CMPUT 175 - Lab 5: Stacks & Exceptions & BROWSE-175

THIS LAB HAS TWO REQUIRED PROBLEMS, A (Stacks & Exceptions) AND B (BROWSE-175)
YOU MUST COMPLETE AND DEMO BOTH PROBLEMS

PROBLEM A

Goal: Become familiar with a new data structure: Stacks. Also practice raising and handling exceptions.

Background Information:
A stack is an *abstract data type* (ADT) that stores items like a list, but you can only access the last item which has been added. To access the second last item, you must remove the last item from the stack, and so on. A stack follows what is called a *first in - last out* structure.

Problem Description:
Suppose you have a partial implementation of a stack, given in **stack.py.** We want you to complete this partial implementation by modifying the *pop()* and *peek()* methods so they raise an Exception with a relevant message as a custom argument if these methods are invoked on an empty stack (see Exercise 1). Furthermore, we want you to implement a web-browser simulator. That is, a program that will act exactly like a web browser does (just like Lab 4).

Ensure that you write proper docstrings and follow the Software Quality Requirements. See Deliverables on the last page.

Exercise 1:

1. Download and save **stack.py** from eClass. This file contains implementation #2 of the Stack class covered in the lectures.

2. Modify the *pop()* and *peek()* methods so that they raise an **Exception** with a relevant message as a **custom** argument if these methods are invoked on an empty stack. The exception should **NOT** be handled in the Stack class.

3. The ADT for the Stack has been updated so that it has an additional behavior:
   clear()
   - Removes all items currently in the stack; does nothing if the stack is currently empty.
   - It needs no parameters, and returns nothing.

Update your Stack implementation by adding a new *clear()* method that follows the updated ADT specification.

Exercise 2:

You are tasked with creating a web browser simulator. The simulator will work the same as in **Lab 4**, but this time you must implement it using two stacks: one to enable the **back button** functionality, and one to enable the **forward button** functionality.

1. Download and save a copy of **lab5_browser.py** from eClass. (Be sure that you save it in the same directory as stack.py). This file contains a *main()* function which controls the flow of operation of a web browser simulation. In the following steps, you will complete the functions that this *main()* function calls.
2. Complete **getAction()**. This function prompts the user to enter either a **'='** (to enter a new website address), **'<'** (back button), **'>'** (forward button), or **'q'** to quit the browser simulation. If the user enters something other than these 4 characters, an **Exception** should be raised with the argument **'Invalid entry.'** This Exception is **NOT** handled in this function, but in *main()*. This function has no inputs. If no exception is raised, this function returns the valid character entered by the user (str).

3. Complete **goToNewSite()**. This function is called when the user enters **'='** during *getAction()*. This function prompts the user to enter a new website address, and returns that address as a string. It also updates the two stacks, as appropriate. (**Hint:** experiment with how the back and forward buttons work on a real web browser like Firefox or Chrome. After a new address is entered, can you still go forward?) Note that you do not need to explicitly return the two stacks because the Stack (as we implemented it) is a mutable object – so *bck* and *fwd* are actually just aliases for the stacks called back and forward in your main function. The inputs for this function are the current website (str), a reference to the Stack holding the webpage addresses to go back to, and a reference to the Stack holding the webpage addresses to go forward to.

4. Complete **goBack()**. This function is called when the user enters **'<'** during *getAction()*. Handle any exceptions that are raised by the Stack class (i.e. when there are no webpages stored in the back history) by displaying an error message and returning the current site (str). Otherwise, the previous webpage is retrieved (and returned as a string), and the two stacks are updated as appropriate. The inputs for this function are the current website (str), a reference to the Stack holding the webpage addresses to go back to, and a reference to the Stack holding the webpage addresses to go forward to.

5. Complete **goForward()**. This function is called when the user enters **'>'** during *getAction()*. Handle any exceptions that are raised by the Stack class (i.e. when there are no webpages stored in the forward history) by displaying an error message and returning the current site (str). Otherwise, the next website is retrieved (and returned as a string), and the two stacks are updated as appropriate. The inputs for this function are the current website (str), a reference to the Stack holding the webpage addresses to go back to, and a reference to the Stack holding the webpage addresses to go forward to.

