Homework 9

Total Points:

100 (Correctness and Style) + 3 EC (Checkpoint) + 5 EC (Question 6)

Due Dates (SD time):

- **Entire Assignment: Tuesday, Mar 12, 11:59pm**
- **Checkpoint (Read below): Sunday, Mar 10, 11:59pm**

Starter Files

Download hw09.zip. Inside the archive, you will find starter files for the questions of this homework. You cannot import anything to solve the problems.

IMPORTANT: Coding and Docstring Style

This is a reminder that your code style will be graded. Here are a few useful links:

Style Guide Document

Style Guide on Course Website

Style Guide Examples

Testing

At any point of the homework, use the following command to test your work:

```
>>> python3 -m doctest hw09.py
>>> py -m doctest hw09.py
>>> python -m doctest hw09.py
```

Also you can call one function at a time using:

```
>>> python3 -i hw09.py
```

Checkpoint Submission

Due Date: Sunday, March 10th, 11:59pm (SD time)

You can earn up to **3 points extra credit** by submitting the checkpoint by the due date above. In the checkpoint submission, you should complete **Question 1, 2** and submit the hw09.py file to gradescope.

Checkpoint submission is graded by completion, which means you can get full points if your code can pass some simple sanity check (no tests against edge cases). Note that in your final submission, you should still submit these questions, and you may modify your implementation if you notice any errors.

General Notes and Requirements

1. **DO NOT IMPORT ANY PACKAGES.**
2. Please add your own doctests (**at least three**) for Questions 3, 4, 5 as the given doctests are not sufficient. **You will be graded on the doctests**. You do not need docstrings or doctests for questions 1, 2, 6.

Required Questions

Intro:

When we think about the *efficiency* of an algorithm, we should be concerned not only with how time-efficient it is but also with how much *space* is needed for the data during the execution.
For example, suppose we want to reverse a list (get items in the reversed order).
This is one way we can do it:

```
def rev(lst):
    temp = []
    for i in lst:
        temp = [i] + temp
    lst = temp
    return lst
```

In order to solve this problem, I used **another** *temp* list and its size is exactly the same as the length of `lst`. Since we had to use **non-constant** extra space to solve the problem, we say that the algorithm is **not** in-place.

In-place algorithm is

- Algorithm that uses a small **constant** amount of extra space in addition to the original input.
- Usually overwrites (changes) the input space during the program execution.

In the example given above, we can reverse a list in-place by utilizing swapping, rather than creating a new list. For instance, using

```
lst[0], lst[1] = lst[1], lst[0]
```

allows us to swap elements in the list **without** needing to create a new list. Think about what pairs of elements you would need to swap in order to reverse an entire list.

Some algorithms might have several solutions with the same time efficiency, but with a different *space* efficiency.

Question 1:

As we know from the lecture, some operations can mutate the content of the object and some return a copy. This question will ask you to investigate different methods and return True if a method *mutates* an object and False otherwise. Please do not google the answers right away but learn how to test it first. Then you can check your answers.

1) Sometimes you can tell just by looking at the context of the object after a method was applied.

2) Another way it can be tested is by looking at the address of **the object(s)** BEFORE an operation was performed and after. It can be done using a few built-in methods together.

3) Use Python Tutor and look at the reference(s) being pointed at.

Example:

```
hex(id(variable_name))
```

- `id()`: function returns a unique id for the specified object.
- `hex()`: function converts an integer to the corresponding hexadecimal string.

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [1, 2, 3]
>>> hex(id(lst1))
'0x10a475940'   # memory address for lst1
>>> hex(id(lst2))
'0x10a4759c0'   # memory address for lst2. Different.


>>> lst1.append([4, 5, 6])
>>> print(lst1)
[1, 2, 3, [4, 5, 6]]        # lst1 has been updated
>>> hex(id(lst1))
'0x10a475940'               # lst1 has the same address after append.
                            # append mutates the list
```

Lists of functions you need to investigate:

1. `str1.lower()`
2. `set1.update({5,6})`
3. `set1 - set2`
4. `set_d.union(set_a)`
5. `lst.sort()`
6. `sorted(lst)`
7. `lst[::-1]`
8. `lst.reverse()`
9. `list(iterable)`
10. `dict1.update(dict2)`

Just like with the complexity question and class question, add your answers (True/False) to the list and return it.

```
def q1_tf_answers():
    """
    >>> q1_answers = q1_tf_answers()
    >>> len(q1_answers) == 10
    True
    >>> all([isinstance(ans, bool) for ans in q1_answers])
    True
```

```
    """
```

Question 2:

This question is similar to the previous one but this time you need to investigate whether a given method is *in-place* or not. Please follow ***this link*** to find these functions. Add your answers to the list: True if the algorithm is *in place* and False otherwise.

