15-112
Spring 2025 Exam 2
March 27, 2025

Name:

Andrew ID:

- You may not use any books, notes, or electronic devices during this exam.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam to receive credit.
- You may use the backs of pages as scratch paper.
- All code samples run without crashing except otherwise specified. Assume any imports are already included as required.
- You may assume that `random`, `math`, `string`, `cmu_graphics` and `copy` are imported; do not import any other modules.
- If you need extra space, use the back of the pages. Do not tear off any page. Doing so may be considered an academic integrity violation.

Don't write anything in the table below.

Question	Points	Score	Time
1	20		15
2	10		7
3	20		15
4	20		15
5	15		11
6	15		12
Total:	100		75

There are 100 points on this exam, lasting 75 minutes.
You can find *estimated* completion times for each question in the table above.

1. **Code Tracing**

Indicate what each will print. Place your answer (and nothing else) in the box next to or below each block of code.

(a) (5 points) CT1

```
def ct1(L):
    M = []
    for i in range(len(L)):
        if i in L:
            j = L.index(i)
            print(j)
            M.append(sum(L[i:j]))
    return M

print(ct1([1,5,1,1,2]))
```

(b) (5 points) CT2

```
import copy
def ct2(a):
    b = a
    c = copy.copy(a)
    d = copy.deepcopy(a)
    a = [d[1], c[0]]
    a.append(42)
    b = b[::-1]
    d[1][1] = 112
    c[0][1] = 17
    c[0] = d[1].pop()
    a[0] = a[1]
    print("a =", a)
    print("b = ", b)
    print("c =", c)
    print("d = ", d)

e = [[1, 2],[3, 4]]
ct2(e)
print("e = ", e)
```

(c) (5 points) CT3

```
def ct3helper(d):
    s = set()
    for k in d:
        s.add(k)
        d[k] = k
        s.add(d[k])
    return s


def ct3(L):
    res = []
    for e in L:
        res += [ct3helper(e)]
        print(f'res {res}')

L = [{'h': 'e'}, {'l': 'lo', 4: 2}]
print(ct3(L))
print(L)
```

(d) (5 points) CT4

```
def ct4(L, d=0):
    if len(L) < 2:
        return [sum(L)]
    else:
        i = len(L)//2
        a = ct4(L[:i], d+1)
        print(f'mid {a} {d}')
        b = ct4(a, d+1)
        return a + [a[0]-b[0]] + b

print(ct4([1,3,5,9]))
```

2. (10 points) **Big-O**

For each function shown below, write the total Big-O runtime of the function in terms of N, the length of the function argument, in the box to the right of the code. **All answers must be simplified- do not include lower-order terms! For full credit, you must include line-by-line Big-O.**

```
def f1(L): # L is a list with N integers
    for p in [2, 3, 5, 8, 11, 13] :
        for k in range(len(L)):
            if L[k] % p == 0:
                L[k] = p
```

```
def f2(L): # L is a list with N integers
    S = set()
    for v in L:
        if v**2 in L:
            S.add(L.count(v**2))
    return S
```

```
def f3(D):  # D is a dictionary with N integer keys
    L = [ ]
    for k in D:
        L.append(k)
    v = max(L)
    while L.count(v) > 0:
        L.remove(v)
    return L
```

```
def f4rec(s, ix = 0):  # s is a string with N characters
    N = len(s)
    if ix >= len(s):
        return ""
    return s[ix] + f4rec(s, ix + N//4)
```

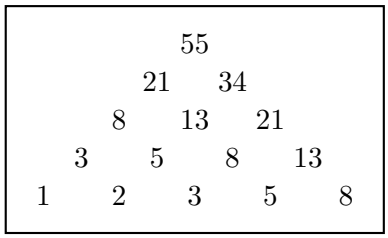
3. (20 points) **Free Response: isSumPyramid**

Write the function `isSumPyramid(P)` that determines whether the given value `P` represents a valid **sum pyramid**.

