Homework 06

Re-submit Assignment

Due Mar 4 by 10pm **Points** 100 **Submitting** a text entry box or a file upload

Assignment Description

Part 1: `list_copy(l: List[Any]) -> List[Any]`

Write a function `list_copy(l)` that takes a list as a parameter and returns a copy of the list **using a list comprehension**. Writing list comprehensions can be a little challenging in the beginning. If you're confused by list comprehensions, first write your list_copy() function without a list comprehension, then once that version is working, see the lecture for instructions on how to convert your solution to use a list comprehension.

Note: you **must** use a list comprehension.

Hint: my solution has only one statement.

Part 2: `list_intersect(l1: List[Any], l2: List[Any]) -> List[Any]`

Write a function `list_intersect(l1, l2)` that takes two lists as parameters and returns a new list with the values that are included in both lists **using a list comprehension**.

E.g.

```
list_intersect([1, 2, 3], [1, 2, 4]) == [1, 2]
```

```
list_intersect([1, 2, 3], [4, 5, 6]) == []
```

Note: you **must** use a list comprehension.

Part 3: `list_difference(l1: List[Any], l2: List[Any]) -> List[Any]`

Write a function `list_difference(l1, l2)` that takes two lists as parameters and returns a new list with the values that are in l1, but NOT in l2. Your solution must **use a list comprehension**.

E.g.

`list_difference([1, 2, 3], [1, 2, 4]) == [3]`

`list_difference([1, 2, 3], [4, 5, 6]) == [1, 2, 3]`

Note: you must use a list comprehension.

Hint: my solution for part 3 is VERY similar to my solution for part 2.

Part 4: `remove_vowels(string: str) -> str`

Write a function `remove_vowels(string)` that given a string, splits the string on whitespace into words and returns a new string that includes only the words that do **NOT** begin with vowels. E.g.

```
remove_vowels("Amy is my favorite daughter") == "my favorite daughter"
remove_vowels('Jan is my best friend') == "Jan my best friend"
```

Note: you **must** use a list comprehension.

Hints:

- Your `remove_vowels(string)` function can be written in one line.
- Be careful where you call string.lower() to insure that capital letters are preserved in the return value

Part 5: Password checker `check_pwd(password: str) -> bool`

Write a function `check_pwd(password: str) -> bool` that takes a **string** as a parameter and returns a **boolean** value.

`check_pwd(password)` returns True if all conditions below are satisfied:

- The password includes at least two upper case characters
- The password includes at least one lower case characters
- The password **starts with** at least one digit

`check_pwd(password)` returns False if any of the three conditions is NOT satisfied.

Note: you **must** use list comprehensions to test for at least two upper case and one lower case characters.

Hints:

- `'a'.islower()` is True, `'A'.isupper()` is True, `'1'.isdigit()` is True.

Part 6: The DonutQueue

Define a ***DonutQueue*** class that tracks customers as they arrive at the donut shop. Customers are added to the queue so they can be served in the order they arrived with the exception that **priority** customers are served before non-priority customers. Priority customers are served in the order they arrive, but before any non-priority customers.

Your ***DonutQueue*** class should support the following methods:

- `arrive(self, name: str, vip: bool) -> None` to note that a customer arrived, where:
  - ***name*** is the customer's name
  - ***vip*** is True if the customer is a priority customer or False
- `next_customer(self) -> Optional[str]` returns the ***name*** of the next customer to be served where all priority customers are served in the order they arrived before any non-priority customer. `next_customer(self)` should return ***None*** if there are no customers waiting
- `waiting(self) -> Optional[str]` returns a comma separated string with the names of the customers waiting in the order they will be served or None if there are no customers waiting.

The following automated tests may help you to understand the requirements:

```
class DonutQueueTest(unittest.TestCase):
    def test_queue(self):
        dq = DonutQueue()
        self.assertIsNone(dq.next_customer())
        dq.arrive("Sujit", False)
        dq.arrive("Fei", False)
        dq.arrive("Prof JR", True)
        self.assertEqual(dq.waiting(), "Prof JR, Sujit, Fei")
        dq.arrive("Nanda", True)
        self.assertEqual(dq.waiting(), "Prof JR, Nanda, Sujit, Fei")
        self.assertEqual(dq.next_customer(), "Prof JR")
        self.assertEqual(dq.next_customer(), "Nanda")
        self.assertEqual(dq.next_customer(), "Sujit")
        self.assertEqual(dq.waiting(), "Fei")
        self.assertEqual(dq.next_customer(), "Fei")
        self.assertIsNone(dq.next_customer())
```

Hints:

- A queue (or two) is a good match for this problem. See the slides for details about implementing a queue.
- Your solution **MUST** define at least one class
- This question was one of the problems on the final exam last semester. You should be comfortable writing programs like this by the end of the semester.

Optional Practice Problem:

Write a function `reorder(l: List[Any]) -> List[Any]` that returns a **copy** of the argument sorted using a list and the algorithm discussed in class. You **MAY NOT** simply use Python's sort utilities. The idea is to start with an empty list, and then iterate through each of the elements in the list to be sorted, inserting each item in the proper spot in the new list.

For example:

`reorder([1, 5, 3, 3])` should sort the elements in the list in following steps:

result before insertion	item to insert	result after insertion
[ ]	1	[1]
[1]	5	[1, 5]
[1, 5]	3	[1, 3, 5]
[1, 3, 5]	3	[1, 3, 3, 5]

Also:

- Your resulting list should be sorted in ascending order.
- Googling "insertion sort" will turn up an algorithm that moves each element to the left until it finds the proper spot. However, for this assignment, **you must start with an empty list** and then insert each element into the list at the proper spot.
- You should not shuffle the current value down to the left until you find the right spot for this assignment.
- You will receive credit **ONLY** for using this approach.

> **Special Offer!** If you like a challenge, solve the homework on your own. Need a hint? **See Canvas for hints**, but **only after you try to solve the problem without the hints**.
>
> **FYI**: My solution has 11 lines, including blanks and comments.

Hint: `for/else` may be helpful for this problem, or just use a while loop.

Deliverable

File Structure

In this assignment, you will need to **separate your code into two files** - one for code logic and one for unit test. It's always a good practice to separate the code and the test.

You are going to have **two .py files** for submission:

1. `HW06_FirstName_LastName.py`
2. `HW06_Test_FirstName_LastName.py`

If your have any concern, please do not hesitate to reach out.