```
def q2_tf_answers():
    """
    >>> q2_answers = q2_tf_answers()
    >>> len(q2_answers) == 5
    True
    >>> all([isinstance(ans, bool) for ans in q2_answers])
    True
    """
```

Question 3:

Once again, lists are mutable objects. It means that one can directly *mutate* the contents of a list. Write a function that will reverse the list parameter WITHOUT creating a new list. That is, reverse all the items in the list within the list itself (*in-place*).

Notes:

1. Do not use `lst[::-1]` to do it (why?);
2. Think of a way to manipulate a given list without creating another list. You should not have a return statement, because your function will *mutate* a given list.

Requirements:

1. Your function should mutate the list (without creating another list), and return None.
2. You **can't** use any built-in methods (.append() not allowed). You can use len, range if needed.
3. No assert statements needed.

```
def reverse_list(lst):
    """
```

```
    >>> x = [3, 2, 4, 5, 1]
    >>> reverse_list(x)
    >>> x
    [1, 5, 4, 2, 3]
    >>> x = []
    >>> reverse_list(x)
    >>> x
    []
    >>> x = [1]
    >>> reverse_list(x)
    >>> x
    [1]
    """
```

Question 4:

Write a function `swap_lists` that takes in two lists and swaps their elements **in place**. Assume the two input lists will have the same size.

Requirements:

1. You **can't** use any built-in methods (.append() not allowed). You can use len, range if needed.
2. You **can't** use another list. You **can not** return lists.
3. Your function should mutate the lists (without creating a new list), and return None.
4. No assert statements needed.

```
def swap_lists(list_1, list_2):
    """
    >>> list_1 = [1, 2]
    >>> list_2 = [3, 4]
    >>> swap_lists(list_1, list_2)
    >>> print(list_1)
    [3, 4]
    >>> print(list_2)
    [1, 2]

    >>> list_3 = [4, 2, 6, 8, 90, 45]
    >>> list_4 = [30, 41, 65, 43, 4, 17]
    >>> swap_lists(list_3, list_4)
```

```
    >>> print(list_3)
    [30, 41, 65, 43, 4, 17]
    >>> print(list_4)
    [4, 2, 6, 8, 90, 45]
    """
```

Question 5:

While journeying through Pythonia, you arrive at the edge of Pythonia called the Iridescent Verge where the treasure and teleportation portal is located. It has been an exciting and fulfilling journey, but now it's time to go home. However, there's a password required to use the portal. You find the hint being carved down on the wall.

Write a function that takes in a list consisting of strings representing the password hint and decodes the hint by taking the first character of each string and finding the corresponding number on the keyboard (the password is case insensitive). Your function should decode by mutating on the input list in-place.

Assumptions:

1. You can assume the characters will only be alphabets or symbols.
2. Decode all symbols by pressing a 0 on the keyboard.

Requirements:

1. You can use len, .lower() and range if needed.
    - **Hint**: You can check how .ord() can be used
2. You are not allowed to create a new dictionary or list.
3. Your function should mutate the list (without creating another list), and return None.
4. No assert statement needed.

```
def decode_password(hint):
    """
    >>> hint = ["Pythonia", "is", "an", "excellent", "Place"]
    >>> decode_password(hint)
    >>> print(hint)
    [6, 3, 1, 2, 6]
```

```
    >>> hint = ["great", "to", "SEE", "U", "!", "Goodbye", "."]
    >>> decode_password(hint)
    >>> print(hint)
    [3, 7, 7, 7, 0, 3, 0]
    """
```

Question 6: (Extra Credit)

While journeying through Pythonia, you stumble upon the Wish River, a magical waterway adorned with wish stones. Each stone carries a wish for a specific amount of gold coins needed. The order of the wishes was crucial - the closer to the source of the river, the sooner the wish would come true. Beware the mischievous Goblin scattering fake stones, disrupting the wish order. Join the quest to restore order and outsmart the playful troublemaker!

Write a function that will take in a list of non-negative integers. Each integer represents the amount of gold coins the wisher needs. The integer 0 represents the fake stone. Your function should move all the zeros to the end of the list, while maintaining the order of the non-zero integers **WITHOUT** creating a new list. That is, your function should manipulate the elements within the list itself (*in-place*).

Requirements:

1. You **can't** use any built-in methods (.append() not allowed). You can use len, and range if needed.
2. Your function should mutate the list (without creating another list, i.e. it is in-place), and return None.
3. No assert statements needed.
4. You can create integer objects to keep track of the positions if needed.
5. No need for doctests (ignore the docstrings in starter code); but proper style is encouraged!

```
def move_non_wishes(seq):
    """
    >>> lst1 = [0, 1, 0, 3, 12]
    >>> move_non_wishes(lst1)
    >>> lst1
    [1, 3, 12, 0, 0]
    >>> lst2 = [0, 0, 0]
    >>> move_non_wishes(lst2)
    >>> lst2
    [0, 0, 0]
```

```
>>> lst3 = []
>>> move_non_wishes(lst3)
>>> lst3
[]
>>> lst4 = [1, 2, 3]
>>> move_non_wishes(lst4)
>>> lst4
[1, 2, 3]
"""
```

It was nice to have you with us in Pythonia!

The portal brought you back home :)

Submission

Please submit the homework via Gradescope. You may submit more than once before the deadline, and only the final submission will be graded. Please refer to the Gradescope FAQ if you have any issues with your submission.