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.

Студентка гр. 0382 _____ Тихонов С.В.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2020

Цель работы.

Рассмотреть понятия парадигм программирования и освоить объектно-ориентированное программирование в Python на практике.

Задание.

Система классов для градостроительной компании

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 '''

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.

Термин “парадигма программирования” — это подход к программированию, описанный совокупностью идей и понятий, определяющих стиль написания компьютерных программ.

Итератор – это своего рода перечислитель для определенного объекта (например, списка, строки, словаря), который позволяет перейти к следующему элементу этого объекта, либо бросает исключение, если элементов больше нет. Итерируемый объект – объект, по которому можно итерироваться (тоесть который можно обходить в цикле, например, цикле for).

Функция filter(). Синтаксис функции: 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.

lambda-выражения.

При помощи лямбда-выражения можно объявлять функции в любом месте кода, в том числе внутри других функций. Синтаксис определения следующий: lambda аргумент1, аргумент2,..., аргумент N : выражение

ООП в Python

Классы содержат атрибуты, которые подразделяются на поля и методы. Под методом понимают функцию, которая определена внутри класса.

Конструктор - это специальный метод, который нужен для создания объектов класса.

Объектно-ориентированная парадигма базируется на нескольких принципах: наследование, инкапсуляция, полиморфизм. Наследование - специальный механизм, при котором мы можем расширять классы, усложняя

их функциональность. В наследовании могут участвовать минимум два класса: суперкласс (или класс-родитель, или базовый класс) - это такой класс, который был расширен. Все расширения, дополнения и усложнения класса-родителя реализованы в классе-наследнике (или производном классе, или классе-потомке) - это второй участник механизма наследования.

Выполнение работы.

Ход решения:

Класс HouseScheme(). Не имеет родителя, имеет два класса-потомка Apartment и CountryHouse. Поля объекта класса living_rooms (количество жилых комнат), territory (жилая площадь (в квадратных метрах)), bathroom_unit (совмещенный санузел (значениями могут быть или False, или True)) инициализируются в переопределяемом методе-конструктуре __init__(). Осуществляется проверка, что переданные в конструктор параметры удовлетворяют требованиям, в противном случае с помощью raise создаётся и выбрасывается исключение ValueError с текстом 'Invalid value' .

Класс CountryHouse(HouseScheme). Потомок класса HouseScheme, не является родителем. В конструкторе __init__() наследует поля объекта класса HouseScheme - living_rooms , territory, bathroom_unit, и инициализируются другие поля – floors (количество этажей), place (площадь участка). Осуществляется проверка, что переданные в конструктор параметры удовлетворяют требованиям и полям присваиваются значение переданных в конструктов аргументов.

Далее переопределяется метод __str__(self). Он возвращает строку заданного формата. И переопределяется метод __eq__(self, object2), который возвращает True, если два объекта класса, переданные в метод равны и False иначе.

Класс Apartment(HouseScheme). Потомок класса HouseScheme, не является родителем. В конструкторе __init__() наследует поля объекта класса HouseScheme - living_rooms , territory, bathroom_unit, и инициализируются другие поля – floor (этаж (может быть число от 1 до 15)), window_view (куда выходят окна (значением может быть одна из строк: N, S, W, E)). Осуществляется проверка, что переданные в конструктор параметры удовлетворяют требованиям, в противном случае с

помощью raise создаётся и выбрасывается исключение ValueError с текстом 'Invalid value' .

Далее переопределяется метод __str__(self). Он возвращает строку заданного формата.

Класс CountryHouseList(list). Потомок класса list, не является родителем. В конструкторе __init__() инициализируется поля объекта класса – name (полю класса присваивается аргумент-строки name). Далее переопределяется метод append(self, p_object). В нём осуществляется проверка, если переданный в метод аргумент p_object удовлетворяет заданным условиям (если p_object - деревенский дом), элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: ‘Invalid type <тип_объекта p_object>’. И переопределяется метод total_square(self), в котором считается и возвращается общая жилая площадь текущего объекта класса.

Класс ApartmentList(list). Потомок класса list, не является родителем. В конструкторе __init__() инициализируется поля объекта класса – name (полю класса присваивается аргумент-строки name). Далее переопределяется метод списка - extend(self, iterable). В нём осуществляется проверка, если элемент iterable - объект класса Apartment, этот элемент добавляется в список, иначе не добавляется. И переопределяется метод floor_view(self, 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 выводит квартиры, удовлетворяющие заданным условиям (этаж которых входит в переданный диапазон (для [1, 5] это 1, 2, 3, 4, 5) и окна которых выходят в одном из переданных направлений), преобразуя их в строку заданного формата.

