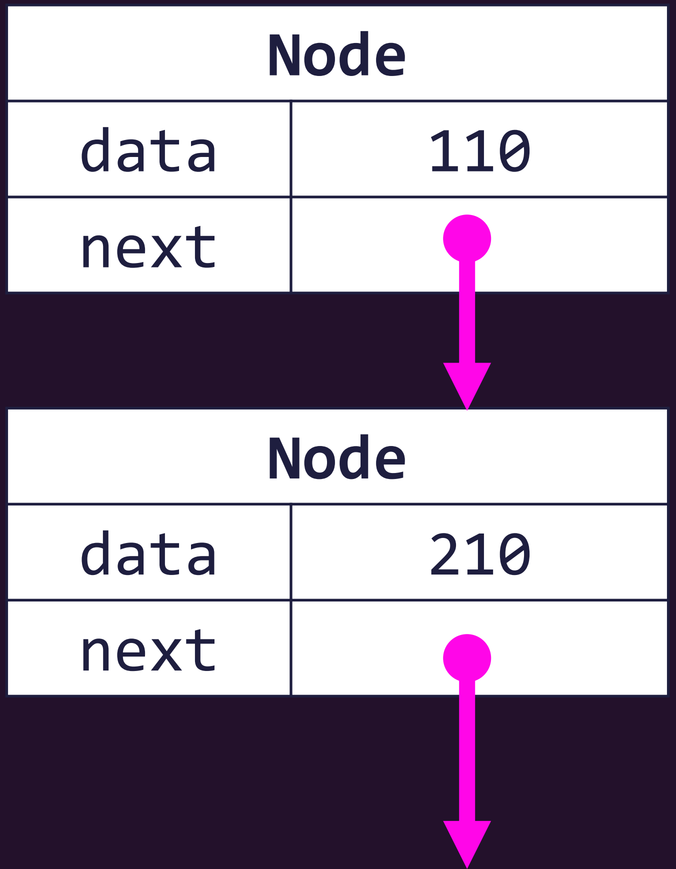
Recursive Structures

in Python

Recursive Data Types

- An attribute *can* refer to another object of the *same type*
- Notice the class **Node**. The attribute named **next** is... *another Node!*
- This is a recursive data type!
- We'll discuss how to initialize a recursive property to avoid infinite recursion shortly...

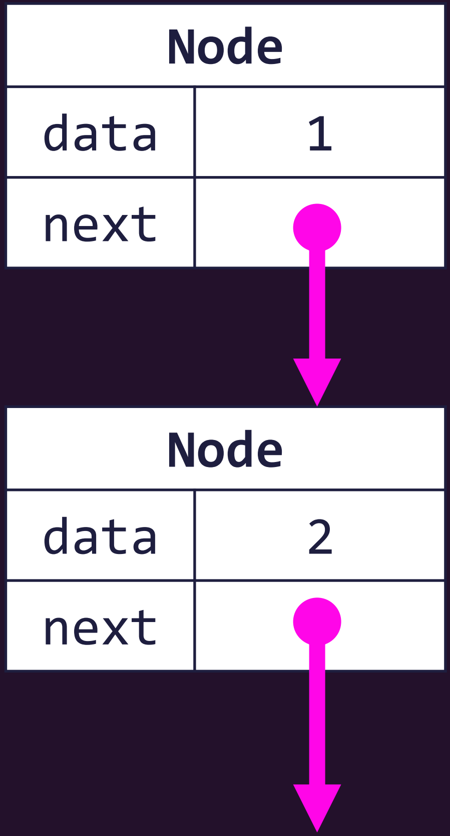
```
class Node:
    data: int
    next: Node
```



