STEP ACADEMY

ЗАНЯТИЕ №11

Основы объектно-ориентированного проектирования

ООП

Объектно-ориентированное программирование (проектирование) — это подход, при котором **программа рассматривается как набор объектов**, взаимодействующих друг с другом. У каждого объекта есть **свойства** и **поведение**. ООП ускоряет написание кода и делает его более читаемым.

Людям проще воспринимать окружающий мир как объекты, которые поддаются определенной **классификации** (например, разделение на живую и неживую природу)

Зачем понадобилось ООП

До ООП в разработке использовался другой подход — **процедурный**. Программа представляется в нем как набор процедур и функций — подпрограмм, которые выполняют определенный блок кода с нужными входящими данными.

Процедурное программирование хорошо подходит для легких программ без сложной структуры. Но если блоки кода большие, а функций сотни, придется редактировать каждую из них, продумывать новую логику. В результате может образоваться много плохо читаемого, перемешанного «спагетти-кода»

Преимущества ООП

В отличие от процедурного, объектно-ориентированный подход позволяет вносить изменения один раз — в объект. Именно он — ключевой элемент программы. Все операции представляются как взаимодействие между объектами. При этом код более понятный, программа проще масштабируется, а её составные части легко переиспользовать в новых проектах.

Процедурный подход: **данные** находятся **отдельно, функции** для работы с ними — **отдельно**;

ООП: **данные и функции** для работы с ними находятся **в одном месте** — в **объекте**.

Свойства и методы

- **Свойства (атрибуты)** — это внутренние переменные объекта, т.е. **данные**.

  Пример: цвет поля или имя пользователя.

- **Методы** — это внутренние **функции** для работы с данными объекта, т.е. которые описывают способы взаимодействия с этим объектом.

  Пример: изменить цвет элемента UI или отправить пользователю email.

Иными словами, **объект = данные + методы**

Пример объекта

Свойства: ?

Методы: ?

Ещё пример

Свойства: ?

Методы: ?

Класс — базовый элемент ООП

Класс — это «**шаблон**» для объекта, который описывает его свойства и поведение. Несколько похожих между собой объектов, например профили разных пользователей, будут иметь одинаковую структуру, а значит, принадлежать к одному классу. **Каждый объект — это экземпляр какого-нибудь класса**.

Объявление простого класса в Python

```
class Имя_класса():
    свойство1 = значение
    свойство2 = значение
    ...
```

Создание экземпляра класса

```
переменная = Имя_класса()
```

Пример простого класса

```
class Kettle():
    material = "steel"
    color  = "red"
    volume = 2.4


my_kettle = Kettle()

print( my_kettle.material )
```

Класс с методами

```
class Имя_класса():
    свойство1 = значение
    свойство2 = значение

    def имя_метода(self, аргументы):
        # тело
        # метода
```

self — специальный аргумент, переменная, указывающая на экземпляр класса. В объявлении метода аргумент **self** нужно указывать обязательно!