A **sum pyramid** is a triangular structure where each number (except those in the bottom-most row) is the sum of the two numbers directly below it. The pyramid is represented as a list of `n` lists, where `P[n-1]` contains the lowest level (widest row), `P[n-2]` contains the next level up, and so on. The first element of `P` (i.e., `P[0]`) contains a single number, which should be the sum of the two numbers directly below it.

Your function should return `True` if the input list `L` represents a valid sum pyramid, as shown in the test cases below. Otherwise, it should return `False`. Here are some examples:

```
P1 = [ [55],
       [21, 34],
       [8, 13, 21],
       [3, 5, 8, 13],
       [1, 2, 3, 5, 8] ]
assert isSumPyramid(P1) == True
```



Since each number in the pyramid is the sum of the two numbers directly below it, the function correctly returns `True`.

```
P2 = [ [10],  # incorrect sum, it should be 9 to return True
       [6, 3],
       [5, 1, 2] ]
assert isSumPyramid(P2) == False

P3 = [ 1, 2, 3, 6, 7,  # Not a list of lists
       [17, 24],
       [8, 9, 15] ]
assert isSumPyramid(P3) == False

P4 = [ [100],
       [42,58]
       [20, 22, 36],
       [12, 8, 14, 22],
       [9, 3, 5],  # Missing values, it should have five elements
       [8, 1, 2, 3, 6, 7] ]
assert isSumPyramid(P4) == False

P5 = "quiz7" # not even a list
assert isSumPyramid(P5) == False

P6 = [[42]]  # simple case
assert isSumPyramid(P6) == True

P7 = [[42, 2025]]  # Not a pyramid; a top row with one elem is missing
assert isSumPyramid(P7) == False

P8 = [ ["42"], # not an integer, it should be (integer) 42 to return True
       [20, 22] ]
assert isSumPyramid(P8) == False
```

Answer space for Question 3

Answer space for Question 3

4. (20 points) **Free Response: Frequent Attendees**

   Imagine you are tracking attendance at a series of Supplemental Instruction (SI) sessions. Each day's sign-in sheet is recorded as a list of participant names, and you have a list of these daily attendance records.

   Adam and Rashid are organizing the sessions and want to identify which participants attended more than one session. Anyone who attended only once should be excluded from the results.

   Your task is to write a function, `getFrequentAttendees(L)`, that identifies these participants. The function should return a set containing the names of participants who attended **more than one** SI session. If no participant attended more than once, the function should return an empty set.

   Some students may accidentally sign in two times during a single SI session. To handle this, you should ensure that each student is counted only once per session before determining the frequent attendees.

   Assume there are at most 10 SI sessions. Your solution should run in O(N), where N is the total number of students in the course.

   Here are some test cases:

```
L1 = [["Ahmad", "Layla", "Ahmad"],  # Ahmad signed in twice
      ["Ahmad", "Omar", "Hana"],
      ["Omar", "Hana", "Yusuf"]]
assert getFrequentAttendees(L1) == {"Ahmad", "Omar", "Hana"}

# Example 2: Everyone attended only once
L2 = [["Fatima", "Zayd"],
      ["Aisha", "Khalid"],
      ["Bilal", "Huda"]]
assert getFrequentAttendees(L2) == set()  # No frequent attendees

L3 = [ ["Ali", "Noor", "Ali", "Salma"],  # Ali signed in twice, should be counted once
       ["Noor", "Salma"],
       ["Omar", "Salma"]]
assert getFrequentAttendees(L3) == {"Noor", "Salma"}
```

Additional Space for Answer to Question 4

5. (15 points) **Free Response: Update List with Indices**

   Given a list `L` of elements and a list of new elements to be inserted at specific indices, your task is to write a non-mutating function `updateList(L, newElements)` that takes a list `L` and a list of tuples `newElements`, where each tuple consists of an element and the index where it should be inserted. The function should return a new list that contains all the original elements of `L` along with the new elements inserted at their respective indices. The new elements should be inserted in order of their indices, and the insertion should respect the original indices in the list. The function should not modify the original list `L`.

