

Homework 3

Functions and Matrices

General instructions (updated for functions)

- Read the questions **carefully** and make sure your programs work according to the requirements.
- The homework needs to be done individually!
- Read the submission rules on the course web page. All the questions need to be submitted together in the file `ex3_012345678.py` attached to the homework, after changing the number 012345678 with your ID number (9 digits, including check digit).
- How to write the solution: in this homework, you need to implement the required functions. **In the outline file, for each question replace the keyword *pass* with your code of the required function.**
- Final submission date: see course web page.
- Check your code: in order to ensure correctness of your programs and their robustness in the presence of faulty input, for each question run your program with a variety of different inputs, those that are given as examples in the question and additional ones of your choice (check that the output is correct and that the program does not crash).
- **Please make sure to respect the exact format of the output (including spaces).**
- **Note that in most questions, the functions that you will write need to return values. In all the functions, you should not print values!**
- **You are not allowed to change the names of the functions and variables that already appear in the attached outline file.** You are allowed to add additional variables and functions.
- **You are not allowed to add an indentation to a line where a function is defined** (in order to ensure that the function remains in the global scope).
- You may not erase the instructions (comments) present in the outline.
- Unless stated otherwise, you can suppose that the input received by the functions is correct.
- **You are not allowed to use external modules, such as numpy, for this exercise.**

Question 1

Implement a function called `sum_divisible_by_k(lst, k)` which receives a list of numbers called `lst` and a positive number `k`, and returns the sum of all numbers in the list that are divisible by `k`.

- If there is no number divisible by `k` (for example when the list is empty), you will return 0.

Example 1:

```
>>> result = sum_divisible_by_k([3, 6, 4, 10, 9], 3)
```

```
>>> print(result)
```

```
18
```

Explanation: among the numbers in the list, those that are divisible by 3 are: 9, 6, 3. Their sum is 18.

Example 2:

```
>>> result = sum_divisible_by_k([45.5, 60, 73, 48], 4)
```

```
>>> print(result)
```

```
108
```

Question 2

Implement a function called `mult_odd_digits(n)` which receives a positive integer `n` and returns the product of its digits that are odd.

- If there are no odd digits, return 1.

Examples:

```
>>> mult_odd_digits(5638)
```

```
15
```

Explanation: the odd numbers in 5638 are 5 and 3. Their product is 15.

```
>>> mult_odd_digits(2048)
```

```
1
```

Explanation: there are no odd numbers in 2048, hence the function returns 1.

```
>>> mult_odd_digits(54984127)
```

```
315
```

Question 3

Implement a function called *count_longest_repetition(s, c)* which receives a string called *s* and another string called *c* containing only one character. The function should return the length of the longest sequence in *s* that contains only the character *c*.

- If the given character *c* is not found in the string *s*, you should return 0.

Examples:

```
>>> s = 'ebbbaaaacccaaddd'
```

```
>>> count_longest_repetition(s, 'a')
```

```
4
```

Explanation: there are three sequences of the character 'a' in *s*. In the first one, 'a' occurs only once (colored red above), in the second sequence it appears 4 times (colored blue), in the third sequence it appears twice (colored green). The longest sequence contains 4 times 'a', hence we return 4.

```
>>> count_longest_repetition('cccccc', 'c')
```

```
6
```

```
>>> count_longest_repetition('abcde', 'z')
```

```
0
```

Question 4

Implement a function called *upper_strings(lst)* a single parameter called *lst* and works thus:

- If *lst* is not a list, the function returns -1
- Otherwise, the function replaces every item in the list which is a string by a string with the lowercase letters converted to uppercase. In this case, the function does not return anything but will rather modify the list given as parameter. **You should not change items that are not strings.**

Hint: you can use the function *type* to get the type of a value and to test equality between types.

Examples:

```
>>> vals = [11, 'TeSt', 3.14, 'cAsE']
```

```
>>> upper_strings(vals)
```

```
>>> print(vals)
```

```
[11, 'TEST', 3.14, 'CASE']
```

Explanation: the input list contains two items that are strings and two items that are numbers (*int* and *float*). The function replaced each of the two strings in the input with a new string that contains the same characters but in uppercase.

'TeSt' → 'TEST'

'cAsE ' → 'CASE'

For the two items that are not strings, they remain the same.

```
>>> vals = [-5, None, True, [1, 'dont change me', 3]]
```

```
>>> upper_strings(vals)
```

```
>>> print(vals)
```

```
[-5, None, True, [1, 'dont change me', 3]]
```

Explanation: in this case, no item in the list is a string, hence no item in the list is changed (the last item in the list is also a list, hence according to the definition of the function it should not be changed, even if it contains a string inside it).

```
>>> upper_strings(42)
```

```
-1
```

```
>>> upper_strings('im not a list')
```

```
-1
```

```
>>> upper_strings(False)
```

```
-1
```

Matrices

In the following questions we will work with two-dimensional lists of numbers that represent matrices. A matrix with dimension $n * m$ (i.e. a matrix of n rows, where each row has a length of m) will be represented by n lists of length m , where each list represents one row. For example, a matrix of 3 rows and 2 columns with the following values:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

will be represented by the following list:

`[[1,2], [3,4], [5,6]]`

Question 5

Implement a function called `div_mat_by_scalar(mat, alpha)` which receives a valid matrix called `mat` and an integer called `alpha` and returns a new matrix with the same dimensions as `mat` where each item of the new matrix is the division of the corresponding item in `mat` divided by `alpha` (using integer division).

- **You should return a new matrix, without modifying `mat`.**
- You need to use integer division (`//`) and not rounding after the division.
- You can assume that `alpha` is different than 0.
- You can assume that the input matrix is not empty.

Examples:

```
>>> mat1 = [[2,4], [6,8]]
>>> mat2 = div_mat_by_scalar(mat1, 2)
>>> print(mat1)
[[2, 4], [6, 8]]
>>> print(mat2)
[[1, 2], [3, 4]]
>>> div_mat_by_scalar([[10,15], [-3,6]], -5)
[[-2, -3], [0, -2]]
```

Question 6

Implement a function called `mat_transpose(mat)` which receives a valid matrix called *mat* and returns a new matrix which is the transpose of matrix *mat*. Matrix transposition consists in switching the rows and columns of the matrix. The function will receive a matrix with *n* rows and *m* columns, and will return a matrix with *m* rows and *n* columns that contains at position (i,j) the item that was contained at position (j,i) of the original matrix.

Here is an example of a matrix and its transpose:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}^T \rightarrow \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

- You must return a new matrix, without changing *mat*.
- You can assume that the input matrix is not empty.

Examples:

```
>>> mat = [[1,2],[3,4],[5,6]]
>>> mat_T = mat_transpose(mat)
>>> print(mat)
[[1, 2], [3, 4], [5, 6]]
>>> print(mat_T)
[[1, 3, 5], [2, 4, 6]]
```

```
>>> mat2 = [[0, 1, 2], [10, 11, 12], [20, 21, 22]]
>>> mat2_T = mat_transpose(mat2)
>>> print(mat2_T)
[[0, 10, 20], [1, 11, 21], [2, 12, 22]]
```