Data Structure

1. ARRAY:-

Arrays are collection of elements stored at contigious memory locations, indexed by integers.

creating an array :-

arr[1, 2, 3, 4, 5]

accessing elements

print (arr[0]) :-

∴ output = 1

updating elements

arr[0] = 10

length of array

print (len(arr))

∴ output = 5

Notes By :- @jpwebdevelop

2. LISTS :-

list are versatile collection in python, capable of holding heterogeneous data types.

```
# creating a list
my_list = [1, 'hello', 3.14, true]
# accessing elements
print (my_list[1])
∴ output = hello
# updating elements
my_list[0] = 10
# length of list
print (len (my_list))
∴ output = 4
```


3. LINKED LIST :-

Linked list are linear data structure where each element is a seperate object.

```
class Node:
    def __init__(self, data):
        self.data = data
```

```
        self.next = None

# creating nodes

    node1 = Node(1)
    node2 = Node(2)
    node3 = Node(3)

# linking nodes

  node1.next = node2
  node2.next = node3
```


4° STACK :-

stack follow the last in first out (LIFO) principle.

```
# using list as stack

    stack = []

# pushing elements

    stack.append(1)
    stack.append(2)

# popping elements

print(stack.pop())
                ∴ output = 2
```

@jp Notes

5° QUEUES:-

Queues follow the first in first out (fifo) principle.

```
from collection import deque
# creating a queue
queue = deque
# enqueuing elements
queue.append(1)
queue.append(2)
# dequeuing elements
print(queue.popleft())
∴ output = 1
```

6° HASH TABLE:-

Hash table store key-value pairs and provide efficient lookup.

```
# creating a hash table
hash_table = {}
# adding elements
hash_table['abbu'] = 10
```

```
hash_table["banana"] = 20

# accessing elements
print(hash_table["apple"])
                    # output = 10
```

7. TREES :-

Trees are hierarchical data structure with nodes connected by edges.

```
class TreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# creating nodes
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
```


8. BINARY SEARCH :-

Binary search is a faster searching algorithm for sorted arrays by repeatedly dividing the

search interval in half

```
def binary_search(arr, target):
    low, high = 0, len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid + 1
    return -1
```



```
# Example usage
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
target = 5
print("binary_search(arr, target)
```

∴ output = 4

9. GRAPH:-

Graphs consist of vertices and edges that connect them.

```
# using dictionary to represent a graph

graph = {
```

```
'A' : ['B', 'C'],
'B' : ['C', 'D'],
'C' : ['D'],
'D' : ['C'],
'E' : ['F'],
'F' : ['C'],
}
```

@jp web developers