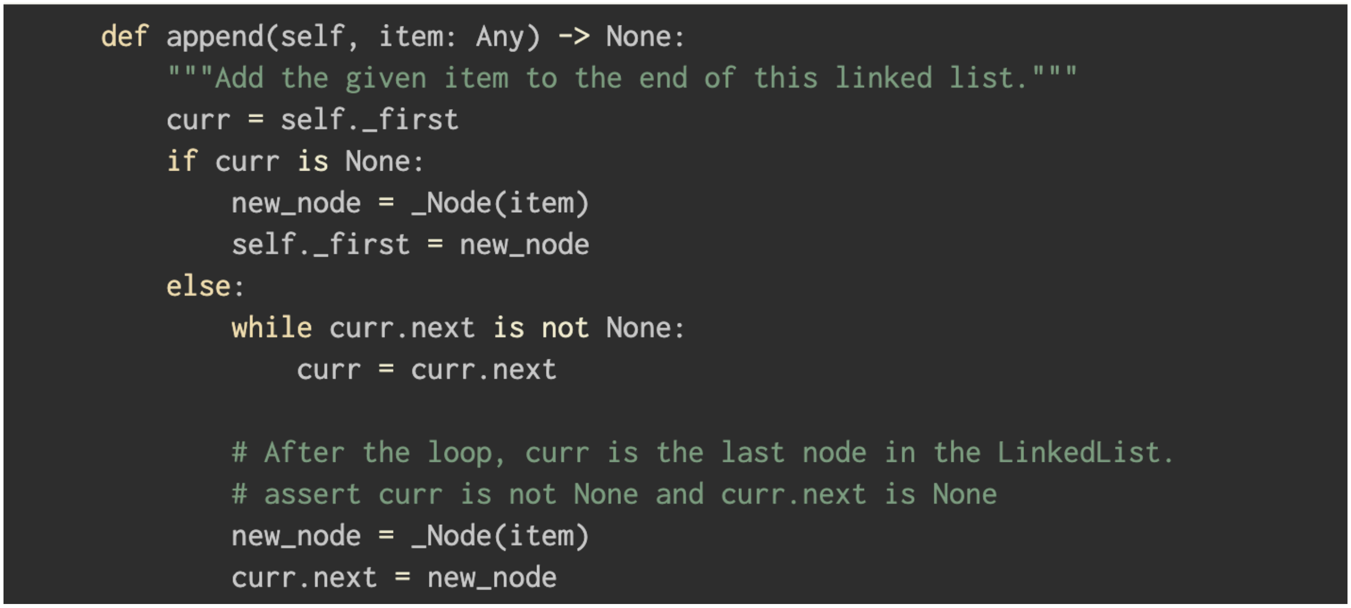
4.3 Linked List Mutation

MAJOR MUTATING OPERATIONS ON A LINKED LIST:

- Inserting items
- Deleting items

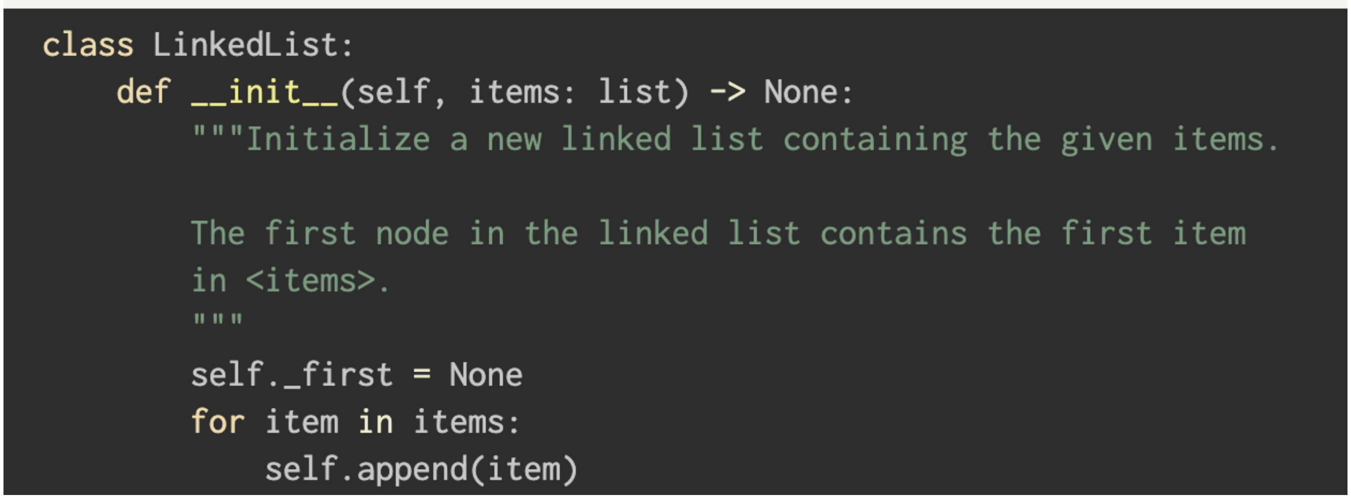
INSERTING ITEMS:

