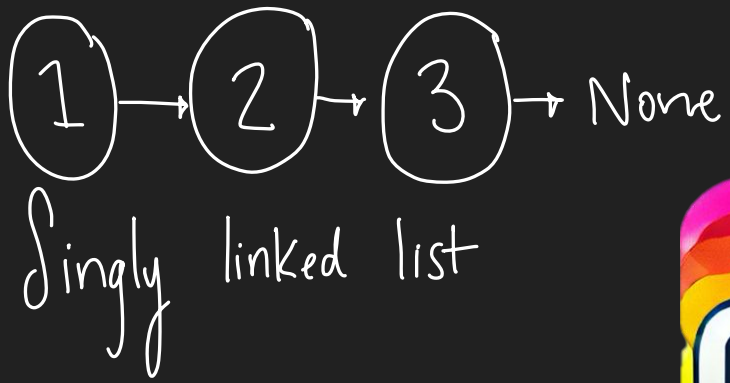


More Practice with Recursive Structures & Processes

Announcements

Re: Assignments:

- LS13: Recursive Structures due today at 11:59pm
- Next EX will be released to the site today!

Re: Quiz 03:

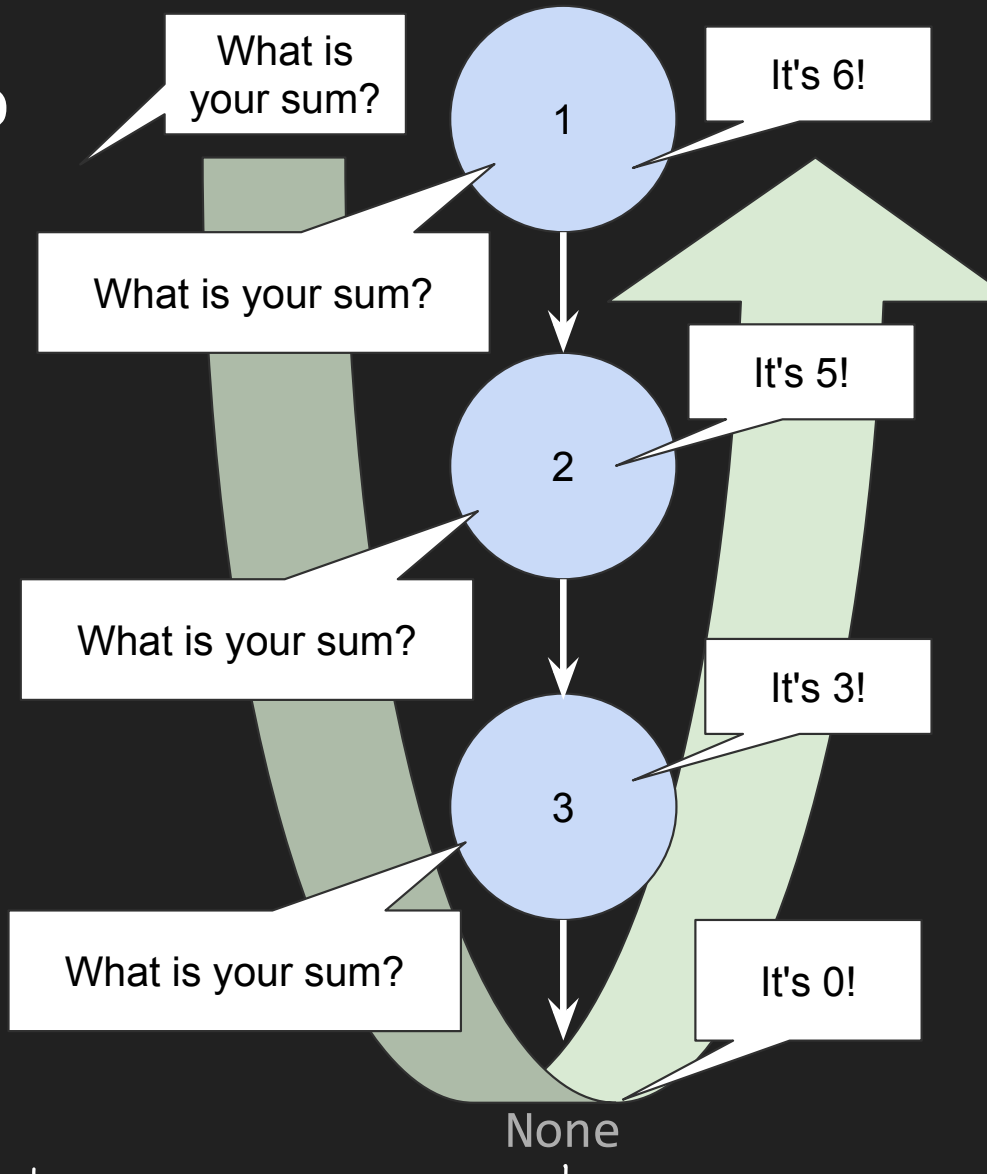
- *Regrade requests will be open **till 11:59pm on Friday!***
  - Please submit a regrade request if you believe your quiz was not graded correctly according to the rubric

