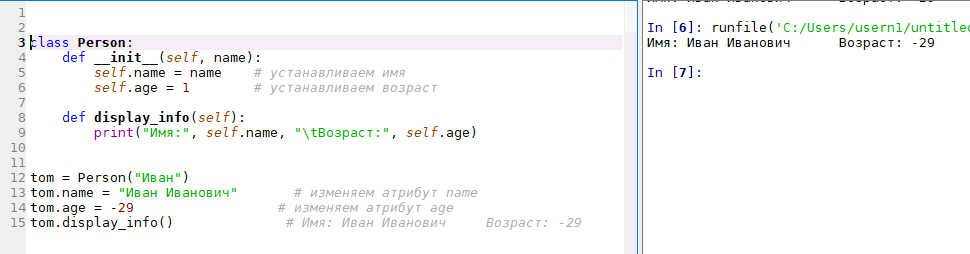
**Инкапсуляция**

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

Например:

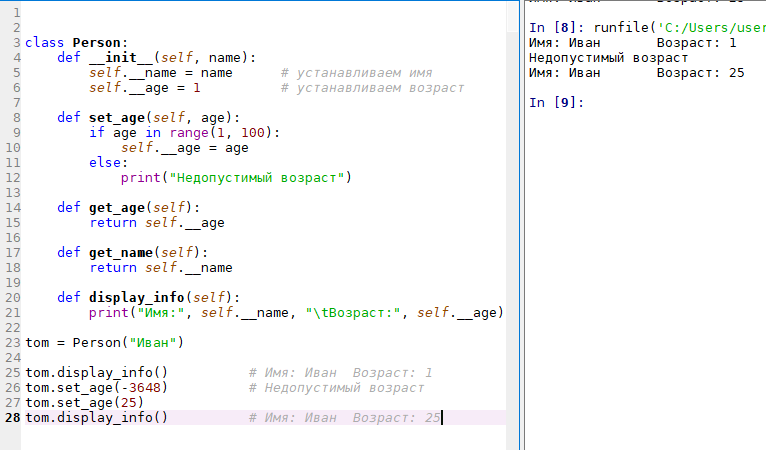


Но в данном случае мы можем, к примеру, присвоить возрасту или имени человека некорректное значение, например, указать отрицательный возраст. Подобное поведение нежелательно, поэтому встает вопрос о контроле за доступом к атрибутам объекта.

С данной проблемой тесно связано понятие инкапсуляции. Инкапсуляция является фундаментальной концепцией объектно-ориентированного программирования. Она предотвращает прямой доступ к атрибутам объекта 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:



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:

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):

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:

def get\_age(self): return self.\_\_age

Данный метод еще часто называют **геттер** или **аксессор**.

Для изменения возраста определено другое свойство:

def set\_age(self, value): if value in range(1, 100): self.\_\_age = value else: print("Недопустимый возраст")

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

Необязательно создавать для каждого приватного атрибута подобную пару свойств. Так, в примере выше имя человека мы можем установить только из конструктора. А для получение определен метод get\_name.

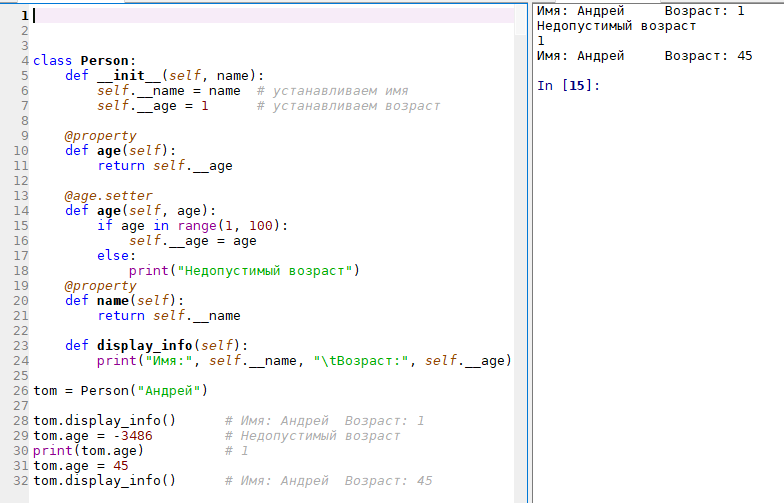
**Аннотации свойств**

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

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

Для создания свойства-сеттера над свойством устанавливается аннотация **имя\_свойства\_геттера.setter**.

Перепишем класс Person с использованием аннотаций:



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**.

**Задание**

Создайте класс BankAccount с приватным атрибутом balance. Реализуйте методы для депозита, снятия и проверки баланса. Используйте методы доступа для работы с приватным атрибутом. Это задание поможет вам понять, как использовать инкапсуляцию для защиты данных и как реализовать методы доступа для работы с приватными атрибутами.