# **Магические методы \_\_eq\_\_ и \_\_hash\_\_**

[Смотреть материал на видео](https://www.youtube.com/watch?v=Cfx4VCnWeCE&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E)

На этом занятии мы затронем тему вычисления хеша для объектов классов. Вначале что это такое и зачем нужно? В Python имеется специальная функция:

hash(123)hash("Python")hash((1, 2, 3))

которая формирует по определенному алгоритму целочисленные значения для неизменяемых объектов. Причем, для равных объектов на выходе всегда должны получаться равные хэши:

hash("Python")hash((1, 2, 3))

А вот обратное утверждение делать нельзя: равные хэши не гарантируют равенство объектов. Это, как в известном выражении: селедка – это рыба, но не каждая рыба селедка. С хэшами все то же самое.

Однако, если хеши не равны, то и объекты точно не равны. Получаем следующие свойства для хеша:

1. Если объекты a == b (равны), то равен и их хэш.
2. Если равны хеши: hash(a) == hash(b), то объекты могут быть равны, но могут быть и не равны.
3. Если хеши не равны: hash(a) != hash(b), то объекты точно не равны.

Причем, обратите внимание, хэши можно вычислять только для неизменяемых объектов. Например, для списков:

hash([1, 2, 3])

получим ошибку «unhashable type» - не хэшируемый объект.

Итак, мы увидели, что для любого неизменяемого объекта можно вычислять хэш с помощью функции hash(), но зачем все это надо? В действительности некоторые объекты в Python, например, словари используют хэши в качестве своих ключей. Вспомним, когда мы у словаря указываем ключ, то он должен относиться к неизменяемому типу данных:

d = {} d[5] = 5 d["python"] = "python" d[(1, 2, 3)] = [1, 2, 3]

В действительности, это необходимо, чтобы можно было вычислить хеш объектов и ключи хранить в виде:

(хэш ключа, ключ)

Для чего это понадобилось? Дело в том, что первоначально нужная запись в словаре ищется по хэшу, так как существует быстрый алгоритм поиска нужного значения хэша. А, затем, для равных хешей (если такие были обнаружены), отбирается запись с указанным в ключе объекте. Такой подход значительно ускоряет поиск значения в словаре.

Думаю, теперь вы представляете себе, что такое хэш и зачем он нужен. Давайте теперь рассмотрим простой класс Point для представления координат на плоскости:

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

Для экземпляров этого класса:

p1 = Point(1, 2) p2 = Point(1, 2)

мы совершенно спокойно можем вычислять хеш:

**print**(hash(p1), hash(p2), sep='**\n**')

Обратите внимание, несмотря на то, что координаты точек p1 и p2 равны, их хэши разные. То есть, с точки зрения функции hash() – это два разных объекта. Но как она понимает, равные объекты или разные? Все просто. Если оператор сравнения:

**print**(p1 == p2)

дает True, то объекты равны, иначе – не равны. Соответственно, для разных объектов будут получаться и разные хэши. Но раз это так, что будет, если мы переопределим поведение этого оператора сравнения с помощью магического метода \_\_eq\_\_()? Давайте попробуем:

**def** \_\_eq\_\_(self, other):         **return** self.x == other.x **and** self.y == other.y

Теперь у нас объекты с одинаковыми координатами будут считаться равными. Но при запуске программы возникает ошибка «unhashable type», то есть, наши объекты стали не хэшируемыми. Да, как только происходит переопределение оператора ==, то начальный алгоритм вычисления хэша для таких объектов перестает работать. Поэтому, нам здесь нужно прописать свой способ вычисления хэша объектов через магический метод \_\_hash\_\_(), например, так:

**def** \_\_hash\_\_(self):         **return** hash((self.x, self.y))

Мы здесь вызываем функцию hash для кортежа из координат точки. Этот кортеж относится к неизменяемому типу, поэтому для него можно применить стандартную функцию hash(). То есть, мы подменили вычисление хэша объекта класса Point на вычисление хэша от координат точки. Теперь, после запуска программы видим, что объекты равны и их хэши также равны.

Что это в итоге означает? Смотрите, если взять пустой словарь:

d = {}

А, затем, сформировать записи через объекты p1 и p2:

d[p1] = 1 d[p2] = 2**print**(d)

то они будут восприниматься как один и тот же ключ, так как объекты равны и их хэши тоже равны. А вот если магические методы в классе Point поставить в комментарии и снова запустить программу, то увидим, что это уже разные объекты, которые формируют разные ключи словаря. Вот для чего может понадобиться тонко настраивать работу функции hash() для объектов классов. И теперь вы знаете, как это делается.