CPE 202 Winter 2020

Project 1: Stack Implementation
Due: January 27, 2020 @ midnight

The goal of this project is to implement the Stack Abstract Data Type (ADT). You are going to implement two versions of the Stack ADT: one with an array (StackArray class), and the other with a linked-list (StackLinked class). For both implementations you are going to allow your stack to grow as large as it can.

This project is based on Lab 2: we are going to reuse the classes and functions we implemented in Lab 2. So, you want to make sure that your array_list.py and linked_list.py work perfectly before starting this project assignment.

StackArray:

- Import ArrayList from your array_list.py (from array_list import ArrayList)
- Also, import array_list.py (import array_list). Now, you can use functions you defined in your array_list.py by prepending array_list. to the name of any of the functions: array_list.insert().
- create a member variable called arr_list (a variable defined in StackArray class: self.arr_list) in the StackArray class. The arr_list will be an object of ArrayList class you created in Lab 2. Initialize the stack by creating an object of ArrayList class, and assign it to self.arr_list in __init__ method of StackArray.
- create a member variable called num_items (a variable defined in StackArray class: self.num_items), which stores the number of items in the Stack, and also works as a pointer pointing to the top of the Stack: the top of the Stack points to a position where a new item will be pushed. The initial value of self.num_items should be 0.
- Implement push method in StackArray class to push an item to the stack. To push an item, you can do so by using your insert function in array_list.py: self.array_list = array_list.insert(self.array_list, item, self.num_items). Do not forget to increment self.num_items by 1 after the insert has been done successfully. In Lab 2, we assumed that an item is an int, but in Project 1 we store an item of any type in the array_list. You should be able to do so without changing your code in array_list.py. If you encounter some problems, fix your array_list.py. The array should be enlarged automatically by the insert function in array_list.py.
- Implement pop method in StackArray class to pop an item from the stack. When a user tries to pop an empty stack, your pop function (method) should raise an IndexError, which is supposed to be done by your pop function in array_list.py: self.array_list, val =

array_list.pop(self.num_items - 1). Do not forget to decrement self.num_items by 1 after the pop has been done successfully. The array should be shrinked automatically by the pop function in array_list.py.

- Create peek method in StackArray class, which returns the value at the top of the Stack. Implement this method by using a function in array_list.py. Raise IndexError when peek is called on an empty Stack.
- Create is_empty method in StackArray class, which returns True if the Stack is empty.
- Create size method in StackArray class, which returns the number of items in the Stack: return self.num_items.

StackLinked

- Import Node from your linked_list.py (from linked_list import Node)
- Also, import linked_list.py (import linked_list). Now, you can use functions you defined in your linked_list.py by prepending linked_list. to the name of any of the functions: linked_list.insert().
- create a class called StackLinked.
- create a member variable called top and initialize it to None in StackLinked class. self.top will point to the last item (Node) in the Stack. self.top is initially None when the Stack is empty. The first item to be pushed will be assigned to self.top. When the next item is pushed, the next item's next will point to self.top, and self.top will be updated so that it points to the next item (self.top will always point to the last item in the Stack (the top of the Stack)). See the description of push method below.
- create a member variable called num_items and initialize it to 0. Update the value of num_items as items are pushed or popped from the Stack.
- Create push method in StackLinked class. The push method has one argument, item, besides the self. This method shall push the item to the Stack. You can do so by using functions in your linked_list.py from Lab 2: self.top = linked_list.insert(self.top, item, ?). Do not forget to increment the self.num_items by 1 after an item has been successfully pushed to the stack. linked_list.insert() raises IndexError if an illegal position is specified. This means that the push method should handle the error. In Lab 2, we assumed that an item is an int, but in Project 1 we store an item of any type in the linked_list. You should be able to do so without changing your code in linked_list.py. If you encounter some problems, fix your linked_list.py.
- Create pop method in StackLinked class to pop an item from the top of the Stack. You can implement the pop method by using pop function in linked_list.py: self.top, item = linked_list.pop(self.top, ?). Do not forget to decrement the self.num_items by 1 after an item has been successfully popped from the stack. When a user tries to pop an empty stack, the pop function raises an IndexError. This means that the pop method in StackLinked will also raise the error. This method needs to return an item (a value not a Node object).

- Create peek method in StackLinked class, which returns the value at the top of the Stack. Implement this method by using a function in linked_list.py. Raise IndexError when peek is called on an empty Stack.
- Create is_empty method in StackLinked class, which returns True if the Stack is empty.
- Create size method in StackLinked class, which returns the number of items in the Stack: return self.num_items.

PolyLearn has a starter file **stacks.py** which provides a starting point.

Submit to polylearn four files as one zip file:

1. **array_list.py and linked_list.py**.
2. **stacks.py** containing an array-list based implementation of stack and a linked-list implementation of stack. The classes must be called: **StackArray** and **StackLinked** . Both implementations should follow the above specification and be thoroughly tested.
3. **test_stacks.py** containing your set of tests to ensure your classes work correctly. Write your own tests using unittest module. The file should contain enough test cases to cover all possible use cases.

Make sure that you follow the design recipe. (Your submission files do not need to contain template.) Submit your work to the grader, and then submit it to the polylearn. Note: Your class names, function names and files name must follow the spec of the project.