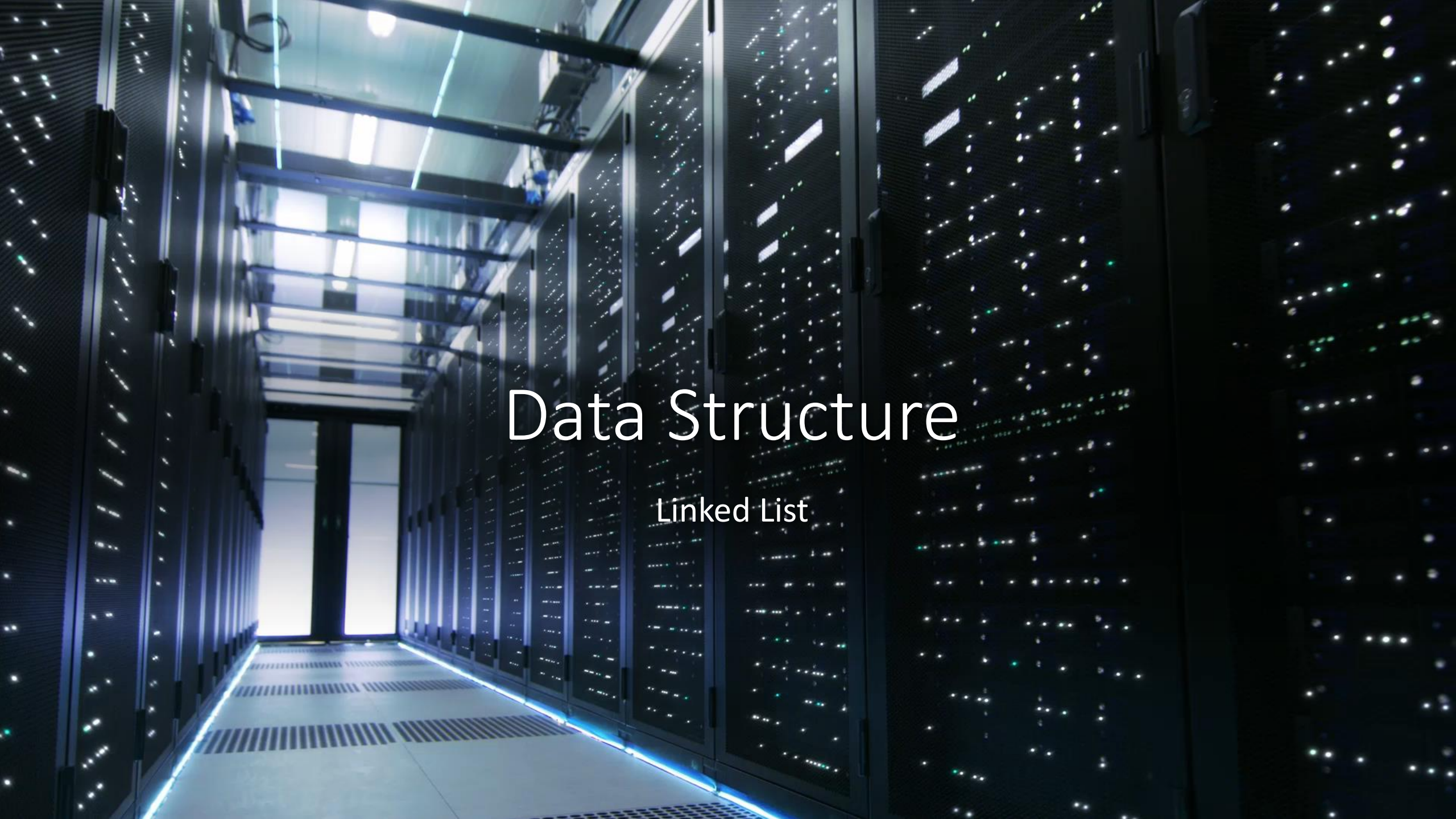


Data Structure

Linked List

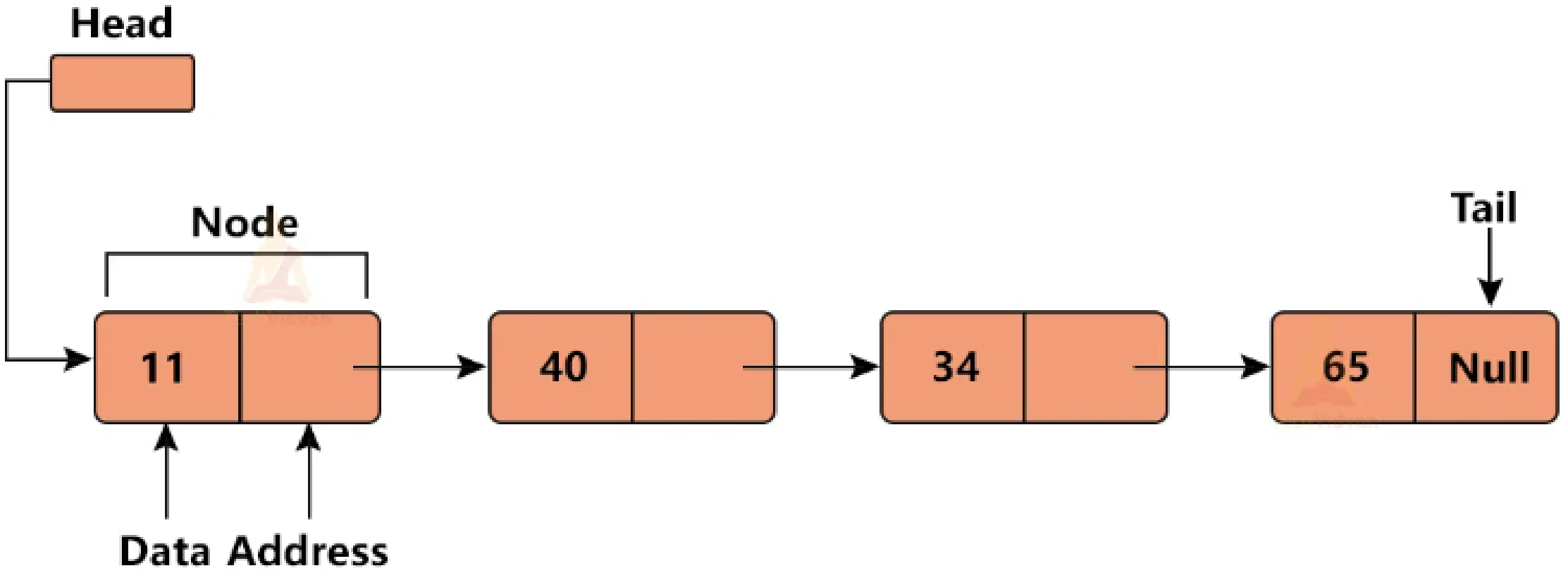
القوائم المرتبطة (Linked Lists)

مجموعة خطية من العناصر حيث يشير كل عنصر إلى العنصر التالي، مما يسمح بتخصيص الذاكرة الديناميكية وعمليات الإدراج والحذف الفعالة.

