МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Информатика»

Тема: Парадигмы программирования

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2020

Цель работы.

Используя язык Python , изучить основы парадигмы программирования .

Задание.

Базовый класс -- схема дома HouseScheme:

```
class HouseScheme:
    ''' Поля объекта класса HouseScheme:
    количество жилых комнат
    площадь (в квадратных метрах, не может быть отрицательной)
    совмещенный санузел (значениями могут быть или False, или True)
    При создании экземпляра класса HouseScheme необходимо
        убедиться, что переданные в конструктор параметры
удовлетворяют требованиям, иначе выбросить исключение ValueError с
текстом
    'Invalid value'
    '''
```

Дом деревенский CountryHouse:

```
class CountryHouse: # Класс должен наследоваться от HouseScheme
    '''Поля объекта класса CountryHouse:
    количество жилых комнат
    жилая площадь (в квадратных метрах)
    совмещенный санузел (значениями могут быть или False, или True)
    количество этажей
    площадь участка
    При создании экземпляра класса CountryHouse необходимо
        убедиться, что переданные в конструктор параметры
```

удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом

'Invalid value' '"

Метод __str__()

'"Преобразование к строке вида:
Country House: Количество жилых комнат <количество жилых комнат>, Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный санузел>, Количество этажей <количество этажей>, Площадь участка <площадь участка>.

'"

Метод __eq__()

'"Метод возвращает True, если два объекта класса равны и False иначе.

Два объекта типа CountryHouse равны, если равны жилая площадь, площадь участка, при этом количество этажей не отличается больше, чем на 1.'"

Квартира городская Apartment:

class Apartment: # Класс должен наследоваться от HouseScheme

'" Поля объекта класса Apartment:
количество жилых комнат
площадь (в квадратных метрах)
совмещенный санузел (значениями могут быть или False, или True)
этаж (может быть число от 1 до 15)
куда выходят окна (значением может быть одна из строк: N, S, W, E)
При создании экземпляра класса Apartment необходимо убедиться,
что переданные в конструктор параметры удовлетворяют

требованиям, иначе выбросить исключение ValueError с текстом

'Invalid value'

'''

Метод __str__()

'''Преобразование к строке вида:
Apartment: Количество жилых комнат <количество жилых комнат>, Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный санузел>, Этаж <этаж>, Окна выходят на <куда выходят окна>.

Переопределите список list для работы с домами:

Деревня:

class CountryHouseList: # список деревенских домов -- "деревня", наследуется от класса list

'''Конструктор:

1. Вызвать конструктор базового класса

2. Передать в конструктор строку name и присвоить её полю name созданного объекта'''

Метод append(p_object):

'''Переопределение метода append() списка.

В случае, если p_object - деревенский дом, элемент добавляется в список,

иначе выбрасывается исключение TypeError с текстом:

Invalid type <тип_объекта p_object>'''

Метод total_square():

'''Посчитать общую жилую площадь'''

Жилой комплекс:

class ApartmentList: # список городских квартир -- ЖК, наследуется от класса list

Конструктор:

'''1. Вызвать конструктор базового класса

2. Передать в конструктор строку name и присвоить её полю name созданного объекта

'''

Метод extend(iterable):

'''Переопределение метода extend() списка.

В случае, если элемент iterable - объект класса Apartment, этот элемент добавляется в список, иначе не добавляется.

'''

Метод floor_view(floors, directions):

'''В качестве параметров метод получает диапазон возможных этажей в виде списка (например, [1, 5]) и список направлений из ('N', 'S', 'W', 'E').

Метод должен выводить квартиры, этаж которых входит в переданный диапазон (для [1, 5] это 1, 2, 3, 4, 5) и окна которых выходят в одном из переданных направлений. Формат вывода:

<Направление_1>: <этаж_1>

<Направление_2>: <этаж_2>

...

Направления и этажи могут повторятся. Для реализации используйте функцию filter().

'''

В отчете укажите:

1. Иерархию описанных вами классов.
2. Методы, которые вы переопределили (в том числе методы класса object).
3. В каких случаях будет вызван метод __str__().
4. Будут ли работать непереопределенные методы класса list для CountryHouseList и ApartmentList? Объясните почему и приведите примеры.

Основные теоретические положения.

