**Классы и объекты. Атрибуты классов и объектов**

[Смотреть материал на видео](https://www.youtube.com/watch?v=P4CNNo8jWj4&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E" \t "_blank)

Меня зовут Сергей Балакирев и на этом занятии мы с вами узнаем, как в Python определять классы, создавать объекты (экземпляры) этих классов, а также добавлять и удалять их атрибуты (то есть, данные).

Предположим, мы хотим определить класс для хранения координат точек на плоскости. Для начала я его запишу без какого-либо содержимого, только имя класса Point и все:

**class** Point:

**pass**

Здесь оператор pass указывает, что мы в классе ничего не определяем. Также обратите внимание, что  в соответствии со стандартом PEP8 имя класса принято записывать с заглавной буквы. И, конечно же, называть так, чтобы оно отражало суть этого класса. В дальнейшем я буду придерживаться этого правила.

Итак, у нас получилось простейшее определение класса с именем Point. Но в таком виде он не особо полезен. Поэтому я пропишу в нем два атрибута: color – цвет точек; circle – радиус точек:

**class** Point:

    color = 'red'

    circle = 2

Обратите внимание, переменные внутри класса обычно называются атрибутами класса или его свойствами. Я буду в дальнейшем использовать эту терминологию. Теперь в нашем классе есть два атрибута color и circle. Но, как правильно воспринимать эту конструкцию? Фактически, сам класс образует пространство имен, в данном случае с именем Point, в котором находятся две переменные color и circle. И мы можем обращаться к ним, используя синтаксис для пространства имен, например:

Point.color = 'black'

или для считывания значения:

Point.circle

(В консольном режиме увидим значение 2). А чтобы увидеть все атрибуты класса можно обратиться к специальной коллекции \_\_dict\_\_:

Point.\_\_dict\_\_

Здесь отображается множество служебных встроенных атрибутов и среди них есть два наших: color и circle.

Теперь сделаем следующий шаг и создадим экземпляры этого класса. В нашем случае для создания объекта класса Point достаточно после его имени прописать круглые скобки:

a = Point()

Смотрите, справа на панели в Python Console у нас появилась переменная a, через которую доступны два атрибута класса: color и circle.

Давайте создадим еще один объект этого класса:

b = Point()

Появилась переменная b, которая ссылается на новый объект (он расположен по другому адресу) и в этом объекте мы также видим два атрибута класса Point. По аналогии можно создавать произвольное количество экземпляров класса.

С помощью функции type мы можем посмотреть тип данных для переменных a или b:

type(a)

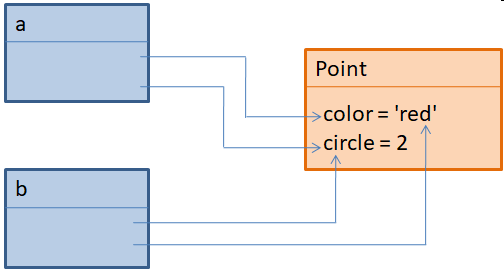
Видим, что это класс Point. Эту принадлежность можно проверить, например, так:

type(a) == Point

или так:

isinstance(a, Point)

То есть, имя класса здесь выступает в качестве типа данных. Но давайте детальнее разберемся, что же у нас в действительности получилось?



Во-первых, объекты a и b образуют свое пространство имен – пространство имен экземпляров класса и, во-вторых, не содержат никаких собственных атрибутов. Свойства color и circle принадлежат непосредственно классу Point и находятся в нем, а объекты a и b лишь имеют ссылки на эти атрибуты класса. Поэтому я не случайно называю их именно **атрибутами класса**, подчеркивая этот факт. То есть, атрибуты класса – общие для всех его экземпляров. И мы можем легко в этом убедиться. Давайте изменим значение свойства circle на 1:

Point.circle = 1

И в обоих объектах это свойство стало равно 1. Мало того, если посмотреть коллекцию \_\_dict\_\_ у объектов:

a.\_\_dict\_\_

то она будет пустой, так как в наших экземплярах отсутствуют какие-либо атрибуты. Но, тем не менее, мы можем через них обращаться к атрибутам класса:

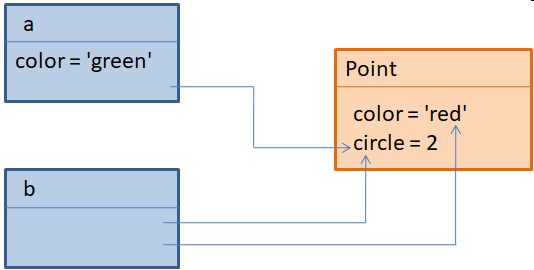
a.color

b.circle

Но, если мы выполним присваивание, например:

a.color = 'green'

То, смотрите, в объекте a свойство color стало 'green', а в b – прежнее. Почему? Дело в том, что мы здесь через переменную a обращаемся к пространству имен уже экземпляра класса и оператор присваивания в Python создает новую переменную, если она отсутствует в текущей локальной области видимости, то есть, создается атрибут color уже непосредственно в объекте a:



Мы можем в этом убедиться, если отобразим коллекцию \_\_dict\_\_ этого объекта:

a.\_\_dict\_\_

То есть, мы с вами создали локальное свойство в объекте a. Этот момент нужно очень хорошо знать и понимать. На этом принципе в Python построено формирование **атрибутов классов** и **локальных атрибутов** их экземпляров.

