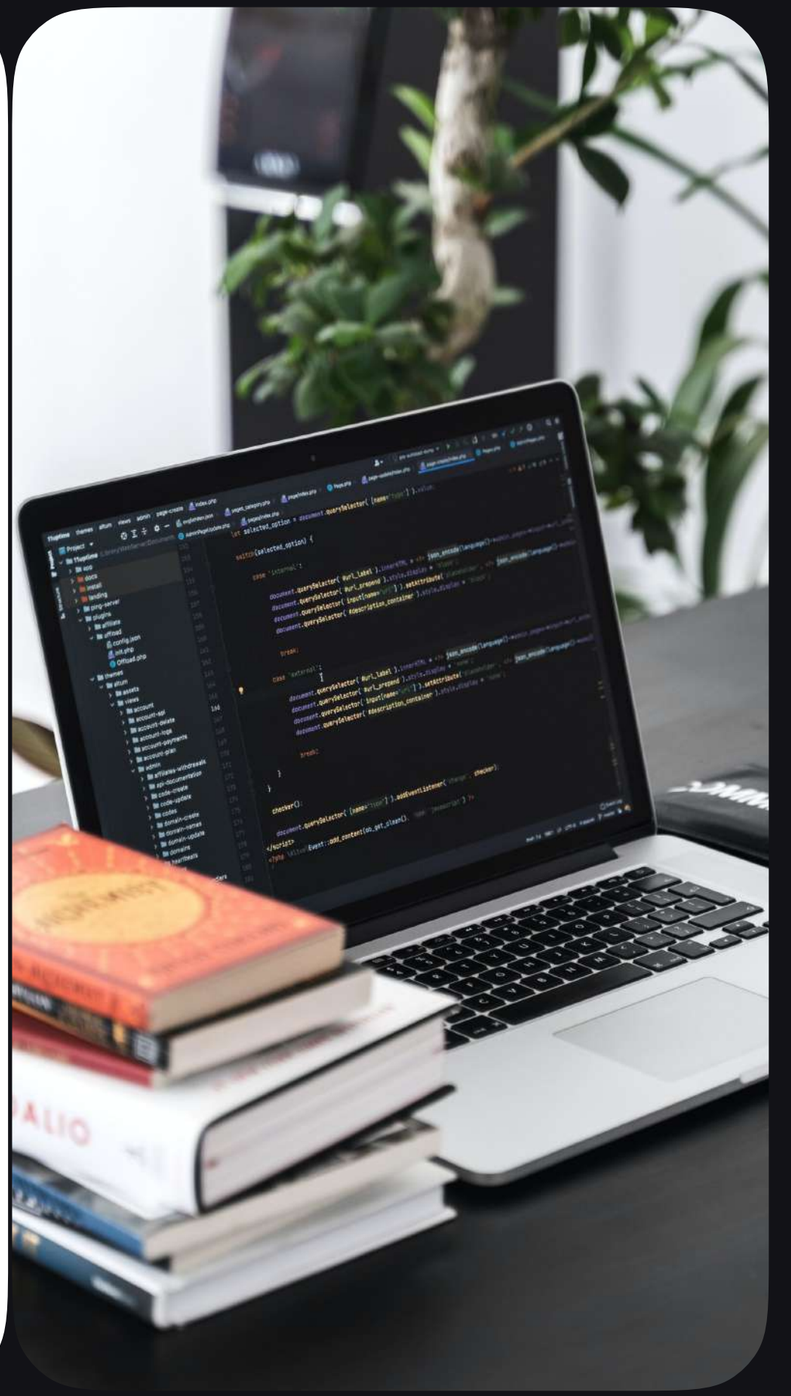
Модуль 2 Занятие 4

# Магические методы. Конструктор класса



Магические методы

Конструктор класса __init__

Финализатор класса __del__

Методы __str__ и __repr__

Другие магические методы

Метод __len__

# На чем мы остановились

```
class Point:
    size = 1
    color = 'black'

    def set_coordinates(self, x, y):
        self.x = x
        self.y = y

    def get_coordinates(self):
        return self.x, self.y

my_point = Point()
my_point.set_coordinates(3, 5)
print(my_point)
print(my_point.get_coordinates())
```

Как можно установить значения атрибутов — координаты, при создании экземпляра класса?

Вывод:

```
<__main__.Point object at 0x00000128C53C7E20>
(3, 5)
```

# Магические методы

Магические методы в Python — это специальные методы, которые вызываются неявно во время вызова функций или выполнения различных операций с объектами. Все магические методы начинаются и заканчиваются двойным знаком подчеркивания, поэтому еще иногда их называют dunder методы (double underscore).

Например, когда мы создаем экземпляр класса, неявно вызывается магический метод __init__, хотя сами мы в это время никакие методы не вызывали.

