Объектно-ориентированное программирование

Объявление класса TextPost (textpost.py)

TextPost
author: str text: str date: datetime

```python
from datetime import datetime, timezone

class TextPost:
    """Класс текстового поста для блога"""

    def __init__(self, author, text):
        print(f"TextPost.__init__(). self: {self}")
        self.author = author
        self.text = text
        self.date = datetime.now(timezone.utc)
```

src/12. OOP/example_01/textpost.py

Создание экземпляра класса TextPost (main.py)


```python
from textpost import TextPost

if __name__ == "__main__":
    post = TextPost("Толстой Л.Н.", "Очень длинный текст...")

    print(f"{type(post)=}")
    print()
    print(dir(post))
    print()
    print("Автор:", post.author)
    print("Дата:", post.date)
```

src/12. OOP/example_01/main.py

```
> python main.py
```

TextPost.__init__(). self: <textpost.TextPost object at 0x7fa090f47380>
type(post)=<class 'textpost.TextPost'>

['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__firstlineno__', '__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__static_attributes__', '__str__', '__subclasshook__', '__weakref__', 'author', 'date', 'text']

Автор: Толстой Л.Н.
Дата: 2025-03-23 07:36:39.146816+00:00

Не рекомендуемый способ добавления полей класса (main.py)


```python
from textpost import TextPost

if __name__ == "__main__":
    post = TextPost("Толстой Л.Н.", "Очень длинный текст...")

    print(dir(post))

    # Не рекомендуемый способ добавления полей класса
    post.title = "Война и мир"
    print()
    print(dir(post))
```

src/12. OOP/example_02/main.py

```
> python main.py
```

TextPost.__init__(). self: <textpost.TextPost object at 0x7f6174b374d0>
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__firstlineno__',
'__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__static_attributes__', '__str__',
'__subclasshook__', '__weakref__', **'author', 'date', 'text'**]

['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__firstlineno__',
'__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__static_attributes__', '__str__',
'__subclasshook__', '__weakref__', **'author', 'date', 'text', 'title'**]

Изменение полей класса (main.py)


```python
from textpost import TextPost

if __name__ == "__main__":
    post = TextPost("Толстой Л.Н.", "Очень длинный текст...")

    print("Автор:", post.author)
    print("Дата:", post.date)
    print("Текст:", post.text)

    post.author = "Чехов А.П."
    print()
    print("Автор:", post.author)
    print("Дата:", post.date)
    print("Текст:", post.text)
```

src/12. OOP/example_03/main.py

```
> python main.py
```

TextPost.__init__(). self: <textpost.TextPost object at 0x7f32ac147380>
Автор: Толстой Л.Н.
Дата: 2025-03-23 07:53:58.041103+00:00
Текст: Очень длинный текст...

Автор: Чехов А.П.
Дата: 2025-03-23 07:53:58.041103+00:00
Текст: Очень длинный текст...

Соглашения об области видимости полей класса (textpost.py)


```python
from datetime import datetime, timezone

class TextPost:
    """Класс текстового поста для блога"""

    def __init__(self, author, text):
        self._author = author
        self._text = text
        self._date = datetime.now(timezone.utc)
        self._save()

    def _save(self):
        print("Пост сохранен.")

    def get_author(self): return self._author

    def get_text(self): return self._text

    def set_text(self, value):
        self._text = value
        self._save()

    def get_date(self): return self._date
```

TextPost
_author: str _text: str _date: datetime
_save(): None get_author(): str get_text(): str set_text(str): None get_date(): datetime

```python
from textpost import TextPost

if __name__ == "__main__":
    post = TextPost("Толстой Л.Н.", "Очень длинный текст...")

    print("Автор:", post.get_author())
    print("Дата:", post.get_date())

    print("Изменяем текст поста")
    post.set_text("Еще более длинный текст...")
```

```
> python main.py
```

Пост сохранен.
Автор: Толстой Л.Н.
Дата: 2025-03-23 08:21:07.757908+00:00
Изменяем текст поста
Пост сохранен.

