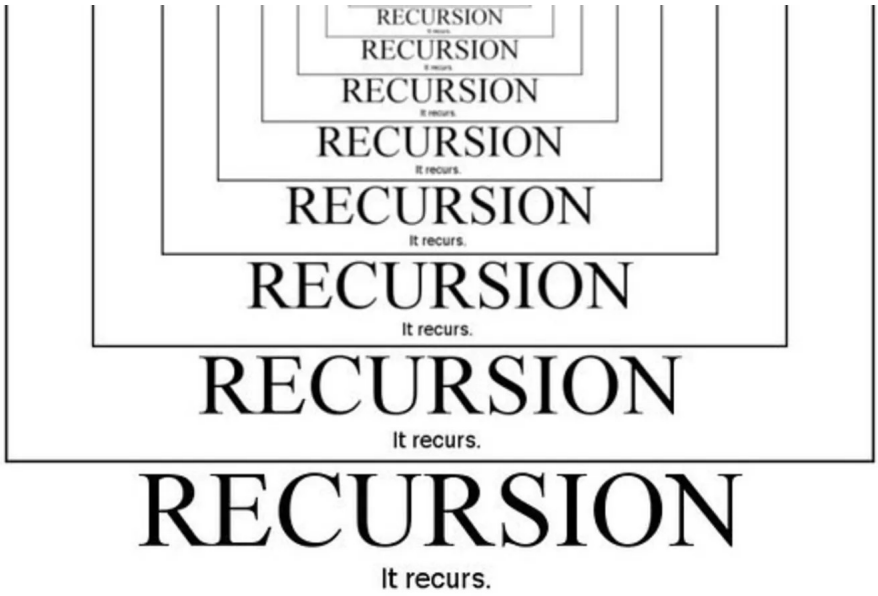
QUIZ 8

Due Date : 15.12.2023 - Friday (23:00)
Advisor : R.A. Görkem AKYILDIZ
Programming Language : Python 3.6.8



1 Introduction

In this quiz, you are expected to gain some basic experience on recursion mechanism. So far you programmed in non-recursive way for this course as you did not need to use recursive algorithms to solve problems. Recursion is a way that enables you to divide a problem and solve it easier as compared to original problem. So, the main aim of recursion is dividing the problem into sub-problems that are similar to original problem but requires less effort to solve. You may think recursion as, say that you do not know how many people are in front of you at a queue and the only thing that you can see is the person behind you and in front of you, and you want to learn how many people there are in front of you to be able to calculate the approximate time you will be waiting in the line. It is obvious that you cannot count the people in front of you by yourself as you cannot see anything other than the two people that mentioned before. So, it may help you to ask them about how many people there are in front of them, if they reply you, you can simply calculate how many people there are in front of you by adding one or subtracting one (according to who answered your question), but you want to select the person that you will ask your question wisely, if you ask this question to the person behind you, the problem becomes greater as he/she also does not have any information about how many people there are in front of him/her, nor the people behind him/her do not have any information about that as they cannot see beginning of the line neither. On the other hand, if you ask this question to the person in front of you, he/she may ask it to the person in front of him/her, and so on. When the question is asked to the person that is at the beginning of the line, he/she can simply answer as zero. So, as the person behind him/her got this answer, he/she can easily think that, if there is one person in front of me, and there is nobody in

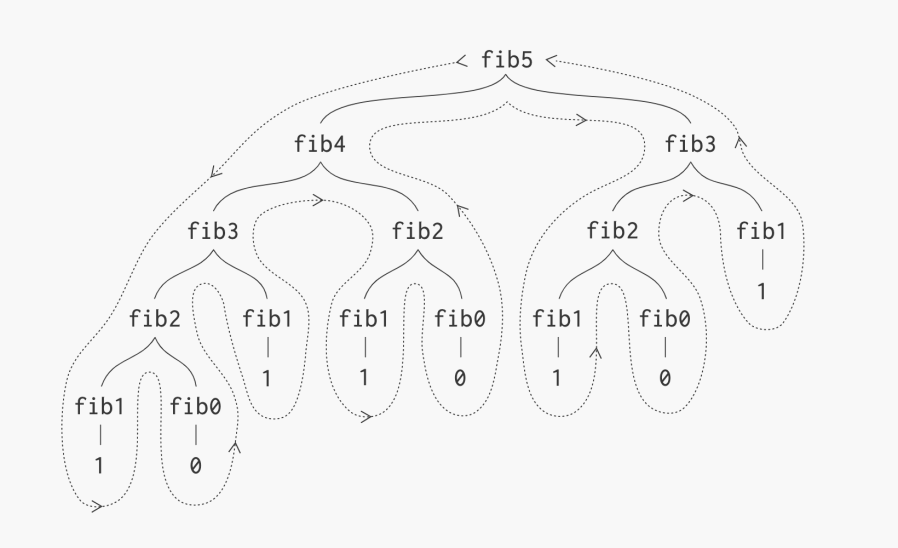
front of him/her, it is clear that there is one person in front of me, and the reply gets directed towards you by adding one at each step. So, when the person in front of you replies you as there are x people in front of me, you can simply say that there are x+1 people in front of me. As it can be clearly seen, when the problem is divided into smaller sub-problem(s) you can solve the case that is impossible to solve when it is as a whole. **Note that, recursion is not a thing that you can apply to everything, it only works if and only if the problem can be divided into sub-problem(s) that is/are similar to original problem with having a relatively easier solution as compared to original one.**

2 Implementation

Fibonacci Series is a problem that is recursive inherently. Its calculation formula is adding up last two elements before current element (first two elements are assumed as 1), and ratio of successive elements approach to the golden ratio (≈ 1.618).

