DATA STRUCTURE IN PYTHON Interview Questions

BY – VIKAS MAURYA

YOUTUBE

1. CODE WITH VIKAS
2. VIKAS MAURYA ACADEMY

1.What is a Data Structure in python?

The Data Structure is a description of how data is organised (stored), modified, and accessed. By specifying how different sorts of data connect to one another, it also develops relationships and forms algorithms. The data structure of any computer language is a crucial concept in algorithmic design.

2.What Are the Various Data Structure Types?

The numerous forms of data structures are as follows:

- Lists : Lists are a collection of related objects that are linked to the data items before and after them.
- Arrays: Arrays are a collection of values with the same value.
- Stacks: LIFO data structures, or stacks, are those in which the element placed last is accessed first.
- Queues: Queues are a first-in-first-out data structure.
- Graphs: In this data structure, data values are stored in nodes connected by edges.
- Trees: Like a linked list, the data values are linked in a hierarchical structure.
- Hash table: A table in which each value is assigned a key and then preserved, making individual values more accessible.
- Heaps: A binary tree data structure that allows you to compare parent and child data values.

3.What Applications Do Data Structures Have?

Data structures are useful for a number of things, including:

- Dynamic memory allocation should be implemented.
- Model sets and other types of data
- Network modelling
- Identifying and resolving the most basic issues

4 Give three reasons why NumPy arrays are better than Python lists.

This is a popular Python data structure interview question. Arrays in NumPy have three advantages over Python lists:

- It is more efficient to use a NumPy array. The NumPy arrays continue to grow in size. It has the potential to be thirty times faster than Python lists.
- NumPy is a programming language that is both faster and easier to use. It comes with a number of free vector and matrix operations that help you save time and effort. They can also be carried out successfully.
- Finally, Python lists have several limitations, such as the inability to perform element-wise addition, multiplication, and other vectorized operations. Python must also keep type information for each entry because lists contain a variety of objects. Arrays, on the other hand, have homogeneous objects and hence do not have these restrictions.

5What are tuples and lists? What is the main distinction between the two?

Lists and Tuples are sequence data structures in Python that can store a collection of items. Both sequences are capable of storing items of various data types. Parantheses are used to represent tuples ('ansh', 5, 0.97), while square brackets are used to represent lists ('sara', 6, 0.19).
The fundamental difference between the two is that lists are mutable objects, but tuples are not. Tuples are fixed and cannot be modified in any way, despite the fact that lists can be changed, appended, or sliced on the fly.
Run the following example in Python IDLE to see the difference:

```
my_tuple = ('sara', 6, 5, 0.97)
my_list = ['sara', 6, 5, 0.97]
print(my_tuple[0])    # output => 'sara'
print(my_list[0])    # output => 'sara'
my_tuple[0] = 'ansh'    # modifying tuple => throws an error
my_list[0] = 'ansh'    # modifying list => list modified
print(my_tuple[0])    # output => 'sara'
print(my_list[0])    # output => 'ansh'
```

6.What is the distinction between Python Arrays and Python Lists?

- Arrays in Python can only include components of the same data type, hence the data type of the array must be homogeneous. It's a small wrapper for C language arrays that utilises a fraction of the memory of lists.
- Lists in Python can contain components of several data types, making list data types heterogeneous.

```
import array
a = array.array('i', [1, 2, 3])
for i in a:
    print(i, end=' ')    #OUTPUT: 1 2 3
a = array.array('i', [1, 2, 'string'])    #OUTPUT: TypeError: an integer is required (got type str)
a = [1, 2, 'string']
for i in a:
   print(i, end=' ')    #OUTPUT: 1 2 string
```

7.What exactly do you mean when you say NumPy?

NumPy is an open-source, python-based, general-purpose tool for processing arrays that is extensively used, simple to use, versatile, and widely used. NumPy is the abbreviation for NUMerical Python. This software is well-known for its highly optimised tools, which result in increased speed and a powerful N-Dimensional array processing function, which is designed specifically for working with complex arrays. Due to its popularity and powerful performance, as well as its versatility to conduct various operations such as trigonometric operations, algebraic and statistical computations, it is most often employed in scientific computations and for a variety of broadcasting purposes.

8.What are the advantages of NumPy arrays versus Python lists?

- The list data structure in Python is incredibly efficient and capable of handling a wide range of jobs. They do, however, have substantial limits when it comes to computing vectorized operations like element-wise multiplication and addition. The type of each element is also required by the python lists, which adds overhead because type dispatching code is called every time an action on any element is performed. This is where NumPy arrays come into play, as they manage all of Python lists' limitations.
- Furthermore, when the size of the NumPy arrays grows larger, NumPy becomes 30x faster than Python List. This is due to the homogeneous nature of Numpy arrays, which causes them to be densely packed in memory. This protects the memory.

