Программирование на Python.

Тема №3. Методы. Статические методы. Методы класса.

Методы

До сих пор мы определяли методы как функции внутри классов, например:

```
class Example:
    mode = []
    color = 'green'
    def __init__(self,value, name):...

    def __del__(self):
        print("Удаление экземпляра: " + str(self))

    def get_value(self):
        return self.__value
```

У каждого из этих методов есть параметр **self**, являющийся ссылкой на экземпляр класса, из которого метод вызывается

Вызов метода

При вызове метода напрямую из класса – нужно явно указать значение **self (**экземпляр класса). При вызове от имени экземпляра, нам не нужно явно указывать значение аргумента **self**:

```
a = Example( value: 100, name: 'Новый')
print(Example.get_value(a)) #100
print(a.get_value())        #100
```

Особенности создания методов

Кроме такого подхода, есть специальные инструменты для создания методов:

- использование декоратора **@classmethod** позволяет создавать методы, которые привязаны к классу, а не к экземплярам
- использование декоратора **@staticmethod** позволяет создавать методы, которые могут вызываться без экземпляров класса

staticmethod

Статические методы не требуют создания экземпляров класса для вызова.

```
class Example:
    mode = []
    color = 'green'

    def __init__(self, value, name):...

    @staticmethod
    def useful_function():
        return 'результат работы функции'

print(Example.useful_function())
```

Удобно создавать такие методы, если мы создали какую-то функцию, которая может быть применена не только к экземплярам класса **Example**

classmethod

Методы класса привязываются непосредственно к классу, а не к экземплярам. Такие методы могут изменять состояние класса, но не могут менять состояние конкретного экземпляра.

Такие методы принимают ссылку на класс – ключевое слово **cls.**

classmethod

Пример:

```
class Example:
    mode = []
    color = 'green'
    def __init__(self, value, name):...

    @classmethod
    def print_class_attributes(cls):
        print('Атрибуты класса:',cls.mode, cls.color)

    @staticmethod
    def useful_function():
        return 'результат работы функции'

Example.print_class_attributes()
```

Проще говоря

Staticmethod – можно рассматривать как обычные функции, локализованные в пространстве имен класса. Они не используют данных класса (статичны)

Classmethod – методы, имеющие доступ к атрибутам класса, через который они вызваны, но не имеющие доступа к экземплярам.

Пример

```
class Cylinder:
    __instance_count = 0
    @staticmethod
    def calculate_surface_area(d,h):
        circle =  3.14 * d**2 / 4
        side = 3.14 * d * h
        return 2 * circle + side
    def __init__(self,diameter, height):
        Cylinder.__instance_count +=1
        self.d = diameter
        self.h = height
        self.area = Cylinder.calculate_surface_area(self.d, self.h)
    @classmethod
    def total_objects(cls):
        print('Сущностей создано: ', cls.__instance_count)
```

Пример

В примере выше создан класс Cylinder, внутри которого есть **статический метод – calculate_surface_area,** который можно вызывать без создания экземпляров для вычисления площади цилиндра. А также **метод класса – total_objects**, отображающий количество созданных экземпляров класса

Использование на практике

Оба этих метода часто используются при реализации механизма наследования (о чем мы поговорим в следующей теме):

- **Classmethod** используется в суперклассе для определения того, как метод должен вести себя, когда он вызывается дочерними классами.
- **Staticmethod** используется, когда нужно вернуть одно и то же значение, независимо от вызываемого дочернего класса.

__str__

Магический метод для отображения информации об объекте класса для пользователей (например, для функций print, str)

```
class Cat:
    def __init__(self, name):
        self.name = name
cat = Cat('Васька')
print(cat)
```

```
<__main__.Cat object at 0x7efef5add650>
```

```
class Cat:
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return f"{self.name}"
cat = Cat('Васька')
print(cat)
```

```
Васька
```

Упражнение

Реализуйте класс Person.

- Инициализатор класса должен принимать два аргумента **name (**имя**), age (**возраст в годах**)**
- Определите внутри класса статический метод **is_adult**(), получающий единственный аргумент – возраст человека, и возвращающий True/False в зависимости от того, достиг ли человек 18-ти летнего возраста.
- Реализуйте метод класса **from_birth_year()**, принимающий два аргумента: имя и год рождения, и создающий экземпляр Person, вычисляя возраст по формуле: текущий_год – указанный год.

Перегрузка операторов

При работе с классами, Python позволяет определять для классов встроенные операторы, такие как операции сложения, вычитания, сравнения и т.д. Для этого, необходимо внутри класса переопределить метод с соответствующим именем.

Перегрузка операторов

Операция	Синтаксис	Функция
Сложение	a + b	__add__(a, b)
Объединение	seq1 + seq2	__concat__(seq1, seq2)
Проверка наличия	obj in seq	__contains__(seq, obj)
Деление	a / b	__truediv__(a, b)
Деление	a // b	__floordiv__(a, b)
Поразрядное И	a & b	__and__(a, b)
Поразрядное XOR	a ^ b	__xor__(a, b)
Поразрядная инверсия	~ a	__invert__(a)
Поразрядное ИЛИ	a | b	__or__(a, b)
Степень	a ** b	__pow__(a, b)
Присвоение по индексу	obj[k] = v	__setitem__(obj, k, v)