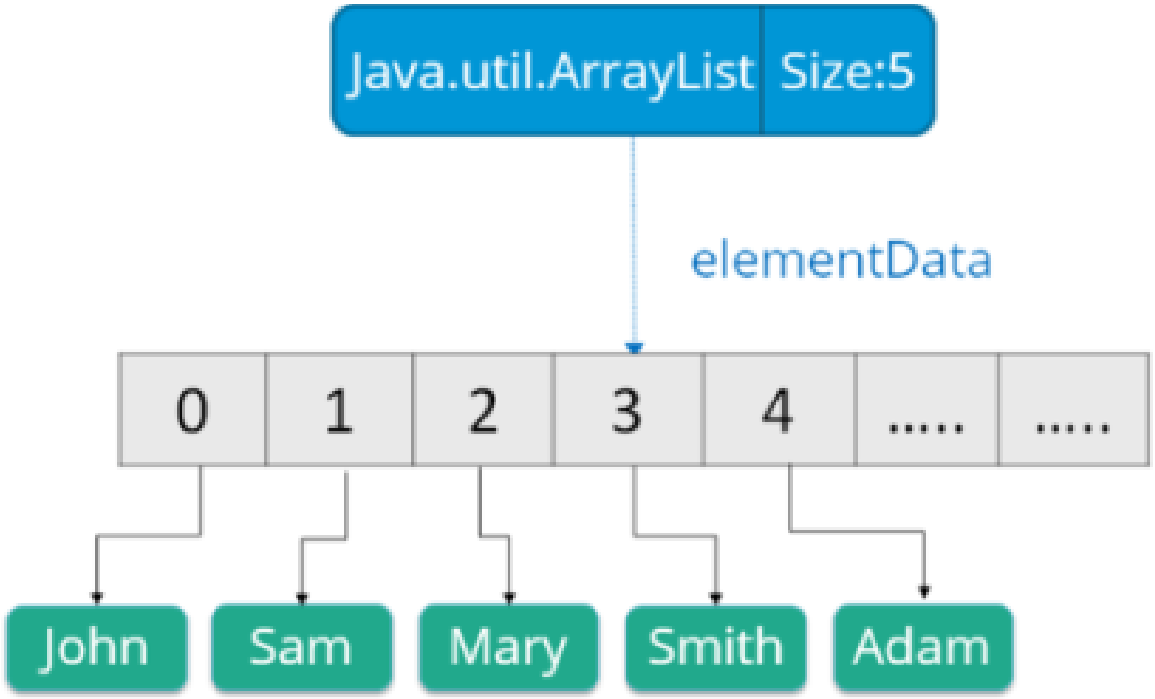
Linked list :هي بنية البيانات الأكثر طلبًا عندما يتعلق الأمر بمعالجة عناصر البيانات الديناميكية.

تتكون القائمة المرتبطة (linked list)من عنصر بيانات يعرف بالعقدة(Node). وتتكون كل عقدة من حقلين(two fields): حقل واحد يحتوي على بيانات(Data)، وفي الحقل الثاني، تحتوي العقدة على عنوان(address) يحتفظ بمرجع إلى العقدة التالية

Linked List



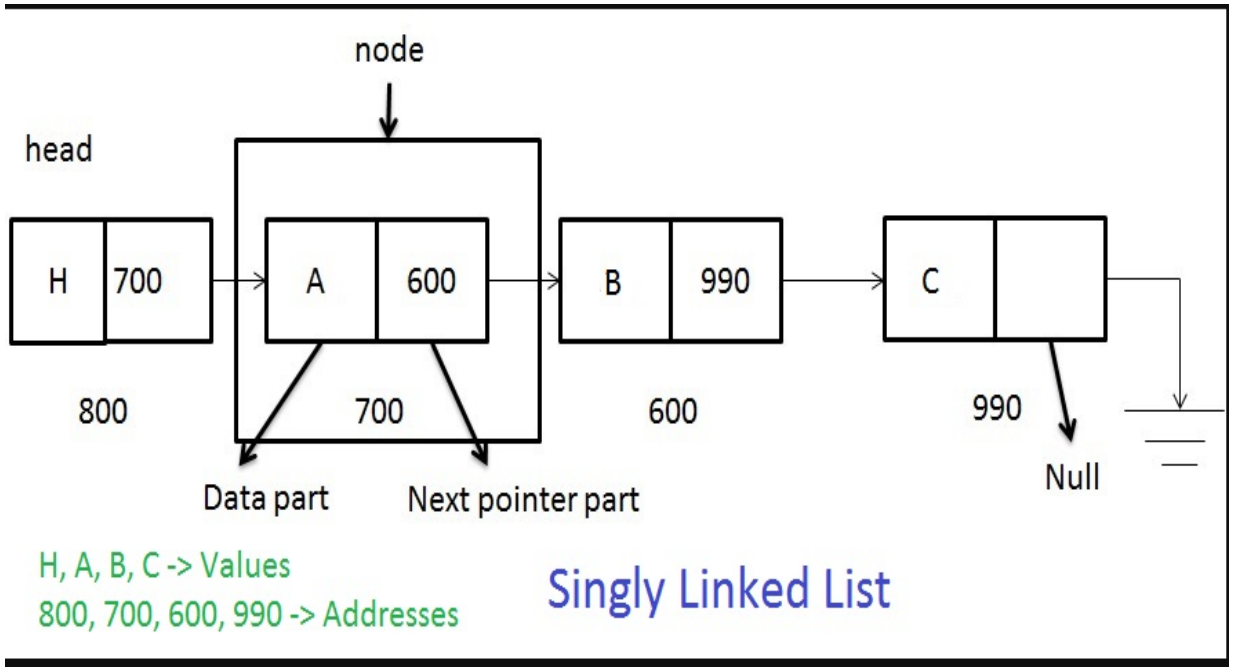
Why linked list?