9.What stages are involved in creating 1D, 2D, and 3D arrays?

- 1D array creation:

```
import numpy as np
one_dimensional_list = [1,2,4]
one_dimensional_arr = np.array(one_dimensional_list)
print("1D array is : ",one_dimensional_arr)
```

- 2D array creation:

```
import numpy as np
two_dimensional_list=[[1,2,3],[4,5,6]]
two_dimensional_arr = np.array(two_dimensional_list)
print("2D array is : ",two_dimensional_arr)
```

- 3D array creation:

```
import numpy as np
three_dimensional_list=[[[1,2,3],[4,5,6],[7,8,9]]]
three_dimensional_arr = np.array(three_dimensional_list)
print("3D array is : ",three_dimensional_arr)
```

10. In Python, explain slicing.

Slicing is a technique for extracting a subset of elements from a sequence type like a list, tuple, or string. For example, slicing a list refers to selecting a section or subset of the list for a function while keeping the rest of the list unaltered. As a result, you can take out a section without affecting the remainder of the material.

The following is the syntax for slicing a list: List name [start:stop:steps]

11. In Python, explain the differences between xrange and range.

Despite the fact that both xrange() and range() produce a sequence of numbers, they are not the same. Range produces a Python list of integers, whereas xrange generates an xrange generator object. As a result, xrange() does not construct a static list; instead, the value is created as needed.

13.What is the difference between linear and non-linear data structure?

A data structure that stores data in a linear order is known as a linear data structure. You can only traverse the data structure using that linear series. Arrays and stacks are examples of linear data structures.
Data is organised in non-linear ways with non-linear data structures. For example, a graph is made up of nodes connected by edges. The edges that connect the nodes, not the order in which they are joined, determine the relationships between the data values. Trees are examples of non-linear data structures.

14.What exactly is a stack?

A stack is an abstract data type that provides a linear data structure, analogous to a physical stack or pile where objects may only be removed from the top. As a result, item insertion (push) and deletion (pop) take place only at one end of the stack, the top of the stack, and in a certain order: LIFO (Last In First Out) or FILO (First In Last Out) (First In Last Out).

Implementation in Python

```
stack = []

# append() function to push
# element in the stack
stack.append('a')
stack.append('b')
stack.append('c')

print('Initial stack')
print(stack)

# pop() function to pop
# element from stack in
# LIFO order
print('\nElements popped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are popped:')
print(stack)

# uncommenting print(stack.pop())
# will cause an IndexError
# as the stack is now empty
```

15. What operations can you do with a stack?

A stack is a linear data structure that works on the same idea as a list, except that in a stack, components are only added and deleted from one end, the TOP. As a result, a stack is known as a LIFO (Last-In-First-Out) data structure, because the last component inserted is the first component removed.
A stack can carry out three basic operations:

1. PUSH: When you use the push action, a new element is added to the stack. The new function is at the top of the priority list. However, before we insert the value, we must first verify if TOP=MAX–1, since if it is, the stack is filled and no more insertions are possible.
2. POP: To remove the topmost member of a stack, use the pop action. We must first verify if TOP=NULL before deleting the item, because if it is, the stack is empty and no further deletions are permitted. You'll get a UNDERFLOW notice if you try to remove a value from a stack that is already empty.

16.What is the definition of a queue Data Structure?

The FIFO action is used to retrieve queue entries, which is an abstract data type that specifies a linear data structure or an ordered list. Only one end, referred to as REAR, is allowed to insert data, while the other end, referred to as FRONT, is only allowed to delete data.

17. List a few uses for the queue Data Structure.

Consider the following job prioritisation scenarios:

- Waiting lists for a single shared resource, such as a printer, CPU, call centre systems, or photo uploads, in which the first one to arrive is processed first.
- Asynchronous data transfer is exemplified by pipes, file IO, and sockets.
- As buffers in applications such as MP3 players and CD players
- To keep the media players' playlists up to date (to add or remove the songs)

18.What is the definition of a doubly-linked list? Give some specific examples.

It's a type of linked list (double-ended LL) in which each node has two links, one to the next node in the sequence and the other to the previous node in the series. This allows traversal of the data elements in both directions.
Here are several examples:

- A music playlist with next and previous track navigation options.
- RECENTLY VISITED PAGES IN THE BROWSER CACHÉ
- Undo and redo functions in the browser, which let you to return to a previous page by reversing the node.

19.In Python, how do you randomise the entries in a list while it's running?

Consider the following scenario:

```
from random import shuffle
x = ['Keep', 'The', 'Blue', 'Flag', 'Flying', 'High']
shuffle(x)
print(x)
```

The following code produces the following result.

