Шпаргалка: классы и объекты

Кратко

Что	Как	О чём помнить

Создать класс	`class <имя класса>():`
	`<переменная> = <значение переменной>`
	
	Пример
	`class Dog():`
	`tail = 1`
	`paws = 4`
		Для имени класса есть несколько правил.
		1) С большой буквы.
		2) Только цифры и латинские буквы.
		3) Имя начинается не с цифры.
		4) Нет пробелов

Создать объект	`имя объекта = имя класса()`
	
	Пример
	`class Dog():`
	`tail = 1`
	`paws = 4`
	`dog_1 = Dog()`	

Создать объект, в котором можно поменять переменные	Нужен метод `__init__()`
	`def __init__(self,<переменная>):`
	`self.<переменная> = <значение переменной>`
	
	Пример
	`class Dog():`
	`tail = 1`
	`paws = 4`
	
	`def __init__(self, name, color):`
	`self.dog_name = name`
	`self.dog_color = color`	

Создать метод	`def <имя метода>(self):`
	`  <тело метода>`
	
	Пример
	`def bark(self):`
	`  print('гав-гав')`
		Имя метода — правила те же, что и для имени функции

Вызвать метод	`<имя объекта>.<имя метода>`
	
	Пример
	`dog1.bark()` — вызвали метод `bark()` у объекта `dog1`	

Подробно

Что такое класс и объект

Эти сущности не получится объяснить одну без другой: они тесно взаимосвязаны. Можно сказать, **объект** — **это экземпляр класса**.

Пример. Представь, что у тебя есть фабрика по производству собачек. Чтобы создать животное, используют шаблон с определенными базовыми параметрами собачки: четыре лапы, одно туловище, одна голова и один хвост.

Сначала создается шаблон, а уже потом с помощью него создают собачек. Собачки могут быть разными: одна белая, вторая рыжая, третья пятнистая. Но у всех будут одинаковые базовые параметры: четыре лапы, одно туловище, одна голова и один хвост — как в шаблоне.

> 🐶 Шаблон собачки с четырьмя лапами и одним хвостом — это **класс** в Python. Животное, которое получилось по этому шаблону, — **объект (экземпляр класса)**.

Можно сказать, что класс сам по себе ничего не делает: он только описывает то, что можно создать с помощью него.

Многие сущности в Python описываются с помощью объектов. Допустим, тебе нужно написать программу, в которой собачка гавкает. Понадобится объект — собачка.

Но создать собаку просто так нельзя. Сначала нужно создать класс: он будет шаблоном. И уже на его основе ты сможешь получить реальную собачку, которая будет гавкать.

Миновать этап создания класса не получится. Это как нельзя прийти и сразу начать строить здание. Сначала нужно нарисовать чертёж: какая у дома планировка, из какого он будет материала. А потом уже строить — опираясь на план. Так и в программировании: сначала нужно создать «шаблон» для объекта — класс. И уже потом сам объект.

Как создать класс

Итак, нужно написать программу, в которой собака гавкает.

Сперва предстоит создать класс «Собака». Это «чертёж» для объекта: в нём ты укажешь характеристики, которые будут у всех собачек.

Чтобы создать класс, нужно:

- написать команду `class`;
- придумать классу имя — например, `Dog`.

Как называть класс

Для имени класса есть несколько правил.

С большой буквы. Имя класса принято начинать с большой буквы, чтобы выделить его среди остальных переменных: `Dog`. Если назвать с маленькой, ошибки не будет. Но так делать не принято.

Только цифры и латинские буквы. Например, `Dog` или `Dog2`. Иначе будет ошибка.

Имя начинается не с цифры. Например, `2Dog` — не подойдёт, программа собьётся.

Нет пробелов. Если в имени несколько слов, первая буква каждого слова — заглавная. Например, `MyDog`. Такой стиль называется **CamelCase** — «верблюжий». А всё потому, что слова напоминают верблюда с горбами.

