Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Отчет по лабораторной работе №4**

Дисциплина: «Классы»

Вариант № 10

Выполнила:

студентка группы ИВТАСбд-21

Микка Е. И.

Проверил:

преподаватель кафедры

«Вычислительная техника»

Исхаков И.И.

Ульяновск, 2023

**Задание по варианту №10.**

Необходимо переделать лабораторную работу №3 с использованием классов, описывающих предметную область, с реализацией следующих особенностей:

* класс должен содержать итератор,
* должна быть реализована перегрузка стандартных функций,
* должно быть реализовано наследование,
* запись значений в свойства с помощью \_\_setattr\_\_,
* возможность доступа к элементам по индексу (\_\_getitem\_\_),
* должны быть реализованы статистические методы,
* должны быть реализованы генераторы.

**Описание реализации.**

Итератором называется объект, которой возвращает свои элементы по одному за раз. Для реализации итератора в Python необходимо реализовать метод \_\_iter\_\_ и метод \_\_next\_\_. Перегрузка стандартных функций \_\_iter\_\_ и \_\_next\_\_:

def \_\_iter\_\_(self):  
 self.n = len(self.\_\_dict\_\_)  
 self.k = 0  
 return self

def \_\_next\_\_(self):  
 if self.k < self.n:  
 nx = self.\_\_dict\_\_[str(self.k)]  
 self.k += 1  
 return nx  
 else:  
 print('Итератор опусташен')

Запись значений в свойства должна осуществляться только через \_\_setattr\_\_. Полученные значения записываются в свойство self.\_\_dict\_\_.

def \_\_setattr\_\_(self, key, value):  
 self.\_\_dict\_\_[str(key)] = value

Получение элемента по индексу должно осуществляться с помощью метода \_\_getitem\_\_.

def \_\_getitem\_\_(self, item):  
 self.n = len(self.\_\_dict\_\_)  
 if item < self.n:  
 return self.\_\_dict\_\_[item]

Генератор в данной программе реализован через метод gen. Реализован генератор следующим образом:

def gen(self):  
 for i in self.\_\_dict\_\_:  
 yield self.\_\_dict\_\_[str(i)]

Наследование в программе реализовано следующим образом: создаёт первый класс Dict, он включает в себя такие стандартные функции, как \_\_setattr\_\_, \_\_getitem\_\_, \_\_next\_\_ и \_\_iter\_\_. После создаётся наследуемый класс Ex\_dict, который расширяет возможности класса-предка. В классе Ex\_dict реализовываются такие стандартные функции, как \_\_repr\_\_ и \_\_str\_\_, а также статистические методы select и copy. Экземпляр, с которым работает программа, создан от класса Ex\_dict и может использовать не только методы своего класса, но класса–предка.

Реализация стандартных функций \_\_repr\_\_ и \_\_str\_\_:

def \_\_repr\_\_(self):  
 return f"{self.\_\_class\_\_}:{self.\_\_dict\_\_}"  
  
def \_\_str\_\_(self):  
 return f"{self.\_\_dict\_\_}"

Реализация статистических методов select и copy:

@staticmethod  
def copy(self):  
 dct = {}  
 lst = []  
 for i in self.\_\_dict\_\_:  
 for j in range(5):  
 if 1 < j < 4:  
 lst.append(self.\_\_dict\_\_[i][j])  
 else:  
 lst.append(int(self.\_\_dict\_\_[i][j]))  
 dct[int(i)] = lst  
 lst = []  
 return dct  
  
@staticmethod  
def select(self):  
 lst = []  
 for i in self.\_\_dict\_\_:  
 if int(self.\_\_dict\_\_[i][4]) > 4:  
 lst.append(int(i))  
 return lst

