Программирование на Python

Объектно-ориентированное программирование. Классы.

Алексей Андреевич Сорокин

спецкурс, ОТИПЛ МГУ,
осенний семестр 2017–2018 учебного года
7 ноября 2017 г.

Три парадигмы программирования

- Процедурное программирование
  - Части программы соединены с помощью функций.
  - Данные и функции отделены.
  - Состояние данных может меняться.

Три парадигмы программирования

- Процедурное программирование
  - Части программы соединены с помощью функций.
  - Данные и функции отделены.
  - Состояние данных может меняться.
- Функциональное программирование:
  - Данные и функции неотделимы (есть только функции).
  - Нет понятия состояния.

Три парадигмы программирования

- Процедурное программирование
  - Части программы соединены с помощью функций.
  - Данные и функции отделены.
  - Состояние данных может меняться.
- Функциональное программирование:
  - Данные и функции неотделимы (есть только функции).
  - Нет понятия состояния.
- Объектно-ориентированное программирование.
  - Данные и методы работы с ними объединены в одном объекте.
  - Состояние данных может меняться.
  - Части программы соединены с помощью объектов.

Классы: введение

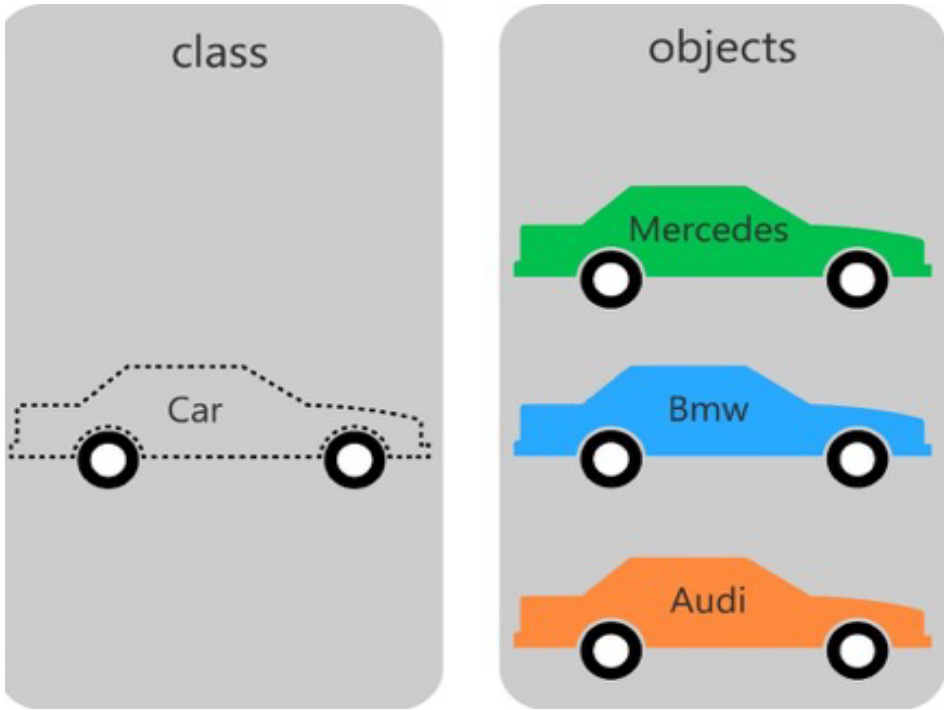
- Класс — абстракция, отражающая модель некоторой сущности.

Классы: введение

- Класс — абстракция, отражающая модель некоторой сущности.
- Объект — конкретная реализация класса, соответствующая некоторому экземпляру сущности.

Классы: введение

- Класс — абстракция, отражающая модель некоторой сущности.
- Объект — конкретная реализация класса, соответствующая некоторому экземпляру сущности.



Классы: введение

- Метод класса — функция, принадлежащая классу/объекту.
- Поле класса — переменная, содержащаяся в классе/объекте.

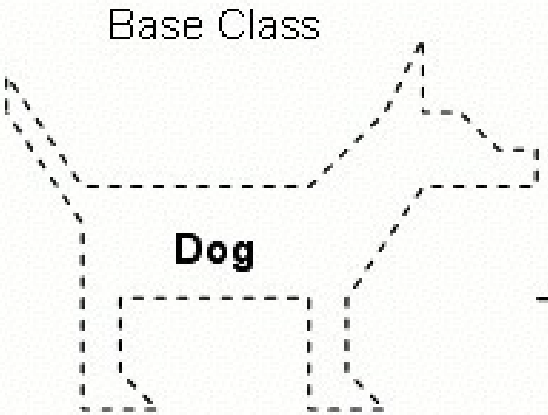
Классы: введение

- Метод класса — функция, принадлежащая классу/объекту.
- Поле класса — переменная, содержащаяся в классе/объекте.
- Специальные методы:
  - Конструктор — функция, создающая объект класса.

