Лабораторная работа №3

Объектно-ориентированное программирование Python

Цель: настоящей работы состоит в том, чтобы изучить объектно-ориентированное программирование с использованием Python

.

Задачи:

Выполнение практической работы предполагает решение ***следующих задач***:

1. Создание и использование простого класса
2. Понятие Объект. Атрибут объекта класса. Атрибут экземпляра класса

Последовательность выполнения работы :

Использован материал И.В. Мироновой, Программирование на языке Python. Часть 2: Учебное пособие - М.: Финансовый университет, департамент анализа данных, принятия решений и финансовых технологий, 2019. -51 с.

Объектно-ориентированное программирование

Создание и использование простого класса

Класс описывает модель объекта, его свойства и поведение. С точки

зрения программиста, класс — это тип данных, который создается для описания сложных объектов. Класс содержит набор переменных и функций. Переменные называются **атрибутами**. В них хранятся характеристики объекта (название, размер, цвет, количество и так далее). Функции называются **методами**. Метод определяет действие, которое объект может выполнять над самим собой или другими объектами

Класс создается с помощью инструкции вида:

class *ИмяКласса*:

    ['''*строка документирования*''']

    Переменная = ЗНАЧЕНИЕ

    …

    def *ИмяМетода*(self, ...):

        self.*Переменная* = ЗНАЧЕНИЕ

        …

    …

В Python принято, чтобы имена классов начинались с большой буквы. Если имя должно состоять из нескольких слов, то между словами не должно быть символов подчеркивания, а каждое слово внутри имени должно начинаться с большой буквы. Например, UniversityStudent.

Объект – экземпляр класса. Объект хранит конкретные значения атрибутов и информацию о принадлежности к классу, может выполнять методы. Например, int – это класс, а число 5 – это объект (экземпляр класса int).

Создать объект (экземпляр класса) можно так:

ИмяОбъекта = ИмяКласса([*Параметры_конструктора*])

Метод экземпляра класса – это функция, объявленная внутри класса. Методы экземпляра имеют обязательный первый параметр, который содержит ссылку на объект, для которого вызван метод. В Python принято

называть этот параметр self. Не следует заменять это имя другим. Вызывая метод, вы не должны передавать значение для self явно – интерпретатор сделает это автоматически. Через эту ссылку выполняется доступ к атрибутам экземпляра класса.

Существуют методы со специальными именами, предназначенныедля выполнения определенных действий. Одним из таких методов является метод_init_() (в начале и конце имени здесь стоят по два символа подчеркивания), который по аналогии с другими языками программирования называют конструктором. Конструктор автоматически вызывается при создании экземпляра класса. Обычно используется для задания начальных значений атрибутов. Этот метод, как любая функция, может иметь параметры (помимо self), которые указываются при создании объекта.

Метод str()_представляет объект в виде строки. Метод вызывается при выводе с помощью функции print(), а также при использовании функцииstr().

Внутри класса можно создать метод, который будет доступен без создания экземпляра класса. Для этого используется конструкция:

@staticmethod

def *ИмяМетода*(…)

…

Такой метод называется статическим. В списке параметров статического метода отсутствует self. Обычно такие методы используют атрибуты класса. Для вызова метода используется конструкция *ИмяКласса.ИмяМетода*(…). Можно вызвать метод и через экземпляр класса *ИмяОбъекта.ИмяМетода*(…).

Атрибут – это переменная, созданная внутри класса. Чтобы создать атрибут нужно выполнить оператор присваивания. Важно понимать, что в классах есть 2 вида атрибутов: атрибуты объекта класса и атрибуты

экземпляра класса.

Атрибут объекта класса (атрибут класса) создается при присваивании значения переменной внутри класса (но вне любого метода). Можно создавать атрибуты класса динамически (не в классе), используя инструкцию

ИмяКласса.ИмяАтрибута = ЗНАЧЕНИЕ

Атрибуты класса можно использовать даже тогда, когда ни одного экземпляра данного класса не существует. Атрибут класса доступен всем экземплярам класса, и для них всех это одна и та же переменная. Если атрибут класса изменить, то значение изменится для всех экземпляров класса. Для получения значения атрибута можно использовать конструкцию *ИмяОбъекта.ИмяАтрибута*, но для изменения значения только конструкцию *ИмяКласса.ИмяАтрибута*.

Атрибут экземпляра класса (атрибут) используется для хранения значений, которые для каждого экземпляра класса уникальны.

Создается атрибут внутри метода класса при выполнении инструкции присваивания:

self.ИмяАтрибута = ЗНАЧЕНИЕ

Для доступа к атрибуту внутри класса нужно использовать конструкцию *self.ИмяАтрибута*, а вне класса *ИмяОбъекта.ИмяАтрибута*. Атрибут можно создать динамически после создания объекта, выполнив инструкцию

