# **Магический метод \_\_bool\_\_ определения правдивости объектов**

[Смотреть материал на видео](https://www.youtube.com/watch?v=a2L5vyCUvzo&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E)

На этом занятии мы поговорим о способах настройки и определения правдивости объектов классов. Что такое правдивость? Это когда к экземпляру явно или неявно применяется функция bool(). С ней мы с вами уже знакомы и применяли к обычным типам данных:

bool(123)bool(-1)bool(0)bool("python")bool("")bool([])

В стандартном поведении она возвращает True для непустых объектов и False – для пустых. Давайте посмотрим, что она будет выдавать для экземпляров классов. Я возьму класс из предыдущего занятия:

**class** Point:     **def** \_\_init\_\_(self, x, y):         self.x = x         self.y = y

Создадим его объект:

p = Point(3, 4)

и применим к нему функцию bool():

**print**(bool(p))

Увидим значение True. В действительности, эта функция всегда возвращает True для любых объектов пользовательского класса. Получается, что смысла в ней особого нет, применительно к экземплярам наших классов? Не совсем. Мы можем переопределить ее поведение либо через магический метод \_\_len\_\_(), либо через метод \_\_bool\_\_():

* \_\_len\_\_() – вызывается функцией bool(), если не определен магический метод \_\_bool\_\_();
* \_\_bool\_\_() – вызывается в приоритетном порядке функцией bool().

Вначале я пропишу магический метод \_\_len\_\_() в классе Point, следующим образом:

**class** Point:     **def** \_\_init\_\_(self, x, y):         self.x = x         self.y = y       **def** \_\_len\_\_(self):         **print**("\_\_len\_\_")         **return** self.x \* self.x + self.y \* self.y

В этом методе я вычисляю и возвращаю квадрат длины радиус-вектора с координатами (x; y). Запустим программу и видим значение True, а также сообщение «\_\_len\_\_». То есть, действительно был вызван метод \_\_len\_\_() и, так как он вернул не нулевое значение, то функция bool() интерпретировала его как True.

Давайте в экземпляре класса пропишем нулевые координаты, чтобы длина вектора была нулевой:

p = Point(0, 0)

Теперь видим ожидаемое значение False.

Конечно, если нам нужно явно описать алгоритм работы функции bool() применительно к нашим экземплярам класса, то следует использовать магический метод \_\_bool\_\_(). Я запишу его в таком виде:

**def** \_\_bool\_\_(self):         **print**("\_\_bool\_\_")         **return** self.x == self.y

Теперь, объект будет считаться правдивым (истинным), если его координаты равны. Запускаем программу и видим, что для нулей отображается значение True. Если же прописать не равные координаты:

p = Point(10, 20)

то получаем значение False. Конечно, такая реализация магического метода \_\_bool\_\_() – это лишь учебный пример, чтобы вы поняли принцип его работы. В реальности, мы можем в этом методе прописывать любую логику. Единственное условие, чтобы данный метод возвращал булево значение True или False. Указывать в операторе return другие типы данных запрещено.

Все это хорошо, но где это используется? Чаще всего в условных конструкциях. Например, если прописать вот такое условие:

**if** p:     **print**("объект p дает True")**else**:     **print**("объект p дает False")

Здесь происходит неявный вызов функции bool() при проверке условия. Поэтому в программах, где требуется описать собственные проверки истинности или ложности объектов, то пользуются или магическим методом \_\_len\_\_(), но чаще всего, магическим методом \_\_bool\_\_().