

**Моя IT Школа**

сертифицированные IT курсы

**ООП**

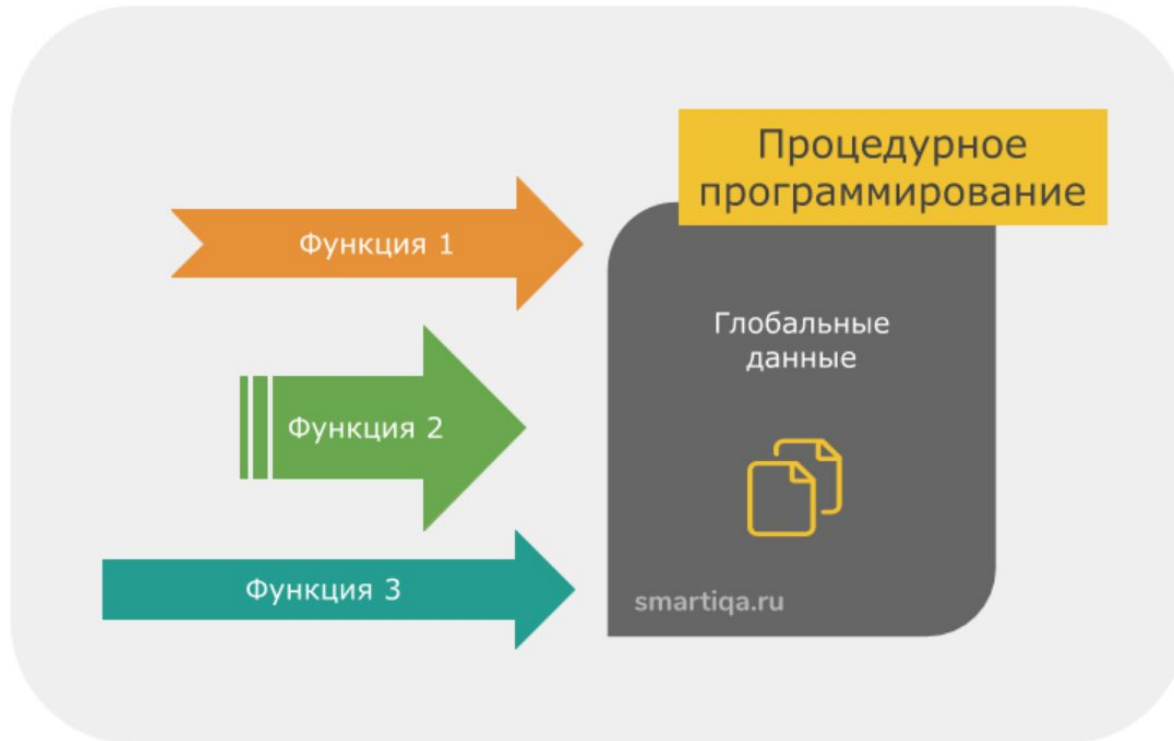


## Что такое ООП?

Вы наверняка слышали, что существуют два главных подхода к написанию программ:

1. Процедурное программирование
2. Объектно-ориентированное программирование (оно же ООП)

