CS 1110 review session for Prelim 2, April 2023

1. **For-loops and dictionaries.** An *inverted string* is a dictionary whose keys are characters and whose values are lists of positions. For example, the string `'hello'` is inverted as

   `{ 'h':[0], 'e':[1], 'l':[2,3], 'o':[4] }`

   You are free to use anything about lists or dictionaries to create an inverted string.

```
def invert(s):
    """Returns: An inverted string representing s

    Example: invert('abcac') is {'a':[0,3], 'b':[1], 'c':[2,4]}
    Example: invert('') is {}.

    Precondition: Parameter s is a (possibly empty) string
    """
```

2. **Classes.** Classes `Course`, `Student`, and `Schedule` (see enroll-handout.pdf) are part of the Registrar's database, which records which courses each student is enrolled in, and which students are enrolled in each course. Two methods are unimplemented: `Student.add_course` (line 84), which updates the database to reflect a student enrolling in a course, and `Student.validate` (line 92), which makes sure a student's schedule follows the rules.

   Read the code to become familiar with these classes. The 3rd page is a set of tests to help you to understand how these classes are used. Implement the two incomplete methods below.

```
class Student():

    def add_course(self, course):
        """Add a course for the current semester.  This means the course
        is added to the student's current schedule, and the student is
        added to the enrollment of the course.
        Pre: <course> is a Course, the student has a current schedule, and
             <course> is not already on current semester's schedule."""
        # TODO: implement this method








    def validate(self, credit_limit):
        """Return: True if the student's schedule for the current semester
        is valid, which means that
           (a) the total number of credits in current semester is not over
               <credit_limit> (credits from prior semesters don't matter)
           (b) student is not taking any courses in current semester that
               they already took in a previous semester. Course titles
               determine when a course is repeated; see Schedule.overlaps.
        Pre: credit_limit [integer] ; student has a current schedule."""
        # TODO: implement this method
        # Take the time to read through all the methods in Schedule:
        # using them makes this method much shorter to implement.
```

3. **Recursion on nested lists.**

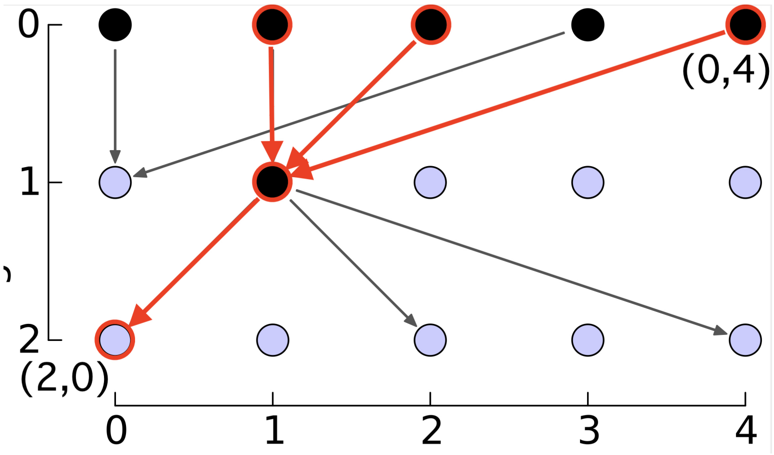
```
def embed(theinput):
    """Returns: depth of embedding--or nesting--in theinput.

    Examples:
        "the dog that barked" -> 0
        ["the", "dog", "that", "barked" ] -> 1
        ["the", ["dog", "that", "barked"]] -> 2
        ["the", ["dog", ["that", "barked"]], ["was bad"]] -> 3
        ["the", ["dog", ["that", ["barked"]]]] -> 4
        [[[["the"], "dog"], "that"], "barked"] -> 4

    Precondition: theinput is a string, or a potentially nested
    non-empty list of strings. No component list can be empty
    """
```

4. **Recursion on object structures.** Assume we have written a definition for class Node where each node has a contacted_by attribute consisting of a (possibly empty) list of nodes that have contacted it.[1] This question asks you to add a new method for class Node; implement it according to its specification. Your solution must make effective use of recursion, though it can involve for-loops as well.

   Example: In the figure below, (2,0) is downstream from (0,1), (0,2), (0,4), and (1,1), but no other nodes.



```
class Node():

    def is_downstream_from(self, older):
        """Returns True if: older is in this node's contacted_by list, OR if
           at least one of the nodes in this node's contacted_by list is
           downstream from older.

           Returns False otherwise

           Pre: older is a Node
        """
```

[1] And we know that anything in a node's contacted_by list is from an earlier "generation". This is a technical condition that prevents the possibility of cycles in the node contacting.