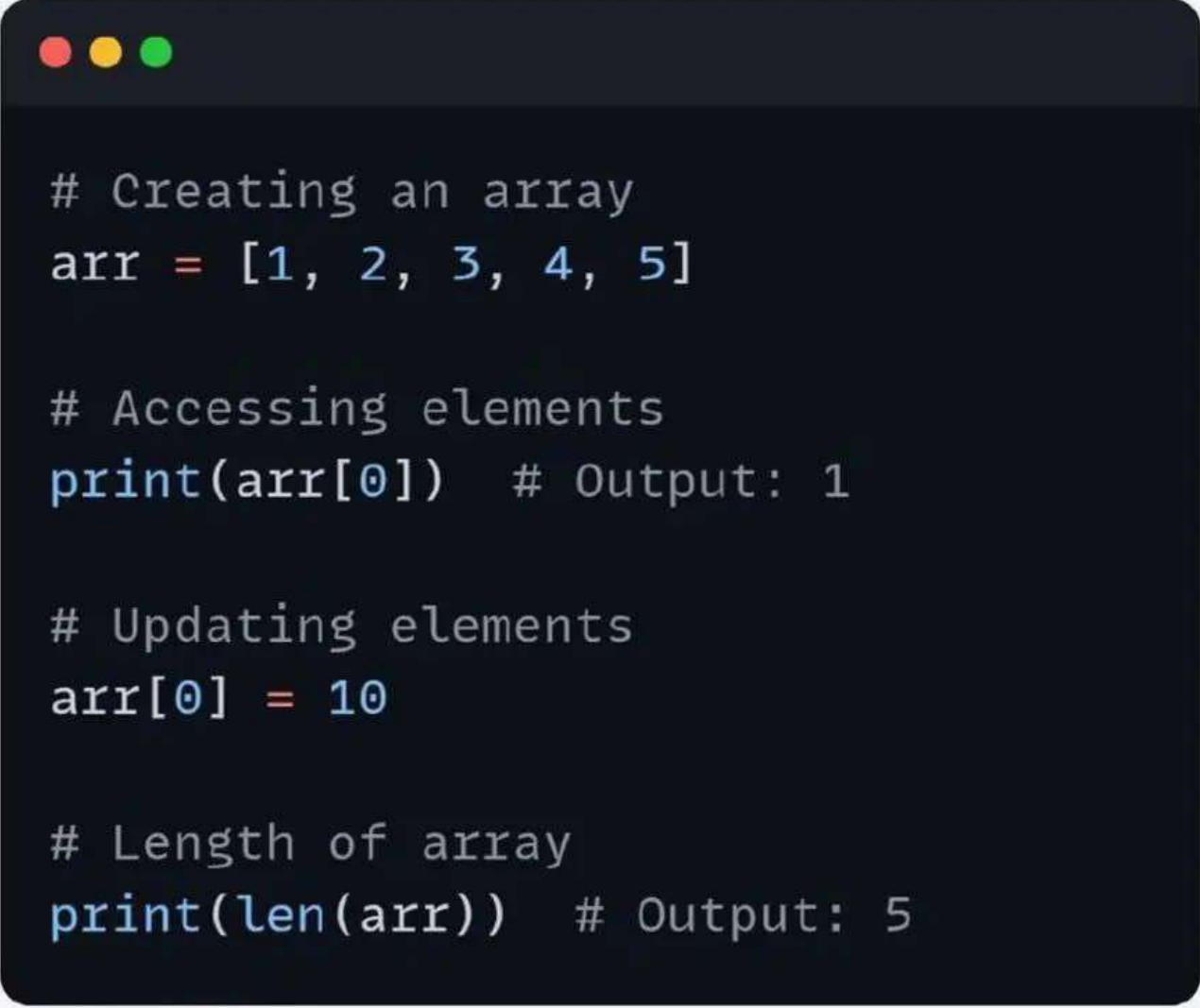


# DSA In Python



# 1. Arrays

Arrays are collections of elements, stored at contiguous 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
```

## 2. Lists

Lists are versatile collections 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 Lists

Linked lists are linear data structures where each element is a separate object.

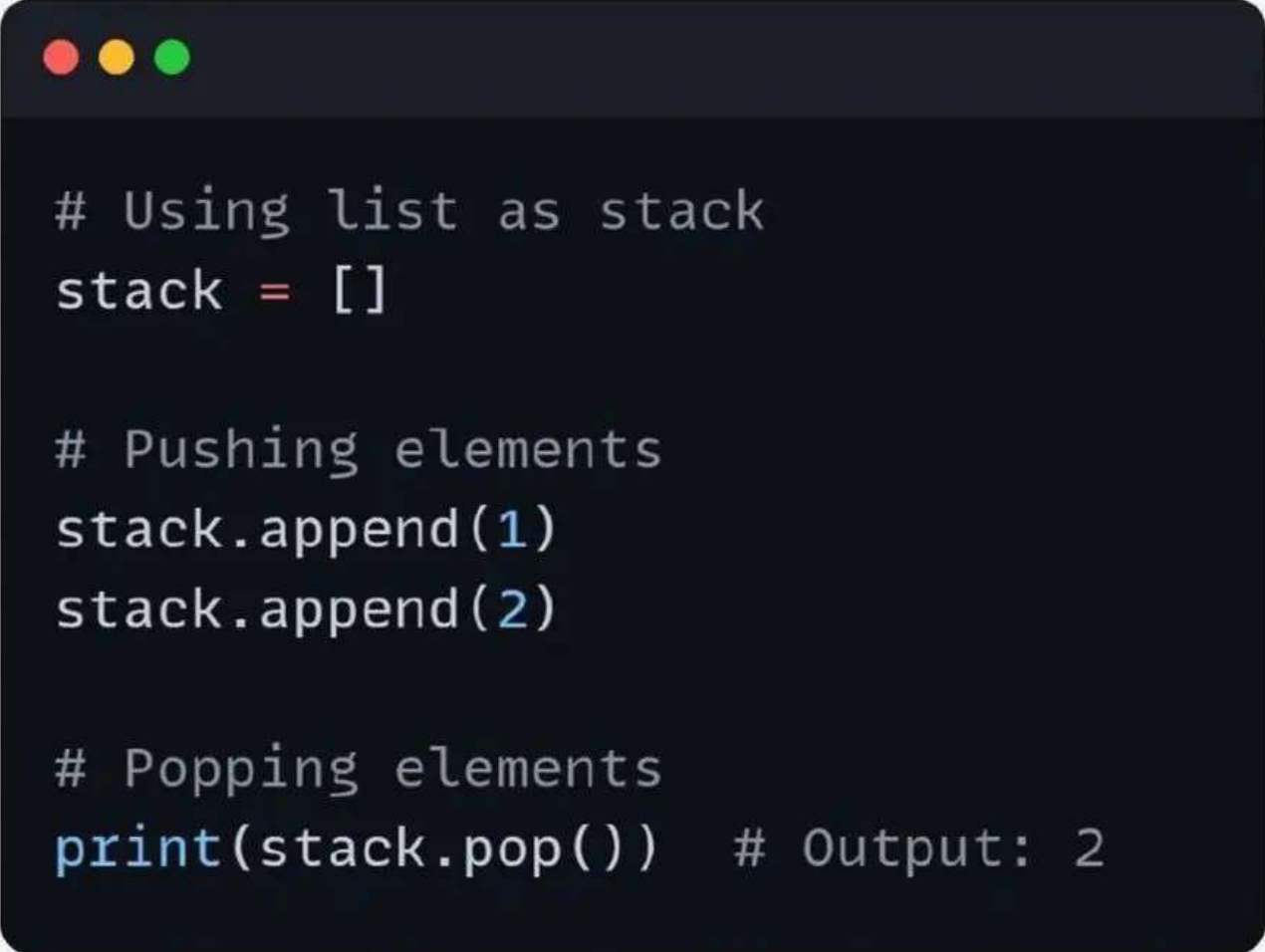
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# Creating nodes
node1 = Node(1)
node2 = Node(2)
node3 = Node(3)

# Linking nodes
node1.next = node2
node2.next = node3
```

## 4. Stacks

Stacks 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
```

## 5. Queues

Queues follow the First In First Out (FIFO) principle.



```
from collections import deque

# Creating a queue
queue = deque()

# Enqueueing elements
queue.append(1)
queue.append(2)

# Dequeueing elements
print(queue.popleft()) # Output: 1
```

## 6. Hash Tables

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

```
# Creating a hash table
hash_table = {}

# Adding elements
hash_table['apple'] = 10
hash_table['banana'] = 20

# Accessing elements
print(hash_table['apple']) # Output: 10
```



## 7. Trees

Trees are hierarchical data structures 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. Graphs

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']
}
```

**If You Want More Like All Sorting, Recursion, Dynamic Programming, etc.. In Python Please Comment**