['Flying', 'Keep', 'Blue', 'High', 'The', 'Flag']

20 In Python, what is a dictionary?

TOne of Python's built-in datatypes is the dictionary datatype. It creates a one-to-one relationship between keys and values. A dictionary entry consists of two keys and associated values. Dictionary indexing is done via keys.

Consider the case below:
The sample below has a few keys. The Prime Minister, the Country, and the Capital. India, Delhi, and Modi are all values that are mutually exclusive.

```
dict={'Country':'India','Capital':'Delhi','PM':'Modi'}
print dict[Country]
```

Output:India

21 What are the benefits of NumPy arrays over (nested) Python lists?

1. Python's lists are versatile containers that can be used for a number of tasks. They make insertion, deletion, appending, and concatenation (relatively) rapid, and Python's list comprehensions make them straightforward to create and use.
2. They have several limitations: they don't support "vectorized" operations like elementwise addition and multiplication, and because they can contain objects of various types, Python must store type information for each element and perform type dispatching code when working on it.
3. NumPy is not only more efficient, but it's also easier to use. You get a number of vector and matrix operations for free, which might save you time by avoiding unnecessary effort. They are also applied effectively.
4. NumPy arrays are faster, and NumPy includes a variety of tools, including as FFTs, convolutions, fast searching, basic statistics, linear algebra, histograms, and more.

22. In Python, how do you add values to an array?

The append(), extend(), and insert (i,x) procedures can be used to add elements to an array.

```
a=arr.array('d', [1.1 , 2.1 ,3.1] )
a.append(3.4)
print(a)
a.extend([4.5,6.3,6.8])
print(a)
a.insert(2,3.8)
```

```
print(a)
```

Output:

```
array(‘d’, [1.1, 2.1, 3.1, 3.4])
array(‘d’, [1.1, 2.1, 3.1, 3.4, 4.5, 6.3, 6.8]) array(‘d’, [1.1, 2.1, 3.8, 3.1, 3.4, 4.5, 6.3, 6.8])
```

23 What's the best way to remove values from a Python array?

The pop() and remove() methods can be used to remove elements from an array. The difference between these two functions is that the first returns the removed value, while the second does not.

```
a=arr.array('d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6])
print(a.pop())
print(a.pop(3))
a.remove(1.1)
print(a)
```

24.Write a program in Python to execute the Bubble sort algorithm.

```
def bs(a):
# a = name of list
  b=len(a)-1nbsp;
# minus 1 because we always compare 2 adjacent values
  for x in range(b):
      for y in range(b-x):
          a[y]=a[y+1]

  a=[32,5,3,6,7,54,87]
  bs(a)
```

25 .What are the benefits of using a heap instead of a stack?

Here are some examples of how to use heap over stack:

- The heap is more versatile than the stack because memory space can be dynamically created and de-allocated as needed.
- Stack variables are only visible to the user as private memory, but things created in the heap are visible to all threads.
- When using recursion, heap memory grows in size, whereas stack memory soon fills up.

26.What is the mechanism behind the Selection sort?

The smallest number from the list is repeatedly selected in ascending order and placed at the top of the selection sort. This operation is repeated as you get closer to the end of the list or sorted subarray.

Scan through all of the items to find the tiniest one. The first item in the position should be replaced. Every time we iterate forward I from 0 to N-1, we swap with the smallest element (always i).

Time complexity: best case O(n2); worst O(n2)

Space complexity: worst O(1)

27 What examples of divide-and-conquer algorithms can you give?

Quicksort is the name of the sorting algorithm. The method selects a pivot element and rearranges the array components so that all items less important than the pivot element are moved to the left side and all elements more important than the pivot element are moved to the right side.

MergeSort is a sorting algorithm as well. The algorithm divides the array into two halves, sorts them in a recursive manner, and then connects the two halves. The purpose of "points that are closest together" is to locate the nearest pair of points in an x-y plane collection of points. The problem can be solved in O(n2) time by computing the distances between each pair of locations and comparing them to find the shortest distance.

28 What is Graph Data Structure?

It's a non-linear data structure that stores and retrieves data by connecting vertices or nodes with edges or arcs. Edges are divided into two categories: directed and undirected.
Using a dictionary with the names of each vertex as the key and the edges list as the values is the best way to implement graphs in Python.

```
# Create the dictionary with graph elements
graph = { "a" : ["b","c"],
              "b" : ["a", "d"],
              "c" : ["a", "d"],
               "d" : ["e"],
               "e" : ["d"]
        }

# Print the graph
print(graph)
```

29.What are some examples of graph Data Structure applications?

- In transport grids, stations are represented as vertices, and routes are represented as graph edges.
- Vertices represent connection locations, while edges represent the wires or pipes that connect them in a power or water utility graph.
- Use social network graphs to determine the flow of information and hotspots (edges and vertices)
- In neural networks, the vertices represent neurons, while the edges represent synapses between them.