   For example:

```
L = ['this', 'midterm']
assert updateList(L, [('is', 1), (2, 4)]) == ['this', 'is', 'midterm', 2]
assert L == ['this', 'midterm'] # L is unchanged

L = [27, 3, 20, 25]
updateList(L, [(15112, 3), (-2, 1), (42, 0) ]) == [42, -2, 27, 15112, 3, 20, 25]
assert L == [27, 3, 20, 25] # L is unchanged

L = ['f', 'u', 'n']
assert updateList(L, [('y', 4), ('n', 2)]) == ['f', 'u', 'n', 'n', 'y']
assert L == ['f', 'u', 'n'] # L is unchanged
```

Additional Space for Answer to Question 5

6. (15 points) **Free Response: Recursive Filter Digits**

Write a **recursive** function `recFilterDigits(n, d)` that takes a positive integer `n` and a nonzero digit `d` and returns an integer where all digits in `n` that are different from `d` are replaced with 0s.

Your solution must use recursion. If you use any loops or iterative functions, you will receive no points on this problem.

This also means that your solution must NOT use any built-in functions that imply iteration like `index`, `in`, `find`, `str`. Furthermore, you cannot convert an integer into a string using `str`, or convert a string into an integer using `int`.

Hint: You can define a helper function to deal with the negative cases. **Your solution must use recursion. If you use any loops or iterative functions, you will receive no points on this problem.**

Here are some test cases:

```
assert recFilterDigits(42424242, 2) == 2020202 # leaves only 2s
assert recFilterDigits(42, 4) == 40 # leaves only the 4s
assert recFilterDigits(-15112, 1) == -10110  # leaves only the 1s
assert recFilterDigits(20250327, 2) == 20200020  # leaves only 2s
assert recFilterDigits(42, 5) == 0 # leaves 5s (but there are no 5s)
```

Additional Space for Answer to Question 6

Reference: Complexity of Python Built-ins

General	
Function/Method	**Complexity**
Print	O(N)
Range in Iteration	Number of iterations = (end - start)/step

Strings: s is a string with N characters	
Function/Method	**Complexity**
Len	O(1)
Membership	O(N)
Get single character	O(1)
Get slice	O(end - start)
Get slice with step	O((end - start)/step)
Chr() and Ord()	O(1)
Concatentation	O(len(s1) + len(s2))
Character Type Methods	O(N)
String Edit Methods	O(N)
Substring Search Methods	O(N)

Lists: L is a list with N elements	
Function/Method	**Complexity**
Len	O(1)
Append	O(1)
Extend	O(K)
Concatentation with +=	O(K), where K is the size of the list to concatenate
Concatentation with +	O(N + K), where K is the size of the list to concatenate
Membership Check	O(N)
Pop Last Value	O(1)
Pop Intermediate Value	O(N)
Count values in list	O(N)
Insert	O(N)
Get value	O(1)
Set value	O(1)
Remove	O(N)
Get slice	O(end - start)
Get slice with step	O((end - start)/step)
Sort	O(N log (N))
Multiply	O(N*D)
Minimum	O(N)
Maximum	O(N)
Copy	O(N)
Deep Copy	O(N*M)

Sets: s is a set with N elements	
Function/Method	**Complexity**
Len	O(1)
Membership	O(1)
Adding an Element	O(1)
Removing an Element	O(1)
Union	O(len(s) + len(t))
Intersection	O(min(len(s), len(t)))
Difference	O(len(s))
Clear	O(len(s))
Copy	O(len(s))

Dictionaries: d is a dictionary with N key-value pairs	
Function/Method	**Complexity**
Len	O(1)
Membership	O(1)
Get Item	O(1)
Set Item	O(1)
Delete Item	O(1)
Clear	O(N)
Copy	O(N)