Lab Week 2: Stack Implementation

The goal of this lab is to implement the Stack Abstract Data Type using two different implementations.

1. Using the built in (Python) List construct with a fixed capacity. You **may not use** the list methods that provide the equivalent functionality to the methods in the stack interface.
2. The simple linked data structure covered in class.

As discussed in class you are to allocate a list of size stack_capacity and use this to store the items in the stack. Although the usual list in python expands when more storage is needed, there is a mechanism for a fixed size list, for example my_list = [None]*10 creates a list of size 10. Note you will implement a method to check if the stack is full. (This prevents the stack from using more space that a user might want. Think of this as a requirement for an application on a small device that has very limited storage.) In the case when a user attempts to push an item on to a full stack, your push function (method) should raise an IndexError. Similarly if a user tries to pop an empty, your pop function (method) should raise an IndexError. Again, you **cannot** use any of the built in list operations to implement your stack classes.

Additional Requirements:

- All stack operations must have O(1) performance (including the size() operation)
- Your stack implementations must be able to hold values of None as valid data

The following starter files are available in Canvas, and after you have added your code, these are the files you will upload to Canvas for this lab:

- **stack_array.py**: Contains starter code for an array (Python List) based implementation of the **Stack** class
- **stack_nodelist.py**: Contains starter code for a linked based implementation of the **Stack** class
- **stack_array_tests.py:** Contains comprehensive tests to ensure your implementations in stack_array.py work correctly.
- **stack_nodelist_tests.py:** Contains comprehensive tests to ensure your implementations in stack_linked.py work correctly.

(Note that the class in each stack implementation is named **Stack**, and both implementations follow the same specification in regard to the operations on the Stack)

Submit to Canvas the four files above. The implementations should follow the specification and be thoroughly tested. The **stack_array_tests.py and stack_nodelist_tests.py** contains **your** set of tests to ensure you classes work correctly. Do not change the names of these files or of their functions.

Make sure that functions have a docstring that describes its purpose.

Avoid pitfalls:

- Submit files with the correct filenames.
- Ensure that the function/method signatures are correct. Again make no changes no matter how minor.
- Do not rely on any additional classes etc that are not part of standard python.
- Make your code as clear as possible. Use comments judiciously. Do not explain the obvious.
  comments like – this is a for loop over the array etc are not helpful.
  better are comments that explain the point of a block of code –
- **push(x)** really only needs a single short comment e.g inserts x at the top of the stack