

АССЕМБЛЕР	C#	JAVA	WEB	PYTHON	C	C++	SQL	MONGODB	GO	VB.NET	SWIFT	KOTLIN	DART	PHP	RUST	F#
								НАСТРОЙКИ								

- Глава 1. Введение в Python
- Глава 2. Основы Python
- Глава 3. Объектно-ориентированное программирование
 - Классы и объекты
 - Инкапсуляция, атрибуты и свойства
 - Наследование
 - Переопределение функционала базового класса
 - Атрибуты классов и статические методы
 - Класс object. Строковое представление объекта
- Глава 4. Обработка ошибок и исключений
- Глава 5. Списки, кортежи и словари
- Глава 6. Модули
- Глава 7. Строки
- Глава 8. Pattern matching
- Глава 9. Работа с файлами
- Глава 10. Работа с датами и временем

Инкапсуляция, атрибуты и свойства

Последнее обновление: 26.01.2022



По умолчанию атрибуты в классах являются общедоступными, а это значит, что из любого места программы мы можем получить атрибут объекта и изменить его. Например:

```
1 class Person:
2     def __init__(self, name):
3         self.name = name # устанавливаем имя
4         self.age = 1     # устанавливаем возраст
5
6     def display_info(self):
7         print(f"Имя: {self.name}\tВозраст: {self.age}")
8
9
10 tom = Person("Tom")
11 tom.name = "Человек-паук" # изменяем атрибут name
12 tom.age = -129            # изменяем атрибут age
13 tom.display_info()        # Имя: Человек-паук    Возраст: -129
```

Но в данном случае мы можем, к примеру, присвоить возрасту или имени человека некорректное значение, например, указать отрицательный возраст. Подобное поведение нежелательно, поэтому встает вопрос о контроле за доступом к атрибутам объекта.

С данной проблемой тесно связано понятие инкапсуляции. **Инкапсуляция** является фундаментальной концепцией объектно-ориентированного программирования. Она предотвращает прямой доступ к атрибутам объект из вызывающего кода.

Касательно инкапсуляции непосредственно в языке программирования Python скрыть атрибуты класса можно сделав их приватными или закрытыми и ограничив доступ к ним через специальные методы, которые еще называются **свойствами**.

Изменим выше определенный класс, определив в нем свойства:

```
1 class Person:
2     def __init__(self, name):
3         self.__name = name # устанавливаем имя
4         self.__age = 1     # устанавливаем возраст
5
6     def set_age(self, age):
7         if 1 < age < 110:
8             self.__age = age
9         else:
10            print("Недопустимый возраст")
11
12     def get_age(self):
13         return self.__age
14
15     def get_name(self):
16         return self.__name
17
18     def display_info(self):
19         print(f"Имя: {self.__name}\tВозраст: {self.__age}")
20
21
22 tom = Person("Tom")
23 tom.display_info() # Имя: Tom    Возраст: 1
24 tom.set_age(-3486) # Недопустимый возраст
25 tom.set_age(25)
26 tom.display_info() # Имя: Tom    Возраст: 25
```

Для создания приватного атрибута в начале его наименования ставится двойной прочерк: `self.__name`. К такому атрибуту мы сможем обратиться только из того же класса. Но не сможем обратиться вне этого класса. Например, присвоение значения этому атрибуту ничего не даст:

```
1 tom.__age = 43
```

Потому что в данном случае просто определяется динамически новый атрибут `__age`, но это он не имеет ничего общего с атрибутом `self.__age`.

А попытка получить его значение приведет к ошибке выполнения (если ранее не была определена переменная `__age`):

```
1 print(tom.__age)
```

Однако все же нам может потребоваться устанавливать возраст пользователя из вне. Для этого создаются свойства. Используя одно свойство, мы можем получить значение атрибута:

```
1 def get_age(self):
2     return self.__age
```

Данный метод еще часто называют геттер или аксессор.

Для изменения возраста определено другое свойство:

```
1 def set_age(self, age):
2     if 1 < age < 110:
3         self.__age = age
4     else:
5         print("Недопустимый возраст")
```

Данный метод еще называют сеттер или мьютейтор (mutator). Здесь мы уже можем решить в зависимости от условий, надо ли переустанавливать возраст.

Необязательно создавать для каждого приватного атрибута подобную пару свойств. Так, в примере выше имя человека мы можем установить только из конструктора. А для получения определен метод `get_name`.

