

Урок 34.

Основы ООП

| | |
|------------------------------|----|
| ООП | 2 |
| ООП vs функциональный подход | 3 |
| Создание класса | 4 |
| Метод <code>__init__</code> | 5 |
| Поля, методы и атрибуты | 7 |
| <code>self</code> | 9 |
| Задания для закрепления | 10 |
| Создание объекта | 11 |
| Доступ к полям объекта | 12 |
| Методы класса | 14 |
| Задания для закрепления | 16 |
| Ответы на задания | 17 |
| Практическая работа | 18 |

ООП



Объектно-ориентированное программирование (ООП) — это подход к написанию программ, в котором основное внимание уделяется объектам и их взаимодействию.

Вместо того чтобы просто писать набор функций, ООП предлагает описывать **сущности реального мира** в виде объектов, которые объединяют **поля** (данные, состояние) и **методы** (поведение).

ООП — это способ **думать о программе не как о последовательности действий**, а как о **взаимодействии объектов**, у каждого из которых есть своя роль и ответственность.



Класс — это **шаблон (чертёж)**, по которому создаются объекты.

Класс описывает:

- Какие **данные** будут у объектов (например, имя, возраст, цена)
- Какие **методы** можно будет с ними выполнять



Пример: класс Book может описывать, что у книги есть title, author, и метод show_description().



Объект — это **конкретный экземпляр класса**, созданный по его шаблону. У него уже есть конкретное состояние и возможность выполнять определённые действия.



Пример: book1 = Book("1984", "Оруэлл") — объект создаётся из класса Book, и у него своё конкретное название и автор.

ООП vs функциональный подход

Существует несколько способов организации кода. Два самых популярных — это **объектно-ориентированное программирование (ООП)** и **функциональный подход**. У каждого есть свои особенности.

| Объектно-ориентированный подход | Функциональный подход |
|--|--|
| Основан на объектах и классах | Основан на функциях как основных строительных блоках |
| Основные единицы — это объекты , которые хранят состояние и имеют поведение | Данные и поведение строго разделены: функции обрабатывают данные, не изменяя их |
| Программа строится как система взаимодействующих объектов | Часто используются чистые функции — одинаковый результат при одинаковом вводе |
| Удобен, когда нужно моделировать сущности реального мира | Предпочтение отдаётся неизменяемым структурам и отсутствию побочных эффектов |
| Хорошо подходит для сложных, расширяемых программ | Хорошо подходит для математических расчётов, анализа данных, обработки потоков информации |

В Python можно использовать **оба подхода**, и часто они дополняют друг друга. ООП хорошо подходит для организации логики и структуры программы, а функциональные элементы — для коротких и чистых операций с данными.

Преимущества ООП

- Позволяет **структурировать** сложные программы
- Упрощает **моделирование реальных сущностей**
- Повышает **переиспользуемость и читаемость** кода
- Облегчает **расширение и сопровождение** программ

Создание класса

Чтобы начать использовать объектно-ориентированный подход, сначала нужно научиться **создавать собственные классы**.

Класс — это **шаблон**, по которому создаются объекты с нужными свойствами и поведением.

Синтаксис:

Для создания класса используется ключевое слово `class`, за которым следует имя класса:

```
Python
# Пустой класс без полей и методов
class ClassName:
    pass
```

Правила именования классов

- Имя класса пишется с **заглавной буквы**
- Если имя состоит из нескольких слов, используется **стиль CamelCase** (каждое слово начинается с заглавной буквы, а между словами не используются подчёркивания)
- Не стоит использовать имена, совпадающие с встроенными типами: `list`, `str`, `object` и т.д.



Примеры хороших имён

Book, UserProfile, ShoppingCart

Метод `__init__`



Магический метод `__init__()` - это инициализатор, который автоматически вызывается при создании объекта.

Он отвечает за **инициализацию** объекта — то есть за установку его начального состояния.

Его часто называют **конструктором**, ведь он участвует в создании объекта. Но на самом деле:

- **Конструктор** — это метод `__new__()`, который создаёт, т.е. конструирует объект (также вызывается автоматически)
- **Инициализатор** — это метод `__init__()` — который инициализирует поля в уже созданном объекте

Синтаксис:

Python

```
class ClassName:
    def __init__(self, arg1, arg2, ...):
        self.attr1 = arg1
        self.attr2 = arg2
        ...
```

- `arg1`, `arg2` — значения, передаваемые при создании объекта
- `attr1`, `attr2` — поля или свойства объекта, сохраняемые через `self`
- `self` — только что созданный объект, которому будут присвоены поля.



