**CS469 Data Structure and Algorithm**

**HOS03 Stack and Recursion**

04/12/2023 Reviewed by Christopher Sharp

01/16/2024 Updated by Anh Nguyen

09/29/2024 Reviewed by Shahid Khan

School of Technology and Computing (STC) @City University of Seattle (CityU)

**Before You Start**

* The document’s examples are written in Python. If you don’t know Python programming language, please finish the Python tutorial in HOS00 before you start the assignment.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

* Implement a stack using list
* Implement a stack using singly linked list
* Understand recursion

**Resources**

* Geeksforgeeks (2021). Implement a stack using singly linked list. Retrieved from: <https://www.geeksforgeeks.org/implement-a-stack-using-singly-linked-list/>
* Geeksforgeeks (2021). Stack in Python. Retrieved from: <https://www.geeksforgeeks.org/stack-in-python/>
* Geeksforgeeks(2021). Recursion. Retrieved from: <https://www.geeksforgeeks.org/recursion/>
* Visualgo. (2021). Stack. Retrieved from: <https://visualgo.net/en/list>

1. **Stack**

Stack is a LIFO (Last In First Out) data structure.

Last In First Out means that:

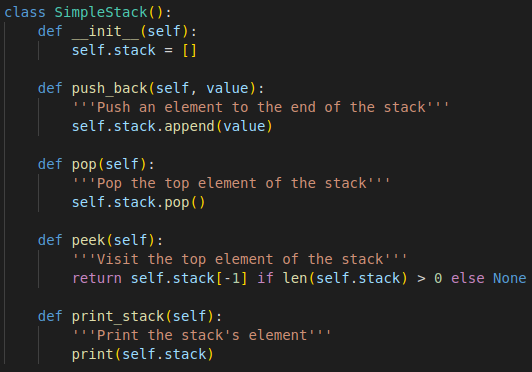
* The element placed on the top of stack will be removed first
* The element placed at the bottom of the stack will be removed last.

Think of a stack of pringles potato chips, if you want to eat the piece of potato chip at the bottom of the stack, how can you do it?

Well, if you can’t cut open the bottom, you will have to eat from top to bottom. It is how stack works. The first piece of potato chip that you can eat must be the last piece of potato that has been placed on the stack.

1.1 Simple Stack

* Open thehosfolder, create a file called **simpleStack.py**
* Type the following code in **simpleStack.py**



**Code explanation:**

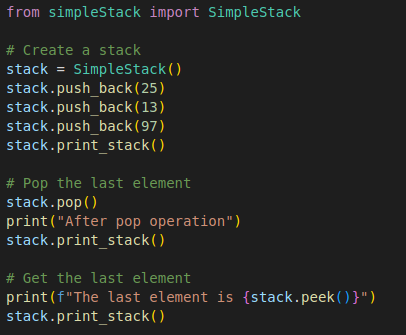
**push\_back():** push an element to the end of the stack

**pop():** pop the top element of the stack

**peek():** visit the top element of the stack

**print\_stack():** print the stack’s element

* Create a file called **testStack.py**
* Type the following code in **testStack.py**



