Activity No. Skill Test	
Course Code: CPE-201L	**Program: BSCPE**
Course Title: Data Structure and Algorithm	**Date Performed:** 8/30/2025
Section: 2A	**Date Submitted:** 8/30/2025
Name: AMPONG, J-KEVIN L.	**Instructor:** Engr. Maria Rizzette H. Sayo

1. Objectives

- To design and implement a fundamental singly linked list data structure from scratch using Python's object-oriented programming features.
- To develop core functionalities for the linked list, specifically an append method to add nodes to the end of the list and a display method to visualize its contents.
- To demonstrate a practical application of the linked list by storing and sequentially connecting a series of characters.

2. Discussion

A linked list is a foundational linear data structure. Unlike arrays, which store elements in contiguous memory locations, a linked list consists of individual objects called nodes. Each node contains two main pieces of information: the actual data (the "cargo") and a reference or pointer to the very next node in the sequence.

The list is accessed through a single entry point, the head, which points to the first node. The last node in the sequence has its pointer set to None (or null), signifying the end of the list. This structure allows for dynamic memory allocation, meaning it can grow or shrink in size easily during program execution.

3. Materials and Equipment

- Google colab - for editing and running the python code.
- GitHub - to store and save the python code.

4. Procedure

1. Node Class Creation: First, a Node class was defined. This class serves as the blueprint for each element in the list, containing an __init__ method to accept data and initialize a next pointer to None.
2. Linked List Class Creation: A LinkedLists class was created to manage the collection of nodes. It was initialized with a head attribute set to None, indicating an empty list.
3. Append Method Implementation: An append method was added to the LinkedLists class.
4. Display Method Implementation: A display method was created to visualize the list's contents. It iterates through each node starting from the head, printing the data of each node and concatenating it into a final formatted string showing the -> linkage.
5. Instantiation and Population: An instance of the LinkedLists class was created. Then, a series of characters ('J', '_', 'K', 'E', 'V', 'I', 'N', etc.) were instantiated as Node objects and added to the list sequentially using the append method.
6. Execution and Verification: Finally, the display() method was called on the linked list object to print the final, connected sequence of characters to the console, verifying that the implementation was successful

5. Output

LinkedLists

```python
class Node:
    def __init__(self, data) -> None:
        self.data = data
        self.next = None


class LinkedLists:
    def __init__(self) -> None:
        self.head = None

    def append(self, node: Node):
        if self.head is None:
            self.head = node
            return

        current_node = self.head
        while current_node.next is not None:
            current_node = current_node.next

        current_node.next = node

    def display(self) -> None:
        if self.head is None:
            return

        current_node = self.head
        data_list = []
        while current_node:
            print(current_node.data)
            data_list += current_node.data
            current_node = current_node.next

        result = ""
        for index, data in enumerate(data_list):
            if len(data_list) > index + 1:
                result += f'{data} -> '
            else:
                result += f'{data}'
        print("results:",result)


linkedlists = LinkedLists()

linkedlists.append(Node('J'))
linkedlists.append(Node('_'))
linkedlists.append(Node('K'))
linkedlists.append(Node('E'))
linkedlists.append(Node('V'))
linkedlists.append(Node('I'))
linkedlists.append(Node('N'))
linkedlists.append(Node(' '))
linkedlists.append(Node('L'))
linkedlists.append(Node('.'))
linkedlists.append(Node(' '))
linkedlists.append(Node('A'))
linkedlists.append(Node('M'))
linkedlists.append(Node('P'))
linkedlists.append(Node('O'))
linkedlists.append(Node('N'))
linkedlists.append(Node('G'))

linkedlists.display()
```

```
J
_
K
E
V
I
N

L
.

A
M
P
```

6. Conclusion

This activity successfully met all its objectives. A functional singly linked list was implemented in Python, complete with Node and LinkedLists classes. The core logic for node creation, list traversal, and appending new elements was correctly programmed and tested.

The key takeaway from this exercise is a deeper, practical understanding of how linked lists operate "under the hood." Manually managing the head pointer and traversing the list to find the end for an append operation highlights the fundamental difference between this data structure and a simple array or Python list. This hands-on experience solidifies the theoretical concepts of nodes, pointers, and sequential access, providing a strong foundation for tackling more complex data structures and algorithms in the future.

Further enhancements could include adding methods for prepend, insert, and delete to create a more robust and fully-featured linked list class.