**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Информатика»**

Тема: **Парадигмы программирования**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8381 |  | Киреев К.А. |
| Преподаватель |  | Размочаева Н.В. |

Санкт-Петербург

2018

**Цель работы**

Изучить основы ООП в Python; научиться создавать собственные классы, а также двусвязные списки и использовать их; изучить понятия императивного и декларативного подхода; научиться применять обработчик исключений.

**Задание**

Реализуйте два класса: класс Node и класс LinkedList, представляющие собой элемент и связный двунаправленный список.

Класс Node должен иметь 3 поля:

*\_\_*data *# данные, приватное поле*

prev *# ссылка на предыдущий элемент списка*

next *# ссылка на следующий элемент списка*

Вам необходимо реализовать следующие методы в классе Node:

\_\_init\_\_(self, data, prev, next)

*конструктор, со значениями по умолчанию для переменных prev и next None.*

getData(self)

*метод возвращает значение поля \_\_data*

\_\_str\_\_(self)

*перегрузка метода \_\_str\_\_.*

Класс LinkedList должен иметь 3 поля:

\_\_length # *длина списка*

first # *данные первого элемента списка*

last # *данные последнего элемента списка*

Вам необходимо реализовать следующие методы в классе LinkedList:

\_\_init\_\_(self, first, last)

*конструктор, со значениями по умолчанию для переменных first и last None.*

*Если значение переменной first равно None, а переменной last не равно None, метод должен вызывать исключение ValueError с сообщением: "invalid value for last".*

*Если значение переменной first не равно None, а переменной last равна None, метод должен создавать список из одного элемента. В данном случае, first равен last, ссылки prev и next равны None, значение поля \_\_data для элемента списка равно first.*

*Если значения переменных не равны None, необходимо создать список из двух элементов. В таком случае, значение поля \_\_data для первого элемента списка равно first, значение поля \_\_data для второго элемента списка равно last.*

\_\_len\_\_(self) *# перегрузка метода \_\_len\_\_.*

append(self, element)

*добавление элемента в конец списка. Метод должен создать объект класса Node, у которого значение поля \_\_data будет равно element и добавить этот объект в конец списка.*

\_\_str\_\_(self) *# перегрузка метода \_\_str\_\_.*

pop(self)

*удаление последнего элемента. Метод должен выбрасывать исключение IndexError с сообщением "LinkedList is empty!", если список пустой.*

popitem(self, element)

*удаление элемента, у которого значение поля \_\_data равно element. Метод должен выбрасывать исключение KeyError, с сообщением "<element> doesn't exist!", если элемента в списке нет.*

clear(self)

*очищение списка.*

**Выполнение работы**

1. Созданы два класса Node и LinkedList.
2. Node используется для реализации элемента связного двунаправленного списка. В данном классе созданы три метода: \_\_init\_\_() – конструктор; getData(), который возвращает значение поля \_\_data; \_\_str\_\_() – преобразовывает объект в текстовый вид.
3. LinkedList – связный список. В качестве элементов класса используются экземпляры Node. В данном классе созданы семь методов: \_\_init\_\_() – конструктор; \_\_len\_\_(), который переопределяет функцию получения длины объекта; append() - добавляет элемент в конец списка. Метод создает объект класса Node, у которого значение поля \_\_data будет равно element и добавляет этот объект в конец списка; \_\_str\_\_() – преобразовывает объект в текстовый вид; pop() – удаление одного узла спереди списка; popitem() – удаляет элемент, у которого значение поля \_\_data равно element; clear() – очищает список.

**Выводы**

В ходе лабораторной работы были получены навыки реализации классов и двусвязных списков в Python и их использования. Был реализован двусвязный список. Также был изучен обработчик исключений.

**Приложение А. Исходный код**

class Node():

\_\_data = None

prev = None

next = None

def \_\_init\_\_(self, data, prev = None, next = None):

self.\_\_data = data

self.prev = prev

self.next = next

def getData(self):

return self.\_\_data

def \_\_str\_\_(self):

end\_str = ''

end\_str += "data: " + str(self.getData()) + ", "

if self.prev != None:

end\_str += "prev: " + str(self.prev.getData()) + ", "

else:

end\_str += "prev: " + "None" + ", "

if self.next != None:

end\_str += "next: " + str(self.next.getData())

else:

end\_str += "next: " + "None"

return end\_str

class LinkedList():

\_\_length = 0

first = None

last = None

def \_\_init\_\_(self, first = None, last = None):

if (first == None) and (last != None):

raise ValueError("invalid value for last")

elif first != None:

if last != None:

self.first = Node(first, None, None)

self.last = self.first.next = Node(last, self.first, None)

self.\_\_length = 2

else:

self.first = self.last = Node(first, None, None)

self.\_\_length = 0

def \_\_len\_\_(self):

return self.\_\_length

def append(self, element):

obj = Node(element, self.last, None)

if self.\_\_length == 0:

self.first = obj

else:

self.last.next = obj

self.last = obj

self.\_\_length += 1

def \_\_str\_\_(self):

result = []

end\_str = ''

if len(self) != 0:

info = self.first

for i in range(self.\_\_length):

result.append(str(info))

info = info.next

res = "; ".join(result)

end\_str += "LinkedList[length = " + str(len(self)) + ", [" + res + "]]"

return end\_str

else:

return "LinkedList[]"

def pop(self):

if len(self) == 0:

raise IndexError("LinkedList is empty!")

elif len(self) != 0:

self.last = self.last.prev

self.last.next = None

self.\_\_length -= 1

def popitem(self, element):

info = self.first

if len(self) != 0:

while info.getData() != element:

if info == self.last:

raise KeyError(str(element) + " doesn't exist!")

info = info.next

else:

raise KeyError(str(element) + " doesn't exist!")

if len(self) == 1:

self.first = self.last = None

elif info == self.first:

self.first = info.next

self.first.prev = None

elif info == self.last:

self.last = self.last.prev

self.last.next = None

else:

info.prev.next = info.next

info.next.prev = info.prev

self.\_\_length -= 1

def clear(self):

self.first = None

self.last = None

self.\_\_length = 0