COMP 110

Recursive Structures and Processes

Announcements

- **LS15: Recursive Structures** due tomorrow and 11:59pm
- **EX07: Linked List Utility Functions** released today

First, a review of recursion…

Review: Recursive function checklist:

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:

- ❑ Does the function have a recursive case that *progresses toward the base case*?
    - ❑ Does the recursive call have the right arguments? The function should call itself on a simpler or smaller version of the problem.
- ❑ Have you tested your function with multiple cases, including edge cases?

Review: Stack Overflow and Recursion Errors

When a programmer writes a function that calls itself indefinitely (*infinitely*), the **function call stack** will *overflow…*

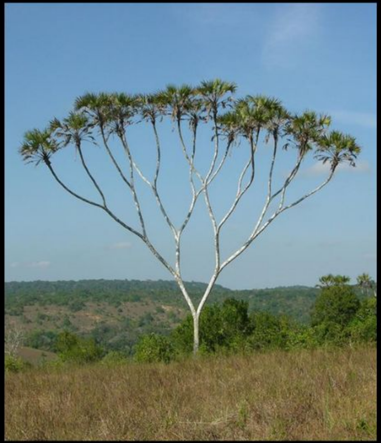
This leads to a **`Stack Overflow`** or **`Recursion Error`**:

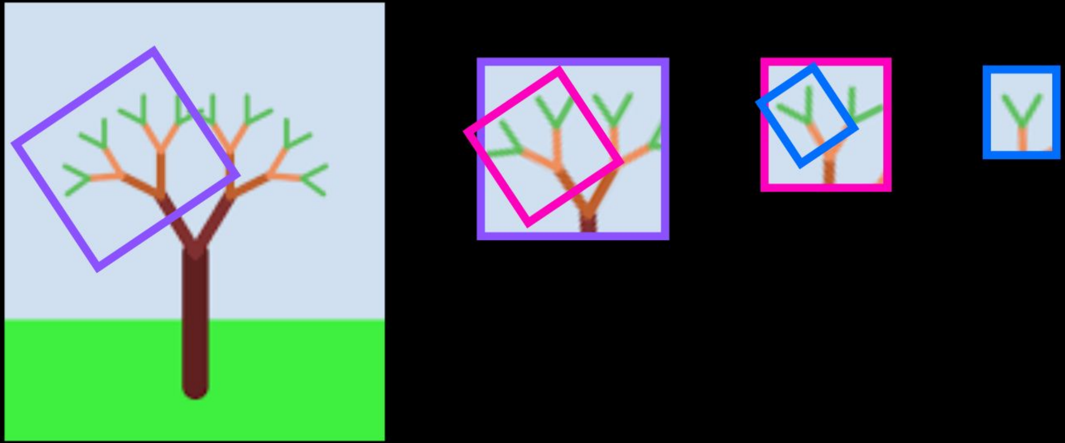
```
RecursionError: maximum recursion depth exceeded while
calling a Python object
```

Recursion: defining an operation/object in terms of itself

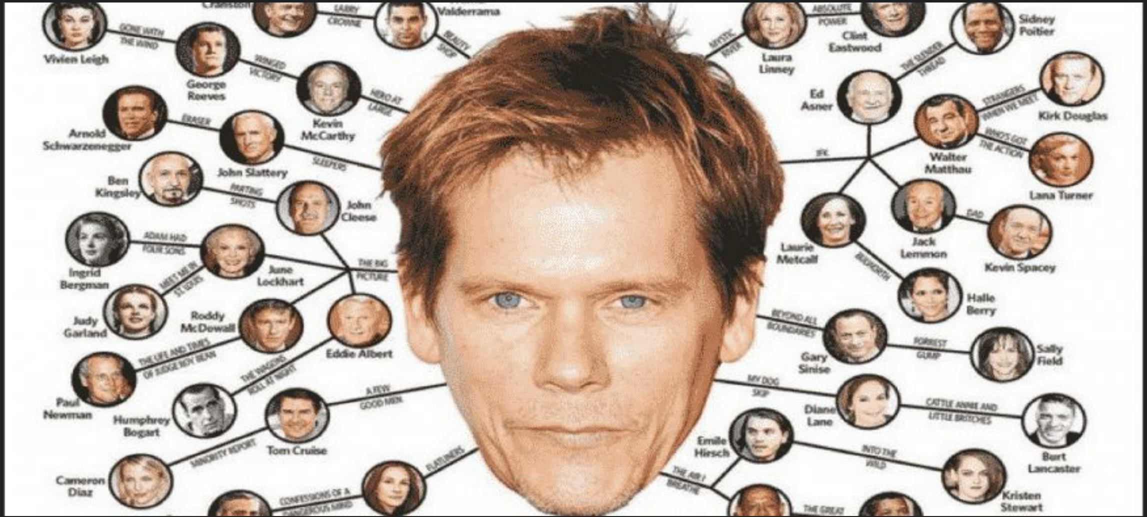
A real-world phenomenon! Examples:

- **You** have **parents**, who have **parents**, who have **parents**, who have **parents**, who…
  … were **early humans**
- A **tree** has **branches**, which have **branches**, which have **branches**, which…
  … have **leaves**