Переменные класса

Класс — это шаблон для объекта. В нём нужно задать определенные параметры. Например, чтобы у всех собачек был один хвост и четыре лапы.

Тут пригодятся **переменные** класса.

Переменные класса хранят в себе параметры, которые будут потом у всех объектов.

Например, для собаки понадобятся лапы — `paws`, а ещё хвост — `tail`:

```
class Dog():
       tail = 1
       paws = 4
```

Теперь у всех собачек будут хвостик и лапы.

А если сделать вот так, всех собачек будет звать Бобик:

```
class Dog():
       tail = 1
       paws = 4
       name = 'Бобик'
```

Если нужно, чтобы в каком-то месте программы у собаки поменялось имя, параметр можно будет изменить. Нужно обратиться к переменной так: `имя класса.имя переменной` — например, `Dog.name`.

Поменяем имя «Бобик» на имя «Шарик»:

```
print(Dog.name) #вывод в консоль "Бобик"

Dog.name = 'Шарик' #переназначаем переменную класса

print(Dog.name) #вывод в консоль "Шарик"
```

Но учти, что если собак будет несколько, имя поменяется у всех сразу.

Как создать объект

Вот есть класс `Dog` с переменными:

```
class Dog():
       tail = 1
       paws = 4
```

Создать объект можно по такой схеме: `имя объекта = имя класса ()`. Например, чтобы создать собаку: `dog = Dog()`.

Объектов-собак можно создать много:

```
class Dog():
       tail = 1
       paws = 4

dog_1 = Dog() # создали объекты
dog_2 = Dog()
dog_3 = Dog()
```

Проблема в том, что все эти объекты будут одинаковые. У всех собак будет четыре лапы и хвост, но вот задать им разный цвет или разные имена мы не сможем.

Метод `__init__`

Чтобы можно было создавать разных собак, нужен **конструктор** — метод `init`**.**

В Python у классов есть особый встроенный метод `__init__`**.** Он называется «конструктор», потому что помогает построить объект: собрать его из разных параметров.

Чтобы вызвать метод `__init__`, тоже нужно ключевое слово `def`:

```
class Dog(): #создали класс
    tail = 1
    paws = 4

    def __init__(...): #вызвали метод
```

```
        #тут будет ещё код
```

Метод ещё не дописан: понадобится обязательный **аргумент** `self`.

Аргумент `self` позволяет обращаться к объекту, чтобы присваивать ему индивидуальные значения. Например, рыжий цвет или имя «Шарик». Это такой вспомогательный аргумент: он сам не содержит значений, а только передаёт параметры объектам.

```
def __init__(self, ..): # обязательный аргумент self
      #тут будет ещё код
```

Аргумент `self` нужно поставить первым. А уже потом перечислить другие свойства, которые хочешь задать для объектов. Например, имя и цвет — `name`, `color`.

```
def __init__(self, name, color): # обязательный аргумент self, а потом name и color
      #тут будет ещё код
```

Вообще этот параметр можно назвать как угодно, но чаще всего его называют именно так — `self`. Главное, чтобы он шёл первым.

Эти значения — цвет и имя — ты задашь каждой собачке отдельно, когда будешь создавать объект.

Нужно указать, в каких атрибутах они будут храниться. Атрибут объявляют через точку после `self`. Например, `self.dog_name = name` — «у объекта будет атрибут `dog_name`, а значением этого атрибута будет `name`».

Получился конструктор, который создаст новую собачку с определёнными параметрами — именем и цветом. При этом у всех собачек будет четыре лапы и хвост:

```
class Dog():  #объявили класс
    tail = 1
    paws = 4

    def __init__(self, name, color):
        self.dog_name = name
        self.dog_color = color
```

📎 Итак, **атрибуты** — это переменные, в которых хранят уникальные свойства объектов. Их инициализируют в конструкторе класса.

Атрибуты ещё называют полями. Это одно и то же. Чтобы не запутаться, мы везде говорим **«**атрибуты**»**.