- Appending a new item at the end of a linked list
- Only one attribute: reference to the first node in the list
- Find the last node in the linked list, and then add the item to the end of that.
- To check for the last node we change the condition to `while curr.next is not None`



```
    def append(self, item: Any) -> None:
        """Add the given item to the end of this linked list."""
        curr = self._first
        if curr is None:
            new_node = _Node(item)
            self._first = new_node
        else:
            while curr.next is not None:
                curr = curr.next

            # After the loop, curr is the last node in the LinkedList.
            # assert curr is not None and curr.next is None
            new_node = _Node(item)
            curr.next = new_node
```

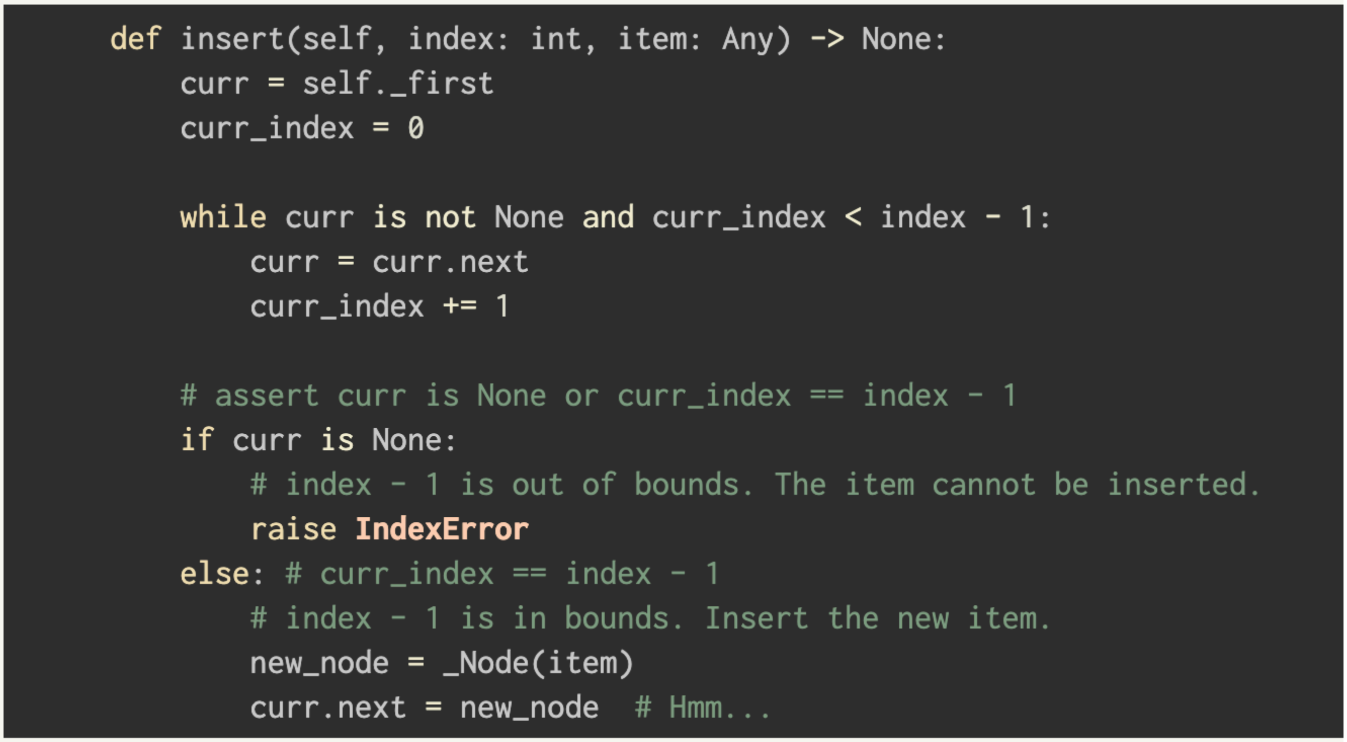
- A more general initializer that takes in a list of values, which are appended one at a time



```
class LinkedList:
    def __init__(self, items: list) -> None:
        """Initialize a new linked list containing the given items.

        The first node in the linked list contains the first item
        in <items>.
        """
        self._first = None
        for item in items:
            self.append(item)
```

