

fullName:\_\_\_\_\_ andrewID:\_\_\_\_\_ section:\_\_\_\_\_

## 15-112 N25

### Quiz4

This is the paper version of Quiz4. You **must write your name on this paper and hand this back** in immediately after the assessment, even if you do not use it to write your answers. If we do not receive it immediately, you will receive a zero on the assessment.

**Quiz4 is located in section 6.9 of CS Academy (the end of Unit 6)**, and contains two parts. You must do Part 1 first, and you must submit it before viewing Part 2. If you cannot see the submit button, call over a TA or Mike and let us know your andrewID.

We will not grade anything you write on these pages unless you initial the box below:

\_\_\_\_\_ **Write your initials here if and only if you wish for us to grade what you have written on this paper *instead of* any answers submitted in CS Academy.** Leave the space blank if you wish for us to grade what is written in CS Academy (the default, recommended option). Note: Even if you write your answers on paper, you may not run any code related to the Part 1 problems (MCs and CTs).

You may not view any other notes, prior work, websites or resources, including any form of AI. You may not communicate with anyone else except for TAs or faculty during the assessment. All syllabus policies apply.

Some students will take this at a different time with testing accommodations. As such, you may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

**Do not recursion or anything else disallowed in the original problem.**

**Do not open this or look inside (even briefly) before we instruct you to begin. Close it once you are done.**

## Part 1[40pts total]: Multiple Choice and Code Tracing

**MC1[4 pts]**: Which of the following statements is FALSE?

- ☐ a. Sets are mutable
- ☐ b. Sets must only contain mutable values
- ☐ c. Sets can check for membership in  $O(1)$  time
- ☐ d. Sets do not store duplicates (all elements in a set are unique)
- ☐ e. Elements in a set should be considered unordered

**MC2[4 pts]**: Which of the following statements is FALSE?

- ☐ a. Searching all values in a dictionary has an efficiency of  $O(1)$
- ☐ b. Dictionaries contain key-value pairs
- ☐ c. Dictionary keys must be unique
- ☐ d. Dictionary values can be mutable or immutable
- ☐ e. A dictionary can be a value in another dictionary
- ☐ f. Finding the value for a specific key in a dictionary is  $O(1)$

**MC3[4 pts]**: What is the big-O efficiency of merge sort?

- ☐ a.  $O(1)$
- ☐ b.  $O(N)$
- ☐ c.  $O(\log N)$
- ☐ d.  $O(N \log N)$
- ☐ e.  $O(N^2)$
- ☐ f.  $O(2^N)$
- ☐ g.  $O(N^N)$

**MC4[4 pts]**: What is the big-O efficiency of selection sort?

- ☐ a.  $O(1)$
- ☐ b.  $O(N)$
- ☐ c.  $O(\log N)$
- ☐ d.  $O(N \log N)$
- ☐ e.  $O(N^2)$
- ☐ f.  $O(2^N)$
- ☐ g.  $O(N^N)$

**CT1[12 pts]:**

Indicate what the following code prints. Place your answer (and nothing else) in the box below.

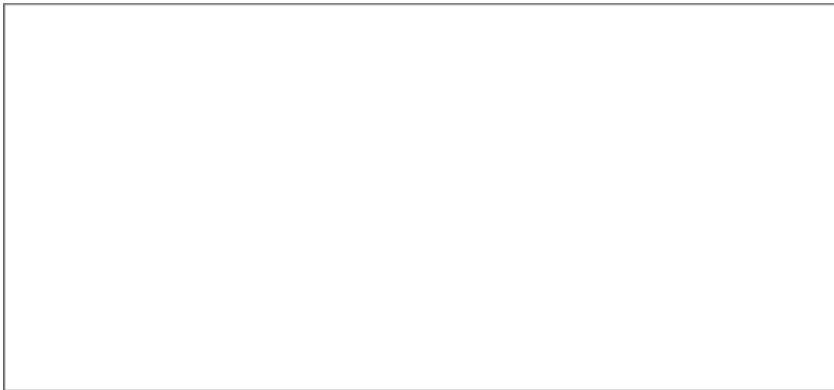
**We strongly recommend making a box-and-arrow diagram for these CTs.**

# Hint: This one is tough. Draw a box and arrow diagram!

# Note: this prints 3 lines

```
import copy
def ct(L, A, n):
    A[-1][0] += n
    A[n%2] += [10*n]
    L.append(n)
    print(A)
```

```
L = [[3], [4]]
C = copy.copy(L)
D = copy.deepcopy(L)
ct(L, C, 1)
ct(L, D, 2)
print(L)
```



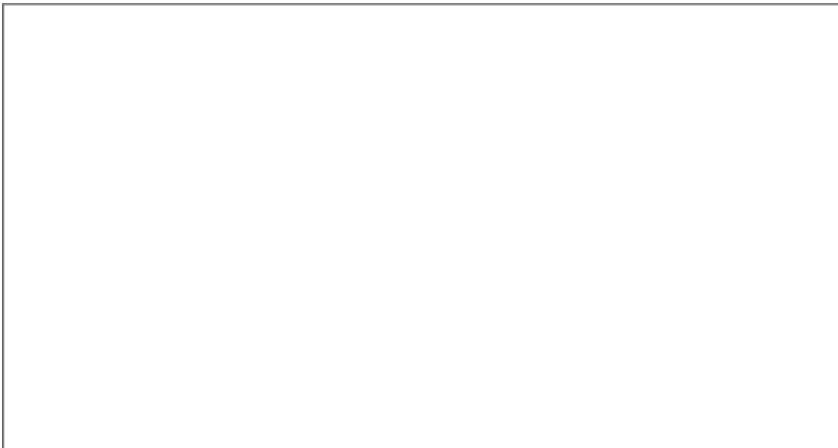
**CT2[12 pts]:**

Indicate what the following code prints. Place your answer (and nothing else) in the box below.