**Добавление и удаление атрибутов класса**

Кстати, по аналогии, мы можем создавать новые атрибуты и в классе, например, так:

Point.type\_pt = 'disc'

Или то же самое можно сделать с помощью специальной функции:

setattr(Point, 'prop', 1)

Она создает новый атрибут в указанном пространстве имен (в данном случае в классе Point) с заданным значением. Если эту функцию применить к уже существующему атрибуту:

setattr(Point, 'type\_pt', 'square')

то оно будет изменено на новое значение.

Если же мы хотим прочитать какое-либо значение атрибута, то достаточно обратиться к нему. В консольном режиме это выглядит так:

Point.circle

Но, при обращении к несуществующему атрибуту класса, например:

Point.a

возникнет ошибка. Этого можно избежать, если воспользоваться специальной встроенной функцией:

getattr(Point, 'a', False)

Здесь третий аргумент – возвращаемое значение, если атрибут не будет найден. Эту же функцию можно вызвать и с двумя аргументами:

getattr(Point, 'a')

Но тогда также будет сгенерирована ошибка при отсутствии указанного атрибута. Иначе:

getattr(Point, 'color')

она возвратит его значение. То есть, эта функция дает нам больше гибкости при обращении к атрибутам класса. Хотя на практике ей пользуются только в том случае, если есть опасность обращения к несуществующим атрибутам. Обычно, все же, применяют обычный синтаксис:

Point.color

Наконец, мы можем удалять любые атрибуты из класса. Сделать это можно, по крайней мере, двумя способами. Первый – это воспользоваться оператором del:

**del** Point.prop

Если повторить эту команду и попытаться удалить несуществующий атрибут, возникнет ошибка. Поэтому перед удалением рекомендуется проверять существование удаляемого свойства. Делается это с помощью функции hasattr:

hasattr(Point, 'prop')

Она возвращает True, если атрибут найден и False – в противном случае.

Также удалить атрибут можно с помощью функции:

delattr(Point, 'type\_pt')

Она работает аналогично оператору del.

И, обратите внимание, удаление атрибутов выполняется только в текущем пространстве имен. Например, если попытаться удалить свойство color из объекта b:

**del** b.color

то получим ошибку, т.к. в объекте b не своих локальных свойств и удалять здесь в общем то нечего. А вот в объекте a есть свое свойство color, которое мы с вами добавляли:

a.\_\_dict\_\_

и его можно удалить:

**del** a.color

Смотрите, после удаления локального свойства color в объекте a становится доступным атрибут color класса Point с другим значение ‘black’. И это логично, т.к. если свойство не обнаруживается в локальной области, то поиск продолжается в следующей (внешней) области видимости. А это (для объекта a) класс Point. Вот этот момент также следует хорошо понимать при работе с локальными свойствами объектов и атрибутами класса.

**Атрибуты экземпляров классов**

Теперь, когда мы знаем, как создаются атрибуты, вернемся к нашей задаче формирования объектов точек на плоскости. Мы полагаем, что атрибуты color и circle класса Point – это общие данные для всех объектов этого класса. А вот координаты точек должны принадлежать его экземплярам. Поэтому для объектов a и b мы определим локальные свойства x и y:

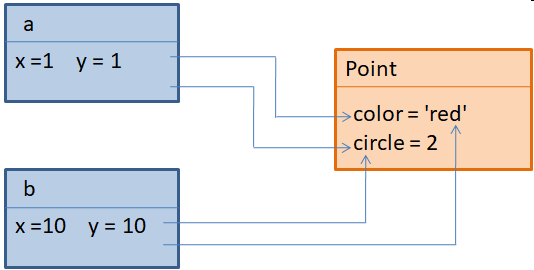
a.x = 1

a.y = 2

b.x = 10

b.y = 20

То есть, свойства x, y будут существовать непосредственно в объектах, но не в самом классе Point:



В результате, каждый объект представляет точку с независимыми координатами на плоскости. А цвет и их размер – общие данные для всех объектов.

В заключение этого занятия отмечу, что в любом классе языка Python мы можем прописывать его описание в виде начальной строки, например, так:

**class** Point:

    "Класс для представления координат точек на плоскости"

    color = 'red'

    circle = 2

В результате, специальная переменная:

Point.\_\_doc\_\_

будет ссылаться на это описание. Обычно, при создании больших программ, в ключевых классах создают такие описания, чтобы в последующем было удобнее возвращаться к ранее написанному коду, корректировать его и использовать, не обращаясь к специальной документации.

**Заключение**

Итак, из этого занятия вы должны себе хорошо представлять, как определяются классы в Python и создаются объекты класса. Что из себя представляют атрибуты класса и атрибуты объектов, как они связаны между собой. Уметь обращаться к этим атрибутам, добавлять, удалять их, а также проверять существование конкретного свойства в классе или объекте класса.