# **Магические методы \_\_iter\_\_ и \_\_next\_\_**

Курс по Python ООП: <https://stepik.org/a/116336>

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

Я Сергей Балакирев и на этом занятии мы будем говорить о методах:

* \_\_iter\_\_(self) – получение итератора для перебора объекта;
* \_\_next\_\_(self) – переход к следующему значению и его считывание.

Давайте разберемся для чего они нужны и как их можно использовать. Вы все знаете, как работает функция range(). Она выдает значения арифметической прогрессии, например:

list(range(5))

дает последовательность целых чисел от 0 до 4. Перебрать значения объекта range также можно через итератор:

a = iter(range(5)) next(a) next(a) …

В конце генерируется исключение StopIteration. Так вот, мы можем создать подобный объект, используя магические методы \_\_iter\_\_ и \_\_next\_\_. Давайте это сделаем для объекта frange, который будет выдавать последовательность вещественных чисел арифметической прогрессии. Для этого я объявлю класс:

**class** FRange:     **def** \_\_init\_\_(self, start=0.0, stop=0.0, step=1.0):         self.start = start         self.stop = stop         self.step = step         self.value = self.start - self.step

Здесь в инициализатор мы передаем начальное значение прогрессии, конечное и шаг изменения. Также формируем свойство value, которое будет представлять собой текущее значение для считывания.

Для перебора элементов добавим в этот класс метод, который будет соответствовать магическому методу \_\_next\_\_:

**def** \_\_next\_\_(self):         **if** self.value + self.step < self.stop:             self.value += self.step             **return** self.value         **else**:             **raise** StopIteration

В этом методе мы увеличиваем значение value на шаг step и возвращаем до тех пор, пока не достигли значения stop (не включая его). При достижении конца генерируем исключение StopIteration, ровно так, как это делает объект range.

Сформируем объект этого класса:

fr = FRange(0, 2, 0.5)

и четыре раза вызовем метод \_\_next\_\_()

**print**(fr.\_\_next\_\_())**print**(fr.\_\_next\_\_())**print**(fr.\_\_next\_\_())**print**(fr.\_\_next\_\_())

Видим четыре значения нашей арифметической прогрессии. Если вызвать \_\_next\_\_() еще раз:

**print**(fr.\_\_next\_\_())

получим исключение StopIteration. В целом получился неплохой учебный пример. В действительности, благодаря определению магического метода \_\_next\_\_ в классе FRange, мы можем применять функцию next() для перебора значений его объектов:

fr = FRange(0, 2, 0.5)**print**(next(fr))**print**(next(fr))**print**(next(fr))**print**(next(fr))

Здесь функция next() вызывает метод \_\_next\_\_ и возвращенное им значение, возвращается функцией next(). При этом, в качестве аргумента мы ей передаем экземпляр самого класса. То есть, объект класса выступает в роли итератора. В нашем случае так и задумывалось. Однако, перебрать объект fr с помощью цикла for не получится:

**for** x **in** fr:     **print**(x)

Появится ошибка, что объект не итерируемый. Почему? Ведь мы прописали поведение функции next()? Этого не достаточно. Необходимо еще, чтобы объект возвращал итератор при вызове функции iter:

it = iter(fr)

Для этого в классе нужно прописать еще один магический метод \_\_iter\_\_. В нашем примере он будет выглядеть, так:

**def** \_\_iter\_\_(self):         self.value = self.start - self.step         **return** self

Мы здесь устанавливаем начальное значение value и возвращаем ссылку на объекта класса, так как этот объект в нашем примере и есть итератор – через него вызывается магический метод \_\_next\_\_.

Теперь, после запуска программы у нас не возникает никаких ошибок и цикл for перебирает значения объекта fr. То же самое мы можем сделать и через next():

fr = FRange(0, 2, 0.5) it = iter(fr)**print**(next(it))**print**(next(it))**print**(next(it))**print**(next(it))

Как вы помните, цикл for именно так и перебирает итерируемые объекты, сначала неявно вызывает функцию iter(), а затем, на каждой итерации – функцию next(), пока не возникнет исключение StopIteration. Кроме того, благодаря магическому методу \_\_iter\_\_ мы теперь можем обходить значения объекта fr много раз с самого начала, например:

it = iter(fr)**print**(next(it))**print**(next(it))**print**(next(it))**print**(next(it))   it = iter(fr)**print**(next(it))**print**(next(it))**print**(next(it))**print**(next(it))

Таким образом, сформировали класс FRange, который воспринимается как итерируемый объект с возможностью перебора функцией next() или циклом for.

В заключение этого занятия я приведу пример еще одного класса FRange2D для формирования таблиц значений:

**class** FRange2D:     **def** \_\_init\_\_(self, start=0.0, stop=0.0, step=1.0, rows=5):         self.fr = FRange(start, stop, step)         self.rows = rows

Здесь в инициализаторе создается одномерный объект FRange, который будет формировать строки таблицы. Параметр rows – число строк. Далее, пропишем два магических метода \_\_iter\_\_ и \_\_next\_\_, следующим образом:

**def** \_\_iter\_\_(self):         self.value\_row = 0         **return** self       **def** \_\_next\_\_(self):         **if** self.value\_row < self.rows:             self.value\_row += 1             **return** iter(self.fr)         **else**:             **raise** StopIteration

Обратите внимание, что метод \_\_next\_\_ возвращает не конкретное значение, а итератор на объект класса FRange. Сейчас вы поймете почему так. Создадим объект класса FRange2D:

fr = FRange2D(0, 2, 0.5, 4)

и для перебора его значений нам понадобятся два цикла for:

**for** row **in** fr:     **for** x **in** row:         **print**(x, end=" ")     **print**()

Первый цикл перебирает первый итератор – объект класса FRange2D и на каждой итерации возвращает итератор объекта класса FRange. Именно поэтому мы в методе \_\_next\_\_ класса FRange2D возвращаем иетратор, иначе бы не смогли перебирать объект row во вложенном цикле for.

После запуска программы увидим на экране следующую таблицу чисел:

0.0 0.5 1.0 1.5  
0.0 0.5 1.0 1.5  
0.0 0.5 1.0 1.5  
0.0 0.5 1.0 1.5

Вот общий принцип создания итерируемых объектов. Надеюсь, эти примеры вам были понятны и вы теперь знаете, как и для чего используются магические методы \_\_iter\_\_ и \_\_next\_\_.

Курс по Python ООП: <https://stepik.org/a/116336>