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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Введение в информационные технологии»

Тема: Парадигмы программирования

Студент гр. 9304 _____ Мохаммед А.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы.

Изучить работу с классами, наследование классов, определение и переопределение методов классов на языке Python; научиться бросать исключения.

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().

Основные теоретические положения.

Класс - это тип объекта. Например, можно рассматривать в качестве класса планеты. При этом конкретные примеры планет: Марс, Меркурий и т.д. - будут объектами. Класс описывает общее поведение: общие черты, свойства и характеристики, а также общие действия, функции, которые можно выполнять над объектами класса. Об этом подробнее речь пойдет в следующем пункте. Поля классов - это общие свойства, характеристики классов. Например, что общего можно выделить у всех планет? Планету можно охарактеризовать длиной радиуса, величиной массы, удаленностью от солнца - для каждой отдельно взятой планеты эти характеристики будут принимать конкретные значения. Методы класса — специфические функции, выполняющиеся над объектами класса. Так, метод Планета.ПолучитьРадиус() вернёт радиус конкретной планеты, которую мы полставим вместо имени класса. Классы могут наследовать у других классов. Так, например, можно создать класс ПланетаГигант, который будет наследовать от класса Планета: тогда в нём будут доступны все те методы, что мы определим для класса Планета. Кроме того, мы можем создать новый метод, который будет присущ только этому классу, например, ПланетаГигант.ПолучитьРадиусКолец(), или переопределить метод, доступный в классе-предке, например, выводить радиус планеты не в километрах, а в тысячах километров, поскольку в случае с планетами-гигантами это может быть удобнее.

Исключения — те ошибки, возникшие при выполнении кода, при появлении которых интерпретатор по умолчанию считает, что исполнение кода не может быть продолжено из-за того, что одна из инструкций не может быть выполнена, и завершает программу.

Это поведение можно изменить с помощью обработчика исключений.

Кратко о блоках обработки исключительных ситуаций:

- В блок **try** помещают код, который может вызвать исключительную ситуацию (потенциально опасный код).
- В блок **except** помещают код для обработки исключительной ситуации. Для одного **try**-блока может быть несколько **except**-блоков.
- В блок **finally** помещают код, который должен выполниться в любом случае, вне зависимости от того, произошла исключительная ситуация или нет. Блок **finally** может быть только один.
- В блок **else** помещают код, который должен выполниться, если в **try**-блоке не случилось исключительной ситуации (**else**-блок выполняется в случае, если утверждение "Исключительная ситуация произошла" ложно). Блок **else** может быть только один.

Собственные исключения можно вызывать с помощью инструкции raise.

Выполнение работы.

Разработанный алгоритм решает поставленную задачу в соответствии с приведённой выше инструкцией.

Тестирование.

Результаты тестирования представлены в табл. 1. и комментариях под ней.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	>>> sch = HouseScheme(1, 20, False) >>> sch	<proj.HouseScheme object at 0xb6fabdec>	Метод__repr__() класса HouseScheme не был переопределен. Получен ожидаемый результат.
2.	>>> sch = HouseScheme(1, 0, False)	Traceback (most recent call last):	Поднято исключение: ожидалось

		File "<stdin>", line 1, in <module> File "proj.py", line 9, in __init__ raise ValueError("Invalid value") ValueError: Invalid value	положительное значение площади. Получен ожидаемый результат.
3.	>>> village = CountryHouseList("Медведково") >>> len(village)	0	Метод_len_() класса list не был переопределен, он работает так же, как и у класса list

4	>>> fazenda = CountryHouse(3, 40, False, 2, 500)
	>>> village.append(fazenda)
	>>> len(village)
		1	Получен ожидаемый результат

5	>>> faz = CountryHouse(5, 100, False, 2, -20)	Traceback (most recent call last): File "<stdin>", line 1, in <module> File "proj.py", line 19, in __init__ raise ValueError("Invalid value") ValueError: Invalid value	Ожидалось положительное значение площади участка. Брошено исключение. Получен ожидаемый результат.
6	>>> faz = CountryHouse(4, 100, False, 2, 500) >>> faz == fazenda	False	Получен ожидаемый результат.
7	>>> faz = CountryHouse(4, 40, False, 1, 500) >>> faz == fazenda	True	Получен ожидаемый результат.
8	>>> print(fazenda)	Country House: Количество жилых комнат	Получен ожидаемый результат.