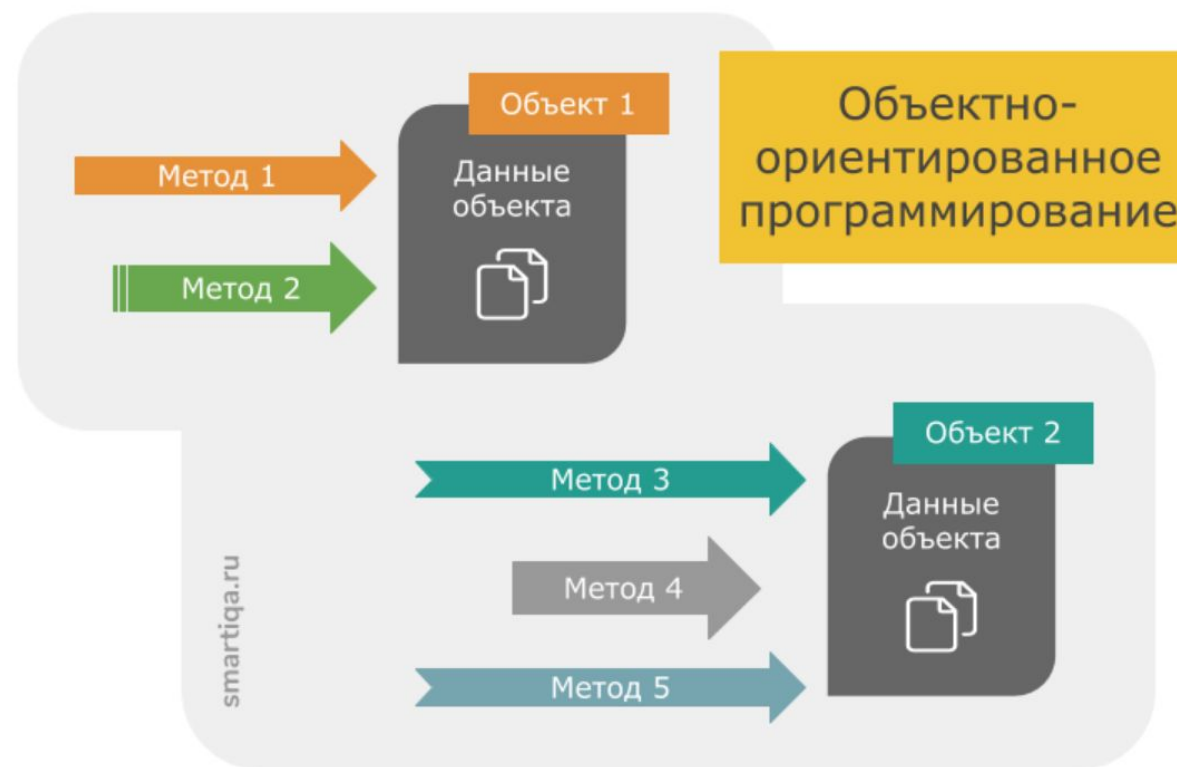
Оба подхода объединены общей целью - сделать процесс программирования максимально эффективным. Это значит, что благодаря им разработка программного обеспечения становится более простой для понимания, легко масштабируемой и содержащей минимальное количество ошибок.



## В чем суть процедурного подхода?

Процедурное программирование – это написание функций и их последовательный вызов в некоторой главной(**main**) функции.

**Объектно-ориентированное программирование (ООП)** — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.



## Итак, чем же хорош подход ООП?

1. Программа разбивается на объекты. Каждый объект отвечает за собственные данные и их обработку. Как результат - код становится проще и читабельней.
2. Уменьшается дубликация кода. Нужен новый объект, содержимое которого на 90% повторяет уже существующий? Давайте создадим новый класс и унаследуем эти 90% функционала от родительского класса!
3. Упрощается и ускоряется процесс написания программ. Можно сначала создать высокоуровневую структуру классов и базовый функционал, а уже потом перейти к их подробной реализации.