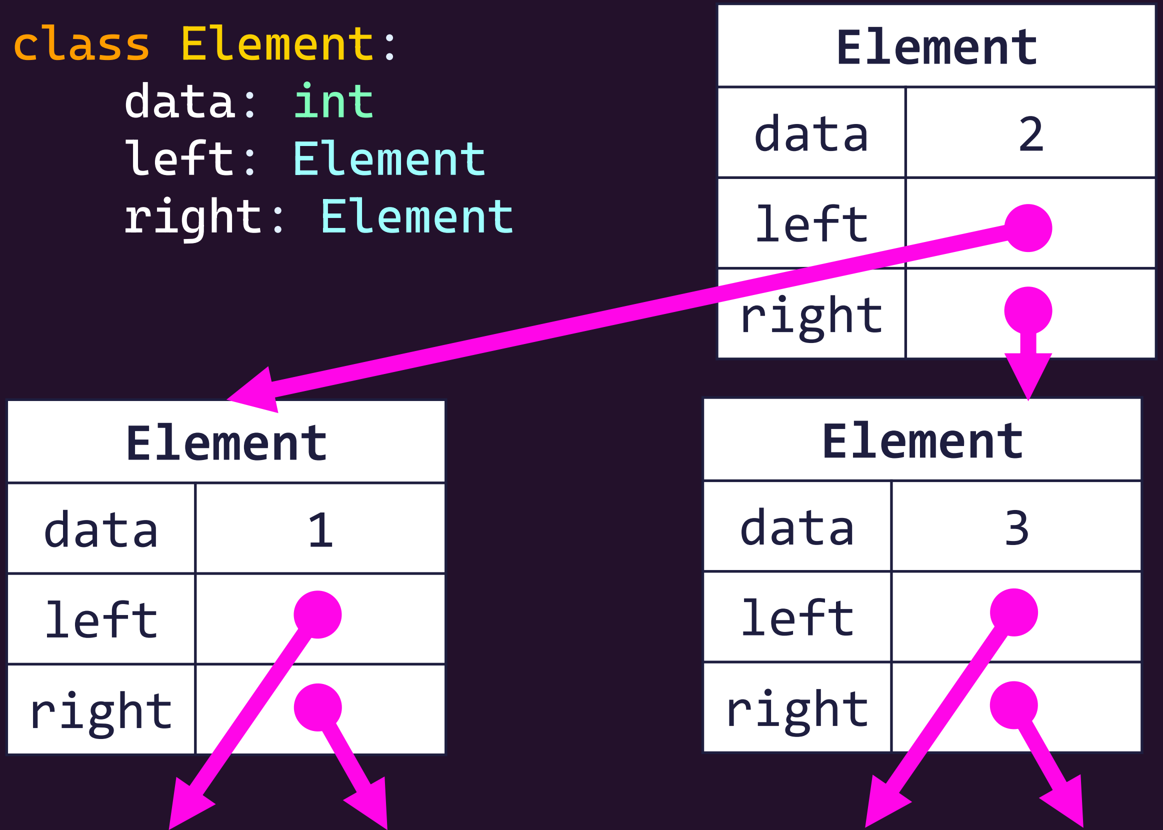
Data Structures

- You can use this ability to form **data structures** with different properties and uses.
- In COMP110, you'll explore the Singly-linked List (left)
- In COMP210, you'll explore other data structures like Trees (right) and Graphs

```
class Node:
    data: int
    next: Node
```




```
class Element:
    data: int
    left: Element
    right: Element
```

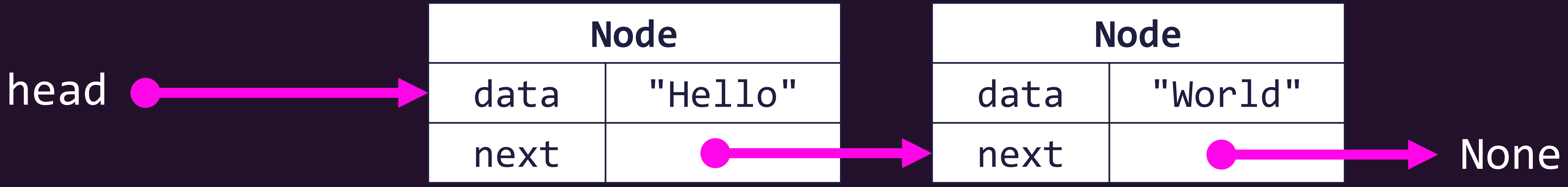


Linked List

- A classic, simple data structure in Computer Science

- Formed by chaining together a sequence of objects
  - The first node is conventionally called the **head**
  - Our focus is on singly-linked lists, meaning a Node only references the Node after it

