Name: ______________________________ Andrew Id: ____________________

15-112 Fall 2024 Quiz 6

Up to 25 minutes. No calculators, no notes, no books, no computers. Show your work!
Do not use dictionaries, sets, try/except, or recursion on this quiz.

1. (6 points) **Code Tracing**: Indicate what the following program prints. Place your answer (and nothing else) in the box below the code.

```
def ct1(a, b, c):
    a.append(9)
    b[0] = 5
    c.append(10)
    a = [20, 21, 22]
    c[1] = 11
    b += [17]
    c = c + [27]
    print(f"a: {a}")
    print(f"b: {b}")
    print(f"c: {c}")

lst1 = [1, 2, 3]
lst2 = lst1
lst3 = copy.copy(lst1)

ct1(lst1, lst2, lst3)
print(f"lst1: {lst1}")
print(f"lst2: {lst2}")
print(f"lst3: {lst3}")
```

2. **Free Response**: Lists of Anagrams

An anagram is a word formed by rearranging the letters of another word, typically using all the original letters exactly once. For example, the word "listen" is an anagram of "silent" because both words contain the exact same letters, just arranged differently.

In this problem, you will write the function `removeNonAnagrams(lst)` two different ways: Destructive and non-destructive. Given a list of strings `lst`, calling `removeNonAnagrams(lst)` results in a list that only contains the words in `lst` that are anagrams of the first word in `lst`.

For example, if `removeNonAnagrams` is called on list
`["acres", "acers", "purple", "cares", "escar", "serac", "race", "races"]`
it results in list `["acres", "acers", "cares", "escar", "serac", "races"]`
because neither "purple" nor "race" are anagrams of "acres".

You may assume that `lst` only contains strings and will always contain at least one string.

(a) (1 point) Write the helper function `areAnagrams(s1, s2)` that takes two strings, `s1` and `s2` and returns `True` if they are anagrams and `False` otherwise.

(b) (2 points) Write the **non-destructive** function `removeNonAnagrams(lst)`. Your approach must be non-destructive in nature. You may *not*, for example, simply make a copy of `lst` and use a destructive approach on it. You may assume that you have a working implementation of `areAnagrams(s1, s2)`, even if yours does not work.

(c) (4 points) Write the **destructive** function `removeNonAnagrams(lst)`. Your approach must be destructive in nature. You may *not*, for example, use a non-destructive approach and then directly manipulate `lst` to contain the correct answer. You may assume that you have a working implementation of `areAnagrams(s1, s2)`, even if yours does not work.

3. (7 points) **Free Response**: Longest Sublist with Distinct Elements

   Write the function `longestDistinctSublist(lst)` that takes a list of integers `lst` as input and returns the longest sublist that contains only distinct elements. If there are multiple sublists with the same maximum length, return the one that appears last.

   Consider the following test cases:

```
assert longestDistinctSublist([1, 2, 3, 1, 4, 2, 7, 9, 2]) == [3, 1, 4, 2, 7, 9]
assert longestDistinctSublist([1, 2, 3, 1, 4, 1, 4, 9, 2, 4]) == [1, 4, 9, 2]
assert longestDistinctSublist([2, 1, 2, 3, 4, 1, 4]) == [2, 3, 4, 1]
assert longestDistinctSublist([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
assert longestDistinctSublist([1, 1, 1, 1]) == [1]
assert longestDistinctSublist([]) == []
```