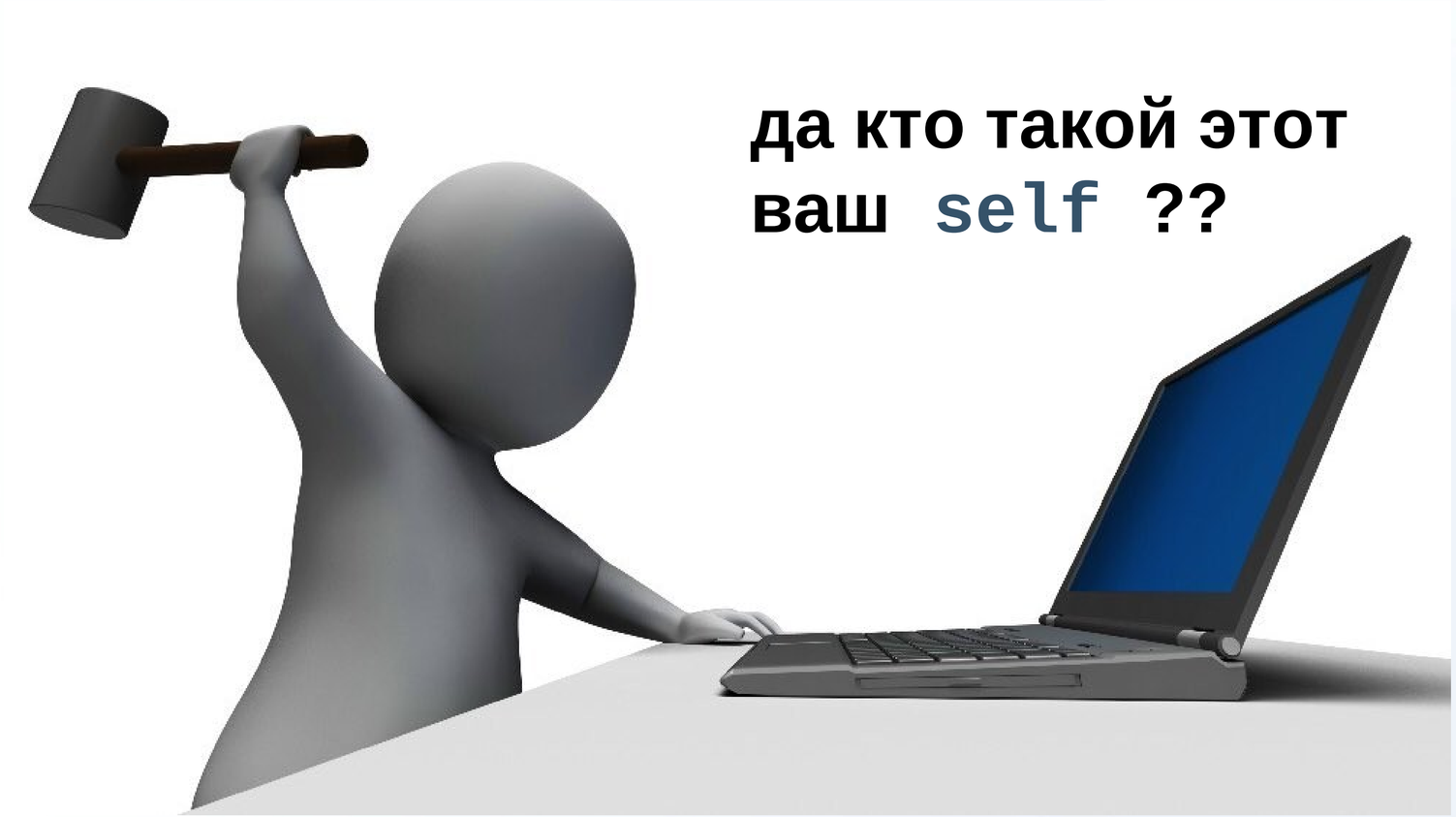
Пример класса с методами

```
class Kettle():
    material = "steel"
    volume = 2.4
    water = 0

    def fill(self, liters):
        self.water += liters
        print("Теперь в чайнике", self.water, "л")

my_kettle = Kettle()
my_kettle.fill(2)
```

self



Что такое self на самом деле

Когда мы вызываем метод нашего объекта

```
my_kettle.fill(2)
```

на самом деле Python автоматически преобразует это в

```
Kettle.fill(my_kettle, 2)
```

Таким образом, первый аргумент метода — всегда является просто ссылкой на тот экземпляр класса, с которым мы в данный момент работаем

Конструктор класса

Конструктор очень полезен! Он позволяет присвоить нужные значения свойствам при создании объекта.

Это специальный метод класса, который неявно вызывается при создании экземпляра класса.

В Python конструктор всегда имеет имя `__init__`.

Кроме того, есть ещё **деструктор** - метод `__del__`, который так же неявно вызывается при удалении объекта. Но в Python его редко требуется явно определять в классе.

Пример класса с конструктором

```
class Kettle():
    material = ""
    color  = ""
    volume = 0

    def __init__(self, material, color, volume):
        self.material = material
        self.color  = color
        self.volume = volume


my_kettle = Kettle("steel", "red", 2.4)
```

Польза конструктора

Конструкторы очень полезны тем, что позволяют легко создавать множество объектов одного класса, но с разными значениями свойств — разные чайники, разных пользователей.

Конструктор также может выполнять другие действия, необходимые для правильной работы нового объекта.

Конец