



UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT



Data Structure and Algorithm

Laboratory Activity No. 6

Singly Linked Lists

Submitted by:
Disomnong, Jalilah M.

Instructor:
Engr. Maria Rizette H. Sayo

08-23-2025

I. Objectives

Introduction

A linked list is an organization of a list where each item in the list is in a separate node. Linked lists look like the links in a chain. Each link is attached to the next link by a reference that points to the next link in the chain. When working with a linked list, each link in the chain is called a Node. Each node consists of two pieces of information, an item, which is the data associated with the node, and a link to the next node in the linked list, often called next.

This laboratory activity aims to implement the principles and techniques in:

- Writing algorithms using Linked list
- Writing a python program that will perform the common operations in a singly linked list

II. Methods

- Write a Python program to create a singly linked list of prime numbers less than 20. By iterating through the list, display all the prime numbers, the head, and the tail of the list. (using Google Colab)
- Save your source codes to GitHub

III. Results

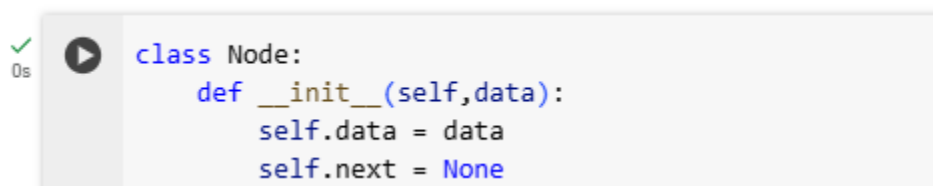
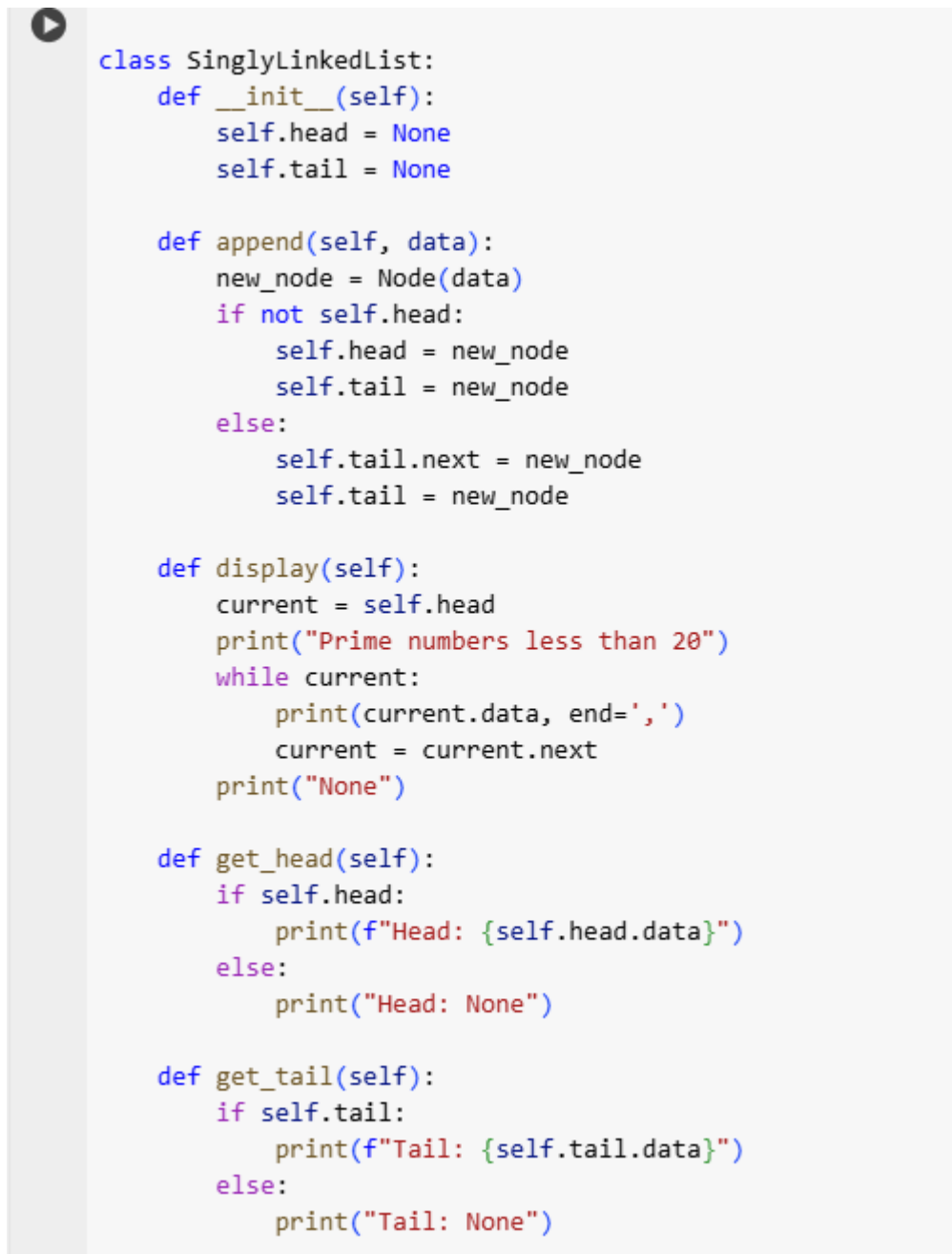