Please see the next page for a sample output

Sample run:

```
Currently viewing www.cs.ualberta.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: 123
Invalid entry.
Enter = to enter a URL, < to go back, > to go forward, q to quit: >
Cannot go forward.

Currently viewing www.cs.ualberta.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: <
Cannot go back.

Currently viewing www.cs.ualberta.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: =
URL: www.google.ca

Currently viewing www.google.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: <

Currently viewing www.cs.ualberta.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: >

Currently viewing www.google.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: =
URL: docs.python.org

Currently viewing docs.python.org
Enter = to enter a URL, < to go back, > to go forward, q to quit: <

Currently viewing www.google.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: <

Currently viewing www.cs.ualberta.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: =
URL: www.beartracks.ualberta.ca

Currently viewing www.beartracks.ualberta.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: >
Cannot go forward.

Currently viewing www.beartracks.ualberta.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: <

Currently viewing www.cs.ualberta.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: >

Currently viewing www.beartracks.ualberta.ca
Enter = to enter a URL, < to go back, > to go forward, q to quit: q
Browser closing...goodbye.
```

BROWSE-175

PROBLEM B

Goal: Use your knowledge of stacks to implement navigation functionality and the displaying of errors in a Graphical User Interface.

Overview:

You are tasked with completing the implementation of the navigation buttons for a Graphical User Interface (GUI) along with displaying error messages if an error should occur for the Web Browser Simulator using stacks which you created in Problem A.

You may NOT import any modules/libraries apart from os, time, and the stack class from stack.py

ANSI escape characters allow you to modify the output of your programs and format them in ways that you normally cannot. You can change styling such as font color, background color, formatting, displaying special characters, and more. They also allow for changing the position of the cursor and erasing lines. To reiterate, this simulator *will not actually work* (it is a simulator), but it should mimic the behavior of a real web browser, and now look closer to one as well!

IMPORTANT NOTE: When running the *browse175.py* file, Wing IDE may not correctly display these escape characters in the output when simply clicking "run". To see the proper output it is important to open your computer's terminal, navigate to the directory (by using `cd` or `chdir` depending on your operating system) where you saved *browse175.py*, and execute `python3 browse175.py`. You may also have to make the window larger or smaller.

Intro to ANSI Escape Characters:

For this lab, the escape characters have been implemented for you in functions which are already completed. Despite this, it is important to understand how they work. We can use ANSI characters within Control Sequence Introducers (CSIs) to display formatted content. CSIs all have the same format:

`\033[X;YZ`

Every sequence begins with `\033` (the "escape character") and is immediately followed by an opening square bracket. `X` and `Y` are numerical parameters to be passed (optional), separated by a semicolon. `Z` is the command to be executed.

For example, one basic use of ANSI characters: we can color text red! This is done by prefixing our text with the appropriate ANSI escape character. The output of `print("\033[31mHello world")` is `Hello world` and this is because the command `31m` (`Z` in the example above) corresponds to the color RED. Note the lack of parameters (`X` and `Y`) in the example here, they are optional. Further information about ANSI escape codes can be found in this Wikipedia article: https://en.wikipedia.org/wiki/ANSI_escape_code#3-bit_and_4-bit