Перегрузка операторов

Операция	Синтаксис	Функция
Удаление по индексу	del obj[k]	__delitem__(obj, k)
Обращение по индексу	obj[k]	__getitem__(obj, k)
Сдвиг влево	a << b	__lshift__(a, b)
Остаток от деления	a % b	__mod__(a, b)
Умножение	a * b	__mul__(a, b)
Умножение матриц	a @ b	__matmul__(a, b)
Арифметическое отрицание	-a	__neg__(a)
Логическое отрицание	not a	__not__(a)
Положительное значение	+a	__pos__(a)
Сдвиг вправо	a >> b	__rshift__(a, b)
Установка диапазона	seq[i:j] = values	__setitem__(seq, slice(i, j), values)

Перегрузка операторов

Операция	Синтаксис	Функция
Удаление диапазона	del seq[i:j]	__delitem__(seq, slice(i, j))
Получение диапазона	seq[i:j]	__getitem__(seq, slice(i, j))
Вычитание	a - b	__sub__(a, b)
Проверка на True/False	obj	__bool__(obj)
Меньше чем	a < b	__lt__(a, b)
Меньше чем или равно	a <= b	__le__(a, b)
Равенство	a==b	__eq__(a, b)
Неравенство	a != b	__ne__(a, b)
Больше чем или равно	a >= b	__ge__(a, b)
Больше чем	a > b	__gt__(a, b)
Сложение с присваиванием	a += b	__iadd__(a, b)

Перегрузка операторов

Операция	Синтаксис	Функция
Объединение с присваиванием	a += b	__iconcat__(a, b)
Поразрядное умножение с присваиванием	a &= b	__iand__(a, b)
Деление с присваиванием	a //= b	__ifloordiv__(a, b)
Сдвиг влево с присваиванием	a <<= b	__ilshift__(a, b)
Сдвиг вправо с присваиванием	a >>= b	__irshift__(a, b)
Деление по модулю с присваиванием	a %= b	__imod__(a, b)
Умножение с присваиванием	a += b	__imul__(a, b)

Перегрузка операторов

Операция	Синтаксис	Функция
Умножение матриц с присваиванием	a @= b	__imatmul__(a, b)
Поразрядное сложение с присваиванием	a |= b	__ior__(a, b)
Возведение в степень с присваиванием	a **= b	__ipow__(a, b)
Вычитание с присваиванием	a -= b	__isub__(a, b)
Деление с присваиванием	a /= b	__itruediv__(a, b)
Операция XOR с присваиванием	a ^= b	__ixor__(a, b)

Пример перегрузки операторов

Пример перегрузки оператора сложения:

```
class Rectangle:
    def __init__(self,area):
        self.area = area
    #переопределение оператора сложения
    def __add__(self, other):
        return Rectangle(self.area + other.area)

a = Rectangle(10)
b = Rectangle(15)
c = a + b
print(c.area)       #25
```

@property

В Python существует инструмент, который позволит превратить метод класса, возвращающий значение в объект-свойство **property.**

Для этого достаточно записать декоратор ***@property*** перед методом-геттером:

```
class Rectangle:
    def __init__(self,h,w):
        self.h = h
        self.w = w
    @property
    def get_area(self):
        return self.h * self.w
r = Rectangle( h: 5, w: 10)
#5*10 = 50 - без вызова метода:
print(r.get_area)
```

@property

Кроме того, можно использовать декоратор следующим образом:

```
class Item:
    def __init__(self,price):
        self.__price = price

    @property
    def price(self):
        return self.__price
    @price.setter
    def price(self,new_price):
        if type(new_price) in (float,int) and new_price > 0:
            self.__price = new_price
        else:
            print('Введите правильное значение')
    @price.deleter
    def price(self):
        print('Атрибут price удален')
        del self.__price
```

Смысл @property

Используя декоратор @property, вы можете использовать имя данного свойства, чтобы избежать засорения пространства имен функциями геттерами, сеттерами и делиттерами:

```
new = Item(500)
print(new.price)    #500
new.price = 1000
print(new.price)    #1000
del new.price       #Атрибут price удален
print(Item.__dict__)
```

```
{'__module__': '__main__', '__init__': <function Item.__init__ at 0x000001C2E96ADC60>,
 'price': <property object at 0x000001C2E969D990>, '__dict__': <attribute '__dict__' of
 'Item' objects>, '__weakref__': <attribute '__weakref__' of 'Item' objects>, '__doc__':
 None}
```

Упражнение

В одной из задач, решаемых ранее, вы создавали класс **Fraction**, представляющий собой рациональную дробь. Экземпляры этого класса имеют числитель **num,** знаменатель **denum**, метод **simplify**() для отображения дроби в виде вещественного числа и метод **show()** для отображения дроби в виде “числитель/знаменатель”.

Доработайте данный класс, выполнив перегрузку операторов сложения, умножения и деления дробей. Перегруженные методы должны возвращать результат в виде экземпляра **Fraction**

Упражнение

Создайте класс **Vector**, представляющий вектор с началом в точке 0,0 и концом в точке с координатами X, Y. При инициализации экземпляра указываются координаты X, Y.

Определите в классе Vector **операторы сложения, вычитания, скалярного умножения векторов.**

Создайте **classmethod** для вычисления длины вектора.

Упражнение

Создайте класс **Employee**, представляющий работника некоторого учреждения. Данные о работнике:

- ФИО – список из трех строк (фамилия, имя, отчество)
- Возраст – целое число в диапазоне от 18 до 120
- Серия и номер паспорта в формате **** ******, где * - это цифра в диапазоне от 0 до 9
- Размер одежды – значение из списка (S,M,L,XL,XXL)

Класс должен иметь: инициализатор, вызывающий вспомогательные методы для проверки всех вводимых данных. Все методы для проверки данных должны быть описаны в виде **классметодов.** Эти методы должны вызываться в инициализаторе до присвоения значений. Все требуемые данные о работнике храните в **объекте-property** с соответствующими геттерами и сеттерами.