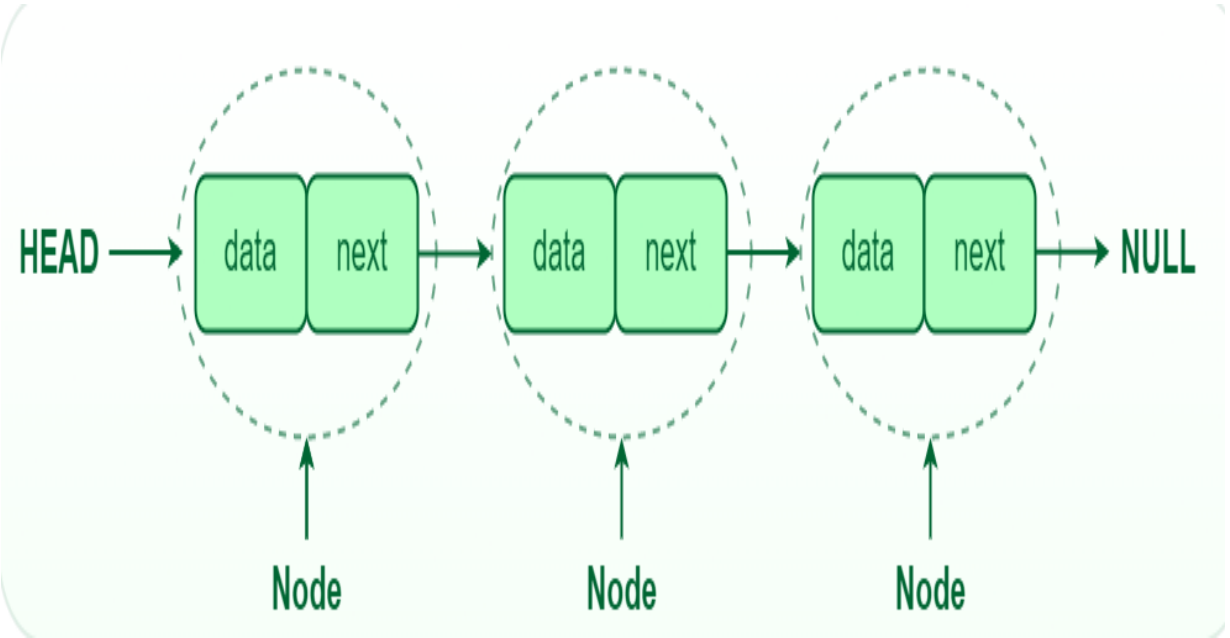


List vs linked list

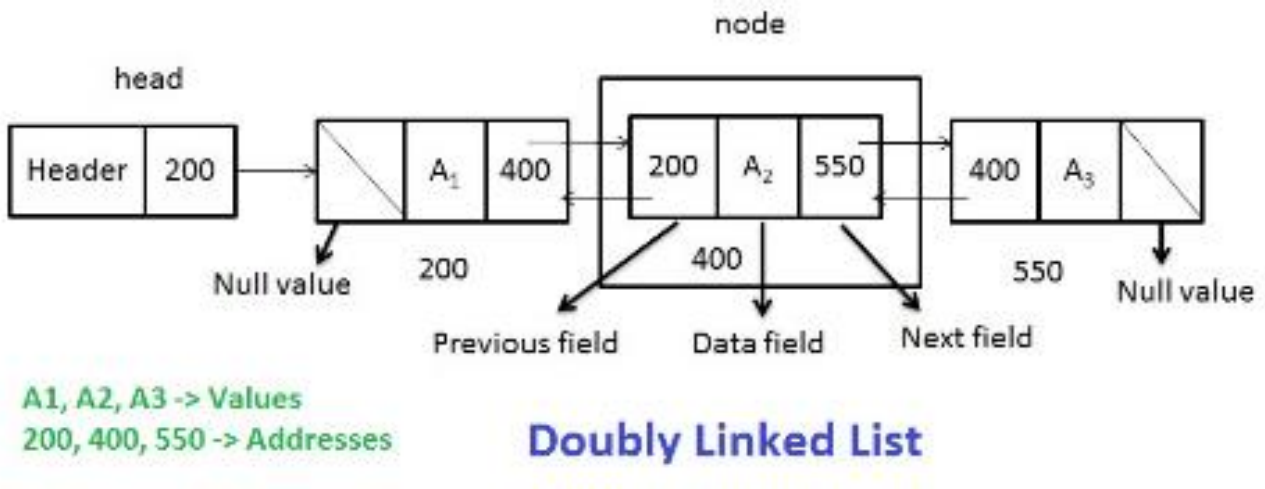
Aspect	Linked List	List (Array-based)
التنفيذ	العناصر تُخزن في عقد مع مؤشرات	العناصر تُخزن في مواقع ذاكرة متسلسلة متجاورة
وقت الوصول	O(n)، حيث يتطلب الأمر الانتقال من العقد الرئيسي	O(1)للوصول إلى العناصر بواسطة الفهرس
وقت الإدراج	O(1)للإدراج في البداية/النهاية، O(n) في أماكن أخرى	O(n)للإدراج في مواقع عشوائية
وقت الحذف	O(1)للحذف في البداية/النهاية، O(n)في أماكن أخرى	O(n)للحذف في مواقع عشوائية
التكلفة الإضافية	ذاكرة إضافية للمؤشرات	لا تكلفة إضافية للذاكرة
الدينامية	حجم ديناميكي، لا حاجة لتخصيص الذاكرة مسبقًا	حجم ثابت، قد تكون هناك حاجة لتغيير الحجم

Single linked list



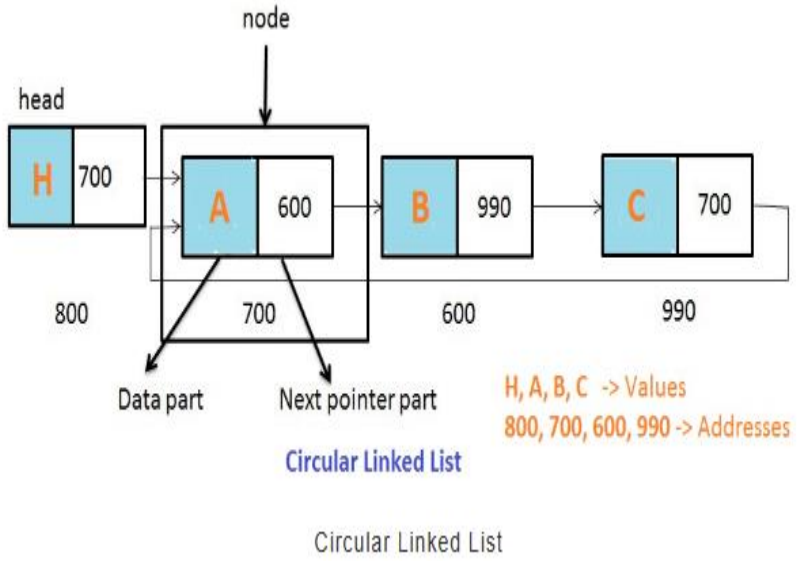


Double linked list



- يتم تعريفها على أنها سلسلة من العقد تشير كل عقدة فيها إلى مساحات الذاكرة.
- تنقسم كل عقدة إلى ثلاثة حقول هي حقل البيانات والحقل التالي أو الحقل الأمامي والحقل السابق أو الخلفي.
- يتم استخدام حقل البيانات للاحتفاظ بقيم البيانات .يتم استخدام الحقل التالي أو الأمامي للاحتفاظ بعنوان العقدة التالية.
- يمكن أن يحتوي الحقل السابق أو الخلفي على عنوان العقدة السابقة .العقدة الأولى في الحقل السابق والعقدة الأخيرة في الحقل التالي تكون دائمًا فارغة

Circular Linked List



Circular Linked List

- Circular Linked List is defined as series of nodes in which each node can carry a data and address information. In this circular linked list, the last node's next pointer does not have null value. Instead, it has the address which point to the first node.