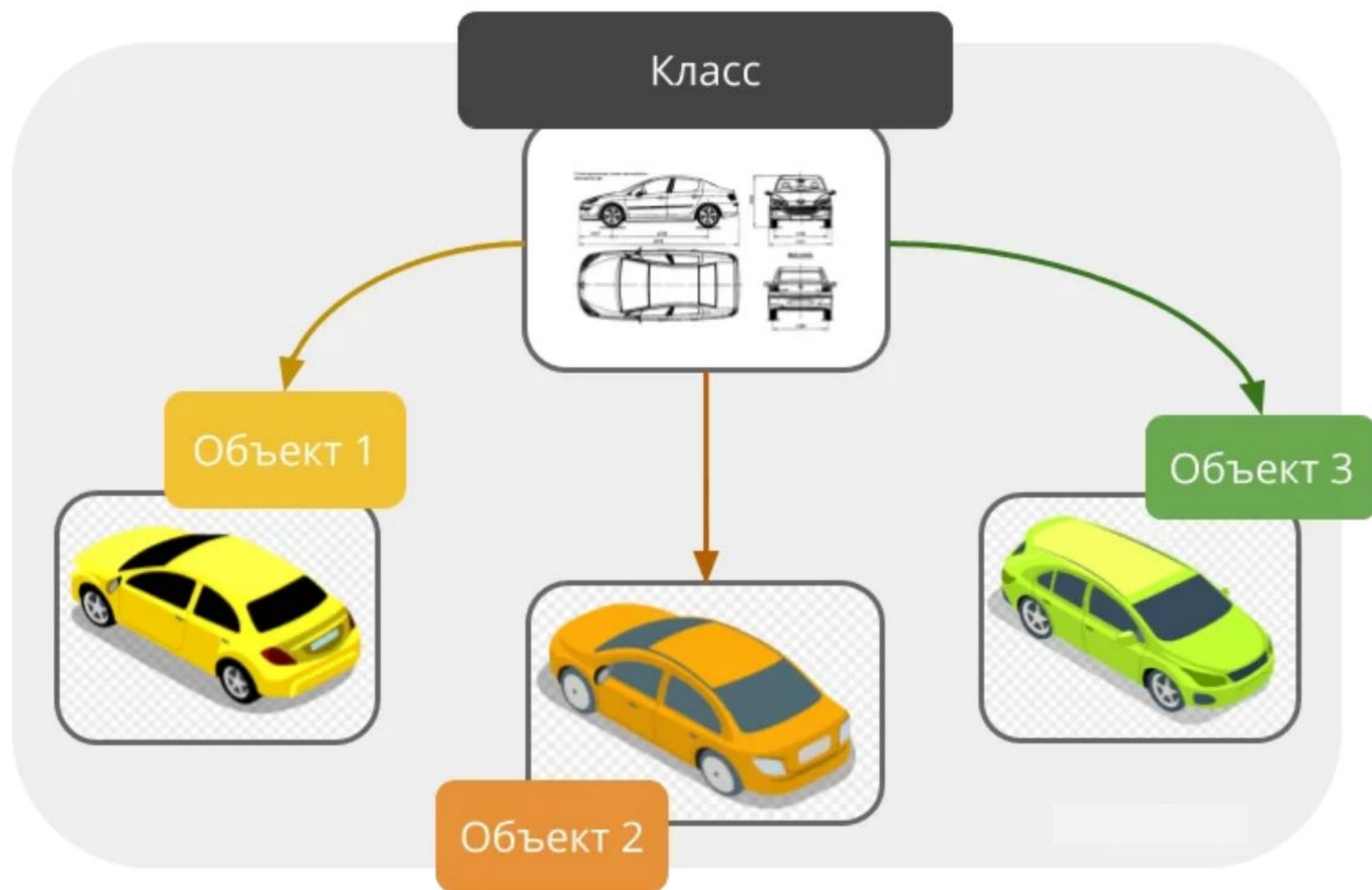
**Класс** — в объектно-ориентированном программировании, представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов.

**Объект** — некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определенные свойства (атрибуты) и операции над ними (методы). Как правило, при рассмотрении объектов выделяется то, что объекты принадлежат одному или нескольким классам, которые определяют поведение (являются моделью) объекта. Термины «экземпляр класса» и «объект» взаимозаменяемы.



1. Класс описывает множество объектов, имеющих общую структуру и обладающих одинаковым поведением. Класс - это шаблон кода, по которому создаются объекты. Т.е. сам по себе класс ничего не делает, но с его помощью можно создать объект и уже его использовать в работе.
2. Данные внутри класса делятся на свойства и методы. Свойства класса (они же поля или атрибуты) - это характеристики объекта класса.
3. Методы класса - это функции, с помощью которых можно оперировать данными класса.
4. Объект - это конкретный представитель класса.
5. Объект класса и экземпляр класса - это одно и то же.

**Класс = Свойства + Методы**





**Абстракция** – принцип ООП, согласно которому объект характеризуется свойствами, которые отличают его от всех остальных объектов и при этом четко определяют его концептуальные границы.

Т. е. абстракция позволяет:

Выделить главные и наиболее значимые свойства предмета.  
Отбросить второстепенные характеристики.



**Инкапсуляция** — принцип ООП, согласно которому сложность реализации программного компонента должна быть спрятана за его интерфейсом.

1. Отсутствует доступ к внутреннему устройству программного компонента.
2. Взаимодействие компонента с внешним миром осуществляется посредством интерфейса, который включает публичные методы и поля.



**Наследование** — способ создания нового класса на основе уже существующего, при котором класс-потомок заимствует свойства и методы родительского класса и также добавляет собственные.

## На что обратить внимание?

1. Класс-потомок = Свойства и методы родителя + Собственные свойства и методы.
2. Класс-потомок автоматически наследует от родительского класса все поля и методы.
3. Класс-потомок может дополняться новыми свойствами.
4. Класс-потомок может дополняться новыми методами, а также заменять (переопределять) унаследованные методы. Переопределить родительский метод - это как? Это значит, внутри класса потомка есть метод, который совпадает по названию с методом родительского класса, но функционал у него новый - соответствующий потребностям класса-потомка.