**We strongly recommend making a box-and-arrow diagram for these CTs.**

```
def ct(L):
    a = set()
    b = set()
    c = dict()
    d = c
    for item in L:
        if item in a:
            b.add(item)
        a.add(item)
    c[L[0]] = a
    c[L[1]] = b
    c[L[2]] = len(a)
    c[L[3]] = len(b)
    x = (d == c)
    c[L[4]] = x
    return c

print(ct([3, 'm', True, True, (5,3), 'm']))
```



## Part 2[60pts total]: Free Response

Your functions should work generally for the kinds of inputs specified in the problem statement, and we may test your code using additional test cases. We will manually grade both of these problems for partial credit if you do not pass all the test cases.

### FR1[25pts]: `findChanges(original, new)`

**Note: Your solution to this problem must have an efficiency of  $O(n)$  or better!**

Background: The TAs download the newest list of students enrolled in 112, and they need to know who (if anyone) has been added to the class and who (if anyone) has dropped it. They also have access to the original list of students enrolled at the beginning of the semester. Each list contains student names as strings, like so:

```
original = ["Jimothy Stego", "Kimchee Wiggles", "Keanu Reeves", "Daisy Ridley"]
new = ["Luz Noceda", "Jimothy Stego", "Kimchee Wiggles",
      "Raine Whispers", "Daisy Ridley"]
```

Write the nonmutating function `findChanges(original, new)` that takes two unsorted lists of names and returns a tuple containing two sets: The first contains the names of anyone who added the course, and the last contains the names of those who dropped the course. For the example above:

```
assert(findChanges(original, new) == ({"Luz Noceda", "Raine Whispers"} , {"Keanu Reeves"}))
```

...because Luz and Raine joined the class, and Keanu dropped the class.  
See the test cases for examples!

Remember: **Your solution must have an efficiency of  $O(n)$  or better**, where  $n$  is the length of the new list. Assume that the original list and the new list are approximately the same size.

Here are some test cases:

```
original = ["Jimothy Stego", "Kimchee Wiggles", "Keanu Reeves", "Daisy Ridley"]
new = ["Luz Noceda", "Jimothy Stego", "Kimchee Wiggles",
      "Raine Whispers", "Daisy Ridley"]
```

```
(added, dropped) = findChanges(original, new)
assert(added == {"Luz Noceda", "Raine Whispers"})
assert(dropped == {"Keanu Reeves"})
```

```
original = ["A", "B", "C", "D"]
new = ["A", "B", "D", "E"]
assert(findChanges(original, new) == ({"E"}, {"C"}))
```

# More on the next page

```
original = ["A", "B", "C", "D"]
new = ["D", "C", "B", "A"]
assert(findChanges(original, new) == (set(), set()))

original = ["A", "B", "C", "D"]
new = ["X", "A", "D", "Y", "C", "B", "Z"]
assert(findChanges(original, new) == ({"X", "Y", "Z"}, set()))
# Make sure the function is non-mutating:
assert(original == ["A", "B", "C", "D"])
assert(new == ["X", "A", "D", "Y", "C", "B", "Z"])
```

#Begin your answer here or on the next page

# Begin or continue your answer here

## FR2[35 pts]: makeSpeciesDictionary(animalData)

Say we are given the following 2D list of `animalData`, where each row contains exactly three elements (a species, a breed, and a name) in order:

```
animalData = [['dog', 'labrador', 'fred'],
               ['cat', 'persian', 'betty'],
               ['dog', 'shepherd', 'barney'],
               ['dog', 'labrador', 'fred'],
               ['dog', 'labrador', 'wilma']]
```

Write the **nonmutating** function `makeSpeciesDictionary(animalData)` that takes data formatted like this, and returns a dictionary mapping each species to another dictionary that maps each breed of that species to a set of the names in that table for that species.

For example, for the `animalData` given above, `makeSpeciesDictionary(animalData)` returns this:

```
{
  'dog':
    { 'labrador' : { 'fred', 'wilma' },
      'shepherd' : { 'barney' }
    },
  'cat':
    { 'persian' : { 'betty' }
    }
}
```

Here are some additional test cases:

```
animalData = [['dog', 'labrador', 'fred'],
               ['cat', 'persian', 'betty'],
               ['dog', 'shepherd', 'barney'],
               ['dog', 'labrador', 'fred'],
               ['dog', 'labrador', 'wilma']]
assert(makeSpeciesDictionary(animalData) == {
    'dog':
        { 'labrador' : { 'fred', 'wilma' },
          'shepherd' : { 'barney' }
        },
    'cat':
        { 'persian' : { 'betty' }
        }
})
```



```

animalData = [['dog','golden','Boo'],
               ['cat','orange','Kitty'],
               ['axolotl','wildType','Chee'],
               ['axolotl','leucistic','Meep'],
               ['dog','unknown','Chee']]
assert(makeSpeciesDictionary(animalData) == {
    'dog':
        {'golden': {'Boo'},
         'unknown': {'Chee'}
        },
    'cat':
        {'orange': {'Kitty'}
        },
    'axolotl':
        {'wildType': {'Chee'},
         'leucistic': {'Meep'}
        }
})

#Make sure the function is nonmutating!
assert(animalData == [['dog','golden','Boo'],
                      ['cat','orange','Kitty'],
                      ['axolotl','wildType','Chee'],
                      ['axolotl','leucistic','Meep'],
                      ['dog','unknown','Chee']])

```

We may use other test cases, including species and breeds not given here, so do not hardcode!

**Write your answer on the following page**

# Write your answer here