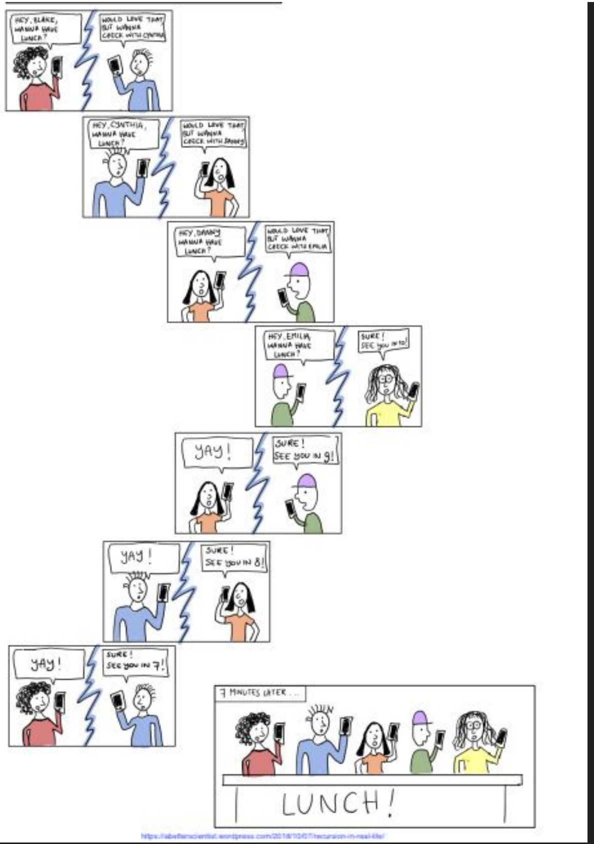


Different recursive structures for different purposes



Six degrees of Kevin Bacon

graph/network


HEY, BLAKE, WANNA HAVE LUNCH?
WOULD LOVE THAT BUT WANNA CHECK WITH CYNTHIA
HEY, CYNTHIA, WANNA HAVE LUNCH?
WOULD LOVE THAT BUT WANNA CHECK WITH DANNY
HEY, DANNY WANNA HAVE LUNCH?
WOULD LOVE THAT BUT WANNA CHECK WITH EMILIA
HEY, EMILIA, WANNA HAVE LUNCH?
SURE! SEE YOU IN 10!
YAY!
SURE! SEE YOU IN 9!
YAY!
SURE! SEE YOU IN 8!
YAY!
SURE! SEE YOU IN 7!
7 MINUTES LATER...
LUNCH!