## Объект Дом:

### СВОЙСТВА

- 1) Тип фундамента
- 2) Материал крыши
- 3) Количество окон
- 4) Количество дверей

### МЕТОДЫ

- 1) Построить
- 2) Отремонтировать
- 3) Заселить
- 4) Снести

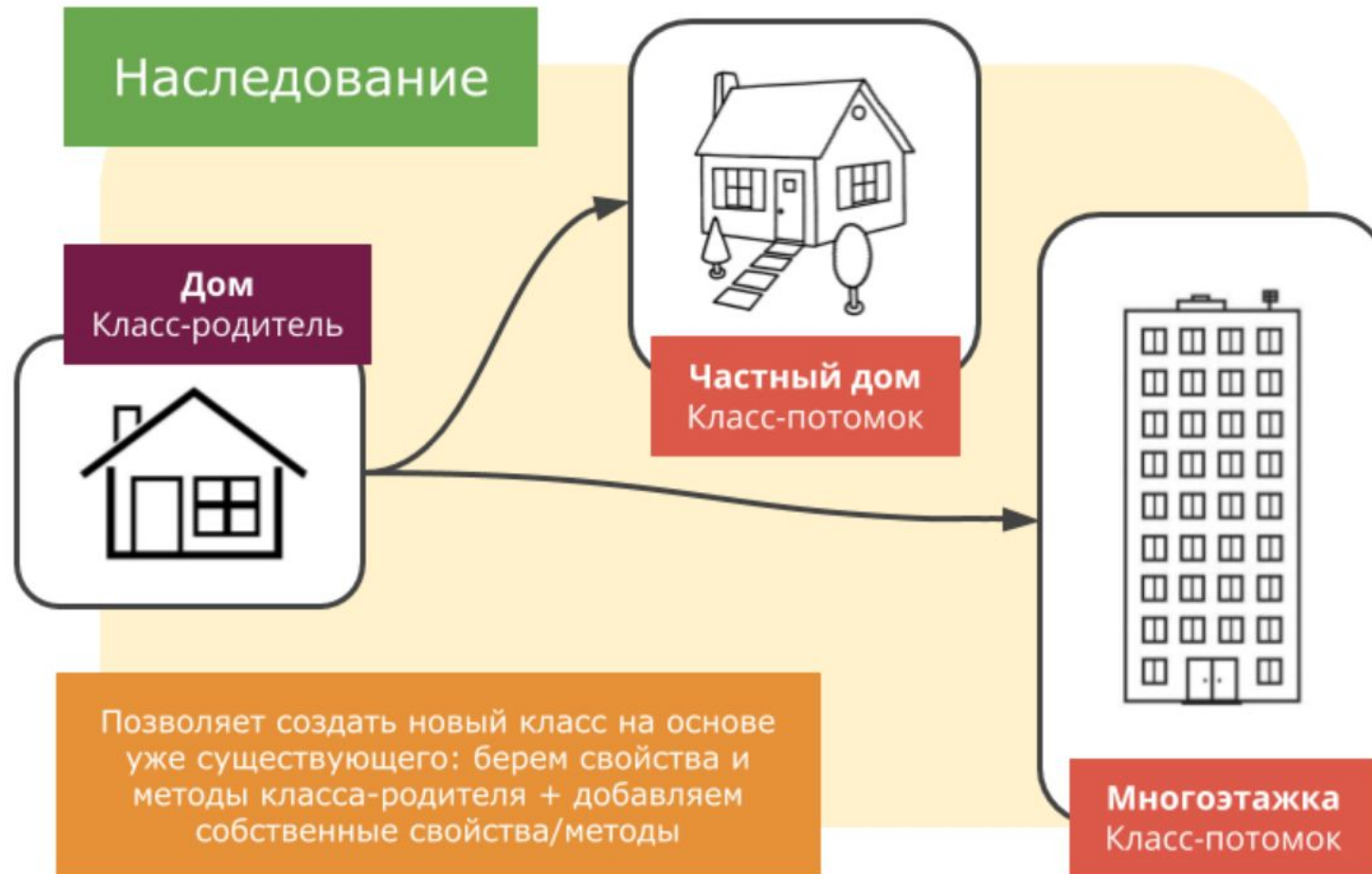
## Объект Частный Дом:

### СВОЙСТВА

- 1) Тип фундамента (УНАСЛЕДОВАНО)
- 2) Материал крыши (УНАСЛЕДОВАНО)
- 3) Количество окон (УНАСЛЕДОВАНО)
- 4) Количество дверей (УНАСЛЕДОВАНО)
- 5) Количество комнат
- 6) Тип отопления
- 7) Наличие огорода