- Linked Lists are more cumbersome to work with than Python's List
  - However, they're *amazing* for understanding and exploring fundamentals including:
  - **`None / "null"`** values
  - References
  - Recursive algorithms

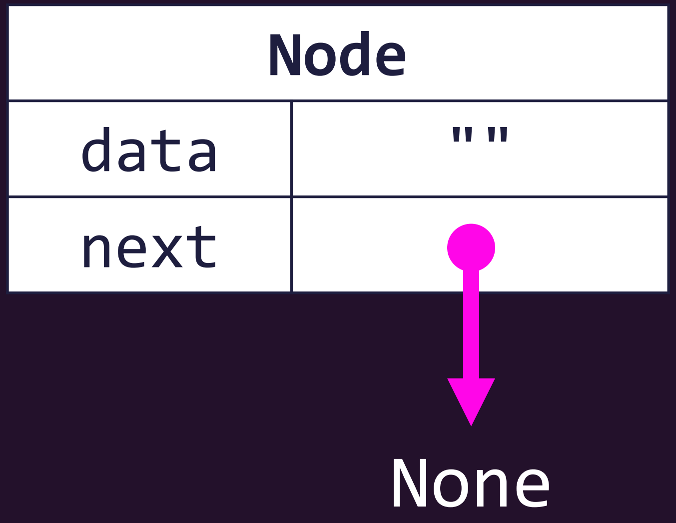


What is a recursive attribute's "base case"?

- If a `Node` refers to a `next` `Node`, and the `next` `Node` refers to another `next` `Node`, then *when does it end?*

```
class Node:
    data: int
    next: Optional[Node]
```

- Recursive attributes are **terminated** with a **None** value.
  - In *many* other languages this is called ***Null***.
  - It is a "reference to nowhere" that you can read as "this attribute refers to nothing."
  - For static typing purposes, we declare `Optional[RecursiveType]`



- Our linked lists is "**None** terminated" or, commonly, "Null terminated"

What are the fundamental operations of our singly Linked List?

1. You can *construct* a new Node at the front of another linked list
   - via the ***`Node`*** constructor

2. You can access a linked list's first value
   - via the ***`data`*** attribute

3. You can access the rest of the list, excluding the first Node
   - via the ***`next`*** attribute

- That's it! These are the fundamental *capabilities* we need.
  - Using these simple operations, you will write more advanced functions, or abstractions, to perform more sophisticated tasks with linked lists.
  - Notice we are intentionally deciding to treat a constructed Node as immutable, we are not going to modify its data or next attributes after construction.

The count Algorithm: Counting Nodes in a Linked List

- How can we write a function that, given a List of any length, we can count the number of elements in it?

- Let's try it with ***pseudo-code*** first!

- **Count Algorithm**, Given any List
  1. *If* the List is empty, *then* the `count` is 0
  2. *Else*, `count` is 1 + the count algorithm applied to *the rest of* the List

Rules of Recursive Algorithms

When processing a recursive data structure recursively:

1. Always test to see if the structure is *empty* (equal to `None`)
   - This is a ***base case***!

2. Make the recursive call on a subpart of the structure
   - With a singly linked list, this is always going to be the *next* Node.

Rules of Recursion using Linked Lists

1. Always check if list is empty! This is the base case.

```
def count(head: Optional[Node]) -> int:
    if head is None:
        return 0
    else:
        after_me = count(head.next)
        return after_me + 1
```

2. Make the recursive call with the *rest of the list.*

Recursive Diagramming

```python
from __future__ import annotations
from typing import Optional

class Node:
    data: int
    next: Optional[Node]

    def __init__(self, data: int, next: Optional[Node]):
        self.data = data
        self.next = next


def count(head: Optional[Node]) -> int:
    if head is None:
        return 0
    else:
        after_me = count(head.next)
        return after_me + 1


n0 = Node(21, None)
n1 = Node(18, n0)
print(count(n1))
```