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

Homework 6

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 ex1_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**), unless stated otherwise.
- The subject of the homework is recursion and memoization. **Non-recursive solutions will not be accepted**. Note also that in all functions, you need to return a value, and not to print it.
- Final submission date: see course web page.

Question 1

We saw in the course the Fibonacci series, where the element at index 0 is 0, the element at index 1 is 1, and every other element is the sum of the two preceding ones. The first element in the series are:

0,1,1,2,3,5,8,13,21,34…

In the same way we can define the series "four-Fibonacci", in which the element at index 0 is 0, the element at index 1 is 1, the one at index 2 is 2 and the one at index 3 is 3. Every other element is the sum of the 4 preceding elements. The first elements in the series are:

0,1,2,3,6,12,23,44,85…

The element at index 4 is 6=0+1+2+3, the element at index 5 is 12=1+2+3+6, etc.

Part A

Implement the function *four_bonacci_rec(n)* which receives an integer *n* and returns the element located at index *n* in the series "four-Fibonacci".

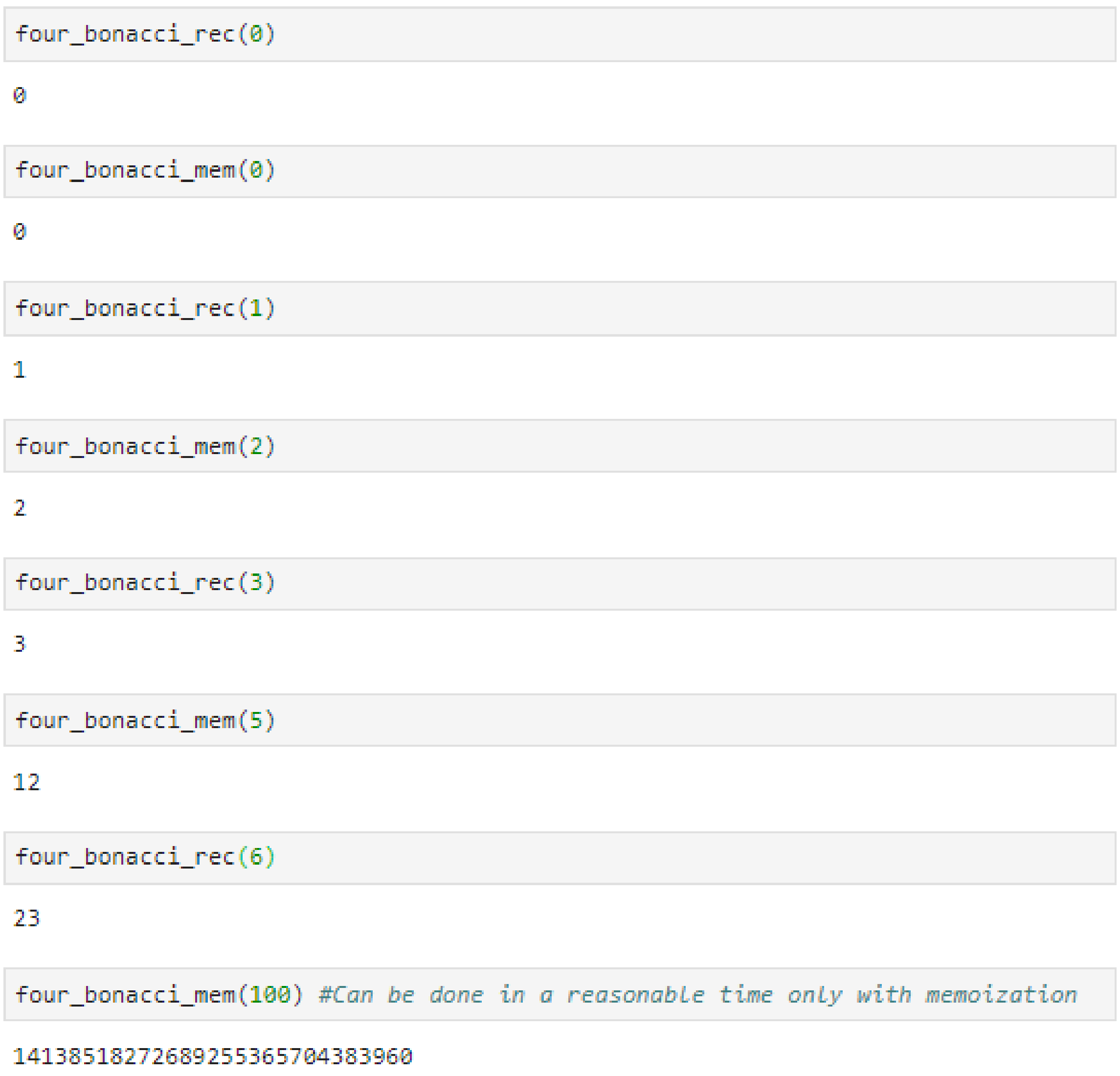
- The implementation must be **recursive and without memoization.**
- You can assume that n>=0 and *n* is an integer.

