CSC 401 – Midterm Exam

Thursday, Oct 15, 6:00pm – 8:00pm

Instructions

- This exam is open book and open notes.
- No collaborations or communications allowed.
- Read the questions **carefully** and **completely** before answering.
- Use the **Midterm** submission folder to submit your solution.
- Submit your solution to all programming problems in one **single python file** using your name as file name.
- You can submit your file multiple times. However, only the **last submission** will be graded. No exceptions. Make sure your last submission is the final and correct version of your solution.
- Submission folder will close at 8:00pm. No late submission will be allowed.
- Good luck

Programming Problems

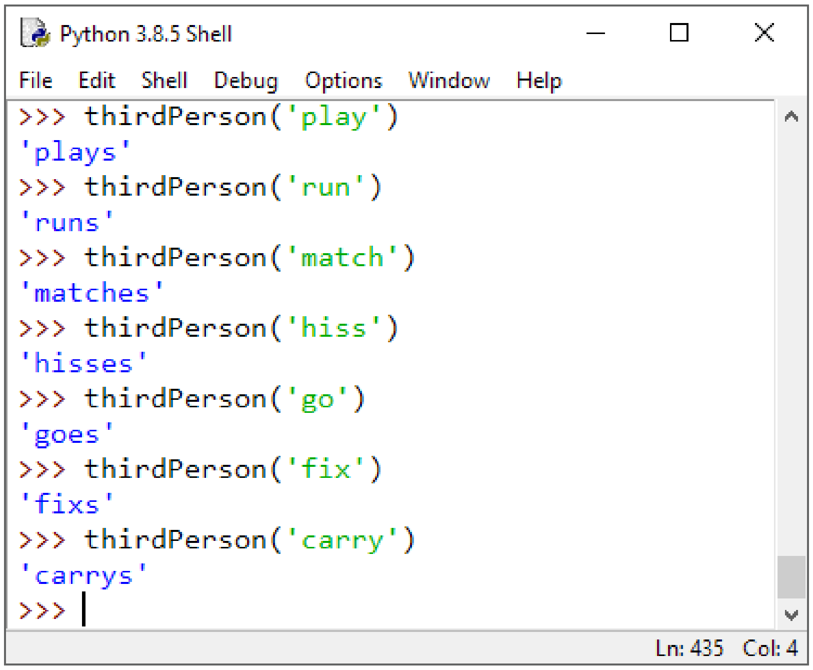
1. Verbs [20 points]

You want to write a function that automatically takes a verb (e.g. ‘play’, ‘run’, ‘match’, ‘go’), converts it into its third person singular form (the form of the verb you use with ‘he’, ‘she’, and ‘it’: ‘he plays’, ‘she runs’, ‘it matches’) and returns the result. The simple rule is: add ‘s’, for example, ‘play’ becomes ‘plays’. That rule doesn’t work for all words (e.g. ‘go’ becomes ‘goes’, not ‘gos’). Here is a slightly more sophisticated set of rules (that still don’t work for all words):

1. if the word ends in ‘o’ add ‘es’
2. if the word ends in ‘ch’, ‘ss’, or ‘sh’ add ‘es’
3. otherwise you add ‘s’

This works in many cases (though not for some words which require doubling of the last consonant, such as ‘stop’, and not for many words that form their past tense irregularly, such as ‘see’, we won’t worry about those).

