DVA245 Lab Assignment 3
Leaky Stack, Linked Scoreboard

Anna Friebe

January 2021

1 Leaky Stack

In the first part of the assignment you will implement a leaky stack, that is a stack with a maximum depth. If you push additional elements onto the stack when it is full, those are added, but the lowermost/ oldest element on the stack is leaked (overwritten with the new object). This kind of data structure can be useful for example for saving the most recent user inputs for undo functionality while bounding the memory use.

The leaky stack implementation shall use a python list as the underlying structure to store the stack elements.

Since the elements are leaked and overwritten with the new element, it means that the top element of the stack will move, and not remain at the last list element. You can see an example of a non-leaky stack with a Python list for underlying storage in `array_stack.py` in ch06 of the code related to the book: https://github.com/mjwestcott/Goodrich. In the same folder you find the `array_queue.py` queue implementation with a list storage. Here you can find hints on how to move the stack top element similarly as the queue front is moved. This is done by using the modulus operator % (that is also explained in this youtube video: Modulus Operator - CS101 - Udacity).

Illustrations of stack functionality and leakage with top index changes is displayed in Figures 1 and 2.

Download files from Canvas that provide tests and a skeleton of your LeakyStack class and tests for it.

1.1 Details

In the LeakyStack skeleton in `leaky_stack.py` you need to implement:

- The constructor.
- The `len` operator that returns the number of elements currently on the stack.
- The `isEmpty` function.

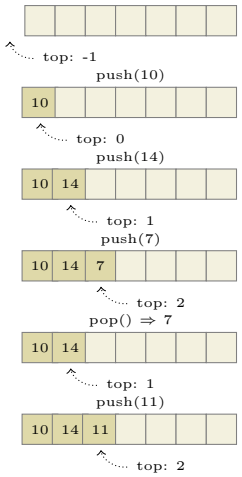

Figure 1: Illustration a stack with top index changes.

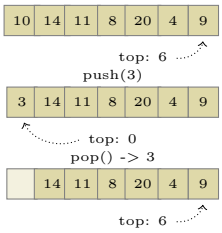

Figure 2: Illustration of stack leakage and top index changes.

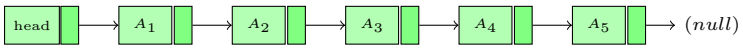

Figure 3: A singly linked list.

- The `top` function that returns the element on the top of the stack without removing it.
- The `pop` function that removes and returns the element on the top of the stack, and adjusts the top index and the current size of the stack accordingly.
- The `push` function that adds an element to the top of the stack. The top index and the current size of the stack are adjusted depending on previous values and the stack capacity.

More details can be found in the `leaky_stack.py` file.

1.2 How to display

You need to run the tests in `leaky_stack.py` and show that they pass.

2 Linked Scoreboard

In the second part of the assignment you will implement a scoreboard to keep track of names and scores of a number of games with the highest scores. The underlying data shall be stored in a linked list structure.

A linked list structure consists of nodes, where each node holds element data (in this case the player's name and the game's score) and a reference to the next node in the list. The scoreboard holds a reference to the first node in the list, the head node (that is None if the scoreboard is empty). An illustration of the linked list is seen in Figure3.

Examples where linked lists are used to implement a linked stack and a linked queue are available in the code related to the book as linked above. They are available in ch07, if you want to study code that updates the next pointer in nodes and a head node reference.

In the file `linked_scoreboard.py` is a skeleton for the Linked Scoreboard where the nested Node class is implemented. It also contains tests for the `LinkedScoreboard` class.

2.1 Details

In the LinkedScoreboard skeleton, you need to implement:

- The constructor. The LinkedScoreboard keeps a reference to the first node, the head of the linked list. It keeps the capacity limit of the scoreboard and possibly the current number of nodes in the list for convenience.

- The `len` operator that returns the current number of nodes in the list.
- An `isEmpty` function that returns True if the linked list is empty.
- The `addScore` function that finds the correct position for a new score, where the node with the highest score of the scoreboard is at the head of the list, and subsequent nodes are sorted in decreasing score order. A node is created if the score is high enough to enter within the capacity of the scoreboard, and next references of nodes are updated as needed. If the scoreboards capacity is exceeded, the last node of the linked list is removed.
- The helper function `_getNode` that returns the node at a specific position in the scoreboard.
- The functions `score` and `playerName` that returns the score and player name respectively at a specific position, and use the `_getNode` function.
- The `printHighScoreList` that prints all the contents of the scoreboard. This shall not use the `_getNode`, `score` or `playerName` functions, but instead go through the nodes in the list once to print the content.

More details can be found in the `linked_scoreboard.py` file.

2.2 How to display

You need to run the tests in `linked_scoreboard.py` and show that they all pass, and show the output of the `printHighScoreList` calls.