STEP IT ACADEMY

Занятие №17

Объектно-ориентированное программирование

Специальные методы

Специальные методы

Специальные методы имеют для интерпретатора особое значение.

Имена специальных методов и их смысл определены создателями языка: создавать новые нельзя, можно только реализовывать существующие. Названия всех специальных методов начинаются и заканчиваются на два подчѐркивания.

Пример такого метода — уже знакомый нам__init__.
Он предназначен для инициализации экземпляров и автоматически вызывается интерпретатором после создания экземпляра объекта.

Специальные методы

Остальные специальные методы также вызываются в строго определѐнных ситуациях.

Так, например, всякий раз, когда интерпретатор встречает запись вида

x + y,

он заменяет еѐ на

x.add(y),

и для реализации сложения нам достаточно определить в классе экземпляра x метод add .

Пример

```
class Time:
    def __init__(self, minutes, seconds):
        self.minutes = minutes
        self.seconds = seconds

    def __add_(self, other):
        m = self.minutes + other.minutes s =
        self.seconds + other.seconds
        m += s // 60  s = s
        % 60
        return Time(m, s)

    def info(self):
        return '{}:{}'.format(self.minutes, self.seconds)
```

создаем новый экземпляр

Пример

```
t1 = Time(5, 50)
print(t1.info())     # 5:50

t2 = Time(3, 20)
print(t2.info())     # 3:20

t3 = t1 + t2
print(t3.info())     # 9:10
```

Все магические методы вступают в силу таким образом и требуют определенного имени функции и сигнатуры метода (иногда сигнатура метода является переменной), и тогда метод будет вызываться при определенных обстоятельствах.

__eq__

```
>>> class MyClass:
        def __eq__(self, other):
            return type(self) == type(other)
```

Обратите внимание, что метод __eq__ получает второй параметр. Это связано с тем, что при использовании «=» в Python для проверки равенства выполняется метод __eq__. В это время объект с другой стороны знака равенства назначается первому Два параметра.

В этом примере __eq__ проверяет, равны ли они только на основе того, являются ли оба параметра экземплярами класса MyClass, поэтому будет получен следующий результат:

```
>>> MyClass() == MyClass()
True
>>> MyClass() == 23
False
>>> mc = MyClass()
>>> mc2 = MyClass()
>>> mc == mc2
True
```

Переопределение функции print()

Пример

```
class Time:
    def init (self, minutes, seconds): self.minutes =
        minutes  self.seconds = seconds

    def __add__(self, other):
        m = self.minutes + other.minutes
        s = self.seconds + other.seconds
        m += s // 60 s = s
        % 60
        return Time(m, s)

    def __str_(self):
        return '{}:{}'.format(self.minutes, self.seconds)
```

Пример

```
t1 = Time(5, 50)
print(t1)     # 5:50

t2 = Time(3, 20)
print(t2)     # 3:20

t3 = t1 + t2
print(t3)     # 9:10
```

Метод __repr__

Метод __repr__

Метод __repr__ внутри себя вызывает функцию repr, предназначенную для выдачи полной информации об объекте для программиста.

Для нашего класса Time этот метод мог бы выглядеть так:

```
class Time:

    ... методы _init_, _add_, _str_____...

    def__repr__(self):
        return 'Time({}, {})'.format(self.minutes, self.seconds)
```

Метод __repr__

```
t1 = Time(5, 50)
print(t1)
print(repr(t1))
```

Пример 1

```
class Point:
    def __init__(self, name, x, y):
        self.name = name
        self.x = x
        self.y = y

    def __repr__(self):
        return "Point('{}', {}, {})".format(self.name, self.x, self.y)

    def __str__(self):
        return '{}({}, {})'.format(self.name, self.x, self.y)

    def get_x(self):
        return self.x

    def get_y(self):
        return self.y

    def get_coords(self):
        return self.x, self.y

    def __invert__(self):
        return Point(self.name, self.y, self.x)
```

Пример 2

```python
from math import ceil


class Number:
    def __init__(self, number):
        self.number = number

    def __eq__(self, other):
        return self.number == other

    def __gt__(self, other):
        return self.number > other

    def __ceil__(self):
        self.number = ceil(self.number)
        print('Округляем...')
        return self.number
```

Магические методы __add__, __sub__, __mul__, __truediv__

- __add__() – для операции сложения;
- __sub__() – для операции вычитания;
- __mul__() – для операции умножения;
- __truediv__() – для операции деления.