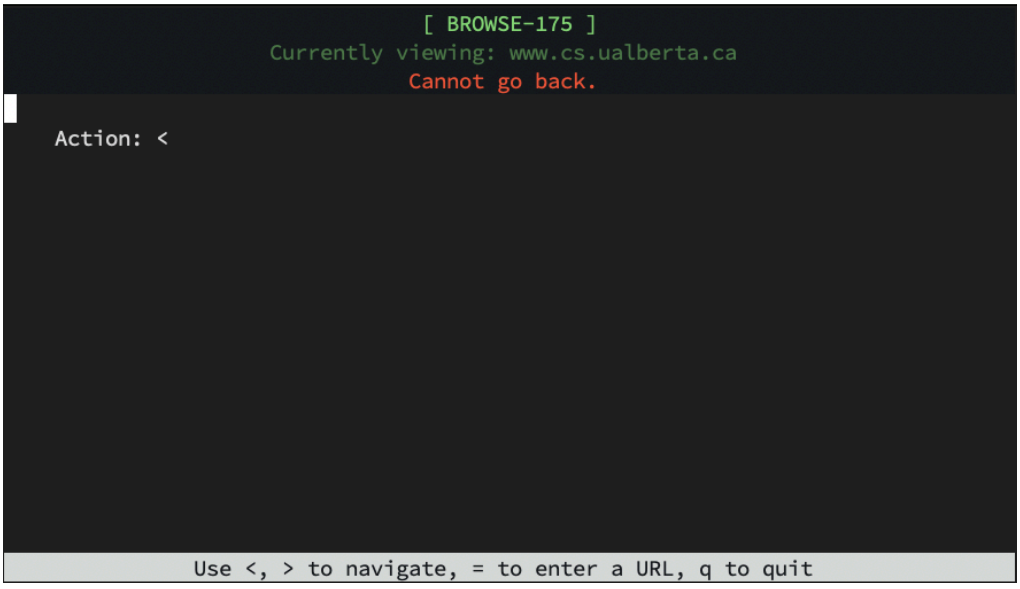
A colleague has implemented the following functions for you (*browse175.py*):

- ***print_location(x, y, text)***: This function will print the desired text at row (x) and column (y) in the GUI.
- ***clear_screen()***: This function will clear the screen to allow new contents to be displayed in the terminal.
- ***display_error(error)***: This function will display a specified error message beneath the header in the GUI.
- ***print_header()***: This function prints the BROWSE-175 header at the top of the GUI.
- ***move_cursor(x, y)***: This function moves the cursor to the row (x) and column (y) specified.
- ***display_current_site(current)***: This function displays the current site in the GUI.
- ***display_hint(message)***: This function displays the context menu ("<" for back, ">" for forward, "=" for new site, "q" for quit) in the GUI.

You should not change the code within these functions. Ensure you understand how each function works.

YOUR TASK:

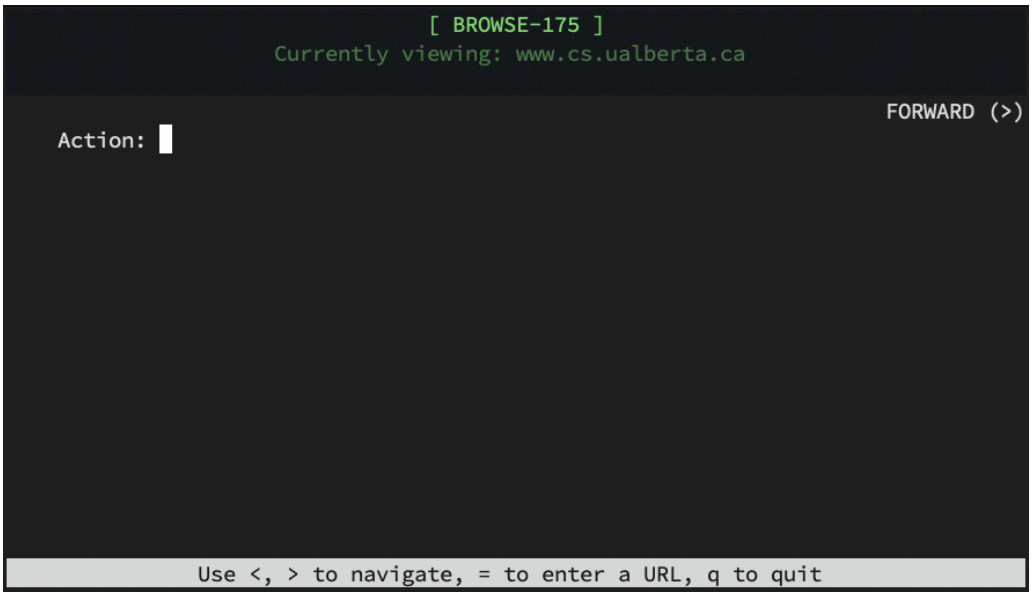
Part 1
Download and save a copy of *browse175.py* from eClass and put it in the same directory as your *stack.py* file from Problem A. Implement modified versions of *goBack()* and *goForward()* where indicated by the comments above *main()*. You can begin by copying these directly from *lab5_browser.py*. Also, copy *goToNewSite()* from *lab5_browser.py* (this one can remain the same). You must modify *goBack() and goForward()* such that **if the back or forward stacks are empty** (i.e. the user cannot go backward or forward) **an error message is displayed in the GUI**: "Cannot go back." or "Cannot go forward." (hint: your colleague has already implemented functions which you can call to help you). Your errors should look like this if implemented correctly:



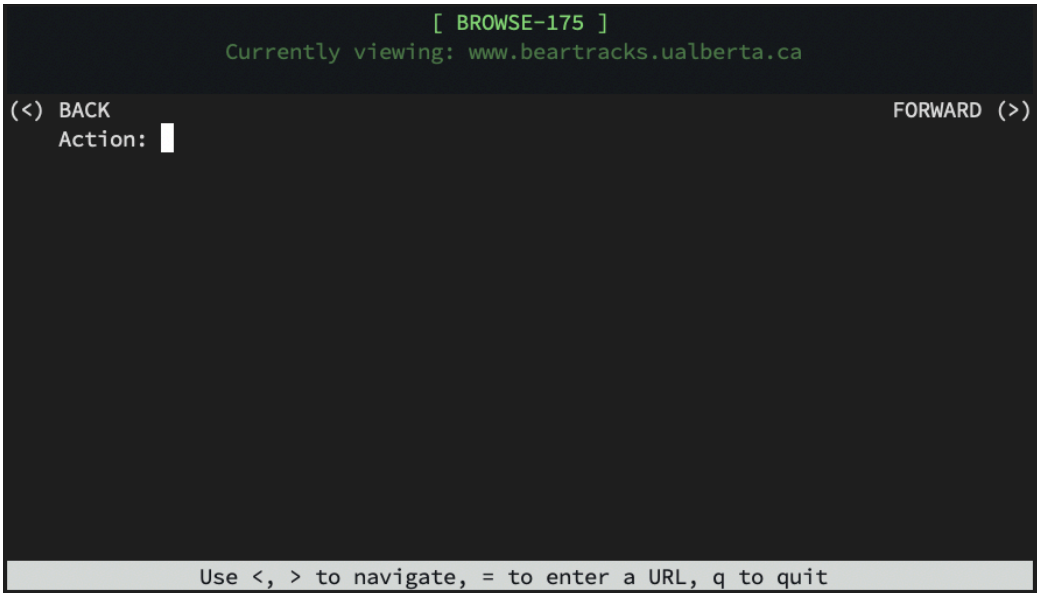
(in this case, the user entered the "<" action when no sites were in the back stack, hence the "Cannot go back." error message is displayed)

Part 2

Complete the implementation of the forward and back button text labels in the GUI. Your colleague has started this function and provided you the *display_buttons()* definition and docstring, but you must complete it so that **the "(<) BACK" and "FORWARD (>)" text labels are each displayed only if there are items in their respective back or forward stack**. To stay consistent with the code your colleague provided elsewhere, the entire linewidth is 80 characters. Note that the back and forward labels should be left and right aligned respectively. If a user enters something other than "<", ">", "=", or "q", your GUI should display the error message "Invalid action!". Your back/forward labels should look like this if implemented correctly:



An example with sites only in the forward stack (conversely, only "(<) BACK" would be displayed if sites are only in the back stack).



An example with sites in both the back and forward stacks.

Deliverables

You will produce and submit **three** files for this lab on eClass by the submission deadline:

- **stack.py**: Python solution to Part 1 of Problem A of this lab
- **lab5_browser.py**: Python solution to Part 2 of Problem A of this lab
- **browse175.py**: Python solution to Parts 1 & 2 of Problem B of this lab