# **Магические методы \_\_str\_\_, \_\_repr\_\_, \_\_len\_\_, \_\_abs\_\_**

[Смотреть материал на видео](https://www.youtube.com/watch?v=Aabdr3yxEhQ&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E)

На этом занятии я расскажу о, так называемых, магических методах, которые определены в каждом классе и записываются через два подчеркивания вначале и в конце имен, например, так:

\_\_str\_\_, \_\_repr\_\_

Как я говорил, их еще называют

dunder-методами (от англ. сокращения double underscope)

Каждый магический метод автоматически срабатывает в определенный момент времени, например:

* \_\_str\_\_() – магический метод для отображения информации об объекте класса для пользователей (например, для функций print, str);
* \_\_repr\_\_() – магический метод для отображения информации об объекте класса в режиме отладки (для разработчиков).

Чтобы лучше понять, как работают эти методы, объявим класс для описания кошек:

**class** Cat:     **def** \_\_init\_\_(self, name):         self.name = name

Перейдем в консоль Python, скопируем (определим) этот класс. Затем, создадим его экземпляр:

cat = Cat('Васька')

При выводе cat, увидим служебную информацию:

<ex1.Cat object at 0x0495D028>

Если нам нужно ее как-то переопределить и отобразить в другом виде (формате), то, как раз для этого используются магические методы \_\_str\_\_ и \_\_repr\_\_. Давайте для начала переопределим метод \_\_repr\_\_ и посмотрим, как это отразится на выводе служебной информации о классе:

**def** \_\_repr\_\_(self):         **return** f"{self.\_\_class\_\_}: {self.name}"

Обратите внимание, этот метод должен возвращать строку, поэтому здесь записан оператор return и формируемая строка. Что именно возвращать, мы решаем сами, в данном случае – это название класса и имя кошки.

Переопределим измененный класс Cat. И, смотрите, теперь при создании экземпляра мы видим другую информацию при его выводе:

<class 'ex1.Cat'>: Васька

Как раз то, что определили в магическом методе \_\_repr\_\_. То же самое увидим и при использовании функции print и str. По идее, здесь должен отрабатывать другой магический метод \_\_str\_\_, но так как он у нас еще не переопределен, то автоматически выполняется метод \_\_repr\_\_.

Давайте добавим второй магический метод \_\_str\_\_ и посмотрим, как это повлияет на отображение данных:

**def** \_\_str\_\_(self):         **return** f"{self.name}"

Снова переопределим класс Cat, создадим его экземпляр и при отображении ссылки:

cat

по-прежнему будем видеть служебную информацию от метода \_\_repr\_\_.  Однако, если выполнить отображение экземпляра класса через print или str, то будет срабатывать уже второй метод \_\_str\_\_. Вот в этом отличие этих двух магических методов.

## **Магические методы \_\_len\_\_ и \_\_abs\_\_**

Следующие два магических метода:

* \_\_len\_\_() – позволяет применять функцию len() к экземплярам класса;
* \_\_abs\_\_() - позволяет применять функцию abs() к экземплярам класса.

Их использование достаточно простое и очевидное. Давайте для примера представим, что у нас есть класс Point, который может хранить произвольный вектор координат точек, и определим его так:

**class** Point:     **def** \_\_init\_\_(self, \*args):         self.\_\_coords = args

А, далее, по программе нам бы хотелось определять размерность координат с помощью функции len(), следующим образом:

p = Point(1, 2)**print**(len(p))

Если сейчас запустить программу, то увидим ошибку, так как функция len не применима к экземплярам классов по умолчанию. Как вы уже догадались, чтобы изменить это поведение, можно переопределить магический метод \_\_len\_\_() и в нашем случае это можно сделать так:

**def** \_\_len\_\_(self):         **return** len(self.\_\_coords)

Смотрите, мы здесь возвращаем размер списка \_\_coords и если после этого запустить программу, то как раз это значение и будет выведено в консоль. То есть, магический метод \_\_len\_\_ указал, что нужно возвращать, в момент применения функции len() к экземпляру класса. Как видите, все просто и очевидно.

Следующий магический метод \_\_abs\_\_ работает аналогичным образом, только активируется в момент вызова функции abs для экземпляра класса, например, так:

**print**(abs(p))

Опять же, если сейчас выполнить программу, то увидим ошибку, т.к. функция abs не может быть напрямую применена к экземпляру. Но, если переопределить магический метод:

**def** \_\_abs\_\_(self):         **return** list( map(abs, self.\_\_coords) )

который возвращает список из абсолютных значений координат точки, то программа отработает в штатном режиме и мы увидим ожидаемый результат.

Вот так можно использовать эти два магических метода. Я, надеюсь, из этого занятия вы узнали, зачем нужны и как можно реализовывать в своих классах четыре магических метода:

\_\_str\_\_, \_\_repr\_\_, \_\_len\_\_, \_\_abs\_\_