Нарушение соглашения об области видимости полей класса (main.py)


```python
from textpost import TextPost

if __name__ == "__main__":
    post = TextPost("Толстой Л.Н.", "Очень длинный текст...")

    print("Автор:", post._author)
    print("Дата:", post._date)

    print("Изменяем текст поста")
    post._text = "Еще более длинный текст..."
```

src/12. OOP/example_05/main.py

```
> python main.py
```

Пост сохранен.
Автор: Толстой Л.Н.
Дата: 2025-03-23 08:26:03.064119+00:00
Изменяем текст поста

Свойства класса

Класс со свойствами (textpost.py)


```python
class TextPost:
    """Класс текстового поста для блога"""

    def __init__(self, author, text):
        self._author = author
        self._text = text
        self._date = datetime.now(timezone.utc)
        self._save()

    def _save(self):
        print("Пост сохранен.")

    @property
    def author(self): return self._author

    @property
    def text(self): return self._text

    @text.setter
    def text(self, value):
        self._text = value
        self._save()

    @property
    def date(self): return self._date
```

TextPost
_author: str _text: str _date: datetime
_save(): None author(): str text(): str text(str): None date(): datetime

src/12. OOP/example_06/textpost.py

Использование свойств класса (main.py)

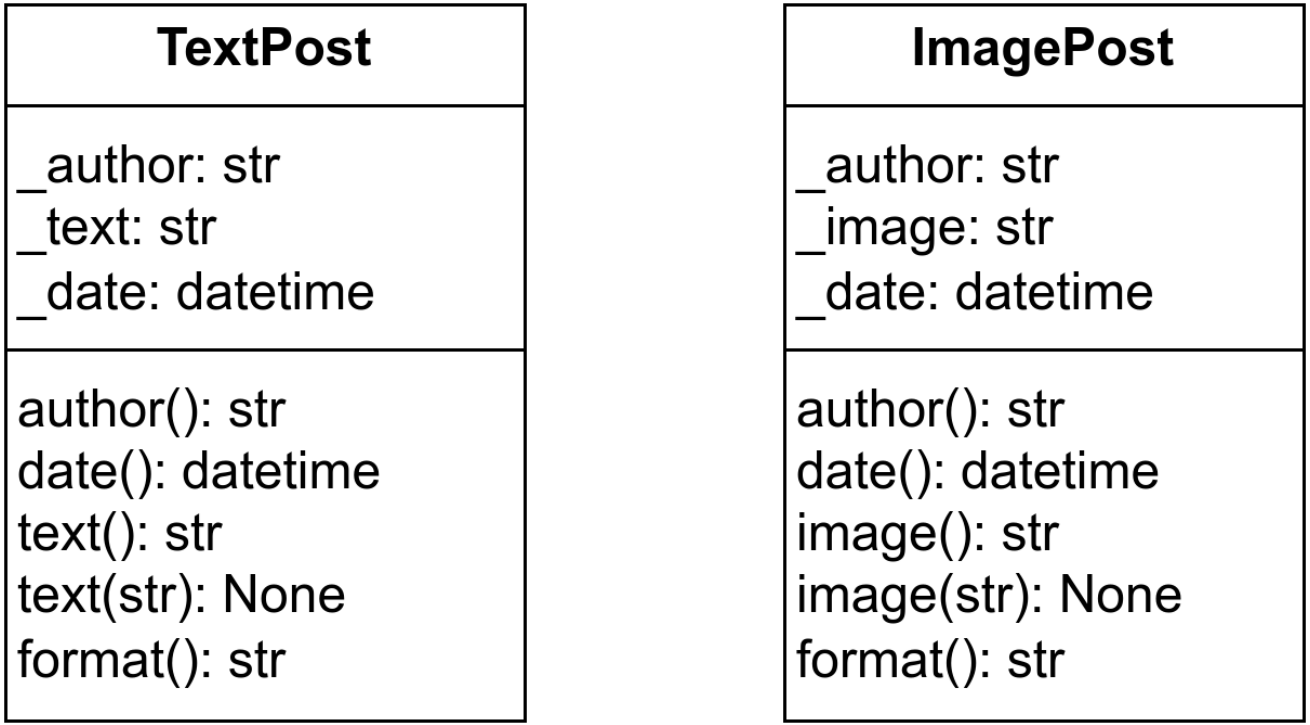
```python
from textpost import TextPost

if __name__ == "__main__":
    post = TextPost("Толстой Л.Н.", "Очень длинный текст...")

    print("Автор:", post.author)
    print("Дата:", post.date)

    print("Изменяем текст поста")
    post.text = "Еще более длинный текст..."
```

```
> python main.py
```

Пост сохранен.
Автор: Толстой Л.Н.
Дата: 2025-03-23 08:42:14.245331+00:00
Изменяем текст поста
Пост сохранен.

