📚 Справочная информация: Типы методов в Python

В классах Python можно определять три типа методов:

1. **Методы экземпляра** (обычные методы)

- Самый распространённый тип.
- Принимают `self` как первый параметр — ссылку на конкретный объект.
- Имеют доступ ко **всем атрибутам и методам объекта**.
- Вызываются **только от объекта**.

```
class Car:
    def __init__(self, brand):
        self.brand = brand

    def get_info(self):          # метод экземпляра
        return f"Машина: {self.brand}"
```

---

2. **Методы класса** (`@classmethod`)

- Принимают `cls` как первый параметр — ссылку на **класс**, а не объект.
- Имеют доступ к **атрибутам класса**, но **не к атрибутам экземпляра** (если явно не передан объект).
- Часто используются как **альтернативные конструкторы**.
- Вызываются **от класса или от объекта**.

```
class Car:
    total_cars = 0  # атрибут класса

    def __init__(self, brand):
        self.brand = brand
        Car.total_cars += 1

    @classmethod
    def get_total(cls):          # метод класса
        return cls.total_cars

    @classmethod
    def from_string(cls, car_str):  # альтернативный конструктор
        brand = car_str.split("-")[0]
        return cls(brand)
```

---

3. **Статические методы** (`@staticmethod`)

- **Не принимают** ни `self`, ни `cls`.
- Ведут себя как **обычные функции**, но логически связаны с классом.
- **Не имеют доступа** к атрибутам класса или объекта.
- Используются для **вспомогательной логики** (валидация, конвертация и т.д.).
- Вызываются **от класса или от объекта**.

```
class Car:
    @staticmethod
    def is_valid_brand(brand):   # статический метод
        return isinstance(brand, str) and len(brand) > 0
```

🧠 Когда что использовать?

Тип метода	Используйте, когда...
Метод экземпляра	Нужно работать с данными конкретного объекта.
Метод класса	Нужно работать с данными класса или создавать объекты особым способом.
Статический метод	Нужна вспомогательная функция, связанная с классом, но не требующая доступа к его данным.

🧪 Практическое задание (60 минут): Система учёта книг в библиотеке

📌 Сценарий

Вы разрабатываете систему для библиотеки. Каждая книга имеет:

- Название
- Автора
- Год издания
- Уникальный ID (автоматически генерируется)

Вам нужно реализовать класс `Book`, который поддерживает:

- Создание книг обычным способом
- Создание книг из строки формата `"Название - Автор - Год"`
- Валидацию данных
- Учёт общего количества созданных книг
- Получение информации о книге

🎯 Требования к классу `Book`

Атрибуты класса:

- `total_books` — счётчик всех созданных книг (начинается с 0)
- `next_id` — следующий доступный ID (начинается с 1)

Атрибуты экземпляра:

- `book_id` — уникальный ID (целое число)
- `title` — название (строка)
- `author` — автор (строка)
- `year` — год издания (целое число)

💮 Требуемые методы

1. **Метод экземпляра**

- `__init__(self, title, author, year)`
  - Проверяет валидность через статический метод (см. ниже).
  - Генерирует `book_id` на основе `next_id` и увеличивает `next_id`.
  - Увеличивает `total_books` на 1.
- `get_info(self)`
  - Возвращает строку: `"Книга[ID={book_id}]: {title} ({author}, {year})"`

2. **Метод класса (`@classmethod`)**

- `get_total_books(cls)`
  - Возвращает общее количество созданных книг (`total_books`).
- `from_string(cls, book_str)`
  - Принимает строку в формате `"Название - Автор - Год"` (например, `"1984 - Джордж Оруэлл - 1949"`).
  - Разбирает строку и создаёт новый объект `Book`.
  - Если формат неверный — выбрасывает `ValueError`.

3. **Статический метод (`@staticmethod`)**

- `is_valid_year(year)`
  - Проверяет, что `year` — целое число в диапазоне от **1 до 2025**.
  - Возвращает `True`/`False`.
- `is_valid_name(name)`
  - Проверяет, что `name` — непустая строка, без цифр.
  - Возвращает `True`/`False`.

> 💡 В `__init__` используйте эти статические методы для валидации. При ошибке — `raise ValueError`.

💻 Демонстрация работы

Напишите код, который:

1. Создаёт 2 книги обычным способом:

```
book1 = Book("Мастер и Маргарита", "Булгаков М.А.", 1967)
book2 = Book("Война и мир", "Толстой Л.Н.", 1869)
```

2. Создаёт 1 книгу через `from_string`:

```
book3 = Book.from_string("Гарри Поттер - Дж.К. Роулинг - 1997")
```

3. Выводит информацию о каждой книге через `get_info()`.

4. Выводит общее количество книг через `Book.get_total_books()`.

5. Проверяет валидацию:

   - Попытка создать книгу с годом 2026 → должна вызвать ошибку.
   - Попытка создать из строки `"Неверный формат"` → ошибка.

📌 Пример корректного вывода:

```
Книга[ID=1]: Мастер и Маргарита (Булгаков М.А., 1967)
Книга[ID=2]: Война и мир (Толстой Л.Н., 1869)
Книга[ID=3]: Гарри Поттер (Дж.К. Роулинг, 1997)
Всего книг: 3
```

📝 Требования к сдаче

- Один файл `.py` с классом `Book`.
- Демонстрационный код внизу файла (без `input()`).
- Обработка ошибок через `try-except` в демонстрации (чтобы программа не падала).
- Использование всех трёх типов методов.
- Атрибуты класса (`total_books`, `next_id`) должны быть **общими для всех объектов**.

💡 Подсказки

- В `from_string` используйте `.split(" - ")` и проверяйте, что получилось ровно 3 части.
- Преобразуйте год в `int` и проверяйте через `is_valid_year`.
- В `__init__`:

```
if not self.is_valid_name(title) or not self.is_valid_name(author):
    raise ValueError("Некорректное название или автор")
if not self.is_valid_year(year):
    raise ValueError("Некорректный год")
```