FINAL EXAM PROFESSIONAL ELECTIVE 2
VISUALIZING LINKED LISTS

NAME: Rafer, Kenneth D.
SECTION: UCOS 4-1

INSTRUCTIONS

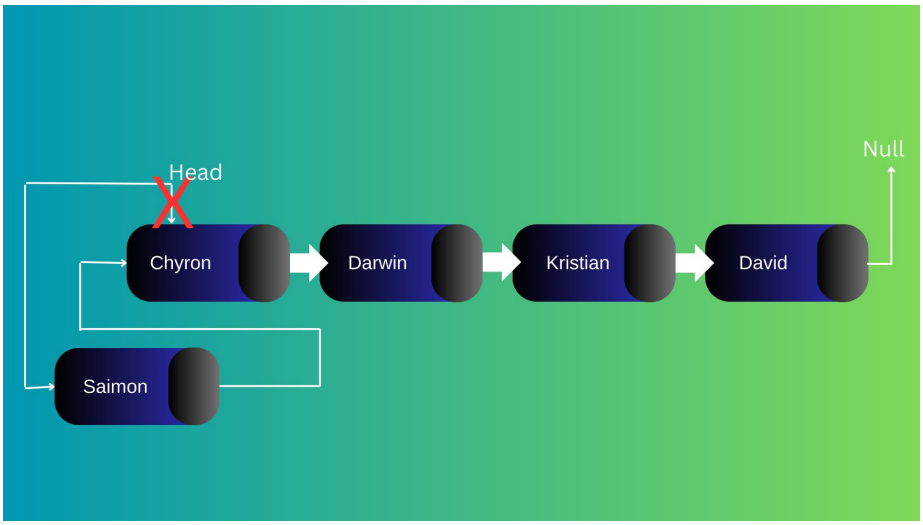
- The goal of this final exam project is to **VISUALIZE HOW LINKED LISTS WORK**. Write first a Linked List implementation using Python OOP with insertion, deletion, updating, and displaying methods. The linked list you'll show in the diagram should contain a minimum of **FIVE ELEMENTS**. You are tasked to **CREATE DIAGRAMS** showing the following:
    - **INSERTION** of a new element to linked list.
    - **DELETION** of a given element inside a linked list.
    - **UPDATING** of an element from the linked list.
    - **DISPLAYING** of all elements from the linked list.

- Please follow the format of this document and **REFER TO THE CRITERIA** on the last page of this document.

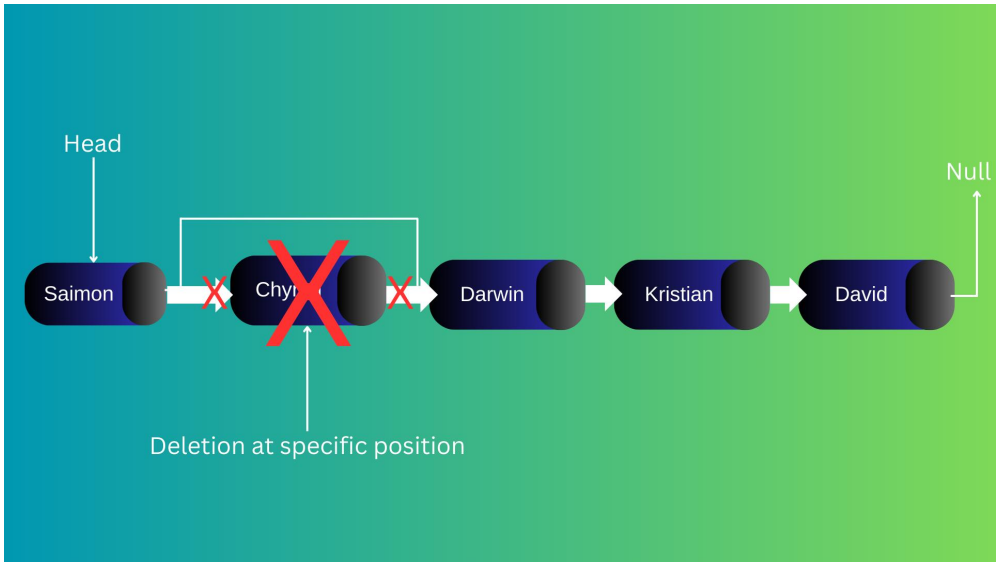
ELEMENTS TO ADD IN LINKED LIST:

