Python programming
Lab. Work n° 7 : Plant a seed, you may got a tree

Preliminaries : Please remind that teachers can be called to help you on any problem you get. Don't get stuck on an issue for too long.

Exercise n°1 : Simple trees

In this exercise a tree is a recursive structure made of node. A node is a list of tree elements :
- the first being the value (the element) represented by this node,
- the second being the left sub-tree
- the third being the right sub-tree

So that in the left (respectively right) sub-tree of any node, you can only find elements that are less (resp. greater or equal) than the value stored in that node. The empty tree is `None`

1. Create a module `tree.py`

2. Write a function `is_in(t, e)` that tests (returns `True` or `False`) if e is in t. Test it with the following given trees :

```
None
[5, None, None]
[6, [1, [0, None, None], [3, None, [3, None, [5, [3, None, None], None]]]], [7, ▷
    ▷ None, [8, [7, None, None], None]]]
[7, [0, None, [3, [1, None, [1, None, None]], [6, None, [6, None, None]]]], [9, ▷
    ▷ [8, None, None], [9, None, None]]]
[5, [2, [0, None, None], [2, None, [3, [2, None, None], None]]], [9, [6, None, ▷
    ▷ None], [9, None, [10, None, None]]]]
```

Note : that you may store these trees into a list of trees, which is then called a forest. For each of these trees, test if $e \in [0, 10[$ is in the tree, like :

```
Tree None
0 is in?  False
1 is in?  False
2 is in?  False
3 is in?  False
4 is in?  False
5 is in?  False
6 is in?  False
7 is in?  False
8 is in?  False
9 is in?  False
Tree [5, None, None]
0 is in?  False
1 is in?  False
2 is in?  False
3 is in?  False
4 is in?  False
5 is in?  True
6 is in?  False
```

```
7 is in?  False
8 is in?  False
9 is in?  False
Tree [6, [1, [0, None, None], [3, None, [3, None, [5, [3, None, None], None]]]], ▷
    ▷ [7, None, [8, [7, None, None], None]]]
0 is in?  True
1 is in?  True
2 is in?  False
3 is in?  True
4 is in?  False
5 is in?  True
6 is in?  True
7 is in?  True
8 is in?  True
9 is in?  False
Tree [7, [0, None, [3, [1, None, [1, None, None]], [6, None, [6, None, None]]]], ▷
    ▷ [9, [8, None, None], [9, None, None]]]
0 is in?  True
1 is in?  True
2 is in?  False
3 is in?  True
4 is in?  False
5 is in?  False
6 is in?  True
7 is in?  True
8 is in?  True
9 is in?  True
Tree [5, [2, [0, None, None], [2, None, [3, [2, None, None], None]]], [9, [6, ▷
    ▷ None, None], [9, None, [10, None, None]]]]
0 is in?  True
1 is in?  False
2 is in?  True
3 is in?  True
4 is in?  False
5 is in?  True
6 is in?  True
7 is in?  False
8 is in?  False
9 is in?  True
```

3. Write a function `insert(t, e)` that inserts element/value e in the tree t, and returning the new tree. This function may be used like this :

```
root = ...
root = insert(root, e)
```

Test it like this :

```
Size of the first list:
Tree None
Value to insert (-1 to stop)? 3
[3, None, None]
Value to insert (-1 to stop)? 2
[3, [2, None, None], None]
Value to insert (-1 to stop)? 6
[3, [2, None, None], [6, None, None]]
Value to insert (-1 to stop)? 5
```

```
[3, [2, None, None], [6, [5, None, None], None]]
Value to insert (-1 to stop)? 7
[3, [2, None, None], [6, [5, None, None], [7, None, None]]]
Value to insert (-1 to stop)? -1
```

4. Write a function `tree_as_string(t)` that can be used to "pretty print" trees as a sorted list of its values. The function may be used like this :

```
root = ...
print("My tree", tree_as_string(root))
```

Rewrite the solution of the preceding question so that the tree will be pretty printed :

```
Tree []
Value to insert (-1 to stop)? 50
Tree is  [ 50, ]
Value to insert (-1 to stop)? 30
Tree is  [ 30,  50, ]
Value to insert (-1 to stop)? 70
Tree is  [ 30,  50,  70, ]
Value to insert (-1 to stop)? 10
Tree is  [ 10,  30,  50,  70, ]
Value to insert (-1 to stop)? 11
Tree is  [ 10,  11,  30,  50,  70, ]
Value to insert (-1 to stop)? 10
Tree is  [ 10,  10,  11,  30,  50,  70, ]
Value to insert (-1 to stop)? -1
```

5. Write a function `height(t)` that will returns the height of the tree t.

6. Write a function `size(t)` that willl return the number of elements in the tree t.

Exercise n°2 : More complex trees (much more difficult)

Here trees are made of quadruplets, the last element of the quadruplet being the "link to the parent"

1. Write a function `remove(t, e)` that will remove the element e in the tree t.