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 _____ Почаев Н.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2018

Цель работы

Реализовать собственные классы и на их основе научиться строить такую структуру данных, как двунаправленный связный список. Научиться перегружать методы классов и работать с их атрибутами. Изучить основы ООП на языке Python и его особенности. Освоить работы с исключениями: «бросать» и «ловить» их.

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)

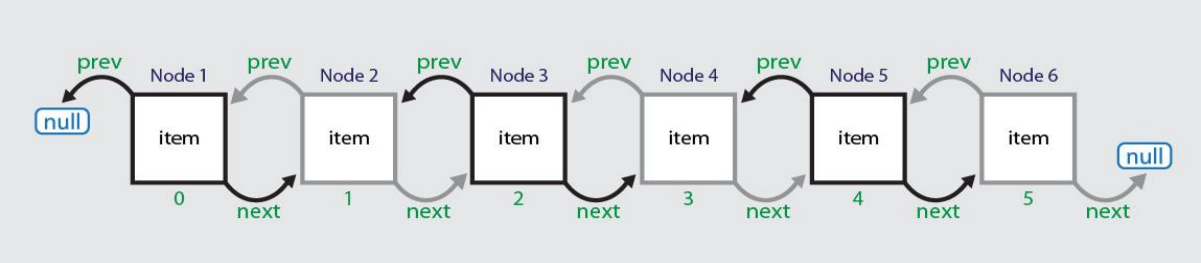
очищение списка.

Выполнение работы

Написание работы производилось на базе операционной системы Linux Arch в интегрированной среде разработки PyCharm.

Ход выполнения работы:

1. В начале был создан класс Node, представляющий собой класс узла – контейнера, хранящего значение элемента и ссылки на следующий/предыдущий элементы списка. По умолчанию поле Data является приватным (в функциональности Python её имя искажено).
2. В данном классе были созданы: конструктор, геттер и переопределение функции str. Первый - специальный блок инструкций, вызываемый при создании объекта. Второй - специальный метод, позволяющий получить данные, доступ к которым напрямую ограничен, в данном случае, поля Data. Переопределение же метода str позволяет преобразовывать объект в строку вида: data: __, prev: __, next: __.
3. Далее был создан класс LinkedList представляющий собой реализацию реализацию двунаправленного списка. Общая схема его работы приведена на рисунке:



Данный класс содержит поля: ссылка на первый/последний элементы списка и его длину (также является приватным полем).

4. В данном классе переопределён метод len, он возвращает текущую длину списка.
5. Конструктор данного класса имеет несколько путей работы: в случае некорректных входных значений (first=None и last != None) он кидает исключение ValueError("invalid value for last"). В других случаях в зависимости от входных значений, он создаёт список из одного/двух элементов.

6. Далее был переопределён метод str для приведения списка к строковому виду. В начале идёт проверка на ненулевую длину, затем в цикле в результирующий список записываются приведённую к строковому представлению элементы двунаправленного списка (т.е. узлы класса Node). Возвращаемое значение генерируется путём конкатенации требуемых элементов строки и содержимого списка res путём применения метода join.
7. Для очистки списка в ходе работы создан отдельный метод clean.
8. Также был создан метод append, позволяющий добавлять в список новое числовое значение. Если длина текущего списка равна 0, то новый элемент становится и последним и первым, иначе он встривается в двусвязный список, где предыдущий элемент начинает указывать на него, а он, в свою очередь, становится последним элементом, указывающим на предыдущий элемент и на Null.
9. В данном классе реализовано два метода для удаления элементов: pop – для удаления элементов из конца (спереди) списка и popitem – для удаления элемента из любой части списка.

   Описание первого: в начале идёт на ненулевую длину списка, иначе «бросается» исключение IndexError("LinkedList is empty!"). Далее, если длина равна 1, то ссылки на следующий и предыдущий элементы становятся равными None , а длина уменьшается на единицу (т.е. до нуля). В ином случае переназначаем последний элемент, как последний, его ссылка на следующий становится None и длина также уменьшается.

   Описание второго: также происходит проверка на ненулевую длину списка, иначе «бросается» исключение KeyError(str(element) + " doesn't exist!"). Оно также вызывается в случае отстутсвия необходимого для удаления элемента в поле Data хотя бы одного элемента (удаляется первый встретившийся). Если список состоит из одного элемента, то он очищается. Далее отдельно обрабатываются случаи, когда удаляемый элемент является первым/последним (происходит соответсвенное перевешивание

ссылок). Иначе просто перевешиваются ссылки на следующий/предыдущий элементы соседних элементов, минуя текущий, тем самым удаляя элемент.