Наследование

UML-диаграмма классов без использования наследования



src/12. OOP/example_07/

Использование классов TextPost и ImagePost (main.py)


```
from textpost import TextPost
from imagepost import ImagePost

post1 = TextPost("Толстой Л.Н.", "Очень длинный текст...")
post2 = ImagePost("Малевич К.С.", "black_square.jpg")

print(post1.format())
print()
print(post2.format())
```

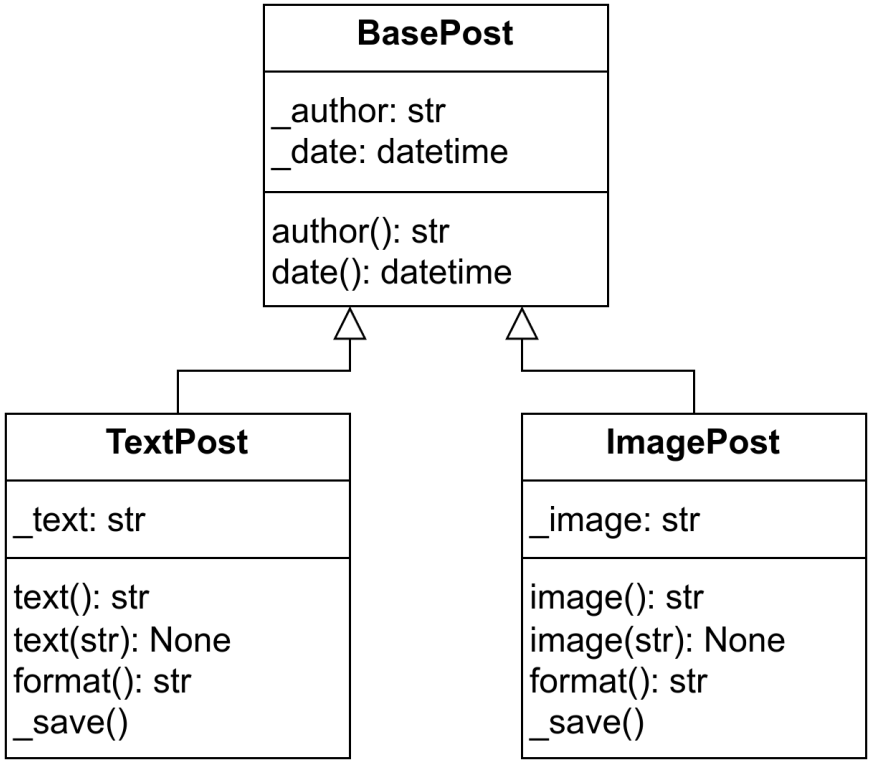
```
> python main.py
```

Текстовый пост сохранен.
Пост с картинкой сохранен.
====
Автор: Толстой Л.Н.
Дата: 2025-03-23 09:30:27.403127+00:00

Очень длинный текст...
====

====
Автор: Малевич К.С.
Дата: 2025-03-23 09:30:27.403145+00:00
Картинка: black_square.jpg
====

UML-диаграмма классов с использованием базового класса



src/12. OOP/example_08/

Базовый класс BasePost

```python
from datetime import datetime, timezone

class BasePost:
    """Базовый класс для постов блога"""

    def __init__(self, author):
        print("BasePost.__init__()")
        self._author = author
        self._date = datetime.now(timezone.utc)

    @property
    def author(self): return self._author

    @property
    def date(self): return self._date
```

Конструкторы производных классов

```
class TextPost(BasePost):
    """Класс текстового поста для блога"""

    def __init__(self, author, text):
        print("TextPost.__init__()")
        super().__init__(author)
        self._text = text
        self._save()
    ...

class ImagePost(BasePost):
    """Класс поста с картинкой для блога"""

    def __init__(self, author, image):
        print("TextPost.__init__()")
        super().__init__(author)
        self._image = image
        self._save()
    ...
```

