CSCI 1133, Spring 2023
Homework 11
Due: 11:55 pm, Tuesday April 11th, 2023

Due Date: Submit your solutions to Gradescope by 11:55 pm, Tuesday, April 11th. Do not upload anything to Canvas and PLEASE be sure to use proper naming conventions for the file, classes, and functions. We will NOT change anything to run it using our scripts.

Late Policy: Submissions will be accepted up to 24 hours late (11:55 pm on Wednesday) for a 1 point penalty, up to 48 hours late (11:55 pm on Thursday) for a 2 point penalty, or up to 72 hours late (11:55 pm on Friday) for a 3 point penalty. Submissions past 11:55 pm on Friday will not be accepted.

Unlike the computer lab exercises, this is not a collaborative assignment. See the syllabus and read section "Academic Dishonesty" for information concerning cheating. Always feel free to ask the instructor or the TAs if you are unsure of something. They will be more than glad to answer any questions that you have. We want you to be successful and learn so give us the chance to help you.

Purpose
The purpose of this homework is to gain practice writing recursive functions and using recursion to solve problems.

Instructions
This assignment consists of 3 problems, worth a total of 30 points. If you have questions, ask!

Because your homework file is submitted and tested electronically, the following are very important:

- Submit the correct file, hw11.py, through Gradescope by the due date.
- Your program should run without errors when we execute it using Python 3.

Submit your work to Gradescope under the Homework 11 assignment. Make sure that your file is called hw11.py: if you don't match the file name exactly then the testing script won't work and you'll lose points.

The following will result in a score reduction equal to a percentage of the total possible points:

- Constraints not followed (up to 40%)
- Bad coding style (missing documentation, meaningless variable names, etc.) (10%)

Some of the points in this assignment are given based on the Gradescope autograder, so doing something that breaks the gradescope tests (naming your file incorrectly, naming your function incorrectly, having syntax errors, or using the input() function somewhere the autograder doesn't expect) are likely to lose you all of those points.

Documentation:
Use the following template for EVERY function that you write. The docstring below should be placed inside of the function, just after the function signature.

```
'''
Purpose:
Parameter(s):
Return Value:
'''
```

A note on Gradescope tests using Recursion:
The gradescope tests will not check whether you actually used recursion for the problems below, so you will get full points from the automatic test cases if you just use a loop. However, you will lose a very large portion of the manually graded points if you do this on problems A or B, since you will be violating the constraints of the problem.

Download the **hw11.zip** folder from Canvas: you'll need it for Problem C. It contains a template hw11.py file that has some of the code you'll need for Problem C already written for you.

Problem A. (10 *points*) **Collatz conjecture sum**

The Collatz conjecture (https://en.wikipedia.org/wiki/Collatz_conjecture) is an unproven mathematical rule that says the following:

Take any positive integer *n*. If *n* is even, divide it by 2 to get *n / 2* (use integer division so that you don't end up with floating point numbers). If *n* is odd, multiply it by 3 and add 1 to obtain *3n + 1*. Repeat the process indefinitely, and eventually you will reach 1.

Write a **recursive** function called `collatz(n)` that takes in a single positive integer argument `n` and returns the *sum* of numbers in the collatz sequence from `n` to 1, inclusive. For example, suppose your initial number was `5`. Then the collatz sequence would be the following:

The initial number is **5**.
5 is odd, so the next number is 5*3 + 1 = **16**
16 is even, so the next number is 16//2 = **8**
8 is even, so the next number is 8//2 = **4**
4 is even, so the next number is 4//2 = **2**
2 is even, so the next number is 2//2 = **1**
We have reached 1, so the sequence is complete, and the function would return `36` `(=5+16+8+4+2+1)`

Hints:

- You may assume that your function will only be called with positive integers.
- This function returns an int, so both your base case and your recursive case(s) need to return an int

Constraints:

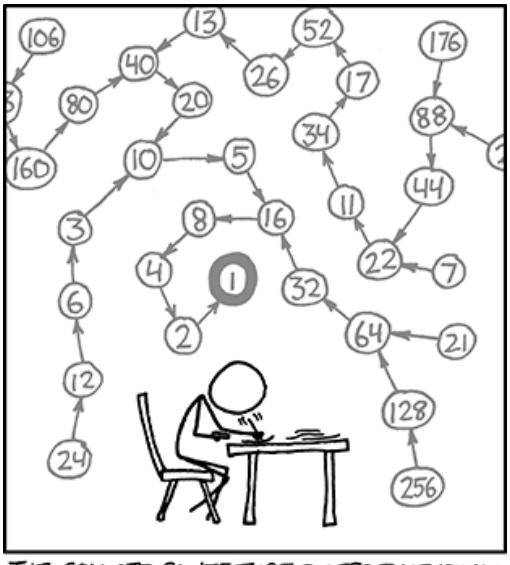
- The `collatz` function must be implemented using recursion. Do not use any loop constructs (`while` or `for`).

Examples:

```
>>> collatz(5)
36

>>> collatz(1)
1

>>> collatz(123)
6390
```



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

Problem B. (10 *points*) **Checking for Decoys**

You are a detective trying to stop a powerful criminal organization from stealing some of the world's treasures and monuments. How your enemies are capable of stealing entire buildings and even larger objects is not fully understood, but you have acquired documents containing information about their next potential targets.

First, though, you have learned that some of the documents are decoys. If at least one line in the file contains exactly two 'e's (lowercase only), then the document represents a current target for the organization - otherwise it's a decoy and should be ignored.

Write a **recursive** function called `two_es(lines)`, which takes in `lines`, a list of strings representing each line of a document, and returns the boolean `True` if that file is a target (that is, if at least one of the strings in the list has exactly two lowercase 'e's), or `False` if it's a decoy.

Hints:

- You don't have to do any File I/O for this problem, that's handled in Part C.
- You only need to use recursion to go through the list - you're free to use the .count method to determine how many 'e's are present in a given string.

Constraints:

- The `two_es` function must be implemented using recursion. Do not use any loop constructs (`while` or `for`).

Examples:

```
>>> two_es(['One line\n', 'Two lines\n', 'Three lines\n'])
True

>>> two_es(['here is a line\n', 'Another linE, starting with A\n',
'One MorE linE'])
False

>>> two_es([])
False

>>> two_es(['More examples\n', 'Here there are two lines that work\n',
'This is one of them\n', 'This is not\n', 'Excellent'])
True
```