https://abetterscientist.wordpress.com/2016/10/07/recursion-in-real-life/

Coordinating plans with individual phone calls

linked list

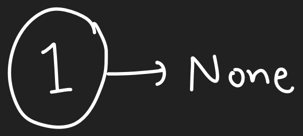
Anatomy of a Singly-Linked List

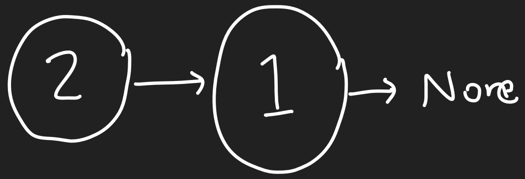
id: 1

Node	
value	1
next	None

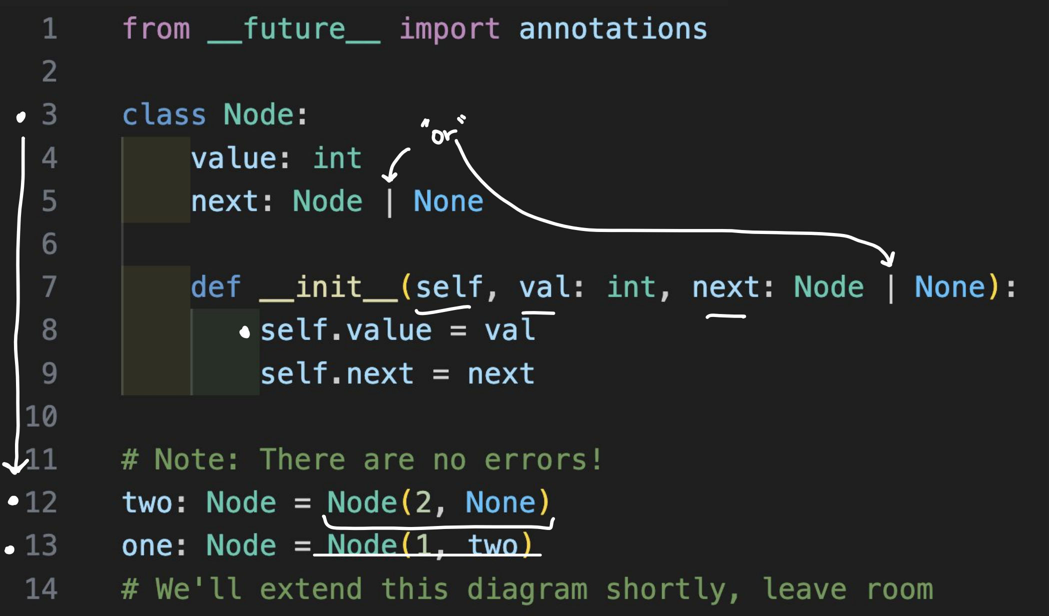
id: 2

Node	
value	2
next	id: 1



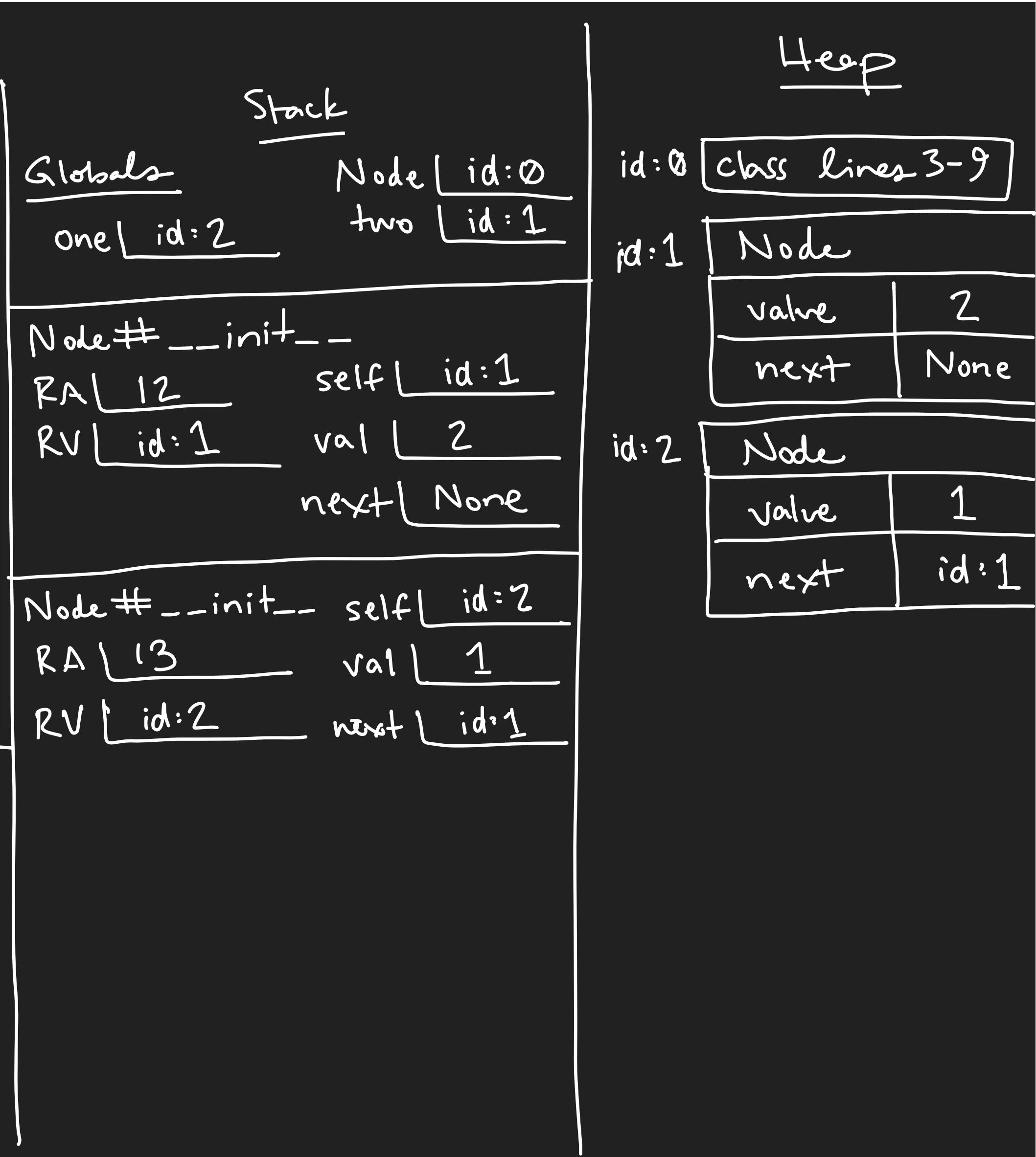


Memory diagram



```
1   from __future__ import annotations
2
3   class Node:
4       value: int
5       next: Node | None
6
7       def __init__(self, val: int, next: Node | None):
8           self.value = val
9           self.next = next
10
11  # Note: There are no errors!
12  two: Node = Node(2, None)
13  one: Node = Node(1, two)
14  # We'll extend this diagram shortly, leave room
```

