

Midterm2

● Graded

Student

Colin Cano

Total Points

11 / 25 pts

Question 1

countExtras(S)

1 / 5 pts

– 0 pts Correct

– 1 pt Small error, right idea

Usually a minor syntax error or other minor issue.

– 2 pts Error that changes what the function does, can tell right idea

Bigger error, such as use of == instead of in, while rest of code seems to be on right track.

– 3 pts potential right idea with many errors

Correct framework with multiple errors

✓ – 4 pts Does not work, potential okay idea but not developed correctly

Some correct elements, but many mistakes and does not work

– 5 pts Does not work as specified, incorrect idea

Code did not provide correct answer, would need significant change to be successful

Question 2

countPeaks(S)

5.5 / 6 pts

General

- 0 pts Correct
- 5 pts Incorrect


Logic Errors

- 0.5 pts Duplicated Variable
- 1 pt Missed End Boundary
- 0.5 pts Incrementing variable in for loop
- 0.25 pts Unnecessary If statement
- 0.5 pts Too many for loops
- 0.5 pts Undeclared variable
- 0.5 pts Resetting value everytime in for loop
- 0.5 pts Unnecessary declaration of i

Iteration Errors

- 0.5 pts Poor use of Recursion
- 0.5 pts Return statement in wrong spot
- 0.75 pts Missing Return Statement
- 0.5 pts 'for i in S followed by S[i]' or 'for i in range(S) followed by S[i]'
- 0.5 pts Pass statement
- 1 pt Not comparing elements with each other
- 1.5 pts No form of iteration (no for loop, while loop, etc.)

Other Errors

-  – 0.5 pts Poor indentation
- 0.5 pts Unknown Syntax
 - 0.25 pts Unnecessary Syntax
 - 0.75 pts Infinite Loop
 - 0.5 pts Not indexing properly
 - 0.25 pts Did not use >= (used > or did >_ in math terms instead of cs terms)

 Right idea!

By having that second if statement inside the for loop, it would run every iteration, meaning the second question for example would add 1 to the count every time

✓ - 0 pts Correct

- 1 pt Incorrect

- 0.5 pts Half Correct

🗨 Nice Job

Question 3

stringify(S, c='.')

3 / 7 pts

- 0 pts Correct

- 7 pts No answer

✓ - 3 pts dosent recurse into nested sequences

- 1 pt doesn't return anything

- 1 pt doesn't index correctly

✓ - 1 pt doesn't filter out empty sequences/the empty set

- 0.5 pts syntax errors

- 1 pt Doesn't use join or do something similar

- 1 pt Poor use of iteration

- 1 pt Poor function call

- 7 pts Significantly deviates from the specification

- 1 pt Ends up with separator at the end

- 1 pt uses head tail recursion

1

might not be nested, should recurse

Question 4

findExtras(S, skip=())

1.5 / 7 pts

4.1 findExtras(S, skip=())

0 / 5 pts

– 0 pts Correct

✓ – 2 pts Not responsive to specification

2 Because you popped S[0] above, here you recurse on S[1:] which is the old S[2:], thereby skipping an element.

– 1 pt Only partially responsive to specification

✓ – 1 pt Missing or incorrect base case

6 Need to stop at ≤ 1 , or S[1:] will fail.

✓ – 1 pt Missing (or broken) recursive step for extra element

4 You've lost all the passed in skip values.

– 1 pt Missing (or broken) recursive step for non-matching element

✓ – 1 pt Syntax error

5 missing argument.

– 0.5 pts Minor syntax error

3 You need to return a set here.

4.2 Tuple as optional argument

0.5 / 1 pt

– 0 pts Correct

– 1 pt Incorrect

✓ – 0.5 pts Correct answer, poor justification.

4.3 Runtime

1 / 1 pt

✓ – 0 pts Correct

– 1 pt Incorrect

– 1 pt Blank or ambiguous

💬 $O(N^2)$ reflects $O(N)$ iterations, each scanning the list in time proportional to $O(N)$.

Name: Colin Cano

Makes list of duplicates, then returns length, which shows how many duplicates.