Использование классов TextPost и ImagePost (main.py)


```
from textpost import TextPost
from imagepost import ImagePost

post1 = TextPost("Толстой Л.Н.", "Очень длинный текст...")
post2 = ImagePost("Малевич К.С.", "black_square.jpg")

print(post1.format())
print()
print(post2.format())
```

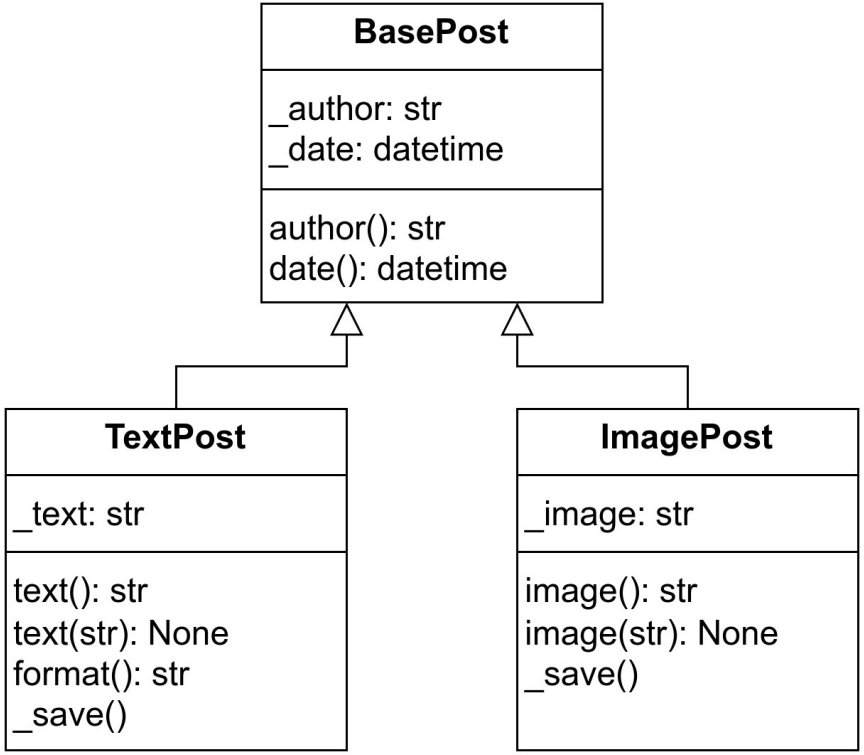
```
> python main.py
```

```
TextPost.__init__()
BasePost.__init__()
Текстовый пост сохранен.
TextPost.__init__()
BasePost.__init__()
Пост с картинкой сохранен.
====
Автор: Толстой Л.Н.
Дата: 2025-03-23 09:44:55.251337+00:00
----
Очень длинный текст...
====

====
Автор: Малевич К.С.
Дата: 2025-03-23 09:44:55.251371+00:00
Картинка: black_square.jpg
====
```

src/12. OOP/example_08/

Абстрактные базовые классы

Пример с ошибкой. Отсутствует ожидаемый метод



src/12. OOP/example_09/

Пример с ошибкой. Отсутствует ожидаемый метод


```
from textpost import TextPost
from imagepost import ImagePost

feed = []
feed.append(TextPost("Толстой Л.Н.", "Очень длинный текст..."))
feed.append(ImagePost("Малевич К.С.", "black_square.jpg"))

for post in feed:
    print(post.format())
```

