**1. Collections Mystery**  
For each call below to the following function, write the value that is returned:

def mystery(a, b):

result = []

for el in a:

if a[el] > 2 or a[el] not in b:

result.append(a[el])

else:

result.append(el)

return result

|  |  |
| --- | --- |
| Function Call | Value Returned |
| mystery({2:3, 3:2, 4:4},{0, 1, 2, 23, 14, 7})  mystery({3:3, 32:2, 2:4}, {1, 2, 32, 13, 7})  mystery({4:3, 1:2, 2:4}, {3, 4}) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

2. 2D List Mystery

def mystery(data, pos, rows, cols):

     for i in range(0, rows):

         sum = 0

         for j in range(0, cols):

             sum = sum + data[pos + i][pos + j]

         print(str(sum) + " ", end='')

     print()

Suppose that a variable called grid has been declared as follows:

      grid = [[4, 6, 8, 8, 2, 1], [7, 4, 8, 8, 7, 7],

              [7, 8, 6, 6, 7, 2], [1, 2, 2, 7, 5, 7],

              [8, 3, 6, 6, 1, 1], [9, 7, 9, 6, 6, 1]]

which means it will store the following 6-by-6 grid of values:

                      4    6    8    8    2    1

                      7    4    8    8    7    7

                      7    8    6    6    7    2

                      1    2    2    7    5    7

                      8    3    6    6    1    1

                      9    7    9    6    6    1

For each call below, indicate what output is produced:

       Function Call                       Output Produced

       mystery(grid, 0, 2, 2)       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

       mystery(grid, 2, 3, 2)       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

       mystery(grid, 1, 4, 1)       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Searching and Sorting

(a) Suppose we are performing a **binary search** on a sorted list called numbers initialized as follows:

# index 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

numbers = [-9, -6, -2, -1, 0, 1, 3, 4, 5, 7, 9, 10, 12, 19, 23, 26]

# search for the value 1

index = binary\_search(numbers, 1)

Write the indexes of the elements that would be examined by the binary search (the mid values in our algorithm's code) and write the value that would be returned from the search. Assume that we are using the binary search algorithm shown in lecture and section.

• Indexes examined: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

• Value Returned: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(b) Write the state of the elements of the list below after each of the first 3 passes of the outermost loop of the selection sort algorithm.

numbers = {37, 29, 19, 48, 23, 55, 74, 12};

selection\_sort(numbers)

(c) Trace the complete execution of the merge sort algorithm when called on the list below, similarly to the example trace of merge sort shown in the lecture slides. Show the sub-lists that are created by the algorithm and show the merging of sub-lists into larger sorted lists.

numbers = [37, 29, 19, 48, 23, 55, 74, 12]

merge\_sort(numbers)

**4. String Programming**  
Write a function called undouble that takes a string as a parameter and that returns a new string obtained by replacing every pair of repeated adjacent letters with one of that letter. For example, the String "bookkeeper" has three repeated adjacent letters ("oo", "kk", and "ee"), so undouble("bookkeeper") should return the String "bokeper". Below are more sample calls:

|  |  |
| --- | --- |
| **Method Call** | **Value Returned** |
| undouble("odegaard") | "odegard" |
| undouble("oops") | "ops" |
| undouble("baz") | "baz" |
| undouble("foobar") | "fobar" |
| undouble("mississippi") | "misisipi" |
| undouble("apple") | "aple" |
| undouble("carry") | "cary" |
| undouble("juggle") | "jugle" |
| undouble("theses")  undouble("") | "theses"  "" |

You may assume that the string is composed entirely of lowercase letters, as in the examples above, and that no letter appears more than two times in a row. But notice that the method might be passed an empty string, in which case it returns an empty string.

5. 2d List Programming

Write a function named swap that accepts as parameters a list of lists, a row number and a column number. Your function should swap the passed in column and row. For example, consider the following input list of lists:

list = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]

Drawn as a grid this list of lists looks like the following:

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

A call of swap(list, 1, 1) produces the following contents:

1 5 3 4

2 6 10 14

9 7 11 12

13 8 15 16

You can assume that the list of lists is square (the length of each internal list is the same as the length of the external list).

6. Collections Programming

Write a function called record\_trip that records information about trips taken by people. Trip information will be stored in a dictionary in which the keys are names of people and the values are sets of place names. The function will take as parameters the dictionary followed by a person's name followed by a place name. For example, if we start with an empty dictionary stored in a variable called trips and we make the following calls:

record\_trip(trips, "John", "London")

record\_trip(trips, "Sally", "Seattle")

record\_trip(trips, "John", "Paris")

record\_trip(trips, "Sally", "San Francisco")

record\_trip(trips, "John", "NYC")

record\_trip(trips, "John", "Paris")

the dictionary would store the following values after these calls:

{John:[London, NYC, Paris], Sally:[San Francisco, Seattle]}

Notice that the dictionary needs to construct a set for each person to store the names of the places they have visited. The sets it constructs should store the place names. You may construct the sets that store place names, but you are not allowed to construct other structured objects (no string, set, list, etc.).