### МЕТОДЫ

- 1) Построить (УНАСЛЕДОВАНО)
- 2) Отремонтировать (УНАСЛЕДОВАНО)
- 3) Заселить (УНАСЛЕДОВАНО)
- 4) Снести (УНАСЛЕДОВАНО)
- 5) Изменить фасад
- 6) Утеплить
- 7) Сделать пристройку



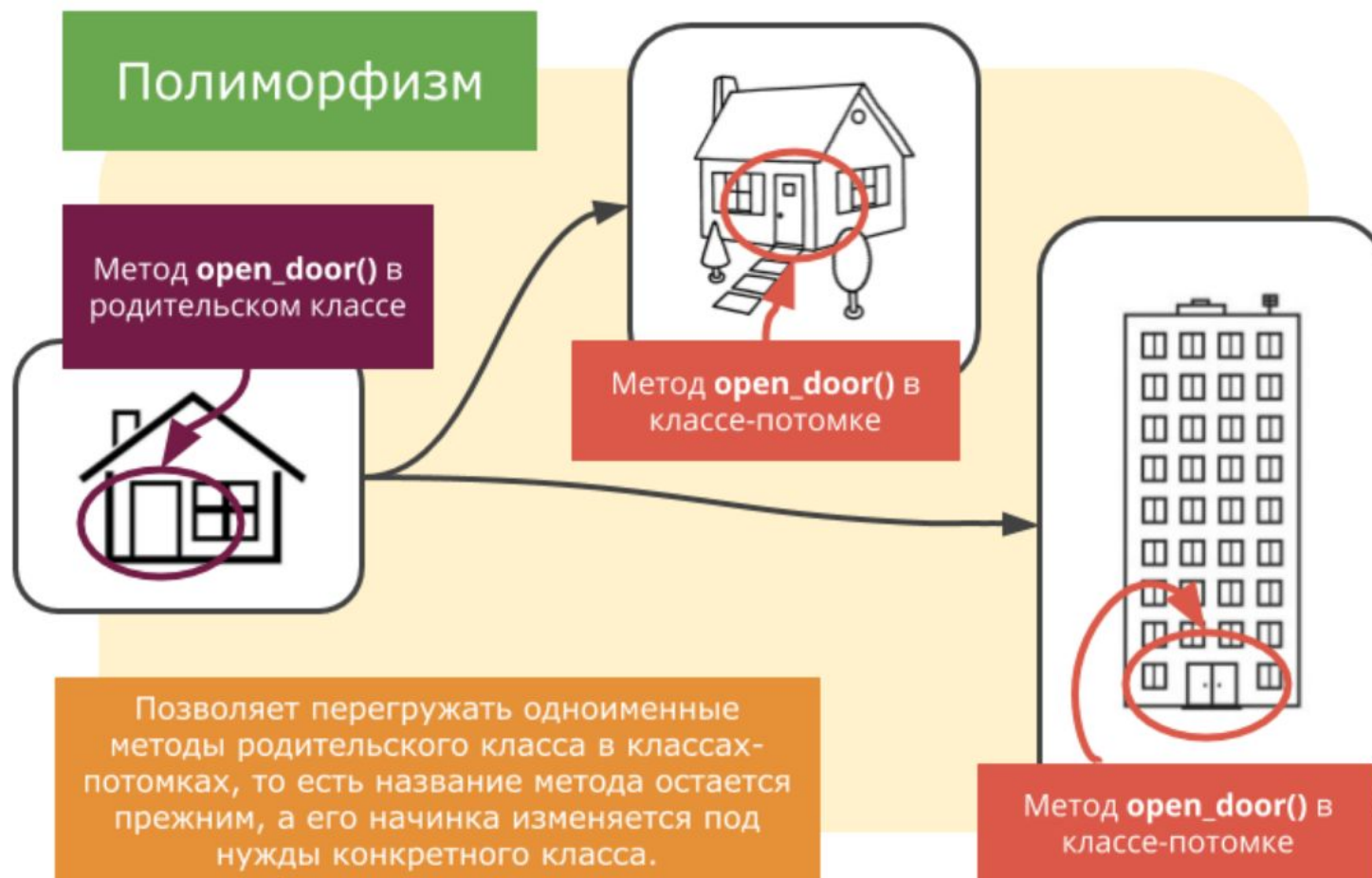
**Полиморфизм** — это поддержка нескольких реализаций на основе общего интерфейса.

Другими словами, полиморфизм позволяет перегружать одноименные методы родительского класса в классах-потомках.

