

**“AZƏRBAYCAN HAVA YOLLARI” CJSC NATIONAL AVIATION ACADEMY**

**Individual Work № 3:**

**Topic:** Lists. Sets. Stacks. Queues. Time complexity of different operations.

**Subject: Obyektyönümlü proqramlaşdırma**

**Teacher: Mamed Shahmaliyev**

**Group: 1459i Student: Azadaliyev Rustam**

**Date: 11.10.2021 Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_**

**What are Heaps in Python**?

Heaps in Python are complete binary trees in which each node is either smaller than equal to or greater than equal to all its children (smaller or greater depending on whether it is a max-heap or a min-heap).

Hence the root node of a heap is either the smallest or the greatest element. **The heap data structure is generally used to represent a priority queue**.

Generally, Heaps are of two forms:

* **Min-Heap**: Min heap is the heap in which all nodes are lesser than their children. The root contains the lowest value in a min-heap.
* **Max-Heap**: Max heap is the heap in which all nodes are greater than their children. The root contains the highest value in a max-heap.

Following is the example for min heap and max heap.

Heaps in Python, by default, are Min-heaps, and further in this article, we will be considering min-heap when we talk about heap. Let us now see how the heap data structure is actually implemented.

## ****How are heaps represented in Python?****

The heap data structure is theoretically in the form of a binary tree, but due to its property of completeness (wherein the tree is complete except for the rightmost nodes in the last layer), the heap is stored in the form of an array in the memory. The first element contains the minimum element (in the case of min-heap).

The heap, which is in the form of a tree, is stored in the array, and its elements are indexed in the following manner:

* The root element will be at the 0th position of the array, that is, Heap[0].
* For any other node, say Heap[i], we have the following:
  + The parent node is given by : Heap[(i -1) / 2].
  + The left child node is given by : Heap[(2 \* i) + 1]
  + The right child node is given by : Heap[(2 \* i) + 2]

## ****Using the heapq module to implement heaps in Python****

Python has the”heapq” module for the implementation of Heap Queue (or simply heap). It contains the functionality that the smallest element will always be at the top and will be popped when called the pop function.

Whenever elements are either pushed or popped, heap property will be maintained, and heap[0] will always give us the smallest function.

The module contains the following major functions for heap:

* **heapify**( iterable\_name ): We use this function to pass any iterable (for example a list) and it converts it into a heap data structure.
* **heappush**( heap\_name, element\_to\_be\_inserted ): As the name suggests, this function pushes/ adds an element to the heap. We need to pass the heap name and element to be inserted as a parameter. The function takes care of rearranging the heap (if need be) to satisfy the heap property.
* **heappop**( heap\_name ): As the name suggests, this function pops/removes an element from the heap passed as a parameter. The function takes care of rearranging the heap (if need be) to satisfy the heap property.

**Implementation of heaps in Python:**

|  |
| --- |
| # The heap functionalities are in the heapq package, so import it  import heapq  # we now initialise a list to be converted to heap  lis = [15, 7, 9, 4, 13]    # converting lis to heap using the heapify function  heapq.heapify(lis)  print ("The heap looks like: ")  print(lis)    #using the heappop function  print ("The popped item using heappushpop() is : ",end="")  print (heapq.heappop(lis))    print ("After popping, the heap looks like: ")  print(lis)    #using the heappush function to push 2  print ("After pushing 2, the heap looks like: ")  heapq.heappush(lis, 2)  print(lis) |

Output:

|  |
| --- |
| The heap looks like:  [4, 7, 9, 15, 13]  The popped item using heappop() is : 4  After popping, the heap looks like:  [7, 13, 9, 15]  After pushing 2, the heap looks like:  [2, 7, 9, 15, 13] |

Here, we can see that the heapq package provides functionalities to create a queue, and push and pop elements to it. After pushing or popping, the heap automatically rearranges itself, as was seen in the output.