1. Иерархия описанных классов.

CountryHouse(потомок) – HouseScheme(родитель)

Apartment(потомок) – HouseScheme(родитель)

CountryHouseList(потомок) - list(родитель)

ApartmentList(потомок) - list(родитель)

2. Методы, которые были переопределены:

def __init__(self,);

def __str__(self);

def __eq__(self, object2);

def append(self, p_object);

def extend(self, iterable).

3. Метод __*str*__() будет вызван:

При вызове функции *str*() - приведении к типу “строка” в явном виде, или неявном, как, например, при вызове функции print().

4.**Будут ли работать непереопределенные методы класса** list для CountryHouseList и ApartmentList? Объясните почему и приведите примеры.

Будут, но если не переопределять - будут работать в их базовом формате, как обычные функции класса list, ведь он является родителем классов CountryHouseList и ApartmentList.

Пример: метод ***list.clear***(), если его не переопределить, будет очищать нынешний список, являющийся объектом класса CountryHouseList или ApartmentList.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
Экземпляры классов и другие переменные для проверки		country1 = CountryHouse(6, 80, False, 1, 100) country2 = CountryHouse(5, 70, True, 2, 200) apartment = Apartment(4, 20, True, 6, 'S')	
1.	print(country1)	Country House: Количество жилых комнат 6, Жилая площадь 80, Совмещенный санузел False, Количество этажей 1, Площадь участка 100.	Программа выводит верный ответ.
2.	print(country1 == country2)	False	Программа выводит верный ответ.
3.	print(apartment)	Apartment: Количество жилых комнат 4, Жилая площадь 20, Совмещенный санузел True, Этаж 6, Окна выходят на S.	Программа выводит верный ответ.

Вывод.

Были рассмотрены понятия парадигм программирования и освоено объектно-ориентированное программирование в Python на практике.

Разработан фрагмент программы, описывающий некоторые классы и их методы. Были использованы исключения с конструкцией raise, lambda-выражения, функция filter.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3.py

```
class CountryHouse(HouseScheme):
    def __init__(self, rooms, livingSpace, combinedBathroom, floors, area):
        if not (isinstance(floors, int) and isinstance(area, int)):
            raise ValueError('Invalid value')
        else:
            super().__init__(rooms, livingSpace, combinedBathroom)
            self.floors = floors
            self.area = area

    def __str__(self):
        return 'Country House: Количество жилых комнат {}, Жилая площадь {},
Совмещенный санузел {}, Количество этажей {}, Площадь участка {}.'.format(
            self.rooms, self.space, self.comBath, self.floors, self.area)

    def __eq__(self, other):
        if isinstance(other, CountryHouse):
            if self.space == other.space and self.area == other.area and abs(
                self.floors - other.floors) <= 1: return True
        return False


class Apartment(HouseScheme):
    def __init__(self, rooms, livingSpace, combinedBathroom, floor, side):
        if not (isinstance(floor, int) and (1 <= floor <= 15) and (side in
['N', 'S', 'W', 'E'])):
            raise ValueError('Invalid value')
        else:
            super().__init__(rooms, livingSpace, combinedBathroom)
            self.floor = floor
            self.side = side

    def __str__(self):
        return 'Apartment: Количество жилых комнат {}, Жилая площадь {},
Совмещенный санузел {}, Этаж {}, Окна выходят на {}.'.format(
            self.rooms, self.space, self.comBath, self.floor, self.side)


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

    def append(self, obj):
```

```
        if isinstance(obj, CountryHouse):
            super().append(obj)
        else:
            raise TypeError(f"Invalid type {type(obj)}")

    def total_square(self):
        return sum(house.space for house in self)


class ApartmentList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def extend(self, iterable):
        for apt in iterable:
            if isinstance(apt, Apartment): super().append(apt)

    def floor_view(self, floors, directions):
        apts = list(filter(lambda apt: apt.floor in range(floors[0],
floors[1] + 1) and apt.side in directions, self))
        for apt in apts: print(f"{apt.side}: {apt.floor}")
```