# Модификаторы доступа

- если имя атрибута начинается с одного нижнего подчеркивания (`_name`), то он считается **защищенным(только внутри класса и унаследованных классов)**
- если имя атрибута начинается с двух нижних подчеркиваний (`__name`), то он считается **приватным(только внутри класса)**

Но в питоне такой механизм работает на уровне имени атрибута, а не как в других яхыках программирования

```
class Cat:
    def __init__(self, name):
        self._name = name


cat = Cat('Кемаль')
print(cat._name)

cat._name = 'Роджер'
print(cat._name)
```

```
Кемаль
Роджер
```

Атрибут защищен, но мы все равно можем менять его извне

```
class Cat:
    def __init__(self, name):
        self.__name = name


cat = Cat('Кемаль')

print(cat.__name)
```

В этом случае уже будет ошибка - `AttributeError: 'Cat' object has no attribute '__name'`

Но если мы выведем все атрибуты, то увидим, что такой атрибут есть - он переименован

```
class Cat:
    def __init__(self, name):
        self.__name = name


cat = Cat('Кемаль')

print(cat.__dict__)
```

```
{'_Cat__name': 'Кемаль'}
```

Ну а зная имя мы можем менять атрибут как угодно

```
class Cat:
    def __init__(self, name):
        self.__name = name


cat = Cat('Кемаль')

cat._Cat__name = 'Роджер'

print(cat.__dict__)
```

```
{'_Cat__name': 'Роджер'}
```

# Методы акссесоры

## Геттеры

Метод, который возвращает значение атрибута и не изменяет его, называется геттером. Таким образом мы избегаем потенциального изменения приватных атрибутов

```
class Cat:
    def __init__(self, name):
        self._name = name                               # имя кошки

    def get_name(self):                                 # геттер, используется для получения имени
        return self._name


cat = Cat('Кемаль')
```

```
print(cat.get_name())
```

## Сеттеры

Метод, который меняет значение атрибута называется сеттер

```
class Cat:
    def __init__(self, name):
        self._name = name

    def get_name(self):
        return self._name

    def set_name(self, name):                           # сеттер, используется для изменения имени
        if isinstance(name, str) and name.isalpha():    # проверка имени перед заменой
            self._name = name
        else:
            raise ValueError('Некорректное имя')


cat = Cat('Кемаль')
print(cat.get_name())

cat.set_name('Роджер')
print(cat.get_name())
```

## Делиторы

Это метод, который удаляет атрибут из объекта

```
class Cat:
    def __init__(self, name):
        self._name = name

    def get_name(self):
        return self._name

    def set_name(self, name):
        if isinstance(name, str) and name.isalpha():
            self._name = name
        else:
            raise ValueError('Некорректное имя')

    def del_name(self):                                 # делитер, используется для удаления имени
        del self._name


cat = Cat('Кемаль')

cat.del_name()
```

```
print(cat.get_name())  # Error
```