Теперь можно создавать сколько угодно разных собачек. Если хочешь сделать их пятнистыми или рыжими, пиши это в скобках:

```
class Dog():
    tail = 1
    paws = 4

    def __init__(self, name, color):
        self.dog_name = name
        self.dog_color = color


dog_1 = Dog('Барбос','Чёрный')
dog_2 = Dog('Шарик', 'Коричневый')
dog_3 = Dog('Тедди', 'Рыжий')
```

Можно проверить, какие параметры у объектов. Например, что у переменной `dog_1` имя — «Барбос» .

С помощью `print()` можно вывести значения параметров в консоль. Укажи имя объекта и параметр через точку:

```
print(dog_1.dog_name) # выведет «Барбос»
print(dog_1.paws) # выведет 4: это одинаково для всех объектов
```

Итак, вот есть класс `Dog`. У всех собак есть параметры — хвост и четыре лапы. Им можно задать имя и цвет.

Можно ещё сделать так, чтобы собаки умели гавкать. Для этого нужно объявить **метод**.

Тебе уже встречались методы, когда нужно было работать со списками. Например, метод `remove()` удалял элемент. Настало время поговорить подробнее, что такое метод и чем он отличается от функции.

Что такое метод

Вспомни, что такое функция. Это набор действий, у которого есть имя. Например, функция `print()` печатает то, что ты укажешь в скобках.

> 📎 Метод — это функция, которая привязана к объекту.

Тонкостей тут предостаточно. Иногда одно и то же действие можно написать как функцией, так и методом. Как тут лучше — решает сам программист.

Как объявить метод

Всё похоже на функцию. Понадобится:

- Ключевое слово `def`.
- Имя метода — правила те же, что и для имени функции.
- Обязательный аргумент `self` — это отличие от функции. Он нужен, чтобы получать доступ к атрибутам объекта и вызывать метод для объектов.
- Тело метода — что метод будет делать.

Создадим метод `bark()`. Он выводит «гав-гав».

```
#объвляем метод, чтобы выводить «гав-гав»
#если не написать self, потом не сможешь вызвать метод для объекта dog
def bark(self):
    print('гав-гав') #тело метода
```

Вот как будет целиком — в классе Dog:

```
class Dog():
    tail = 1
    paws = 4

    def __init__(self, name, color):
        self.dog_name = name
        self.dog_color = color

    def bark(self):  #объявляем метод, чтобы выводить «гав-гав»
        print('гав-гав')
```

Обычно методу нужно обращаться к атрибутам объекта. Допустим, нужно вывести «гав-гав, меня зовут (имя собачки)». Методу нужно получить имя собаки.

Для этого нужно просто указать имя атрибута:

```
class Dog():
    tail = 1
    paws = 4

    def __init__(self, name, color):
        self.dog_name = name
        self.dog_color = color

    def bark(self):
        print('гав-гав, меня зовут ', self.dog_name) #указываем атрибут dog_name, обращаемся к нему через self
```

Пока что метод создан, но собачка не гавкает. Чтобы она загавкала, метод нужно вызвать. Тут всё как с функциями.

Как вызвать метод

Нужно:

- Написать имя объекта, для которого вызывается метод. Например, `lst`.
- Поставить точку и написать имя метода. Например, `.insert`.
- Поставить скобки. В них указать параметры, если они есть.

Можно создать объект-собачку и вызвать для него метод:

```
class Dog():
    tail = 1
    paws = 4

    def __init__(self, name, color):
        self.dog_name = name
        self.dog_color = color

    def bark(self):
        print('гав-гав, меня зовут ', self.dog_name)

dog1 = Dog ('Шарик', 'рыжий') #создали объект dog1
dog1.bark() #вызвали метод у объекта

#программа выведет «гав-гав, меня зовут Шарик»
```

Встроенные методы

Методы можно разделить на два типа. Одни создаёт сам программист — например, как `bark()` из прошлого урока. А другие методы **встроенные** — они уже есть в Python. Их можно вызывать в любой программе.