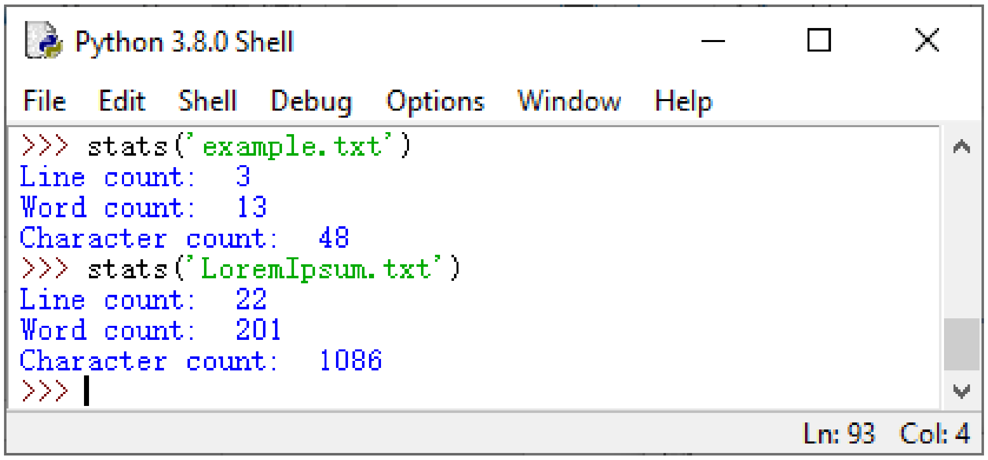
Write a function `thirdPerson(verb)` that takes a string as parameter representing a verb, and returns the verb's third person singular form using the 3 rules listed above.



```
>>> thirdPerson('play')
'plays'
>>> thirdPerson('run')
'runs'
>>> thirdPerson('match')
'matches'
>>> thirdPerson('hiss')
'hisses'
>>> thirdPerson('go')
'goes'
>>> thirdPerson('fix')
'fixs'
>>> thirdPerson('carry')
'carrys'
>>> 
```

2. File stats [30 points]

Write a function `stats(filename)` that takes one parameter which is a string corresponding to the name of a text file. The function should print to the screen the number of lines, words, and characters in the file. Your function can only open and read the contents of the file **once**, which means your function can only use one of `read()` or `readlines()`. The following shows what the function would print when called on some sample files provided in the zip file containing the lab template:



```
>>> stats('example.txt')
Line count:   3
Word count:   13
Character count:   48
>>> stats('LoremIpsum.txt')
Line count:   22
Word count:   201
Character count:   1086
>>> 
```

Hint:

- Remember to initialize your counts
- You can safely assume that comma and dot are the only punctuations in the text file.
- Use `strip()` for each line to remove leading and trailing white spaces(including newline).
- Use `replace(old, new)` to remove comma and dot from the text.
- Use `split()` to separate words.

3. Fibonacci Sequence [30 points]

The Fibonacci Sequence is the series of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

It starts with number 0 and 1.

Starting from the third number, the number is found by adding up the two numbers before it.

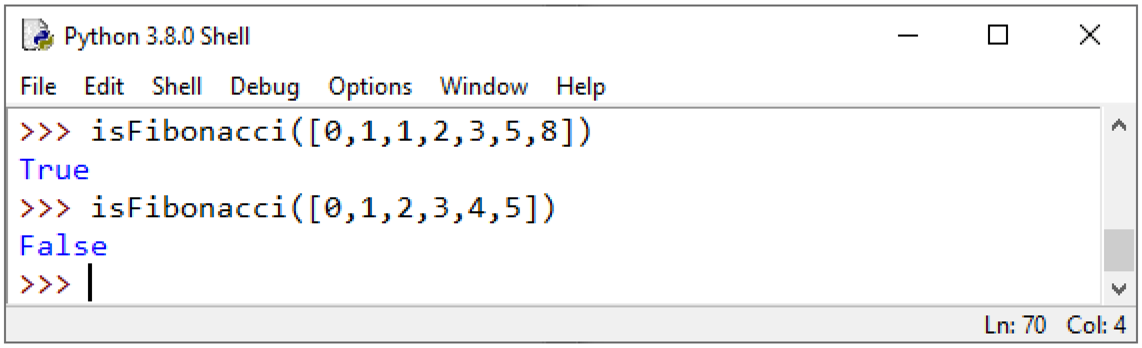
The 2 is found by adding the two numbers before it (1+1)

The 3 is found by adding the two numbers before it (1+2)

And the 5 is (2+3)

Write a function `isFibonacci(lst)` that determines if the given list is a list of Fibonacci Sequence. Return True if the list is a Fibonacci Sequence and False otherwise.