- قائمة مرتبطة دائرية
- يتم تعريف القائمة المرتبطة الدائرية على أنها سلسلة من العقد التي يمكن لكل عقدة أن تحمل فيها بيانات ومعلومات عنوان. في هذه القائمة المرتبطة الدائرية، لا يحتوي المؤشر التالي للعقدة الأخيرة على قيمة فارغة. وبدلاً من ذلك، فهو يحتوي على العنوان الذي يشير إلى العقدة الأولى.

Data Structures Data Structures - List (krivalar.com)

Example of using Linked list

1. **Music Playlist**: A music playlist can be represented using a linked list where each node contains information about a song (e.g., title, artist) and a reference to the next song in the playlist.
2. **Browser History**: Browser history can be implemented using a linked list where each node contains information about a visited webpage and a reference to the previous and next webpages visited.
3. **Undo/Redo Functionality**: Undo and redo functionality in applications like text editors or graphic design software can be implemented using two separate linked lists, where each node represents a state of the document and its changes.
4. **Queue Implementation**: Linked lists can be used to implement a queue data structure, where elements are added to the back of the queue and removed from the front. Each node represents an element, and the "front" of the queue is represented by the head of the linked list.
5. **Stack Implementation**: Linked lists can also be used to implement a stack data structure, where elements are added and removed from the same end. Each node represents an element, and the top of the stack is represented by the head of the linked list.
6. **Employee Management System**: In an employee management system, a linked list could be used to store information about employees, such as their name, ID, department, etc., allowing for easy insertion, deletion, and traversal of employee records.
7. **Task Scheduler**: A task scheduler application can use a linked list to manage tasks, where each node represents a task with information such as priority, deadline, etc., and tasks are ordered based on their priority or deadline.
8. **Train or Bus Reservation System**: In a train or bus reservation system, linked lists can be used to manage the list of available seats, with each node representing a seat and its status (e.g., available, booked).

Ex1: Web browser Nodes

- Node 1: (Google Homepage)
  - URL: "https://www.google.com"
- - Previous: None
- - Next: Node 2

- Node 2: (Wikipedia Article)
- - URL: "https://en.wikipedia.org/wiki/Linked_list"
- - Previous: Node 1
- - Next: Node 3

- Node 3: (YouTube Video)
- - URL: "https://www.youtube.com/watch?v=dQw4w9WgXcQ"
- - Previous: Node 2
- - Next: None

Ex2:nodes(Musicplaylist)

- class SongNode:
- def __init__(self, title, artist):
- self.title = title
- self.artist = artist
- self.next_song = None

Creating a Node Class

a Node class in which we have defined a **__init__** function to initialize the node with the data passed as an argument and a reference with None because if we have only one node then there is nothing in its reference.

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

Linked list Operations

Insert - Insert an element into the list.

