Algorithms & Data Structures

Recursion

Write all your code for questions 1 to 6 in a file recursion.py.

# Exercise 1:

Write a **recursive** function ispalindrome(word) that returns true if the string word is a palindrome, false otherwise. You can start with an implementation that does not deal with punctuation, and then refactor your code to consider punctuation.

# Exercise 2:

To compute the sum of all elements in a list, you can use the built-in function sum.

For example:

>>> sum([1,2,3,4])

10

>>> sum([])

0

Write a recursive function rec\_sum(numbers) that take a list of integers as a parameter and returns the sum of all the elements in the list. The function should return 0 if the list is empty.

# Exercise 3: (from **week 3** practical)

During week 3, we implemented the **function** sum\_digits(number) to calculate and return the sum of digits of a given whole number (**int**) given as parameter. For example,

>>> print(sum\_digits(1234))

10

At the time we used loops in our implementation. This time you **must** use **recursion**. In addition, you are not allowed to convert the int into a list or a string.

# Exercise 4:

Write a recursive function flatten(mlist) where mlist is a multidimensional list that returns all the element from mlist into a one-dimensional list. Note, empty lists are ignored. For examples:

>>> flatten([1,[2,3],4])

[1,2,3,4]

>>> flatten([1,[2,[3,[4]]]])

[1,2,3,4]

>>> flatten([1,2,3,4])

[1,2,3,4]

>>> flatten([1,[]])

[1]

# Exercise 5:

Write a **recursive** function merge(sorted\_listA, sorted\_listB) that merges the two **sorted** lists into a single **sorted** list and returns it. The two parameters are list of comparable objects that are sorted in ascending order. For example, the lists contain only strings, or the lists contain only numbers. Neither of the two lists in the parameters must be modified.

# Exercise 6: An unexpected coding journey

A word is considered **elfish** if it contains all the letters: **e**, **l**, and **f** in it, in any order. For example, we would say that the following words are elfish: *whiteleaf*, *tasteful*, *unfriendly*, and *waffles*, because they each contain those letters.

1. Write a predicate function called iselfish(word) that, given a word, tells us if that word is elfish or not. The function must be **recursive**.
2. Write something\_ish(pattern, word)a more generalized predicate function that, given two words, returns true if all the letters of pattern are contained in word.

I did not provide a unit test for this exercise, if you wish you could try to create a unit test for that exercise and share it with someone else to test their code.