7. Collections Programming

Write a function called index\_map that takes a list of strings as a parameter and that returns a map that associates each string from the list to a list of indexes at which that string occurs. For example, if a variable called list stores the following:

[to, be, or, not, to, be]

then the call index\_map(list) should return the following map:

{be=[1, 5], not=[3], or=[2], to=[0, 4]}

Notice that each string from the original list maps to a list of indexes. For example, the string "to" appeared at index 0 and 4 in the original list, so it maps to a list with the values 0 and 4. The lists that each string maps to should have indexes in increasing order, as in the example.

Your method should construct the new map and each of the lists contained in the map but should otherwise not construct any new data structures. It should also not modify the list of words passed as a parameter and it should be reasonably efficient.

8. List Programming

Write a function named contains that accepts two lists of integers *a1* and *a2* as parameters and that returns a boolean value indicating whether or not *a2*'s sequence of elements appears in *a1* (True for yes, False for no). The sequence of elements in *a2* may appear anywhere in *a1* but must appear consecutively and in the same order. For example, if variables called list1 and list2 store the following values:

list1 = {1, 6, 2, 1, 4, 1, 2, 1, 8}

list2 = {1, 2, 1}

Then the call of contains(list1, list2) should return True because list2's sequence of values {1, 2, 1} is contained in list1 starting at index 5. If list2 had stored the values {2, 1, 2}, the call of contains(list1, list2) would return False because list1 does not contain that sequence of values. Any two lists with identical elements are considered to contain each other, so a call such as contains(list1, list1) should return True.

You may assume that both lists passed to your function will have lengths of at least 1. You may not use any Strings to help you solve this problem, nor functions that produce Strings such as str().

9. Programming

Write a function called distribute\_tokens that takes a list of token counts and a player index and that distributes the tokens for the given player in a particular way. The idea is that a set of players are playing a game where they accumulate tokens and we have a list that keeps track of how many tokens each player has.

The function should distribute tokens as follows. All of the tokens for the given player are collected and removed so that they can be distributed to other players. Tokens are distributed one at a time starting with the player that comes after the given player until all tokens have been distributed. If you reach the last player in the list, then distribution of tokens proceeds starting at the front of the list at index 0.

For example, suppose that the current token counts are stored in an list called tokens as follows:

[3, 2, 5, 10, 1]

And suppose we make this call:

distribute\_tokens(tokens, 1)

The function is being asked to distribute the tokens of the player at index 1. That player has 2 tokens that are given to the players who come after, leaving you with this list:

[3, 0, 6, 11, 1]

Suppose that we then make this call:

distribute\_tokens(tokens, 3)

The function is being asked to distribute the tokens for the player at index 3. That player has 11 tokens which are distributed one at a time leaving you with these token counts:

[5, 2, 8, 2, 4]

Notice that the player with index 3 ends up getting two of the tokens being distributed.

You may assume that the list is not empty, that all list values are non-negative integers, and that the player index is a legal index of the list.

# Solutions

|  |  |
| --- | --- |
| 1.  [3, 3, 4]  [32, 4, 3]  [2, 4, 3]  2.  Function Call         Output Produced     --------------------------------------     mystery(grid, 0, 2, 2)     10 11     mystery(grid, 2, 3, 2)     12 9 12     mystery(grid, 1, 4, 1)     4 8 2 3 | 3.   1. Indexes examined: 7, 3, 5   Value returned: 5  (b) {**12**, 29, 19, 48, 23, 55, 74, 37}  {12, **19, 29**, 48, 23, 55, 74, 37}  {12, 19, **23**, 48, **29**, 55, 74, 37}  (c) {37, 29, 19, 48, 23, 55, 74, 12}  {37, 29, 19, 48} {23, 55, 74, 12} split  {37, 29} {19, 48} {23, 55} {74, 12} split  {37} {29} {19} {48} {23} {55} {74} {12} split  {29, 37} {19, 48} {23, 55} {12, 74} merge  {19, 29, 37, 48} {12, 23, 55, 74} merge  **{12, 19, 23, 29, 37, 48, 55, 74}** merge |
|  |  |

4.

def undouble(s):

result = ""

if (len(s) > 0):

last = s[0]

result += last

for i in range(0, len(s)):

if (s[i] != last):

result += s[i]

last = s[i]

return result

5.

def swap(list, row, col):

for i in range(0, len(list)):

temp = list[row][i]

list[row][i] = list[i][col]

list[i][col] = temp

6.

def record\_trip(trips, person, place):

if person not in trips

trips[person] = set()

trips[person].add(place)

7.

def index\_map(data):

result = {}

for i in range(0, len(data)):

next = data[i]

if (next not in result):

result[next] = []

result[next].append(i)

return result

8.

def contains(a1, a2):

for i in range(0, len(a1)):

found = True

for j in range(0, len(a2)):

if (a1[i + j] != a2[j]):

found = False

if (found):

return True

return False

9.

Two possible solutions appear below.

def distribute\_tokens(tokens, player):

n = tokens[player]

tokens[player] = 0

for i in range(1, n + 1):

tokens[(player + i) % len(tokens)] += 1

def distribute\_tokens(tokens, player):

num = tokens[player]

tokens[player] = 0

while num > 0:

player += 1

if player == len(tokens):

player = 0

tokens[player] += 1

num -= 1