# Управление атрибутами в ваших классах

Когда вы определяете класс на объектно-ориентированном языке программирования, вы, вероятно, в конечном итоге получите некоторые атрибуты экземпляра и класса. Другими словами, вы получите переменные, доступные через экземпляр, класс или даже и то, и другое, в зависимости от языка. Атрибуты представляют или содержат внутреннее состояние данного объекта, к которому вам часто потребуется обращаться и изменять его.

Как правило, у вас есть как минимум два способа управления атрибутом. Либо вы можете получить доступ и изменить атрибут напрямую, либо вы можете использовать методы. Методы — это функции, прикрепленные к данному классу. Они обеспечивают поведение и действия, которые объект может выполнять со своими внутренними данными и атрибутами.

Если вы предоставляете свои атрибуты пользователям вашей программы, они становятся частью общедоступного API( *Application Programming Interface* — «программный интерфейс приложения») ваших классов. Пользователь вашего класса будет получать к ним доступ и изменять их непосредственно в своем коде. И тут могут возникнуть ситуации, что пользователь попытается сохранить недопустимое значение в атрибуты экземпляров вашего класса. Через сеттер(setter) вы можете повлиять на значение, которое сохраняется в ваш атрибут. А геттер(getter) поможет управлять доступом к вашему атрибуту. А для создания геттеров и сеттеров в Python вам может пригодится `property`

`property` - эта функция позволяет вам превращать атрибуты класса в свойства или управляемые атрибуты. Поскольку property() — это встроенная функция, вы можете использовать ее, ничего не импортируя.

> *Примечание. Обычно* `property` *называют встроенной функцией. Однако* `property` *— это класс, предназначенный для работы как функция, а не как обычный класс. Вот почему большинство разработчиков Python называют это функцией. Это также причина, по которой property() не следует соглашению Python по именованию классов.*

С помощью `property` вы можете прикрепить методы получения(getter) и установки(setter) к заданным атрибутам класса. Таким образом, вы можете обрабатывать внутреннюю реализацию этого атрибута, не раскрывая методы получения и установки в вашем API. Вы также можете указать способ обработки удаления атрибута и предоставить соответствующую строку документации для ваших свойств.

```
property(fget=None, fset=None, fdel=None, doc=None)
```

Параметры:

- `fget=None` - функция для получения значения атрибута
- `fset=None` - функция для установки значения атрибута
- `fdel=None` - функция для удаления значения атрибута
- `doc=None` - строка, для строки документации атрибута

Возвращаемое значение `property` — это сам управляемый атрибут. Если вы обращаетесь к управляемому атрибуту, как в obj.attr, тогда Python автоматически вызывает fget(). Если вы присваиваете атрибуту новое значение, как в obj.attr = value, тогда Python вызывает fset(), используя входное значение в качестве аргумента. Наконец, если вы запустите оператор del obj.attr, то Python автоматически вызовет fdel().

> *Примечание. Первые три аргумента функции* `property` *должны принимают функциональные объекты. Вы можете думать об объекте функции как об имени функции без вызывающей пары круглых скобок*

Четвертым аргументом вы можете передать строку документации doc для вашего свойства.

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

    def _get_name(self):
        print("Get name")
        return self._name

    def _set_name(self, value):
        print("Set name")
        self._name = value

    def _del_name(self):
        print("Delete name")
        del self._name

    name = property(
        fget=_get_name,
        fset=_set_name,
        fdel=_del_name,
        doc="The name property."
    )
```

В этом фрагменте создаете класс Person. Инициализатор класса .__init__() принимает имя в качестве аргумента и сохраняет его в защищенном атрибуте с именем ._name. Затем вы определяете три непубличных метода:

1. `_get_name()` возвращает текущее значение `._name`
2. `_set_name()` принимает `value` и присваивает его в атрибут экземпляра `._name`
3. `_del_name()` удаляет у экземпляра атрибут `._name`

```
>>> person = Person('Jack')

>>> person.name
Get name
Jack

>>> person.name= 'Jamal'
Set name
>>> person.name
Get name
Jamal

>>> del person.name
Delete name
>>> person.name
Get name
Traceback (most recent call last):
    ...
AttributeError: 'Person' object has no attribute '_name'

>>> help(person)
Help on Person in module __main__ object:

class Person(builtins.object)
    ...
 |  name
 |      The name property.
```