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:

```
class Person:
    def __init__(self, name):
        self.name = name    # устанавливаем имя
        self.age = 1        # устанавливаем возраст

    def display_info(self):
        print("Имя:", self.name, "\tВозраст:", self.age)


tom = Person("Иван")
tom.name = "Иван Иванович"       # изменяем атрибут name
tom.age = -29                    # изменяем атрибут age
tom.display_info()               # Имя: Иван Иванович     Возраст: -29
```

```
In [6]: runfile('C:/Users/usern1/untitle
Имя: Иван Иванович      Возраст: -29

In [7]:
```

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:

```
class Person:
    def __init__(self, name):
        self.__name = name      # устанавливаем имя
        self.__age = 1          # устанавливаем возраст

    def set_age(self, age):
        if age in range(1, 100):
            self.__age = age
        else:
            print("Недопустимый возраст")

    def get_age(self):
        return self.__age

    def get_name(self):
        return self.__name

    def display_info(self):
        print("Имя:", self.__name, "\tВозраст:", self.__age)

tom = Person("Иван")

tom.display_info()          # Имя: Иван  Возраст: 1
tom.set_age(-3648)          # Недопустимый возраст
tom.set_age(25)
tom.display_info()          # Имя: Иван  Возраст: 25
```

```
In [8]: runfile('C:/Users/user
Имя: Иван       Возраст: 1
Недопустимый возраст
Имя: Иван       Возраст: 25

In [9]:
```

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:

```
class Person:
    def __init__(self, name):
        self.__name = name  # устанавливаем имя
        self.__age = 1      # устанавливаем возраст

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, age):
        if age in range(1, 100):
            self.__age = age
        else:
            print("Недопустимый возраст")
    @property
    def name(self):
        return self.__name

    def display_info(self):
        print("Имя:", self.__name, "\tВозраст:", self.__age)

tom = Person("Андрей")

tom.display_info()      # Имя: Андрей  Возраст: 1
tom.age = -3486         # Недопустимый возраст
print(tom.age)          # 1
tom.age = 45
tom.display_info()      # Имя: Андрей  Возраст: 45
```

```
Имя: Андрей     Возраст: 1
Недопустимый возраст
1
Имя: Андрей     Возраст: 45

In [15]:
```

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.