Статические поля и методы классов
Те поля и методы, которые мы изучали до этого, были полями и методами экземпляра. Это значит, что для доступа к ним нам нужен экземпляр класса, а при вызове методов, в них неявным параметром передается указатель на экземпляр класса this.

Если нам нужно описать поле или метод, который характеризует сам класс, а не отдельные его экземпляры, к ним можно применить модификатор static. Для полей это означает, что существует только один экземпляр статического поля вне зависимости от того, сколько экземпляров класса существует (даже если не существует ни одного экземпляра класса). Статические методы не получают неявный параметр this, таким образом от свободных функций они отличаются только тем, что отнесены по смыслу к классу.

Для обращения к статическим полям и методам не нужен экземпляр класса и обращение производится по имени класса.

Статические методы
В классе DateTime удобно проверять високосность произвольного года, а не только текущего, поэтому доступ к this для этого метода не нужен. Так как по смыслу он относится к этому классу, поэтому объявим его статическим методом.

```
class DateTime(Time):
    # …
    @staticmethod
    def is_leap(year): # нет self
        return year % 4 == 0 and year % 100 != 0 \
            or year % 400 == 0
    # …
```

Для обращения к статическому методу необходимо указывать имя класса, в котором он объявлен (но не его наследников!).

```
def shift_days(self, days):
    # …
    if DateTime.is_leap(self.year):
        # …
    # …
```

Так как статический метод не получает self, из него напрямую нельзя обращаться к полям и методам экземпляра (не static). При наличии экземпляра, переданного в статический метод явно, с его помощью обращаться к нестатическим полям и методам можно.

Классовые методы

Особой формой статических методов в питоне являются классовые методы. Они как и методы экземпляра получают дополнительный первый параметр. Здесь он обычно называется `cls` и указывает на класс (не объект в отличие от self) у которого он вызван. Это допускает вызов этого метода не только на классе, в котором этот метод объявлен, но и на его наследниках. Также допускается переопределение этого метода в наследниках (подробнее рассмотрим в разделе про полиморфизм).

Одним из применений классовых методов является написание методов, создающих экземпляры класса (позже мы узнаем, что такие методы называются фабричными).

```
class Gear:
    # …
    @classmethod
    def create(cls):
        return cls() # создадим экземпляр
    # …

class SpecialGear(Gear):
    # …

g = Gear.create() # создается Gear
sg = SpecialGear.create() # унаследовали метод create()
              # и он знает, что ему надо создавать SpecialGear
```

Статические поля

Если для экземпляров класса нужно поле, которое будет общим для всех экземпляров (и доступно для всех экземпляров) его можно пометить статическим. Например, максимальное количество патронов в инвентаре игроков может быть помечено статическим. Чтобы пометить поле как статическое, достаточно объявить его в теле класса.

```
class Inventory:
    MAX_AMMO = 300
    # …
```

Аналогично статическим методам, для обращения к статическому полю необходимо указывать имя класса, в котором он объявлен.

```
def add_ammo(self, amount):
    # …
    if self.current_ammo + amount > Inventory.MAX_AMMO:
```

```
        # ...
    # ...
```

Статические свойства

Прямого способа создания статических свойств в Python нет. Нельзя применить к методу одновременно @staticmethod и @property. Этот механизм можно реализовать самостоятельно, однако это выходит за рамки данного курса.