Homework 6 Instructions

UCSD Extension CSE-41273, Summer 2022.

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, you should not need to import anything other than from `itertools`. If you think you need to, email me to tell me what and why so I can tell you why not. And, as always, you can assume that you will get valid input, so you do not need to have any try/except blocks or other validation code.

Review Exercises

Exercise 1: separate (10 pts)

Edit the function `separate`, that takes a string as input and returns a list containing the individual characters of the string. Return all the characters lowercased and do not remove duplicates. Add an optional keyword argument `sort`, with a default of `False`. If `sort` is `True`, will return the characters in "ASCII-betical" sorted order. Namely, sorted by their ASCII character definitions, which is pretty much what you would expect.

Note: You have to make your own argument list for this one! Do not use `*args` and/or `**kwargs`!

Hints: Remember that strings are *iterables*. Consider the built-in function `sorted`, or the list method `sort`.

```
>>> from HW6 import separate
>>> separate("hello")
['h', 'e', 'l', 'l', 'o']
>>> separate("hello", sort=True)
['e', 'h', 'l', 'l', 'o']
>>> separate("hello", sort=False)
['h', 'e', 'l', 'l', 'o']
>>> separate("A LONGER string")
['a', ' ', 'l', 'o', 'n', 'g', 'e', 'r', ' ', 's', 't', 'r', 'i', 'n', 'g']
>>> separate("A LONGER string", sort=True)
[' ', ' ', 'a', 'e', 'g', 'g', 'i', 'l', 'n', 'n', 'o', 'r', 'r', 's', 't']
```

Exercise 2: number_non_vowels (10 pts)

Edit the function `number_non_vowels` so that it returns the **number** of *characters* of the input string that are **not vowels.** Ignore the case of the letters in the string. Note that we do not count `y` as a vowel; vowels are `'aeiou'`.

Do not count any whitespace as a non-vowel!

Do not loop and count letter by letter! That is a non-Pythonic way. We are using Python; it has many tools for doing something like this! Your answer should use some form of comprehension.

Remember that Python has the `in` operator, and we can also use `not in`:


```
>>> 'd' in 'howdy'
True
>>> 'a' in 'howdy'
False
>>> 'a' not in 'howdy'
True
```

The function should work like this:

```
>>> from HW6 import number_non_vowels
>>> string = 'How many characters are not vowels?'
>>> number_non_vowels(string)
20
>>> string = '9 non-vowels!'
>>> number_non_vowels(string)
9
```

Generator & Iterator Exercises

Exercise 3: special_nums (10 pts)

Edit the function `special_nums` so that it returns a **generator** that *yields*, in order, the numbers from 1 through (and including) 450 that are divisible by both 6 and 10. **Use a generator expression or `yield`. Do not use built-in `iter()` function**

```
>>> from HW6 import special_nums
>>> nums = special_nums()
>>> iter(nums) == nums
True
>>> next(nums) == 30
True
>>> next(nums) == 60
True
>>> next(nums)
90
>>> next(nums)
120
>>> next(nums)
150
>>> next(nums)
180
>>> next(nums)
210
>>> next(nums)
240
>>> next(nums)
270
>>> next(nums)
300
>>> next(nums)
330
>>> next(nums)
360
>>> next(nums)
390
>>> next(nums)
420
>>> next(nums)
450
>>> next(nums)
Traceback (most recent call last):
  [...]
```

```
StopIteration
```

Exercise 4: evens (10 pts)

Edit the function `evens` so that it returns a **generator** that *yields* the numbers in the input sequence that are even, in order.

Do not use built-in `iter()` function and do not create any new lists.

It should work like this:

```
>>> from HW6 import evens
>>> numbers = [89, 32, 4, 35, 22, 8, 14, 3]
>>> even_numbers = evens(numbers)
>>> even_numbers
<generator object evens.<locals>.<genexpr> at 0x10d2c29d0>
>>> iter(even_numbers) == even_numbers
True
>>> next(even_numbers) == 32
True
>>> next(even_numbers) == 4
True
>>> next(even_numbers) == 22
True
>>> next(even_numbers)
8
>>> next(even_numbers)
14
>>> next(even_numbers)
Traceback (most recent call last):
  [...]
StopIteration
```

Exercise 5: continuous_nums (10 pts)

Edit the function `continuous_nums` so that it returns an inexhaustible (never-ending) **generator/iterator** that *returns* the numbers `1` through `num` forever. **Do not use `iter()`. If you need it, you're doing something wrong.**

Absolute Requirement: Use something from itertools. Note the types of the things in itertools. This is a test of your ability to read and understand documentation. ;-)

It should work like this:

```
>>> from HW6 import continuous_nums
>>> nums = continuous_nums(2)
>>> next(nums) == 1
True
>>> next(nums) == 2
True
>>> next(nums) == 1
True
>>> next(nums)
2
>>> next(nums)
1
>>>
>>> nums = continuous_nums(5)
>>> next(nums)
1
>>> next(nums)
2
>>> next(nums)
```

```
3
>>> next(nums)
4
>>> next(nums)
5
>>> next(nums)
1
>>> next(nums)
2
>>> next(nums)
3
>>> next(nums)
4
>>> next(nums)
5
>>> next(nums)
1
>>> iter(nums) == nums
True
```

Exercise 6: reverse_iter (20 pts)

Edit the function `reverse_iter` so that it returns a **generator** that *yields* the items given in the input sequence 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.

Do not create any new lists.

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, if `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
```

Exercise 7: ReverseIter class (30 pts)

Create an *iterator* class `ReverseIter` that takes an input sequence and creates an instance that is an *iterator* that iterates over the input sequence in reverse order. (When done, you will appreciate generator functions as in problem 6).

Note that after using `ReverseIter`, the original sequence should be unchanged. A `ReverseIter` instance returns an iterator of the input sequence, which will return the elements of the sequence in reverse order, raising a StopIteration error when the reversed sequence is exhausted.

Remember that this is an *iterator*, and iterators are single-use *iterables*. When the instance is exhausted (finished iterating over the sequence), you need to raise your own `StopIteration` Error when there are no more items. Subsequent calls to next() should continue to generate a `StopIteration` Error, because iterators are single-use iterables. There should be no try/except blocks in the class definition.

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, if `nums = (1, 2, 3, 4)` in the example below).

The class should not print anything!

```
>>> 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
>>> iter(it) == it
True
>>> next(it)
'b'
>>> next(it)
'a'
>>> next(it)
Traceback (most recent call last):
  [...]
StopIteration
>>> items
['a', 'b', 'c']
>>> next(it)  # Make sure it stays finished.
Traceback (most recent call last):
  [...]
StopIteration
```

Note. Remember that from the point of view of the code using them, the behavior of a generator and an iterator appear the same. They are both single-use iterators; the difference is internal to the code.

My email is dianechen.ucsdext@gmail.com. Please do not hesitate to email me if you have questions.