Recursion

```
# ListOfNumbers = List[Number]
numbers = [3, 4, 2, 52, 101]
print(sum(numbers))
```

```
162
```

```
def recursive_sum(nums):
    if not nums:
        # BASE CASE
        return 0
    # REDUCTION
    first_number, rest = nums[0], nums[1:]
    # COMBINATION
    result = first_number + recursive_sum(rest)
    return result

recursive_sum(numbers)
```

```
162
```

```
# ListOfNumbers = List[Number | ListOfNumbers]

numbers = [42, [2, 3], 101, [2, [2, 3, 4]]]

def recursive_sum(nums):
    if not isinstance(nums, list):
        return nums
    if not nums:
        # BASE CASE
        return 0
    # Reduction
    first, *rest = nums
    # combination
    result = recursive_sum(first) + recursive_sum(rest)
    return result

recursive_sum(numbers)
```

```
159
```

```
# (2 + 3) * 5

# Tree has 2 children (left and right).
# Children are either a number (leaf node), or a Tree

import dis
def _get_lambda_op(func):
    # ignore this function, I just want to print the binary operator
```

```python
from the lambda.
    for instruction in dis.get_instructions(func.__code__):
        if instruction.opname == 'BINARY_OP':
            return instruction.argrepr

class ArithemeticTree:
    def __init__(self, left, right=0, operation=None):
        self.left = left  # Tree | Number
        self.right = right  # Tree | NoneType
        self.operation = operation

    def result(self):
        if self.operation:
            # combination + reduction
            return self.operation(self.left.result(),
                                  self.right.result())
        else:
            # Base case
            return self.left

    def __str__(self):
        if not self.operation:
            return str(self.left)
        else:
            return f'({self.left} {_get_lambda_op(self.operation)}
{self.right})'

tree = ArithemeticTree(
    left=ArithemeticTree(
        left=ArithemeticTree(2),
        right=ArithemeticTree(3),
        operation=lambda x, y: x+y),
    right=ArithemeticTree(5),
    operation=lambda x, y: x*y
    )

print(f'{tree} = {tree.result()}')
```

```
((2 + 3) * 5) = 25
```