جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

Intelligent Medical Systems Department

Lap (5)

Lists & Linked List

Part 1

Subject: Data structure

Class: second

Lecturer: Prof.Dr. Mehdi Ebady Manaa

Programmer:- Fatima Hussein Jawad

Study Year: 2024-2025

Lists

In Python, lists are one of the most common and widely used data types. They are a data structure that allows you to store multiple items in a single variable. You can store any type of data in a list, such as numbers, strings, or even other lists.

Importance of Lists in Python

- **Stores Multiple Data**: Lists allow you to group multiple pieces of data in one variable, making it easier to manage and work with the data.
- **Dynamic:** Unlike in some other languages, lists in Python can change in size, meaning you can easily add or remove items.
- **Can Store Multiple Data Types**: You can store different types of data in a single list, such as **numbers, strings, and booleans**.

How to Create a List in Python ?

To create a list, use square brackets [], and place the elements inside separated by commas. Here's a simple example:

```
numbers = [1, 2, 3, 4, 5]      # A list of integers
fruits = ["apple", "banana", "cherry"]      # A list of strings
mixed = [1, "hello", 3.14, True]      # A list with mixed data types
```

Basic Operations on Lists: Adding Items: You can add items to a list using the append() function to add to the end or insert() to add at a specific position.

1-Adding Items: You can add items to a list using the append() function to add to the end or insert() to add at a specific position.

```
fruits.append("orange")     # Adds "orange" at the end
fruits.insert(1, "mango")   # Inserts "mango" at position 1
```

2-Accessing List Items: Access items using indexes, where indexing starts at 0.

```
print(fruits[0])  # Prints the first item, "apple"
```

3-Updating a List Item:

```
fruits[1] = "blueberry"  # Changes the second item to "blueberry"
```

4-Removing Items: You can remove items using remove(), pop(), or del.

```
fruits.remove("banana")  # Removes "banana"
fruits.pop(2)            # Removes item at position 2
del fruits[0]            # Deletes the first item
```

5-Length of the List: You can get the number of items in the list using len().

```
print(len(fruits))  # Prints the length of the list
```

<u>Additional Examples of Lists</u>

Looping Through a List:

```
for fruit in fruits:
  print(fruit)
```

Checking if an Item Exists:

```
if "apple" in fruits:
  print("Apple is in the list!")
```

Benefits of Using Lists

1-Easier data management and manipulation.

2-Flexibility in working with data of various types and sizes.

3-Provides many useful operations, such as sorting, filtering, and iterating.

4-Lists are a powerful tool for organizing and processing data in various ways, making them essential for Python programmers.

" How to create Class in Python "

1. Defining a Class in Python

A class is a kind of "blueprint" or "template" used to create objects with certain attributes and methods.

Basic class structure:

```
class ClassName:
    def __init__(self, parameters):      # Constructor method
        self.attribute = value           # Here, we define the attributes
```

2. Simple Example: Creating a Class for a "Student"

You can illustrate with an example that defines a Student class with attributes for the student's name and age, as well as a method to display this information.

```
class Student:
    def __ init__(self, name, age):     # Constructor (initializer) method defines attributes
        self.name = name  # Attribute for name
```

```
    self.age = age    # Attribute for age    # Method to display student's information
  def display_info(self):
      print(f"Name: {self.name}, Age: {self.age}")
```

Here, the **__init__** method is a special method called automatically when an object is created from the class. It serves as a constructor to initialize attribute values.

3. Creating an Object and Calling a Method

To show how to create an object from the class and call its methods, you can follow up with this example:

```
student1 = Student("Ali", 21)    # Creating an object of the class and setting attribute values
student1.display_info()    # Calling display_info method to show student information
```

❖ **Running this code will output**:

```
Name: Ali, Age: 21
```

4. Adding a New Method to the Class

To make the example more practical, you can add a new method, such as update_age, to update the student's age.

```
class Student:
 def__ init__(self, name, age):
      self.name = name
      self.age = age
  def display_info(self):
      print(f"Name: {self.name}, Age: {self.age}")
```

```
def update_age(self, new_age):      # Method to update the student's age
    self.age = new_age
    print(f"Updated age to {self.age}")
```

5. Calling the New Method

After updating the class, you can now create an object and use the new methods:

```
student2 = Student("Fatima", 20)
student2.display_info()  # Before updating
student2.update_age(21)  # Updating the age
student2.display_info()  # After updating
```

Key Points to Emphasize:

1. **self** : Refers to the current instance of the class and must be passed as the first parameter in any class method.

2. **__init__**: This is a special method acting as a constructor, used to initialize the primary attributes of an object.

