Linked Lists: Takeaways

Syntax

- Linked list node implementation:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None
```

- Linked list implementation:

```
class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None
        self.length = 0
    def append(self, data):
        new_node = Node(data)
        if self.length == 0:
            self.head = self.tail = new_node
        else:
            self.tail.next = new_node
            new_node.prev = self.tail
            self.tail = new_node
        self.length += 1
    def __iter__(self):
        self._iter_node = self.head
        return self
    def __next__(self):
        if self._iter_node is None:
            raise StopIteration
        ret = self._iter_node.data
        self._iter_node = self._iter_node.next
        return ret
    def prepend(self, data):
        new_node = Node(data)
        if self.length == 0:
            self.head = self.tail = new_node
        else:
            self.head.prev = new_node
            new_node.next = self.head
            self.head = new_node
        self.length += 1
    def __len__(self):
```

```
        return self.length
    def __str__(self):
        return str([value for value in self])
```

Concepts

- We implement linked lists by linking nodes. Unlike array-based implementations, we can't access elements by index in constant time. To do so, we would need to iterate over the nodes from the head to the index we want to access.
- Linked lists allow us to append and prepend elements in constant time.
- An iterable is any Python object capable of returning its members one at a time, permitting us to iterate over it in a for-loop.
- To make an object iterable, we implement the `__iter__()` and `__next__()` method. The `__iter__()` method should initialize the necessary data to start the iteration. The `__next__()` method should return the current iteration value and move to the next value. It should raise a `StopIteration` exception when the iteration is over.
- To use the `len()` built-in function on a custom class, we need to implement the `__len__()` method.
- To provide a readable string representation of an object, we need to implement the `__str__()` method.

Resources

- Linked Lists
- Iterables
- `__iter__()` method
- `__next__()` method
- `__len__()` method
- `__str__()` method