Stack Implementation

```
class Stack:
    def __init__(self):
        self.stack = []
    def push(self, item):
        self.stack.append(item)
        print(item, "pushed")

    def pop(self):
        if self.is_empty():
            return "Stack is empty"
        return self.stack.pop()
    def peek(self):
        if self.is_empty():
            return "Stack empty"
        return self.stack[-1]
    def is_empty(self):
        return len(self.stack) == 0
s = Stack()
s.push(10)
s.push(20)
print("Top:", s.peek())
print("Pop:", s.pop())
```

```
10 pushed
20 pushed
Top: 20
Pop: 20
```

Task 2 — Queue Implementation

```
class Queue:
    def __init__(self):
        self.queue = []
    def enqueue(self, item):
        self.queue.append(item)
        print(item, "enqueued")
    def dequeue(self):
        if self.is_empty():
            return "Queue empty"
        return self.queue.pop(0)
    def front(self):
        if self.is_empty():
            return "Queue empty"
        return self.queue[0]
    def size(self):
        return len(self.queue)
    def is_empty(self):
        return len(self.queue) == 0
q = Queue()
q.enqueue(1)
q.enqueue(2)
print("Front:", q.front())
print("Dequeue:", q.dequeue())
```

```
1 enqueued
2 enqueued
Front: 1
Dequeue: 1
```

Task 3 — Singly Linked List

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        temp = self.head
        while temp.next:
```

```
            temp = temp.next
        temp.next = new_node
    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
ll = LinkedList()
ll.insert(10)
ll.insert(20)
ll.insert(30)
ll.display()
```

```
10 -> 20 -> 30 -> None
```

Task 4 — Binary Search Tree

```
class BSTNode:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key
def insert(root, key):
    if root is None:
        return BSTNode(key)
    if key < root.val:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)
    return root
def inorder(root):
    if root:
        inorder(root.left)
        print(root.val, end=" ")
        inorder(root.right)
root = None
for i in [50, 30, 70, 20, 40, 60]:
    root = insert(root, i)
print("Inorder Traversal:")
inorder(root)
```

```
Inorder Traversal:
20 30 40 50 60 70
```

Task 5 — Hash Table

```
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)]
    def hash_function(self, key):
        return key % self.size
    def insert(self, key, value):
        index = self.hash_function(key)
        self.table[index].append((key, value))
    def search(self, key):
        index = self.hash_function(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return "Not found"
    def delete(self, key):
        index = self.hash_function(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                del self.table[index][i]
                return "Deleted"
        return "Not found"
ht = HashTable()
ht.insert(1, "A")
ht.insert(11, "B")
print("Search:", ht.search(11))
ht.delete(1)
```

```
Search: B
'Deleted'
```