

# Объектно-ориентированное программирование

## Лекция 1. Концепции ООП и первые шаги в Python

Объекты и классы. Атрибуты и методы. Состояние и поведение объектов. Разница между парадигмами. Определение классов в Python, создание экземпляров.  
Использование `__init__`.

**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.

**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 ( поля) 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.

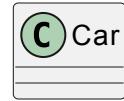
# Классы и объекты

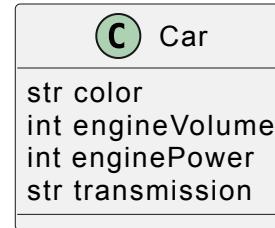
1. **Класс** описывает множество объектов, имеющих общую структуру и обладающих одинаковым поведением. Класс - это шаблон кода, по которому создаются объекты.
2. Данные внутри класса делятся на свойства и методы. **Свойства класса** (они же поля) - это характеристики объекта класса. Они описывают **состояние** объекта.
3. **Методы класса** - это функции, с помощью которых можно оперировать данными класса. Они описывают **поведение** объекта.
4. **Объект** - это конкретный представитель класса.
5. Объект класса и **экземпляр класса** - это одно и то же.

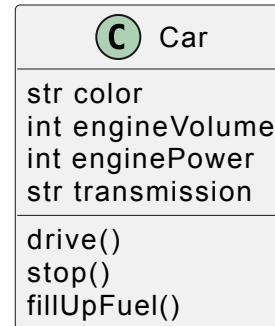
*Класс = Свойства + Методы*



Университет  
**Сириус** Пример  
Колледж







# Разница между парадигмами

## Процедурное программирование

Basic, C, Pascal, Go

Главное - код для обработки данных, сами данные имеют второстепенное значение

Простые программы, где весь функционал можно реализовать несколькими десятками процедур/ функций.

Много дубликации кода и дублирование данных.

Все данные внутри процедуры видны только локально, а значит их нельзя использовать в другом месте.

## Объектно-ориентированное программирование

C++, Java, Python, ...

Главное в программе - данные. Именно они определяют, какие методы будут использоваться для их обработки. Т.е. данные первичны, код для обработки этих данных - вторичен.

Программа разбивается на объекты. Каждый объект отвечает за собственные данные и их обработку.

Уменьшается дупликация кода.

Упрощается и ускоряется процесс написания программ.

## Разница между парадигмами. Пример

```
1 from statistics import mean
2
3 cars = [
4     {"brand": "Toyota", "year": 2020, "price": 18000},
5     {"brand": "BMW", "year": 2022, "price": 35000},
6     {"brand": "Audi", "year": 2021, "price": 40000},
7     {"brand": "Ford", "year": 2019, "price": 22000},
8     {"brand": "Kia", "year": 2023, "price": 19500},
9 ]
10
11 # 1. Фильтрация (только машины дороже 20 000)
12 filtered = filter(lambda c: c["price"] > 20000, cars)
13
14 # 2. Сортировка по году (от новых к старым)
15 sorted_cars = sorted(filtered, key=lambda c: c["year"], reverse=True)
```

# Разница между парадигмами. Пример

```
1 from statistics import mean
2
3
4 class Car:
5     def __init__(self, brand, year, price):
6         self.brand = brand
7         self.year = year
8         self.price = price
9
10    def __repr__(self):
11        return f"{self.brand} ({self.year}): ${self.price}"
12
13
14 class CarDataset:
15     def __init__(self, cars):
```

```
1 class <название_класса>:  
2     <тело_класса>  
3  
4 <имя_объекта> = <имя_класса>()
```

# Классы и объекты в Python

```
1 class Car:  
2     pass  
3  
4 car_object = Car()
```

**Атрибут** - это любой элемент класса или объекта.

Все атрибуты можно разделить на 2 группы:

1. Встроенные (служебные) атрибуты
2. Пользовательские атрибуты

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

# Разница между `__new__` и `__init__`

В Python создание объекта происходит в два этапа:

Метод	Назначение	Когда вызывается	Что делает
<code>__new__(cls, ...)</code>	<b>Конструктор</b>	Перед созданием объекта	Создает и возвращает пустой объект класса
<code>__init__(self, ...)</code>	<b>Инициализатор</b>	После создания объекта	Заполняет объект значениями начальными (инициализация состояния)

```
1 class Example:
2     def __new__(cls):
3         print("Создание объекта (__new__)")
4         return super().__new__(cls)
5
6     def __init__(self):
7         print("Инициализация объекта (__init__)")
8
9 obj = Example()
```

Создание объекта (\_\_new\_\_)

Инициализация объекта (\_\_init\_\_)

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

```
1 class Phone:  
2     pass  
3  
4 print(dir(Phone))  
5 # ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',  
6 # '__firstlineno__', '__format__', '__ge__', '__getattribute__',  
7 # '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__',  
8 # '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',  
9 # '__setattr__', '__sizeof__', '__static_attributes__', '__str__', '__su  
10 # '__weakref__']
```

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

```
1 class Phone:  
2     color = 'Grey'  
3  
4     def turn_on(self):  
5         pass  
6  
7     def call(self):  
8         pass  
9  
10    print(dir(Phone))  
11    #['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__  
12    #...  
13    #__weakref__', 'call', 'color', 'turn_on']
```

## Атрибуты

Первым аргументом любого метода является *self* - ссылка на объект, вызвавший метод

```
1 class MyClass:  
2     def __init__(self):  
3         self.x = 10  
4         self.y = 20  
5  
6     obj = MyClass()  
7     setattr(obj, 'z', 30)  
8     print(f"getattr(obj, \"z\") = {obj.z}")  
9 # obj.z = 30
```

## Атрибуты

Первым аргументом любого метода является *self* - ссылка на объект, вызвавший метод

```
1 class MyClass:  
2     def __init__(self):  
3         self.x = 10  
4         self.y = 20  
5  
6     obj = MyClass()  
7     obj.d = 40  
8     print(f"{obj.d = }")  
9 # obj.d = 40
```

# Поля (свойства) класса в Python

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

- Статические поля
- Динамические поля

Это переменные, которые объявляются внутри тела класса и создаются тогда, когда создается класс.

```
1 class Phone:  
2     # Статические поля (переменные класса)  
3     default_color = 'Grey'  
4     default_model = 'C385'
```

Это переменные, которые создаются на уровне экземпляра класса. Для создания динамического свойства необходимо обратиться к `self` внутри метода:

```
1 class Phone:  
2     # Статические поля (переменные класса)  
3     default_color = 'Grey'  
4     default_model = 'C385'  
5  
6     def __init__(self, color, model):  
7         # Динамические поля (переменные объекта)  
8         self.color = color  
9         self.model = model  
10  
11 my_phone_red = Phone('Red', 'I495')
```