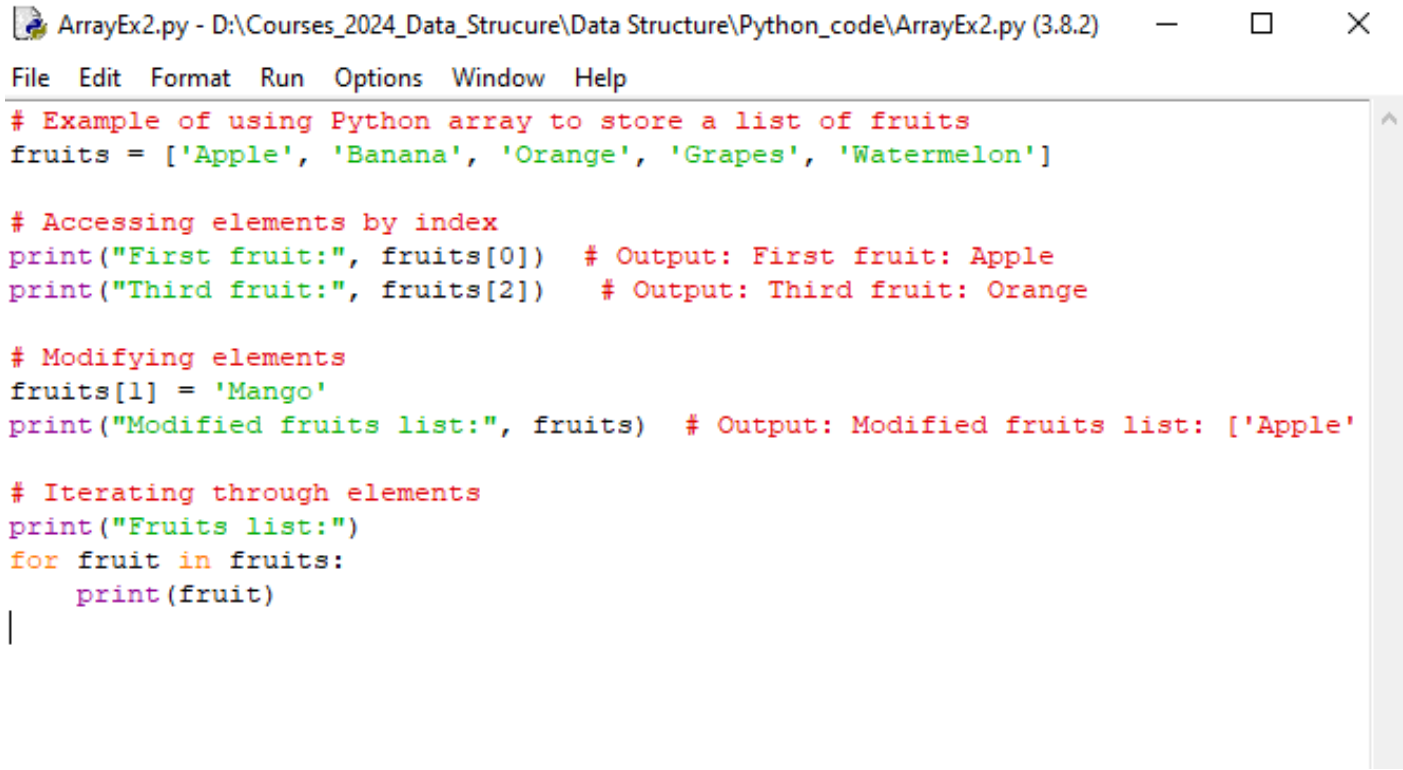
Delete - Delete an element from the list.

Find - Return first occurrence of a key.

PrintList - Print the entire List

how you can create a node for a linked list in Python

- We define a class `Node` with two attributes: `data` to store the value of the node and `next` to store a reference to the next node in the linked list.
- We create instances of the `Node` class to represent individual nodes (`node1`, `node2`, `node3`).
- We link the nodes together by setting the `next` attribute of each node to the next node in the sequence.
- Finally, we traverse the linked list starting from the first node (`node1`) and print the data of each node.



Traversal in a linked list

traverse.py - D:/Courses_2024_Data_Strucure/Data Structure/Python_code/Week2/traverse.py ...