3. Calling Methods: Methods are called using **object_name.method_name()**.

❖ **HOME WORK**

1- What is the purpose of **__init__** in a class?
2- What role does (self)play in a class?
3- How can we call a method inside a class using a specific object?

linked list:

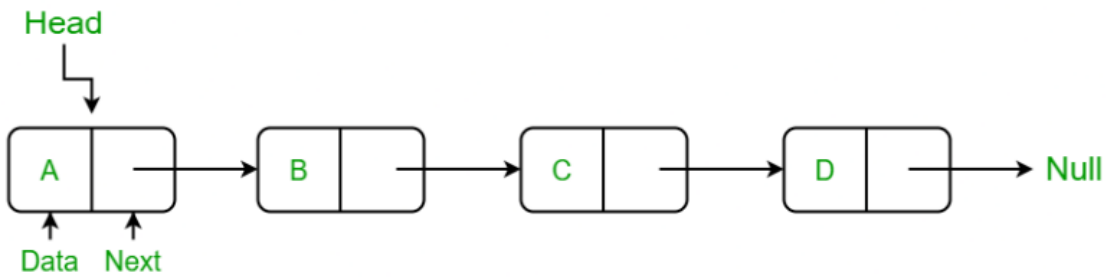
in this Lecture, we will learn about the implementation of a linked list in Python. To implement the linked list in Python, we will use classes in Python. we know that a linked list consists of nodes and nodes have two elements i.e. data and a reference to another node. Let's implement the node first.

Types of linked lists

- ❑ Single Linked List
- ❑ Double Linked List
- ❑ Circular Linked List
- ❑ Linked List with Header
- ❑ Sorted Linked List

What is Linked List in Python ?

A linked list is a type of linear data structure similar to arrays. It is a collection of nodes that are linked with each other. A node contains two things first is data and second is a link that connects it with another node. Below is an example of a linked list with four nodes and each node contains character data and a link to another node. Our first node is where head points and we can access all the elements of the linked list using the head.



Creating a linked list in Python

In this LinkedList class, we will use the Node class to create a linked list. In this class, we have an **__init__** method that initializes the linked list with an empty head. Next, we have created an **insertAtBegin()** method to insert a node at the beginning of the linked list, an **insertAtIndex()** method to insert a node at the given index of the linked list, and **insertAtEnd()** method inserts a node at the end of the linked list. After that, we have the **remove_node()** method which takes the data as an argument to delete that node. In the **remove_node()** method we traverse the linked list if a node is present equal to data then we delete that node from the linked list. Then we have the **sizeOfLL()** method to get the current size of the linked list and the last method of the LinkedList class **is printLL()** which traverses the linked list and prints the data of each node.

Creating a Node Class

We have created 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
```

Insertion in Linked List

Insertion at Beginning in Linked List

This method inserts the node at the beginning of the linked list. In this method, we create a new_node with the given data and check if the head is an empty node or not if the head is empty then we make the new_node as head and return else we insert the head at the next new_node and make the head equal to new_node.

```
def insertAtBegin(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return
    else:
        new_node.next = self.head
        self.head = new_node
```

Insert a Node at a Specific Position in a Linked List

This method inserts the node at the given index in the linked list. In this method, we create a new_node with given data , a current_node that equals to the head, and a counter 'position' initializes with 0. Now, if the index is equal to zero it means the node is to be inserted at begin so we called insertAtBegin() method else we run a while loop until the current_node is not equal to None or (position+1) is not equal to the index we have to at the one position back to insert at a given position to make the linking of nodes and in each iteration, we increment the position by 1 and make the current_node next of it. When the loop breaks and if current_node is not equal to None we insert new_node at after to the current_node. If current_node is equal to None it means that the index is not present in the list and we print "Index not present".

```
# Indexing starts from 0.
def insertAtIndex(self, data, index):
    new_node = Node(data)
    current_node = self.head
    position = 0
    if position == index:
```

```
        self.insertAtBegin(data)
    else:
        while(current_node != None and position+1 != index):
            position = position+1
            current_node = current_node.next
        if current_node != None:
            new_node.next = current_node.next
            current_node.next = new_node
        else:
            print("Index not present")
```

Insertion in Linked List at End

This method inserts the node at the end of the linked list. In this method, we create a new_node with the given data and check if the head is an empty node or not if the head is empty then we make the new_node as head and return else we make a current_node equal to the head traverse to the last node of the linked list and when we get None after the current_node the while loop breaks and insert the new_node in the next of current_node which is the last node of linked list.

```
def inserAtEnd(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return
    current_node = self.head
```

```
    while(current_node.next):
        current_node = current_node.next
    current_node.next = new_node
```

#Example

```
# Node definition
class Node:
    def __init__(self, data):
        self.data = data  # Store the data
        self.next = None  # Pointer to the next node
# LinkedList definition
class LinkedList:
    def __init__(self):
        self.head = None  # Head of the linked list
    # Add a new node to the end of the linked list
    def append(self, data):
        new_node = Node(data)  # Create a new node
        if not self.head:  # If the linked list is empty
            self.head = new_node
            return
        current = self.head
        while current.next:  # Traverse to the end of the list
            current = current.next
        current.next = new_node
```

```
  # Display the elements of the linked list
  def display(self):
      current = self.head
      while current:  # Traverse through the nodes
          print(current.data, end=" -> ")
          current = current.next
      print("None")  # End of the linked list
# Testing the LinkedList
linked_list = LinkedList()
linked_list.append(10)
linked_list.append(20)
linked_list.append(30)
print("Linked List elements:")
linked_list.display()
```

output :

Linked List elements:

10 -> 20 -> 30 -> None

Explanation:

1. `Node` Class:
   - Represents each node in the linked list.
   - `data`: Holds the value of the node.
   - `next`: A pointer to the next node.
2. `LinkedList` Class:
   - Represents the linked list as a whole.
   - `head`: Points to the first node of the list.
   - `append(data):` Adds a new node with the given data to the end of the list.
   - `display():` Prints all elements of the linked list.