		3, Жилая площадь 40, Совмещенный санузел False, Количество этажей 2, Площадь участка 500.	

Иерархия созданных классов:

Класс-потомок	Класс-родитель
CountryHouseList	list
ApartmentList	list
CountryHouse	HouseScheme
Apartment	HouseScheme
HouseScheme	object
list	object

Переопределённые методы:

Метод	Класс-потомок	Класс-родитель
__init__	Apartment	HouseScheme
__init__	CountryHouse	HouseScheme
__init__	CountryHouseList	list
__init__	ApartmentList	list
__str__	CountryHouse	object
__str__	Apartment	object
__eq__	CountryHouse	object
append	CountryHouseList	list
extend	ApartmentList	list

Метод_str классов CountryHouse и Apartment будет вызван при вызове функции print как в интерпретаторе, так и в модуле, а также при непосредственном вызове метода_str_, функции str и функции format. При обращении к объекту по имени в интерпретаторе метод вызываться не будет.

Непереопределённые методы класса list будут работать для его наледников: классов CountryHouseList и ApartmentList. Пример из таблицы 1:

```
>>> village = CountryHouseList("Медведково")
>>> len(village)
0
```

Выводы.

Были изучены классы, наследование классов, методы классов, исключения в языке Python, создана система классов для градостроительной компании; конструкторы этих классов могут бросать исключения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class HouseScheme:
    def__init__(self, NumberOfRooms, Square, SharedBathroom):
        if (NumberOfRooms >= 0) and (Square > 0) and
(type(SharedBathroom) == bool):
           self.NumberOfRooms = NumberOfRooms;
           self.Square = Square;
           self.SharedBathroom = SharedBathroom;
        else:
           raise ValueError("Invalid value")


class CountryHouse(HouseScheme): # Класс должен наследоваться от
HouseScheme
    def__init__(self, NumberOfRooms, Square, SharedBathroom,
NumberOfFloors, PlotSquare):
        super().__init__(NumberOfRooms, Square, SharedBathroom);
        if (NumberOfFloors > 0) and (PlotSquare > 0):
           self.NumberOfFloors = NumberOfFloors;
           self.PlotSquare = PlotSquare;
        else:
           raise ValueError("Invalid value")

    def__str__(self):
        return "Country House: Количество жилых комнат " +
str(self.NumberOfRooms) + ", Жилая площадь " +str(self.Square) + ",
Совмещенный санузел "+str(self.SharedBathroom) + ", Количество этажей
" +str(self.NumberOfFloors) + ", Площадь участка "
+str(self.PlotSquare) + "."
def__eq__(self, other):
        if (self.Square == other.Square) and (self.PlotSquare ==
other.PlotSquare) and (abs(self.NumberOfFloors - other.NumberOfFloors)
<= 1):
            return True;
        else:
            return False;


class Apartment(HouseScheme): # Класс должен наследоваться от
HouseScheme
    def__init__(self, NumberOfRooms, Square, SharedBathroom, Floor,
View):
        super().__init__(NumberOfRooms, Square, SharedBathroom);
        if (1 <= Floor <= 15) and (View in ['N', 'S', 'W', 'E']):
           self.Floor = Floor;
           self.View = View
        else:
           raise ValueError("Invalid value");
    def__str__(self):
        return "Apartment: Количество жилых комнат "
+str(self.NumberOfRooms) + ", Жилая площадь " +str(self.Square) + ",
```

```
Совмещенный санузел " +str(self.SharedBathroom) + ", Этаж "
+str(self.Floor) + ", Окна выходят на " +str(self.View) + "."

class CountryHouseList(list):
    def__init__(self, name):
       super().__init__();
       self.name = name;

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

    def total_square(self):
       result = 0;
       for element in self:
           result += element.Square;
       return result;

class ApartmentList(list):

      def__init__(self,  name):
            super()._init__();
            self.name = name;

      def extend(self, iterable):
          for element in iterable:
              if type(element) == Apartment:
                  super().append(element)

      def floor_view(self, floors, directions):

          answer = list(filter(lambda apartment: apartment.Floor in
range(floors[0], floors[1] + 1) and apartment.View in directions,
self))
          for element in answer:
              print(element.View, ": ", element.Floor, sep='')
```