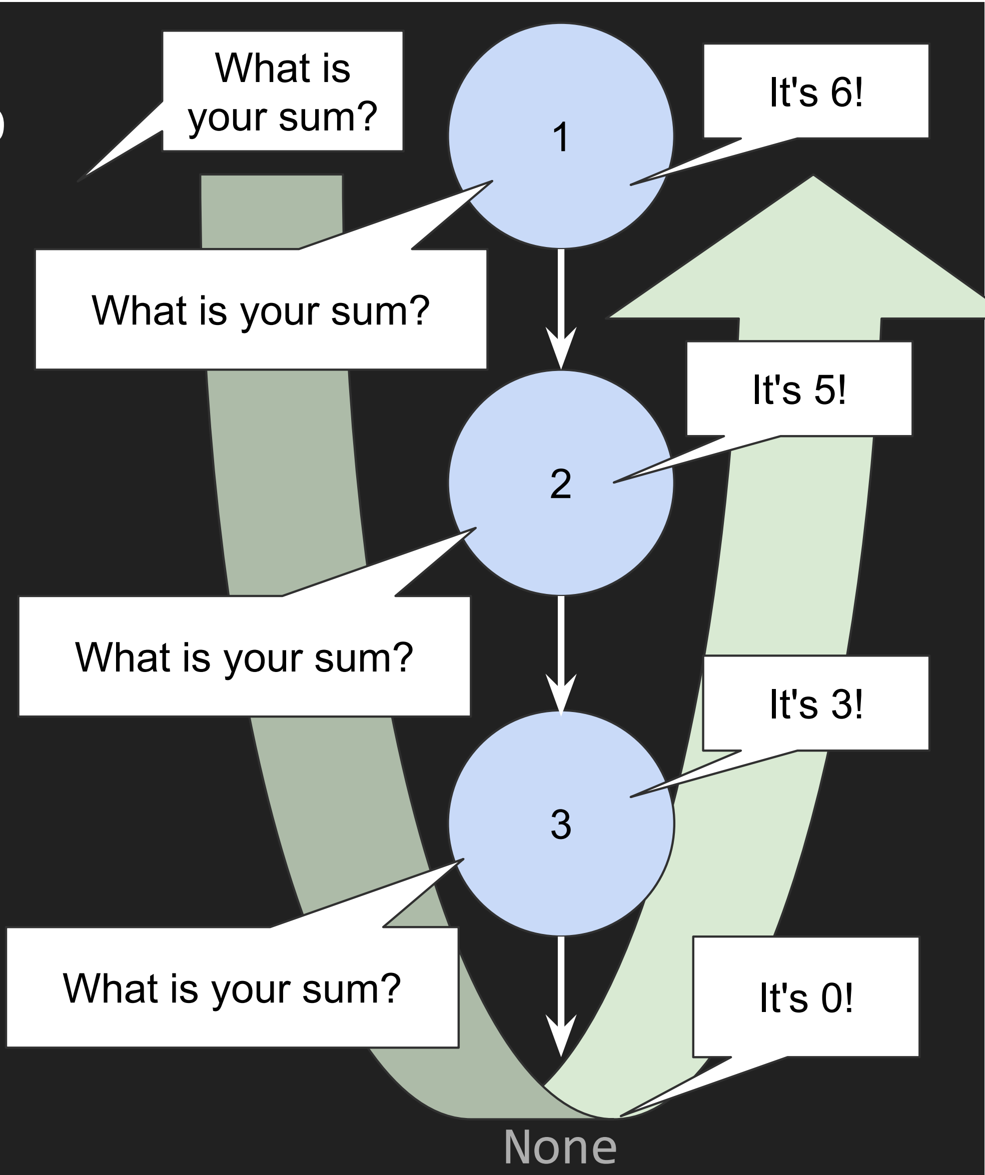
Output



A Recursive `sum` Algorithm Demo

1. When you are asked, "what is your sum?"
2. Ask the ***next*** Node, "what is your sum?"

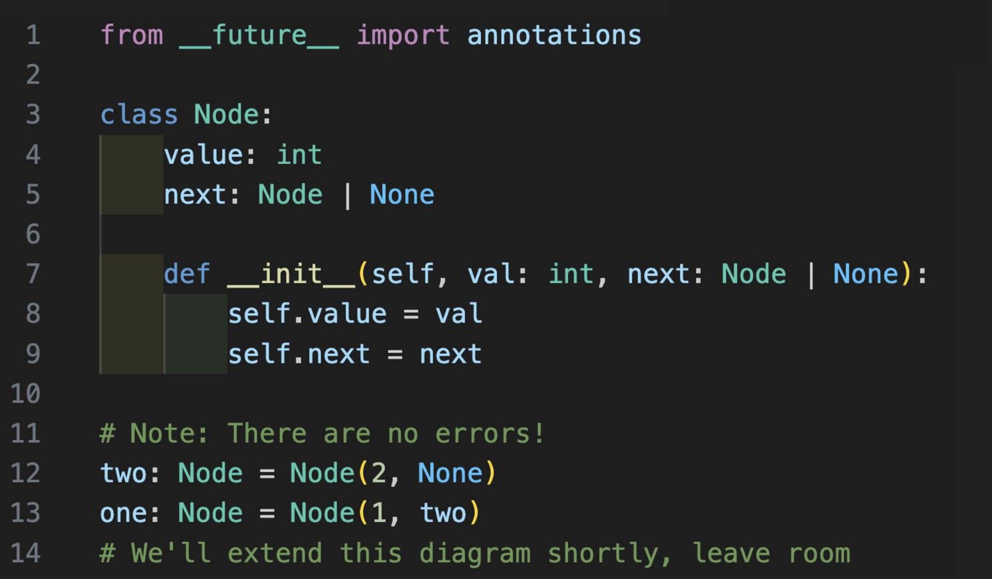
   Wait patiently for an answer!
3. Once the answer is returned back to you, add ***your value to it***, then turn to the person who asked you and give them this answer.



