National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2024/2025

Tutorial 3
Advanced Iteration

Release date: 2nd September 2024
Due: 8th September 2024, 23:59

General Restrictions

- No importing additional packages unless explicitly allowed to do so.
- Do not use any compound data structures, such as `tuple`, `list`, `dict`, `set`, etc.

Questions

1. Determine the exact output when the expressions are evaluated.

   ```
   def baz():
       count = 0
       for i in range(1, 6, 1):
           if i % 3 > 0:
               count = count + 3
           if i % 2 == 0:
               continue
           print(count)
       while count != 0:
           count = count // 2
           if count <= 1:
               break
           print(count)
   print(baz())
   ```

2. The `num_pairs` function takes a **string** as input and returns the **number of consecutive character pairs in the string**. For example, the strings `'balloon'` and `'missisippi'` given as input both return 2 because `"balloon"` has the pairs `"ll"` and `"oo"` while `"missisippi"` has the pairs `"ss"` and `"pp"`.

   Note that a character may contribute to multiple pairs. For example, the string `'crosssection'` contains 2 pairs of `"ss"` and the string `'ahhhh!!'` contains 4 pairs (3 `"hh"` and 1 `"!!"`).

   Provide an **iterative** implementation of the function `num_pairs`.

3. You are in a maze at coordinates (0,0) (top left corner), and need to retrieve some building materials at coordinates (x,y). You can only travel from left to right on even-numbered layers (starting from the first layer on the top being layer 0), and vice-versa (right to left) on odd-numbered layers.

   An example is shown in the image below, where the dots are the path, the arrows denote which direction you will go when you make your next move after reaching that point, and the asterisk (*) denoting the goal state (where your tools are).

   ```
   >.......V
   V.......<
   >..*....V
   ........<
   ```

   The traversal of the maze can be thought of as a repetition of the following two steps:

   - If on an even-numbered layer, keep **moving right**! In each step, if you have reached the destination, then we are done. Otherwise, if you reach the **right** boundary, then you need to move down into the next layer.
   - If on an odd-numbered layer, keep **moving left**! In each step, if you have reached the destination, then we are done. Otherwise, if you reach the **left** boundary, then you need to move down into the next layer.

   Write the functions `right_to_left` and `left_to_right`. Both functions take in the following four inputs: `row`, `width`, `x`, and `y`. `row` is the current row we are moving left or right in, `width` is the width of the maze, `x` and `y` describes the coordinates of the building materials.

   Both functions **prints** the coordinates that you have to traverse while moving left or right, and **returns** a `True` or `False` depending on whether you have reached the destination.

   Thereafter, write a function `coord` which takes in the height & width of the maze, the coordinates x & y, and prints out the coordinates that you have to traverse (in order). This function should make use of `right_to_left` and `left_to_right` to do the printing!

Sample Execution:

```
>>> left_to_right(0, 3, 3, 3) # row, width, x, y
    (0,0)
    (1,0)
    (2,0)
    False
>>> right_to_left(1, 3, 1, 1)
    (2,1)
    (1,1)
    True

>>> coord(4, 9, 2, 3) # height, width, x, y
(0,0)                                  (0,1)
(1,0)                                  (0,2)
(2,0)                                  (1,2)
(3,0)                                  (2,2)
(4,0)                                  (3,2)
(5,0)                                  (4,2)
(6,0)                                  (5,2)
(7,0)                                  (6,2)
(8,0)                                  (7,2)
(8,1)                                  (8,2)
(7,1)                                  (8,3)
(6,1)                                  (7,3)
(5,1)                                  (6,3)
(4,1)                                  (5,3)
(3,1)                                  (4,3)
(2,1)                                  (3,3)
(1,1)                                  (2,3)
```