Specification for Assignment 3 for COMP1405 (Fall 2016)

Instructions

There are two questions in this assignment. For each question you will be submitting your solution as multiple files, each containing source code written in Python 3. These files must be compressed into a single "zip" file, and you will submit this file using cuLearn.

- ☐ *The source file for question 1 must be named "a3q1.py".*
- ☐ *The source file for question 2 must be named "a3q2.py".*
- ☐ *The compressed archive (i.e., the .zip file) must be named "a3.zip".*
- ☐ *The due date for this assignment is October 15, 2016, by 11:30pm.*

♦

Late assignments will be accepted for 48 hours after the deadline, but the penalty for submitting a late assignment is a loss of 2.0% per hour.

♦

You are expected to demonstrate good programming practices at all times (e.g., choosing descriptive variable names, provide comments in your code, etc.) and your code will be penalized if it is poorly written.

♦

You are expected to do the necessary preparatory work (e.g., devising an algorithm) before you begin coding. Whenever appropriate, you will be asked to present either pseudocode or a flowchart before you will receive any assistance from the instructor or a teaching assistant.

♦

This assignment is uniquely generated; every student in the class is required to complete a slightly different version of this assignment. To ensure that each unique assignment shares the same level of difficulty, a unique assignment generator (and supporting files) has been made available on cuLearn.

To receive the assignment instructions that are specific to you, download the "unique-assignment-generator-for-A3.zip" file from cuLearn. Once you have extracted the contents to your working folder, use the command prompt to run the "generator-for-A3.py" program and then enter and confirm your student number.

Question 2 (of 2)

When you ran the unique assignment generator for this assignment, it showed you a maze and specified two locations, on the top and bottom row, respectively. For the first question you will be writing a program that uses several sequenced loops to move from the start (on the top row, in green) to the end (on the bottom row, in red) without crossing any walls.

You should start by actually solving your maze. If you assume the existence of four functions - `goUp()`, `goDown()`, `goLeft()`, and `goRight()` - then you should be able to see that the path solving a maze could be represented as a sequence of calls these functions. The maze depicted right, for instance, could be solved by a sequence of function calls - `goRight()` `goRight()` `goDown()` `goDown()` `goDown()` `goLeft()` `goLeft()` `goLeft()` `goLeft()` - but this is an inefficient solution. Consider, instead, three looping control structures - with the body of the first, second, and third loops being single calls to the `goRight()`, `goDown()`, and `goLeft()` respectively. The first loop would repeat its body twice, while the second and third loops would repeat their bodies three and four times, respectively.

For this question, you will write a program that consists of a sequence of implementations of different types of loop. Each of your loops will have (as its body) only a single call to one of those four functions. Your program is not permitted to call any of these functions unless that function appears in the body of the loop, and a loop that calls a specific function in its body must not immediately precede or follow another loop that calls the same function.

The last loop (i.e., the one that causes you to arrive at the red ending point) to appear in your program must be a postcondition loop that was implemented using a "while" loop and the "break" statement. The loop before that must also be a postcondition loop, this time implemented a "while" loop and a Boolean "flag". Every other time you need to move left or right, you must use a single call to either `goLeft()` or `goRight()` and this call must appear in the body of a counter-controlled loop that was implemented using a "for" loop, and every other time you need to move up or down, you must use a single call to either `goUp()` or `goDown()` and this call must appear in the body of a precondition loop that was implemented using a "while" loop.

Within the zip file was a file named "helper.py". This is a module that you can import into your own code that will allow you to actually run your solution to this problem from the command line. Attempting to run the program you write without this module will result in NameErrors.

Once again, because the teaching assistants will be provided software to help them mark these submissions, your solution must not include anything other than the requested loops - do not make function calls to input, print, etc. and do not use lists, dictionaries, sets, or any other advanced data types in your solution to this problem.

Question 2 (of 2)

When you ran the unique assignment generator for this assignment, it also specified 6 different "types" of characters. It may, for example, have mentioned numbers between 0 and 5, uppercase letters between A and D, etc. For this question you are trying to generate all possible codes that use each of these different character types by using a single nested loop to iterate through all the different possibilities. Alternatively, assume that you are trying to exhaustively determine all the possible identification codes that follow a specific format.

As a clarifying example, the forward sortation area code is the first three characters of a Canadian postal code (e.g., the "K1S" in postal code "K1S 5B6") and it has the format:

a letter between A and Z

a number between 0 and 9 including zero and including 9

a letter between A and Z

Thus, if your assignment generator instructed you to write a nested loop to exhaustively print all possible forward sortation codes, you would expect to see:

A0A
A0B
A0C
...
A0Z
A1A
A1B
...
Z9Z

You are not permitted to use Python lists, dictionaries, or sets to generate letters in a specific range. Instead you are expected to use the `ord()` and/or `chr()` functions (for more details refer to https://docs.python.org/3/library/functions.html) and please also recall that the modulation operator (i.e., "`%`") provides the remainder after a long division operation and, consequently, can be used to determine the divisibility of a number. If you are not yet certain how to prevent calls to the print function from making a new line every time, the reference above has that information as well.