Five random names: Saimon, Chyron, Darwin, Kristian, David

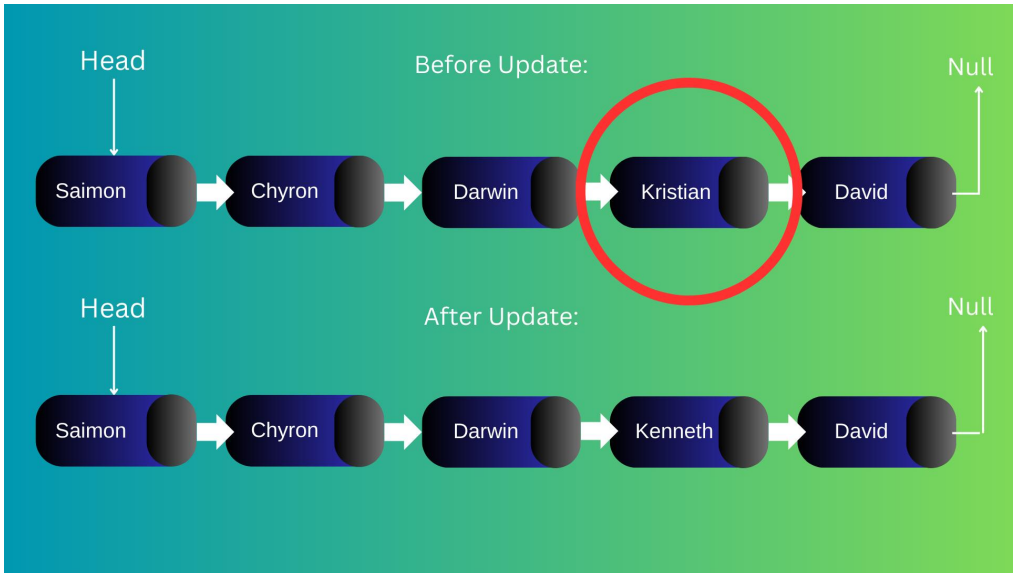
INSERTION OF ELEMENT:



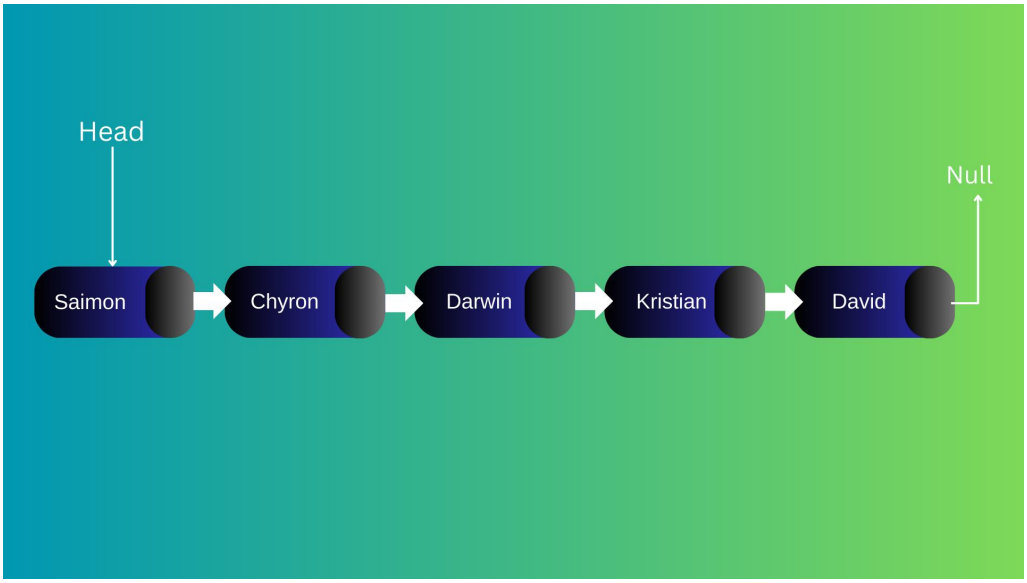
DELETION OF ELEMENT:



UPDATING OF ELEMENT:



DELETION OF ELEMENT:



LINKED LIST IMPLEMENTATION USING PYTHON OOP (INSERTION, DELETION, UPDATING, AND DISPLAYING)