A Recursive `sum` Algorithm Demo

1. When you are asked, "what is your sum?"

2. Ask the ***next*** Node, "what is your sum?"
   Wait patiently for an answer!

3. Once the answer is returned back to you, add ***your value to it***, then turn to the person who asked you and give them this answer.



Diagramming the `sum` function call

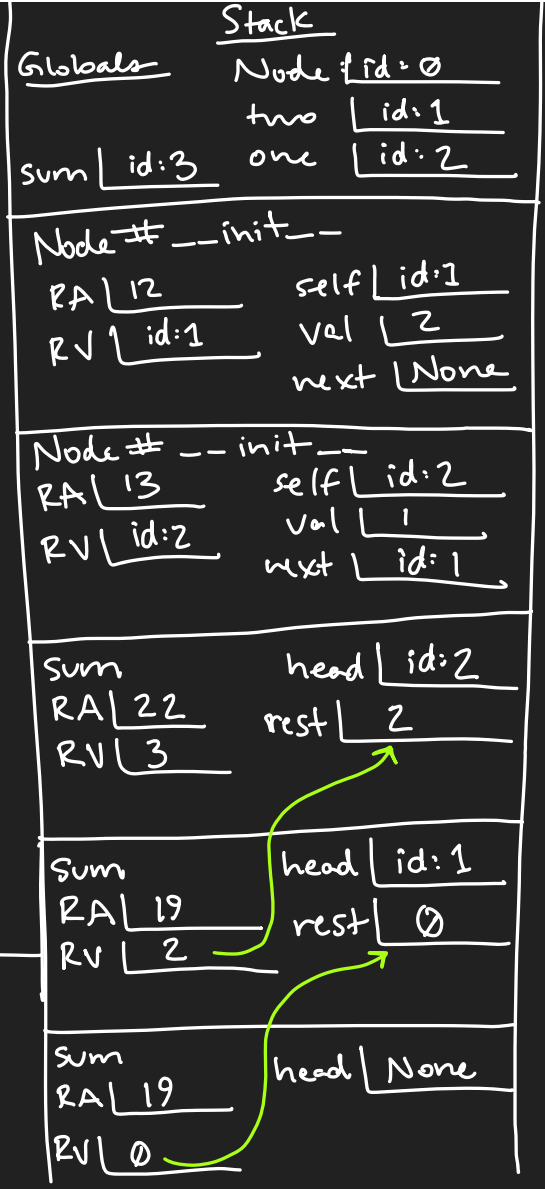
```python
from __future__ import annotations

class Node:
    value: int
    next: Node | None

    def __init__(self, val: int, next: Node | None):
        self.value = val
        self.next = next

# Note: There are no errors!
two: Node = Node(2, None)
one: Node = Node(1, two)

def sum(head: Node | None) -> int:
    if head is None:
        return 0
    else:
        rest: int = sum(head.next)
        return head.value + rest

print(sum(one))
```

