Trees — Part 1

In this sequences of works, we will for the first time create new object types (classes). We will create the Tree class, which we'll use to manipulate trees, see Listing 1.

```
1  class Tree:
2      def __init__(self, data, left=None, right=None):
3          self.data = data
4          self.left = left
5          self.right = right
6
7      def __str__(self):
8          return str(self.data)
```

Listing 1: The Tree data structure.

In this data structure, each tree can either be an empty tree (`None`), or contains some data, and has two children (the left and right children). Listing 2 is an example of use with mathematical expressions. You can download it from http://www.barsamian.am/2022-2023/S7ICTB/TP9_Trees.py.

```
1  two = Tree(2)
2  four = Tree(4)
3  five = Tree(5)
4  thirteen = Tree(13)
5  tree1 = Tree("^", two, five)
6  tree2 = Tree("*", four, thirteen)
7  alsacian_tree = Tree("*", tree1, tree2)
8
9  def print_tree(tree):
10     if tree is None: return
11     print_tree(tree.left)
12     print(tree.data, end=" ")
13     print_tree(tree.right)
14
15 print_tree(alsacian_tree)
```

Listing 2: Mathematical expression.

1. Draw the tree associated to the variable `alsacian_tree`.
2. What is the result of the mathematical expression in `alsacian_tree`?
3. Define a new tree `final_tree` associated to the expression $(13 + 8) \times 2$.
4. What is printed if you call the function `print_tree` on this tree? Can you modify this function so that the printing gives the correct mathematical value when read?
5. Write a function `compute_tree` that computes the mathematical result of an expression contained in a tree. You can assume that the field `data` only contains numbers, or the strings `+`, `-`, `*`, `/` and `^`.

BONUS All trees seen in this work are either None, or a number, or an operator (that requires two sub-expressions). How would you handle other mathematical expressions like $\sqrt{5}$, i.e. expression that use functions with only one sub-expression?