INSERTION

main.py

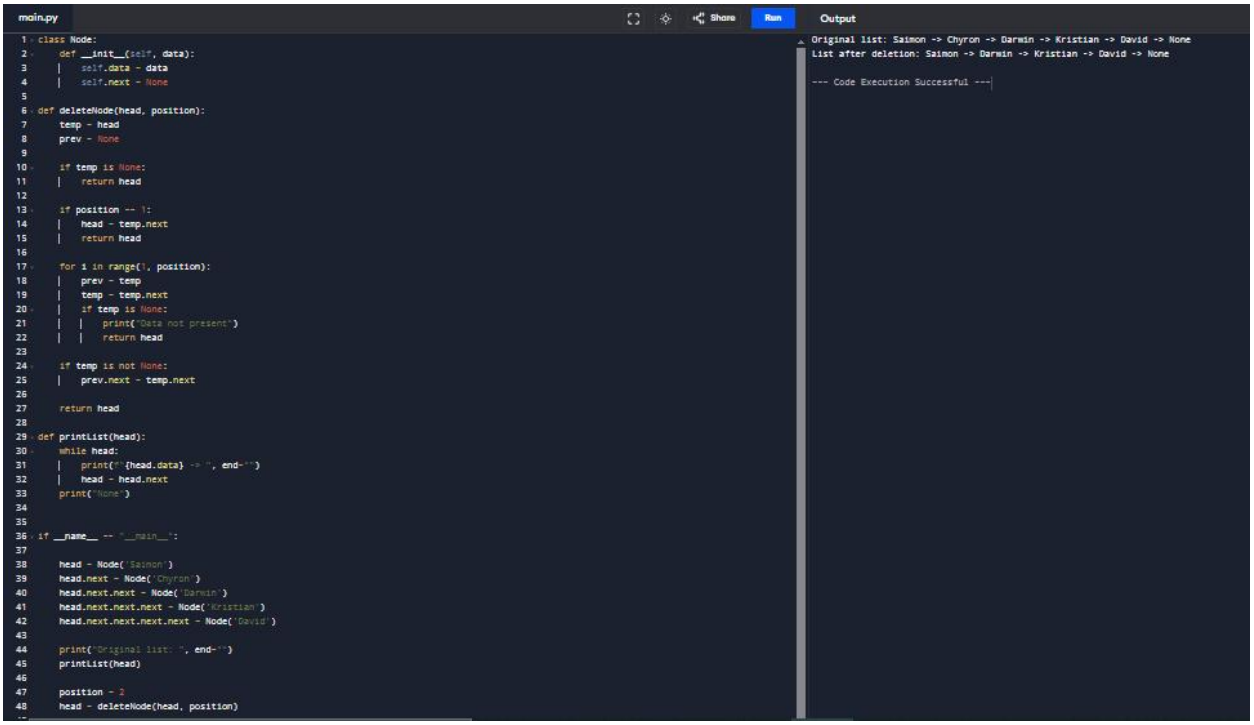
Share Run

```
class Node:
    def __init__(self, new_data):
        self.data = new_data
        self.next = None

def insert_at_front(head, new_data):

    new_node = Node(new_data)

    new_node.next = head

    return new_node


def print_list(head):
    curr = head
    while curr is not None:
        print(f" {curr.data}", end="")
        curr = curr.next
    print()


if __name__ == "__main__":

    head = Node('Chyron')
    head.next = Node('Darwin')
    head.next.next = Node('Kristian')
    head.next.next.next = Node('David')

    data = 'Saimon'
    head = insert_at_front(head, data)

    print_list(head)
```