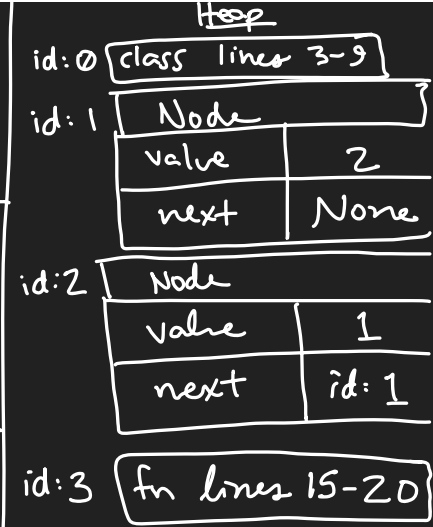
where we left off

rest is initialized to hold the return value of this recursive call

Output

3





Visualizing the `sum` function calls

A Recursive `last` Algorithm Demo

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

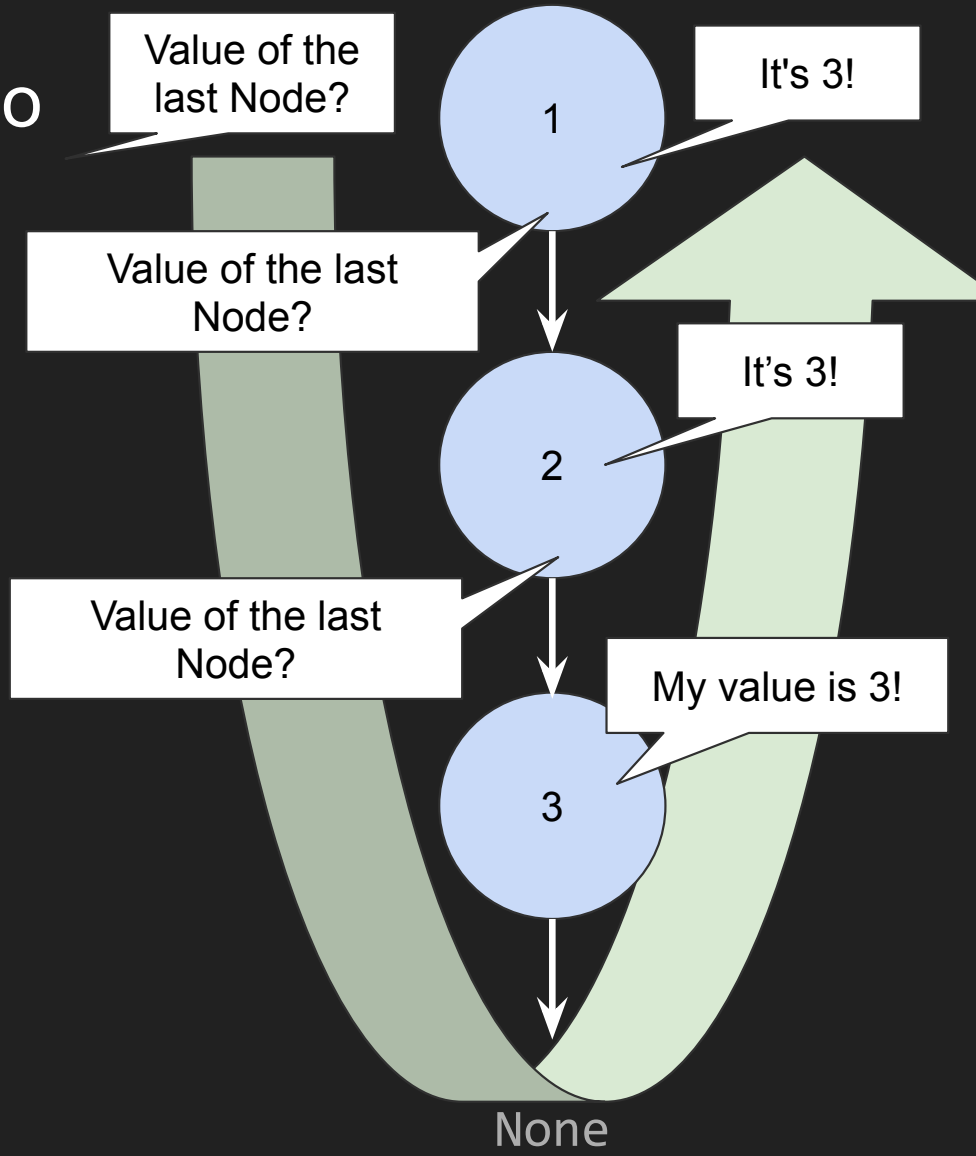
If you're ***not the last Node***:

2. Ask the ***next*** Node,
   "What is the value of the last Node?"
   Wait patiently for an answer!

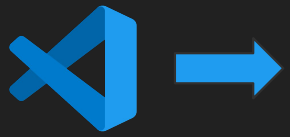
3. Once the answer is returned back to you, turn to the person who asked you and give them this answer.