In Computer Science domain, Fibonacci Series is one of the basics for understanding recursion. So, for example, if someone wants to calculate fourth Fibonacci number, he/she must know/or calculate the second and third Fibonacci number beforehand. The second Fibonacci number is one by definition, and the third Fibonacci number is summation of the first and second Fibonacci number, which are one for both by definition, so, as fourth Fibonacci number is summation of the second and the third, it is two plus one, which is three.

There are several ways to calculate Fibonacci Series, some of them are naive, some of them are eager. For this quiz, you are expected to implement one naive and one eager recursive algorithm to calculate Fibonacci Series.



For the naive approach, you must use memoryless approach. The only thing your program will keep in the memory is value of the first two Fibonacci numbers are both one. Then when it is asked to calculate a Fibonacci number, it will calculate whole tree element by element from top to down as in shown at the figure with arrows.

For the eager approach, everything seems to be same with the naive one but with a slight and very important difference, which is memorizing. Your program will memorize its previous calculations. So, for example, as you can see at the figure, third Fibonacci number is

calculated twice, once for calculating fourth Fibonacci number and then for the fifth Fibonacci number. If the program have had memorized third Fibonacci number once, it would save it from calculating 4 steps to get the third Fibonacci number again on the right side of the graph, which means faster computing with a small sacrifice from the memory. Moreover, if that program is prompted to calculate Fibonacci numbers again and again, as it memorizes its previous calculations, it can answer immediately. You may try to calculate fiftieth Fibonacci number to see the difference at the runtimes to understand the differences between naive and eager approach.

3 Definition of Input

Each line contains the Fibonacci number that is wanted to calculate. Each line can be assumed as a different calculation, but keep in mind that your eager approach must store the previous calculations not only for one calculation but also for all of the calculations that are requested at that runtime.

You may use only one global variable which is a reference to a list that contains Fibonacci numbers calculated so far for the eager approach for the sake of simplicity. Note that it must only be used for holding Fibonacci numbers calculated so far, nothing different than it, abusing it may result with getting a grade that is as low as zero.

4 Definition of Output

Each solution will be separated from each other with dash (-) character printed by exactly 32 times.

This explanation is just for one of the solutions, but the structure is the same for all of the solutions:

- First line will contain "Calculating <N>. Fibonacci number:" where <N> is the number that is wanted to calculate.
- Intermediate lines will be in either in "fib(n) = fib(n-1) + fib(n-2)" or "fib(n) = <RESULT>" structure according to the step. For example, for naive approach, as the only thing that program knows is first two Fibonacci numbers are one, it will show the second structure just for the cases that n equals to one or two, and first structure for the rest. On the other hand, for the eager approach, your program will print the first structure if the number that is being calculated at that step is not calculated beforehand at that runtime, it will use the second structure if it has been already calculated at that runtime. **Note that, if you are prompted to calculate a non-positive Fibonacci number, your program will give an error that is the same as the one in the Sample I/O files.**
- Last line will contain "<N>. Fibonacci number is: <RESULT>" where <N> is same as in the first line, and <RESULT> is the result of your calculation.

Note that you must print the list structure that you held for eager approach of yours at the end of the output for the eager approach that is the same as the one in the Sample I/O files.

5 Restrictions

- Your code must be able to execute on our department's developer server (dev.cs.hacettepe.edu.tr).
- You must obey given submit hierarchy and get score (1 point) from the submit system.
- **You must benefit from recursion.**
- Your code must be clean, do not forget that main method is just a driver method that means it is just for making your code fragments run, not for using them as a main container, create functions in necessary situations but use them as required.
- You must use comments for this project and you must give brief information about the challenging parts of your code. Do not over comment as it is against clean code approach. Design your comments so that they make your code fully understandable and not excessive for others.
- You can benefit from Internet sources for inspiration but do not use any code that does not belong to you.
- You can discuss high-level (design) problems with your friends but do not share any code or implementation with anybody.
- Do not miss the submission deadline.
- Source code readability is a great of importance. Thus, write READABLE SOURCE CODE, comments, and clear MAIN function. This expectation will be graded as "clean code".
- Use UNDERSTANDABLE names to your variables, classes, and functions regardless of the length. The names of classes, attributes and methods should obey Python naming convention. This expectation will be graded as "coding standards".
- You can ask your questions through course's Piazza group, and you are supposed to be aware of everything discussed in the Piazza group. General discussion of the problem is allowed, but **DO NOT SHARE** answers, algorithms, source codes and reports.
- All quizzes must be original, individual work. Duplicate or very similar quizzes are both going to be considered as cheating.

6 Execution and Test

Your code must be executed under **Python 3.6.8** at **dev.cs.hacettepe.edu.tr**. If your code does not run at department's developer server during the testing stage, then you will be graded as 0 for code part even if it works on your own machine.
Sample run command is as follows:

- python3 fibonacci.py input.txt output_naive.txt output_eager.txt

7 Grading

Task	Point
Naive Solution	50
Eager Solution	50
Total	100

Note that you must score one at the submit system, otherwise 20% of your grade will be deducted, moreover, you must implement a main function otherwise 10% of your grade will be deducted! There may also be other point deductions if you do not obey the given rules, such as if you do not use recursion as necessary.

8 Submit Format

File hierarchy must be zipped before submitted (Not .rar, only not compressed .zip files because the system just supports .zip files).

```
- b<StudentID>.zip
    - fibonacci.py
```