UNC TECHNOLOGY, ETHICS & CULTURE IN STOCKHOLM

UNC
COLLEGE OF
ARTS AND SCIENCES
Study Abroad

COMP 380
Technology,
Ethics, & Culture

May 18 - June 5, 2026

Cece McMahon
mcmahon@cs.unc.edu

More Info & Apply
go.unc.edu/
tech-ethics-culture

COMP110

🪄Magic Method🪄 & Building Linked Lists with Recursive Algorithms

Announcements

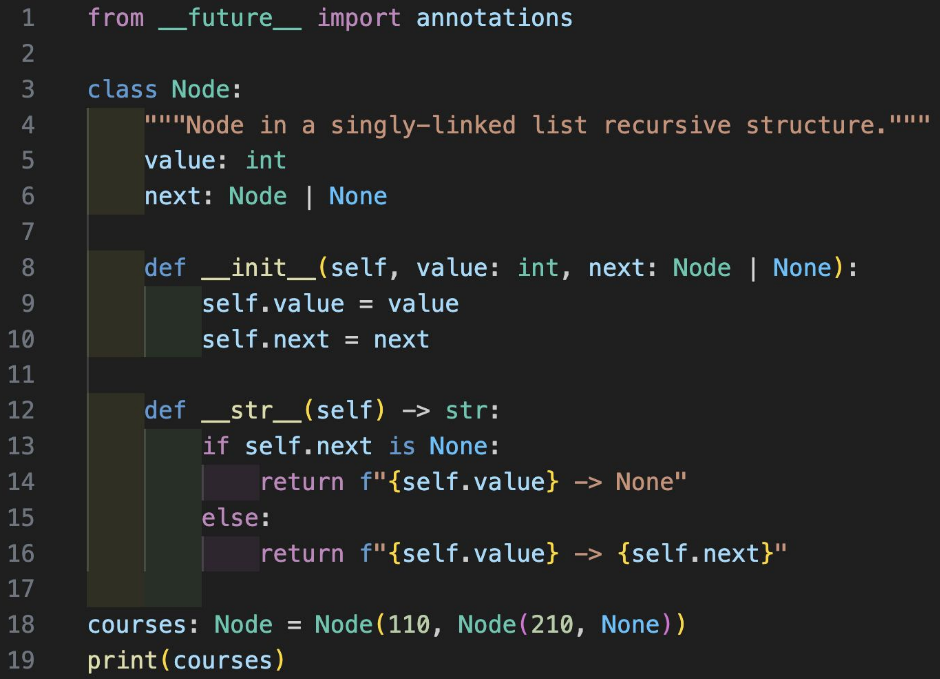
Re: Assignments:

- **EX07: Linked List Utility Functions** due Tues. Nov 25th

Re: Quiz 03:

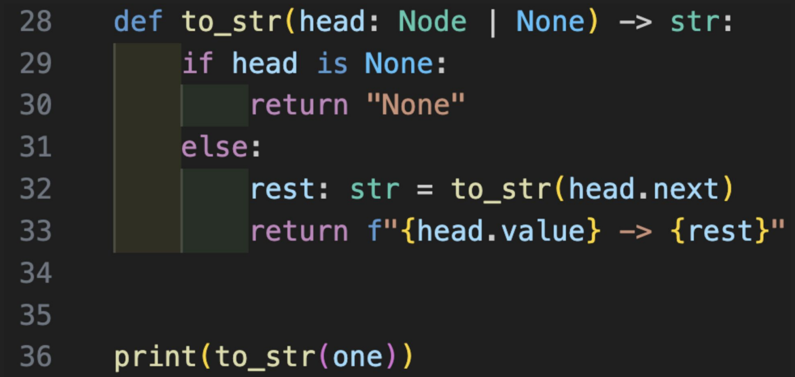
- *Regrade requests will be open* ***till 11:59pm on Friday!***
  - Please submit a regrade request if you believe your quiz was not graded correctly according to the rubric

