CSE 101: Introduction to Computers – Section 2 (MW 2:30-3:50 pm)

Stony Brook University

Lab #10

Spring 2017

Assignment Due: April 21, 2017 by 11:59 pm

Assignment Objective

This lab will give you some hands-on practice writing recursive functions.

Getting Started

For the labs and homework assignments in this course you will be writing functions that solve computational problems. To help you get started on each assignment, we will give you on Piazza a "bare bones" file with a name like `lab10.py`. These files will contain *function stubs* and a few tests you can try out to see if your code seems to be correct. Stubs are functions that have no bodies (or have very minimal bodies). You will fill in the bodies of these functions for the assignments. Do not, under any circumstance, change the names of the functions or their parameter lists. The automated grading system will be looking for exactly those functions provided in `lab10.py`. Please note that the test cases we provide you in the bare bones files will not necessarily be the same ones we use during grading!

Directions

- Solve the following problems to the best of your ability.
- At the top of the `lab10.py` file, include the following information in comments, with each item on a separate line:
  - your first and last name as they appear in Blackboard
  - your Stony Brook ID #
  - the course number (CSE 101)
  - the assignment name and number (Lab #10)
- Your functions must be named as indicated below. Submissions that have the wrong function names can't be graded by our automated grading system.
- Upload your `.py` file to Blackboard by the indicated due date and time. Late work will not be accepted for grading. Work is late if it is submitted after the due date and time.
- Programs that crash will likely receive a grade of zero, so make sure you thoroughly test your work before submitting it.

Some Advice on Writing Recursive Functions

Many students find it helpful when writing recursive functions to first write the function using iteration (loops). Doing so helps them to figure out the basic algorithm for solving the problem that can be transformed into a recursive solution. **However, your final solutions MUST NOT not be iterative. Use of loops for a function will result in 0 points for that function.**

Part I: Sum of Odd Integers (2 points)

Write a **recursive** function `sum_odds()` that takes a non-empty list of integers as an argument and returns the sum of only the odd integers in the list. In class we explored a recursive function called `rsum()` that recursively computes the sum of a list of integers – use it as a model to get started. Your function must be recursive and must not use any loops.

Examples:

Function Call	Return Value
`sum_odds([1, 8, 5, -6, 3, 4, 9])`	`18`
`sum_odds([5])`	`5`
`sum_odds([2, 6, 4, 8, -2, -20])`	`0`

Part II: Drop Every n-th Element (2 points)

Write a **recursive** function `drop()` that takes a non-empty list of integers and a positive integer n as its arguments, and returns a new list formed by dropping every n-th item from the original list. You may assume that $n > 1$. Your function must be recursive and must not use any loops.

Examples:

Function Call	Return Value
`drop([0, 1, 2, 3, 4, 5, 6, 7, 8], 3)`	`[0, 1, 3, 4, 6, 7]`
`drop([2, 4, 6, 8, 10, 12, 14], 2)`	`[2, 6, 10, 14]`
`drop([8, 6, 7], 4)`	`[8, 6, 7]`

Part III: Duplicating Letters (2 points)

Write a **recursive** function `duplicate()` that takes a non-empty string and a positive integer as its arguments, and returns a new string in which each letter of the original string has been duplicated the given number of times. The examples below will help to further clarify what the function is supposed to do. Your function must be recursive and must not use any loops.

Examples:

Function Call	Return Value
`duplicate('abcd', 3)`	`aaabbbcccddd`
`duplicate('Stony', 1)`	`Stony`
`duplicate('moonmoon', 2)`	`mmoooonnmmoooonn`

Part IV: Count Uppercase Letters (2 points)

Write a **recursive** function `count_upper()` that takes a non-empty string as its argument and returns a count of how many letters in the string are uppercase letters. Your function must be recursive and must not use any loops.

To solve this problem you will want to use the `isupper()` and `islower()` methods of string objects. For example, suppose we had the following variables:

```
letter1 = 'g'
letter2 = 'Q'
```

`letter1.isupper()` would evaluate to `False`, whereas `letter2.isupper()` would evaluate to `True`.

Examples:

Function Call	Return Value
`count_upper('Stony Brook University')`	3
`count_upper('HAPPINESS')`	9
`count_upper('no more labs, yay!')`	0

Part V: Count Uppercase and Lowercase Letters (2 points)

Write a **recursive** function `count_upper_lower()` that takes a non-empty string as its argument and returns a *tuple* containing the counts of how many letters in the string are uppercase and how many are lowercase (in that order). To solve this problem you will need to write a function that returns a pair of values (i.e., a *tuple*). For example, suppose we wanted to write a function that returned the sum and product of two values passed as arguments to the function. We might write this code:

```
def sum_prod(a, b):
    return a+b, a*b
```

Here is an example of how we might call the function and access the return values.

```
s, p = sum_prod(3, 6)
print(s,p)
```

In this example, the two return values will be assigned to `s` and `p` accordingly based on their position in the tuple. In other words, `a+b` will be assigned to `s`, and `a*b` will be assigned to `p`.

Here is some pseudocode you can consider implementing if you get stuck:

```
IF the word has only 1 character: (BASE CASE)
    1. return a pair of integers to indicate whether that character is an
       uppercase letter, lowercase letter, or neither
       (for example, the tuple (1,0) would indicate an uppercase letter)
OTHERWISE, the word has more than one character: (RECURSIVE STEP)
    1. determine if the first character is an uppercase letter, lowercase
          letter, or neither   (name this tuple (upper, lower))
    2. recursively count the number of uppercase and lowercase letters in
```

```
        the remainder of the word
    3. add the returned counts to your (upper, lower) tuple from above
```

Examples:

Function Call	Return Value
`count_upper_lower('Stony Brook University')`	`(3, 17)`
`count_upper_lower('HAPPINESS')`	`(9, 0)`
`count_upper_lower('no more labs, yay!')`	`(0, 13)`
`count_upper_lower('&@*:!@88??')`	`(0, 0)`

How to Submit Your Work for Grading

To submit your `.py` file for grading:

1. Login to `http://blackboard.stonybrook.edu` and locate the course account for CSE 101.
2. Click on "Assignments" in the left-hand menu and find the link for this lab assignment.
3. Click on the link for this lab assignment.
4. Click the "Browse My Computer" button and locate the `.py` file you wish to submit. You should be submitting only one file!
5. Click the "Submit" button to submit your work for grading.

Oops, I messed up and I need to resubmit a file!

No worries! Just follow the above directions again. We will grade only your last submission.