Также для понимания работы этого принципа важным является понятие абстрактного метода:

**Абстрактный метод** (он же виртуальный метод) - это метод класса, реализация для которого отсутствует.

**Как итог - за одинаковым названием могут скрываться методы с совершенно разным функционалом, который в каждом конкретном случае соответствует нуждам класса, к которому он относится.**



Теперь давайте посмотрим, как реализуется ООП в рамках языка программирования Python. Синтаксис для создания класса выглядит следующим образом:

Синтаксис

**class <название\_класса>:**  
    **<тело\_класса>**

А вот так компактно смотрится пример объявления класса с минимально возможным функционалом:

```
1 class Car:
2     pass
```



Чтобы создать объект класса, нужно воспользоваться следующим синтаксисом:

### Синтаксис

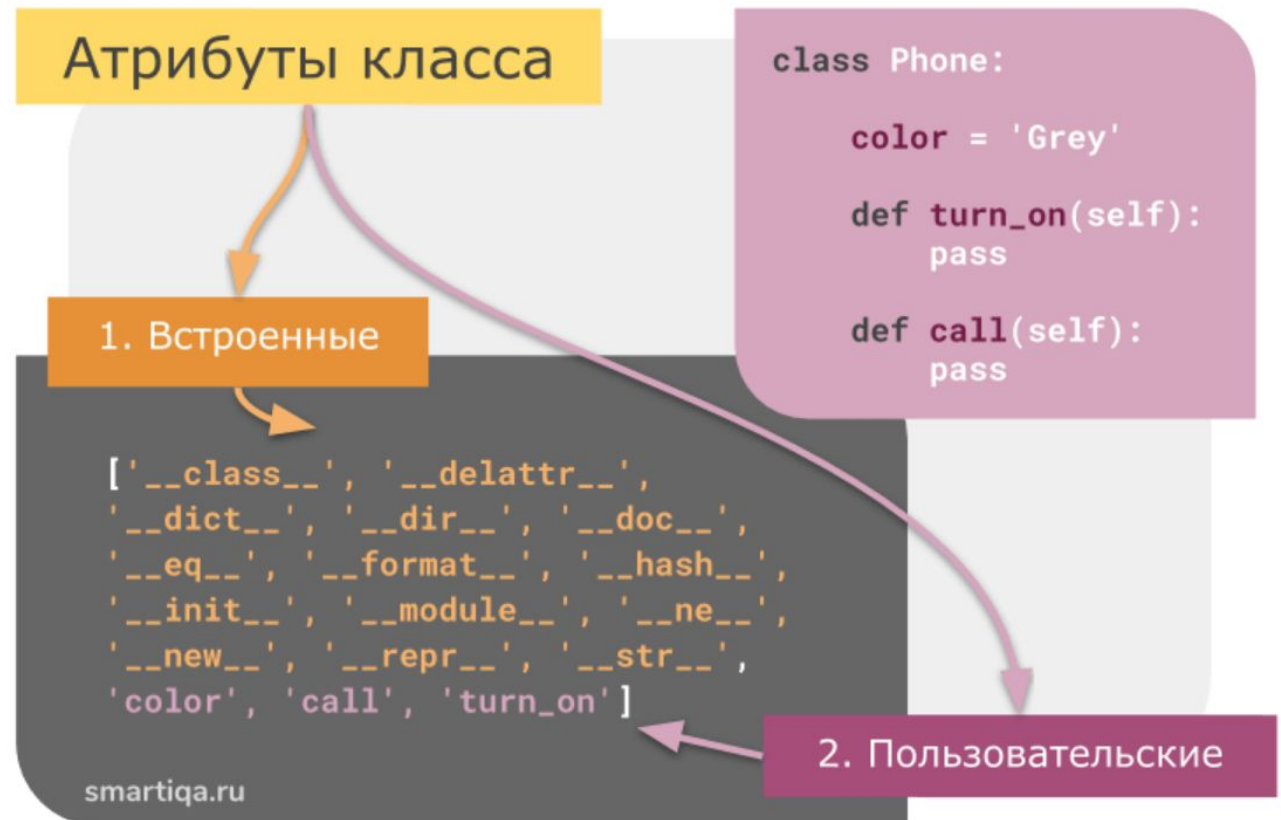
**<имя\_объекта> = <имя\_класса>()**

И в качестве примера создадим объект класса **Car**:

```
4 car_object = Car()
```