Problem C. (10 *points*) **Searching All Files**

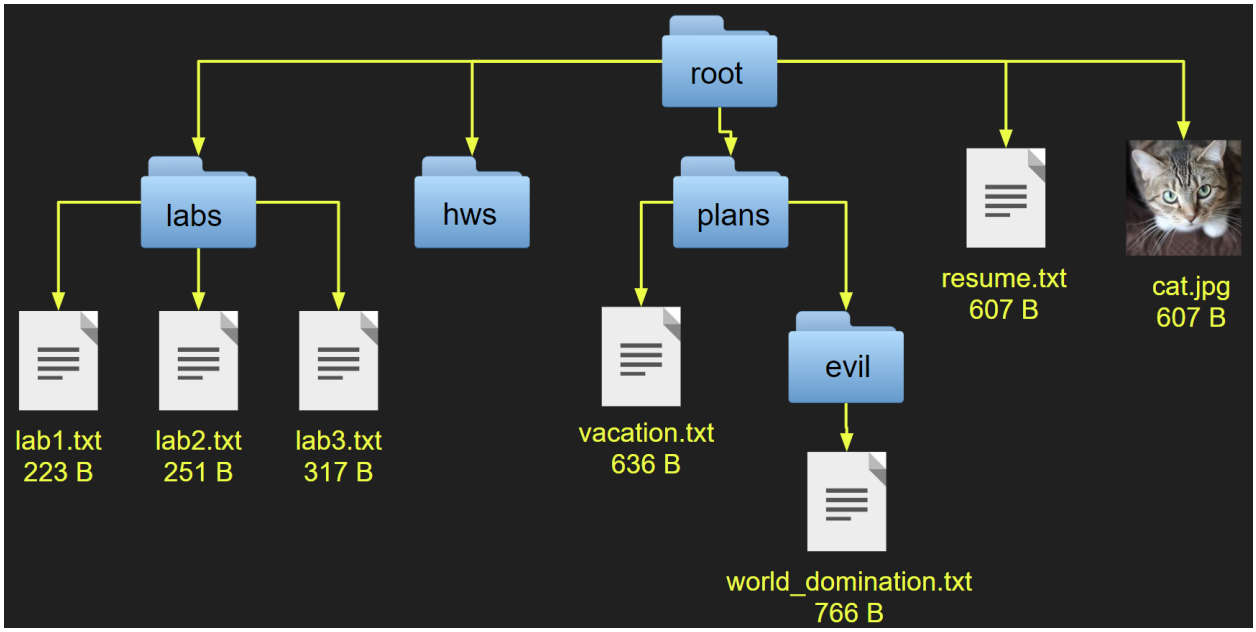
Continuing from the previous question, you must now search through all of the documents and generate a list of the ones that represent targets of the evil organization. The problem is that the documents are not all in one directory: they're in a nested directory structure, and could be in the top-level directory, or in subdirectories of that directory, or in subdirectories of those subdirectories, and so on.

The following code, given to you in the hw11.py template, uses recursion to print out all .txt files in a nested directory structure when given the file path to the top-level directory:

```
import os
def get_targets(path):
    for file in os.listdir(path):
        if os.path.isfile(path+'/'+file):
            if file.endswith('.txt'):
                print(path+'/'+file)  #.txt file, print out the path
        else:
            get_targets(path+'/'+file)  #Go into a subdirectory
```

`os.listdir` generates a list of all of the names of all of the files and directories within some directory that we specify the file path to, and `os.path.isfile` takes in the path to a file and returns `True` if it's a file, or `False` if it's a directory. For example, if given the directory structure below, with hw11.py in the same directory as root, then `get_targets('root')` would print out:

```
root/labs/lab1.txt
root/labs/lab2.txt
root/labs/lab3.txt
root/plans/vacation.txt
root/plans/evil/world_domination.txt
root/resume.txt
root/cat.jpg
```



Try out the function on some of the sample directories that were in the hw11.zip folder, to ensure that you understand what it currently does. If you're not sure, ask a TA to explain it to you.

Then, alter the `get_targets` function above so that rather than printing the paths to all of the files in the nested directory, it **returns** a list of the paths to all of the files that are targets (but not the ones that are decoys, as defined in problem B). The order of the list does not matter.

Hints:

- This problem uses both loops AND recursion: the loop is used to iterate through the files within each directory, while the recursive call is used to go deeper into one of the subdirectories.
- You will need to open each file and read its contents to determine whether it's a target.
- Use `.readlines()` to get a list of strings representing the lines in each file, to pass into your `two_es` function.
- You will need to write documentation for get_targets, even though we provide you with some of the code.

Constraints:

- You must call your `two_es` function somewhere in `get_targets`.
- Don't import any modules other than `os`, and don't use any `os` module methods other than those already used in the given function.

Examples (assumes that the directory structure given in hw11.zip is intact; remember that the order does not matter for the output list)

```
>>> get_targets('docs1')
['docs1/things.txt']

>>> get_targets('docs2')
['docs2/Africa/pyramids.txt', 'docs2/antarctica.txt']

>>> get_targets('docs3')
['docs3/Asia/India/taj_mahal.txt',
'docs3/North_America/Canada/Alberta/west_edmonton_mall.txt',
'docs3/North_America/Mexico/chichen_itza.txt',
'docs3/North_America/panama_canal.txt',
'docs3/North_America/United_States/Minnesota/Minneapolis/keller_hall.txt']
```