Exercise Lists, Functions, Strings

August 12, 2019

1 Fibonacci Problem

Sometimes Recursion comes with a price to pay and this is the time of computation, due to the times it has to call itself (it can grow exponentially with the number of recursions it does. a perfect example of this is Fibonacci sequence. the n-th element can be obtained by calling its previous

$$F(n) = F(n-1) + F(n-2),$$

with $F(0) = 0, F(1) = 1$ which means that we can obtain the sequences by calling a function of the previous elements.

Your task is to first implement and see how recursion can make the code grow exponentially in function of n. To do this:

1. make a function that is going to be called `fib_recursion` which is gonna take n as a parameter and returns the n-th term of the Fibonacci sequence. Test your code with $n < 30$. your code should look like this:

   ```
   def fib_recursion(n):
       #Your answer here
       return n_fibonacci
   ```

2. In order to observe the performance of this routine make a loop in which n varies from 1 to 40 and register the time each iteration takes to compute. to do this you'll have to use a library called `time`, and it can be called as follows:

   ```
   import time
   for i in range(n):
       init=time.time()
       #your process
       difference_time=time.time()-init
   ```

   So keep the times of each iteration and the number of the iteration in different lists (to do this use the append method) and then plot the time taken in function of the number of the iteration. To plot use the next command

   ```
   import matplotlib.pylab as plt
   plt.plot(list_of_n,list_of_times)
   plt.show()
   ```

3. That was the warm up, now we are going to implement an optimization of the last function. to do this you can see that the Fibonacci sequence can be done without calling many times itself, now implement a function that takes the a number n and returns the n-th term of the Fibonacci sequence. To do this see that you can start with $a = 1, b = 0$ and then create the new term by doing

```
a <-- a+b
b <-- a
```

if you get confused check this Help to guide you in the process.

4. Once you implemented your code test it again with little numbers and repeat the same part of the 3-th numeral and measure the time it take to compute the first 100 Fibonacci numbers and plot it. What is the main difference you notice in both plots. what can you conclude about the first method implemented?. Give a reason why it is so different and why they can be that different if they are both doing the same thing.

5. In order to check if your code is working properly compute F_{n+1}/F_n for every n and plot this ratio in function of n. Where does it converge (if it does)?

1.1 Bonus

Implement the last method but this time the function has to be such that it can receive negative values of n

$F(-1) = 1$, $F(-2) = -1$, $F(-3) = 2$, $F(-4) = -3$, $F(-5) = 5, F(-6) = -8$, $F(-7) = 13\, F(-8) = -21$

2 Euler's method for a first-order ODE

We want to calculate the shape of an unknown curve which starts at a given point with a given slope. This curve satisfies an ordinary differential equation (ODE):

$$\frac{dy}{dx} = f(x, y(x)) \text{ with } y(x_0) = y_0$$

The starting point A_0 (x_0, y_0) is known as well as the slope to the curve at A_0 and then the tangent line at A_0 .

Take a small step along that tangent line up to a point A_1. Along this small step, the slope does not change too much, so A_1 will be close to the curve. If we suppose that A_1 is close enough to the curve, the same reasoning as for the point A_1 above can be used for other points. After several steps, a polygonal curve A_0, A_1, .. , An is computed. The error between the 2 curves can be small if the step is small.

We define points A_0, A_1, A_2, ... , An whose x-coordinates are x_0, x_1 , ... , x_n and y-coordinates are such that $y_{k+1} = y_k + f(x_k, y_k) * h$ where h is the common step. If T is the length $x_n - x_0$ we have $h = T/n$.

We will take as an example the differential equation:

$$\frac{dy}{dt} = 2 - exp(-4t) - 2y \text{ with } x_0 = 0, y_0 = 1, T = 1.$$

We know that an exact solution is $y = 1 + 0.5exp(-4t) - 0.5exp(-2t)$. For each x_k we are able to calculate the y_k as well as the values z_k of the exact solution.

1. Our task is, for a given number n of steps, to return the mean (truncated to 6 decimals) of the relative errors between the y_k of the $n+1$ points A_k of our polygonal line and the $n+1$ z_k of our exact solution. For that we can use:

   error in $A_k = abs(y_k - z_k)/z_k$ and then the mean is sum(errors in A_k)$/(n+1)$.
   Examples:

```
ex_euler(10) should return: 0.026314 (from 0.026314433214799246)
with
Y = [1.0,0.9..,0.85..,0.83..,0.83..,0.85..,0.86..,0.88..,0.90..,0.91..,0.93..]
Z = [1.0,0.9..,0.88..,0.87..,0.87..,0.88..,0.89..,0.90..,0.91..,0.93..,0.94..]
Relative errors = [0.0,0.02.,0.04..,0.04..,0.04..,0.03..,0.03..,0.02..,0.01..,0.01..,0.01..]

ex_euler(17) should return:0.015193 (from 0.015193336263370796)
```

   Check Reference for more information
   Also plot the solutions and the error in function of time. Comment about the main source of error, what could you do to reduce the error

2. Repeat the last part but this time reduce the size of the of the step h and plot again the solution the result and the error. What can you conclude from this?

3 Strings

Write a function that takes a string of braces, and determines if the order of the braces is valid. It should return true if the string is valid, and false if it's invalid.

All input strings will be nonempty, and will only consist of parentheses, brackets and curly braces: ().

What is considered Valid?. A string of braces is considered valid if all braces are matched with the correct brace.

Examples:

```
"()"                    =>  True
")(()))"                =>  False
"("                     =>  False
"(())((()())())"        =>  True
"[(]"                   =>  False
"{[[(({(((())}))))]]}"=>  True
```

3.0.1 Constraints

```
0 <= input.length <= 100
```

Along with opening (() and closing ()) parenthesis, input may contain any valid ASCII characters. Furthermore, the input string may be empty and/or not contain any parentheses at all. Do not treat other forms of brackets as parentheses (e.g. [], {}, <>).

4 List of lists

given a list of random integers create a list of lists such that each element is going to be a list times the number it has. the result must be organized in increasing order for example:

Given:

```
[1,3,2,8,6] ---> [[1],[2,2],[3,3,3],[6,6,6,6,6,6],[8,8,8,8,8,8,8,8]]
```

to do this first you must implement your own method to sort the list and then use the append method.

5 Sorting things in other way

in this task you're expected to sort an array of 32-bit integers in ascending order of the number of on bits they have.

E.g Given the array [7, 6, 15, 8]

```
7 has 3 on bits (000...0111)
6 has 2 on bits (000...0011)
15 has 4 on bits (000...1111)
8 has 1 on bit (000...1000)
```

So the array in sorted order would be [8, 6, 7, 15].

In cases where two numbers have the same number of bits, compare their real values instead.

E.g between 10 (...1010) and 12 (...1100), they both have the same number of on bits '2' but the integer 10 is less than 12 so it comes first in sorted order.

Your task is to write the function sortBybit() that takes an array of integers and sort them as described above.

```
sortByBit([3, 8, 3, 6, 5, 7, 9, 1]) // => [1, 8, 3, 3, 5, 6, 9, 7]
```

6 From mRNA to Protein

Given a protein string of length at most 1000 aa. (Check How a protein is written in terms of Aminoacids, your task is to return the total number of different RNA strings from which the protein could have been translated (Don't neglect the importance of the stop codon in protein translation!!).

Amino acid	3-letter	1-letter
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamine	Gln	Q
Glutamic acid	Glu	E

Amino acid	3-letter	1-letter
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V