Homework 6 Instructions

UCSD Extension CSE-41273, Spring 2019.

Included with this instruction file is the file `HW6.py`, containing the skeleton code for the homework.

Please read and follow all the directions carefully!

Put your name in the appropriate comment at the top of the program file `HW6.py`. Turn in a zipped file as described in the *About Homework* document.

Note: None of these exercises should print anything! Also, do not import anything other than from itertools. If you think you need to, email me to tell me what and why and I can tell you why not. And, as always, you can assume that you will get valid input.

Review Exercises

1. (10 pts) **squash:** Edit the function `squash`, that will take a list of lists and return a new list that is a squashed version of all elements - that is, it contains all elements together in one list. **Use a list comprehension**.

```
>>> from HW6 import squash
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
>>> flat = squash(matrix)
>>> flat
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
>>> matrix = [[1, 2], [3, 4, 5], [6], [7, 8, 9, 10]]
>>> squash(matrix)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

2. (10 pts) **special_nums:** Edit the function `special_nums` that returns a list of all the numbers from 1 through 300 that are divisible by both 6 and 10. **Use a list comprehension**. Make it work like this:

```
>>> from HW6 import special_nums
>>> list_result = special_nums()
>>> list_result
[30, 60, 90, 120, 150, 180, 210, 240, 270, 300]
```

3. (10 pts) **switch_dict:** Edit the function `switch_dict`, that returns a new dictionary with the dictionary's keys and values switched. **Use a dictionary comprehension**. Assume the input dictionary has values that are valid keys.

```
>>> from HW6 import switch_dict
>>> data = {'a': 2, 'b': 5, 'c': 6}
>>> switched = switch_dict(data)
>>> switched
{2: 'a', 5: 'b', 6: 'c'}
```

Generator & Iterator Exercises

4. (10 pts) **is_prime:** Re-write the `is_prime` function (given here) in one statement of code. Note: one *statement*, not "one line of code". You can assume the input will be larger than 2, so no need to worry about special cases. **Use a generator expression** within the function and one of the built-in functions from week 1, and *don't copy random stuff*

from the internet. **Note** that the function should *contain* a generator expression, not be a generator function.

No Sieve of Eratosthenes, or other unusual stuff. **We are not interested in efficiency**, the point is to figure out how to do it with the tools you have been given. Think about what the for loop and the test inside really mean in order for it to return `True`.

```
def is_prime(candidate):
    for n in range(2, candidate):
        if candidate % n == 0:
            return False
    return True
```

You can use these to test:

```
>>> from HW6 import is_prime
>>> is_prime(2)
True
>>> is_prime(3)
True
>>> is_prime(999979)
True
>>> is_prime(53)
True
>>> is_prime(345)
False
>>> is_prime(95)
False
>>> is_prime(51)
False
```

5. (10 pts) **number_9:** Create an inexhaustible (never-ending) generator `number_9` that always returns the number 9. **Absolute Requirement: Use something from itertools**. This is a test of your ability to read and understand documentation - of course this function is pointless in the real world, so don't try to figure a reason anyone might actually *want* to use this. :-)

```
>>> from HW6 import number_9
>>> number = number_9()
>>> next(number) == 9
True
>>> next(number) == 9
True
>>> next(number)
9
>>> next(number)
9
>>> next(number)
9
>>> next(number)
9
>>> next(number)
9
>>> iter(number) is number
True
```

6. (20 pts) **reverse_iter:** Write a *generator function* `reverse_iter` that accepts an iterable sequence and yields the items in reverse order. Think about what we learned in week 3, and also what we learned in week 2 about how to reverse a sequence. Note that after using `reverse_iter`, the original sequence should be unchanged.

Don't use built-in functions or methods like `reverse()`, `reversed()`, `iter()`. **Please note that this should also work when the input is a tuple (eg, `nums = (1, 2, 3, 4)` in the example below), because tuples are sequences.**

```
>>> from HW6 import reverse_iter
>>> nums = [8, 3, 6]
>>> it = reverse_iter(nums)
>>> next(it) == 6
True
>>> next(it)
3
>>> next(it)
8
>>> next(it)
Traceback (most recent call last):
  [...]
StopIteration
>>> nums
[8, 3, 6]
>>> items = ['a', 'b', 'c']
>>> it = reverse_iter(items)
>>> iter(it) is it
True
>>> next(it)
'c'
>>> next(it)
'b'
>>> next(it)
'a'
>>> next(it)
Traceback (most recent call last):
  [...]
StopIteration
```

7. (30 pts) **ReverseIter class:** Create an *iterator* class `ReverseIter` that takes an iterable sequence and iterates it in the reverse order. (When done, you will appreciate generator functions from problem 6). Note that after using `ReverseIter`, the original sequence should be unchanged. Creating a `ReverseIter` instance returns an iterator of the given sequence, which will return the elements of the sequence in reverse order, raising a StopIteration error when the reversed sequence is exhausted.

Don't use built-in functions or methods like `reverse()`, `reversed()`, `iter()`. You may use the `len()` function. **Please note that this should also work when the input is a tuple (eg, `nums = (1, 2, 3, 4)` in the example below).**

```
>>> from HW6 import ReverseIter
>>> nums = [8, 3, 6]
>>> it = ReverseIter(nums)
>>> iter(it) is it
True
>>> next(it) == 6
True
>>> next(it)
3
>>> next(it)
8
>>> next(it)
Traceback (most recent call last):
  [...]
StopIteration
>>> nums
[8, 3, 6]
>>> items = ['a', 'b', 'c']
>>> it = ReverseIter(items)
>>> next(it) == 'c'
True
>>> next(it)
'b'
>>> next(it)
'a'
>>> next(it)
```

```
Traceback (most recent call last):
  [...]
StopIteration
>>> items
['a', 'b', 'c']
```