Implementing a Binary Heap: Takeaways

Syntax

- Implementing a max-heap:

```
class MaxHeap:
def __init__(self):
    self.nodes = []
def insert(self, value):
    self.nodes.append(value)
    index = len(self.nodes) - 1
    parent_index = math.floor((index-1)/2)
    while index > 0 and self.nodes[parent_index] < self.nodes[index]:
        self.nodes[parent_index], self.nodes[index] = self.nodes[index],
self.nodes[parent_index]
        index = parent_index
        parent_index = math.floor((index-1)/2)
    return self.nodes
```

- Returning a list of the top 100 elements:

```
heap = MaxHeap()
class MaxHeap(BaseMaxHeap):
    def top_n_elements(self, n):
        return [self.pop() for _ in range(n)]
heap = MaxHeap()
heap.insert_multiple(heap_list)
top_100 = heap.top_n_elements(100)
```

- Using Python's heap to return the top 100 elements:

```
top_100 = heapq.nlargest(100, heap_list, key=lambda x: x[1])
```

- Using Python's heap to return the bottom 100 elements:

```
bottom_100 = heapq.nsmallest(100, heap_list, key=lambda x: x[1])
```

Concepts

- A binary heap is a version of a complete binary tree.
- A binary tree is complete if the tree's levels are filled in except for the last that has nodes filled in from left to right.
- A binary heap, or heap, requires the two following conditions to hold:
    - It must be a complete binary tree.
    - The value of a parent is greater than or equal to OR less than or equal to any of its child nodes.
- Categories of heaps:
    - If it is a max-heap, than each of the parent nodes is greater than or equal to any of its child nodes.
    - If it is a min-heap, each of the parent nodes is less than or equal to any of its child nodes.
- `heapq` is Python's own heap implementation and works both as a max-heap and min-heap.

Resources

- Binary Heap
- Heap queue algorithm