Пример: создание класса книги

Python

```
class Book:
    def __init__(self, title, author): # ожидаемые данные для книги
        self.title = title # сохраняем название книги в объекте
        self.author = author # сохраняем имя автора
```

- Метод `__init__()` вызывается автоматически при создании объекта
- `self` — ссылка на сам объект, которому будет присвоено состояние
- `self.title` и `self.author` — это **поля или свойства объекта**

Особенности метода `__init__`

- Называется строго `__init__`
- Вызывается **автоматически** при создании объекта
- Обязан принимать первым аргументом `self` — ссылку на текущий объект
- Через `self` задаются **поля объекта** — переменные, которые будут ему "принадлежать"
- Может отсутствовать, если класс не имеет состояния

Поля, методы и атрибуты



Поля — это переменные, которые хранят состояние объекта.

Они задаются при создании объекта, обычно через метод `__init__`, и сохраняются внутри самого объекта.

Каждый объект имеет свои собственные значения полей.



Пример

Python

```
class Book:
    def __init__(self, title, author):
        self.title = title # поле
        self.author = author # поле
```



Методы — это функции, определённые внутри класса.

Они описывают, **что умеет делать объект**: например, выводить информацию, менять значения полей, выполнять расчёты.

Методы вызываются через объект и имеют доступ к его полям через `self`.



Пример

Python

```
class Book:
    def __init__(self, title, author): # метод
        self.title = title # поле
        self.author = author # поле
```

Всё, что связано с объектом — и поля, и методы — называется **атрибутами**.



Атрибут — это общее название для любой части объекта, доступной через точку: `object.something`.

Замечание: в официальной документации Python нет деления атрибутов на поля и методы, но это удобно для понимания разницы между ними. Кроме того слово "поля" (field) используется во многих обучающих источниках, field встречается в django документации, а также в IDE атрибуты с данными помечаются буквой f - сокращением от field.

Параметр self



self — это обязательный первый параметр в методах класса, который является ссылкой на объект.

self не является зарезервированным словом, но принято использовать именно его.

Зачем нужен self

- Позволяет сохранить данные **внутри объекта**
- Даёт доступ к **полям и методам** объекта
- Отличает **свойства объекта** от обычных переменных внутри метода



Пример: сохраняем данные в объекте

Python

```
class Book:
    def __init__(self, book_title, book_author): # self - созданный объект
        self.title = book_title # присвоение переменной title объекту
        self.author = book_author # присвоение переменной author объекту
```

Без self переменные title и author были бы локальными переменными метода и исчезли бы после выхода из метода.

Задания для закрепления

1. Выберите верные утверждения об объектно-ориентированном программировании:

- a. Каждый объект создаётся по шаблону, называемому классом
- b. Метод класса может использовать `self` для доступа к данным объекта
- c. Метод — это поле объекта
- d. ООП объединяет данные и поведение в единую структуру

[Посмотреть ответ](#)

2. Что делает метод `__init__`?

- a. Создаёт объект
- b. Создаёт ссылку на объект
- c. Создаёт класс
- d. Создаёт поля объекта

[Посмотреть ответ](#)

3. Сопоставьте понятие с описанием:

- 1. класс
 - 2. `self`
 - 3. атрибут
 - 4. метод
-
- a. Ссылка на объект
 - b. Всё, что доступно через точку у объекта
 - c. Функция внутри класса
 - d. Шаблон, по которому создаются объекты

[Посмотреть ответ](#)

Создание объекта

После того как класс определён, можно создавать **объекты** класса. Каждый объект — это **независимый экземпляр**, созданный по шаблону класса, но со своими значениями.

Синтаксис:

Python

```
object_name = ClassName(arg1, arg2, ...)
```

- `ClassName` — имя класса, по шаблону которого создаётся объект
- `arg1`, `arg2` — значения, которые ожидаются в методе `__init__`
- `object_name` — переменная, в которую сохраняется созданный объект



Пример

Python

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

# Создание книг
book1 = Book("1984", "George Orwell")
book2 = Book("Brave New World", "Aldous Huxley")
```

Что происходит при создании объекта

1. Python вызывает метод `__new__()` — создаёт пустой объект
2. Затем вызывает `__init__()` — инициализирует поля объекта (через `self`)
3. Возвращается готовый объект, с которым можно работать

Доступ к полям объекта

После создания объекта можно обратиться к его **полям** (свойствам), чтобы получить или изменить их значения.

Для этого используется символ `.` (точка) — он даёт доступ к атрибутам, хранящимся внутри объекта в его собственной области памяти.

Синтаксис:

```
Python
object_name.attribute
```

- `object_name` — имя переменной, в которой хранится ссылка на объект
- `attribute` — имя поля, к которому обращаемся



Пример: доступ к полям

