IT CAREER HUB

Урок 40.
Магические методы

IT CAREER HUB

Методы сравнения

Магические методы сравнения позволяют определить, как объекты **сравниваются между собой** при использовании операторов `==`, `!=`, `<`, `<=`, `>`, `>=`.

Они делают возможным:

- сравнение объектов по содержимому, а не по адресу в памяти
- сортировку и поиск в списках
- работу с множествами и словарями

Методы и операторы:

Метод	Оператор	Назначение
`__eq__`	`==`	Равенство
`__ne__`	`!=`	Неравенство
`__lt__`	`<`	Меньше
`__le__`	`<=`	Меньше или равно
`__gt__`	`>`	Больше
`__ge__`	`>=`	Больше или равно

Особенности:

- Если не реализован `__ne__`, Python использует `not __eq__()`
- Если реализован только `__lt__`, то `sort()` всё равно будет работать

Пример: сравнение книг по названию

```python
class Book:
    def __init__(self, title):
        self.title = title
```

```
    def __eq__(self, other):
        if not isinstance(other, Book):
            return NotImplemented
        return self.title == other.title

    def __lt__(self, other):
        return self.title < other.title

    def __repr__(self):
        return f"Book(title={self.title})"


b1 = Book("1984")
b2 = Book("1984")
b3 = Book("Brave New World")

print(b1 == b2)
print(b1 < b3)
print(sorted([b3, b1, b2]))  # сортировка по названию
```

Особенности:

- Всегда проверяйте тип `other` внутри метода, чтобы избежать ошибок (`isinstance`)
- Если метод возвращает `NotImplemented`, Python попробует сравнить наоборот или выбросит исключение

IT CAREER HUB

Декоратор functools.total_ordering

Если необходимо реализовать **сравнение объектов**, то вместо написания **всех шести методов сравнения**, можно воспользоваться **декоратором** `@total_ordering` из модуля `functools`.

Что делает @total_ordering

- Позволяет **сократить количество кода**
- Вы реализуете только `__eq__` **и один из методов:** `__lt__`, `__le__`, `__gt__`, `__ge__`
- Остальные сравнения Python сгенерирует **автоматически**

Пример

```python
from functools import total_ordering

@total_ordering
class Book:
    def __init__(self, title):
        self.title = title

    def __eq__(self, other):
      if not isinstance(other, Book):
          return NotImplemented
      return self.title == other.title

    def __lt__(self, other):
        return self.title < other.title


b1 = Book("1984")
b2 = Book("Brave New World")

print(b1 < b2)    # < реализовано вручную
print(b1 >= b2)   # >= создано автоматически
```

IT CAREER HUB

Методы индексации и доступа к элементам

Чтобы объекты вашего класса можно было использовать **как списки или словари**, нужно реализовать специальные магические методы, которые позволяют:

Метод	Цель	Возвращает	Использование
`__getitem__`	Получать элемент по ключу или индексу	Значение	`value = obj[key]`
`__setitem__`	Устанавливать значение по ключу	`None`	`obj[key] = value`
`__contains__`	Проверять наличие элемента	`True` или `False`	`key in obj`
`__len__`	Возвращать количество элементов	Целое число	`len(obj)`

Пример: коллекция заметок

Допустим, мы хотим создать класс `Notes`, в котором:

- автоматически фильтруются пустые строки,
- ключи автоматически приводятся к нижнему регистру,
- можно использовать `len()`, `in`, индексацию и присваивание.

```python
class Notes:
    def __init__(self):
        self._data = {}

    def __getitem__(self, key):
        return self._data[key.lower()]  # доступ по ключу (в нижнем регистре)

    def __setitem__(self, key, value):
        if value.strip():  # сохраняем только непустые строки
```

```
            self._data[key.lower()] = value
        else:
            raise ValueError("Empty notes are not allowed")

    def __contains__(self, key):
        return key.lower() in self._data  # проверка in

    def __len__(self):
        return len(self._data)  # длина коллекции

    def __str__(self):
        result = "Notes:\n"
        for key, value in self._data.items():
            result += f"- {key}: {value}\n"
        return result.strip()


notes = Notes()
notes["Idea"] = "Build a game"        # присваивание по ключу
notes["TODO"] = "Finish the project"  # присваивание по ключу

print(notes["idea"])      # доступ по ключу
print("todo" in notes)    # вхождение элемента игнорируя регистр
print(len(notes))         # количество элементов
print(notes)              # строковое представление
```

IT CAREER HUB

Задания для закрепления 1

Укажите верные утверждения о @total_ordering:

a. Нужно реализовать все шесть методов сравнения вручную
b. Достаточно реализовать `__eq__` и один из `__lt__, __le__, __gt__, __ge__`
c. `@total_ordering` автоматически создаёт недостающие методы сравнения
d. Работает только с числами

Посмотреть ответ

IT CAREER HUB

Арифметические методы

Магические методы позволяют переопределить поведение **арифметических операторов** (`+`, `-`, `*`, `/` и т.д.) для объектов пользовательских классов. Это удобно, когда объект представляет **число, вектор, матрицу** или любую другую структуру, с которой логично выполнять арифметические действия.

