

CS 120 (Spring 21): Introduction to Computer Programming II

## Short Project #7 - Annoying Recursion (Short)

due at 5pm, Thu 3 Mar 2022

Go ahead and mock.  
Yes, this project is silly.  
But you will learn much.

### 1 Overview

In this project, you will be practicing recursion. You will turn in two files. In `annoying_recursion_short.py`, you will write several recursive functions - but you're going to write them in a **very strange form** - the “annoying” version! I'll describe the functions, and tell you exactly what I mean by “annoying,” later in this spec.

In the second file, `linked_list_recursion_short.py`, you will write one pair of functions, which performs an operation on linked lists: they will both do the same thing, but one version will use loops and the other will use recursion. (But you'll be happy to hear that the recursive one does **not** have to follow the Annoying Requirements!)

#### 1.1 Loops Banned

In every function in this project (except for `is_sorted()`, where loops are required) loops are banned! (You are allowed to use string/list multiplication, however.)

#### 1.2 Helper Functions Banned

In every function in this project - whether “annoying” or not - helper functions are banned. Every one of these functions can be completed with only a single, recursive function. (And you have to use exactly the parameters I require, or you won't pass the testcases.)

Some of you may know about default arguments in Python (if not, that's OK) - these are banned as well! Default arguments are a cool feature - but they are basically just a way to write a helper function, so they aren't allowed in this project, either.

### 2 Make Recursion Annoying Again

What's weird about the “annoying” functions? What's annoying is that you are going to have to implement **lots** of different cases for each one - and the various cases will have **very specific** limitations.

For simplicity, every one of these functions will take exactly one integer argument; they will either print things to the screen, or return something. Every one can be called with any non-negative integer (that is, zero or anything positive).

And just to make clear that these are “annoying” functions (that is, you would **never** write them like this in the Real World!) the name of every one begins with `annoying_`.

## 2.1 4 Base Cases

Every “annoying” function must have special cases for the values 0,1,2,3. For each of these values, you must **implement it as a base case** - that is, simply return (or print) the answer - without any recursion at all.

## 2.2 3 Hard-Coded, but Recursive Cases

Every “annoying” function must also have special cases for the values 4,5,6. These must be implemented recursively, but you **must** hard-code what you recursively call. That is, don’t use the value  $n$  in those cases at all (except to figure out that you are in that case). So, for instance, the factorial function might include this snippet:

```
if n == 5:
    return 5 * annoying_factorial(4)
```

## 2.3 The General-Purpose Case

Finally, every “annoying” function must have a general case. (This is what would be the body of the recursive in a more typical solution.) This general case:

- Must not execute except for  $n \geq 7$ . (But it must work properly for all such large values!)
- Must recurse
- Must not loop

## 3 `annoying_factorial(n)`

In `annoying_recursion.py`, write the function `annoying_factorial(n)`, which takes a single integer parameter (which must be non-negative). It must return  $n!$  - that is,

$$1 \cdot 2 \cdot 3 \cdot 4 \dots \cdot n$$

It must obey the Annoying Requirements.

**REMEMBER:**  $0!$  (that is, “zero, factorial”) is equal to 1.

## 4 `annoying_fibonacci(n)`

In `annoying_recursion.py`, write the function `annoying_fibonacci(n)`, which takes a single integer parameter (which must be non-negative). It must return the Fibonacci Number ([https://en.wikipedia.org/wiki/Fibonacci\\_number](https://en.wikipedia.org/wiki/Fibonacci_number)).

The first two Fibonacci Numbers are fixed:

$$F_0 = 0$$

$$F_1 = 1$$

and from then on, all Fibonacci Numbers are defined recursively:

$$F_n = F_{n-1} + F_{n-2}$$

However, your function must obey the Annoying Requirements. This means that you must **hardcode** the answer for 0,1,2,3. For 4,5,6 you must recurse, but you must **hardcode** the arguments that you pass to the recursive calls. Of course, you will use the general formula, as well - but **only** on the last case!

## 5 `annoying_very_quizzical(n)`, `annoying_parens(n)`

These two functions each return a string. The first function returns a string that looks like this:

`what?!?!?!?`

where `n` is the number of question marks and exclamation marks at the end. Thus, if `n=0`, return `"what"`, if `n=1`, return `"what?"`, and if `n=6`, return `"what?!?!?!?"`.

The second function returns the variable `x`, surrounded by deeply nested parentheses. But to make things more readable, we will add a space, on both ends, after each pair of parens. For example, if `n=5`, you will return

`( ( ( ( (x) ) ) ) )`

Of course, if `n=0`, return `"x"`. **NOTE:** If the number of parens is even, do **not** put spaces on the outside. Thus, if `n=4`, return `"(( ( (x) ) ))"`.

Both functions must obey the Annoying Requirements.

## 6 `annoying_countdown(n)`

This function must print a countdown to the screen, counting from `n` down to 0, one per line.

It must follow the Annoying Requirements.

## 7 `is_rev_sorted(head), is_rev_sorted_recursive(head)`

In `linked_list_recursion_short.py` , you will write two functions, both of which do exactly the same thing: but one must be recursive, and the other must **not** be. (The recursive problem is **not** required to follow the Annoying Requirements, though.)

Each of these functions take a linked list (which might be empty) and checks to see if the values are sorted **in reverse order**; if so, it returns `True`; if not, it returns `False`.

If the list is empty or has a single node, they must return `True`.

The list may contain duplicate elements; this still counts as being in order, so long as they are next to each other.

## 8 Turning in Your Solution

You must turn in your code using GradeScope.