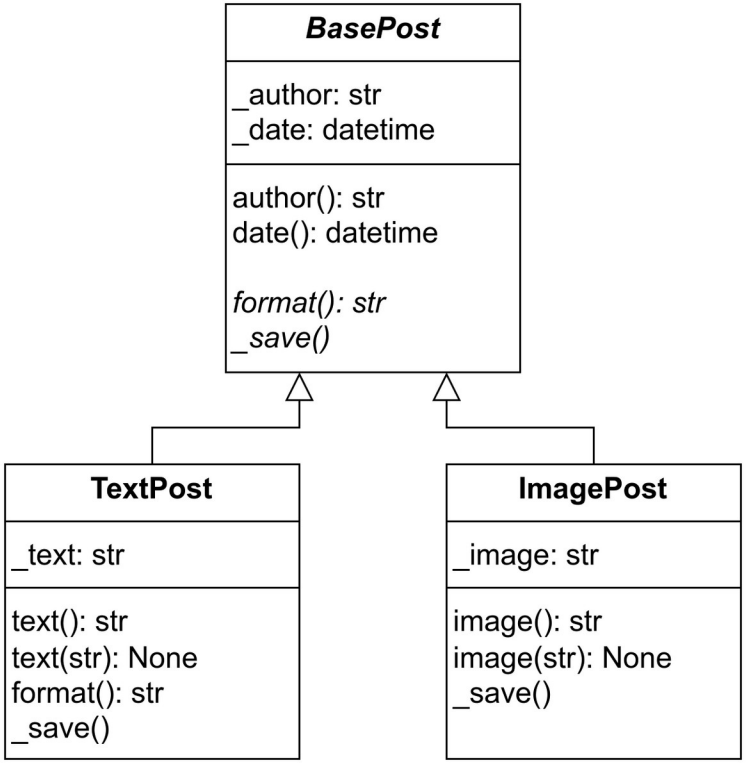
src/12. OOP/example_09/main.py

```
> python main.py
```

```
TextPost.__init__()
BasePost.__init__()
Текстовый пост сохранен.
TextPost.__init__()
BasePost.__init__()
Пост с картинкой сохранен.
====
Автор: Толстой Л.Н.
Дата: 2025-03-23 09:53:35.771003+00:00
---
Очень длинный текст...
====
Traceback (most recent call last):
  File ".../main.py", line 9, in <module>
    print(post.format())
          ^^^^^^^^^^^
AttributeError: 'ImagePost' object has no attribute 'format'
```

src/12. OOP/example_09/

Диаграмма с абстрактным базовым классом BasePost и отсутствующим методом format() в классе ImagePost



src/12. OOP/example_10/

Абстрактный базовый класс BasePost

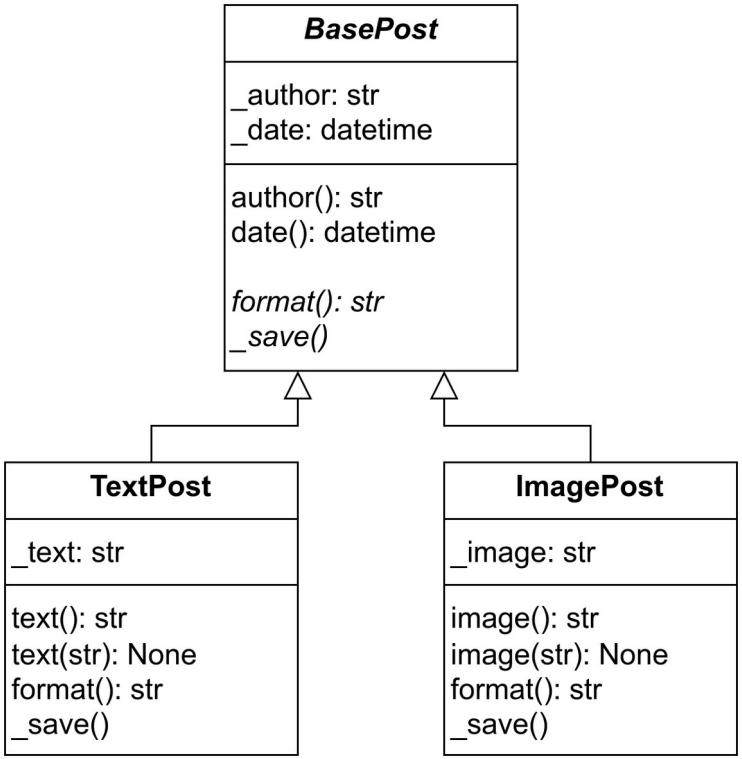
```python
from abc import ABCMeta, abstractmethod
from datetime import datetime, timezone

class BasePost(metaclass=ABCMeta):
    """Базовый класс для постов блога"""

    def __init__(self, author):
        print("BasePost.__init__()")
        self._author = author
        self._date = datetime.now(timezone.utc)

    @property
    def author(self): return self._author

    @property
    def date(self): return self._date

    @abstractmethod
    def format(self):
        ...

    @abstractmethod
    def _save(self):
        ...
```

```
> python main.py
```

```
TextPost.__init__()
BasePost.__init__()
Текстовый пост сохранен.
Traceback (most recent call last):
  File ".../main.py", line 6, in <module>
    feed.append(ImagePost("Малевич К.С.", "black_square.jpg"))
                ~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: Can't instantiate abstract class ImagePost without an implementation for abstract method 'format'
```

