Assignment 6.1: The AVL Tree (4 points)

Implement a AVL tree data structure in Python. Create following classes:

- `AVLNode`: stores a value (`int`) and links to its left and right child nodes.
- `AVL`: stores the tree structure built using `AVLNode` objects and maintains the AVL-tree property

To get started, implement half of the AVL tree using the code presented in the AVL Tree example written in Python. Finalize the class `AVL` by creating following methods:

- `preorder()`: enumerates the keys and their balance values in preorder
  - format: "key"+"sign of balance variable (+/-)" (no sign if zero), every node separated with space (see example)
- `left_rotation(node: AVLNode)` (**2 pts.**): symmetrical to `right_rotation`
- `right_left_rotation(node: AVLNode)` (**2 pts.**): symmetrical to `left_right_rotation`

Then, finalize the class method `insert_help` so that it functions properly.

A code template with an example program:

```
# avl.py


class AVLNode:
    # TODO


class AVL:
    # TODO


if __name__ == "__main__":
    Tree = AVL()
    for key in [9, 10, 11, 3, 2, 6, 4, 7, 5, 1]:
        Tree.insert(key)
    Tree.preorder()
```

Output:

Submit your solution in CodeGrade as `avl.py`.

Assignment 6.2: The Min Heap (4 points)

Implement the min heap structure in Python. Create class `MinHeap` which takes a `list` of numbers (`int`) as an input value and forms the heap from them. The class must have following methods:

- `__init__(A: list)` (**2 pts.**): initializer, forms the heap from a list `A` using the efficient method
- `print()`: prints the heap in breadth-first order
  - the format: key values separated by spaces (e.g. 1 2 3 4).
- `push(key: int)` (**1 pts.**): inserts a new key to the heap while maintaining the min heap property.
- `pop()` (**1 pts.**): removes the smallest key from the heap and returns its value.

Make sure that the heap always maintains the min heap property. The best practice is to store the heap structure in a list where a key in index i

- has a left child at index $2i + 1$
- has a right child at index $2i + 2$
- has its parent at index (rounds down to the nearest integer)

A code template with an example program:

```
# minheap.py


class MinHeap:
    # TODO


if __name__ == "__main__":
    heap = MinHeap([4, 8, 6, 5, 1, 2, 3])
    heap.print()        # 1 4 2 5 8 6 3
    print(heap.pop())   # 1
    heap.push(9)
    heap.print()        # 2 4 3 5 8 6 9
```

Output:

```
$ python avl.py
1 4 2 5 8 6 3
1
2 4 3 5 8 6 9
```

Submit your solution in CodeGrade as `minheap.py`

Last modified: Monday, 28 October 2024, 2:23 PM