CSE 101: Introduction to Computers

Lab #4

Spring 2017

Assignment Due: March 3, 2017 by 11:59 pm

Assignment Objectives

By the end of this assignment you should be able to design, code, run and test original Python functions that solve simple problems involving Python strings, `for` and `while` loops, and lists.

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 `lab4.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 `lab4.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 `lab4.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 #4)
- 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.

Part I: Abundant Numbers (3 points)

An *abundant* number is an integer that is less than the sum of its *perfect divisors* (a perfect divisor is a value that divides a number evenly but is strictly less than the number itself). For example, 12 is an abundant number because its perfect divisors (1, 2, 3, 4, and 6) add up to 16.

Complete the `abundant()` function, which takes a single positive integer value n as its argument. The function returns a Python list containing exactly the first n abundant numbers in ascending order. For example, `abundant(3)` would return a list with the first three abundant numbers.

Hint: You will need two loops and several helper variables to solve this problem. Your helper variables (outside the loop) should track (among other things) the number of abundant numbers seen so far as well as the integer currently being examined (which will start at 2 and increase at the end of your outer loop). The first (outer) loop should continue until you have found the required number of abundant numbers. Inside that loop, use a second loop to find and add up the perfect divisors of the number currently being examined.

Examples:

Function Call	Return Value
`abundant(2)`	`[12, 18]`
`abundant(5)`	`[12, 18, 20, 24, 30]`
`abundant(10)`	`[12, 18, 20, 24, 30, 36, 40, 42, 48, 54]`

Part II: Elementary Arithmetic (3 points)

Susie is learning arithmetic, but she's not so good at it yet. When the teacher writes down the sum of several numbers together, like $1+3+2+1$, Susie has to rearrange the numbers into ascending order before adding them. For example, she would rearrange the previous example to $1+1+2+3$ before doing the math.

Complete the function `math_help(problem)` that takes a string representing a math problem and rearranges the digits so that Susie can compute the sum. The function will return the rearranged sum as a string.

You may assume that only digits 1, 2 and 3 appear in the argument string `problem`. You may also assume that at least one $+$ symbol appears in `problem` and that `problem` is always formatted properly.

Examples:

Function Call	Return Value
`math_help('3+2+1+1')`	`'1+1+2+3'`
`math_help('1+2+3')`	`'1+2+3'`
`math_help('2+2')`	`'2+2'`
`math_help('1+3+2+3+2+1+3+2+3+1+2')`	`'1+1+1+2+2+2+2+3+3+3+3'`

Part III: The Collatz Sequence (3 points)

Define a new function named `collatz` that will create a sequence of numbers that starts with a specified value. Initialize a list so it contains only the value passed as a parameter. Then use a while loop to extend the list using the following process: Let n be the value currently at the end of the list. If n is even, extend the list with $\frac{n}{2}$, but if

n is odd, extend the list with $3 \times n + 1$. An unproven conjecture from number theory is that eventually the number 1 will be appended to the list, which is when the process terminates.

Examples:

Function Call	Return Value
`collatz(2)`	`[2, 1]`
`collatz(15)`	`[5, 16, 8, 4, 2, 1]`
`collatz(17)`	`[17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]`

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.