Part B

Implement the function *four_bonacci_mem(n,mem=None)* which receives an integer *n* and returns the element located at index *n* in the series "four-Fibonacci". This function will use memoization in order to avoid repeated computations of the same value, and thus to improve the running time of the function. Therefore, the function receives an additional argument: *memo*.

- The implementation must be **recursive with memoization.**
- You can assume that n>=0 and *n* is an integer.

Examples of execution:



```
four_bonacci_rec(0)
```

```
0
```

```
four_bonacci_mem(0)
```

```
0
```

```
four_bonacci_rec(1)
```

```
1
```

```
four_bonacci_mem(2)
```

```
2
```

```
four_bonacci_rec(3)
```

```
3
```

```
four_bonacci_mem(5)
```

```
12
```

```
four_bonacci_rec(6)
```

```
23
```

```
four_bonacci_mem(100) #Can be done in a reasonable time only with memoization
```

```
14138518272689255365704383960
```

Complement (no need to submit):

Compare the running times of the two functions using different values of *n* (in the following example *n*=28), using the following piece of code:

```
from timeit import default_timer as timer
n = 28
start = timer ( )
four_bonacci_rec(n=n)
end =  timer ( )
print('Time without memoization for ',n,':',end - start)
start = timer ( )
four_bonacci_mem(n=n)
end = timer ( )
print('Time with memoization for ',n,':',end - start)
```

Question 2

Remember question 4 of the preceding homework: you needed to climb a staircase with *n* steps (with integer *n* >0). Each time you can choose whether to climb one step only or two steps at a time. In how many different ways can you climb the whole staircase?

Write the **recursive** function *climb_combinations_memo(n,memo=None)* which receives *n* and returns the number of different ways to climb the staircase. This function should use **memoization** in order to avoid repeated computation and thus to improve the running time of the function you write in the preceding homework. Therefore, the function receives an additional parameter: *memo*.

Examples of execution:

```
climb_combinations_memo(1)
```

```
1
```

```
climb_combinations_memo(2)
```

```
2
```

```
climb_combinations_memo(7)
```

```
21
```

```
climb_combinations_memo(42)
```

```
433494437
```

Question 3

The *Catalan* numbers are a series of natural numbers that appear in diverse problems in combinatorics. Here is the recursive formula that defines Catalan numbers:

$$C_0 = 1$$

$$C_{n+1} = \sum_{i=0}^{n} C_i \, C_{n-i}$$

Here are the first 8 Catalan numbers (note that the numbering starts at 0, thus the last element is C_7):

1,1,2,5,14,42,132,429

Implement the **recursive** function *catalan_rec(n,mem=None)* which receives an integer *n* >= 0 and returns the *n*-th Catalan number. This function will use **memoization** in order to avoid repeated computations of the same value, and thus to improve the running time of the function. Therefore, the function receives an additional argument: *memo*.

- The function should make use of the **recursive formula given above**, and will not use any other formula for computing Catalan numbers.
- Indication: note that when computing Catalan numbers, we often make use of the computation of smaller Catalan numbers. These smaller numbers need to be saved in the dictionary, as you saw in the course and the recitation.
- You can assume that *n >=0* and is an integer.
- You may read more about Catalan numbers here.