- map(<функция>, <объект_1> [, <объект_2>, … ,<объект_N-1>]) - функция применяет к элементам итерируемого объекта (объектов) переданную функцию.
- lambda аргумент1, аргумент2,..., аргументN : выражение – это специальный элемент синтаксиса для создания анонимных (т.е. не имеющих имени) функций сразу в том месте, где эту функцию необходимо вызвать.
- filter(<функция>, <объект>) - Функция <функция> применяется для каждого элемента итерируемого объекта <объект> и возвращает объект-итератор, состоящий из тех элементов итерируемого объекта <объект>, для которых <функция> является истиной.
- Синтаксис создания класса:

  class <Название_класса>:

  <Тело_класса>
- Чтобы инициализировать поля объекта при его создании, мы используем конструктор. Синтаксис конструктора:

def __init__(self, <аргумент_1>, …, <аргумент_n>):

<Тело_конструктора>

- Наследование в Python Синтаксис:

  class A: # класс-родитель

  pass

  class B(A): # класс-потомок

  pass

- isinstance(obj_, class_) - функция возвращает True, если obj_ является экземпляром класса class_ или если class_ является суперклассом для класса, объектом которого является obj_.
- issubclass(class1, class2) - функция возвращает True, если class1 является наследником класса class2 (или наследником наследника, с любым уровнем вложенности).

Выполнение работы.

В процессе выполнения поставленного задания было выстроена следующая иерархия классов:

Базовый класс — HouseScheme;

Его потомки —CountryHouse, Apartment;

Наследники класса **list** — CountryHouseList и ApartmentList;

Также были переопределены методы следующих классов :

У класса CountryHouse :

- Метод __init__(self, n_room, area_room, sanuzl,n_level, area_home) — конструктор объектов класса CountryHouse , на основе метода __init__ класса родителя HouseScheme с добавлением новых

полей объекта self.n_level, self.area_home соответственно равных n_level, area_home .

- Метод __str__(self) — возвращает строку вида :
  Country House: Количество жилых комнат <количество жилых комнат>, Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный санузел>, Количество этажей <количество этажей>, Площадь участка <площадь участка>.

- Метод __eq__(self, home1) - Метод возвращает True, если два объекта класса равны и False иначе.Два объекта типа CountryHouse равны, если равны жилая площадь, площадь участка, при этом количество этажей не отличается больше, чем на 1.

У класса Apartment :

- Метод __init__(self, n_room, area_room, sanuzl, level, window) — конструктор объектов класса Apartment , на основе метода __init__ класса родителя HouseScheme с добавлением новых полей объекта self.level, self.window соответственно равных level, window.

- Метод __str__(self) — возвращает строку вида :
  Apartment: Количество жилых комнат <количество жилых комнат>, Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный санузел>, Этаж <этаж>, Окна выходят на <куда выходят окна>.

У класса CountryHouseList :

- Метод __init__(self, name) - конструктор объектов класса CountryHouseList , на основе метода __init__ класса родителя list с добавлением нового поля объекта self.names равного name.

- Метод append(self, p_object) - В случае, если p_object является объектом класса CountryHouse, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом:

  Invalid type <тип_объекта p_object>

- Метод total_square(self) — считает сумврную жилую площадь объектов списка self .

У класса ApartmentList :

- Метод __init__(self, name) - конструктор объектов класса ApartmentList , на основе метода __init__ класса родителя list с добавлением нового поля объекта self.names равного name.

- Метод extend(self, iterable) — в случае, если элемент iterable - объект класса Apartment, этот элемент добавляется в список, иначе не добавляется.

- Метод floor_view(floors, directions) - В качестве параметров метод получает диапазон возможных этажей в виде списка (например, [1, 5]) и список направлений из ('N', 'S', 'W', 'E'). Метод должен выводить квартиры, этаж которых входит в переданный диапазон (для [1, 5] это 1, 2, 3, 4, 5) и окна которых выходят в одном из переданных направлений. Формат вывода:

  <Направление_1>: <этаж_1>

<Направление_2>: <этаж_2>

…

При использовании функция print() к объектам класса Apartment и CountryHouse используется метод __str__ .

