CSC148 - A client function for class Stack: size

```
class Stack:
    """A last-in-first-out (LIFO) stack of items.

    Stores data in a last-in, first-out order. When removing an item from the
    stack, the most recently-added item is the one that is removed.
    """
    # === Private Attributes ===
    # _items:
    #     The items stored in this stack. The end of the list represents the top of the stack.
    _items: List

    def __init__(self) -> None:
        """Initialize a new empty stack."""

    def is_empty(self) -> bool:
        """Return whether this stack contains no items.

        >>> s = Stack()
        >>> s.is_empty()
        True
        >>> s.push('hello')
        >>> s.is_empty()
        False
        """

    def push(self, item: Any) -> None:
        """Add a new element to the top of this stack.
        """

    def pop(self) -> Any:
        """Remove and return the element at the top of this stack.

        >>> s = Stack()
        >>> s.push('hello')
        >>> s.push('goodbye')
        >>> s.pop()
        'goodbye'
        """
```

We are writing client code and need a function (outside the class) to determine the number of items on a stack.

1. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.

    >>> s = Stack()
    >>> size(s)
    0
    >>> s.push('hi')
    >>> s.push('more')
    >>> s.push('stuff')
    >>> size(s)
    3
    """
    count = 0
    for _ in s:
        count += 1
    return count
```

stack is not iterable! so this won't work!

BAD solution

2. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    count = 0
    while not s.is_empty():
        s.pop()
        count += 1
    return count
```

removes elements

→ doctests do pass
→ BUT it destroys the stack! :(
BAD!

3. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    return len(s._items)
```

private attribute!
Client should not touch

If we change the implementation of the stack, _items might no longer be a list, or may not even exist! This function would crash.
If instead we respect the privacy of attributes and use only the public interface of our stack class, we get 'plug-out/plug-in' compatibility.

BAD!

4. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    s_copy = s
    count = 0
    while not s_copy.is_empty():
        s_copy.pop()
        count += 1
    return count
```

alias! same ID

We try here to solve the problem from Q2, BUT s_copy is an alias -- it refers to the same object as s, so any changes we make to it affect the original stack, so we're at the same issue as Solution 2 :(

BAD!

Hint: You're allowed to create new stacks

5. Given what you've learned, implement the function yourself on a separate sheet of paper.

see stack_size.py