5.2 Nested Lists: A Recursive Data Structure

NEW APPROACH:
- Breaking down an object/ problem into smaller instances with the same structure as the original

RECURSIVE DEFINITION:
- It defines nested lists in terms of other nested lists
- Solve a problem by using an algorithm that calls itself on a smaller problem

DEPTH OF A NESTED LIST:
- **The maximum number of times a list is nested inside other lists**
- **Depth 0 ->** `[1, 2, 3]`
- **Depth 1 ->** `[1, [2, 3]]`

RECURVISE FUNCTION:
- A function that calls itself in the body
- Has base case & recursive case
- A.K.A *self-referential definition*

BASE CASE:
- Case where the object is an integer
- Straightforward, doesn't involve recursion
- SIMPLEST PROBLEM, CAN'T BE BROKEN DOWN FURTHER (*Where we stop recursing*)

RECURSIVE CASE:
- Case where object is a list
- Decomposing the input into smaller nested lists by calling itself on these individually.
- MUST GUARANTEE TO GET TO BASE CASE

```
def sum_nested(obj: Union[int, List]) -> int:
    """Return the sum of the numbers in a nested list <obj>.
    """
    if isinstance(obj, int):
        # obj is an integer
        return obj
    else:
        # obj is a list of nested lists: [lst_1, ..., lst_n]
        s = 0
        for sublist in obj:
            # each sublist is a nested list
            s += sum_nested(sublist)
        return s
```

Base Case

Recursive Case

WHEN WE ARE GIVEN A RECURSIVE FUNCTION... WE USE PARTIAL TRACING:

1. The input corresponds to a base case:
   - Trace the if branch directly and ignore the else brach
   - In our function the sum of a single int is that integer itself!
2. The input corresponds to a recursive case:
   - We assume it is correct , use the correct return value & continue tracing the rest of the code!
   - One approach is to make a table of values:
   - USE THE STEP OVER IN PYCHARM instead of the Step Into
   -

sublist	sum_nested(sublist)	s
N/A	N/A	0 (initial value)
1	1	1 (s += 1)
[2, [3, 4], 5]	14	15 (s += 14)
[6, 7]	13	28 (s += 13)
8	8	36 (s += 8)

WHY DOES PARTIAL TRACING WORK?

ASSUMPTION IS VALID AS LONG AS:

1. You are sure the base case is correct
2. Every time you make a recursive call, it is on a smaller input than the original input
3. Idea is formed from the *Principle of Mathematical induction* 😎🙌

IF RECURSIVE FUNCTION IS INCORRECT ... THREE POSSIBLE PROBLEMS:

- A base case is incorrect
- One or more recursive calls are not being made on a smaller input
- The recursive case is incorrect ... the code surrounding the recursive call is the problem

DESIGN RECIPE FOR RECURSIVE FUNCTIONS?

1. **FIND RECURSIVE STRUCTURE OF THE IMPUT**
   - Can the data type be expressed recursively?
   - Write a code template!! (Below for nested lists)
   -
```
def f(obj: Union[int, List]) -> ...:
    if isinstance(obj, int):
        ...
    else:
        for sublist in obj:
            ... f(sublist) ...
```
2. **IDENTIFY & IMPLEMENT CODE FOR THE BASE CASE(S)**
   - Based on the structure of the input of the function
3. **EXAMPLE OF THE FUNCTION CALL ON AN INPUT OF SOME COMPLEXITY**
   - Write down relevant recursive function calls (determined by structure of input)
   - Write down output (Based on Docstring)
4. TAKE RESULTS & COMBINE THEM
   - To produce correct output for the original call
   - Implement the recursive step in ur code!

ANOTHER EXAMPLE (LECTURE)