You can assume the list only contains integers and it **always contains more than two elements**.



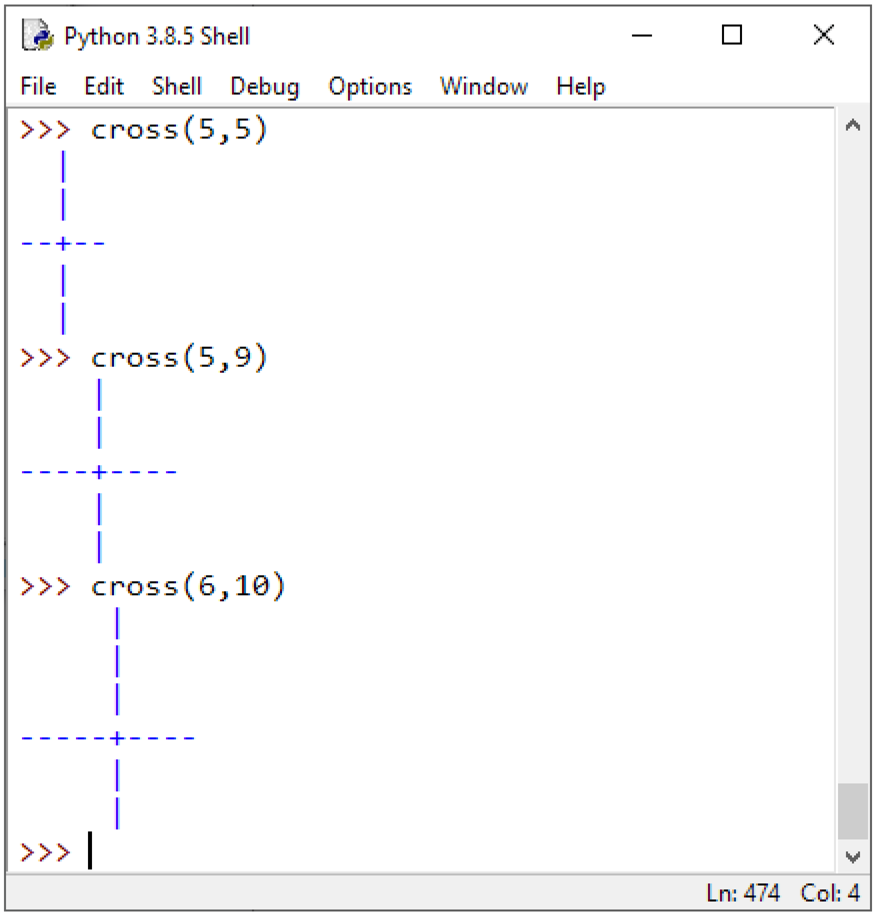
```
>>> isFibonacci([0,1,1,2,3,5,8])
True
>>> isFibonacci([0,1,2,3,4,5])
False
>>> 
```

So you will first check if the first element is 0 and second element is 1. Then starting from the third element, go through each element and check if it equals the sum of previous two elements.

4. ASCII Art [20 points]

Write a function `cross(n,m)` that prints a picture of an `n` x `m` cross as pictured below. The cross should consist of a horizontal line made of minus signs (`'-'`), a vertical middle line made of pipes (`'|'`) and a plus sign (`'+'`) at center.

*It is OK if the center is off for an even `n` or `m` because you use `n//2` or `m//2`.



```
>>> cross(5,5)
  |
  |
--+--
  |
  |
>>> cross(5,9)
    |
    |
----+----
    |
    |
>>> cross(6,10)
     |
     |
     |
-----+-----
     |
     |
>>> 
```