# Метод __init__

**Метод __init__** — инициализатор (конструктор) класса, вызывается после создания экземпляра класса

```
class Point:
    def __init__(self):
        print('Был вызван конструктор класса')


my_point = Point()
```

Вывод:

```
Был вызван конструктор класса
```

# Пример

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

    def get_coordinates(self):
        return self.x, self.y

my_point = Point(3, 5)
print(my_point.get_coordinates())
```

Вывод: `(3, 5)`

Для точки в пространстве,
с координатами x, y, z придется
менять наш класс

Как это можно исправить?

# Пример

```
class Point:
    def __init__(self, *args):
        self.coords = args

    def get_coordinates(self):
        return self.coords

my_point1 = Point(3)
my_point2 = Point(3, 5)
my_point3 = Point(3, 5, 7)
print(my_point1.get_coordinates())
print(my_point2.get_coordinates())
print(my_point3.get_coordinates())
```

Вывод:

```
(3,)
(3, 5)
(3, 5, 7)
```

# Метод __del__

**__del__** — финализатор (деструктор) класса, вызывается перед удалением экземпляра класса

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

    def __del__(self):
        print(f'Удаление экземпляра {id(self)}')

my_point = Point(1, 2)
print(id(my_point))
my_point = Point(3, 5)
print(id(my_point))
```

Вывод:

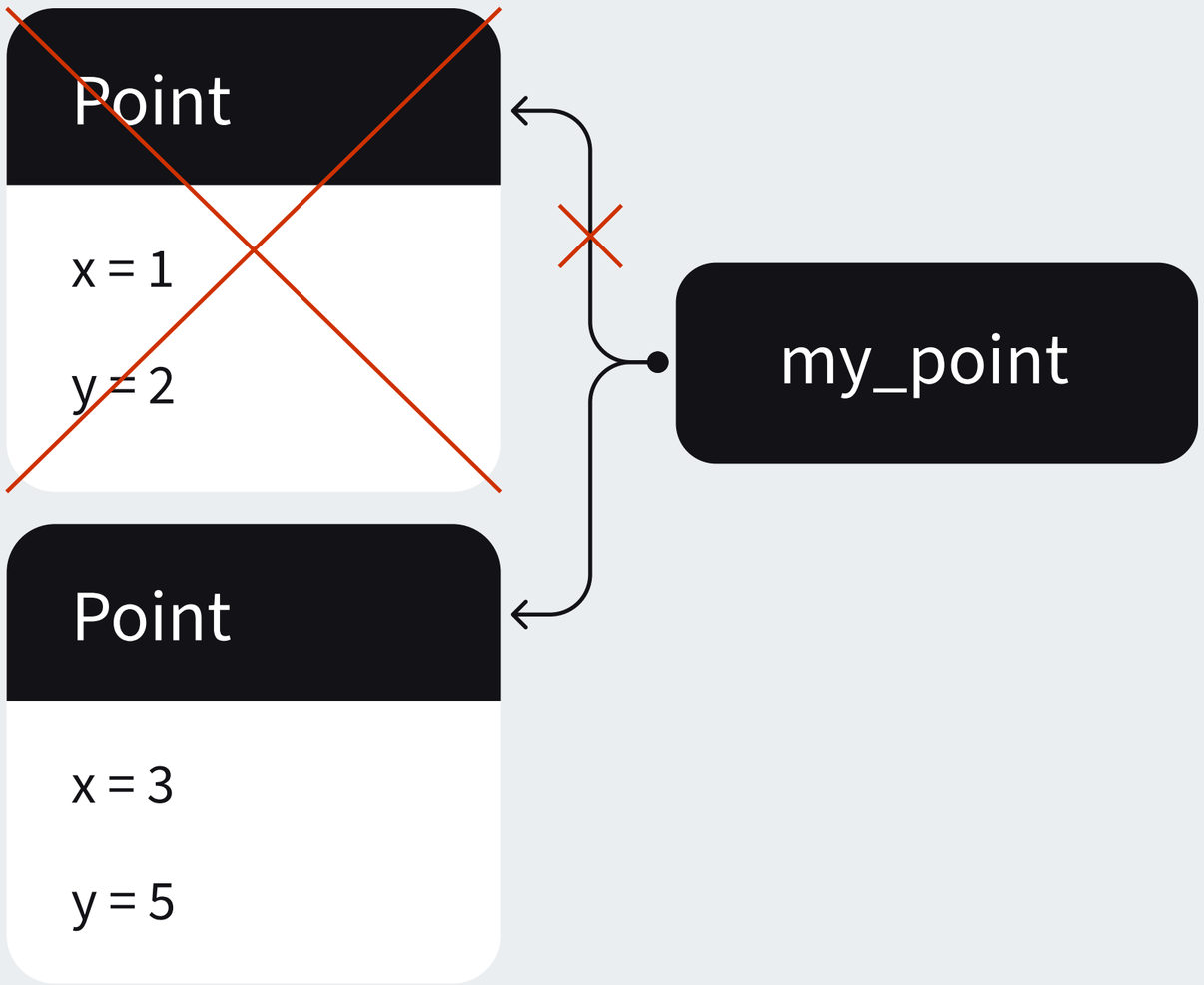
```
2221563080384
Удаление экземпляра 2221563080384
2221563081392
Удаление экземпляра 2221563081392
```