**Все атрибуты можно разделить на 2 группы:**

1. Встроенные(служебные) атрибуты
2. Пользовательские атрибуты



Атрибут	Назначение	Тип
<b><code>__new__(cls[, ...])</code></b>	Конструктор. Создает экземпляр(объект) класса. Сам класс передается в качестве аргумента.	Функция
<b><code>__init__(self[, ...])</code></b>	Инициализатор. Принимает свежесозданный объект класса из конструктора.	Функция
<b><code>__del__(self)</code></b>	Деструктор. Вызывается при удалении объекта сборщиком мусора	Функция
<b><code>__str__(self)</code></b>	Возвращает строковое представление объекта.	Функция
<b><code>__hash__(self)</code></b>	Возвращает хэш-сумму объекта.	Функция
<b><code>__setattr__(self, attr, val)</code></b>	Создает новый атрибут для объекта класса с именем attr и значением val	Функция
<b><code>__doc__</code></b>	Документация класса.	Строка (тип str)
<b><code>__dict__</code></b>	Словарь, в котором хранится пространство имен класса	Словарь (тип dict)

## Это важно

В теории ООП конструктор класса - это специальный блок инструкций, который вызывается при создании объекта. При работе с питоном может возникнуть мнение, что метод `__init__(self)` - это и есть конструктор, но это не совсем так. На самом деле, при создании объекта в Python вызывается метод `__new__(cls, *args, **kwargs)` и именно он является конструктором класса.

Также обратите внимание, что `__new__()` - это метод класса, поэтому его первый параметр `cls` - ссылка на текущий класс. В свою очередь, метод `__init__()` является так называемым инициализатором класса. Именно этот метод первый принимает созданный конструктором объект. Как вы уже, наверное, не раз замечали, метод `__init__()` часто переопределяется внутри класса самим программистом. Это позволяет со всем удобством задавать параметры будущего объекта при его создании.