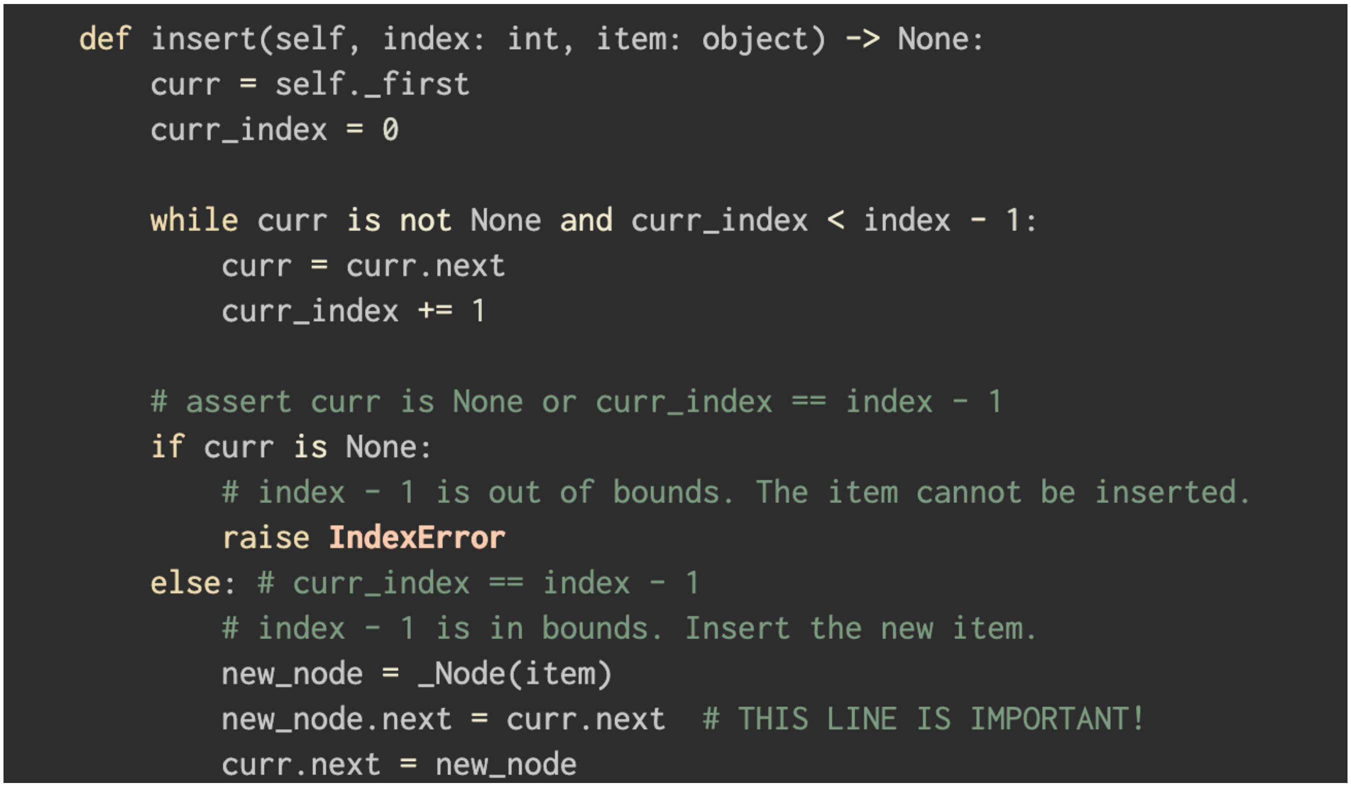
INDEX-BASED INSERTION:

- Transverse the list until we reach the correct index
- We need insert into position `index` and access the node at position (`index-1`)
- If curr None then the list doesn’t have a node at position index-1 (*so index is out of bounds*)
- If curr is not None then we have reached desired index & can append the new node.



```
    def insert(self, index: int, item: Any) -> None:
        curr = self._first
        curr_index = 0

        while curr is not None and curr_index < index - 1:
            curr = curr.next
            curr_index += 1

        # assert curr is None or curr_index == index - 1
        if curr is None:
            # index - 1 is out of bounds. The item cannot be inserted.
            raise IndexError
        else: # curr_index == index - 1
            # index - 1 is in bounds. Insert the new item.
            new_node = _Node(item)
            curr.next = new_node  # Hmm...
```

- **Since curr might have other nodes after it, its important to store the old nodes, so that we don’t lose the reference to the old node at position index.**

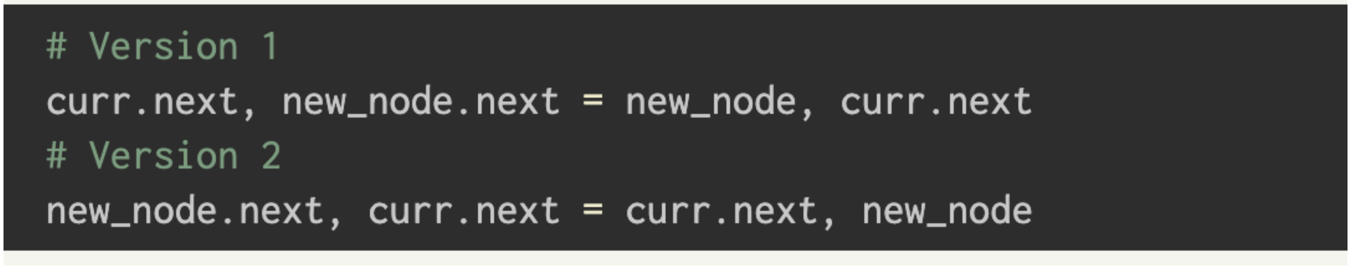


```python
    def insert(self, index: int, item: object) -> None:
        curr = self._first
        curr_index = 0

        while curr is not None and curr_index < index - 1:
            curr = curr.next
            curr_index += 1

        # assert curr is None or curr_index == index - 1
        if curr is None:
            # index - 1 is out of bounds. The item cannot be inserted.
            raise IndexError
        else: # curr_index == index - 1
            # index - 1 is in bounds. Insert the new item.
            new_node = _Node(item)
            new_node.next = curr.next  # THIS LINE IS IMPORTANT!
            curr.next = new_node
```

- The order in which we update the links REALLY MATTERS!!!! -> ONLY ONE ORDER results in the correct behaviour

MULTIPLE ASSIGNMENT IN PYTHON:

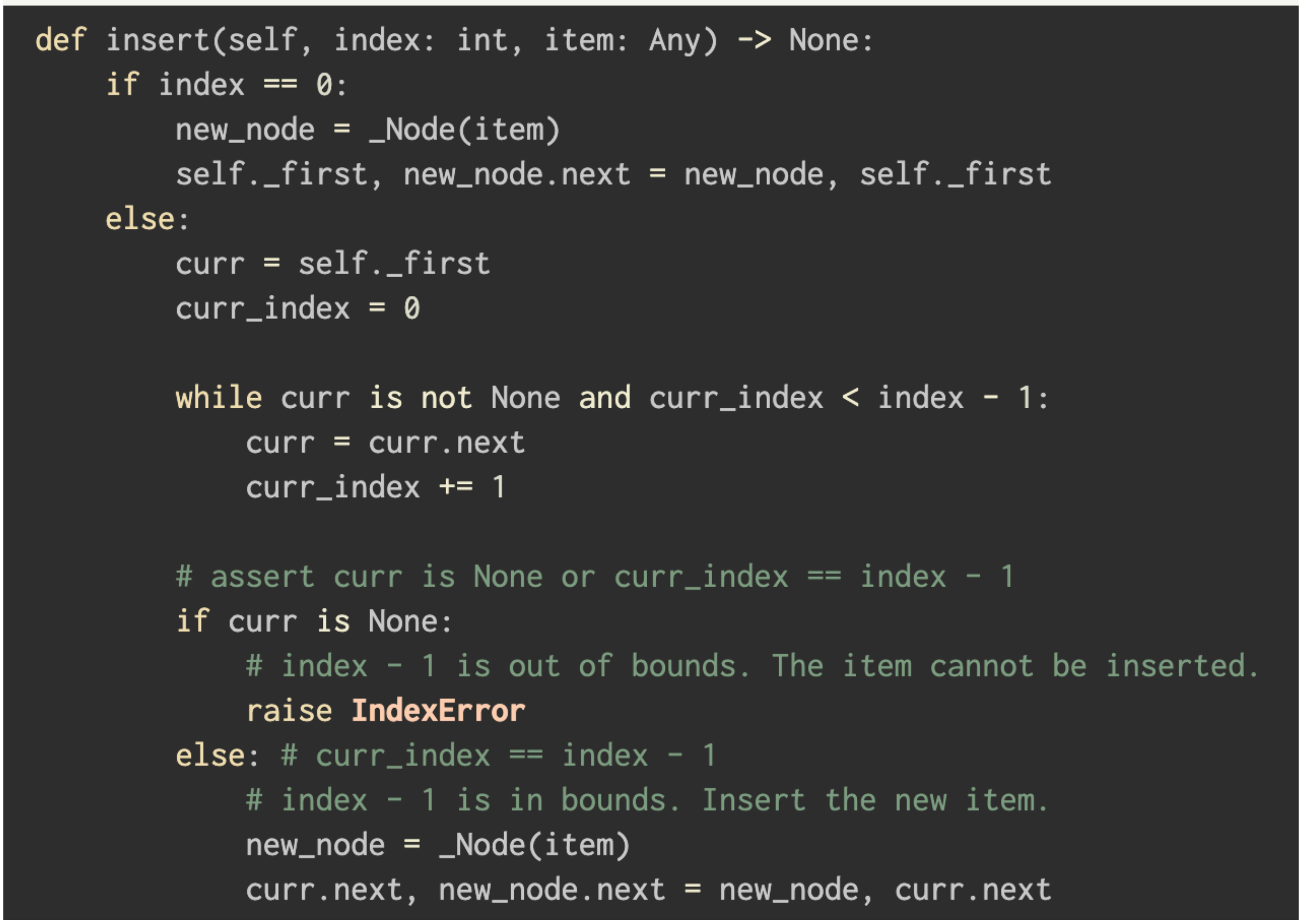
- This is soooo cooooooooool😎😎😎🙌🙌🙌🤘🔥🔥🔥
- Since the expressions on the right side are evaluated before any new values are assigned, You don’t need to worry about the order in which they are written !!



```python
# Version 1
curr.next, new_node.next = new_node, curr.next
# Version 2
new_node.next, curr.next = curr.next, new_node
```

CORNER CASES TO THINK ABOUT:

- What if the index = 0 -> doesn’t make sense to iterate to the (`index-1`)-th node
- Modifying `self._first` since we are inserting it in the front of the list.



```
def insert(self, index: int, item: Any) -> None:
    if index == 0:
        new_node = _Node(item)
        self._first, new_node.next = new_node, self._first
    else:
        curr = self._first
        curr_index = 0

        while curr is not None and curr_index < index - 1:
            curr = curr.next
            curr_index += 1

        # assert curr is None or curr_index == index - 1
        if curr is None:
            # index - 1 is out of bounds. The item cannot be inserted.
            raise IndexError
        else: # curr_index == index - 1
            # index - 1 is in bounds. Insert the new item.
            new_node = _Node(item)
            curr.next, new_node.next = new_node, curr.next
```