File Edit Format Run Options Window Help

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

class LinkedList:
    def __init__(self):
        self.head = None

    def append(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 traverse(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Example usage:
if __name__ == "__main__":
    linked_list = LinkedList()
    for i in range(1, 5):
        linked_list.append(i)
    linked_list.traverse()
```

ينضمن زيارة كل عقدة في القائمة، وعادةً ما يبدأ من رأس)أو بداية (القائمة ويتحرك نحو ذيل)أو نهاية (القائمة

.أثناء الاجتياز، يمكنك الوصول إلى كل عقدة واحدة تلو الأخرى وإجراء بعض العمليات، مثل قراءة بياناتها أو تعديل قيمتها أو إجراء بعض العمليات الحسابية بناءً على بياناتها.

Traversal in a linked list

1. الوصول المتسلسل: يتيح لك الاجتياز الوصول إلى كل عقدة في القائمة بشكل تسلسلي، مما يضمن أنه يمكنك معالجة كل عنصر في القائمة.

2. نقطة البداية: يبدأ الاجتياز عادةً من رأس القائمة، حيث يكون لديك مرجع إلى العقدة الأولى، ويستمر حتى تصل إلى نهاية القائمة.

3. التكرار: يتضمن الاجتياز التكرار عبر القائمة عن طريق الانتقال من عقدة إلى أخرى حتى تصل إلى نهاية القائمة. ويتم ذلك عادة باستخدام بناء حلقة.

4. العمليات: أثناء الاجتياز، يمكنك إجراء عمليات مختلفة على كل عقدة، مثل قراءة بياناتها، أو تعديل قيمتها، أو إجراء عمليات حسابية بناءً على بياناتها.

5. حالة النهاية: يستمر الاجتياز حتى تصل إلى نهاية القائمة. في القائمة المرتبطة بشكل فردي، يحدث هذا عادةً عندما تواجه عقدة يكون مرجعها "التالي "فارغًا.

يعد الاجتياز عملية أساسية في معالجة القائمة المرتبطة ويستخدم في العديد من الخوارزميات والمهام التي تتضمن قوائم مرتبطة، مثل البحث أو الإدراج أو الحذف أو ببساطة فحص محتويات القائمة.

Insertion in a Linked List

1

append(data) appends a new node with the given data at the end of the linked list.

2

prepend(data) inserts a new node with the given data at the beginning of the linked list.

3

insert_after_node(prev_node, data) inserts a new node with the given data after the specified **prev_node**.

Insertion example

```
    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 prepend(self, data):
    new_node = Node(data)
    new_node.next = self.head
    self.head = new_node

def insert_after_node(self, prev_node, data):
    if not prev_node:
        print("Previous node is not in the list.")
        return
    new_node = Node(data)
```

```
            print("Previous node is not in the list.")
            return
        new_node = Node(data)
        new_node.next = prev_node.next
        prev_node.next = new_node

    def print_list(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Example usage:
if __name__ == "__main__":
    linked_list = LinkedList()
    linked_list.append(1)
    linked_list.append(2)
    linked_list.append(4)
    linked_list.print_list()  # Output: 1 -> 2 -> 4 -> None
    linked_list.insert_after_node(linked_list.head.next, 3)
    linked_list.print_list()  # Output: 1 -> 2 -> 3 -> 4 -> None
    linked_list.prepend(0)
    linked_list.print_list()  # Output: 0 -> 1 -> 2 -> 3 -> 4 -> None
```

Deletion

```
def delete_node(self, key):
    current = self.head
    if current and current.data == key:
        self.head = current.next
        current = None
        return
    prev = None
    while current and current.data != key:
        prev = current
        current = current.next
    if current is None:
        return
    prev.next = current.next
    current = None
```

- Deletion from a Linked List:
- Just like insertion, deletion also works in three different cases:
- 1. Deletion from beginning
- 2. Deletion at the end
- 3. Deleting a node other than the first and the last nod

WebHistory

```
class WebPage:
    def __init__(self, url):
        self.url = url
        self.previous = None
        self.next = None

class BrowserHistory:
    def __init__(self):
        self.head = None
        self.current_page = None

    def visit_page(self, url):
        new_page = WebPage(url)
        if not self.head:
            self.head = new_page
            self.current_page = new_page
        else:
            self.current_page.next = new_page
            new_page.previous = self.current_page
            self.current_page = new_page

    def go_back(self):
        if self.current_page.previous:
            self.current_page = self.current_page.previous

    def go_forward(self):
        if self.current_page.next:
            self.current_page = self.current_page.next

    def print_history(self):
        current = self.head
        while current:
            print(current.url)
            current = current.next
```

```
# Example Usage
if __name__ == "__main__":
    browser = BrowserHistory()
    browser.visit_page("https://www.google.com")
    browser.visit_page("https://en.wikipedia.org/wiki/Linked_list")
    browser.visit_page("https://www.youtube.com/watch?v=dQw4w9WgXcQ")

    print("Browser History:")
    browser.print_history()

    print("\nGo back one page:")
    browser.go_back()
    print("Current page:", browser.current_page.url)

    print("\nGo forward one page:")
    browser.go_forward()
    print("Current page:", browser.current_page.url)
```

Assignment 2

Write a python code to use the following methods in linked list?

- **delete_first()** deletes the first node in the linked list.
- **delete_last()** deletes the last node in the linked list.
- **delete_node(key)** deletes a node with the specified key from the linked list.