Магические методы

Магические методы

Магические методы, или **базовые методы** – способствуют реализации свойств объекта при их взаимодействие (добавляют «магию»).

При записи магические методы указывается при помощи *двух нижних подчеркиваний с обеих сторон* (без пробелов). Пример: __init__.

Методы __init__ и __del__

_ _init_ _

Метод _ _init_ _ («dunder-init», «дандер инит») – метод отвечающий за инициализацию (задание начального значения) экземпляров класса после их создания.

Инициализатор позволяет получить полностью настроенный экземпляр класса.

Запись метода __init__(self[, ...])

```
1  class User:
2      def __init__(self, first_name, last_name):
3          self.first_name = first_name
4          self.last_name = last_name
5
6  user = User("Иван", "Иванов")
```

1 2

1 2

__del__(self)

Метод __del__ - деструктор класса.

```python
class User:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def __del__(self):
        del self.first_name

    def line(self):
        print(self.first_name, self.last_name)

user = User("Иван", "Иванов")
user.line()
user.__del__()
user.line()
```

```
Иван Иванов
Traceback (most recent call last):
  File "/tmp/sessions/65125d8753cdd428/main.py", line 15, in <module>
    user.line()
  File "/tmp/sessions/65125d8753cdd428/main.py", line 10, in line
    print(self.first_name, self.last_name)
AttributeError: 'User' object has no attribute 'first_name'. Did you mean: 'last_name'?
```

Методы __str__ и __repr__
Методы __len__ и __abs__

__str__(self)

```python
class User:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def __str__(self):
        return f"__str__ method: {self.first_name} {self.last_name}"

user = User("Иван", "Иванов")
print(f"{user}")
```

```
__str__ method: Иван Иванов
```

__repr__(self)

```python
class User:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def __repr__(self):
        return f"__repr__ method: {self.first_name} {self.last_name}"

user = User("Иван", "Иванов")
print(f"{repr(user)}")
print(f"{user!r}")
```

```
__repr__ method: Иван Иванов
__repr__ method: Иван Иванов
```

__len__(self)

```python
class Line:
    def __init__(self, *args):
        self.__coords = args


a = Line(1, 2)
print(len(a))
```

```
Traceback (most recent call last):
  File "/tmp/sessions/436b820d9b715975
/main.py", line 7, in <module>
    print(len(a))
TypeError: object of type 'Line' has no len()
```

```python
class Line:
    def __init__(self, args):
        self.__values = args #Создаем приватный атрибут
    def __len__(self):
      return len(self.__values)

a = Line("len")
print(len(a))
```

```
3
```

```python
class Line:
    def __init__(self, *args):
        self.__values = args #Создаем приватный атрибут
    def __len__(self):
      return len(self.__values)

a = Line(1, 2, 3, "len")
print(len(a))
```

```
4
```

__abs__(self)

```python
class Otrezok:
    def __init__(self, x1, x2):
        self.x1 = x1
        self.x2 = x2

    def __abs__(self):
        return abs(self.x2 - self.x1)

a = Otrezok(5, 9)
print(abs(a))
```

```
4
```

Магические методы для арифметических операций

- __add__(self, other) – для операции +;
- __sub__(self, other) – для операции -;
- __mul__(self, other) – для операции *;
- __truediv__(self, other) – для операции /;
- __floordiv__(self, other) – для операции //;
- __mod__(self, other) – для операции %.

__add__(self, other)

```
class BankAccount:
    def __init__(self, *args):
        self.name = args[0]
        self.balance = args[1]


Monty = BankAccount("Monty", 12000)
print(Monty.balance + 100)
print(Monty.balance)
```

```
12100
12000
```

```
class BankAccount:
    def __init__(self, *args):
        self.name = args[0]
        self.balance = args[1]
    def __add__(self, other):
        self.balance += other
        return self.balance


Monty = BankAccount("Monty", 12000)
Monty.__add__(150)
print(Monty.balance)
Monty+200
print(Monty.balance)
```

```
12150
12350
```

