Python does not have built-in support for Arrays, but Python lists can be used instead.

By default, python lists are dynamic and can hold elements of different data types.

But if you need to use only arrays there are **ways like importing an array**

Syntax:

import array  
new\_arr=array.array('i',[1,2,3,4,5])  
print(new\_arr[0]) #1

**also, we can use Numpy**

syntax:

import numpy as np  
arr1=np.array([1,2,3,4,5])  
arr2=np.array([6,7,8,9,10])  
print(arr1+arr2) #[ 7 9 11 13 15]

Linked lists came to solve arrays issues like:

**Size:**

**Arrays** are fixed-size Elements stored in contiguous memory locations.

**Lists** Can easily grow or shrink in size by allocating or deallocating memory as needed.

**Linked Lists**: Insertion and deletion of elements can be done in constant time O(1) if you have a reference to the node, no need to shift elements like arrays.

**memory-efficient:**

**Linked Lists**: This can be **more memory-efficient** for collections that change size frequently.

**Arrays**: This can lead to **wasted memory** if the allocated array is larger than needed.

**Use Case Suitability**:

**Linked Lists** Are preferable when the data structure size changes frequently, and you need efficient insertions and deletions.

**Arrays** are better when you need efficient random access to elements and the collection size is known and relatively static.

**Cache Performance**:

**Arrays and lists**: Better cache performance due to contiguous memory allocation contiguous memory allocation.

**Linked Lists**: Poorer cache performance because of non-contiguous memory allocation.

**Arrays** are optimal when you need *fast access to elements* and have a fixed-size dataset.

**Linked Lists** are ideal for scenarios involving frequent insertions and deletions, especially when the exact size of the dataset is unknown or changes frequently.

**Python Lists** offer a good balance for many applications, providing the benefits of dynamic resizing and efficient random access, suitable for general-purpose use.

Used built-in Python lists as they provide benefits of both arrays and linked lists in a coding exercise

A *left rotation* operation on an array of size shifts each of the array's elements unit to the left. Given an integer, , rotate the array that has many steps left and return the result.

#A left rotation operation on an array of size n shifts each of the array's elements 1 unit to the left.  
def rotateLeft(d, arr):  
 # time complexity O(n) + O(n) = O(n) n is the length of the array  
 return arr[d:] + arr[:d]  
  
print(rotateLeft(2, [1, 2, 3, 4, 5]))

This is to practice traversing a linked list. Given a pointer to the head node of a linked list, print each node's data element, one per line. If the head pointer is null (indicating the list is empty), there is nothing to print.

def printLinkedList(head):

    while head:

        print(head.data)

        head=head.next