Оператор	Метод оператора	Оператор	Метод оператора
x + y	__add__(self, other)	x += y	__iadd__(self, other)
x - y	__sub__(self, other)	x -= y	__isub__(self, other)
x * y	__mul__(self, other)	x *= y	__imul__(self, other)
x / y	__truediv__(self, other)	x /= y	__itruediv__(self, other)
x // y	__floordiv__(self, other)	x //= y	__ifloordiv__(self, other)
x % y	__mod__(self, other)	x %= y	__imod__(self, other)

Задача:

Для примера рассмотрим класс, описывающий слово. Мы можем сравнивать слова лексиграфически (по алфавиту), что является дефолтным поведением при сравнении строк, но можем захотеть использовать при сравнении какой-нибудь другой критерий, такой, как длина или количество слогов.

В этом примере мы будем сравнивать по длине.

```
class Word(str):
    '''Класс для слов, определяющий сравнение по длине слов.'''

    def __new__(cls, word):
        # Мы должны использовать __new__, так как тип str неизменяемый
        # и мы должны инициализировать его раньше (при создании)
        if ' ' in word:
            print "Value contains spaces. Truncating to first space."
            word = word[:word.index(' ')] # Теперь Word это все символы до первого пробела
        return str.__new__(cls, word)

    def __gt__(self, other):
        return len(self) > len(other)
    def __lt__(self, other):
        return len(self) < len(other)
    def __ge__(self, other):
        return len(self) >= len(other)
    def __le__(self, other):
        return len(self) <= len(other)
```

Class Methods:

Методы класса принимают в качестве первого параметра cls (вместо self в обычных методах) - класс, на котором был вызван метод.

Данный тип методов может использоваться, когда не требуется привязка к экземпляру объекта, но при этом нужно иметь информацию о классе, на
котором он был вызван (например, дополнительные методы инициализации).

Static Methods

Статические методы ничего не знают о классе или об объекте, на котором они вызываются, просто принимая параметры без какого-либо специального аргумента типа self и могут быть вызваны, как через сам класс, так и через его экземпляр.
Данный тип методов может использоваться, когда функция логически принадлежит классу, но не использует сам объект или класс при выполнении.

Рассмотрим первый пример

```
from datetime import date
 class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
     # a class method to create a Person object by birth year.
    @classmethod
    def fromBirthYear(cls, name, year):
        return cls(name, date.today().year - year)
     # a static method to check if a Person is adult or not.
    @staticmethod
    def isAdult(age):
        return age > 18
 person1 = Person('mayank', 21)
person2 = Person.fromBirthYear('mayank', 1996)

print(person1.age)
print(person2.age)

# print the result
print(Person.isAdult(22))
```

Рассмотрим второй пример

```
class Man:
  instances_count = 0
  def __init__(self,name):
    self.name=name
    Man.instances_count+=1
  @staticmethod
  def counter():
    return Man.instances_count

a=Man("a")
b=Man("aa")
c=Man("fga")


print(Man.counter())
```

Рассмотрим третий пример

```
class Point2D:

    instances_count = 0

    def __init__(self, x, y):
        self.x = x
        self.y = y

                Point2D.instances_count += 1

    def __str__(self):
                return 'Точка 2D ({}, {})'.format(self.x, self.y)
```

```
def __add__(self, other):

        if isinstance(other, self.__class__):
                return Point2D(self.x + other.x, self.y + other.y)
    elif isinstance(other, (int, float)):
            self.x += other
        self.y += other
        return self
    else:
                raise TypeError("Не могу добавить {1} к {0}".
                 format(self.__class__, type(other)))
```

```
def __sub__(self, other):
        """Создать новый объект как разность координат self и other."""
        return Point2D(self.x - other.x, self.y - other.y)

    def __neg__(self):
        """Вернуть новый объект, инвертировав координаты."""
        return Point2D(-self.x, -self.y)

    def __eq__(self, other):
        """Вернуть ответ, являются ли точки одинаковыми."""
        return self.x == other.x and self.y == other.y
```

```
def __ne__(self, other):
        return not (self == other)

    @staticmethod
    def sum(*points):
        assert len(points) > 0, "Количество суммируемых точек = 0!"

        res = points[0]
        for point in points[1:]:
            res += point

        return res
```

```
@classmethod
    def from_string(cls, str_value):
            values = [float(x) for x in str_value.split(',')]
        assert len(values) == 2

        return cls(*values)
```

```
if __name__ == "__main__":

    p1 = Point2D(0, 5)
    p2 = Point2D(-5, 10)
    p3 = Point2D.from_string("5, 6")

    print(p1 + p3)  # Точка 2D (5.0, 11.0)

    print(Point2D.instances_count)  # 4 (p1, p2, p3, p1 + p2)

    p4 = Point2D.sum(p1, p2, p3, Point2D(0, -21))
    print(p4)
```


STEP
IT ACADEMY