Выводы

В ходе лабораторной работы были изучены основыреализации ООП на языке программирования Python. Были получены знания по перегрузке методов, созданию «приватных» полей и т.д. Также была освоена работа с исключениями. В результате был реализован двусвязный список, содержащий основные методы для работы с ним и исключения для корректной работы с ним.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
# класс узла - контейнера, хранящего значение и ссылку на след/пред эл.
class Node:
    __data = None  # искажение имени (приватное поле)
    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

    # переопределение метода пребразования объекта в строку [data: __,
prev: __, next: __]
    def __str__(self):
        out_for_s = "data: " + str(self.getData())  # юзаем геттер
        out_for_prev = ", prev: " + (str(self.prev.getData()) if
self.prev is not None else "None")
        out_for_next = ", next: " + (str(self.next.getData()) if
self.next is not None else "None")
        return out_for_s + out_for_prev + out_for_next


'''
Каждый узел содержит элемент, ссылку на предыдущий и следующий узел.
Связанный список состоит из последовательности узлов,
каждый из которых предназначен для хранения объекта определенного при
создании типа.
Также хранит свою длину.
'''


class LinkedList:
    __length = 0
    first = None
    last = None
```

```
    # переопределение метода получения длины
    def __len__(self):
        return self.__length

    def __init__(self, first=None, last=None):
        if first is None:
            if last is not None:
                raise ValueError("invalid value for last")
        else:
            f_node = Node(first)
            self.first = f_node
            if last is None:
                self.last = f_node  # список из одного элемента
                self.__length = 1
            else:
                l_node = Node(last, self.first)  # список двунаправный ->
ссылка на пред.
                self.first.next = l_node  # ссылка на предыдущий эл.
                self.last = l_node
                self.__length = 2

    '''преобразование объекта к строковому представлению.
    Вызывается, когда объект передается функциям print() и str()'''
    def __str__(self):
        res = []
        if len(self) == 0:
            return "LinkedList[]"
        else:
            now_el = self.first  # берём первый эл.
            while now_el != self.last:
                res.append(str(now_el))  # приводим элемент к строковому
представлению
                now_el = now_el.next
            res.append(str(now_el))  # последний эл.
            return "LinkedList[length = " + str(len(self)) + ", [" + ";
".join(res) + "]]"
```

```
    # очистка списка
    def clear(self):
        self.first = None
        self.last = None
        self.__length = 0

    # метод добавления в список одного узлового эл. спереди
    def append(self, element):
        new_el = Node(element, self.last)  # передаём в генератор класса
значение Data и ссылку на последний. эл.
        if len(self) == 0:
            self.first = new_el
        else:
            self.last.next = new_el  # ссылка последнего элемента на
следующий
        self.last = new_el
        self.__length += 1

    # метод удаления одного узлового эл. спереди списка
    def pop(self):
        if len(self) == 0:
            raise IndexError("LinkedList is empty!")
        elif len(self) == 1:
            self.first.prev = None
            self.first.next = None
            self.__length -= 1
        elif len(self) > 1:
            self.last = self.last.prev  # перевешиваем последий элемент
на предпоследний
            self.last.next = None  # новый последний эл. указывает на
None
            self.__length -= 1

    def popitem(self, element):
        now_el = self.first
        if len(self) == 0:
            raise KeyError(str(element) + " doesn't exist!")
        # находим элемент с need Data
```

```
        while now_el.getData() != element:
            if now_el == self.last:
                raise KeyError(str(element) + " doesn't exist!")
            now_el = now_el.next
        if len(self) == 1:
            self.clear()
            self.__length = 0
        # если первый элемент содержит Data
        elif now_el == self.first:
            self.first = now_el.next  # перенавешиваем указатель на след.
эл.
            self.first.prev = None  # у ного первого эл. указатель на
предыдущий None
            self.__length -= 1
        # если последний элемент содержит Data
        elif now_el == self.last:
            self.last = self.last.prev  # перенавешиваем указатель на
предыдущий эл.
            self.last.next = None  # у последнего элемента указатель на
след. None
            self.__length -= 1
        else:
            now_el.prev.next = now_el.next  # указатель на следующий эл.
предыдущего эл. перенавешиваем, минуя текущий
            now_el.next.prev = now_el.prev  # указатель на предыдущий эл.
след. эл перенавешиваем , минуя текущий
            self.__length -= 1
```