# Lab 6: Mutability, Iterators **lab06.zip (lab06.zip)**

*Due by 11:59pm on Wednesday, October 4.*

## Starter Files

Download lab06.zip (lab06.zip). Inside the archive, you will find starter files for the questions in this lab, along with a copy of the Ok (ok) autograder.

# Required Questions

Getting Started Videos

## Mutability

Consult the drop-down if you need a refresher on mutability. It's okay to skip directly to the questions and refer back here should you get stuck.

Mutability

### Q1: WWPD: List-Mutation

> **Important:** For all WWPD questions, type `Function` if you believe the answer is `<function...>`, `Error` if it errors, and `Nothing` if nothing is displayed.

Use Ok to test your knowledge with the following "What Would Python Display?" questions:

```
python3 ok -q list-mutation -u
```

```
>>> s = [6, 7, 8]
>>> print(s.append(6))
------

>>> s
------

>>> s.insert(0, 9)
>>> s
------

>>> x = s.pop(1)
>>> s
------

>>> s.remove(x)
>>> s
------

>>> a, b = s, s[:]
>>> a is s
------

>>> b == s
------

>>> b is s
------

>>> s = [3]
>>> s.extend([4, 5])
>>> s
------

>>> s.extend([s.append(9), s.append(10)])
>>> s
------
```

## Q2: Insert Items

Write a function which takes in a list `s`, a value `before`, and a value `after`. It inserts `after` just after each value equal to `before` in `s`. It returns `s`.

**Important:** No new lists should be created or returned.

> **Note:** If the values passed into `before` and `after` are equal, make sure you're not creating an infinitely long list while iterating through it. If you find that your code is taking more than a few seconds to run, the function may be in an infinite loop of inserting new values.

```
def insert_items(s, before, after):
    """Insert after into s after each occurrence of before and then return s.

    >>> test_s = [1, 5, 8, 5, 2, 3]
    >>> new_s = insert_items(test_s, 5, 7)
    >>> new_s
    [1, 5, 7, 8, 5, 7, 2, 3]
    >>> test_s
    [1, 5, 7, 8, 5, 7, 2, 3]
    >>> new_s is test_s
    True
    >>> double_s = [1, 2, 1, 2, 3, 3]
    >>> double_s = insert_items(double_s, 3, 4)
    >>> double_s
    [1, 2, 1, 2, 3, 4, 3, 4]
    >>> large_s = [1, 4, 8]
    >>> large_s2 = insert_items(large_s, 4, 4)
    >>> large_s2
    [1, 4, 4, 8]
    >>> large_s3 = insert_items(large_s2, 4, 6)
    >>> large_s3
    [1, 4, 6, 4, 6, 8]
    >>> large_s3 is large_s
    True
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q insert_items
```

# Iterators

Consult the drop-down if you need a refresher on iterators. It's okay to skip directly to the questions and refer back here should you get stuck.

Iterators

## Q3: WWPD: Iterators

**Important:** Enter `StopIteration` if a `StopIteration` exception occurs, `Error` if you believe a different error occurs, and `Iterator` if the output is an iterator object.

Use Ok to test your knowledge with the following "What Would Python Display?" questions:

```
python3 ok -q iterators-wwpd -u
```

Python's built-in `map`, `filter`, and `zip` functions return **iterators**, not lists. These built-in functions are different from the `my_map` and `my_filter` functions we implemented in Lab 04 (/lab/sol-lab04/).

```
>>> s = [1, 2, 3, 4]
>>> t = iter(s)
>>> next(s)
______

>>> next(t)
______

>>> next(t)
______

>>> iter(s)
______

>>> next(iter(s))
______

>>> next(iter(s))
______

>>> u = t
>>> next(u)
______

>>> next(t)
______

>>> next(t)
______
```

```
>>> r = range(6)
>>> r_iter = iter(r)
>>> next(r_iter)
______

>>> [x + 1 for x in r]
______

>>> [x + 1 for x in r_iter]
______

>>> next(r_iter)
______
```

```
>>> map_iter = map(lambda x : x + 10, range(5))
>>> next(map_iter)
______

>>> next(map_iter)
______

>>> list(map_iter)
______

>>> for e in filter(lambda x : x % 4 == 0, range(1000, 1008)):
...     print(e)
______

>>> [x + y for x, y in zip([1, 2, 3], [4, 5, 6])]
______

>>> for e in zip([10, 9, 8], range(3)):
...   print(tuple(map(lambda x: x + 2, e)))
______
```

## Q4: Count Occurrences

Implement `count_occurrences`, which takes an iterator `t` and a value `x`. It returns the number of elements equal to `x` that appear in the first `n` elements of `t`.

