Work with your neighbor. (This will be graded for participation only.)

1. Write a *function* `reverse(s)` that reverses the string `s` using a `Stack`. The function returns the reversed string.

   ANS:

```
def reverse(s):
    # get a Stack
    my_stack = Stack()

    # Push all of the characters in string
    # onto the stack
    for char in s:
        my_stack.push(char)

    rev_string = ""
    # pop all of the elements off of the stack
    # and concatenate onto a new string
    while not my_stack.is_empty():
        char = my_stack.pop()
        rev_string += char

    #return the reversed string
    return rev_string
```

2. Write a *function* `balanced(s)` that returns `True` if the string `s` is balanced with respect to the bracket characters `'['` and `']'` and `False` otherwise. Use a `Stack` in your implementation of this function.

ANS:

```
def balanced(s):

    my_stack = Stack()
    bal = True
    for c in s:
        if c == "[" :
            my_stack.push(c)
        elif c == "]":
            if my_stack.is_empty():
                bal = False
            else:
                my_stack.pop()

    return bal and my_stack.is_empty()
```

3. Below is an implementation of the `Stack` class:

   Change the implementations of the `push()` and `pop()` methods so that the top of the stack is at the beginning of the built-in Python list.

ANS:

```
class Stack:

    def __init__(self):
        self._items = []

    def push(self, item):
        ~~self._items.append(item)~~

        # the top of the stack will now be
        # at the beginning of the Python list

        self._items.insert(0,item)

    def pop(self):
        ~~return self._items.pop()~~

        # the top of the stack is the first item in the list
        # remove the first item in the list and return it
        return self._items.pop(0)
```

```
def is_empty():
    return self._items == []

def __str__(self):
    return str(self._items)
```