In your exercises folder, create a folder named ex07. In ex07, create a file named linked_list.py. Then, copy the following code into linked_list.py

```python
from __future__ import annotations


class Node:
    """Node in a singly-linked list recursive structure."""

    value: int
    next: Node | None

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


two: Node = Node(2, None)
one: Node = Node(1, two)
```

Let’s write a recursive function called **sum**!



```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)
# We'll extend this diagram shortly, leave room
```

Write a function called **sum** that adds up the **value**s of all **Node**s in the linked list.

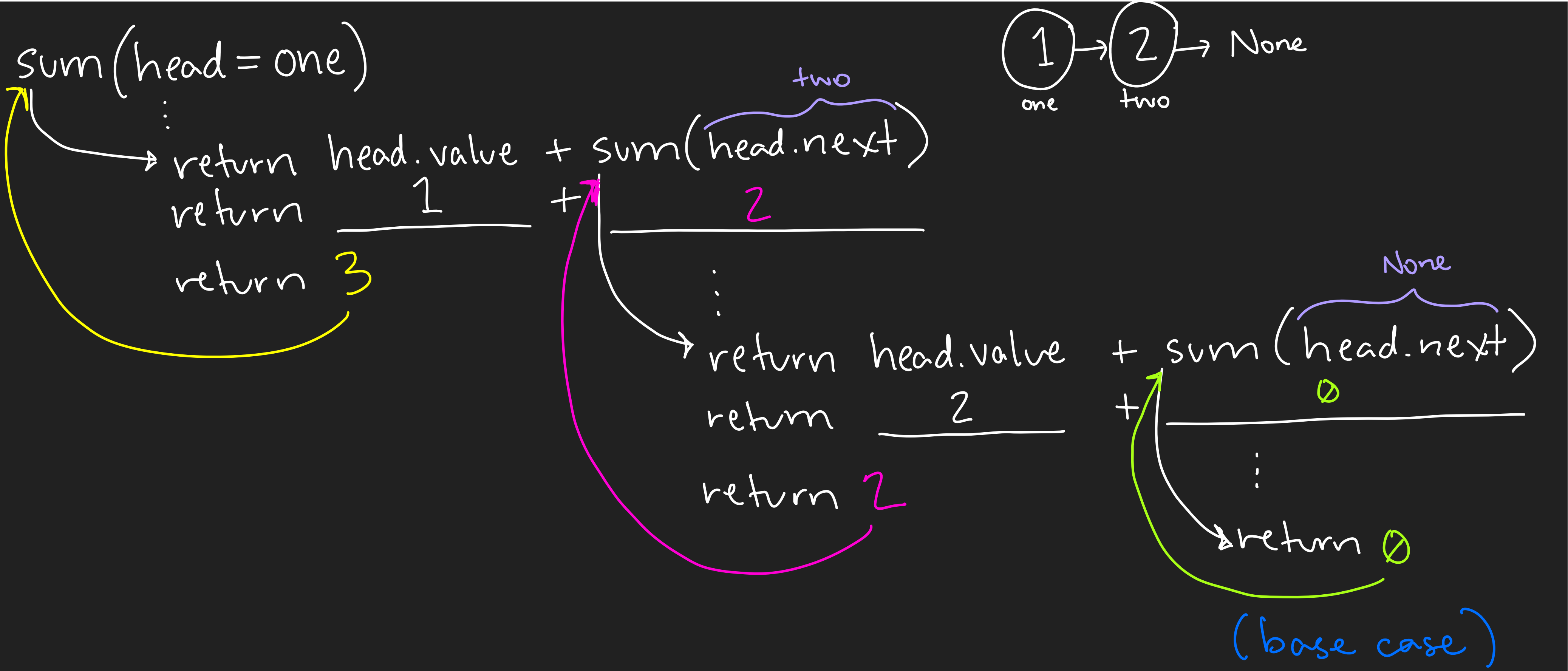
For reference: recursive function checklist:

Base case:

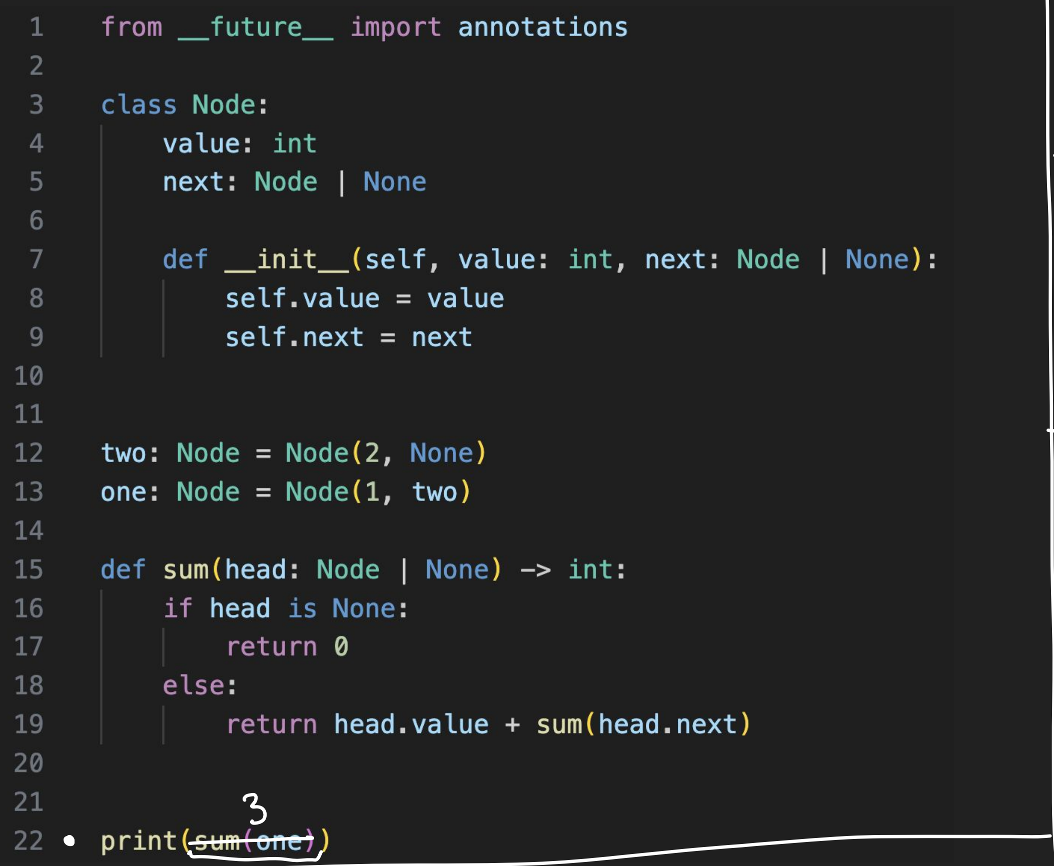
- ❑ Does the function have a clear base case?
    - ❑ Ensure the base case returns a result directly (without calling the function again).
- ❑ Will the base case *always* be reached?

Recursive case:

- ❑ Does the function have a recursive case that *progresses toward the base case*?
    - ❑ Does the recursive call have the right arguments? The function should call itself on a simpler or smaller version of the problem.
- ❑ Have you tested your function with multiple cases, including edge cases?


1 → 2 → None
one
two
sum(head = one)
⋮
return head.value + sum(head.next)
two
return 1 + 2
return 3
⋮
return head.value + sum(head.next)
None
return 2 + ∅
return 2
⋮
return ∅
(base case)

Diagramming the **sum** function call

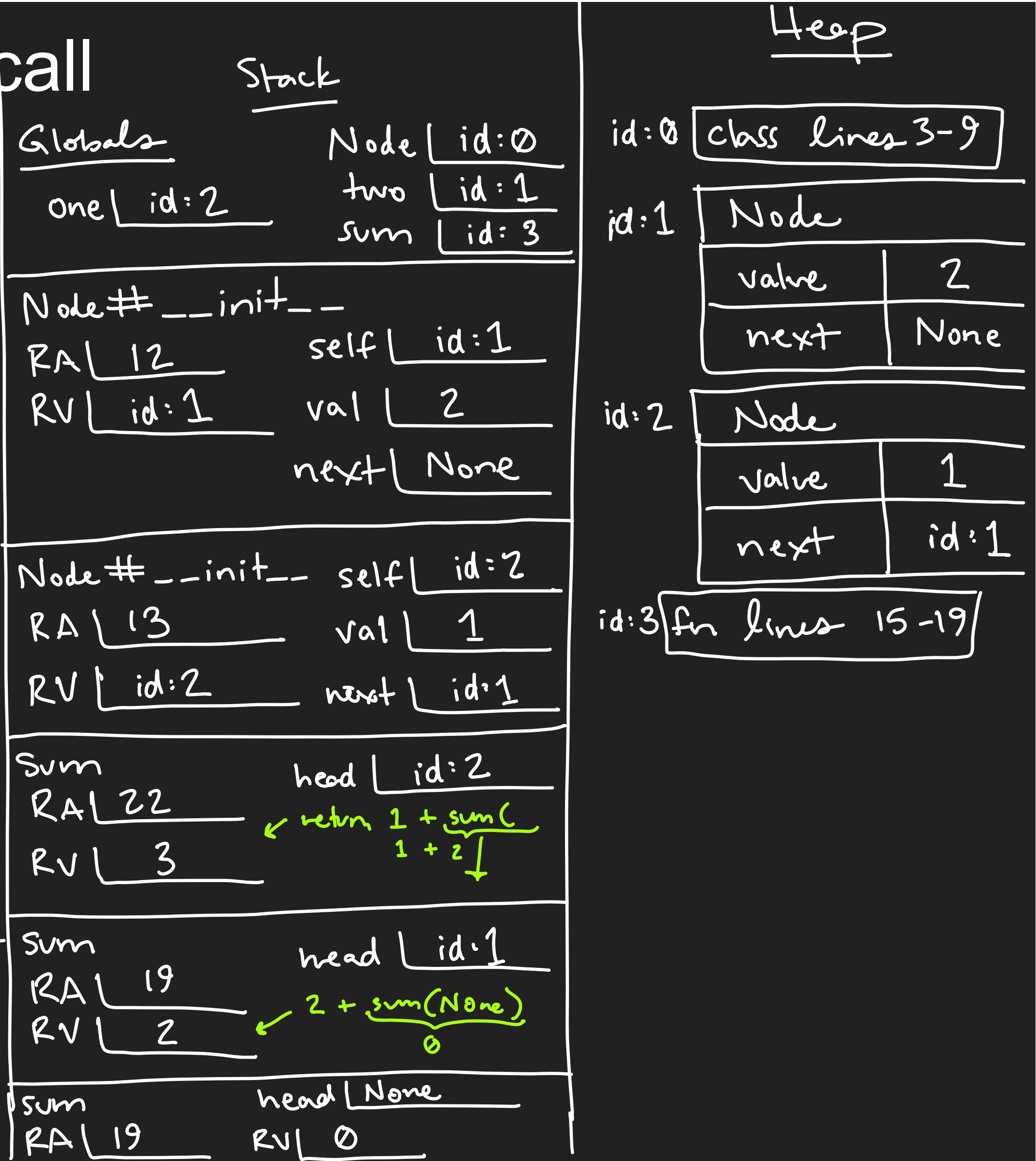


```python
from __future__ import annotations

class Node:
    value: int
    next: Node | None

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


two: Node = Node(2, None)
one: Node = Node(1, two)

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


print(sum(one))
```