**Important**: Call `next` on `t` exactly `n` times. Assume there are at least `n` elements in `t`.

> **Hint:** When the same iterator is passed into a function a second time, it should pick up where it left off after the first call, as with `s` from the doctest below.

```
def count_occurrences(t, n, x):
    """Return the number of times that x is equal to one of the
    first n elements of iterator t.

    >>> s = iter([10, 9, 10, 9, 9, 10, 8, 8, 8, 7])
    >>> count_occurrences(s, 10, 9)
    3
    >>> s2 = iter([10, 9, 10, 9, 9, 10, 8, 8, 8, 7])
    >>> count_occurrences(s2, 3, 10)
    2
    >>> s = iter([3, 2, 2, 2, 1, 2, 1, 4, 4, 5, 5, 5])
    >>> count_occurrences(s, 1, 3)  # Only iterate over 3
    1
    >>> count_occurrences(s, 3, 2)  # Only iterate over 2, 2, 2
    3
    >>> list(s)                     # Ensure that the iterator has advanced the right amou
    [1, 2, 1, 4, 4, 5, 5, 5]
    >>> s2 = iter([4, 1, 6, 6, 7, 7, 6, 6, 2, 2, 2, 5])
    >>> count_occurrences(s2, 6, 6)
    2
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q count_occurrences
```

# Q5: Repeated

Implement `repeated`, which takes in an iterator `t` and an integer `k` greater than 1. It returns the first value in `t` that appears `k` times in a row.

**Important:** Call `next` on `t` only the minimum number of times required. Assume that there is an element of `t` repeated at least `k` times in a row.

> **Hint**: If you are receiving a `StopIteration` exception, your `repeated` function is likely not identifying the correct value.

```
def repeated(t, k):
    """Return the first value in iterator t that appears k times in a row,
    calling next on t as few times as possible.

    >>> s = iter([10, 9, 10, 9, 9, 10, 8, 8, 8, 7])
    >>> repeated(s, 2)
    9
    >>> s2 = iter([10, 9, 10, 9, 9, 10, 8, 8, 8, 7])
    >>> repeated(s2, 3)
    8
    >>> s = iter([3, 2, 2, 2, 1, 2, 1, 4, 4, 5, 5, 5])
    >>> repeated(s, 3)
    2
    >>> repeated(s, 3)
    5
    >>> s2 = iter([4, 1, 6, 6, 7, 7, 8, 8, 2, 2, 2, 5])
    >>> repeated(s2, 3)
    2
    """
    assert k > 1
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q repeated
```

# Check Your Score Locally

You can locally check your score on each question of this assignment by running

```
python3 ok --score
```

**This does NOT submit the assignment!** When you are satisfied with your score, submit the assignment to Gradescope to receive credit for it.

# Submit

Make sure to submit this assignment by uploading any files you've edited **to the appropriate Gradescope assignment.** For a refresher on how to do this, refer to Lab 00 (https://cs61a.org/lab/lab00/#submit-with-gradescope).

# Optional Questions

> These questions are optional, but you must complete them in order to be checked off before the end of the lab period. They are also useful practice!

## Q6: Partial Reverse

When working with lists, it is often useful to reverse the list. For example, reversing the list `[1, 2, 3, 4, 5]` will give `[5, 4, 3, 2, 1]`. However, in some situations, it may be more useful to only partially reverse the list and keep some of its elements in the same order. For example, partially reversing the list `[1, 2, 3, 4, 5]` starting from index 2 until the end of the list will give `[1, 2, 5, 4, 3]`.

Implement the function `partial_reverse` which reverses a list starting from `start` until the end of the list. This reversal should be *in-place*, meaning that the original list is modified. Do not create a new list inside your function, even if you do not return it. The `partial_reverse` function returns `None`.

```
def partial_reverse(s, start):
    """Reverse part of a list in-place, starting with start up to the end of
    the list.

    >>> a = [1, 2, 3, 4, 5, 6, 7]
    >>> partial_reverse(a, 2)
    >>> a
    [1, 2, 7, 6, 5, 4, 3]
    >>> partial_reverse(a, 5)
    >>> a
    [1, 2, 7, 6, 5, 3, 4]
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q partial_reverse
```