**Описание возникших затруднений.**

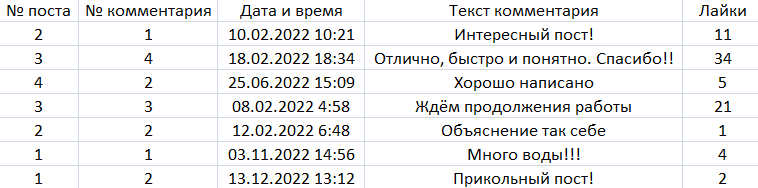
Реализация стандартной функции \_\_setattr\_\_ вызвала наибольшие затруднения. Трудности возникали на этапе записи данных в свойства экземпляра. Все записи сохранялись под единым ключом, отчего в словарь сохранялась лишь одна запись из всех имеющихся. Для устранения данной проблемы в метод \_\_setattr\_\_ передаётся целочисленная, постоянно увеличивающиеся на единицу, переменная, которая и служит ключом для self.\_\_dict\_\_.

**Описание альтернативных способов решения.**

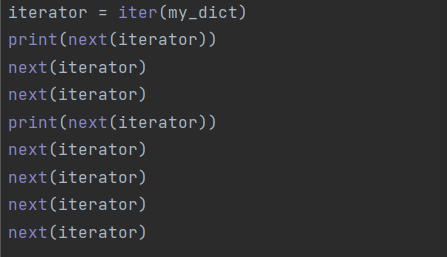
Перегрузка стандартных функций может быть реализована иначе, самое главное, чтобы каждый метод действовал определённым образом. Например, метод \_\_next\_\_ должен возвращать следующий элемент коллекции относительно текущего, данную задачу можно реализовать с помощью иного кода.

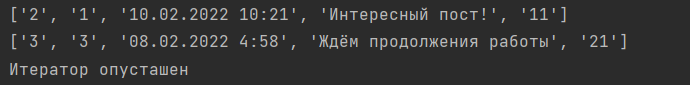
**Пример работы программы.**

Исходный файл:

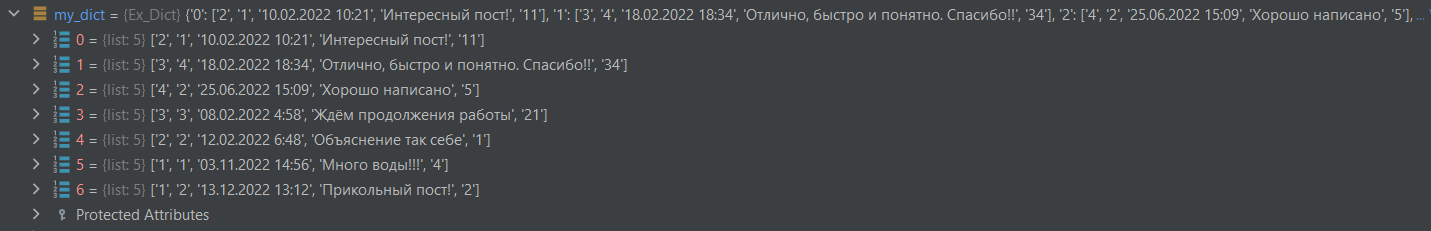


Пример работы итератора:

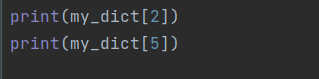


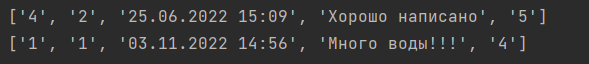
****

Пример работы метода \_\_setattr\_:

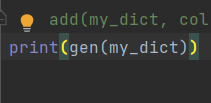


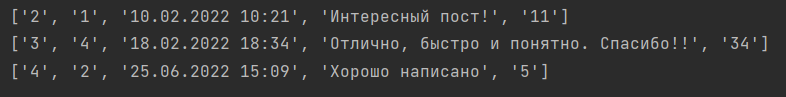
Пример работы метода \_\_getitem\_\_:





Пример работы генератора:

**C:\Users\eliza\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Снимок экрана (699).png**

****