```
Python
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

book = Book("1984", "George Orwell")

print(book.title)
print(book.author)
```



Пример: изменение значения поля

```
Python
class Book:
    def __init__(self, title, author):
```

```
        self.title = title
        self.author = author

book = Book("1984", "George Orwell")

print(book.title) # старое имя
print(book.author)

book.title = "Animal Farm" # изменение значения поля title
print(book.title) # измененное имя
```

Особенности:

- Каждое поле относится **конкретно к своему объекту** — изменение в одном объекте не влияет на другие

Методы класса



Методы — это функции, определённые внутри класса. Они описывают поведение объекта: что он умеет делать, как работает с данными, которые в нём хранятся.

Синтаксис:

```
Python
class ClassName:
    def method_name(self, ...):
        ...
```

Особенности:

- Метод обязательно принимает первым параметром `self` — это ссылка на текущий объект
- Через `self` метод может обращаться к полям объекта и вызывать другие его методы



Пример

```
Python
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def get_description(self): # self - ссылка на объект, у которого вызван метод
        return f"{self.title} - {self.author}"

    def show_info(self): # self - ссылка на объект, у которого вызван метод
        print(self.get_description()) # вызов другого метода через self

    def make_author_upper(self): # self - ссылка на объект, у которого вызван
```

метод

```
self.author = self.author.upper()
```

```
book = Book("1984", "George Orwell")
```

```
book.show_info() # вызов метода у объекта
```

```
book.make_author_upper() # изменение объекта с помощью метода
```

```
book.show_info() # вывод нового состояния
```

Особенности:

- Методы вызываются через точку: `object.method()`
- При вызове `book.show_info()` Python автоматически подставляет ссылку из переменной `book` в переменную `self`
- Методы могут не только выводить информацию, но и **изменять поля**, выполнять расчёты, возвращать значения и т.д.

Задания для закрепления

1. Сопоставь элементы, связанные с классом, с их описанием:

1. `self.title = title`
2. `def show_info(self):`
3. `book = Book("1984", "Orwell")`
4. `book.title`

- a. Метод класса
- b. Создание объекта
- c. Доступ к полю
- d. Инициализация поля

[Посмотреть ответ](#)



Ответы на задания

| | |
|--|--------------------------------------|
| Задания на закрепление 1 | Вернуться к заданиям |
| 1. Утверждения об ООП | Ответ: a, b, d |
| 2. Метод <code>__init__</code> | Ответ: d |
| 3. Сопоставление понятий с их описаниями | Ответ: 1-d, 2-a, 3-b, 4-с |
| Задания на закрепление 2 | Вернуться к заданиям |
| 1. Сопоставление элементов с их описаниями | Ответ: 1-d, 2-a, 3-b, 4-с |

Практическая работа

1. Класс User

Создайте класс User, который описывает пользователя.

- У каждого пользователя должно быть поля: `username` и `email`, а также счётчик входов `login_count`.
- Добавьте метод `show_info()`, который выводит имя и почту пользователя.
- Добавьте метод `login()`, который приветствует пользователя и фиксирует новый вход.
- Добавьте метод `get_logins()`, возвращающий текущее количество входов.
- Создайте пользователя, выполните несколько входов и выведите информацию.

Пример вывода:

Python

```
-----
Пользователь: alice
Почта: alice@example.com
-----
alice вошёл в систему
alice вошёл в систему
Количество входов: 2
```

Решение:

Python

```
class User:
    def __init__(self, username, email):
        self.username = username
        self.email = email
        self.login_count = 0

    def show_info(self):
        print("-" * 30)
        print("Пользователь:", self.username)
        print("Почта:", self.email)
        print("-" * 30)

    def login(self):
        self.login_count += 1
        print(f"{self.username} вошёл в систему")

    def get_logins(self):
        return self.login_count

user = User("alice", "alice@example.com")
user.show_info()
user.login()
user.login()
print("Количество входов:", user.get_logins())
```

2. Класс Product

Реализуйте класс Product, который описывает товар в магазине.

- Каждый объект должен хранить название (name) и цену (price).
- Добавьте метод `apply_discount()`, который уменьшает цену на заданный процент и выводит информацию о размере примененной скидки.
- Добавьте метод `info()`, который выводит название и текущую цену товара.
- Проверьте работу класса: создайте товар, выведите его данные, примените скидку, затем снова выведите информацию.

Пример вывода:

Python

Название: Молоко

Цена: 120

Применяем скидку 25%

Новая цена: 90.0

Решение:

Python

```
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def apply_discount(self, percent):
        print(f"Применяем скидку {percent}%")
        self.price -= self.price * percent / 100

    def info(self):
        print(f"Название: {self.name}")
        print(f"Цена: {self.price}")

p = Product("Молоко", 120)
p.info()
p.apply_discount(25)
p.info()
```