

?????

- [Магические методы](#)



## 2. Арифметические операторы

Возвращается всегда новая переменная. Существует три варианта в случае арифметических операций:

- Базовый. Вызывается первым, для объекта слева. Объект справа может быть любого типа. Внутри нужно проверять принадлежность к классу для второго аргумента. В случае ошибки / несоответствия типов / ... вызывается NotImplemented.

```
class MyClass:
    def __add__(self, other):
        if isinstance(other, MyClass):
            return MyClass(self.value + other.value)
        elif isinstance(other, (int, float)):
            return MyClass(self.value + other)
        else:
            # Не знаю, что делать
            return NotImplemented
```

- Если был вызван NotImplemented, право сделать что-то передается объекту справа (правые версии). Если правая версия не определена, то будет TypeError. Но если порядок не важен, можно сделать так:

```
class MyClass:
    ...
    def __radd__(self, other):
        return self.__add__(other)
```

- Существует способ записи например += Для него может быть отдельный (расширенный) метод. Если не определять, то вызывается соответствующий базовый метод. Можно менять сам объект. Обязательно возвращать объект.

Операция	Базовые методы	Правые версии		Расширенные версии	
+	<code>__add__(self, other)</code>	<code>__radd__(self, other)</code>	<code>other + self</code>	<code>__iadd__(self, other)</code>	<code>+=</code>
-	<code>__sub__(self, other)</code>	<code>__rsub__(self, other)</code>	<code>other - self</code>	<code>__isub__(self, other)</code>	<code>-=</code>
*	<code>__mul__(self, other)</code>	<code>__rmul__(self, other)</code>	<code>other * self</code>	<code>__imul__(self, other)</code>	<code>*=</code>
/	<code>__truediv__(self, other)</code>	<code>__rtruediv__(self, other)</code>	<code>other / self</code>	<code>__itruediv__(self, other)</code>	<code>/=</code>

Операция	Базовые методы	Правые версии		Расширенные версии	
//	<code>__floordiv__(self, other)</code>	<code>__rfloordiv__(self, other)</code>	<code>other // self</code>	<code>__ifloordiv__(self, other)</code>	<code>//=</code>
%	<code>__mod__(self, other)</code>	<code>__rmod__(self, other)</code>	<code>other % self</code>	<code>__imod__(self, other)</code>	<code>%=</code>
** pow()	<code>__pow__(self, other[, modulo])</code>	<code>__rpow__(self, other)</code>	<code>other ** self</code>	<code>__ipow__(self, other)</code>	<code>**=</code>
-obj	<code>__neg__(self)</code>				
+obj	<code>__pos__(self)</code>				
abs()	<code>__abs__(self)</code>				
round()	<code>__round__(self[, n])</code>				

### 3. Битовые операторы

Операция	Базовые методы	Правые версии	Расширенные версии	
&	<code>__and__(self, other)</code>	<code>__rand__(self, other)</code>	<code>__iand__(self, other)</code>	<code>&amp;=</code>
	<code>__or__(self, other)</code>	<code>__ror__(self, other)</code>	<code>__ior__(self, other)</code>	<code> =</code>
^	<code>__xor__(self, other)</code>	<code>__rxor__(self, other)</code>	<code>__ixor__(self, other)</code>	<code>^=</code>
<<	<code>__lshift__(self, other)</code>	<code>__rlshift__(self, other)</code>	<code>__ilshift__(self, other)</code>	<code>&lt;&lt;=</code>
>>	<code>__rshift__(self, other)</code>	<code>__rrshift__(self, other)</code>	<code>__irshift__(self, other)</code>	<code>&gt;&gt;=</code>
~	<code>__invert__(self)</code>			

### 4. Методы для контекстного менеджера (`with`)

<code>__enter__(self)</code>	Вход в контекст
<code>__exit__(self, exc_type, exc_val, exc_tb)</code>	Выход из контекста (с обработкой исключений)

### 5. Работа с атрибутами

<code>__getattr__(self, name)</code>	при обращении к несуществующему атрибуту
<code>__setattr__(self, name, value)</code>	при установке любого атрибута

<code>__delattr__(self, name)</code>	<code>del obj.name</code>
<code>__getattr__(self, name)</code>	при обращении к ЛЮБОМУ атрибуту (осторожно, рекурсия!), редко используется
<code>__dir__(self)</code>	<code>dir()</code>
<code>__hasattr__(self, name)</code>	<code>hasattr()</code> (не нужен — <code>__getattr__</code> обработает)

## 6. Вызов объекта как функции

`__call__(self, *args, **kwargs) # obj()`

## 7. Работа с классами и метаклассами

<code>__init_subclass__(cls, **kwargs)</code>	при создании подкласса
<code>__set_name__(self, owner, name)</code>	при создании дескриптора в классе
<code>__prepare__(name, bases, **kwargs)</code>	метакласс: подготовка пространства имён
<code>__instancecheck__(self, instance)</code>	<code>isinstance()</code> (для метаклассов)
<code>__subclasscheck__(self, subclass)</code>	<code>issubclass()</code> (для метаклассов)

## 8. Дескрипторы (управление атрибутами другого класса)

`__get__(self, instance, owner) # получить атрибут`

`__set__(self, instance, value) # установить атрибут`

`__delete__(self, instance) # удалить атрибут`

## 9. Сериализация

<code>__reduce__(self)</code>	<code>pickle</code> (возвращает <code>(callable, args[, state])</code> )
<code>__reduce_ex__(self, protocol)</code>	расширенная версия для <code>pickle</code>
<code>__getstate__(self)</code>	что сохранять в <code>pickle</code>
<code>__setstate__(self, state)</code>	восстановление из <code>pickle</code>

## 10. Математические и другие

<code>__complex__(self)</code>	<code>complex()</code>
<code>__int__(self)</code>	<code>int()</code>
<code>__float__(self)</code>	<code>float()</code>

<code>__index__(self)</code>	для преобразования в int (для срезов, <code>bin()</code> , <code>hex()</code> )
<code>__trunc__(self)</code>	<code>math.trunc()</code>
<code>__floor__(self)</code>	<code>math.floor()</code>
<code>__ceil__(self)</code>	<code>math.ceil()</code>
<code>__matmul__(self, other)</code>	@ (матричное умножение в Python 3.5+)
<code>__rmatmul__(self, other)</code>	right @
<code>__imatmul__(self, other)</code>	@=

## 11. Асинхронные методы (async/await)

<code>__await__(self)</code>	await obj
<code>__aiter__(self)</code>	async for
<code>__anext__(self)</code>	async next()
<code>__aenter__(self)</code>	async with
<code>__aexit__(self, exc_type, exc_val, exc_tb)</code>	async with exit

### Самые частые методы:

`__init__`, `__str__`, `__repr__` — 90% обычных классов

`__add__`, `__radd__`, `__iadd__` — для своего класса с `+`

`__getitem__`, `__setitem__`, `__len__` — чтобы объект вёл себя как коллекция

`__enter__`, `__exit__` — для `with`

`__call__` — сделать объект вызываемым (как функция)

`__eq__`, `__lt__`, `__hash__` — для сравнения и словарей