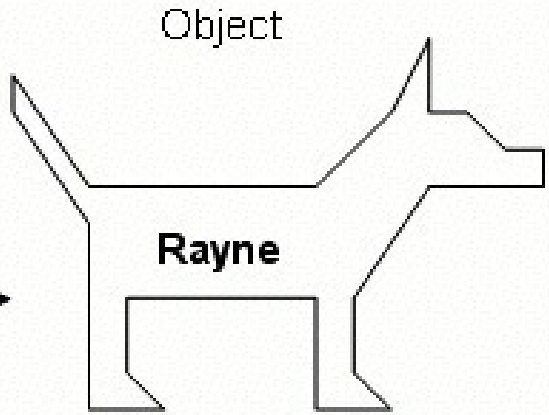
Классы: введение

- Метод класса — функция, принадлежащая классу/объекту.
- Поле класса — переменная, содержащаяся в классе/объекте.
- Специальные методы:
  - Конструктор — функция, создающая объект класса.
  - Деструктор — функция, удаляющая объект класса.
  - В Python'е деструкторов нет!

Классы: введение



Create Instance



Properties	**Methods**
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property values	**Methods**
Color: Gray, White, and Black	Sit
Eye Color: Blue and Brown	Lay Down
Height: 18 Inches	Shake
Length: 36 Inches	Come
Weight: 30 Pounds	

Основные понятия ООП

- Полиморфизм: единообразная обработка различных типов данных.
  - Для всех типов данных реализована операция сравнения.
  - Сложение всегда реализуется через оператор + (перегрузка операторов).

Основные понятия ООП

- Полиморфизм: единообразная обработка различных типов данных.
  - Для всех типов данных реализована операция сравнения.
  - Сложение всегда реализуется через оператор + (перегрузка операторов).
- Инкапсуляция: данные связаны с методами, реализация отделена от интерфейса, принцип “чёрного ящика”.
  - Неважно, как реализованы словари, важно, что они поддерживают операции [], get и items.

Основные понятия ООП

- Полиморфизм: единообразная обработка различных типов данных.
  - Для всех типов данных реализована операция сравнения.
  - Сложение всегда реализуется через оператор + (перегрузка операторов).
- Инкапсуляция: данные связаны с методами, реализация отделена от интерфейса, принцип “чёрного ящика”.
  - Неважно, как реализованы словари, важно, что они поддерживают операции [], get и items.
- Наследование: создание нового класса (потомка) на основе уже имеющегося базового класса.
  - Классы **defaultdict** и **OrderedDict** наследуются от класса **dict**.

Реализация класса

- Конструктор класса — функция `__init__`

Реализация класса

- Конструктор класса — функция __init__
- Обязательный первый аргумент **self** (указывает на экземпляр класса).

```
class Simple:
  def __init__(self):
    pass
```

- Этот конструктор ничего не делает и не получает аргументов.

Реализация класса

- Конструктор класса — функция `__init__`
- Обязательный первый аргумент **self** (указывает на экземпляр класса).

```
class Simple:
  def __init__(self):
    pass
```

- Этот конструктор ничего не делает и не получает аргументов.
- Можно и более содержательный пример:

```
class Complex:
  def __init__(self, re, im):
    self.re = re
    self.im = im
```

Пример реализации класса

- Обращение к переменным и методам класса производится через точку.

Пример реализации класса

- Обращение к переменным и методам класса производится через точку.
- Напишем класс для комплексных чисел.
- Инициализация:

```
class Simple:
def __init__(self, re, im):
  self.re = re
  self.im = im
```

Пример реализации класса

- Обращение к переменным и методам класса производится через точку.
- Напишем класс для комплексных чисел.
- Инициализация:

```
class Simple:
def __init__(self, re, im):
  self.re = re
  self.im = im
```

- Сложение:

```
def plus(self, other):
  re = self.re + other.re
  im = self.im + other.im
  return Complex(re, im)
```

Пример реализации класса

- Унарный и бинарный минус:

```
def unary_minus(self):
  return Complex(-self.re, -self.im)
```

```
def minus(self, other):
  return self.plus(other.unary_minus())
```

Пример реализации класса

- Унарный и бинарный минус:

```
def unary_minus(self):
  return Complex(-self.re, -self.im)
```

```
def minus(self, other):
  return self.plus(other.unary_minus())
```

- Умножение $((a + bi)(c + di) = (ac - bd) + (ad + bc)i$.

```
def mult(self, other):
  re = self.re * other.re -self.im * other.im
  im = self.im * other.re + self.re * other.im
  return Complex(re, im)
```

Пример реализации класса

Взятие обратного, деление и модуль:

```
def inverse(self):
  squared_module =\
    self.re * self.re + self.im * self.im
  re = self.re / squared_module
  im = -self.im / squared_module
  return Complex(re, im)

def divide(self, other):
  return self.mult(other.inverse())

def module(self):
  return math.sqrt(
    self.re * self.re + self.im * self.im)
```

Пример реализации класса

- Мы создали класс “комплексные числа”. позволяющий производить с ними стандартные операции.

Пример реализации класса

- Мы создали класс “комплексные числа”. позволяющий производить с ними стандартные операции.
- Однако хотелось бы обозначать эти операции стандартными символами? Как этого добиться?

Пример реализации класса

- Мы создали класс “комплексные числа”. позволяющий производить с ними стандартные операции.
- Однако хотелось бы обозначать эти операции стандартными символами? Как этого добиться?
- Для этого применяется *перегрузка операторов*. Например, сложению соответствует `__add__`:

```
def __add__(self, other):
  return Complex(self.re + other.re,
                 self.im + other.im)
```

Перегрузка операторов

Другие операторы:

Операция	знак	оператор
сложение	+	__add__
вычитание	−	__sub__
умножение	+	__mult__
деление	+	__div__
унарный минус	+	__neg__
сравнение	+	__cmp__
длина	+	__len__
строковое представление		__str__