Building Linked Lists with Recursive Algorithms

Announcements

Re: Assignments:

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

Re: Quiz 03:

- *Today is the last day to submit a regrade request!*
  - Please submit a regrade request if you believe your quiz was not graded correctly according to the rubric

Re: Quiz 04:

- Quiz practice will be added to the site today!

Warm-up: With a partner, answer the following questions.

Question 4: Traversing a Linked List   Print the output of the function calls below. Write "Error" if code would result in an error.

```
 1  from __future__ import annotations
 2
 3  class Node:
 4    value: int
 5    next: Node | None
 6
 7    def __init__(self, value: int, next: Node | None):
 8      self.value = value
 9      self.next = next
10
11    def __str__(self) -> str:
12      rest: str
13      if self.next is None:
14        rest = "None"
15      else:
16        rest = str(self.next)
17      return f"{self.value} -> {rest}"
18
19  sun: Node = Node(4, None)
20  moon: Node = Node(7, sun)
```

4.1. Print the output.

```
1  print(moon)
```

4.2. Print the output.

```
1  print(sun.value)
```

4.3. Print the output.

```
1  print(moon.next)
```

4.4. Print the output.

```
1  print(moon.next.next)
```

`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)
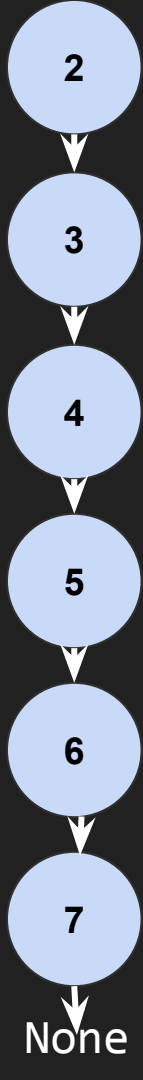
`recursive_range(6, 8)` returns (6)

`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! ➡️
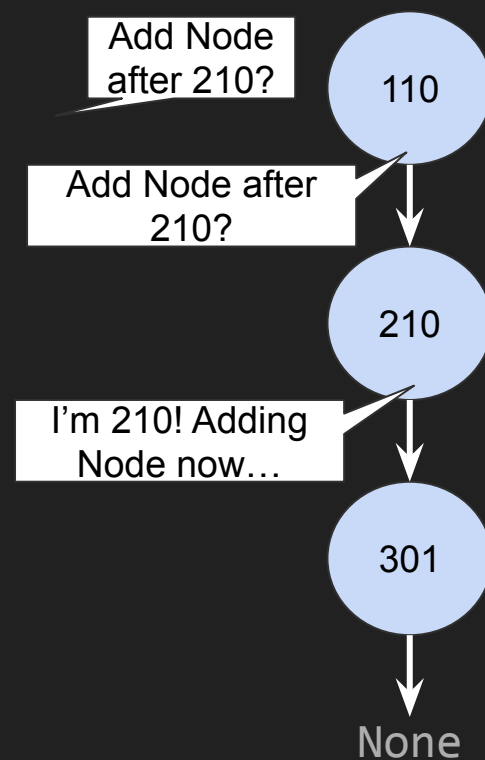
`insert_after` Algorithm Demo

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!!"

Add Node after 210?

Add Node after 210?

110

210

I'm 210! Adding Node now…

301
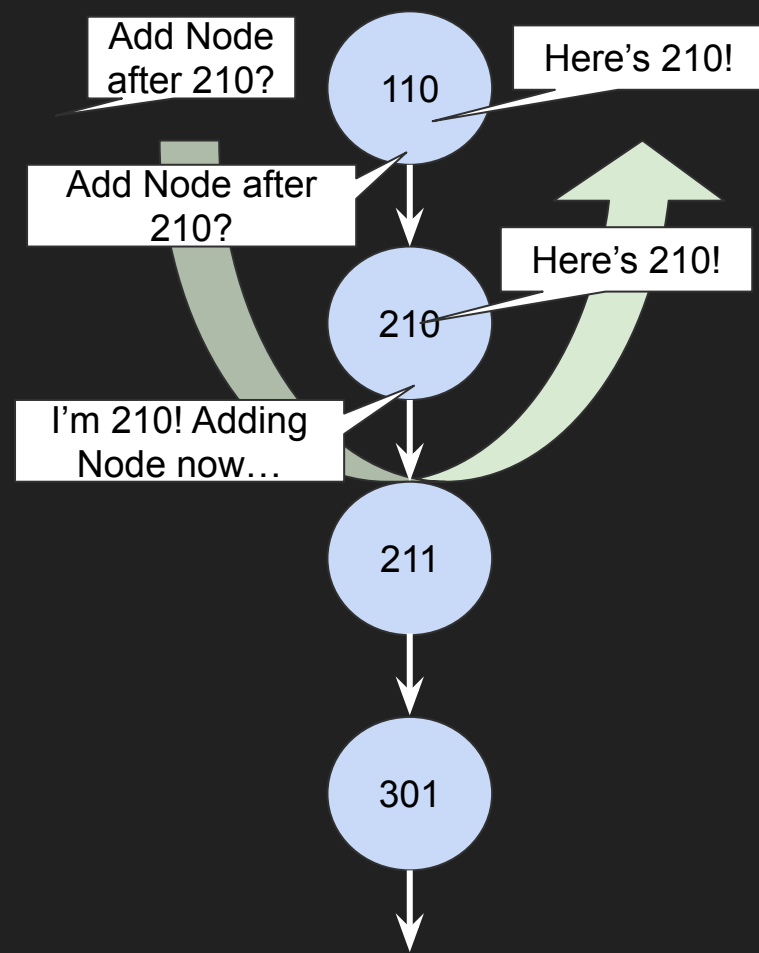
None

`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!