CSE 101: Introduction to Computational and Algorithmic Thinking

Stony Brook University

Lab Assignment #7

Spring 2018

Assignment Due: March 25, 2018 by 11:59 pm

Assignment Objectives

This lab assignment will give you practice with simple recursive functions.

Getting Started

Visit Piazza and download the "bare bones" file `lab7.py` onto your computer. Open `lab7.py` in PyCharm and fill in the following information at the top:

1. your first and last name as they appear in Blackboard
2. your Net ID (e.g., jsmith)
3. your Stony Brook ID # (e.g., 111999999)
4. the course number (CSE 101)
5. the assignment name and number (Lab #7)

Submit your final `lab7.py` file to Blackboard by the due date and time. Late work will not be graded. Code that crashes and cannot be graded will earn no credit.

Part I: Recursive Hailstone Sequence (20 points)

Recall the **hailstone sequence** from an earlier assignment: the integer sequence that results from manipulating a positive integer value n as follows:

- If n is even, divide it by 2 (using integer division)
- If n is odd, multiply it by 3 and then add 1

Repeat this process until you reach 1.

For example, starting with $n = 5$, we get the sequence $5, 16, 8, 4, 2, 1$.

If $n = 6$, we get the sequence $6, 3, 10, 5, 16, 8, 4, 2, 1$.

Complete the function `rec_hailstone_length(n)`, which *recursively* computes and then returns the length of the hailstone sequence starting with `n`.

One possible recursive algorithm you can consider implementing is this:

```
if n = 1 then
    return 1
otherwise
    if n is even
        return 1 + the next hailstone value according to the "even" formula
    else
        return 1 + the next hailstone value according to the "odd" formula
```

Note: Non-recursive solutions to the problem that use loops will not earn credit.

Examples:

Function Call	Return Value
`rec_hailstone_length(2596)`	`147`
`rec_hailstone_length(641)`	`52`
`rec_hailstone_length(3153)`	`62`
`rec_hailstone_length(3148)`	`62`
`rec_hailstone_length(3012)`	`23`
`rec_hailstone_length(779)`	`60`
`rec_hailstone_length(308)`	`25`
`rec_hailstone_length(3722)`	`39`
`rec_hailstone_length(3448)`	`44`
`rec_hailstone_length(1171)`	`58`

Part II: Recursive Work Order Calculator (20 points)

Write a *recursive* function `rec_work_cost()` that takes two arguments, in this order:

1. `work_orders`: a list of strings representing work orders
2. `hourly_costs`: a tuple of three integers representing the costs per hour to fix a pipe, window and sprinkler, respectively

Each work order in `work_orders` starts with a single digit and is followed by `'pipe'`, `'window'` or `'sprinkler'`. The digit indicates the number of hours that will be spent repairing the given item. For instance, `'5sprinkler'` indicates that it will take 5 hours to repair a particular sprinkler. For the moment, assuming that `hourly_costs = (6, 2, 9)`. This means that it will cost $5 \times \$9 = \$45$ to repair the sprinkler. In general, `hourly_costs` will contain a different trio of values.

The function *recursively* processes the entire list of work orders and returns the total cost of performing all the work orders.

One possible recursive algorithm you can consider implementing is this:

```
if work_orders is empty then
    return 0
otherwise
    let remaining_costs be the cost of completing all of the other work orders
```

```
        (i.e., work_orders[1] and all later work orders)
    if work_orders[0] is for a pipe then
        return (the cost to repair the pipe + remaining_costs)
    otherwise, if work_orders[0] is for a window then
        return (the cost to repair the window + remaining_costs)
    otherwise, work_orders[0] must be for a sprinkler, so
        return (the cost to repair the sprinkler + remaining_costs)
```

Note: Non-recursive solutions to the problem that use loops will not earn credit.

Examples:

Function Call	Return Value
`rec_work_cost(['1sprinkler', '4window', '3pipe', '1window', '4pipe', '3pipe', '2window'], (7, 4, 6))`	`104`
`rec_work_cost(['3pipe', '3sprinkler', '4pipe', '3pipe', '1pipe', '3window', '2sprinkler', '4sprinkler', '3pipe', '4window'], (3, 6, 6))`	`138`
`rec_work_cost(['2pipe', '4pipe', '4pipe', '3sprinkler', '4window', '1sprinkler'], (6, 4, 5))`	`96`
`rec_work_cost(['3window', '4window', '1sprinkler', '4window', '2pipe'], (7, 4, 8))`	`66`
`rec_work_cost(['4window', '3window', '2pipe', '2sprinkler', '3sprinkler', '2pipe'], (8, 6, 4))`	`94`
`rec_work_cost(['1window', '3pipe', '3window', '3window', '3sprinkler', '2sprinkler'], (7, 7, 7)]`	`105`
`rec_work_cost(['2sprinkler', '2window', '3sprinkler', '2pipe', '4sprinkler', '2sprinkler'], (3, 5, 3))`	`49`
`rec_work_cost(['4window', '3sprinkler', '2pipe', '2pipe', '3window', '2window', '4sprinkler', '1window', '2sprinkler', '1pipe'], (7, 2, 7))`	`118`
`rec_work_cost(['3sprinkler', '2sprinkler', '2pipe', '1pipe', '1pipe', '2pipe', '1pipe', '4pipe', '1pipe', '3sprinkler'], (2, 3, 5))`	`64`
`rec_work_cost(['4pipe', '4sprinkler', '4sprinkler', '3window', '4pipe', '4pipe', '3pipe', '1window', '4pipe', '2window'], (3, 7, 7))`	`155`

How to Submit Your Work for Grading

To submit your `.py` file for grading:

1. Login to Blackboard and locate the course account for CSE 101.
2. Click on "Assignments" in the left-hand menu and find the link for this assignment.
3. Click on the link for this assignment.
4. Click the "Browse My Computer" button and locate the `.py` file you wish to submit. Submit only that one

`.py` 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.