3

Output

3



More practice!

A Recursive `last` Algorithm Demo

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

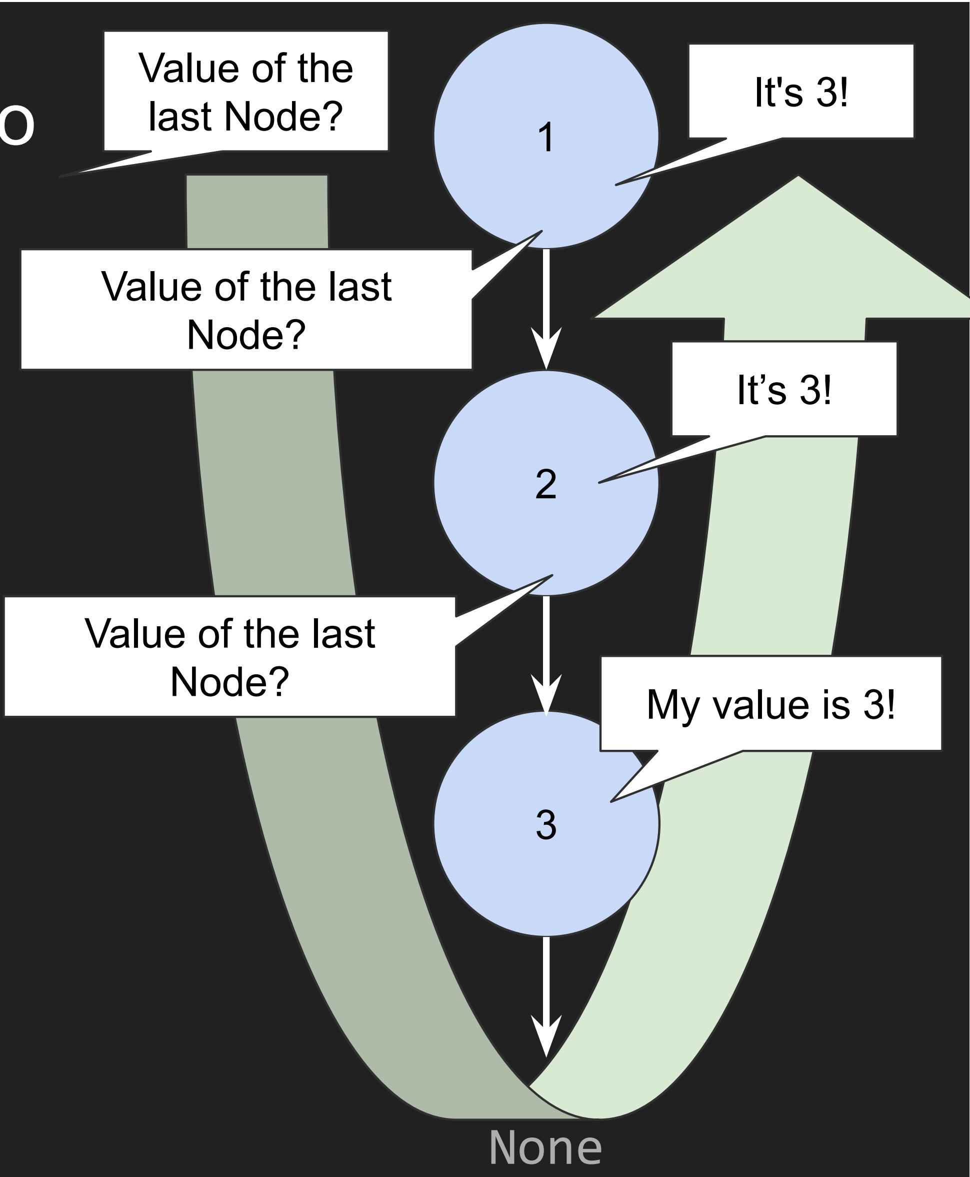
If you're ***not the last Node***:

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!