# Как работает удаление объектов?

```
my_point = Point(1, 2)
print(id(my_point))
my_point = Point(3, 5)
print(id(my_point))
```

## Вывод:

```
2221563080384
Удаление экземпляра 2221563080384
2221563081392
Удаление экземпляра 2221563081392
```



# Метод __str__

Метод __str__ вызывается при использовании функций str() и print() и отвечает за строковое представление объекта. Должен возвращать строку.

```
class Point:
    def __str__(self):
        print('Был вызыван метод __str__')
        return 'Точка'


my_point = Point()
str(my_point)
print(my_point)
```

Вывод:

```
Был вызыван метод __str__
Был вызыван метод __str__
Точка
```

# Пример

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

    def __str__(self):
        return f'Точка с координатами {self.x}, {self.y}'

my_point = Point(3, 5)
print(my_point)
```

Вывод:

```
Точка с координатами 3, 5
```

# Пример

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

    def __str__(self):
        return f'Точка с координатами {self.x}, {self.y}'

lst = [Point(3, 5), Point(1, 2)]
print(lst)
```

Почему в этом случае работает по-другому?
Что изменилось?

Вывод:

```
[<__main__.Point object at 0x000001B369E06EF0>,
<__main__.Point object at 0x000001B369E07250>]
```

# Метод __repr__

Метод __repr__ , также как и __str__ отвечает за строковое представление объекта и также возвращает строку.

Но их отличие в их назначении. Если __str__ предназначен для чтения людьми, то __repr__ предназначен для отладки и должен возвращать строку, которая является представлением объекта **в коде программе**.

# Пример

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

    def __str__(self):
        return f'Точка с координатами {self.x}, {self.y}'

    def __repr__(self):
        return f'Point({self.x}, {self.y})'

lst = [Point(3, 5), Point(1, 2)]
print(lst[0])
print(lst)
Точка с координатами 3, 5
[Point(3, 5), Point(1, 2)]
```

Метод __str__ возвращает представление для человека

Метод __repr__ возвращает представление объекта в программе

# Пример

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

    def __str__(self):
        return f'Точка с координатами {self.x}, {self.y}'

    def __repr__(self):
        return f'{self.__class__.__name__}({self.x}, {self.y})'

lst = [Point(3, 5), Point(1, 2)]
print(lst[0])
print(lst)
Точка с координатами 3, 5
[Point(3, 5), Point(1, 2)]
```

Имя класса можно взять из специальной переменной __class__

# Метод __len__

```
class Point:
    def __init__(self, *args):
        self.coords = args

    def __len__(self):
        return len(self.coords)

my_point1 = Point(1, 2)
my_point2 = Point(3, 4, 5)
print(len(my_point1))
print(len(my_point2))
```

Метод __len__ , вызывается при использовании функции len(). Обычно функция len возвращает количество элементов в контейнере, например число элементов в списке

Будем возвращать количество координат точки

Вывод:

```
2
3
```

# Метод __abs__

```
class Point:
    def __init__(self, *args):
        self.coords = args

    def __abs__(self):
        return tuple(map(abs, self.coords))

my_point1 = Point(-1, 2)
my_point2 = Point(3, -4, -5)
print(abs(my_point1))
print(abs(my_point2))
```

Метод __abs__ , вызывается при использовании функции abs()

Определим метод таким образом, чтобы он возвращал все положительные значения координат точки

Вывод:

```
(1, 2)
(3, 4, 5)
```

# __bool__

```
class Point:
    def __init__(self, *args):
        self.coords = args

    def __bool__(self):
        if 0 in self.coords:
            return False
        else:
            return True

my_point1 = Point(0, 2)
my_point2 = Point(3, -4, -5)
print(bool(my_point1))
if my_point2:
    print('Ни одна из координат точки не равно 0')
```

Метод __bool__ , вызывается
при использовании функции bool()
и при проверке условия
в условном операторе

Будем возвращать False если одна
из координат точки будет равна 0
и True в противном случае

## Вывод:

```
Ни одна из координат точки не равно 0
```

# Итоги

- def __init__(self, x, y): … — инициализатор (конструктор) класса, вызывается после создания экземпляра класса
- def __del__(self): …— финализатор класса, вызывается перед удалением объекта
- def __str__(self): … — возвращает строковое представление объекта для человека
- def __repr__(self): … — возвращает строковое представление объекта в программе
- __len__(self):... - вызывается при использовании функции len()
- __abs__(self):... — вызывается при использовании функции abs()
- __bool__(self):... — вызывается при использовании функции bool() и при проверке условия в условном операторе