# Week 2 - Hands On Exercise - Review Questions

ASD103A-21: Object-Oriented Data Structures using Python, Part1

Prof: Debsankar Mukhopadhyay

Delvis Soto-Soto

Software Developer

Question 1: Select any three options (from A to L) below that represent lists, and in your own words, explain why or why not they can (or cannot) be considered as lists.

Customers waiting in a checkout line: Yes, this can be considered a list. Customers form a linear order, and they are served in a first-come, first-served manner, similar to a list.

A deck of playing cards: No, this is not a list. A deck of playing cards doesn't represent a linear order like a list; cards can be drawn from anywhere in the deck.

A file directory system: Yes, this can be considered a list. Directories and files in a file system can be organized in a linear structure, similar to a list.

A line of cars at a tollbooth: Yes, this can be considered a list. Cars wait in line and are processed in the order they arrived at the tollbooth, resembling a list.

Laundry in a hamper: Yes, this can be considered a list. The order in which laundry is added to the hamper represents a linear arrangement.

A group of friends having a picnic: No, this is not a list. There's no inherent linear order in how friends join or leave the picnic.

Question 2: What are the preconditions on index-based operations with a list?

The preconditions on index-based operations with a list are:

The index should be within the valid range of indices for the list.

Attempting to access an index outside the valid range will result in an "IndexError."

Question 3: How is recursion used in the linked list data structure?

Recursion is often used in linked list operations, particularly in traversing and manipulating the elements of a linked list. For example, when traversing a linked list, a recursive function can move through each node by calling itself on the next node in the list until the end is reached. Similarly, recursion can be used for reversing a linked list, searching for an element, or performing other operations.

Question 4: What is an iterator in Python and what is the purpose of it? Outline 2 methods with a code snippet for it.

An iterator in Python is an object that can be iterated (looped) over. It implements two methods, \_\_iter\_\_() and \_\_next\_\_(). The \_\_iter\_\_() method returns the iterator object itself, and the \_\_next\_\_() method returns the next element from the iterator. When there are no more items to return, it should raise the StopIteration exception.

Here's an example of a simple iterator:

class MyIterator:

    def \_\_iter\_\_(self):

        self.counter = 0

        return self

    def \_\_next\_\_(self):

        if self.counter < 5:

            result = self.counter

            self.counter += 1

            return result

        else:

            raise StopIteration

# Using the iterator

my\_iterator = MyIterator()

iter\_obj = iter(my\_iterator)

for item in iter\_obj:

    print(item)