__mul__(self, other)

```python
class BankAccount:
    percent = 1.2

    def __init__(self, *args):
        self.name = args[0]  # Имя
        self.balance = args[1]  # Баланс

    def __mul__(self, other):
        if isinstance(other, (int, float)):
            BankAccount.percent = other
        if isinstance(self.balance, (int, float)):
            self.balance *= BankAccount.percent
        print(f"Capitalization of the deposit - {BankAccount.percent}")
        return self.balance


Monty = BankAccount("Monty", 12000)
Monty * BankAccount.percent
print(Monty.balance)
Monty.__mul__(1.5)
print(Monty.balance)
Monty * 1.3
print(Monty.balance)
```

```
Capitalization of the deposit - 1.2
14400.0
Capitalization of the deposit - 1.5
21600.0
Capitalization of the deposit - 1.3
28080.0
```

Магические методы для операторов сравнения

- __eq__(self, other) – для равенства ==
- __ne__(self, other) – для неравенства !=
- __lt__(self, other) – для оператора меньше <
- __le__(self, other) – для оператора меньше или равно <=
- __gt__(self, other) – для оператора больше >
- __ge__(self, other) – для оператора больше или равно >=

__eq__(self, other)

```python
class BankAccount:
    def __init__(self, *args):
        self.name = args[0]  # Имя
        self.balance = args[1]  # Баланс


Monty = BankAccount("Monty", 12000)
Bill = BankAccount("Bill", 12000)
print(Monty == Bill)
print(id(Monty), id(Bill))
```

```
False
139733144682448 139733144682208
```

```python
class BankAccount:
    def __init__(self, *args):
        self.name = args[0]  # Имя
        self.balance = args[1]  # Баланс

    def __eq__(self, other):
        r = other if isinstance(other, int) else other.balance
        return f"{self.balance == r} - {self.balance} == {r}"


Monty = BankAccount("Monty", 12000)
Bill = BankAccount("Bill", 12000)
print(Monty == Bill)
print(id(Monty), id(Bill))
Monty.balance = 14200
Bill.balance = 13999
print(Monty == Bill)
```

```
True - 12000 == 12000
140690213863376 140690213863136
False - 14200 == 13999
```

__bool__(self)

bool(123)
bool(-1)
bool(0)
bool("python")

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

p = Point(0, 0)
print(bool(p))
```

```
True
```

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __bool__(self):
        print("__bool__")
        return self.x == self.y
p = Point(0, 0)
print(bool(p))
p.x = 1
print(bool(p))
```

```
__bool__
True
__bool__
False
```

__getitem__(self, item)

- __getitem__(self, item) – получение значения по ключу item.

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = list(marks)

s1 = Student('Сергей', [5, 5, 3, 2, 5])
print(s1.marks[1])
print(s1[1])
```

```
5
Traceback (most recent call last):
  File "/tmp/sessions/0cef3e70470f829f
/main.py", line 8, in <module>
    print(s1[1])
TypeError: 'Student' object is not
subscriptable
```

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = list(marks)

    def __getitem__(self, item):
        if not isinstance(item, int):
            raise TypeError("Индекс должен быть целым числом")
        if 0 <= item < len(self.marks):
            return self.marks[item]
        else:
            raise IndexError("Неверный индекс")


s1 = Student('Сергей', [5, 5, 3, 2, 5])
print(s1.marks[1])
print(s1[1])
```

```
5
5
```

__setitem__()

- __setitem__(self, key, value) – запись значения value по ключу key.

```python
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = list(marks)

    def __setitem__(self, key, value):
        if not isinstance(key, int) or key < 0:
            raise TypeError("Индекс должен быть целым неотрицательным числом")

        if key >= len(self.marks):
            off = key + 1 - len(self.marks)
            self.marks.extend(["_"] * off) # Расширение списка
            self.marks.append(value) # Добавление элемента в список

        self.marks[key] = value


s1 = Student('Иван', [5, 4, 5, 3, 4])
s1[7] = 4
print(s1.marks)
```

```
[5, 4, 5, 3, 4, '_', '_', 4, 4]
```

__delitem__(self, key)

- __delitem__(self, key) – удаление элемента по ключу key.

```python
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = list(marks)

    def __delitem__(self, key):
        if not isinstance(key, int):
            raise TypeError("Индекс должен быть целым числом")

        del self.marks[key]

s1 = Student('Иван', [5, 4, 5, 3, 4])
del s1[2]
print(s1.marks)
```

```
[5, 4, 3, 4]
```

__call__

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

    def __call__(self, *args, **kwargs):
        print("__call__")
        self.__counter += 1
        return self.__counter

c = Counter()
c()
c()
res = c()
print(res)
```

```
__call__
__call__
__call__
3
```