Examples of execution:

```
catalan_rec(0)
```

```
1
```

```
catalan_rec(1)
```

```
1
```

```
catalan_rec(2)
```

```
2
```

```
catalan_rec(3)
```

```
5
```

```
catalan_rec(4)
```

```
14
```

```
catalan_rec(42)
```

```
39044429911904443959240
```

Question 4

The owners of the local grocery store are interested in how many different ways they can change a sum of money *n* using a list of coins *lst*.

For example, *n*=5 can be changed in 4 different ways using the list of coins lst=[1,2,5,6] :

[1,1,1,1,1], [1,2,2], [1,1,1,2], [5]

If we rather wanted to change *n*=4 using the same coins, there would be 3 possibilities:

[1,1,1,1], [2,2], [1,1,2]

Clarifications:

- Note that, in the change, the order of the coins is not significant, only the number of occurrences of each coin is significant. For example [1,2,2] is the same as [2,1,2] and [2,2,1].
- You can assume that the owners of the grocery store have an infinite supply of each type of coin in the list *lst*.
- You can assume that the list *lst* contains only numbers greater than 0 and that all the numbers in the list are different.

Part A

Implement the function *find_num_changes_rec(n,lst)*, which receives the integer *n* and the list of coins *lst* , and returns the number of ways one can change *n* using coins whose values are in the list *lst*.

- The implementation must be **recursive and without memoization.**
- Hint: as in the exercises done in class, we will look at the last element of the list of coins, and we will choose if we want to include it in the change or not (think carefully at the recursive step). For example, if the coins list contains the following coins: [1,2,5], we will choose if we want to use the coin 5, or if we choose not to use it. If we choose not to use it, we will not use this coin in the following calls. Note that every coin can be used more than once, so you need to make sure that your recursive solution allows this.
- When n=0, the function should return 1, because changing n=0 using no coins at all is still considered as one way to change it. Also, think at what happens in the following cases: when *lst*=[] and when *n<0* (look carefully at the execution examples below).

Part B

Implement the function *find_num_changes_mem(n,lst)* which receives the integer *n* and the list of coins *lst* , and returns the number of ways one can change *n* using coins whose values are in the list *lst*. This function will use **memoization** in order to avoid repeated computations of the same value, and thus to improve the running time of the function. Therefore, the function receives an additional argument: *memo*.

- The implementation must be **recursive with memoization.**
- Hint: note that when saving values in the dictionary you need to use a tuple as a key (think why).

Examples of execution:

```
find_num_changes_rec(0,[9,6,1,2,0.5])
```

```
1
```

```
find_num_changes_rec(5,[5,6,1,2])
```

```
4
```

```
find_num_changes_rec(-1,[1,2,5,6])
```

```
0
```

```
find_num_changes_rec(1,[2,5,6])
```

```
0
```

```
find_num_changes_rec(4,[1,2,5,6])
```

```
3
```

```
find_num_changes_rec(0.9,[0.8,2,5,6])
```

```
0
```

```
find_num_changes_rec(2,[0.5,1])
```

```
3
```

```
find_num_changes_mem(5,[1,2,5,6])
```

```
4
```

```
find_num_changes_mem(-1,[1,2,5,6])
```

```
0
```

```
find_num_changes_mem(5,[1,2,5,6])
```

```
4
```

```
find_num_changes_mem(1,[2,5,6])
```

```
0
```

```
find_num_changes_mem(4,[1,2,5,6])
```

```
3
```

```
find_num_changes_mem(0.9,[0.8,2,5,6])
```

```
0
```

```
find_num_changes_mem(2,[0.5,1])
```

```
3
```

```
find_num_changes_mem(1430,[1,2,5,6,13]) # Cannot be done in a reasonable time without memoization
```

```
231919276
```

Complement (no need to submit): compare the running time of both functions for different values of *n* and different lists *lst* (the example below uses *n*=[143] and *lst*=[1,2,5,6,13]), using the following code snippet:

```
from timeit import default_timer as timer
lst = [1,2,5,6,13]
n = 143
start = timer()
find_num_changes_rec(n,lst)
end =  timer()
print('Time without memoization for ',n,lst,':',end - start)
start = timer()
find_num_changes_mem(n,lst)
end = timer()
print('Time with memoization for ',n,lst,':',end - start)
```