

## ◆ Что такое геттеры и сеттеры?

**Геттер** — это метод, который **возвращает** значение какого-то атрибута.

**Сеттер** — это метод, который **устанавливает** новое значение атрибута.

В Python — это **не обязательно**, но **очень полезно** для:

- **Валидации данных** (например, нельзя установить отрицательный возраст).
  - **Скрытия внутреннего устройства** объекта.
  - **Управляемого доступа** к атрибутам (как будто это обычные атрибуты, но с логикой внутри).
- 

## ◆ **@property** — магия Python

В Python есть специальный декоратор **@property**, который позволяет использовать **геттер и сеттер**, как будто вы работаете с обычным атрибутом.

```
class Person:
    def __init__(self, name, age):
        self._name = name
        self._age = age

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

    @age.setter
    def age(self, value):
        if value < 0:
            raise ValueError("Возраст не может быть отрицательным")
        self._age = value
```

Теперь можно писать:

```
p = Person("Анна", 25)
print(p.age)      # вызывает геттер
p.age = 30        # вызывает сеттер
p.age = -5        # вызывает сеттер → ошибка
```

## ◆ Зачем это нужно?

- **Контроль**: вы можете проверять, какие значения устанавливаются.
- **Совместимость**: вы можете сначала использовать `obj.age = 25`, а потом добавить валидацию — и код, использующий ваш класс, **не сломается**.
- **Инкапсуляция**: пользователь класса не знает, как данные хранятся внутри — он просто работает с ними как с атрибутами.



## Практическое задание (60 минут):

---

Система учёта пользователей с валидацией данных

### ↗ Сценарий

Вы разрабатываете систему учёта пользователей. У каждого пользователя есть:

- Имя (не может быть пустым)
- Возраст (не может быть отрицательным или больше 150)
- Email (должен содержать символ @)

Вам нужно создать класс `User`, в котором **все эти данные защищены**, и доступ к ним осуществляется через **геттеры и сеттеры** с валидацией.

---

### ⌚ Часть 1: Создайте класс `User` (20 минут)

Атрибуты:

- `_name` — защищённый атрибут (хранит имя)
- `_age` — защищённый атрибут (хранит возраст)
- `_email` — защищённый атрибут (хранит email)

Свойства:

#### 1. `name` — только для чтения

- **Геттер:** возвращает `_name`
- **Сеттер:** **не нужен** (имя нельзя менять после создания)

#### 2. `age` — с валидацией

- **Геттер:** возвращает `_age`
- **Сеттер:** проверяет, что `value` — целое число от 0 до 150.  
Если нет — вызывает `ValueError` с понятным сообщением.

#### 3. `email` — с валидацией

- **Геттер:** возвращает `_email`
- **Сеттер:** проверяет, что `value` — строка и содержит @.  
Если нет — вызывает `ValueError`.

Конструктор:

- `__init__(self, name, age, email)`  
Устанавливает `name`, `age`, `email` через сеттеры (чтобы сразу сработала валидация).
-

## ⌚ Часть 2: Добавьте методы (10 минут)

### get\_info()

- Возвращает строку: "Пользователь: {name}, возраст: {age}, email: {email}"

## ⌚ Часть 3: Демонстрация и тестирование (30 минут)

Создайте функцию test\_user\_system():

1. Создайте пользователя:

```
user = User("Анна", 25, "anna@example.com")
```

2. Выведите его данные через get\_info().

3. Попробуйте изменить возраст:

```
user.age = 30
print(f"Новый возраст: {user.age}")
```

4. Попробуйте установить **неправильные данные**:

- user.age = -5 → должно вызвать ошибку
- user.email = "bad-email" → должно вызвать ошибку
- user.name = "Новое имя" → ошибка (сеттера нет)

5. Оберните каждую попытку в try...except и выведите сообщение об ошибке.

❖ Пример вывода:

```
Пользователь: Анна, возраст: 25, email: anna@example.com
Новый возраст: 30
Ошибка при установке возраста: Возраст не может быть отрицательным
Ошибка при установке email: Email должен содержать символ '@'
Ошибка: у атрибута 'name' нет сеттера
```

## 📝 Требования к сдаче

- Один файл .py
- Класс User с тремя свойствами: name, age, email
- Валидация в сеттерах
- Демонстрация работы в функции test\_user\_system()
- Обработка исключений

- Никакого `input()` — только код и вызов функции
- 

## 💡 Подсказки

- Используйте `@property` и `@x.setter` для создания свойств.
  - В `__init__` используйте `self.age = age` — это вызовет сеттер.
  - Для проверки email: `if "@" not in value:`
  - Для проверки возраста: `if not 0 <= value <= 150:`
- 

## ✓ Пример решения (частично)

```
class User:  
    def __init__(self, name, age, email):  
        self.name = name      # вызывает сеттер name (но его нет – только геттер)  
        # Нужно сначала установить защищённые атрибуты напрямую  
        self._name = name  
        self.age = age         # вызывает сеттер age  
        self.email = email     # вызывает сеттер email  
  
    @property  
    def name(self):  
        return self._name  
  
    @property  
    def age(self):  
        return self._age  
  
    @age.setter  
    def age(self, value):  
        if not isinstance(value, int) or not (0 <= value <= 150):  
            raise ValueError("Возраст должен быть целым числом от 0 до 150")  
        self._age = value  
  
    @property  
    def email(self):  
        return self._email  
  
    @email.setter  
    def email(self, value):  
        if "@" not in value:  
            raise ValueError("Email должен содержать символ '@'")  
        self._email = value  
  
    def get_info(self):  
        return f"Пользователь: {self.name}, возраст: {self.age}, email: {self.email}"
```