CS 2302 Lab Report

Lab 5\_Option B

Jose Lujan

Lab 5 Option B

Min-Heap

**Objective of the Lab:**

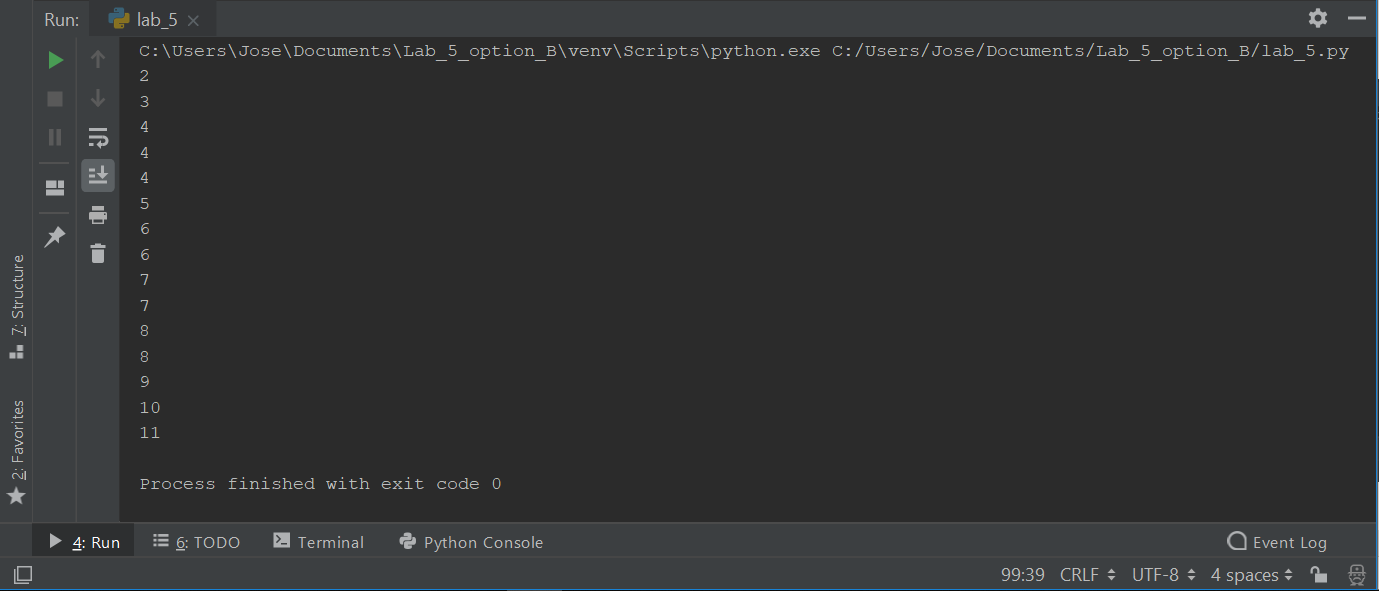
The main objective is to create a heap, in this case a min-heap. The heap will be implemented using the code that the professor gave us. To test the lab we have three different options. The option we decide to do, will have to print as an output the heap sorted.

**Solution (code):**

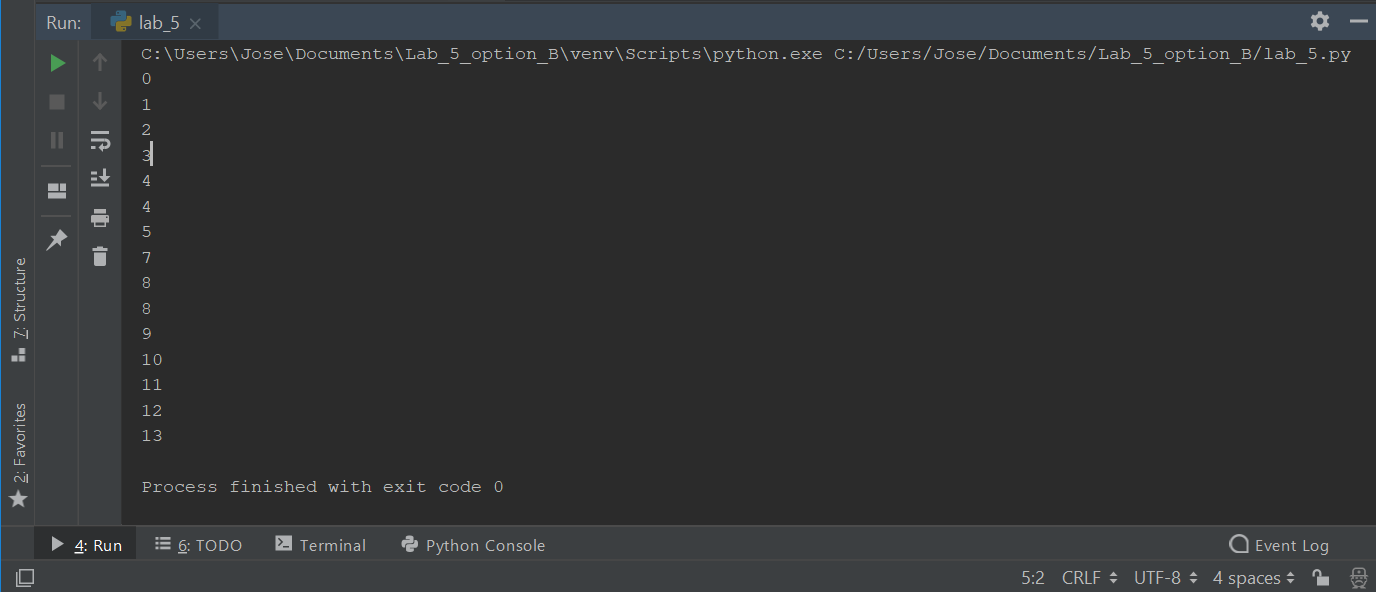
from lab\_5 import Heap  
from lab\_5 import heap\_sort  
  