30. What are the different kinds of trees?

- The Generic Tree: A generic tree is one whose hierarchy isn't restricted. All other trees are subsets of the General Tree, which can have an endless number of progeny.

- The binary tree is a type of tree in which each parent has at least two offspring. The left and right youngsters are referred to as the left and right youngsters. The name of this tree is more well-known than that of the others. When a Binary tree is given specific restrictions and features, trees like the AVL tree, BST (Binary Search Tree), RBT tree, and others are utilised.

- BST (Binary Search Tree) is a binary tree extension with a variety of constraints. In BST, a node's left child value must be less than or equal to the parent value, whereas the correct child value must always be larger than or equal to the parent's value.

- AVL Tree: Adelson-Velshi and Landis, the two inventors, are represented by the initials AVL. This was the first tree to achieve dynamic equilibrium. Each node is given a balancing factor based on whether the AVL tree is balanced or not. The node kids can only reach a maximum height of one AVL vine.

- Red and Black Tree: The red-black tree is another type of auto-balancing tree. The term red-black is derived from the characteristics of a red-black tree, which has red or black painted on each node. It contributes to the forest's overall balance. Even though this tree isn't perfectly balanced, the search method takes only O (log n) time.

- N-ary Tree: In this sort of tree with a node, the maximum number of children is N. A binary tree is a two-year tree since each binary tree node has just two offspring. A full N-ary tree is one where each node's children are either 0 or N.

31. What are the distinctions between the B and B+ trees?

The B tree is a self-balancing m-way tree, with m denoting the tree's order. Btree is an extension of the Binary Search tree in which, depending on the number of m, a node can contain more than one key and more than two children. The data is organised in a B tree, with lower values on the left and higher values on the right subtrees. The B+ tree is an advanced self-balanced tree since every path from the tree's root to its leaf is the same length. The leaf nodes all occur at the same level since they are all the same length. At the third level, some leaf nodes are not permitted to develop, while others are permitted.

32 .What's the difference between a BFS (Breadth First Search) and a DFS (Depth First Search)?

1.BFS and DFS are two graph traversing algorithms. Graph traversal is the process of visiting all of the nodes in a graph.
2. BFS examines level by level, whereas DFS takes a path from the start to the end node, then another path from the start to the end, and so on until all nodes have been visited.
3. In addition, BFS uses a queue data structure to store nodes, whereas DFS uses a stack data structure to implement node traversal.

4. DFS produces deeper, non-optimal answers, but it works well when the solution is dense, whereas BFS produces ideal solutions.

33. What is Linked List? Implement linked list in python.

Linked lists are a sort of data structure in which each data node has a relational pointer that connects it to the next node in the list.
Unlike arrays, linked lists do not have objective positions in the list. They have relative positions instead, which are determined by the nodes in their immediate surroundings.
In a linked list, the head node is the first node, and the tail node, which has a null pointer, is the last.

The linked list is either singularly or doubly linked depending on whether each node has a single pointer to the next node or a second pointer to the previous node. Individual links in linked lists are similar to those in a chain in that they only connect to their immediate neighbours, but when all of the links are combined together, they form a larger structure.
You'll need to develop a Node class to hold a data value and one or more pointers because Python doesn't have a built-in implementation of linked lists.

```python
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

class SLinkedList:
    def __init__(self):
        self.headval = None

list1 = SLinkedList()
list1.headval = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")
# Link first Node to second node
list1.headval.nextval = e2

# Link second Node to third node
e2.nextval = e3
```

34. What are some of the uses of the graph data structure?

The graph can be used for the following purposes:

- In circuit networks, vertices are shown as points of connection, and component cables are drawn as edges of the graph.
- In transportation networks, stations are shown as vertices while routes are drawn as edges of the graph.
- Cities/states/regions are drawn as vertices on maps, while adjacency relations are drawn as edges.

- Procedures or modules are treated as vertices in programme flow analysis, and calls to these procedures are shown as edges of the graph.

35. What are some of the uses of the Tree-data structure?

Tree-data structure applications include:

- The art of manipulating arithmetic expressions,
- Construction of the Symbol Table
- Analyze the syntax
- Hierarchal data model

36. Write a programme to generate a third list l3 from two lists, l1 and l2, by selecting an odd-index element from l1 and even-index elements from l2.

To access a range of items in a list, use the slicing operator :. With this operator, you can specify where to start the slicing, end, and specify the step.

```python
list1 = [3, 6, 9, 12, 15, 18, 21]
list2 = [4, 8, 12, 16, 20, 24, 28]
res = list()

odd_elements = list1[1::2]
print("Element at odd-index positions from list one")
print(odd_elements)

even_elements = list2[0::2]
print("Element at even-index positions from list two")
print(even_elements)

print("Printing Final third list")
res.extend(odd_elements)
res.extend(even_elements)
print(res)
```