# **Методы сравнений \_\_eq\_\_, \_\_ne\_\_, \_\_lt\_\_, \_\_gt\_\_ и другие**

[Смотреть материал на видео](https://www.youtube.com/watch?v=l3jMyZKDxXE&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E)

На этом занятии мы будем рассматривать магические методы для реализации операторов сравнения:

* \_\_eq\_\_() – для равенства ==
* \_\_ne\_\_() – для неравенства !=
* \_\_lt\_\_() – для оператора меньше <
* \_\_le\_\_() – для оператора меньше или равно <=
* \_\_gt\_\_() – для оператора больше >
* \_\_ge\_\_() – для оператора больше или равно >=

Рассматривать работу этих методов мы будем на примере нашего класса Clock, который использовали на предыдущем занятии:

**class** Clock:     \_\_DAY = 86400   *# число секунд в одном дне*       **def** \_\_init\_\_(self, seconds: int):         **if** **not** isinstance(seconds, int):             **raise** TypeError("Секунды должны быть целым числом")         self.seconds = seconds % self.\_\_DAY       **def** get\_time(self):         s = self.seconds % 60            *# секунды*         m = (self.seconds // 60) % 60    *# минуты*         h = (self.seconds // 3600) % 24  *# часы*         **return** f"{self.\_\_get\_formatted(h)}:{self.\_\_get\_formatted(m)}:{self.\_\_get\_formatted(s)}"       @classmethod     **def** \_\_get\_formatted(cls, x):         **return** str(x).rjust(2, "0")

Изначально для класса реализован только один метод сравнения на равенство, например:

c1 = Clock(1000) c2 = Clock(1000)**print**(c1 == c2)

Но здесь объекты сравниваются по их id (адресу в памяти), а мы бы хотели, чтобы сравнивались секунды в каждом из объектов c1 и c2. Для этого переопределим магический метод \_\_eq\_\_(), следующим образом:

**def** \_\_eq\_\_(self, other):         **if** **not** isinstance(other, (int, Clock)):             **raise** TypeError("Операнд справа должен иметь тип int или Clock")           sc = other **if** isinstance(other, int) **else** other.seconds         **return** self.seconds == sc

Теперь, после запуска программы видим значение True, т.к. объекты содержат одинаковое время. Кроме того, мы можем совершенно спокойно выполнять проверку и на неравенство:

**print**(c1 != c2)

Смотрите, если интерпретатор языка Python не находит определение метода ==, то он пытается выполнить противоположное сравнение с последующей инверсией результата. То есть, в данном случае находится оператор == и выполняется инверсия:

not (a == b)

Давайте в этом убедимся, поставим точку останова в метод \_\_eq\_\_ и запустим программу. Как видите, он срабатывает и результат в последствии меняется на противоположный.

Отлично, на равенство и неравенство мы теперь можем сравнивать объекты класса Clock, а также с целыми числами. Однако, сравнение на больше или меньше пока не работает. Строчка программы:

**print**(c1 < c2)

приведет к ошибке. Добавим эту операцию сравнения:

**def** \_\_lt\_\_(self, other):         **if** **not** isinstance(other, (int, Clock)):             **raise** TypeError("Операнд справа должен иметь тип int или Clock")           sc = other **if** isinstance(other, int) **else** other.seconds         **return** self.seconds < sc

Как видите, у нас здесь получается дублирование кода. Поэтому, я вынесу общее для методов сравнения в отдельный метода класса:

    @classmethod     **def** \_\_verify\_data(cls, other):         **if** **not** isinstance(other, (int, Clock)):             **raise** TypeError("Операнд справа должен иметь тип int или Clock")           **return** other **if** isinstance(other, int) **else** other.seconds

А сами методы примут вид:

**def** \_\_eq\_\_(self, other):         sc = self.\_\_verify\_data(other)         **return** self.seconds == sc       **def** \_\_lt\_\_(self, other):         sc = self.\_\_verify\_data(other)         **return** self.seconds < sc

Итак, мы определили сравнение на равенство и меньше. Теперь, можно сравнивать объекты класса Clock на эти операции и дополнительно на неравенство и больше. Сейчас команда:

c1 = Clock(1000) c2 = Clock(2000)**print**(c1 < c2)

Выдаст True, так как первое время меньше второго. И также мы можем совершенно спокойно делать проверку на больше:

**print**(c1 > c2)

Здесь сработает тот же метод меньше, но для объекта c2:

c2 < c1

То есть, в отличие от оператора ==, где применяется инверсия, здесь меняется порядок операндов. Разумеется, если в классе определен метод больше:

**def** \_\_gt\_\_(self, other):         sc = self.\_\_verify\_data(other)      **return** self.seconds > sc

то он будет найден и выполнен. Подмена происходит только в случае отсутствия соответствующего магического метода.

И то же самое для методов сравнения на меньше или равно и больше или равно:

**def** \_\_le\_\_(self, other):         sc = self.\_\_verify\_data(other)         **return** self.seconds <= sc

Если мы его вызовем непосредственно для объектов класса:

**print**(c1 <= c2)

то он сработает и результат отобразится в консоли. Но, если пропишем обратное сравнение:

**print**(c1 >= c2)

то просто изменится порядок операндов и будет взято все то же сравнение меньше или равно.

То есть, для определения операций сравнения достаточно в классе определить только три метода: ==, <, <=, если остальные являются их симметричной противоположностью. В этом случае язык Python сам подберет нужный метод и выполнит его при сравнении объектов.

Я, думаю, теперь вы хорошо себе представляете, как можно реализовывать операции сравнения для объектов класса.