CMPSC-132: Programming and Computation II
Spring 2020

Lab #3

Due Date: 02/21/2020, 11:59PM

Read the instructions carefully before starting the assignment. Make sure your code follows the stated guidelines to ensure full credit for your work.

Instructions:

- The work in this lab must be completed alone and must be your own.
- **Download the starter code file from the LAB3 Assignment on Canvas. Do not change the function names on your script.**
- A doctest is provided as an example of code functionality. Getting the same result as the doctest does not guarantee full credit. You are responsible for debugging and testing your code with enough data, you can share ideas and testing code during your recitation class. As a reminder, Gradescope should not be used to debug and test code!
- Each function must return the output (Do not use print in your final submission, otherwise your submissions will receive a -1 point deduction)
- **Do not include test code outside any function in the upload. Printing unwanted or ill-formatted data to output will cause the test cases to fail. Remove all your testing code before uploading your file (You can also remove the doctest). Do not include the input() function in your submission.**
- **For this assignment, you can assume that the parameters will follow the specifications for each function**

Goal

[2.5 pts] Write the **recursive** function *thirtyTwos*(*n*) that takes an integer greater or equal to 0 and returns an integer that represents the number of times that a 2 directly follows a 3 in the digits of *n*. *Hint*: The % and // operations from *sumDigits* could be helpful here

```
>>> thirtyTwos(132432601)
2
```

[2.5 pts] Write the **recursive** function *flat*(*aList*) that takes a possibly deep list and flattens it. The function should not mutate the original list. *Hint*: you can check if something is a list by using the built-in functions type() or isinstance()

```
>>> x = [3, [[5, 2]], 6, [4]]
>>> flat(x)
[3, 5, 2, 6, 4]
>>> x
[3, [[5, 2]], 6, [4]]
```

[2.5 pts] In the starter code, there is a function called *triangle* that calls the function *recursiveTriangle*(*n*,*n*) once. For this exercise you must write the **recursive** function *recursiveTriangle*(*x*, *n*) that **returns** a string with the LAST *x* lines of a right triangle of base and height *n*.

- *recursiveTriangle*(*x*, *n*) must be a recursive function, otherwise, no credit is given
- *recursiveTriangle*(*x*, *n*) must return the pattern as ' ***\n **\n *'. As shown in the example, you can use the print method during testing to check if your pattern is correct.
- **Don't modify anything in the triangle function**. If your recursive function is correct, calling *triangle*(*n*) should return the complete right triangle.

```
>>> triangle(4)
'****\n ***\n  **\n   *'
>>> print(triangle(4))
****
 ***
  **
   *

>>> recursiveTriangle(2,4)
'  **\n   *'
>>> print(recursiveTriangle(2,4))
  **
   *
```

This is the output that will be graded, make sure your function returns the value in this form

You can call the print method on your function to see the triangle pattern

```
>>> triangle(6)
'******\n *****\n  ****\n   ***\n    **\n     *'
>>> print(triangle(6))
******
 *****
  ****
   ***
    **
     *
>>> recursiveTriangle(3,6)
'   ***\n    **\n     *'

>>> print(recursiveTriangle(3,6))
   ***
    **
     *
```

Notes:

- The doctest includes \\n instead of \n in order to keep consistent leading whitespace required by the shell session, your code must return output with as \n, otherwise it will fail the test cases!

[2.5 pts] Write the **recursive** function *isPrime*(*num*) that takes an integer as a parameter and returns a boolean value, True if the number is prime, False otherwise. A prime number is a positive integer that has exactly two positive integer factors, 1 and itself.

- You can assume the function only receives integers
- If needed, the function could take a second argument, but it will not be provided by the user. This means it should be a preloaded value and the original function call will be fed only with *num*
- Remember to consider the special cases 0 and 1
- Based on your recursive algorithm, you might encounter the runtime error 'maximum recursion depth exceeded' when using large values of *num*. While changing the limit of recursive calls could solve the error, you should try to optimize your code rather than changing the recursion limit to accommodate an unoptimized algorithm. See the references at the end of this file for ideas on how to optimize your algorithm if needed.

NOTE: All functions should not contain any for or while loops, or global variables. Use recursion otherwise, no credit will be given

Deliverables:

- Submit your code in a file name LAB3.py to the Lab3 Gradescope assignment before the due date

References:

http://mathandmultimedia.com/2012/06/02/determining-primes-through-square-root/

https://www.smartickmethod.com/blog/math/operations-and-algebraic-thinking/divisibility/prime-numbers-sieve-eratosthenes/