Warm-up: Discuss with a neighbor:



```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

    def __str__(self) -> str:
        if self.next is None:
            return f"{self.value} -> None"
        else:
            return f"{self.value} -> {self.next}"

courses: Node = Node(110, Node(210, None))
print(courses)
```

1. What does the __str__ method do?
   a. How is it similar to the to_str function you diagrammed for LS13 (shown below)? How is it different?



```python
def to_str(head: Node | None) -> str:
    if head is None:
        return "None"
    else:
        rest: str = to_str(head.next)
        return f"{head.value} -> {rest}"


print(to_str(one))
```

2. Is this method recursive? How do we know?

Memory Diagram

```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

    def __str__(self) -> str:
        if self.next is None:
            return f"{self.value} -> None"
        else:
            return f"{self.value} -> {self.next}"

courses: Node = Node(110, Node(210, None))
print(courses)
```

Copy this into VS Code!

```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

    def __str__(self) -> str:
        if self.next is None:
            return f"{self.value} -> None"
        else:
            return f"{self.value} -> {self.next}"

courses: Node = Node(110, Node(210, Node(211, None)))
print(courses)
```

`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:

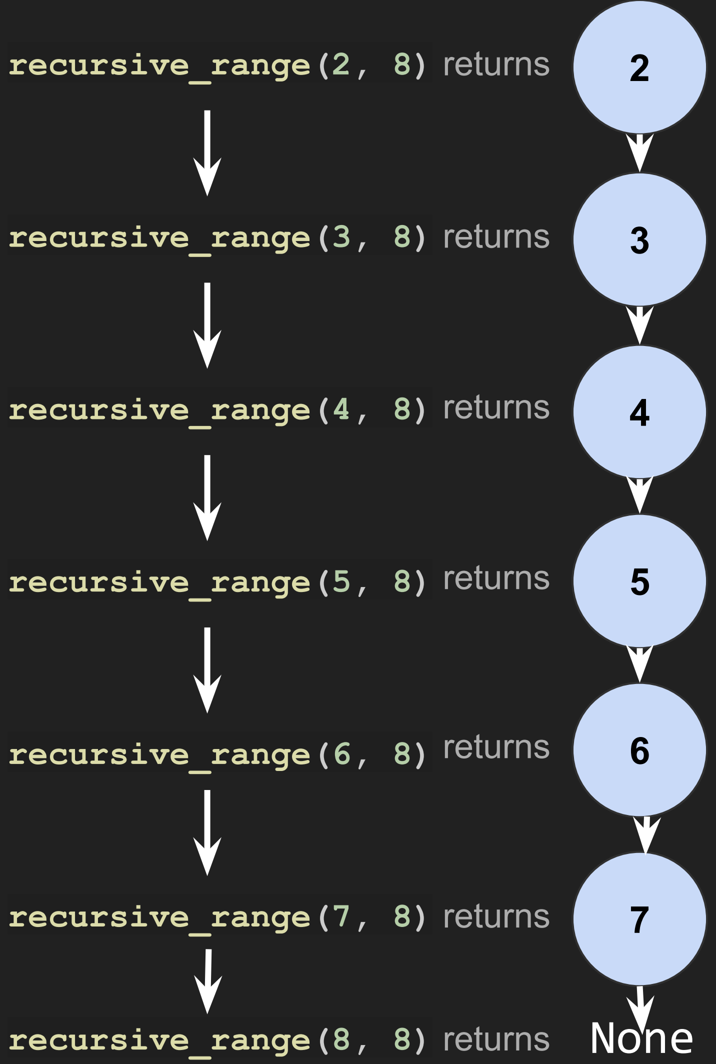
2 -> 3 -> 4 -> 5 -> 6 -> 7 -> None

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 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!

More practice!

`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?"

