Классы и Объекты в Python

Создать класс

```
class C:
    pass
```

Методы класса
Методы бывают статическими, классовыми (среднее между статическими и обычными) и уровня класса (будем их называть просто словом метод).
Статический метод создается с декоратором *@staticmethod*,
классовый – с декоратором *@classmethod,* первым аргументом в него передается *cls,*
обычный метод создается без специального декоратора, ему первым аргументом передается *self*

```
class MyClass:
    @staticmethod
    def ex_static_method():
        print("static method")
    @classmethod
    def ex_class_method(cls):
        print("class method")
    def ex_method(self):
        print("method")
```

В *Python* разделяют конструктор класса и метод для инициализации экземпляра класса. Конструктор класса это метод *__new__(cls, *args, **kwargs)* для инициализации экземпляра класса используется метод *__init__(self)*. При этом, как вы могли заметить *__new__* – это классовый метод, а *__init__* таким не является. Метод *__new__* редко переопределяется, чаще используется реализация от базового класса *object*, *__init__* же наоборот является очень удобным способом задать параметры объекта при его создании.

```
class Rectangle:
    def __new__(cls, *args, **kwargs):
        print("Hello from __new__")
        return super().__new__(cls)
    def __init__(self, width, height):
        print("Hello from __init__")
        self.width = width
        self.height = height
```

Для чтения/изменения какого-то атрибута должны использоваться специальные методы, которые называются *getter*/*setter.*
Метод или атрибут, который начинается с нижнего подчеркивания, является скрытым.

```
class Rectangle:
    def __init__(self, width, height):
        self._width = width
        self._height = height
    def get_width(self):
        return self._width
    def set_width(self, w):
        self._width = w
    def get_height(self):
        return self._height
    def set_height(self, h):
        self._height = h
    def area(self):
        return self._width * self._height
```

Если же атрибут или метод начинается с двух подчеркиваний, то тут напрямую вы к нему уже не обратитесь.

```
class Rectangle:
    def __init__(self, width, height):
        self.__width = width
        self.__height = height
    def get_width(self):
        return self.__width
    def set_width(self, w):
        self.__width = w
    def get_height(self):
        return self.__height
    def set_height(self, h):
        self.__height = h
    def area(self):
        return self.__width * self.__height
```

Свойства класса

Свойством называется такой метод класса, работа с которым подобна работе с атрибутом. Для объявления метода свойством необходимо использовать декоратор *@property* **.**

```
class Rectangle:
    def __init__(self, width, height):
        self.__width = width
        self.__height = height
    @property
    def width(self):
        return self.__width
    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError
    @property
    def height(self):
        return self.__height
    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError
    def area(self):
        return self.__width * self.__height
```

Можно не только читать, но и задавать новые значения свойствам.

```
>>> rect.width = 50
>>> rect.width
50
>>> rect.height = 70
>>> rect.height
70
```