fullName:____________________________andrewID:____________________________ recitationLetter:_____

## 15-112 F23

# Quiz8 version B (25 min)

You **MUST** stop writing and hand in this **entire** quiz when instructed in lecture.

- You may not unstaple any pages.
- Failure to hand in an intact quiz will be considered cheating. Discussing the quiz with anyone in any way, even briefly, is cheating. (You may discuss it only once the quiz has been posted to the course website.)
- You may not use your own scrap paper. If you must use additional scrap paper, raise your hand and we will provide some. You must hand any scrap paper in with your paper quiz, and we will not grade it.
- You may not ask questions during the quiz, except for English-language clarifications. If you are unsure how to interpret a problem, take your best guess.
- You may not use any concepts (including builtin functions) which we have not covered in the notes in weeks 1-8 / units 1-6.
- You may not use recursion.
- We may test your code using additional test cases. Do not hardcode.
- Assume almostEqual(x, y) and rounded(n) are both supplied for you. You must write all other helper functions you wish to use, unless we specify otherwise.

## Multiple Choice [3pts ea, 9pts total]

**MC1.** If we wish to find whether one item exists in a very large unsorted list of length N, which of the following options would we expect to be fastest in the worst case?
Select the best answer (fill in one circle).

○ a) Performing a linear search on the unsorted list

○ b) Sorting the list with selection sort, then performing a binary search

○ c) Sorting the list with merge sort, then performing a linear search

○ d) Sorting the list with merge sort, then performing a binary search

**MC2.** Of the following, which is the least efficient?
Select the best answer (fill in one circle).

○ a) O(2**O.5)

○ b) O(2**N)

○ c) O(N**0.5)

○ d) O(log(N))

○ e) O(N*log(N))

○ f) O(N**2)

**MC3.** Which of the following may require Python to visit all N elements in the list data, assuming N = len(data)? Select ALL that apply.

☐ a)

```
for x in data:
    print(x)
```

☐ b)

```
for i in range(len(data)):
    x = data[i]
    print(x)
```

☐ c)

```
if x in data:
    print(“Found it”)
```

☐ d)

```
x = data[-1]
```

☐ e)

```
x = max(data)
```

☐ f) None of the above

## CT1: Code Tracing [8pts]

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

```
def ct1(L):
    d = dict()
    for x in range(len(L)):
        y = L[x]
        d[y] = set(L[:x])
    for x in d:
        if x in d[x]:
            d[x] = x
        else:
            d[x].add(x+1)
    return d

print(ct1([5, 2, 5, 7, 2, 4]))
```

## CT2: Code Tracing [8pts]

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

```
def ct2(s):
    s1 = set()
    s2 = set()
    for c in s:
        if c in s2:
            s2.remove(c)
        elif c in s1:
            s2.add(c)
            s1.remove(c)
        else:
            s1.add(c)
    print(f'1: {sorted(s1)}')
    print(f'2: {sorted(s2)}')

ct2('slupipupups')
```

## Free Response 1: getZeroPair(L) [35pts]

Write the function `getZeroPair(L)` which takes a list L of integers, and returns a set of two **different** integers in L that sum to zero. If no such pair exists, the function returns None. If there is more than one valid pair, you can return any of them.

For example, `getZeroPair([1, 2, 3, 10, -3])` should return the set `{-3, 3}` since these sum to 0. `getZeroPair([1, 2, 3, 10])` should return None since there is no pair of numbers in `[1, 2, 3, 10]` that sums to 0.

**Important Note:** For full credit, your solution must run in no worse than `O(N)` time, where N is the length of L.

```
def testGetZeroPair():
    assert(getZeroPair([1, 2, 3, 10, -3]) == {-3, 3})
    assert(getZeroPair([1, 2, 3, 10]) == None)
    assert(getZeroPair([-1, 2, 3, 0, 1]) == {-1, 1})
    assert(getZeroPair([-1, 2, -3, 0, -2]) == {-2, 2})
    assert(getZeroPair([0, 1, 2, 3, 0]) == None) #Integers must be different
    assert(getZeroPair([]) == None)
```

**Begin your FR1 answer here or on the next page:**

Begin or continue your FR1 answer here

## Free Response 2: findClosestMatch(people) [40pts]

Write the function `findClosestMatch(people)` which takes a dictionary mapping names of people to sets of personality traits and returns a set containing the two names who are the "closest match", meaning that those individuals share the greatest number of traits.

Here is an example dictionary:

```
people = {
    'Amy': {'Creative', 'Ambitious', 'Kind'},
    'Ben': {'Ambitious', 'Kind', 'Loyal'},
    'Claire': {'Flexible', 'Patient', 'Kind', 'Honest'},
    'Dale': {'Courageous', 'Honest'},
    'Evan': {'Creative', 'Loyal'},
    'Fred': {'Courageous', 'Resilient', 'Kind'},
    'Grace': {'Patient', 'Curious', 'Honest', 'Kind'}
}
```

In the example above, the closest match is `{'Claire', 'Grace'}` since Claire and Grace share three traits: Patient, Kind, and Honest.

The keys in people are all strings, and the values are all sets of strings.

If there are any ties for the closest match, return any set of two people who have the greatest number of shared traits. If none of the people have any shared traits, return None.

**Important note:** For full credit, your function must run in no worse than `O(N**2)` time, where N is the number of people, i.e. the number of keys in the dictionary. Assume that the number of traits for each individual stays relatively small.

```
def testFindClosestMatch():
    print('Testing findClosestMatch...')
    people = {
        'Amy': {'Creative', 'Ambitious', 'Kind'},
        'Ben': {'Ambitious', 'Kind', 'Loyal'},
        'Claire': {'Flexible', 'Patient', 'Kind', 'Honest'},
        'Dale': {'Courageous', 'Honest'},
        'Evan': {'Creative', 'Loyal'},
        'Fred': {'Courageous', 'Resilient', 'Kind'},
        'Grace': {'Patient', 'Curious', 'Honest', 'Kind'}
    }
    assert(findClosestMatch(people) == {'Claire', 'Grace'})

    #Continued on next page
```

```
#Continued on previous page

people = {
    'Steve': {'Creative', 'Ambitious', 'Kind'},
    'Steven': {'Ambitious', 'Kind', 'Loyal'},
    'Stevo': {'Flexible', 'Patient', 'Kind', 'Honest'},
}
assert(findClosestMatch(people) == {'Steve', 'Steven'})

people = {
    'Jen': {'Creative'},
    'Jenny': {'Ambitious'},
    'Jean': {'Flexible'},
}
assert(findClosestMatch(people) == None)

assert(findClosestMatch({}) == None)
```

**Begin your FR2 answer here or on the next page:**

Begin or continue your FR2 answer here

Continue your FR2 answer here