Lec 2 explanation material

Array

Today we are looking at creation of array in python and how we handle data using arrays in Python, with a focus on the NumPy library.

**What is NumPy?**

"NumPy is a core library for numerical operations in Python. It provides a high-performance multidimensional array object.

Creation of Array using NumPy

**arr = np.array([1, 2, 3, 4, 5, 6])**

print("Array + 2:", arr + 2)  # Element wise addition

print("Array \* 2:", arr \* 2)  # Element wise multiplication

print("Array squared:", arr \*\* 2)  # Element wise multiplication

**Read Operation:**

**arr[2]**

**accessing the element in the array is always on constant time**

**Update Operation:**

**arr[2] = 10**

**Updating the element in the array is always on constant time**

**Delete operation is bit different from the deletion in the dynamic data structure.**

**During deletion in array using NumPy it creates a new array.**

**So, use of this is crucial for the efficient handling of large numerical datasets, as it allows for more predictable memory usage and better performance.**

**Question are there arrays in Python**

**Answwr is yes,**

Python does not have built-in support for Arrays, but [Python Lists](https://www.w3schools.com/python/python_lists.asp) can be used instead.

Lists are built into the Python programming language, whereas arrays aren't. Arrays are not a built-in data structure, and therefore need to be imported via the array module in order to be used.

Way to use them is by importing array module

List

Python lists are dynamic and can store different types of data. We're going to look at some basic operations you can perform on lists, often referred to as CRUD operations.

In Python, a list is a collection which is ordered and changeable. It allows us to store a sequence of items.

**Creating a List:**

"We start by creating a simple list. For example, **list\_example = [10, 20, 30, 40, 50]**. This list contains five elements, all of which are numbers."

**Read Operation:**

Python lists are indexed, meaning each element in the list can be accessed using its position or index.

**Create/Insert Operation:**

"Adding a new element to a list is straightforward. We use the **append** method. For instance, **list\_example.append(60)** adds a new element, 60, at the end of the list."

**Update Operation:**

Modifying elements in a list is just as simple. We directly access the element by its index and assign a new value. For example, **list\_example[2] = 100** changes the third element of the list from 30 to 100.

Delete Operation

To remove an element, we can use the **remove** method. This method removes the first occurrence of the value. **list\_example.remove(40)** will remove the element 40 from our list.

Question   
**Why Lists Are Used**Lists in Python are used because of their flexibility and ease of use. They can grow and shrink in size, hold different types of data, and have several built-in methods that make them incredibly versatile for various tasks.

LinkedList

A linked list is a fundamental data structure that consists of a sequence of elements, each linked to the next

Some basic idea of some fcution and parameters in the code

The **\_\_init\_\_** method in Python is significant as it serves as the constructor for a class. It is automatically invoked when a new instance of a class is created. The primary purpose of this method is to initialize the attributes of the class.

**Initialization**: When you create an object of a class (e.g., **obj = MyClass()**), Python automatically calls the **\_\_init\_\_** method for that class.

The **self** keyword in Python is significant for its role in object-oriented programming (OOP). It is used to represent **an instance** of the class, **allowing access** to the **attributes** and **method**s of the class in Python.

**Class Node:**

Our linked list is made up of nodes, so we start with a **Node** class. Each node has data and a reference to the next node.

"In the **\_\_init\_\_** method, we initialize the **data** and **next** attributes. **data** holds the value, and **next** will point to the next node in the list."

So as we know each node is connected with each other so that’s which we need to have the point to next node so that’s why we have the next value in the node.

**Display Method:**

"The **display** method prints out the values in the linked list. We traverse the list from **head** to the end, printing each node's data."

**Append Method:**

The **append** method adds a new node to the end of the list. We create a new node and, if the list is empty, set it as the head. Otherwise, we traverse to the end of the list and then add the new node.

**Insert at Beginning:**   
  
"To insert a node at the beginning, we create a new node and set its **next** to the current **head**. Then, we update **head** to be the new node."

**Insert After a Specific Node:**This method inserts a new node after a given node. We first check if the given node exists. If it does, we insert the new node after it.  
**Delete Node:**

In the **delete\_node** method, we remove a node with a specific key. We handle two cases: deleting the head node and deleting a node other than the head. We traverse the list to find the node to delete and update the pointers accordingly.

Stack  
  
Stack is a fundamental data structure in computer science, which follows the Last In, First Out (LIFO) principle.

A stack is like a stack of books. You can only add or remove books from the top of the stack. In programming, we refer to adding an item as 'pushing' and removing an item as 'popping'

Using Python List as a Stack:

* **"Python's list provides an efficient way to use it as a stack. We'll create a list named** stack\_example **to demonstrate this."**

Push Operation:

* **"The push operation adds an element to the top of the stack. In a Python list, we use the** append **method to do this."**
* **[Demonstrate pushing elements onto the stack]**
* **"We've added elements 10, 20, 30, 40, 50, and 60 to the stack. Let's print the stack:** print('Stack elements after Push:', stack\_example)**."**

Pop Operation:

* **"The pop operation removes the top element from the stack. The** pop **method in Python lists removes the last element, mimicking the stack's pop behavior."**

Use of stack

Stacks are used in various applications like undo mechanisms in software, function call management in programming, and more.