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!



```
def last(head: Node) -> int:
    """Return value of last node in linked list."""
    # Base case
    if head.next is None:
        return head.value
    else:
        return last(head.next)  # Recursive case
```

Visualizing the `last` function calls

`insert_after` Algorithm Demo

1. When you are asked,
   "Can you add a Node with a value of 211 after the Node with value 210?"

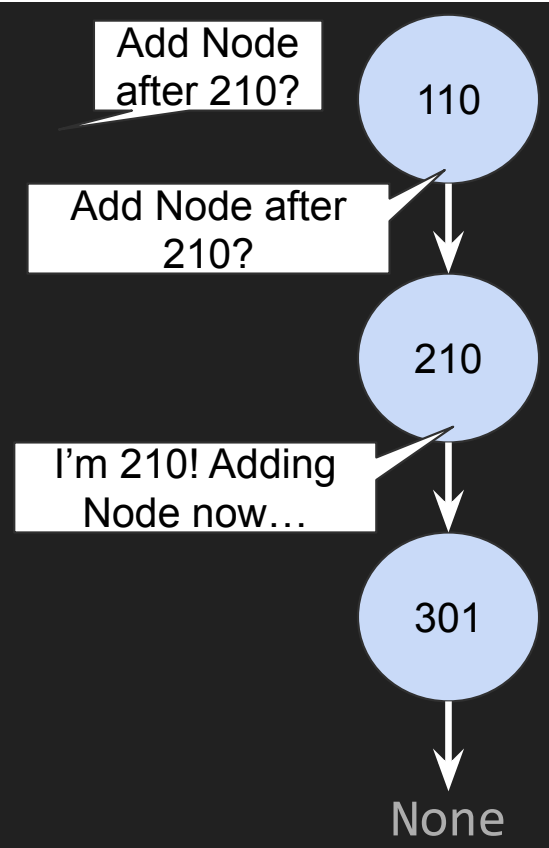
If your value ***is not 210***:

2. Ask the ***next*** Node,
   "Can you add a Node with a value of 211 after the Node with value 210?"
   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 your value **is 210**:

2. Invite a new friend to the list! You will now point to them, and they will point to the person you were previously pointing to. New Node, you'll say "I was added!!"



`insert_after` Algorithm Demo

1. When you are asked,
   "Can you add a Node with a value of 211 after the Node with value 210?"

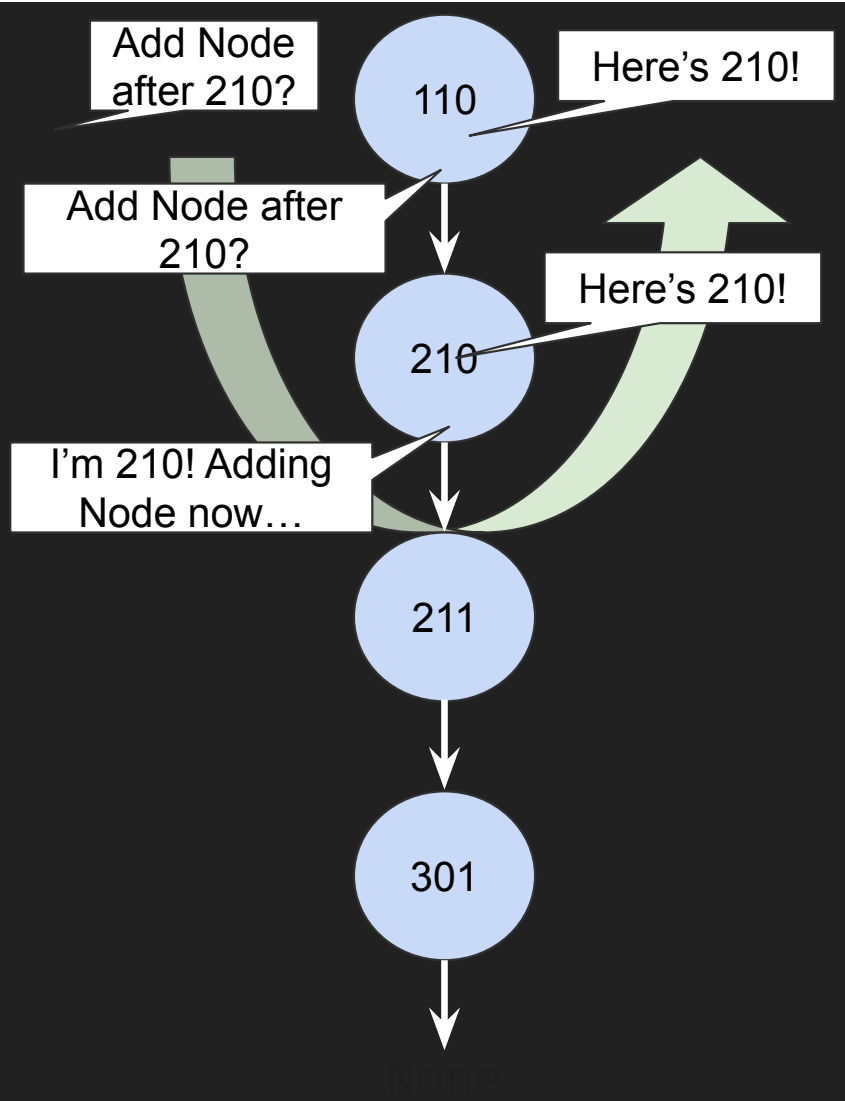
If your value ***is not 210***:

2. Ask the ***next*** Node,
   "Can you add a Node with a value of 211 after the Node with value 210?"
   Wait patiently for an answer!

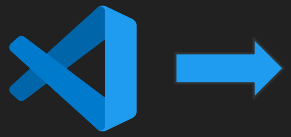
3. Once the answer is returned back to you, turn to the person who asked you and give them this answer.

If your value **is 210**:

2. Invite a new friend to the list! You will now point to them, and they will point to the person you were previously pointing to. New Node, you'll say "I was added!!"



Let’s write pseudocode for the `insert_after` function

Let’s write the `insert_after` function in VS Code! 

More practice!

`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.,

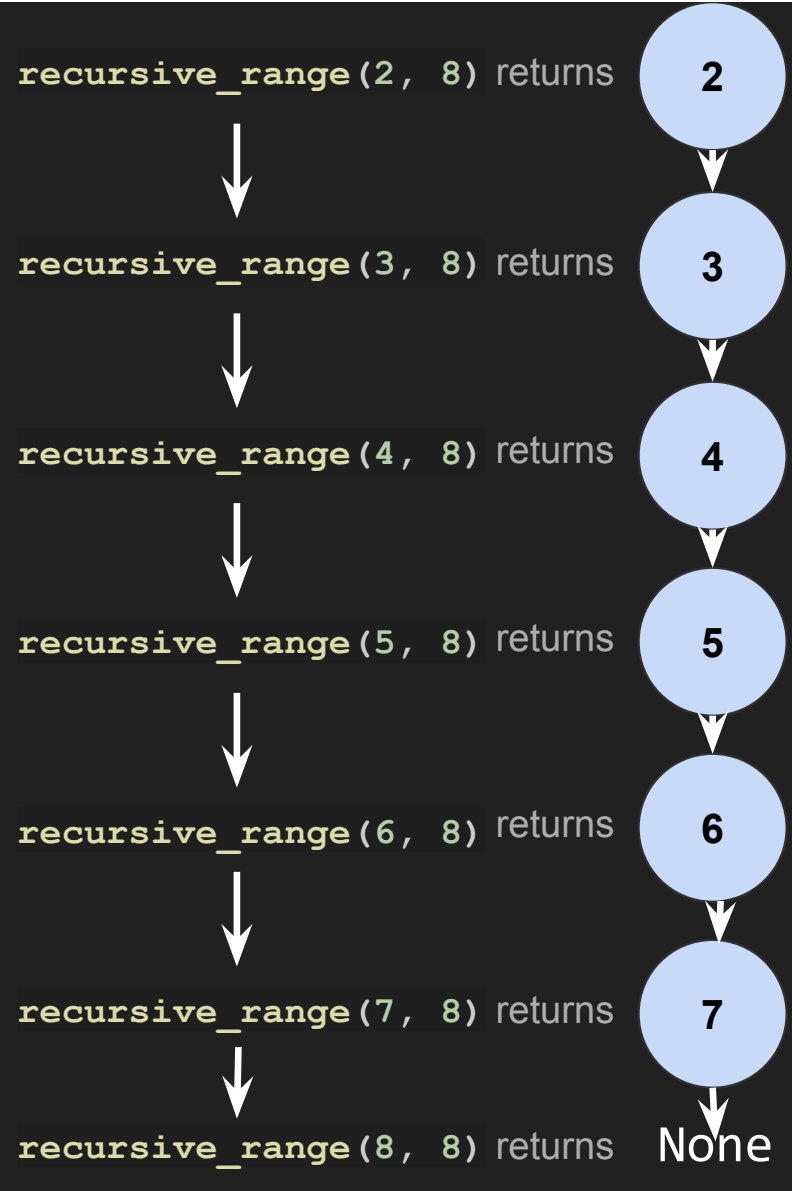
`recursive_range(start=2, end=8)` would return:

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?



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 linked 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 new linked list

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