src/12. OOP/example_10/

Диаграмма с абстрактным базовым классом BasePost



src/12. OOP/example_11/

```
> python main.py
```

```
TextPost.__init__()
BasePost.__init__()
Текстовый пост сохранен.
TextPost.__init__()
BasePost.__init__()
Пост с картинкой сохранен.
====
Автор: Толстой Л.Н.
Дата: 2025-03-23 10:13:56.335096+00:00
---
Очень длинный текст...
====
====
Автор: Малевич К.С.
Дата: 2025-03-23 10:13:56.335112+00:00
Картинка: black_square.jpg
====
```

src/12. OOP/example_11/

Специальные методы класса ("магические" методы, dunder-методы)

* dunder — **d**ouble **under**scores

```python
def add(a, b):
    return a + b

def mul(a, b):
    return a * b

def spam(a, b, action):
    return action(a, b) * 2 + 1

foo = spam(2, 3, add)
bar = spam(2, 3, mul)

print(dir(add))
print()

print(f"{foo=}")
print(f"{bar=}")
```

```
['__annotations__', '__builtins__', '__call__', '__class__', '__closure__', '__code__', '__defaults__', '__delattr__',
'__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get__', '__getattribute__', '__getstate__',
'__globals__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__kwdefaults__', '__le__', '__lt__', '__module__',
'__name__', '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__']

foo=11
bar=13
```

```python
import math

class MathAction:
    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def __call__(self, a, b) -> float:
        print("MathAction.__call__()")
        return a * math.sin(self.x) + b * math.cos(self.y)


def add(a, b):
    return a + b

def spam(a, b, action):
    return action(a, b) * 2 + 1

foo = spam(2, 3, add)
bar = spam(2, 3, MathAction(math.pi / 3, math.pi / 4))

print(f"{foo=}")
print(f"{bar=}")
```

src/12. OOP/example_12/

"Магический" метод	Назначение
__init__	Конструктор класса.
__call__	Делает объект вызываемым, т. е. добавляет возможность применения оператора "круглые скобки".
__hash__	Делает объект хэшируемым (см. главу 6).
__len__	Позволяет применять функцию len() к объекту.
__getitem__	Добавляет возможность применения оператора "квадратные скобки" для получения значения.
__setitem__	Добавляет возможность применения оператора "квадратные скобки" для изменения значения.

"Магический" метод	Назначение
__add__	Позволяет применять к объекту оператор суммирования "+".
__sub__	Позволяет применять к объекту оператор вычитания "–".
__mul__	Позволяет применять к объекту оператор умножения "*".
__truediv__	Позволяет применять к объекту оператор деления "/".
__floordiv__	Позволяет применять к объекту оператор целочисленного деления "//".
__mod__	Позволяет применять к объекту оператор вычисления остатка от деления "%".

"Магический" метод	Назначение
__str__	Возвращает строковое представление объекта.
__repr__	Возвращает строковое представление объекта при передачи его в функцию repr(). Иногда это более подробный формат, используемый для отладки, по сравнению с __str__().
__bool__	Позволяет преобразовывать объект в булево значение.
__int__	Позволяет преобразовывать объект в целое число.
__float__	Позволяет преобразовывать объект в дробное число.

"Магический" метод	Назначение
__lt__	Позволяет применять к объекту оператор "<"
__le__	Позволяет применять к объекту оператор "<="
__gt__	Позволяет применять к объекту оператор ">"
__ge__	Позволяет применять к объекту оператор ">="
__eq__	Позволяет применять к объекту оператор "=="
__ne__	Позволяет применять к объекту оператор "!="
__contains__	Позволяет применять к объекту оператор in.