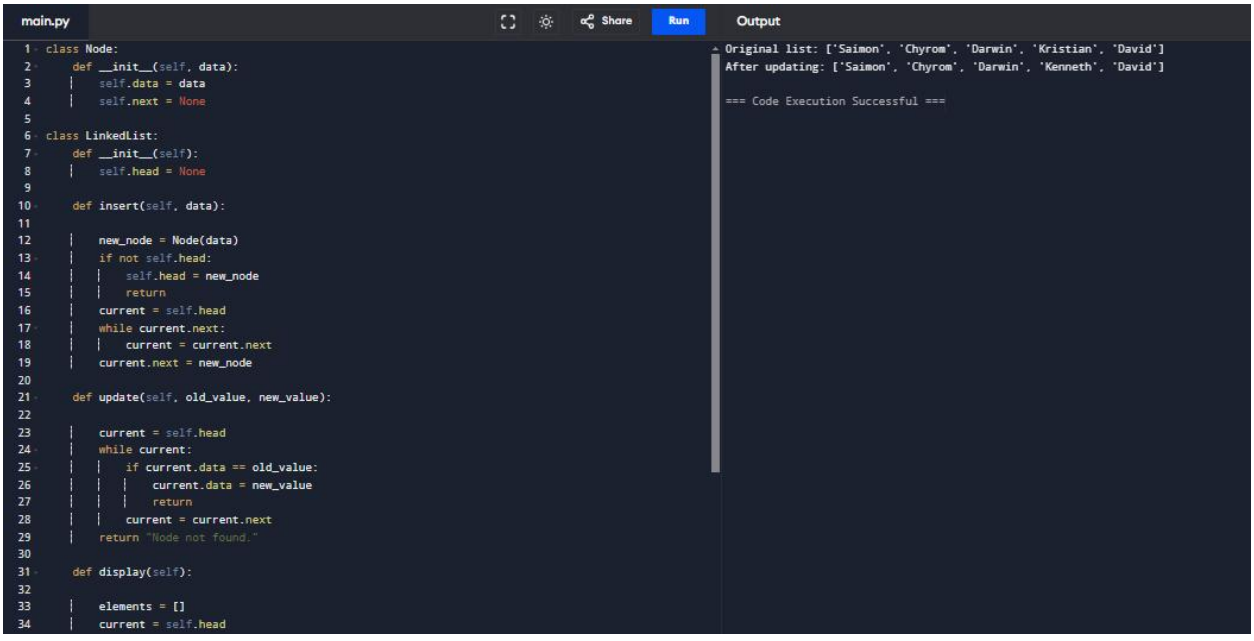
Output

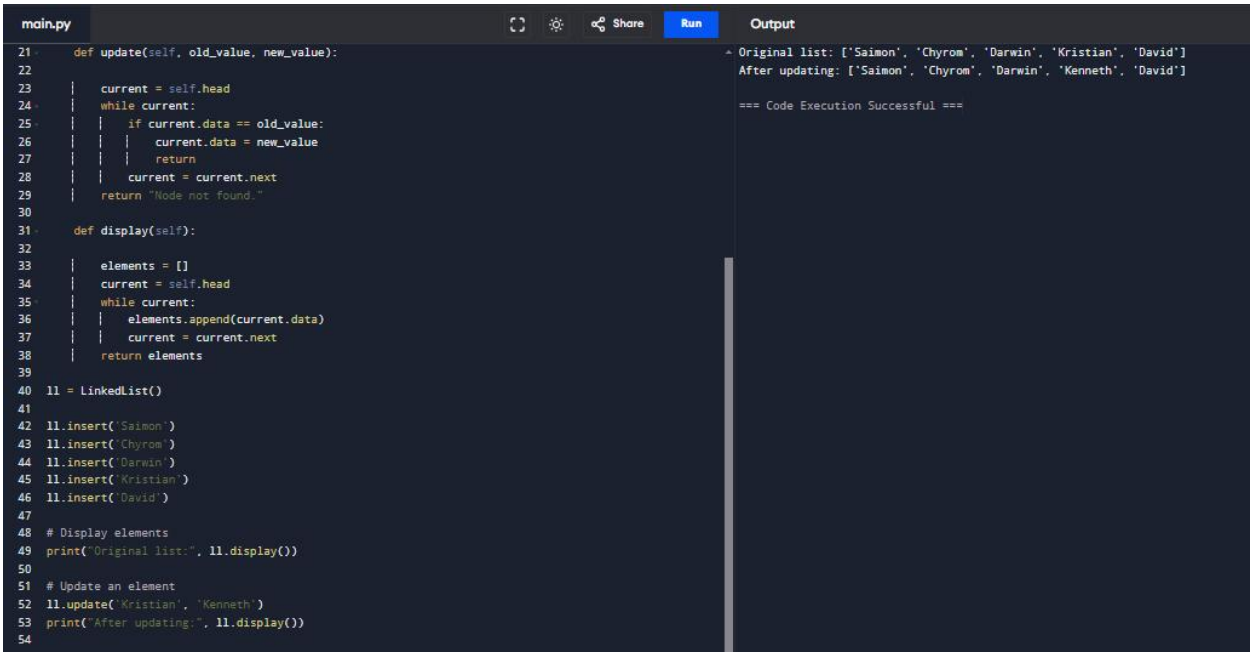
```
Saimon Chyron Darwin Kristian David

=== Code Execution Successful ===
```