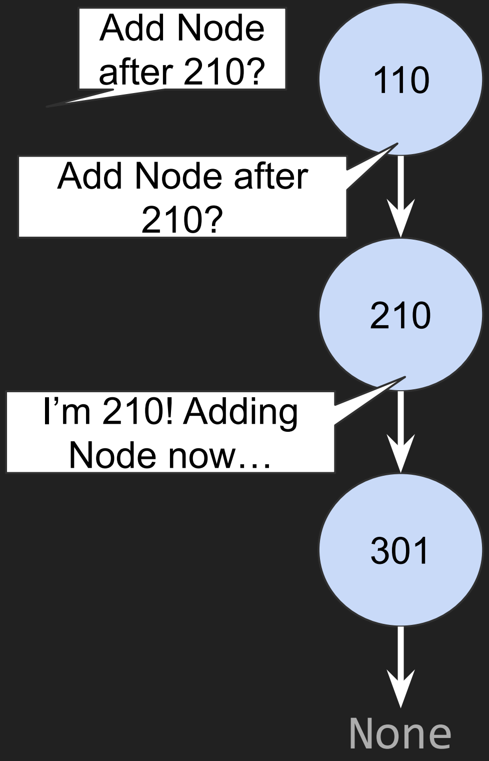
If your value ***is not 210***:

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?"

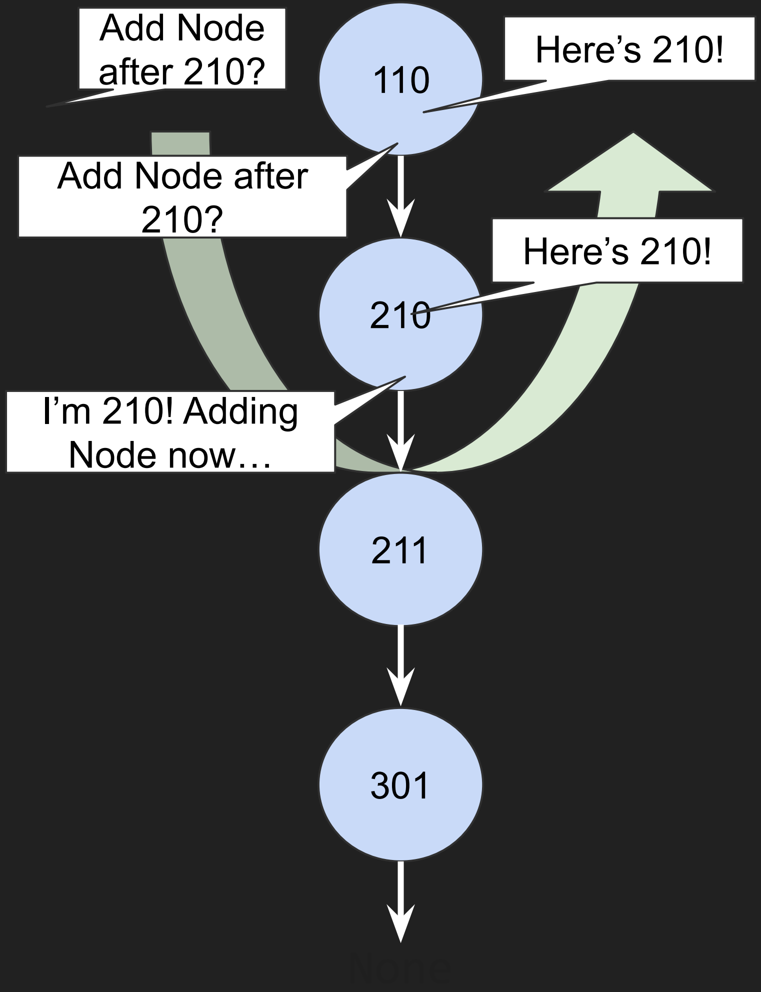
If your value ***is not 210***:

2. Ask the ***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!