1. **Implement a Python class MaxHeap that supports the following operations: insert, delete, and get\_max. Ensure the operations maintain the properties of a max-heap.**

class MaxHeap:

def \_\_init\_\_(self):

self.heap = []

#Insert

def insert(self, val):

self.heap.append(val)

self.\_heapify\_up(len(self.heap) - 1)

#delete

def delete(self, val):

try:

index = self.heap.index(val)

self.heap[index] = self.heap[-1]

self.heap.pop()

if index < len(self.heap):

self.\_heapify\_down(index)

self.\_heapify\_up(index)

except ValueError:

print("Value not found in heap")

#get\_max

def get\_max(self):

return self.heap[0] if self.heap else None

def \_heapify\_up(self, index):

parent\_index = (index - 1) // 2

if index > 0 and self.heap[index] > self.heap[parent\_index]:

self.heap[index], self.heap[parent\_index] = self.heap[parent\_index], self.heap[index]

self.\_heapify\_up(parent\_index)

def \_heapify\_down(self, index):

largest = index

left\_child = 2 \* index + 1

right\_child = 2 \* index + 2

if left\_child < len(self.heap) and self.heap[left\_child] > self.heap[largest]:

largest = left\_child

if right\_child < len(self.heap) and self.heap[right\_child] > self.heap[largest]:

largest = right\_child

if largest != index:

self.heap[index], self.heap[largest] = self.heap[largest], self.heap[index]

self.\_heapify\_down(largest)

def \_\_str\_\_(self):

return str(self.heap)

# Example usage

heap = MaxHeap()

heap.insert(10)

heap.insert(20)

heap.insert(5)

heap.insert(30)

print("Heap after inserts:", heap)

max\_val = heap.get\_max()

print("Max value:", max\_val)

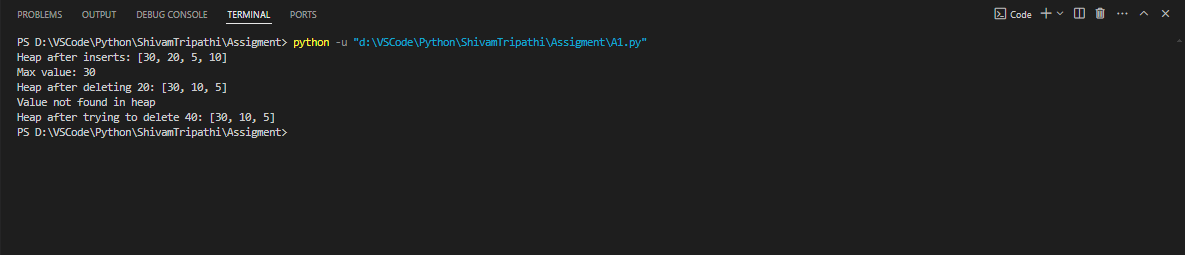
heap.delete(20)

print("Heap after deleting 20:", heap)

heap.delete(40) # Value not in heap

print("Heap after trying to delete 40:", heap)

**OUTPUT:**

****