DELETION

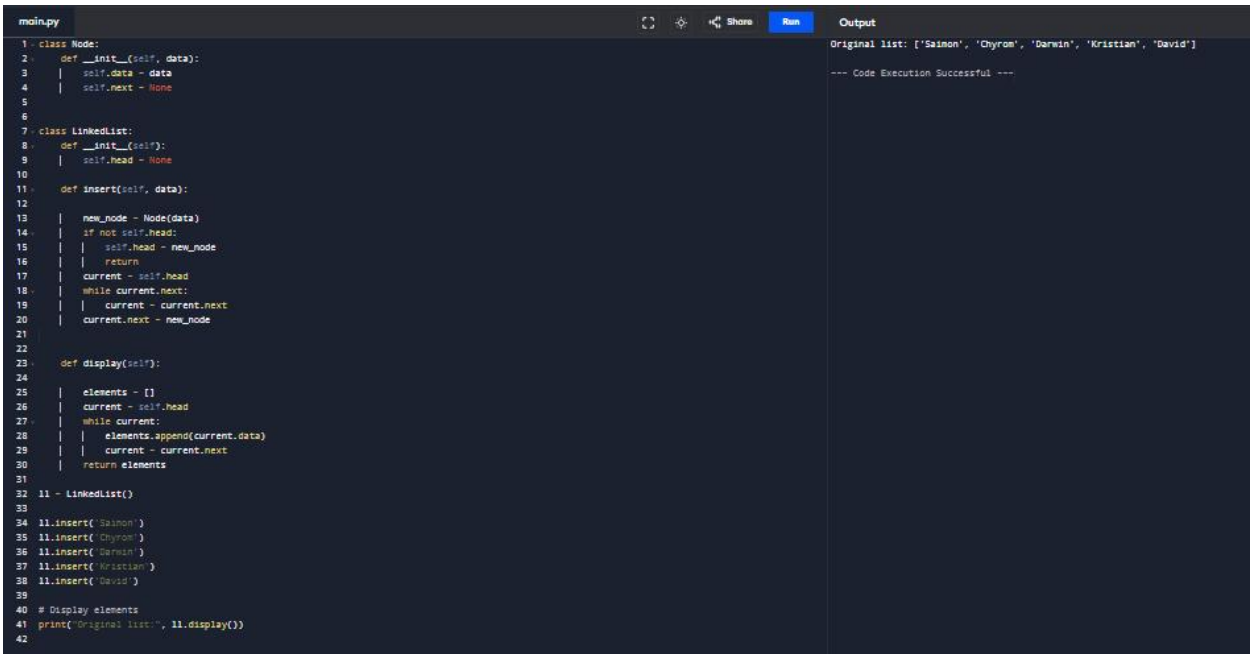


UPDATING




main.py
Share
Run
Output
def update(self, old_value, new_value):
current = self.head
while current:
if current.data == old_value:
current.data = new_value
return
current = current.next
return "Node not found."
def display(self):
elements = []
current = self.head
while current:
elements.append(current.data)
current = current.next
return elements
ll = LinkedList()
ll.insert('Saimon')
ll.insert('Chyrom')
ll.insert('Darwin')
ll.insert('Kristian')
ll.insert('David')
# Display elements
print("Original list:", ll.display())
# Update an element
ll.update('Kristian', 'Kenneth')
print("After updating:", ll.display())
Original list: ['Saimon', 'Chyrom', 'Darwin', 'Kristian', 'David']
After updating: ['Saimon', 'Chyrom', 'Darwin', 'Kenneth', 'David']
=== Code Execution Successful ===

DISPLAYING


main.py
Share
Run
Output
class Node:
def __init__(self, data):
self.data = data
self.next = None
class LinkedList:
def __init__(self):
self.head = None
def insert(self, data):
new_node = Node(data)
if not self.head:
self.head = new_node
return
current = self.head
while current.next:
current = current.next
current.next = new_node
def display(self):
elements = []
current = self.head
while current:
elements.append(current.data)
current = current.next
return elements
ll = LinkedList()
ll.insert('Saimon')
ll.insert('Chyrom')
ll.insert('Darwin')
ll.insert('Kristian')
ll.insert('David')
# Display elements
print("Original list:", ll.display())
Original list: ['Saimon', 'Chyrom', 'Darwin', 'Kristian', 'David']
--- Code Execution Successful ---

CRITERIA:

CATEGORY	**PERCENTAGE**
Technical concepts about Linked Lists are made easier because of the diagram. Diagram clearly represents what takes place in the operations stated.	40%
The diagram created is based on the linked list implementation source code that you provided.	30%
Labels were shown to communicate ideas.	20%
The diagram is done neatly. Color combination is considered carefully.	10%
TOTAL	**100%**