Assignment 4 - Revising old problems with lists

* **Due** Dec 3 by 11:59pm
* **Points** 100
* **Submitting** a file upload
* **File Types** txt
* **Available** Nov 15 at 11:59pm - Dec 5 at 11:59pm 20 days

You will submit 2 python code files to Canvas. 1 file for each question. Be sure to rename them as .txt files before submission.

At the top of each of your source code files include this comment with your student name and student number filled in. Failure to include this statement can result in a penalty up to 100%.

**# I student\_name, student\_number, certify that this work is my own effort and that I have not allowed anyone else to copy from it.**

Late assignments will be penalized 25% for up to 1 day, 50% for up to 2 days, and 100% thereafter.

**Question 1 – Old Macdonald Had a List [/35 marks]**

You will create a program that generates the lyrics of the children’s song Old MacDonald had a farm. The user will supply the names and sounds for the animals and when they are finished you will print the lyrics to the song. The user should be able to enter as many pairs of names and sounds as they wish before the lyrics are shown.

Your program must define a module for the lyrics that accepts 2 parameters as input, as shown in the Modules unit video. You will collect user input in a loop and store the pair of answers in a list. Store these lists inside another list. Use some special input to indicate that the user is done entering data. Once the user has completed their data entry, output the lyrics one verse at a time by working through your list and calling the module.  Print a blank link between each verse.

Your output will resemble, but does not have to match, with the user’s reply printed in red:

Please enter an animal name (or -1 to quit): cow  
Please enter an animal sound: moo  
Please enter an animal name (or -1 to quit): dog  
Please enter an animal sound: woof  
Please enter an animal name (or -1 to quit): droid  
Please enter an animal sound: beep  
Please enter an animal name (or -1 to quit): -1  
   
Old MacDonald had a farm, E-I E-I O!  
And on that farm there was a cow, E-I E-I O!  
With a woof, woof here, and a woof, woof there,  
Here a woof, there a woof, everywhere a woof, woof.  
Old MacDonald had a farm, E-I E-I O!

(you would also print the next 2 verses below here)

 If done correctly your list will look something like [[‘cow’,’moo’],[‘dog’,’woof’],[‘droid’,’beep’]]. Solutions that do not make use of a nested list structure will be penalized.

Hint: when calling the lyric printing method, use at least 1 for loop to make your syntax easier.

Bonus marks may be awarded if the function parameter list is changed to accept a 2 element 1 dimension list that you use instead of 2 separate string parameters.

**Question 2 - Greek Primes [/35 marks]**

We saw one method of creating a list of prime numbers by checking if the number in question could be divided by any number bigger than 1, but less than the number itself. If it could, then we printed it out and moved on.  This method of prime number creation means that we divide with numbers that are themselves not prime.  For example, when we checked if 7 is prime, we divided it by 2, then 3, then 4, then 5, then 6.  We will have already seen that 4 and 6 are not prime though.

Another algorithm exists called the sieve of Eratosthenes, that comes from ancient Greece. It is modestly quicker than our standard algorithm\*

In this algorithm you create a list of numbers from 1 up to your goal. Starting at 2 eliminate its multiples in the list and move to the next number that is still in the list. When you reach the end of the list, all the numbers must be prime.

For example, if I wished to check for all the primes below 10, my first list would be [1,2,3,4,5,6,7,8,9,10].  Starting at 2, I would cross off 4, 6, 8, and 10. The next remaining number is 3, I would cross off 9, remembering that 6 was already eliminated. The next number in the list is 5. No multiples of 5 are in the list since 10 was already eliminated. Finally we reach 7 and there are no multiples of 7 in the list, so you have reached the answer of [1,2,3,5,7].

Another way of looking at it, published on Wikipedia is shown [here (Links to an external site.)](https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes#/media/File:Sieve_of_Eratosthenes_animation.gif).

Create a program that asks the user for an integer greater than or equal to 100. Generate a list of primes up to and including this number. Print the list to the screen on a single line. You may assume that the user will enter valid numeric input in the correct range.​

Hint: this algorithm requires you to pay close attention to your array index ranges when building your loops.  You will not be going from the start to the end of both loops every time.  Use your planning techniques or try doing it it on paper a couple of times to get the hang of it.  If done correctly using functions like len() you will not need to do any special techniques to manage the list size potentially changing every iteration.

To help you check if your answer is correct, here is a list adapted version of our old algorithm that can return something for you to check against.

def makeprimes( upper ):  
    count = 2  
    primes = [1]  
      
    while count <= upper:  
        prime = True  
        for i in range(2, count):  
            if count % i == 0:  
                prime = False

| Assignment 4 | | |
| --- | --- | --- |
| **Criteria** | **Ratings** | **Pts** |
| This criterion is linked to a Learning OutcomeQ1: Module is defined that accepts appropriate parameters  Note: The basic version of this is given in the video in the Modules unit, therefore it is not worth many marks.  If the students correctly implement this with 2d lists award 5 bonus marks to this category (10/5) | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ1: User input is stored in a list/parallel lists | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ1: Special input is provided to halt user input | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ1: Loop through created list(s) and output lyrics | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ1: General output formatting/Space between verses | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ1: Initialization of outer list, 0 if 1D lists are used | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ1: Appropriate input statements, storage in list, and omitting of special value | |  |  | | --- | --- | | **10 pts**  **Full Marks** | **0 pts**  **No Marks** | | 10 pts |
| This criterion is linked to a Learning OutcomeQ1: Storage in lists if student used 2d lists | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ2: Input/data conversion/>100 in prompt | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ2: Printing sieve on 1 line by any method | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ2: Correct function call to create prime list  n/a for 2021 | |  |  | | --- | --- | | **0 pts**  **Full Marks** | **0 pts**  **No Marks** | | 0 pts |
| This criterion is linked to a Learning OutcomeQ2: Sieve fn def is correctly formed  n/a for 2021 | |  |  | | --- | --- | | **0 pts**  **Full Marks** | **0 pts**  **No Marks** | | 0 pts |
| This criterion is linked to a Learning OutcomeQ2: Initializing the sieve list/count to 1/dealing with 1 as prime by any means | |  |  | | --- | --- | | **10 pts**  **Full Marks** | **0 pts**  **No Marks** | | 10 pts |
| This criterion is linked to a Learning OutcomeQ2: Outer loop from 1 to len by any means | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ2 Inner loop range setup  - 5 for using slice, or any method to search only the unprocessed part of the list -5 for each of the range elements | |  |  | | --- | --- | | **15 pts**  **Full Marks** | **0 pts**  **No Marks** | | 15 pts |
| This criterion is linked to a Learning OutcomeQ2: Primeness check | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ2: Correct removal of non primes | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| This criterion is linked to a Learning OutcomeQ2: Correct iteration/returns | |  |  | | --- | --- | | **5 pts**  **Full Marks** | **0 pts**  **No Marks** | | 5 pts |
| Total Points: 100 | | |

                break  
        if prime:  
            primes.append(count)  
              
        count = count + 1  
          
    return primes

\* - When testing this it took about .44 seconds using the old method and about .30 using the new to generate primes up to 100.  When I generated primes up to 1,000,000 the difference was about 57.8 minutes vs 51.5 minutes. When I used Java with the old algorithm to generate primes up to 1,000,000 it took about 100 seconds. Something to motivate using Java next semester.