import random

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def append(self, data):

if not isinstance(data, int):

raise ValueError("Data must be an integer.")

new\_node = Node(data)

if self.head is None:

self.head = new\_node

return

last\_node = self.head

while last\_node.next:

last\_node = last\_node.next

last\_node.next = new\_node

def print\_list(self):

curr\_node = self.head

while curr\_node:

print(curr\_node.data)

curr\_node = curr\_node.next

def length(self):

curr\_node = self.head

count = 0

while curr\_node:

count += 1

curr\_node = curr\_node.next

return count

def print\_all\_elements(self):

curr\_node = self.head

while curr\_node:

print(curr\_node.data)

curr\_node = curr\_node.next

def main():

llist = LinkedList()

for \_ in range(10):

llist.append(random.randint(1,100))

print("Length of linked list:",llist.length())

llist.print\_all\_elements()

if \_\_name\_\_ == "\_\_main\_\_":

main()

In diesem Code habe ich eine eigene main() Methode hinzugefügt. Diese Methode demonstriert die Verwendung der Klasse LinkedList, indem sie eine neue Instanz der Klasse erstellt und mit zufälligen Ganzzahlwerten (1-100) gefüllt wird. Anschließend wird die Länge der Liste ausgegeben und die Elemente der Liste werden mithilfe der Methode print\_all\_elements() ausgegeben. Das if \_\_name\_\_ == "\_\_main\_\_"-Block ruft die main Methode auf. Dieser Ansatz ermöglicht es Ihnen, Ihre Code-Struktur zu organisieren, indem er die Ausführungslogik von der Definition der Klassen und Methoden trennt und es Ihnen ermöglicht, Ihre Anwendung leichter zu testen und wiederverwenden.

implementiere auch noch andere listenmethoden

2 / 2

Hier sind einige Beispiele für andere Methoden, die Sie der Klasse LinkedList hinzufügen könnten:

* prepend(data): Fügt ein neues Element am Anfang der Liste hinzu.
* delete(data): Entfernt das erste Element mit dem gegebenen Wert aus der Liste.
* find(data): Gibt das erste Element mit dem gegebenen Wert zurück oder gibt eine Fehlermeldung aus, wenn der Wert nicht gefunden wird.
* insert\_after(node, data): Fügt ein neues Element nach dem gegebenen Knoten in der Liste ein.
* swap(node1, node2): Vertauscht die Daten der gegebenen Knoten in der Liste.
* reverse(): Kehrt die Reihenfolge der Elemente in der Liste um.

Hier ist ein Beispiel für die Implementierung der Methode prepend:

class LinkedList:

# ...

def prepend(self, data):

if not isinstance(data, int):

raise ValueError("Data must be an integer.")

new\_node = Node(data)

new\_node.next = self.head

self.head = new\_node

Diese Methode erst