# **Наследование. Атрибуты private и protected**

Курс по Python ООП: <https://stepik.org/a/116336>

[Смотреть материал на видео](https://www.youtube.com/watch?v=zHgPAm-imvY&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E)

Мы продолжаем изучение темы «наследование». На этом занятии мы увидим, как влияет режим доступа private и protected атрибутов при наследовании классов.

Ранее мы с вами об этом уже говорили и в частности отмечали, что:

* \_attribute (с одним подчеркиванием) – режим доступа **protected** (служит для обращения внутри класса и во всех его дочерних классах)
* \_\_attribute (с двумя подчеркиваниями) – режим доступа **private** (служит для обращения только внутри класса).

Давайте посмотрим, как ведут себя атрибуты с этими режимами доступа при наследовании. Возьмем пример из предыдущего занятия с двумя классами:

**class** Geom:     name = 'Geom'       **def** \_\_init\_\_(self, x1, y1, x2, y2):         **print**(f"инициализатор Geom для {self.\_\_class\_\_}")         self.\_\_x1 = x1         self.\_\_y1 = y1         self.\_\_x2 = x2         self.\_\_y2 = y2     **class** Rect(Geom):     **def** \_\_init\_\_(self, x1, y1, x2, y2, fill='red'):         super().\_\_init\_\_(x1, y1, x2, y2)         self.\_\_fill = fill

Здесь мы пытаемся в инициализаторе базового класса Geom сформировать приватные локальные свойства с координатами прямоугольника. Дополнительно в инициализаторе самого класса создается приватное свойство \_\_fill.

Ниже создадим объект класса Rect:

r = Rect(0, 0, 10, 20)

и выведем все его локальные атрибуты в консоль:

**print**(r.\_\_dict\_\_)

После запуска программы увидим следующие строчки:

инициализатор Geom для <class '\_\_main\_\_.Rect'>  
{'\_Geom\_\_x1': 0, '\_Geom\_\_y1': 0, '\_Geom\_\_x2': 10, '\_Geom\_\_y2': 20, '\_Rect\_\_fill': 'red'}

Смотрите, локальные свойства с координатами имеют префикс \_Geom, то есть, префикс того класса, в котором они непосредственно были прописаны. Несмотря на то, что параметр self является ссылкой на объект класса Rect. Это особенность поведения (формирования) приватных атрибутов в базовых классах. У них всегда добавляется префикс именно базового класса, а не класса объекта self. А вот последнее свойство \_\_fill имеет ожидаемый префикс \_Rect, так как оно было создано в классе Rect.

Что из этого следует? Во-первых, мы, конечно же, не можем обратиться в свойствам-координатам в дочернем классе Rect. Если в нем прописать метод get\_coords():

**def** get\_coords(self):         **return** (self.\_\_x1, self.\_\_y1, self.\_\_x2, self.\_\_y2)

а, затем, вызвать через объект класса Rect:

r.get\_coords()

то увидим ошибку AttributeError. Но если перенести этот метод в базовый класс Geom, то все сработает без ошибок, так как приватным свойствам будет добавлен правильный префикс \_Geom.

Возможно, вам кажется это немного запутанным? Но давайте вспомним, а для чего вообще нужны и когда используются приватные атрибуты. Мы говорили, что это закрытые от внешнего вмешательства свойства или методы текущего класса, доступные только внутри этого класса и недоступные из других, в том числе и из дочерних классов. Именно поэтому приватные атрибуты жестко привязываются к текущему классу, в котором они создаются, так как по логике предполагается их использовать только внутри этого класса и больше нигде.

Если же нам нужно определить закрытые атрибуты, доступные в текущем классе и во всех его дочерних классах, то для этого следует использовать метод определения protected – одно нижнее подчеркивание. Поэтому правильнее было бы создавать свойства-координаты в базовом инициализаторе в режиме protected:

**class** Geom:     name = 'Geom'       **def** \_\_init\_\_(self, x1, y1, x2, y2):         **print**(f"инициализатор Geom для {self.\_\_class\_\_}")         self.\_x1 = x1         self.\_y1 = y1         self.\_x2 = x2         self.\_y2 = y2     **class** Rect(Geom):     **def** \_\_init\_\_(self, x1, y1, x2, y2, fill='red'):         super().\_\_init\_\_(x1, y1, x2, y2)         self.\_fill = fill       **def** get\_coords(self):         **return** (self.\_x1, self.\_y1, self.\_x2, self.\_y2)

Тогда никаких проблем с доступом уже не возникает:

r = Rect(0, 0, 10, 20)**print**(r.\_\_dict\_\_) r.get\_coords()

После запуска программы увидим следующие строчки:

инициализатор Geom для <class '\_\_main\_\_.Rect'>  
{'\_x1': 0, '\_y1': 0, '\_x2': 10, '\_y2': 20, '\_fill': 'red'}

Опять же, как я ранее отмечал, режим доступа protected в реальности никак не ограничивает доступ к атрибутам объектов класса или самого класса. Например, мы можем обратиться к координатам напрямую через экземпляр класса:

**print**(r.\_x1)

Никаких ошибок не будет. Нижнее подчеркивание лишь предупреждает (сигнализирует) программиста о защищенном атрибуте, к которому напрямую лучше не обращаться. Этот атрибут был создан для внутренней логики работы алгоритма в классе и не предназначен для обращения извне. В дальнейшем это может привести к проблемам, например, при изменении версии класса, в котором такого атрибута уже не будет, или он будет играть другую роль и т.п.

## **Атрибуты private и protected на уровне класса**

Все также работает и с атрибутами уровня класса. Например, сейчас мы совершенно спокойно можем обратиться к свойству name класса Geom через объект класса Rect:

**print**(r.name)

Добавив одно нижнее подчеркивание, функционал останется прежним, мы лишь отметим, что к этой переменной извне лучше не обращаться:

**print**(r.\_name)

Но, если прописать два подчеркивания, то доступ будет закрыт всюду, кроме самого класса Geom:

**print**(r.\_\_name)

или так:

**class** Rect(Geom):     **def** \_\_init\_\_(self, x1, y1, x2, y2, fill='red'):         super().\_\_init\_\_(x1, y1, x2, y2)         self.\_fill = fill         self.\_name = self.\_\_name

Но в Geom мы можем к ней обращаться:

**class** Geom:     \_\_name = 'Geom'       **def** \_\_init\_\_(self, x1, y1, x2, y2):         **print**(f"инициализатор {self.\_\_name}")         self.\_x1 = x1         self.\_y1 = y1         self.\_x2 = x2         self.\_y2 = y2

Те же ограничения доступа можно накладывать и на методы. Если в базовом классе Geom определить приватный метод, например, для проверки корректности значений координат:

**class** Geom:     ...       **def** \_\_verify\_coord(self, coord):         **return** 0 <= coord <= 100

то он будет доступен только внутри этого класса и вызвать его, скажем, в дочернем классе Rect уже не получится:

**class** Rect(Geom):     **def** \_\_init\_\_(self, x1, y1, x2, y2, fill='red'):         super().\_\_init\_\_(x1, y1, x2, y2)         super().\_\_verify\_coord(x1)         self.\_fill = fill

Кстати, этот пример также показывает, что приватность запрещает переопределение методов в дочерних классах. Если же у метода прописать только одно подчеркивание, то его можно будет вызывать во всех дочерних классах.

Я, надеюсь, вы теперь знаете, как работают режимы доступа атрибутов при наследовании классов и сможете их грамотно применять в своих программах.

Курс по Python ООП: <https://stepik.org/a/116336>