ИмяОбъекта.ИмяАтрибута = ЗНАЧЕНИЕ

В качестве примера рассмотрим класс Счет, моделирующий банковский счет:

```
#класс Счет
class Счет():
    """Банковский счет"""

    count = 0 #создаем атрибут класса

    def __init__(self, number):
        self.number = number #создаем атрибуты
        self.balance = 0     #экземпляра
        Счет.count += 1      #изменяем атрибут класса

    def __str__(self):
        r = "Счет(number = " + str(self.number)
                             #используем атрибут экземпляра
        r = r + ", balance = " + str(self.balance) + ")"
        return r

    def change(self, summa):
        self.balance += summa #изменяем атрибут экземпляра

    @staticmethod             #создаем статический метод
    def info():               #используем атрибут класса
        print("Всего счетов: ", Счет.count)
#использование класса
print(Счет.count)   #используем атрибут класса

A = Счет(1234)      #создаем объект,
                    #вызываем метод __init__
A.change(2000)      #вызываем метод
print(A.balance)    #используем атрибут экземпляра

B = Счет(2453)      #создаем еще один объект
B.change(3000)


print(A)            #проверяем метод __str__()
print(B)
print(A.count)      #используем атрибут класса
Счет.info()         #вызываем статический метод
```

Свойства

Все элементы класса в языке Python являются открытыми, т.е.

доступными вне класса. Однако между программистами существует соглашение о том, что переменная или метод, у которых имя начинается с одиночного подчеркивания, не предназначены для использования вне класса (защищенные элементы класса). Если же имя начинается с двух символов подчеркивания (закрытые члены класса), то оно автоматически заменяется в системе на имя *_ИмяКласса_ИмяЭлемента*. Этот механизм называется искажением имен. Он нужен для избегания конфликтов имен при наследовании.

При разработке собственных классов рекомендуется создавать методы для работы с атрибутами. Это гораздо безопаснее. Например, после разработки класса вам может понадобиться добавить какую-то проверку или преобразование значения атрибута. Если вы использовали методы, то необходимые действия добавляются в метод, и не нужно вносить какие-либо изменения в остальной код. Если же вы использовали переменную, то изменения нужно делать во всех местах, где была использована эта переменная. Технология сокрытия информации о внутреннем устройстве объекта за внешним интерфейсом из методов называется **инкапсуляцией**.

Работа с методом не так удобна, как работа с переменной. Вызов метода обычно длиннее, чем просто обращение к переменной, поэтому в классах появилась возможность создавать специальные идентификаторы, через которые можно выполнять операции над атрибутами. Они называются свойствами.

Свойства при использовании выглядят как атрибут, но в отличие от атрибута для них можно задать необходимые действия в момент выполнения операций получения значения, присваивания значения и удаления значения. Создается свойство с помощью функции property():

Свойство = property(*МетодДляЧтения* [,

МетодДляПрисваивания [,

МетодДляУдаления [,

Строка_документирования]]])

Обязательным в данном случае является только первый параметр. Первые три параметра являются ссылками на методы класса.

Ниже приведен пример класса, в котором создано свойство v, и показано, как его можно использовать.

```
class Class1:
    def __init__(self, value):
        self.__var = value

    def __getVar(self):         # Чтение
        return self.__var

    def __setVar(self, value): # Запись
        self.__var = value

    def __delVar(self):         # Удаление
        del self.__var

    v = property(__getVar, __setVar, __delVar,
                 "Строка документирования")
c1 = Class1(10)
c1.v = 101        # Вызывается метод __setVar()
print (c1.v)      # Вызывается метод __getVar()
del c1.v          # Вызывается метод __delVar()
```

Методы setVar(),_getVar(),_delVar реализованы в этом примере самым простым способом. На практике в них можно выполнять необходимые проверки и преобразования значений.

Добавим свойство balance в класс Счет и сделаем атрибуты закрытыми:

```
#класс Счет
class Счет():
    """Банковский счет"""

    __count = 0 #создаем атрибут класса

    def __init__(self, number):
        self.__number = number #создаем атрибуты
        self.__balance = 0     #экземпляра
        Счет.__count += 1      #изменяем атрибут класса

    def __str__(self):
        r = "Счет(number = " + str(self.__number)
                               #используем атрибут экземпляра
        r = r + ", balance = " + str(self.balance) + ")"
                               #используем свойство
        return r
```

```
    def change(self, summa):
        self.__balance += summa #изменяем атрибут экземпляра

    @staticmethod               #создаем статический метод
    def info():                 #используем атрибут класса
        print("Всего счетов: ", Счет.count)

    def __get_balance(self):
        x = input('Введите ваш код доступа: ')
        if  x == '123':
            return self.__balance
        else:
            print('Неверный код доступа')
            return None

    balance = property(__get_balance) #создаем свойство
                                      #только для чтения


#использование класса
A = Счет(1234);
print(A.balance)
A.change(2000)
print(A.balance)
A.balance = 2000    #возникнет ошибка
                    #AttributeError: can't set attribute
```

Требования к результатам выполнения работы

Условия успешной сдачи лабораторной работы:

1. Подготовлено пошаговое описание решений представленных задач.
2. Разработана программная реализация представленных задач.
3. Подготовлено краткое описание разработанного программного кода.
4. Программная реализация представленных примеров выложены в личный репозиторий на GitHub.

Литература

Копырин, А. С. Программирование на Python : учебное пособие / А. С. Копырин, Т. Л. Салова. — Сочи : СГУ, 2018. — 48 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/147665 (дата обращения: 15.07.2021). — Режим доступа: для авториз. пользователей.