Список атрибутов класса / объекта можно получить с помощью команды **dir()**

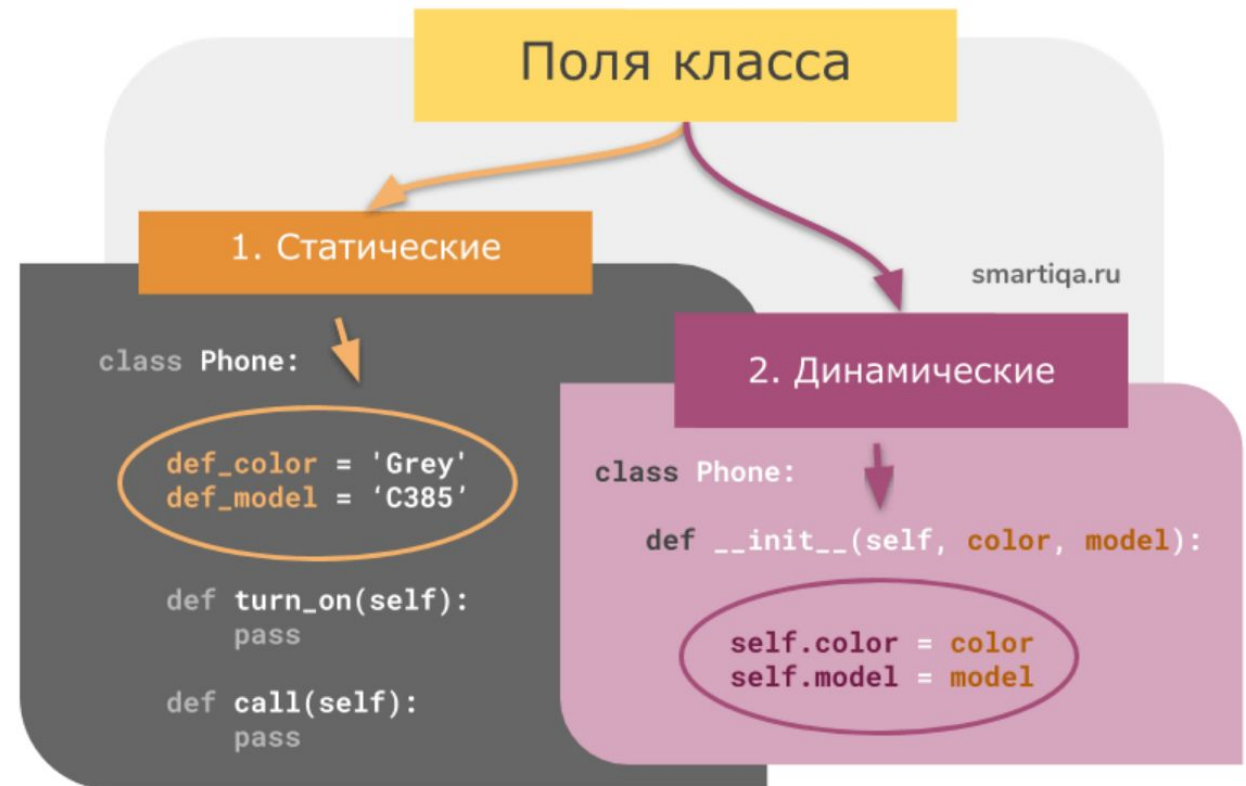
```
6 print(dir(Car))
```

Как видим, в нем есть только встроенные атрибуты, которые наш класс по умолчанию унаследовал от базового класса **object**. А теперь добавим ему функционала:

```
1 class Car:
2     color = 'Grey'
3
4     def turn_on(self):
5         pass
6
7     def ride(self):
8         pass
9
10    car_object = Car()
```

Поля(они же свойства или переменные) можно (также условно) разделить на две группы:

- ❖ Статические поля
- ❖ Динамические поля



Это переменные, которые объявляются внутри тела класса и создаются тогда, когда создается класс. Создали класса - создалась переменная:

```
1 class Car:
2
3     # Статические поля (переменные класса)
4     default_color = 'Grey'
5     default_weight = 5000
6
```

Это переменные, которые создаются на уровне экземпляра класса. Нет экземпляра - нет его переменных. Для создания динамического свойства необходимо обратиться к **self** внутри метода

```
1 class Car:
2
3     # Статические поля (переменные класса)
4     default_color = 'Grey'
5     default_weight = 5000
6
7     def __init__(self, color, model):
8         # Динамические поля (переменные объекта)
9         self.color = color
10        self.model = model
11
12    def turn_on(self):
13        pass
14
```



Служебное слово self — это ссылка на текущий экземпляр класса. Как правило, эта ссылка передается в качестве первого параметра метода Python:

**class Apple:**

**\_\_\_\_# Создаем объект с общим количеством яблок 12**

**\_\_\_\_def \_\_init\_\_(self):**

**\_\_\_\_\_self.whole\_amount = 12**

**\_\_\_\_# Съедаем часть яблок для текущего объекта**

**\_\_\_\_def eat(self, number):**


**\_\_\_\_\_self.whole\_amount -= number**

Стоит обратить внимание, что на самом деле слово self не является зарезервированным. Просто существует некоторое соглашение, по которому первый параметр метода именуется self и передает ссылку на текущий объект, для которого этот метода был вызван. Хотите назвать первый параметр метода по-другому — пожалуйста.



## Задание №1

Создайте класс Example. В нём пропишите 3 (метода) функции. Две переменные задайте статически, две динамически.  
Первая функция: создайте переменную и выведите её  
Вторая функция: верните сумму 2-ух глобальных переменных  
Третья функция: верните результат возведения первой динамической переменной во вторую динамическую переменную  
Создайте объект класса.  
Напечатайте обе функции  
Напечатайте переменную a

```
1 class TheExample:
2     a = 2
3     b = 3
4
5     def __init__(self, t, r):
6         self.t = t
7         self.r = r
8
9     def func(self):
10         self.c = 5
11         print(self.c)
12
13     def func1(self):
14         return self.a + self.b
15
16     def func2(self):
17         return self.t**self.r
18
19 
20 example = TheExample(4, 2)
21 print(example.a)
22 print(example.func1())
23 print(example.func2())
24 example.func()
```



## Задание №2

Калькулятор.

Создайте класс, где реализованы функции(методы) математических операций. А также функция, для ввод данных.

```
class TheExample:
    def __init__(self):
        self.func4()

    def func(self):
        return self.a + self.b

    def func1(self):
        return self.a - self.b

    def func2(self):
        return self.a * self.b

    def func3(self):
        if self.b == 0:
            return "error"
        else:
            return self.a / self.b

    def func4(self):
        self.a = int(input())
        self.b = int(input())
```

```
while True:
    print("+,-,*,/")
    x = input()
    print("Numbers:")
    example = TheExample()
    if x == "6":
        break
    if x == "+":
        print(example.func())
    if x == "-":
        print(example.func1())
    if x == "*":
        print(example.func2())
    if x == "/":
        print(example.func3())
```



## Домашнее задание

Два метода в классе, один принимает в себя либо строку, либо число.

Если я передаю строку, то смотрим:

если произведение гласных и согласных букв меньше или равно длине слова, выводить все гласные, иначе согласные;

если число то, произведение суммы чётных цифр на длину числа.

Длину строки и числа искать во втором методе.