Так как CountryHouseList и ApartmentList являются наследниками класса list , то для них сохраняются все не переопределенные методы класса list ,
пример :

```
A = CountryHouse(5,30,True,6,100)
B = CountryHouse(7,300,False,10,320)
arr = CountryHouseList('Hello')
arr.append(A)
arr.append(B)

print(arr.clear())
```

Вывод :

None

Аналогично и для ApartmentList.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии

1.	A = CountryHouse(5,300,True,6,400) B = CountryHouse(7,300,False,7,400) arr = CountryHouseList('Hello') arr.append(A) arr.append(B) print(A) print(B) print(A==B) print(arr.total_square())	Country House: Количество жилых комнат 5, Жилая площадь 300, Совмещенный санузел True, Количество этажей 6, Площадь участка 400. Country House: Количество жилых комнат 7, Жилая площадь 300, Совмещенный санузел False, Количество этажей 7, Площадь участка 400. True 600	Программа работает правильно
2.	A = Apartment(5,300,True,4,"W") B = Apartment(7,300,False,7,"S") arr = ApartmentList('Hello') arr.extend([A,B]) print(A) arr.floor_view([1,6],['W',"N"])	Apartment: Количество жилых комнат 5, Жилая площадь 300, Совмещенный санузел True, Этаж 4, Окна выходят на W. W: 4	Программа работает правильно

Выводы.

Были изучены основы парадигмы программирования , с использованием языка программирования Python .

Была разработана программа, в соответствии с поставленными требованиями , с использованием полученных знаний , таких как :

- Использование функциональное программирование.
- Понимание ООП в Python и работа с классами .

- И работа с исключениями .

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb_3.py

```
class HouseScheme:

    def __init__(self, n_room, area_room, sanuzl):

        if area_room < 0: raise ValueError("Invalid value")
        if not isinstance(sanuzl, bool):  raise ValueError("Invalid value")

        self.n_room = n_room
        self.area_room = area_room
        self.sanuzl = sanuzl


class CountryHouse(HouseScheme):

    def __init__(self, n_room, area_room, sanuzl, n_level, area_home):
        super().__init__(n_room, area_room, sanuzl)

        self.n_level = n_level
        self.area_home = area_home

    def __str__(self):
                return "Country House: Количество жилых комнат " +
str(self.n_room) + ', Жилая площадь ' + str(
            self.area_room) + ", Совмещенный санузел " + str(self.sanuzl) + ",
Количество этажей " + str(
            self.n_level) + ", Площадь участка " + str(self.area_home) + '.'

    def __eq__(self, home1):

            if (self.area_room == home1.area_room) and (self.area_home ==
home1.area_home) and (
                abs(self.n_level - home1.n_level) <= 1):
            return True
        else:
            return False
```

```
class Apartment(HouseScheme):

    def __init__(self, n_room, area_room, sanuzl, level, window):
        super().__init__(n_room, area_room, sanuzl)
        if not isinstance(level, int): raise ValueError("Invalid value")
        if not (level >= 1 and level <= 15): raise ValueError("Invalid value")
          if (not window == 'N') and (not window == 'S') and (not window ==
'W') and (
        not window == 'E'): raise ValueError("Invalid value")

        self.level = level
        self.window = window

    def __str__(self):

        return "Apartment: Количество жилых комнат " + str(self.n_room) + ',
Жилая площадь ' + str(
            self.area_room) + ", Совмещенный санузел " + str(self.sanuzl) + ",
Этаж " + str(
            self.level) + ", Окна выходят на " + str(self.window) + '.'


class CountryHouseList(list):

    def __init__(self, name):
        super().__init__()
        self.name = name

    def append(self, p_object):
        if isinstance(p_object, CountryHouse):
            super().append(p_object)
        else:
            raise TypeError("Invalid type " + str(type(p_object)))

    def total_square(self):
        sum_area_room = 0

        for i in self:
            sum_area_room += i.area_room

        return sum_area_room
```

```
class ApartmentList(list):

    def __init__(self, name):
        super().__init__()
        self.name = name

    def extend(self, iterable):
        # if isinstance(iterable, Apartment):
        #   super().extend(iterable)
        super().extend(filter(lambda x: type(x) == Apartment, iterable))

    def floor_view(self, floors, directions):
        list_room = list(
                filter(lambda room: (room.level >= floors[0]) and (room.level <=
floors[1]) and (room.window in directions),
                    self))

        for i in list_room:
            print(str(i.window) + ': ' + str(i.level))
```