Метод	Оператор	Назначение
`__add__`	`+`	Сложение
`__sub__`	`-`	Вычитание
`__mul__`	`*`	Умножение
`__truediv__`	`/`	Деление
`__floordiv__`	`//`	Целочисленное деление
`__mod__`	`%`	Остаток от деления
`__pow__`	`**`	Возведение в степень

Пример: класс для 2D-вектора

Python

```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        if not isinstance(other, Vector):
            return NotImplemented
        return Vector(self.x + other.x, self.y + other.y)

    def __str__(self):
        return f"({self.x}, {self.y})"
```

```
a = Vector(2, 3)  # координаты смещения
b = Vector(1, 4)  # координаты смещения
c = a + b         # объединение координат смещения
print(c)
```

Особенности:

- Метод получает **два аргумента**: `self` (до оператора) и `other` (после оператора)
- Можно проверять тип `other`, чтобы обрабатывать только нужные случаи

Инкрементные арифметические методы

Если объект поддерживает **изменение на месте**, можно реализовать специальные методы для работы с операторами `+=`, `-=`, `*=`, `/=` и др. Они называются **инкрементными** и начинаются с `__i...__`. Метод должен **изменять объект и возвращать его**.

Метод	Оператор	Назначение
`__iadd__`	`+=`	Сложение с присваиванием
`__isub__`	`-=`	Вычитание с присваиванием
`__imul__`	`*=`	Умножение с присваиванием
`__itruediv__`	`/=`	Деление с присваиванием
`__ifloordiv__`	`//=`	Целочисленное деление с присваиванием
`__imod__`	`%=`	Остаток от деления с присваиванием
`__ipow__`	`**=`	Возведение в степень с присваиванием

IT CAREER HUB

Пример

Python

```python
class Counter:
    def __init__(self, value=0):
        self.value = value

    def __iadd__(self, other):
        self.value += other
        return self

    def __str__(self):
        return f"Counter({self.value})"

c = Counter(5)
print(id(c))
c += 3
print(id(c))  # id объекта не изменился
print(c)
```

IT CAREER HUB

Магический метод __bool__

Метод `__bool__` определяет, как объект ведёт себя в логических выражениях — например, при проверке в `if`, `while`.

Пример: считается «пустым» при нуле и меньше

Python

```
class Counter:
    def __init__(self, value):
        self.value = value

    def __bool__(self):
        return self.value > 0


c1 = Counter(-5)
c2 = Counter(3)

print(bool(c1))
print(bool(c2))

if c1:
    print("Есть элементы")
else:
    print("Пусто")
```

IT CAREER HUB

Магический метод __call__

Метод `__call__` позволяет делать объект вызываемым, то есть обращаться к нему как к функции — через круглые скобки: `obj()`. Он позволяет использовать объект как функцию с внутренним состоянием

Пример: объект-конвертер

```python
class CurrencyConverter:
    def __init__(self, rate):
        self.rate = rate  # сколько единиц нужно за 1 доллар

    def __call__(self, dollars):
        return dollars * self.rate

# Курс: 1 доллар = 0.88 евро
euro_converter = CurrencyConverter(0.88)

print(euro_converter(10))
print(euro_converter(5.5))
```

Особенности:

- Метод вызывается при `obj(...)`
- Может принимать аргументы, как обычная функция

IT CAREER HUB

Задания для закрепления 2

1. **Найдите ошибку в коде:**

```python
class Box:
    def __init__(self, weight):
        self.weight = weight

    def __add__(self):
        return Box(self.weight + 1)
```

Посмотреть ответ

2. **Укажите, что верно о методе __call__:**

a. Делает объект вызываемым, как функцию
b. Используется только для арифметических операций
c. Может принимать аргументы

Посмотреть ответ

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
Утверждения о @total_ordering	Ответ: b, c
Задания на закрепление 2	Вернуться к заданиям
1. Ошибка в коде	Ответ: у метода `__add__` должно быть два аргумента (`self, other`).
2. Утверждения о методе __call__	Ответ: a, c