session. The language designers chose to randomize the hash seed each time Python is started, to make it harder to mount a hash collision attack on Python programs; great!

But if the hash determines where keys are placed in a dict, and the hashes change from run to run, how can dicts give us a consistent ordering of their contents?

Great. Apparently those randomized hash codes don't matter and the dict will give the keys back in a consistent ordering... Let's try it on the CS Machines:

It broke! Here's the trick: my laptop uses Python3.6, the CS machines use Python3.5. In 3.6 the language designers experimented with a new dict implementation that uses a second storage array that keeps track of the insertion order. They state that it is an implementation detail, but may become part of the contract for future versions of the language.

So, should your Python program rely on the order of elements in a dict?

Only if you are willing to have your program locked in to the implementation details of a specific version of the Python interpreter

```
6. def times_two(a):
   a *= 2
   def append_number(L, number):
      L.append(number)
   def main():
      a = 7
```
times_two(a)
print(a)
L = [1, 2]
append_number(L, 3)
print(L)
What does this program print?
7 [1, 2, 3]

7. def times_two(a):
   return a * 2
def multiply_all_by_two(L):
   return [x * 2 for x in L]
def main():
   a = 7
   a = times_two(a)
   print(a)
   L = [1, 2]
multiply_all_by_two(L)
   print(L)
What does this program print?
14 [1, 2]

8. def times_two(a):
   return a * 2
def multiply_all_by_two(L):
   return [x * 2 for x in L]
def main():
   a = 7
   a = times_two(a)
   print(a)
   L = [1, 2]
   L = multiply_all_by_two(L)
   print(L)
What does this program print?
14 [2, 4]

9. The elements of sets and the keys of dicts must be immutable (not mutable) (mutable means it can mutate [change]). In Python3, identify the types that can and cannot be used as members of sets or keys of dicts (by virtue of being immutable or not):
   str immutable
   int immutable
   float immutable
   tuple immutable
   list mutable
   set mutable
   dict mutable
   frozenset immutable