Python for Engineers 0509-1820, Semester 1 2019-2020

Homework 5

General instructions

- 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 ex5_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 complete the code in the attached outline file.
- 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).
- The questions are checked automatically. You thus need to write your solutions only in the specified spaces in the outline file.
- Unless stated otherwise, you can suppose that the input received by the functions is correct.
- You are not allowed to change the names of the functions and variables that already appear in the attached outline file.
- You are not allowed to erase the instructions that appear in the outline file (except the lines that contain the keyword “pass”).
- You are not allowed to use outside libraries (you may not use **import**)
- Final submission date: see course web page.

Special instructions for homework 5

- **All the functions that you write need to be recursive (non-recursive functions will not be accepted)**
- **Loops are not allowed (*for*, *while*)**
- **You can assume that the input is correct. For example, if it is stated that the function receives a non-empty list of non-negative numbers, you can assume that such is the input you will receive, and you do not need to check it in your code.**
- **To ease your work, we have introduced tests at the end of the outline file (bear in mind thought that these tests do not necessarily cover all the possible cases).**

Question 1

Write a recursive function *reverse_string* which receives a string as input and returns the same string with the characters in reverse order.

Note

You may use slicing for the recursive call, but you may not reverse the string using slicing (with a negative step). You may not use any other function which realizes string inversion, like the function *reverse*.

Examples:

```
>>> reverse_string("abc")
'cba'
>>> reverse_string("Hello!")
'!olleH'
```

Question 2

Write a recursive function called `find_maximum` which receives a list of non-negative numbers (zero and above) and returns the maximum among them. If the list is empty, the function will return -1.

Note: you may not use the built-in function *max*.

Examples:

```
>>> find_maximum([9,3,0,10])
10
>>> find_maximum([9,3,0])
9
>>> find_maximum([])
-1
```

Instructions:

In each step of the recursion, if we divide the list into the last element and the rest of the list, there are two possibilities:

- The last element of the list is the largest one
- The largest element is located in the rest of the list

Thus, in each step of the recursion we will compute the values of the two possibilities, and we will return the largest one among them.

Question 3

A string is called a "palindrome" if reading it from the beginning to the end or from the end to the beginning gives the same sequence of characters. We assume that a palindrome has at least one character, and the first character is identical to the last one, the second one is identical to the one before last, etc. The characters must be exactly identical, i.e. *Aba* is not a palindrome because the letter *A* is not identical to the letter *a*.

For example, the strings 'abba' and 'a' are palindromes but the string 'abcab' is not a palindrome.

You are asked to write a function called `is_palindrome` which receives a non-empty string and returns *True* if the string is a palindrome and *False* if it is not.

Note: you are not allowed the invert the string, or part of the string, in your solution.

Examples:

```
>>> is_palindrome("aa")
True

>>> is_palindrome("aa ")
False

>>> is_palindrome("caca")
False

>>> is_palindrome("abcbbcba")
True
```

Instructions: a (non-empty) string is a palindrome if the following two conditions are met:

- The first character is identical to the last character.
- The substring comprising all the characters from the second one to the character before the last one is also a palindrome.

Thus, *abcba* is a palindrome because the first character is equal to the last character and, in addition, the substring *bcb* is also a palindrome. This is a recursive definition that reduces the size of the string by two characters at each recursive call. You also need to add the adequate base condition(s) to your function.

Question 4

You need to climb a staircase containing *n* steps (a strictly positive number). At each moment during the climbing, you can choose whether to climb one step or two steps at a time. How many different ways are there to climb the whole *n* steps?

Write a recursive function called `climb_combinations` which receives a strictly positive number *n* and returns the numbers of ways to climb a staircase comprising *n* steps.

Examples:

```
>>>climb_combinations(3)
3
```

Explanation: there are exactly 3 ways to climb a staircase of 3 steps:

- Climbing one step three times
- Climbing two steps at a time, then climbing one step
- Climbing one step, then climbing two steps at a time

```
>>>climb_combinations(10)
89
```

Question 5

A correct string of parentheses is a string in which to every left parenthesis '(' corresponds a right parenthesis ')' later in the string, and vice-versa. In other words, each time one opens a parenthesis (with the left parenthesis character) one needs to close the parenthesis in the sequel (with a right parenthesis character) and it is forbidden to close a parenthesis if there does not appear before it a yet unclosed open parenthesis. For example, the string '((()()))' is correct but the following strings are not correct:

```
')'
'('
') ('
```

The following strings are also incorrect:

```
'(()'
```

Explanation: note that each right parenthesis should close only one left parenthesis, exactly in the way we are used to use parentheses. Thus, in the above example, there remains one unclosed left parenthesis, thus the string is not correct.

'())() '
Explanation: the right parenthesis shown in blue is not correct because the only left parenthesis before it is already closed.

Write a recursive function called `is_valid_paren` which contains a string made of several characters (not necessarily parentheses) and returns *True* if the string is correct with respect to its parentheses and *False* otherwise. The function will receive an additional value called *cnt*, which will help in counting the left and the right parentheses (see instructions below).

The header of the function provided in the outline file defines the default value of the *cnt* parameter to 0. This means that if we call the function without providing an explicit value for *cnt*, then its value will be automatically set to 0.

```
def is_valid_paren(s, cnt=0):
```

Examples:

```
>>> is_valid_paren("(.(a)")
False

>>> is_valid_paren("p(()r((0)))")
True
```

The last call is equivalent to the call:

```
is_valid_paren("p(()r((0)))", 0)
```

Instructions: when you go over the string, at the moment you encounter an opening parenthesis, that is the character '(', you need to check that in the sequel the parenthesis gets closed with the character ')'. You also need to take into consideration the case where you encounter a right parenthesis without a matching left parenthesis, for example the third character in the string "())(". In this example, the number of open parentheses is equal to the number of closed parentheses, but yet the string is not correct. Use *cnt* in order to count the number of open parentheses that have not been closed yet.