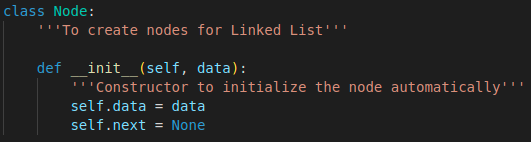
**Output should look like this:**

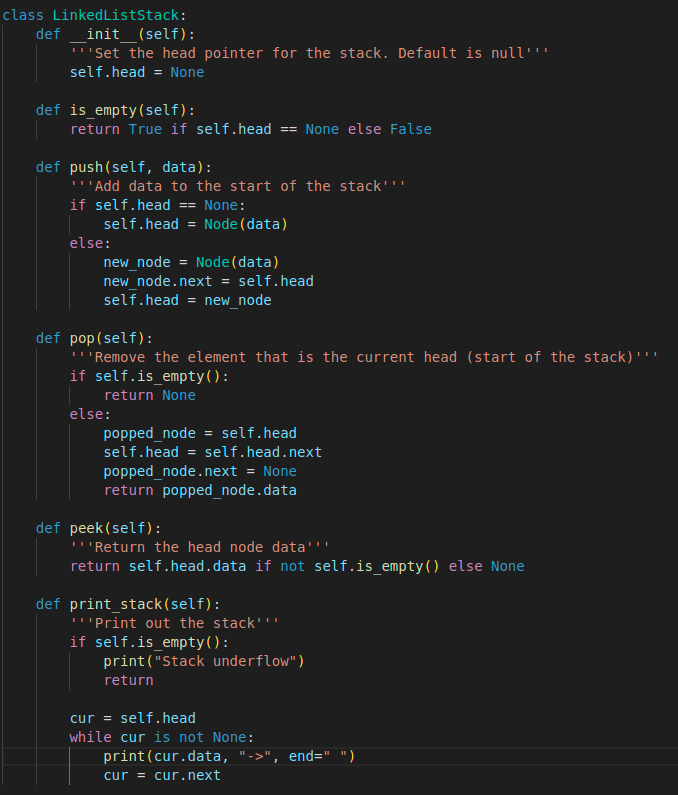
Graphical user interface, text, application

Description automatically generated

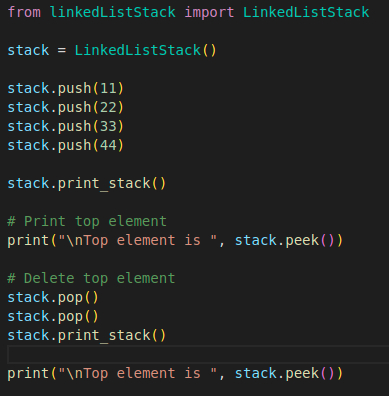
**1.2** Simple Stack using singly linked list

* Create a file called **linkedListStack.py**
* Add the following code for class Node and LinkedListStack in **linkedListStack.py**





* Create a file called **testLinkedListStack.py**
* Type the following code in **testLinkedListStack.py**



**Output should look like this:**

Text

Description automatically generated with medium confidence

1. **Recursion**

Recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem. In simple words, recursion is a process that a function calls **itself**.

To use recursion, the first thing is to define the base cases, which is the export of the recursion – the condition to stop recursion. If there are no base cases, the function will **infinitely** call itself.

Common recursion examples include Tower of Hanoi, Fibonacci sequence.

Let’s take the Fibonacci sequence as an example.

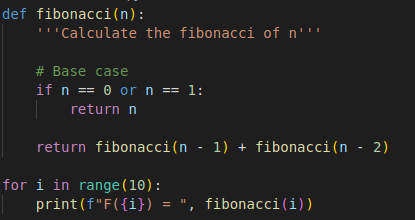
The Fibonacci sequence is a sequence in which each number is the sum of the two preceding ones. Numbers that are part of the Fibonacci sequence are Fibonacci numbers (Fn).The sequence commonly starts from 0 and 1. The Fibonacci numbers can be represented with the following mathematical formula:

F0 = 0, F1 = 1 and Fn = Fn-1 + Fn-2

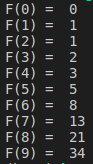
It represents, for n == 0 or n == 1, the Fibonacci of n equals to themselves, elsewise it equals to the sum of Fibonacci n – 1 and Fibonacci n – 2.

Recursion uses a system stack to store intermediate values generated during recursion process. If the function called itself too many times during the recursion process, it would lead to the stack overflow problem.

* Create a file called **fibNumbers.py**
* Type the following code in **fibNumbers.py**



**Output should look like this:**



**Q: Try n == 2\*\*10, what message do you see in the terminal, and why?**

**Save your answer + console output screenshot in a pdf file with the name Submission.pdf.**

**Push Your Work to GitHub**

Open terminal and make sure you’re in the repository folder.

(i.e: hos03\_courseName\_GitHubUserName)

**Type the following command to upload your work**:

>>>> git add .

>>>> git commit -m “Submission for HOS03 - <Your name>”

>>>> git push origin master