Аннотации свойств

Выше мы рассмотрели, как создавать свойства. Но Python имеет также еще один - более элегантный способ определения свойств. Этот способ предполагает использование аннотаций, которые предворяются символом `@`.

Для создания свойства-геттера над свойством ставится аннотация `@property`.

Для создания свойства-сеттера над свойством устанавливается аннотация `имя_свойства.setter`.

Перепишем класс `Person` с использованием аннотаций:

```
1 class Person:
2     def __init__(self, name):
3         self.__name = name # устанавливаем имя
4         self.__age = 1     # устанавливаем возраст
5
6     @property
7     def age(self):
8         return self.__age
```

```
9 @age.setter
10
11 def age(self, age):
12     if 1 < age < 110:
13         self.__age = age
14     else:
15         print("Недопустимый возраст")
16
17 @property
18 def name(self):
19     return self.__name
20
21 def display_info(self):
22     print(f"Имя: {self.__name}\tВозраст: {self.__age}")
23
24
25 tom = Person("Tom")
26
27 tom.display_info() # Имя: Tom  Возраст: 1
28 tom.age = -3486 # Недопустимый возраст
29 print(tom.age) # 1
30 tom.age = 36
31 tom.display_info() # Имя: Tom  Возраст: 36
```

Во-первых, стоит обратить внимание, что свойство-сеттер определяется после свойства-геттера.

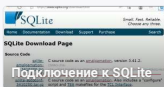
Во-вторых, и сеттер, и геттер называются одинаково - age. И поскольку геттер называется age, то над сеттером устанавливается аннотация @age.setter.

После этого, что к геттеру, что к сеттеру, мы обращаемся через выражение tom.age.

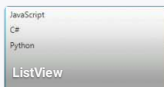
[Назад](#) [Содержание](#) [Вперед](#)



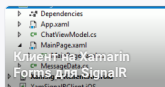
ТАКОЖЕ НА METANIT.COM



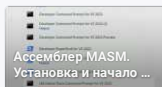
5 месяцев назад · 1 комме...
Библиотека sqlite3, подключение к базе данных SQLite в ...



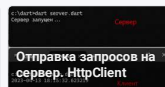
2 месяца назад · 1 коммент...
ListView в JavaFX, создание списков, получение выбранных в ...



22 дня назад · 1 комментар...
Клиентское приложение на Xamarin Forms для SignalR в ASP.NET Core, ...



3 месяца назад · 4 коммент...
Ассемблер MASM. Установка и начало работы, Visual Studio, ...



6 месяцев назад · 1 комме...
Отправка запросов на сервер HttpClient с помощью класса ...



5 м...
Пайтон, зап... при...

36 Комментариев

[Войти](#)



Присоединиться к обсуждению...

войти с помощью

или через DISQUS



Имя



2

• Поделиться

[Лучшие](#)

[Новые](#)

[Старые](#)



Buter

5 лет назад

какое говно, просто супер отвратительный синтаксис, это мне весь код придется засерать какими-то геттерами и сеттерами ненужными, когда мне нужна просто переменная область видимости которой не будет выходить за рамки класс, в C++ просто пишешь private: int a; и используешь её, но только не в пайтоне, нет, кому нужна такая строгая инкапсуляция, давайте добавим везде нижних подчеркиваний и лишнего кода(сеттеры и геттеры), чтобы код стал нечитаемым.



53



13

Ответить • Поделиться



Богдан Товкач

5 лет назад

Ну так и пиши на своем C++. Кто тебе запрещает?



10



1

Ответить • Поделиться



Unknownamed

5 лет назад

Правду говорить - с одной стороны реально хочется плюсы, но с другой стороны - это же змея, зачем ее ругать?



5



0

Ответить • Поделиться



Asdasd Asdasda

4 года назад

не знаю как в c++, но в C# тоже делаются сеттеры и геттеры. И как писать код без них я немного не понимаю.



4



0

Ответить • Поделиться



Fontane

3 года назад

edited

В C# тоже есть проперти. Но вместо

```
@property
def x(self):
    return self.__x

@x.setter
def age(self, x):
    self.__x = x
```

Пишется элегантно

```
int x {get;set;}
```



4



0

Ответить • Поделиться



Борис

3 года назад

edited

Fontane