



Классы в Python

Как создавать собственные типы данных, объекты и методы



Что такое класс?

Класс – это чертёж

Шаблон для создания объектов, как чертёж здания определяет его структуру

Объект – это экземпляр

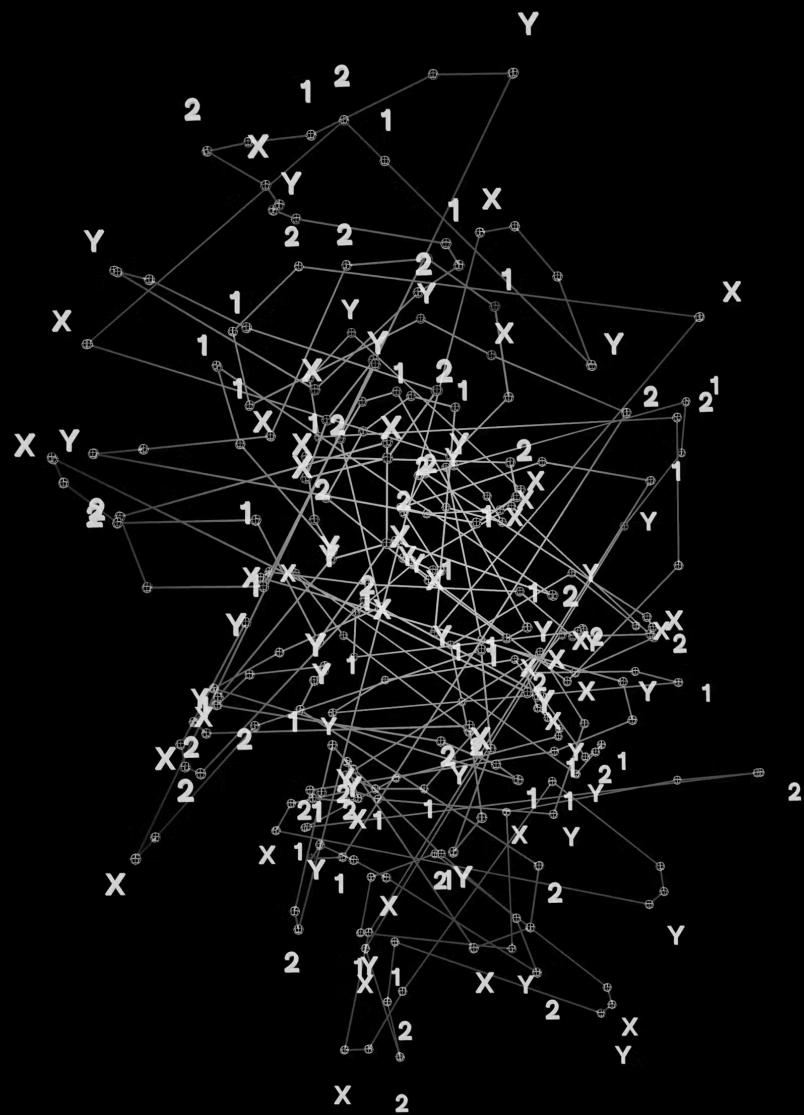
Конкретный пример класса, созданный по этому чертежу

```
class Dog:  
    pass
```

```
my_dog = Dog() # создали объект класса Dog
```

Теперь **my_dog** – это конкретная собака, созданная по шаблону Dog

Зачем нужны классы



Три главные причины

01

Объединяют данные и функции

Держат вместе информацию и действия над ней

02

Избегают дублирования

Переиспользуют код вместо его копирования

03

Моделируют реальность

Описывают студентов, машины, заказы – любые сущности

```
class Student:  
    def __init__(self, name, age):  
        self.name = name # данные  
        self.age = age # данные
```

`__init__`: конструктор класса

1

Автоматический запуск

Метод `__init__` запускается автоматически при создании нового объекта

2

Инициализация

Задаёт начальные значения атрибутов объекта

```
class Car:  
    def __init__(self, brand, year):  
        self.brand = brand # сохраняем марку  
        self.year = year # сохраняем год
```

```
car1 = Car("Toyota", 2020) # __init__ вызывается здесь  
print(car1.brand) # Toyota
```

Конструктор **настраивает** каждый новый объект при его рождении



self: ссылка на сам объект



self – это я

Ссылка на конкретный объект



Доступ к данным

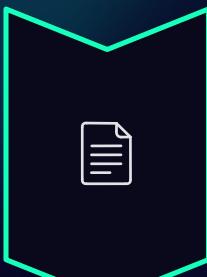
Через self обращаемся к атрибутам и методам

```
class Car:  
    def __init__(self, brand):  
        self.brand = brand  
  
    def show(self):  
        print(f"Марка: {self.brand}")
```

```
car = Car("BMW")  
car.show() # self → car  
# Выведет: Марка: BMW
```

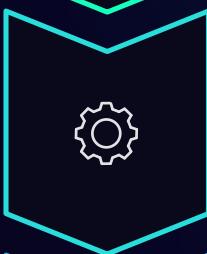
❑ **Важно:** Python автоматически передаёт объект как первый параметр self