`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?"

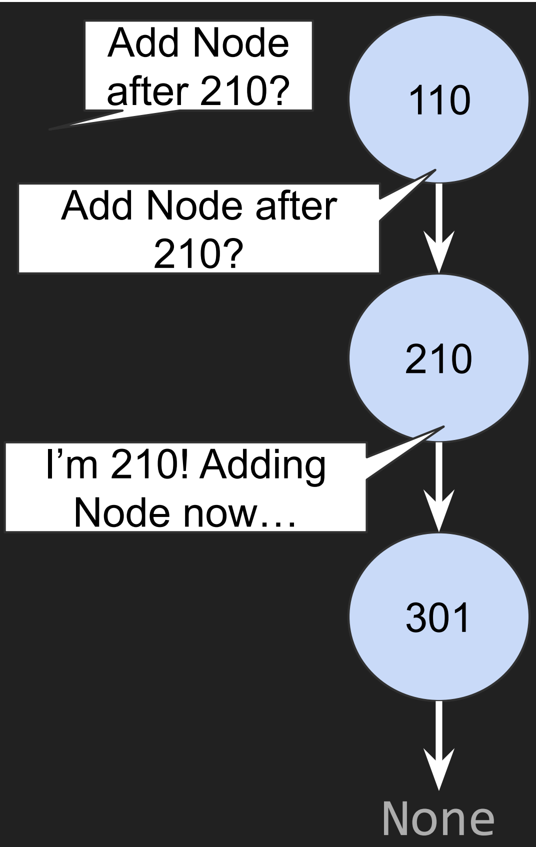
If your value ***is not 210***:

2. Ask the ***<u>next</u>*** 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?"

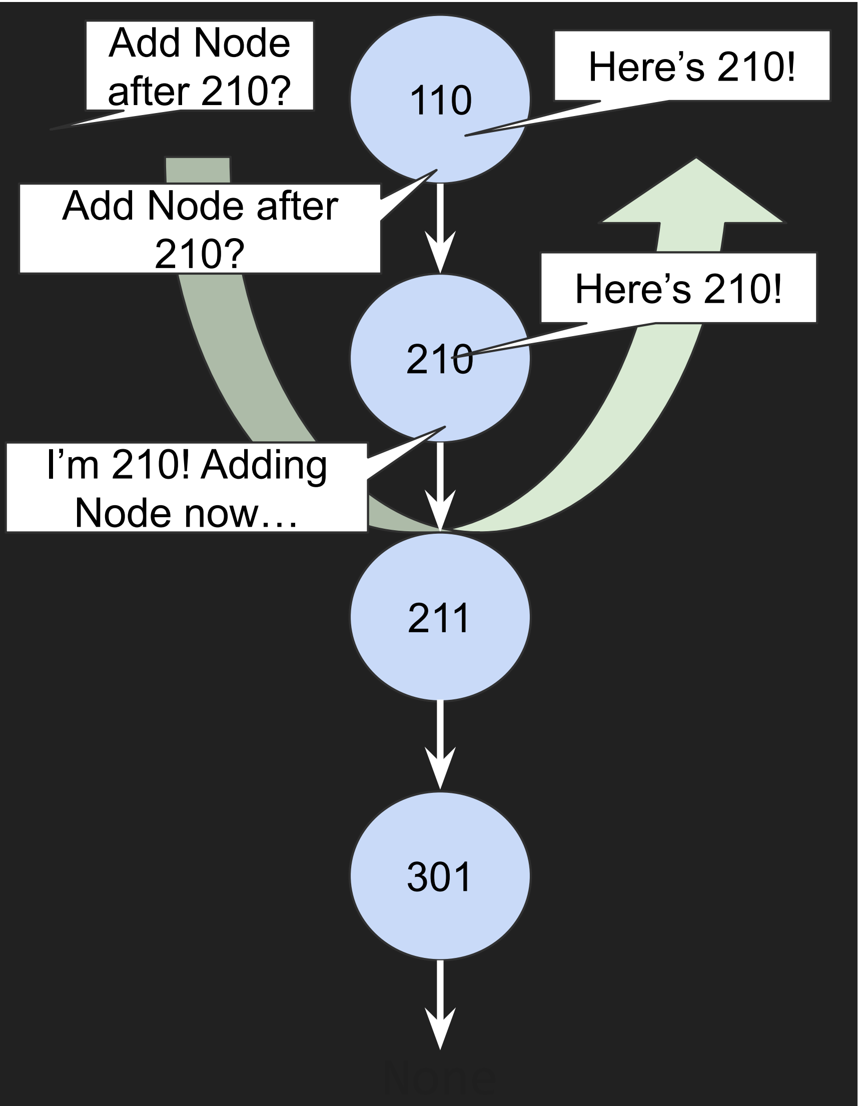
If your value ***is not 210***:

2. Ask the ***<u>next</u>*** 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!!"



Add Node after 210?

110

Here's 210!

Add Node after 210?

Here's 210!

210

I'm 210! Adding Node now…

211

301

None

Let’s write pseudocode for the `insert_after` function

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