Python exercises

Checking Groups

In English and programming, groups can be made using symbols such as () and {} that change meaning. However, these groups must be closed in the correct order to maintain correct syntax.

Your job in this kata will be to make a program that checks a string for correct grouping. For instance, the following groups are done correctly:

({})
[[]()]
[{()}]

The next are done incorrectly:

{(})
([]
[])

A correct string cannot close groups in the wrong order, open a group but fail to close it, or close a group before it is opened.

Your function will take an input string that may contain any of the symbols (), {} or [] to create groups.

It should return True if the string is empty or otherwise grouped correctly, or False if it is grouped incorrectly.

Digitize

Given a non-negative integer, return an array / a list of the individual digits in order.

Examples:

123 => [1,2,3]

1 => [1]

8675309 => [8,6,7,5,3,0,9]

Find Count of Most Frequent Item in an Array

Complete the function to find the count of the most frequent item of an array. You can assume that input is an array of integers. For an empty array return 0

input array: [3, -1, -1, -1, 2, 3, -1, 3, -1, 2, 4, 9, 3]

ouptut: 5

The most frequent number in the array is -1 and it occurs 5 times.

Square Matrix Multiplication

Write a function that accepts two square (NxN) matrices (two dimensional arrays), and returns the product of the two. Only square matrices will be given.

How to multiply two square matrices:

We are given two matrices, A and B, of size 2x2 (note: tests are not limited to 2x2). Matrix C, the solution, will be equal to the product of A and B. To fill in cell [0][0] of matrix C, you need to compute: A[0][0] * B[0][0] + A[0][1] * B[1][0].

More general: To fill in cell [n][m] of matrix C, you need to first multiply the elements in the nth row of matrix A by the elements in the mth column of matrix B, then take the sum of all those products. This will give you the value for cell [m][n] in matrix C.

Example

A B C

|1 2| x |3 2| = | 5 4|

|3 2| |1 1| |11 8|

Detailed calculation:

C[0][0] = A[0][0] * B[0][0] + A[0][1] * B[1][0] = 1*3 + 2*1 = 5

C[0][1] = A[0][0] * B[0][1] + A[0][1] * B[1][1] = 1*2 + 2*1 = 4

C[1][0] = A[1][0] * B[0][0] + A[1][1] * B[1][0] = 3*3 + 2*1 = 11

C[1][1] = A[1][0] * B[0][1] + A[1][1] * B[1][1] = 3*2 + 2*1 = 8

Link to Wikipedia explaining matrix multiplication (look at the square matrix example): http://en.wikipedia.org/wiki/Matrix_multiplication

A more visual explanation of matrix multiplication: http://matrixmultiplication.xyz

N smallest elements in original order

Your task is to write a function that does just what the title suggests (so, fair warning, be aware that you are not getting out of it just throwing a lame bas sorting method there) with an array/list/vector of integers and the expected number n of smallest elements to return.

Also:

- the number of elements to be returned cannot be higher than the array/list/vector length;
- elements can be duplicated;
- in case of duplicates, just return them according to the original order (see third example for more clarity).

Same examples and more in the test cases:

first_n_smallest([1,2,3,4,5],3) == [1,2,3]

first_n_smallest([5,4,3,2,1],3) == [3,2,1]

first_n_smallest([1,2,3,4,1],3) == [1,2,1]

first_n_smallest([1,2,3,-4,0],3) == [1,-4,0]

first_n_smallest([1,2,3,4,5],0) == []

Performance version by FArekkusu also available.

Thinkful - List and Loop Drills: Inverse Slicer

You're familiar with list slicing in Python and know, for example, that:

>>> ages = [12, 14, 63, 72, 55, 24]

>>> ages[2:4]

[63, 72]

>>> ages[2:]

[63, 72, 55, 24]

>>> ages[:3]

[12, 14, 63]

write a function inverse_slice() that takes three arguments: a list items, an integer a and an integer b. The function should return a new list with the slice specified by items[a:b] *excluded*. For example:

>>>inverse_slice([12, 14, 63, 72, 55, 24], 2, 4)

[12, 14, 55, 24]

The input will always be a valid list, a and b will always be different integers equal to or greater than zero, but they *may* be zero or be larger than the length of the list.

Don't Drink the Water

Given a two-dimensional array representation of a glass of mixed liquids, sort the array such that the liquids appear in the glass based on their density. (Lower density floats to the top) The width of the glass will not change from top to bottom.

==========

| Density Chart |

==========

| Honey | H | 1.36 |

| Water | W | 1.00 |

| Alcohol | A | 0.87 |

| Oil | O | 0.80 |

```
[                       [
 ['H', 'H', 'W', 'O'],       ['O','O','O','O']
 ['W', 'W', 'O', 'W'],  =>    ['W','W','W','W']
 ['H', 'H', 'O', 'O']        ['H','H','H','H']
 ]                       ]
```

Who won the election?

In democracy we have a lot of elections. For example, we have to vote for a class representative in school, for a new parliament or a new government.

Usually, we vote for a candidate, i.e. a set of eligible candidates is given. This is done by casting a ballot into a ballot-box. After that, it must be counted how many ballots (= votes) a candidate got.

A candidate will win this election if he has the absolute majority.

Your Task

Return the name of the winner. If there is no winner, return *null* (in Java and JavaScript), *None* (in Python) or *nil* (in Ruby).

Task Description

There are no given candidates. An elector can vote for anyone. Each ballot contains only one name and represents one vote for this name. A name is an arbitrary string, e.g. "A", "B", or "XJDHD".

There are no spoiled ballots.

The ballot-box is represented by an unsorted list of names. Each entry in the list corresponds to one vote for this name. You do not know the names in advance (because there are no candidates).

A name wins the election if it gets more than n/2 votes (n = number of all votes, i.e. n is equal to the size of the given list).

Examples

#3 votes for "A", 2 votes for "B" -> "A" wins the election

getWinner(["A", "A", "A", "B", "B"]) == "A" #true

#2 votes for "A", 2 votes for "B" -> No winner

getWinner(["A", "A", "B", "B"]) == None #true

#1 vote for each name -> No winner

getWinner(["A", "B", "C", "D"]) == None #true

#3 votes for "A", 2 votes for "B", 1 vote for "C"

#-> No winner ("A" does not have more than n/2 = 3 votes)

getWinner(["A", "A", "A", "B", "B", "C"]) == None #true