Figure 1 Screenshot of source code refer to this link: [Lab 6 - Colab](#)

In figure 1, The Node class represents an individual element in the linked list, containing two attributes—data, which holds the value of the node, and next, which points to the next node in the sequence. Initially, next is set to None, indicating that the node does not yet link to another.



```

class SinglyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.next = new_node
            self.tail = new_node

    def display(self):
        current = self.head
        print("Prime numbers less than 20")
        while current:
            print(current.data, end=',')
            current = current.next
        print("None")

    def get_head(self):
        if self.head:
            print(f"Head: {self.head.data}")
        else:
            print("Head: None")

    def get_tail(self):
        if self.tail:
            print(f"Tail: {self.tail.data}")
        else:
            print("Tail: None")

```

Figure 2 Screenshot of source code refer to this: [Lab 6 - Colab](#)

In figure 2, The SinglyLinkedList class manages the overall structure of the list. It maintains two pointers: head, which refers to the first node, and tail, which refers to the last node. The append method is responsible for adding new nodes to the list. If the list is empty (i.e., head is None), the new node becomes both the head and the tail. Otherwise, the method links the current tail to the new node and updates the tail reference accordingly. This ensures that nodes are added sequentially, preserving the order of insertion. To visualize the contents of the list, the display method traverses from the head to the tail, printing each node's data. Additionally, the get_head and get_tail methods provide a quick way to inspect the values at the beginning and end of the list, respectively.

```
def is_prime(x):
    if x < 2:
        return False
    for i in range(2, int(x ** 0.5) + 1):
        if x % i == 0:
            return False
    return True

linked_list = SinglyLinkedList()

for num in range(0, 20):
    if is_prime(num):
        linked_list.append(num)
```

Figure 4 Screenshot of source code refer to this link: [Lab 6 - Colab](#)

```
linked_list.display()
linked_list.get_head()
linked_list.get_tail()
```

Figure 3 Screenshot of source code refer to this link: [Lab 6 - Colab](#)

In figure 3 and 4, the function `is_prime` determines whether a given number is prime. It returns `False` for numbers less than 2 and checks divisibility up to the square root of the number for efficiency. The main logic then creates an instance of `SinglyLinkedList` and iterates through numbers from 0 to 19. For each number that passes the `is_prime` check, it is appended to the linked list.

```
➡ Prime numbers less than 20
2,3,5,7,11,13,17,19,None
Head: 2
Tail: 19
```

Figure 5 Screenshot of Output refer to this link: [Lab 6 - Colab](#)

Lastly, in figure 5 it shows the output of a list of prime numbers less than 20 stored in a linked list. It prints the numbers in order, ending with `None` to mark the end of the list. Then it displays the head (first prime: 2) and tail (last prime: 19) of the list.

IV. Conclusion

In this laboratory activity, we performed the fundamental concepts and operations of singly linked lists using Python. Through the implementation of a program that stores and displays prime numbers less than 20, we were able to apply key principles such as node creation, dynamic data insertion, and list traversal. The use of custom classes and methods determine our understanding of how linked lists function as dynamic data structures, particularly in managing sequential data without relying on built-in arrays.

References

[1] Google. (2024). Google Colaboratory. Retrieved April 18, 2024, from <https://colab.research.google.com/>