2. countPeaks(S) [6 points]

Specification: `countPeaks(S)` takes a sequence, `S`, and returns an integer representing the number of "peaks" in `S`. A peak is an index, $0 \leq i < \text{len}(S)$ where `S[i]` is greater than or equal to `S[i-1]` and `S[i]` is greater than `S[i+1]` (i.e., `S[i]` is followed directly by a drop in value, where `S[0]` follows `S[-1]`). **Your solution should use an appropriate form of iteration**, and should work for lists, tuples, strings, and ranges:

```
>>> countPeaks(range(10))
1
>>> countPeaks((1, 3, 2, 4, 6, 10, 9, 7, 7))
3
>>> countPeaks('AaBbCcDd')
4
```

 $S[i] \geq S[i-1]$
 $S[i] > S[i+1]$
 $\text{count} = 0$

Points may be deducted for inelegant solutions.

 $\text{len}(S)$

```
def countPeaks(S):
```

```
    count = 0
    for i in range(len(S)-1): # range len(S)-1 keeps index in range
        if S[i] >= S[i-1] and S[i] > S[i+1]:
            count += 1
```

```
    if S[-1] >= S[-2] and S[-1] > S[0]: # check last element
        count += 1
    return count
```

Assuming `S` has `N` elements, what is the runtime of your solution?

$O(1)$ ____ $O(\log(N))$ ____ $O(N)$ X $O(N \log(N))$ ____ $O(N^2)$ ____ $O(2^N)$ ____

[]

type
remove spaces**3. stringify(S, c='.') [7 points]**

Specification: **stringify(S, c)** takes a structure, S, consisting of (possibly nested) lists, tuples, sets, and other printable elements (ints, floats, or strings, but not dictionaries) and returns a string that contains all of these printable elements, in order, separated by character c. For full credit, make sure there are no extra spurious separating characters in the result, but without also dropping the 0 in the first example. So:

```
>>> stringify((0, ["paper", 3, 1], [(range(4, 16, 3))]))
'0.paper.3.1.4.7.10.13'
>>> stringify([[1, 2, 'abc', (3, 4.0, 'xyq'), 'uf'], 3], '|')
'1|2|abc|3|4.0|xyq|uf|3'
>>> stringify(({ 'alpha' }, (), [[ 'omega' ]], []), ':')
'alpha:omega' # and not 'alpha::omega'
```

As always, points may be deducted for inelegant solutions.

```
def stringify(S, c='.'):
    result = [] # empty list
    for i in S:
        for j in S: # inside lists etc
            if type(i) == int or str or float: # checks if nested
                result.append(i) # add to list
            else:
                result.append(' ' + join(j)) # make nested list into string and then put in list
    return(c + join(result)) # separates by c
```


4. findExtras(S, skip=()) [7 points]

Specification: `findExtras(S, skip=())` takes a sequence, `S`, and a tuple, `skip`, and returns the set of replicated elements in `S`, excluding any elements of `skip`. **Your solution should be strictly recursive**, and should work for lists, tuples, strings and ranges.

```
>>> findExtras((1, 2, 3, 1, 1, 4, 5))
{1}
>>> findExtras("the rain in spain falls mainly in the plains", ('the ',))
{'n', 'a', 'i', 'p', 'l', 's'}
```

You will note this problem closely resembles Problem 1, except for the extra `skip` parameter and the fact that we collect and return the set of duplicated elements rather than just counting them.

```
def findExtras(S, skip=()):
```

```
    S = S.split()
    new = []
    if len(S) == 0: # base case
        return None
    if S[0] == skip: # remove the skip from sequence
        S.pop(S[0])
    else:
        if S[0] in S[1:]: # check for duplicates
            new.append(S[0])
        else:
            S.pop(S[0])
    return (findExtras(S[1:], skip), set(new))
```

You'll notice that the default optional argument `skip` is specified as a tuple and not a list. Thinking back to our in-class discussion of Lab7, why do you think this might be a wise choice?

List are mutable

Assuming `S` has `N` elements, what is the runtime of your solution?

$O(1)$ ____ $O(\log(N))$ ____ $O(N)$ ____ $O(N \log(N))$ ____ $O(N^2)$ X $O(2^N)$ ____