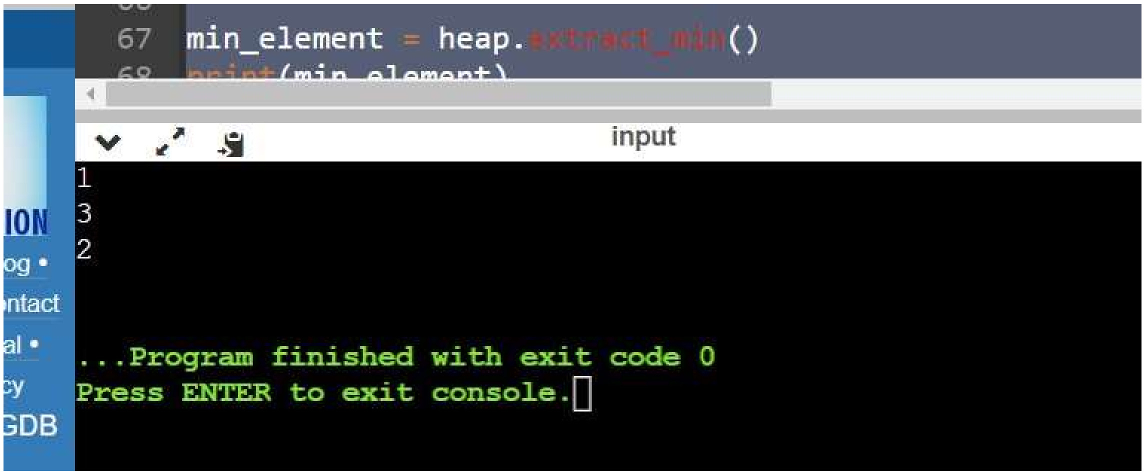
Day – 12 Heaps

Problem 1: Min Heap & Max Heap

1. **Min Heap**

```
class MinHeap:
    def___init__(self):
        self.heap = []

    def parent(self, i):
        return (i – 1) // 2

    def left_child(self, i):
        return 2 * i + 1

    def right_child(self, i):
        return 2 * i + 2

    def swap(self, i, j):
        self.heap[i], self.heap[j] = self.heap[j], self.heap[i]

    def insert(self, item):
        self.heap.append(item)
        self.heapify_up(len(self.heap)  –  1)

    def  extract_min(self):
        if len(self.heap) == 0:
            return None

        min_item = self.heap[0]
        self.swap(0, len(self.heap) – 1)
        self.heap.pop()
        self.heapify_down(0)
        return min_item

    def heapify_up(self, i):
        while i > 0 and self.heap[i] < self.heap[self.parent(i)]:
            self.swap(i, self.parent(i))
            i = self.parent(i)

    def heapify_down(self, i):
        smallest = i
        left = self.left_child(i)
        right = self.right_child(i)

        if left < len(self.heap) and self.heap[left] < self.heap[smallest]:
            smallest = left

        if right < len(self.heap) and self.heap[right] < self.heap[smallest]:
            smallest = right

        if smallest != i:
            self.swap(i, smallest)
            self.heapify_down(smallest)

heap = MinHeap()

heap.insert(5)
heap.insert(3)
heap.insert(8)
heap.insert(1)
heap.insert(10)
```

```
min_element = heap.extract_min()
print(min_element)

min_element = heap.extract_min()
print(min_element)

heap.insert(2)

min_element = heap.extract_min()
print(min_element)
```

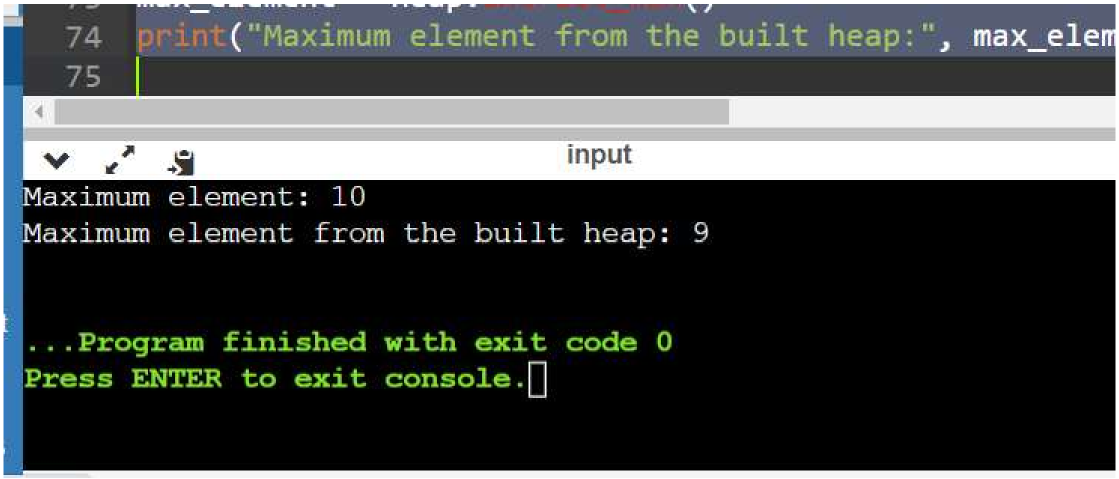


```
min_element = heap.extract_min()
print(min_element)
```

```
1
3
2


...Program finished with exit code 0
Press ENTER to exit console.
```

2. Max Heap

```
class MaxHeap:
  def __init_(self):
    self.heap = []

  def parent(self, i):
    return (i - 1) // 2

  def left_child(self, i):
    return 2 * i + 1

  def right_child(self, i):
    return 2 * i + 2

  def swap(self, i, j):
        self.heap[i], self.heap[j] = self.heap[j], self.heap[i]

    def insert(self, value):
        self.heap.append(value)
        current = len(self.heap) - 1
        while (
            current > 0
            and self.heap[current] > self.heap[self.parent(current)]
        ):
            self.swap(current, self.parent(current))
            current = self.parent(current)

    def heapify(self, n, i):
        largest = i
        left = self.left_child(i)
        right = self.right_child(i)

        if left < n and self.heap[left] > self.heap[largest]:
            largest = left

        if right < n and self.heap[right] > self.heap[largest]:
            largest = right

        if largest != i:
            self.swap(i, largest)
            self.heapify(n, largest)

    def build_heap(self, arr):
        n = len(arr)
        self.heap = arr
        for i in range(n // 2 - 1, -1, -1):
            self.heapify(n, i)

    def extract_max(self):
        if len(self.heap) == 0:
            return None

        max_value = self.heap[0]
        self.heap[0] = self.heap[-1]
        self.heap.pop()
        self.heapify(len(self.heap), 0)
        return max_value

# Create a new max heap
heap = MaxHeap()

heap.insert(5)
heap.insert(10)
heap.insert(3)
heap.insert(8)
heap.insert(1)

max_element = heap.extract_max()
print("Maximum element:", max_element)

arr = [7, 2, 9, 4, 6]
heap.build_heap(arr)

max_element = heap.extract_max()
print("Maximum element from the built heap:", max_element)
```



```
print("Maximum element from the built heap:", max_elem
```

```
Maximum element: 10
Maximum element from the built heap: 9


...Program finished with exit code 0
Press ENTER to exit console.
```