Due No Due Date **Points** 0 **Available** after Oct 29 at 12am

Home (https://northeastern.instructure.com/courses/156930/pages/landing-page)

Resources (https://northeastern.instructure.com/courses/156930/pages/resources

Lab 8: Using Classes

Assignment Part 1

Sample solutions will be available on Friday at 9am: ***palindrome (https://northeastern.instructure.com/courses/156930/files/22372036/preview)*** ***(https://northeastern.instructure.com/courses/156930/files/22372036/download?download_frd=1)*** , ***postfix (https://northeastern.instructure.com/courses/156930/files/22371985/preview)*** ***(https://northeastern.instructure.com/courses/156930/files/22371985/download?download_frd=1)*** , ***Stack (https://northeastern.instructure.com/courses/156930/files/22371596/preview)*** ***(https://northeastern.instructure.com/courses/156930/files/22371596/download?download_frd=1)***

Stacks: Palindrome: The Encore!

Remember our palindrome solutions? We did the recursive version last week, and the non-recursive version a few weeks prior. For this part of today's lab, write a non-recursive function that uses a Stack (you can use the Stack class I shared in lecture, or write your own) to determine whether a given string is a palindrome or not. Your function can assume all-lowercase, all-alphabetic input. It should have the following specs:

- Name: is_pal_stack
- Parameter: string
- Returns: boolean indicating if the string is a palindrome

The following strings are palindromes:

'tacocat'

'radar'

'borroworrob'

'madamimadam'

'aa'

'a'

''

Assignment Part 2

Stacks: Postfix Calculator

When you write an arithmetic expression such as **`B * C`**, the form of the expression provides you with information so that you can interpret the statement. Typically, we use what is called infix notation like the example above. However, if we rewrite the expression in postfix notation, **`B C *`**, the operator comes AFTER the operands.

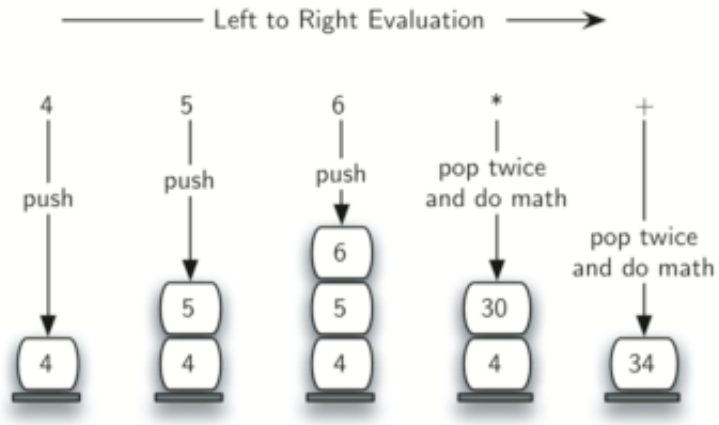
One way to think about the postfix is that whenever an operator is seen on the input, the two most recent operands will be used in the evaluation.

For this part of the lab, develop the code for a simple postfix calculator, using a Stack (which is really, really useful when doing postfix stuff). This exercise is focusing on stacks again, so we'll keep the calculator simple. Only concern yourself with addition, subtraction, multiplication and division. AND, limit your operands to single digit integers (assume good data input too).

Example:
Consider the postfix expression `4 5 6 * +`. As you scan the expression from left to right, you first encounter the operands 4 and 5. We are not sure what to do with them until we see the next symbol. Push each element on the stack to make them available for any operator that comes next.

In this example, the next symbol (6) is another operand. So, as before, push it and check the next symbol. Now we see an operator, *. This means that the two most recent operands need to be used in a multiplication operation. Pop the stack twice (to get the two operands), and then perform the multiplication (in this case getting the result 30).

You're not done yet! Take that result and push IT on the stack so that it can be used as an operand for the later operators in the expression. When the final operator is processed, there will be only one value left on the stack. Pop and return it as the result of the expression. The picture below shows the stack contents as this entire example expression is being processed.



Do this:
Write a function called `postfix_eval()` that takes one parameter: a string with the arguments in postfix form. Here's how we'd call it, and the expected results are commented to the right of each call. See if you can get your calculator working using a Stack to match the results below. Feel free to have other "helper functions" if you want.

```
print(postfix_eval('4 2 /')) # 2.0
print(postfix_eval('7 8 *')) # 56
print(postfix_eval('7 8 +')) # 15
print(postfix_eval('7 8 + 3 2 + /')) # 3.0
print(postfix_eval('7 8 + 3 4 2 + +')) # 9
print(postfix_eval('7 8 + 3 4 2 + + +')) # 24
```