# heap to populate  
heap = Heap()  
  
# hard coded list of values  
list = [7, 4, 8, 10, 4, 6, 9, 5, 6, 3, 5, 11, 7, 8, 2]  
  
# populate the heap  
for elem in list:  
 heap.insert(elem)  
  
# apply heapsort to the heap and return the sorted array  
result = heap\_sort(list)  
  
# print the values in the sorted array  
for i in result:  
 print(i)

class Heap:  
 def \_\_init\_\_(self):  
 self.heap\_array = []  
  
 def insert(self, k):  
 self.heap\_array.append(k)  
  
 # *TODO: Complete implementation* self.sort\_up(len(self.heap\_array) - 1)  
  
 def extract\_min(self):  
 if self.is\_empty():  
 return None  
  
 min\_elem = self.heap\_array[0]  
  
 # *TODO: Complete implementation* # move the last value in the array to the first index  
 last\_value = self.heap\_array.pop()  
 if len(self.heap\_array) > 0:  
 self.heap\_array[0] = last\_value  
  
 self.heap\_sort\_sort\_down(0)  
  
 return min\_elem  
  
 def is\_empty(self):  
 return len(self.heap\_array) == 0  
  
 # Used in the insert function to sort the heap after an element has been inserted  
 def sort\_up(self, node\_index):  
 # sort the heap starting from the node that passed into the method and traverse through its parents  
 while node\_index > 0:  
 # get the index of the parent of the node  
 parent\_index = (node\_index - 1) // 2  
  
 # if the value of the parent is larger, make the switch  
 if self.heap\_array[node\_index] < self.heap\_array[parent\_index]:  
 temp = self.heap\_array[node\_index]  
 self.heap\_array[node\_index] = self.heap\_array[parent\_index]  
 self.heap\_array[parent\_index] = temp  
  
 node\_index = parent\_index  
 else:  
 return  
  
 def heap\_sort\_sort\_down(self, node\_index):  
 child\_index = 2 \* node\_index + 1  
 value = self.heap\_array[node\_index]  
  
 while child\_index < len(self.heap\_array):  
 min\_value = value  
 min\_index = -1  
 i = 0  
 while i < 2 and (i + child\_index) < len(self.heap\_array):  
 if self.heap\_array[i + child\_index] < min\_value:  
 min\_value = self.heap\_array[i + child\_index]  
 min\_index = i + child\_index  
 i += 1  
  
 if min\_value == value:  
 return  
  
 temp = self.heap\_array[node\_index]  
 self.heap\_array[node\_index] = self.heap\_array[min\_index]  
 self.heap\_array[min\_index] = temp  
  
 node\_index = min\_index  
 child\_index = 2 \* node\_index + 1  
  
  
def create\_list\_from\_file(file):  
 file = open(file)  
 list = []  
 for line in file:  
 list\_line = line.split(",")  
 for num in list\_line:  
 list.append(int(num))  
  
 return list  
  
  
def heap\_sort(list):  
 heap = Heap()  
 result = []  
  
 # heapify the list  
 for elem in list:  
 heap.insert(elem)  
  
 while not heap.is\_empty():  
 result.append(heap.extract\_min())  
  
 return result  
  
  
def main():  
 list = create\_list\_from\_file("Test1.txt")  
  
 result = heap\_sort(list)  
  
 for i in result:  
 print(i)  
  
  
main()

**Testing with files: #1**



**Testing with file #2**



**Conclusion:**

To verify that we created the heap correctly. I created two text files, which I used to test the code. The program will open the file and it splits the numbers when we see a “,” it creates a new node. Once the programs had separated all the numbers from the file. It will populate the Heap, with the root being the smaller number. We have different options to test the lab. In my case I decided to create a new file, that I can just change the name of the file.