



Practice with Recursive  
Structures & Processes

# Warmup: Memory Diagram

```
1  from __future__ import annotations
2
3  class Node:
4      """Node in a singly-linked list recursive structure."""
5      value: int
6      next: Node | None
7
8      def __init__(self, value: int, next: Node | None):
9          self.value = value
10         self.next = next
11
12     def __str__(self) -> str:
13         if self.next is None:
14             return f"{self.value} -> None"
15         else:
16             return f"{self.value} -> {self.next}"
17
18 courses: Node = Node(110, Node(210, None))
19 print(courses)
```

and discuss with a neighbor:

1. What does the `__str__` method do?
2. Is this method recursive? How do we know?

# A Recursive `last` Algorithm Demo

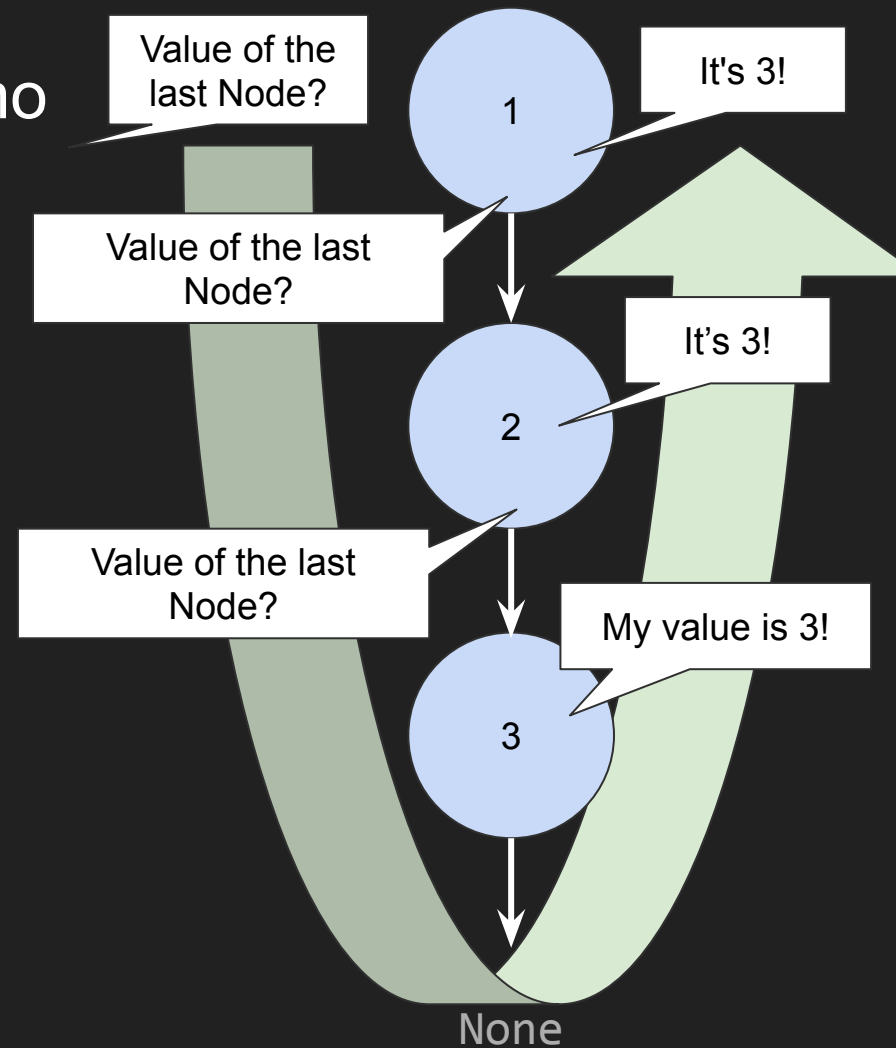
1. When you are asked,  
"What is the value of the last Node?"

If you're ***not the last Node***:

2. Ask the ***next*** Node,  
"What is the value of the last Node?"  
Wait patiently for an answer!
3. Once the answer is returned back to you,  
turn to the person who asked you and  
give them this answer.

If you ***are the last Node***:

2. Tell them, "my value is \_\_\_\_!" and share your value.



Let's write the `last` function in VS Code!



# recursive\_range Algorithm

Create a recursive function called `recursive_range` that will create a linked list of Nodes with values that increment from a start value up to an end value (exclusive). E.g.,

`recursive_range(start=2, end=8)` would return:  
2 -> 3 -> 4 -> 5 -> 6 -> 7 -> None

Conceptually, what will our **base case** be?

What will our **recursive case** be?

What is an **edge case** for this function?

How could we account for it?

`recursive_range(2, 8)` returns

2



`recursive_range(3, 8)` returns

3



`recursive_range(4, 8)` returns

4



`recursive_range(5, 8)` returns

5



`recursive_range(6, 8)` returns

6



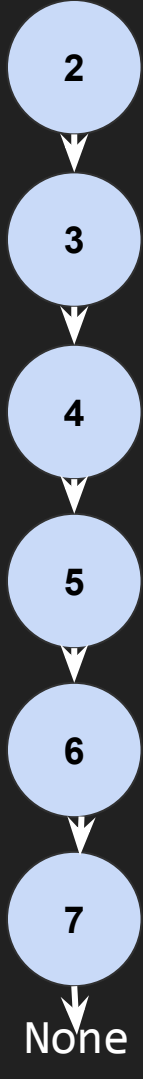
`recursive_range(7, 8)` returns

7



`recursive_range(8, 8)` returns

None



# When "building" a new linked list in a recursive function:

## Base case:

- ❑ Does the function have a clear base case?
  - ❑ Ensure the base case returns a result directly (without calling the function again).
- ❑ Will the base case *always* be reached?

## Recursive case:

- ❑ Determine what the ***first*** value of the new list will be
- ❑ Then "build" the ***rest*** of the list by recursively calling the building function
- ❑ Finally, return a new ***Node(first, rest)***, representing the a new list

Let's write the `recursive_range` function in VS Code!