Один из таких встроенных методов — `__dict__`. Он помогает представить объект в удобном виде.

Объект очень похож на словарь. Он тоже содержит элементы в виде `ключ:значение`.

Например, есть объект `dog_1`. У него есть свойства:

- количество лап — `paws`;
- количество хвостов — `tail`;
- цвет — `color`;
- имя — `name`.

Если попытаться вывести все параметры объекта через функцию `print()`, в консоли будет отображаться не объект с парами `ключ : значение`, а имя его класса и адрес в памяти.

```
print(barbos)    #выводим в консоль объект, сохраненный в переменной "barbos"

>>  <__main__.Dog object at 0x0000019F19B1BB20>    #ссылка на экземпляр класса
```

Чтобы отобразить объект в виде читаемой структуры словаря, нужно вызвать встроенный метод `__dict__`:

```
print(barbos.__dict__)    #обращаемся к встроенному методу "__dict__" экземпляра класса

>> {'dog_name': 'Барбос', 'dog_age': 3, 'dog_color': 'Черный'}  #вывод в консоль свойств объекта в виде словаря
```

Остальные встроенные методы можно посмотреть в документации Python.

Как взаимодействуют классы

Чаще всего программа состоит из нескольких классов. Они могут взаимодействовать: один класс обращается к переменным и методам другого.

Вот класс `Dog`:

```
class Dog():
    tail = 1 #переменные класса одинаковые у всех объектов
    paws = 4

    def __init__(self, name, color): #метод __init__, задаёт имя и цвет
        self.dog_name = name
        self.dog_color = color

    def bark(self):  #метод, с помощью которого собака гавкает
        print('гав-гав')
```

И есть второй класс — `Human`. Человек может завести собачку: за это отвечает метод `adopt_dog()`.

Вот что он делает: строчка `self.my_dog = dog` создаёт у человека атрибут `my_dog` и кладёт в него объект класса `Dog`. Так у человека появляется атрибут — и в нём лежит целый объект-собачка:

```
class Human():
    def __init__(self,name): #у человека есть имя
        self.name = name

    def adopt_dog(self, dog): #метод заводит собачку
        print('У меня есть собачка')
        self.my_dog = dog #добавили собачку как атрибут человеку
```

Допустим, когда человек заводит собачку, он просит её погавкать — `ask_dog_to_bark()`. Но способность гавкать лежит в классе `Dog`. Поэтому нужно обратиться из класса `Human` в класс `Dog` и «позвать» оттуда метод `bark()`:

```
class Human():
    def __init__(name):
        self.name = name

    def adopt_dog(self, dog):
        print('У меня есть собачка')
        self.my_dog = dog
```

```
    def ask_dog_to_bark(self):#метод просит собачку погавкать
        self.my_dog.bark() #метод вызывает у атрибута my_dog метод bark()
```

Обратились к методу чужого класса по такой схеме:

```
экземпляр_класса.имя_метода_другого_класса()
self.dog.bark()
```

Теперь человек может завести собачку, и она тут же погавкает:

```
class Dog():
    tail = 1 #переменные класса одинаковые у всех объектов
    paws = 4

    def __init__(self, name, color): #метод __init__, задаёт имя и цвет
        self.dog_name = name
        self.dog_color = color

    def bark(self):  #метод, с помощью которого собака гавкает
        print('гав-гав')

class Human():
    def __init__(self):
        self.name = name

    def adopt_dog(self, dog): #метод, который заводит собачку
        print('У меня есть собачка')
        self.my_dog = dog

    def ask_dog_to_bark(self): #метод просит собачку погавкать
        self.my_dog.bark()



human1 = Human('Саша') #создали человека
dog1 = Dog('Шарик', 'рыжий') #создали собачку
human1.adopt_dog(dog1) #человек завёл собачку dog1, поэтому её передали как аргумент
human1.ask_dog_to_bark() #собака погавкала
```