Homework 2 Instructions

UCSD Extension CSE-41273, Spring 2019.

The zip download file has, besides this file, the file `HW2.py` with skeleton functions.

Please read and follow all the directions carefully! Read everything before coding! Please review the *About Homework* document for reminders and pointers. Put your name in the appropriate comment at the top of the program file `HW2.py`. When you are ready to turn in the file, use a zip utility to compress the file and *rename the zipped file* to `HW2_your_name.zip`. Be sure to run `flake8` on your code before you turn it in.

As I have said before, **your best chance of a good grade is to follow directions**. If the directions say to use a slice, list comprehension or dictionary comprehension, you will lose points if you do not follow these directions. If you aren't sure, please review the lecture materials and contact me if you need help. Except for where I tell you to look up useful methods, **everything you need to know is in the lessons or the supplemental required reading.**

Be sure to contact me if you have any questions about the functions and how they are supposed to work.

Grading: Point values as shown. 100 points total.

Function 1: combine_lists (10 points)

Edit the function `combine_lists` so that it takes two lists (sequences) as input and returns a new list containing all elements from both lists; the elements of the first list, followed by the elements of the second list. **The original lists should not be modified.** No "for loops"!

It should work like this:

```
>>> first = [1, 2, 3]
>>> second = [4, 5, 6]
>>> new = combine_lists(first, second)
>>> new
[1, 2, 3, 4, 5, 6]
>>> first
[1, 2, 3]
>>> second
[4, 5, 6]
>>> third = [7, 8, 9, 10]
>>> new = combine_lists(first, third)
>>> new
[1, 2, 3, 7, 8, 9, 10]
>>> fruits = ['apples', 'grapes', 'peaches', 'apricots', 'bananas']
>>> new = combine_lists(first, fruits)
>>> new
[1, 2, 3, 'apples', 'grapes', 'peaches', 'apricots', 'bananas']
```

Function 2: last_n_elements (10 points)

Edit the function `last_n_elements` so that it **uses a slice** to return the last n items of a sequence. The original sequence should not be modified. Do not use `len()` in your slice. If you feel you need to use `len()`, please review the lecture about slices.

It should work like this:

```
>>> fruits = ['apples', 'grapes', 'peaches', 'apricots', 'bananas']
>>> last = last_n_elements(fruits, 3)
>>> last
['peaches', 'apricots', 'bananas']
>>> last_n_elements(fruits, 1)
['bananas']
>>> numbers = [41, 25, 54, 15, 76, 68, 32, 38]
>>> last_n_elements(numbers, 4)
[76, 68, 32, 38]
```

Function 3: last_n_reversed (10 points)

Edit the function `last_n_reversed` so that it **uses a slice** to return the last n items of a sequence, in reversed order. The original sequence should not be modified. Do not use `len()` in your slice. If you are unsure how to get started, please review the lecture about slices. **Do not use the builtin `reversed` function or the sequence method `reverse()`.**

It should work like this:

```
>>> fruits = ['apples', 'grapes', 'peaches', 'apricots', 'bananas']
>>> last = last_n_reversed(fruits, 3)
>>> last
['bananas', 'apricots', 'peaches']
>>> last_n_reversed(fruits, 1)
['bananas']
>>> numbers = [41, 25, 54, 15, 76, 68, 32, 38]
>>> last_n_reversed(numbers, 4)
[38, 32, 68, 76]
```

Function 4: power_list (10 points)

Edit the function `power_list` so that it accepts a list of numbers and returns a new list that contains each number raised to the i-th power where i is the index of that number in the given list. **Use a list comprehension.** No "for loops"!

It should work like this:

```
>>> power_list([2, 2, 2, 2, 2, 2])
[1, 2, 4, 8, 16, 32]
>>> new_list = power_list([9, 6, 5, 4])
>>> new_list
[1, 6, 25, 64]
>>> numbers = [33, 700, 82.25, 16, 2, 3, 9.5]
>>> power_list(numbers)
[1, 700, 6765.0625, 4096, 16, 243, 735091.890625]
>>> power_list([2.5, 2.5, 2.5, 2.5, 2.5, 2.5])
[1.0, 2.5, 6.25, 15.625, 39.0625, 97.65625]
```

Function 5: rotate_list (20 points)

Edit the function `rotate_list` so that it removes the first item from a given list, adds it to the end of the list, and returns the item that was moved. This function is a little different than most as it *mutates the input list*. No "for loops"!

Hint: Look through the documentation on sequence types (ie, lists) for methods that would be useful. **Do not use `del`.**

It should work like this:

```
>>> numbers = [1, 2, 3, 4]
>>> item = rotate_list(numbers)
>>> item
1
>>> numbers
[2, 3, 4, 1]
>>> rotate_list(numbers)
2
>>> numbers
[3, 4, 1, 2]
>>> fruits = ['apples', 'grapes', 'peaches', 'apricots', 'bananas']
>>> rotate_list(fruits)
'apples'
>>> fruits
['grapes', 'peaches', 'apricots', 'bananas', 'apples']
```

Function 6: matrix_fill (20 points)

Edit the function `matrix_fill` so that it takes 2 numbers: the number of rows and the number of columns. It returns a matrix (list of lists) where the elements are numbered sequentially. **Use a list comprehension with arithmetic.** No "for loops"!

Do not make any temporary lists of any kind.

It should work like this:

```
>>> my_matrix = matrix_fill(3, 4)
>>> my_matrix
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
>>> matrix_fill(5, 3)
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15]]
>>> matrix_fill(2, 2)
[[1, 2], [3, 4]]
>>> matrix_fill(2, 1)
[[1], [2]]
>>> matrix_fill(1, 1)
[[1]]
>>> matrix_fill(1, 2)
[[1, 2]]
```

Function 7: get_word_codes (20 points)

Edit the function `get_word_codes` so that it accepts a *list* of strings and returns a dictionary containing the strings as keys and a list of corresponding character codes as values. It should work on any strings. **Use a dictionary comprehension.** No "for loops"! Do not make any temporary lists of any kind.

Hint: Look through Python's built-in functions for a useful function to use for this.

It should work like this:

```
>>> words = ['yes', 'no']
>>> codes = get_word_codes(words)
>>> codes
{'yes': [121, 101, 115], 'no': [110, 111]}
>>> words = ['hello', 'python', 'bye']
>>> codes = get_word_codes(words)
>>> codes
{'hello': [104, 101, 108, 108, 111], 'python': [112, 121, 116, 104, 111, 110], 'bye': [98, 121,
101]}
codes = get_word_codes(['Python is fun!'])
```

```
>>> codes
{'Python is fun!': [80, 121, 116, 104, 111, 110, 32, 105, 115, 32, 102, 117, 110, 33]}
```

Notes

A student asked:

When you say no for loops, does that also exclude list comprehensions?

That is a good question. For the homework problems where I specifically say "No for loops" (1 & 5), **yes**, that includes list comprehensions, as they are not needed for those problems.

For problem 1, think about what kinds of operations can be performed on lists; how can we make use of them?

For problem 5, look carefully at the documentation linked in the homework instructions. It's there for a reason.