Heaps can be used as priority queues. The special property of both min-heap and max-heap allows us to always access the highest (or smallest) priority element in constant time (O(1)), and that's critical for a priority queue because accessing the highest (or lowest) priority element is a common operation.

Here's how heaps work as priority queues in pseudo code, assuming we're using a min heap (i.e., smallest element has the highest priority):

```
minHeap = new Heap()  // Create a new heap

minHeap.add(10)  // Add some elements to it
minHeap.add(20)
minHeap.add(15)

minPriorityItem = minHeap.peek()  // Get the min (highest priority) element. It's 10.

minPriorityItem = minHeap.poll()  // Remove the min element and get it. It's 10.

minPriorityItem = minHeap.peek()  // Now, the min element is 15.
```

As for the second question, we can make a max heap work like a min heap (or vice versa) using a little trick: we insert the negative of each number into the heap. When we retrieve numbers, we again take its negative.

python:
```
import heapq

# normally, heapq in python works as min heap
minHeap = []
heapq.heappush(minHeap, 10)
heapq.heappush(minHeap, 20)
heapq.heappush(minHeap, 15)
print(heapq.heappop(minHeap))  # prints 10

# Now, let's make it work as a max heap by inserting negatives
maxHeap = []
heapq.heappush(maxHeap, -10)
heapq.heappush(maxHeap, -20)
heapq.heappush(maxHeap, -15)

# when popping, take the negative again
print(-heapq.heappop(maxHeap))  # prints 20, which is the max.
```

This trick negates the numbers as we distribute them and pops out the maximum value when wanted. This way, we don't need to write a whole new implementation for the heap. The same can be done vice versa for converting min heap into max heap.