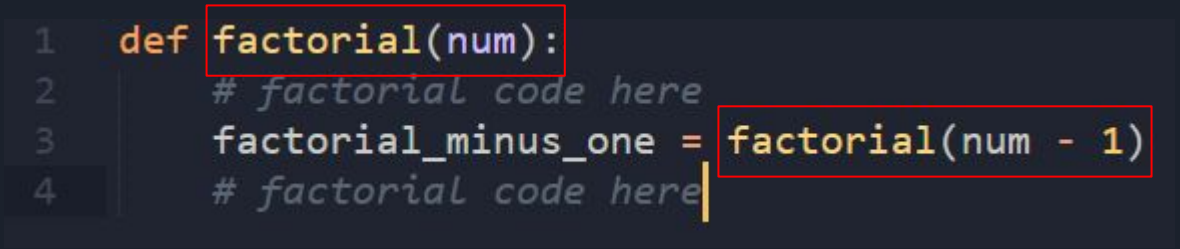
Recursion

What is a Recursive Function?

- By definition, a recursive function is one that calls itself
- Can be thought of as taking a big problem and reducing it to smaller chunks
- When you cannot reduce anymore, then you combine the solutions
  - When you can't combine anymore, it's often called a *base case*



Template for a Recursive Function

- Check for base cases first
- Then, reduce the problem as many times as you can
- Combine all of the reduced results with your arg
- Return combined results
- Note: usually only 1 base case and only 1 reduction/ combination

```
def recursion(arg):
    if arg is at base case 1:
        return base case
    ...
    if arg is at base case n:
        return base case

    reduced_result_1 = recursion(reduction 1 on arg)
    ...
    reduced_result_m = recursion(reduction m on arg)

    combine arg and reduced_result_1 and ... and reduced_result_m
    return combined arg and reduced_results
```

Live Coding: Recursive Sum Function

```python
def recursive_sum(num):
    # Base cases - 2 in this case
    if num == 0:
        return 0
    if num < 0:
        return 0

    # Reduction - only 1 in this case
    reduced_result_1 = recursive_sum(num - 1)

    # Combining reduced result and num
    combined_value = reduced_result_1 + num

    return combined_value
```

Recursive sum walkthrough

recursive_sum(5) => 0 + 1 + 2 + 3 + 4 + 5 recursive_sum(4) + 5 => 10 + 5; return 15

recursive_sum(4) => 0 + 1 + 2 + 3 + 4 recursive_sum(3) + 4 => 6 + 4; return 10

recursive_sum(3) => 0 + 1 + 2 + 3 recursive_sum(2) + 3 => 3 + 3; return 6

recursive_sum(2) => 0 + 1 + 2 recursive_sum(1) + 2 => 1 + 2; return 3

recursive_sum(1) => 0 + 1 recursive_sum(0) + 1 => 0 + 1; return 1

recursive_sum(0) => 0 Problem is at its simplest; return base case (0)

Large Group Project 1: Factorial

Recursive Data Structures

- By definition, a recursive data structure is one that uses itself in its member variables
- Usually, can be thought of as “pointing to” another version of itself
- Linked lists, trees, and graphs can all be implemented using a recursive data structure
- None typically indicates the end of the data structure, if needed

```
class Node:
    def __init__(self, value, next_node):
        self.value = value          # Value of item in List
        self.next_node = next_node  # Must be a Node, or None if the end of list

linked_list = Node(1, Node(2, Node(3, Node(4, None))))
```

Recursive Data Structure (cont.)

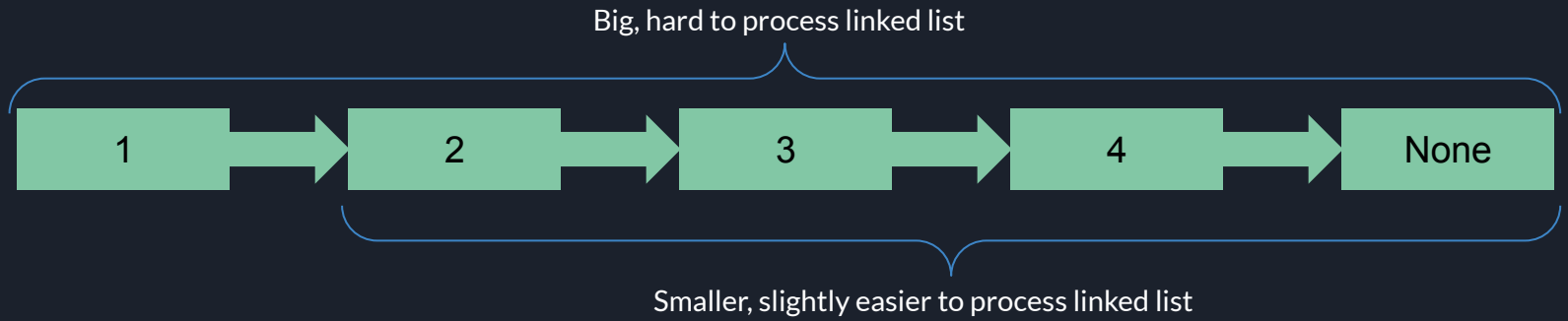
```
linked_list = Node(1, Node(2, Node(3, Node(4, None))))
```

Live Coding: Summing a Linked List

```python
class Node:
    def __init__(self, value, next_node):
        self.value = value          # Value of item in List
        self.next_node = next_node  # Must be a Node, or None if the end of List

def sum_linked_list(linked_list):
    # Base Case
    if linked_list == None:
        return 0

    # Reduction
    reduced_linked_list_sum = sum_linked_list(linked_list.next_node)
    # Combining reduced result and current value
    combined_value = reduced_linked_list_sum + linked_list.value

    # Return combined values
    return combined_value
```

Recursive Functions and Data Structures

- Recursive functions work really well with recursive data structures
- Recursive data structures by their own definition can easily be split up into smaller data structures
- In our linked list example, a large linked list was split into many smaller linked lists until you got to the smallest linked list possible: an empty list represented by None



Large Group Project 2: Find max in Tree

Pair Project: Family Tree