This lab gets you to work with the array and count implementation of array lists. It also briefly looks at priority queues.

Marks (max 5): Questions 1-2: 1 | Questions 1-5: 3 | Questions 6-7: 1 each

For the first 5 questions, you are asked to work on the Python implementation of the `ArrayList` class that we saw in the lecture (week 6). The code is in `lab7.ipynb`.

Question 1

Add a function

```
def appendAll(self, A)
```

that appends all elements of the array `A` in the array list (the one represented by `self`). For example, if `ls` is [2,3,4,5] then `ls.appendAll([42,24])` changes `ls` to [2,3,4,5,42,24].

Question 2

Add a function

```
def removeVal(self, e)
```

that removes the first occurrence of `e` and returns `True`, or returns `False` and does not change the array list if `e` is not in it.

Question 3

Add a function

```
def clone(self)
```

that returns a new array list containing the same elements as the array list represented by `self`. Note that the new and the old array lists should be separate copies, i.e. changing one of them should not affect the other one.

Question 4

Add a function

```
def toArray(self)
```

that returns a new array containing the same elements as the array list represented by `self`. Note that the length of the array should be the same as that of the array list, and not the same as that of its internal array.

Question 5

Modify the functions `get`, `set`, `remove` and `insert` so that they throw an exception if the input `i` is out of the bounds of the array list. To help you in this, you can use the method `_checkBounds(self,i,hi)` that we have provided, and which checks whether `i` is between 0 and `hi` (inclusive) and, if this is not the case, throws an exception.

Question 6

Add a function

```
sort(self)
```

that sorts the elements in the array list using insertion sort.

Note that the function should only sort the elements in the array list, not the whole of `inArray`. That is because `inArray` has many "garbage" elements that, if sorted in position, will essentially ruin the array list.

For the next question, we look at another data structure, namely priority queues. A priority queue is a queue in which each element has a priority, and where dequeueing always returns the item with the greatest priority in the queue.

We start by defining a class of priority queue elements (PQ-elements for short):

```
class PQElement:
    def __init__(self, v, p):
        self.val = v
        self.priority = p
```

So, a PQ-element is a pair consisting of a value (which can be anything, e.g. an integer, a string, an array, etc.) and a priority (which is an integer).
In `lab7.ipynb` we also implemented the `__str__` function to be able to print PQ-elements.

Question 7

Write a Python class `PQueue` that implements a priority queue using an array list of `PQElement`'s. In particular, you need to implement 5 functions:

- one for creating an empty priority queue
- one for returning the size of the priority queue
- one for enqueueing a new PQ-element in the priority queue
- one for dequeueing from the priority queue the PQ-element with the greatest priority
- one that prints the elements of the priority queue into a string (call this one `__str__`)

Test each of the functions on examples of your own making. For example, running:

```
ls = PQueue()
for i in range(15):
    ls.enq(PQElement(i,10-i))
print(ls)
print(ls.deq(),ls)
```

should give this printout:

```
[(14,-4),(13,-3),(12,-2),(11,-1),(10,0),(9,1),(8,2),(7,3),(6,4),(5,5),(4,6),(3,7),(2,8),(1,9),(0,10)]
(0,10) [(14,-4),(13,-3),(12,-2),(11,-1),(10,0),(9,1),(8,2),(7,3),(6,4),(5,5),(4,6),(3,7),(2,8),(1,9)]
```

Hint: you can start by adapting the class `ArrayList` so that it implements an array list of `PQElement`'s (actually, there is not so much to adapt). Accordingly, you can modify the class `Queue` that we saw in the lecture exercises so that it implements a queue of `PQElement`'s. You then need to modify the latter so that dequeueing always removes the element with the highest priority. For full marks, dequeueing should have complexity $\Theta(1)$.