Создание и использование объектов



Шаг 1: Объявить класс

Определить структуру с помощью ключевого слова **class**



Шаг 2: Определить методы

Добавить `__init__` и другие функции класса



Шаг 3: Создать объект

Вызвать класс как функцию для создания экземпляра



Шаг 4: Использовать объект

Вызывать методы через точечную нотацию

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

```
    def meow(self):  
        print(f"{self.name} говорит мяу!")
```

```
c = Cat("Барсик")  
c.meow() # Барсик говорит мяу!
```

Магические методы Python

Python автоматически вызывает «магические» методы в определённых ситуациях

Метод	Когда вызывается	Пример использования
__init__	При создании объекта	a = MyClass()
__str__	При выводе print()	print(a)
__call__	При вызове как функции	a()
__len__	При вызове len()	len(a)

```
class Greeter:  
    def __call__(self):  
        print("Привет, я вызвал объект как функцию!")
```

```
g = Greeter()  
g() # автоматически вызовет __call__
```

`__str__`: красивый вывод объектов

Без `__str__`

```
class Book:  
    def __init__(self, title):  
        self.title = title  
  
b = Book("Python для начинающих")  
print(b)
```

✗ Результат:

```
<Book object at 0x7f8b3c4d5e80>
```

Непонятная информация о памяти

С `__str__`

```
class Book:  
    def __init__(self, title):  
        self.title = title  
  
    def __str__(self):  
        return f"Книга: {self.title}"
```

```
b = Book("Python для начинающих")  
print(b)
```

✓ Результат:

```
Книга: Python для начинающих
```

Понятная читаемая информация

-  Метод `__str__` делает вывод объектов человекочитаемым

Наследование: переиспользуем код



Базовый класс

Родительский класс с общей функциональностью



Дочерний класс

Наследует методы и может их переопределить



Специализация

Добавляет или изменяет поведение под свои
нужды

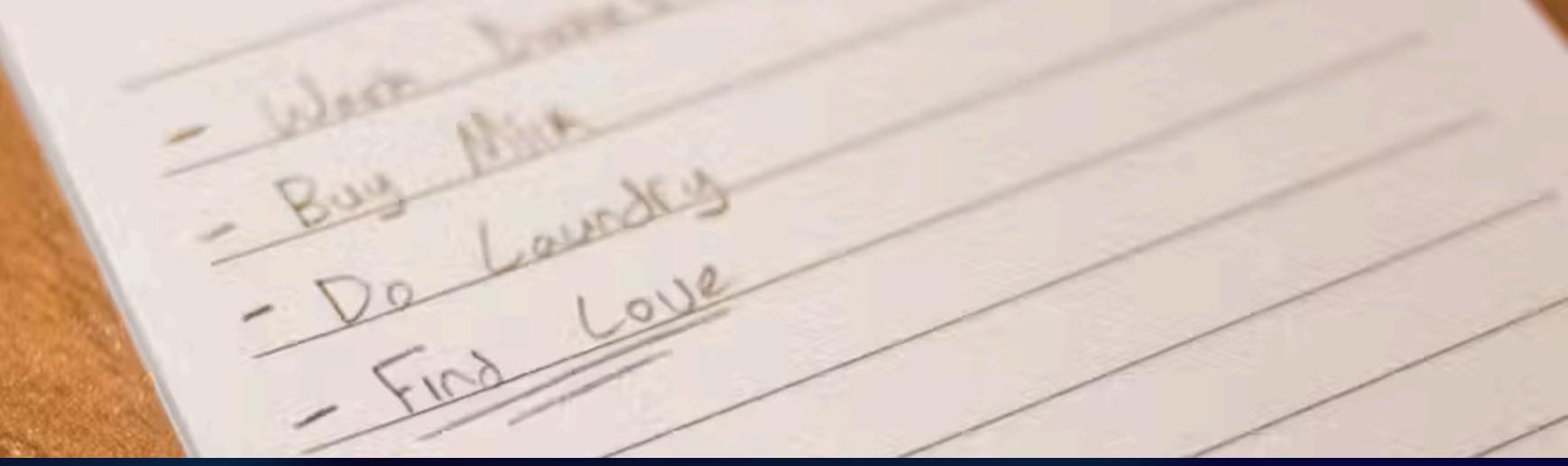
```
class Animal:  
    def speak(self):  
        print("Some sound")
```

```
class Dog(Animal): # Dog наследует Animal  
    def speak(self): # переопределяем метод  
        print("Гав-гав!")
```

```
Dog().speak() # Гав-гав!
```

Наследование позволяет избежать копирования
одинакового кода в разных классах





Подводим итоги



Класс = шаблон

Объект = конкретный экземпляр этого шаблона



`_init_` и `self`

Сердце класса – конструктор и ссылка на объект



Магические методы

`__str__`, `__call__` и другие делают код удобнее



Когда использовать

Если логика и данные связаны – класс идеален

«Классы – это способ организовать код так, чтобы он отражал структуру реального мира»

Совет: Начните с простых классов и постепенно усложняйте. Практика – лучший учитель! 🐍