Объекты Пайтон

Чарльз Северанс

Пайтон для всех
www.py4e.com

Внимание!

- Эта лекция посвящена определениям и механике объектов
- Лекция больше о том, «как это работает» и в меньшей степени о том, «как это используется»
- Вы не получите цельной картины без рассмотрения в контексте реальной проблемы
- Отбросьте сомнения и изучите технику! Приблизительно 40 слайдов ждут вас…

5. Data Structures

This chapter describes some things you've learned about already in more detail, and adds some new things as well.

5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

`list.`**`append`**`(x)`
Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

`list.`**`extend`**`(L)`
Extend the list by appending all the items in the given list. Equivalent to `a[len(a):] = L`.

`list.`**`insert`**`(i, x)`
Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.`**`remove`**`(x)`
Remove the first item from the list whose value is *x*. It is an error if there is no such item.

`list.`**`pop`**`([i])`
Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

https://docs.python.org/3/tutorial/datastructures.html

12.6. `sqlite3` — DB-API 2.0 interface for SQLite databases

Source code: Lib/sqlite3/

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The sqlite3 module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by PEP 249.

To use the module, you must first create a `Connection` object that represents the database. Here the data will be stored in the `example.db` file:

```
import sqlite3
conn = sqlite3.connect('example.db')
```

You can also supply the special name `:memory:` to create a database in RAM.

Once you have a `Connection`, you can create a `Cursor` object and call its `execute()` method to perform SQL commands:

```
c = conn.cursor()

# Create table
c.execute('''CREATE TABLE stocks
             (date text, trans text, symbol text, qty real, price real)''')
```

https://docs.python.org/3/library/sqlite3.html

Начнем с программ

```
inp = input('Этаж в Европе?')
usf = int(inp) + 1
print('Этаж в США', usf)
```

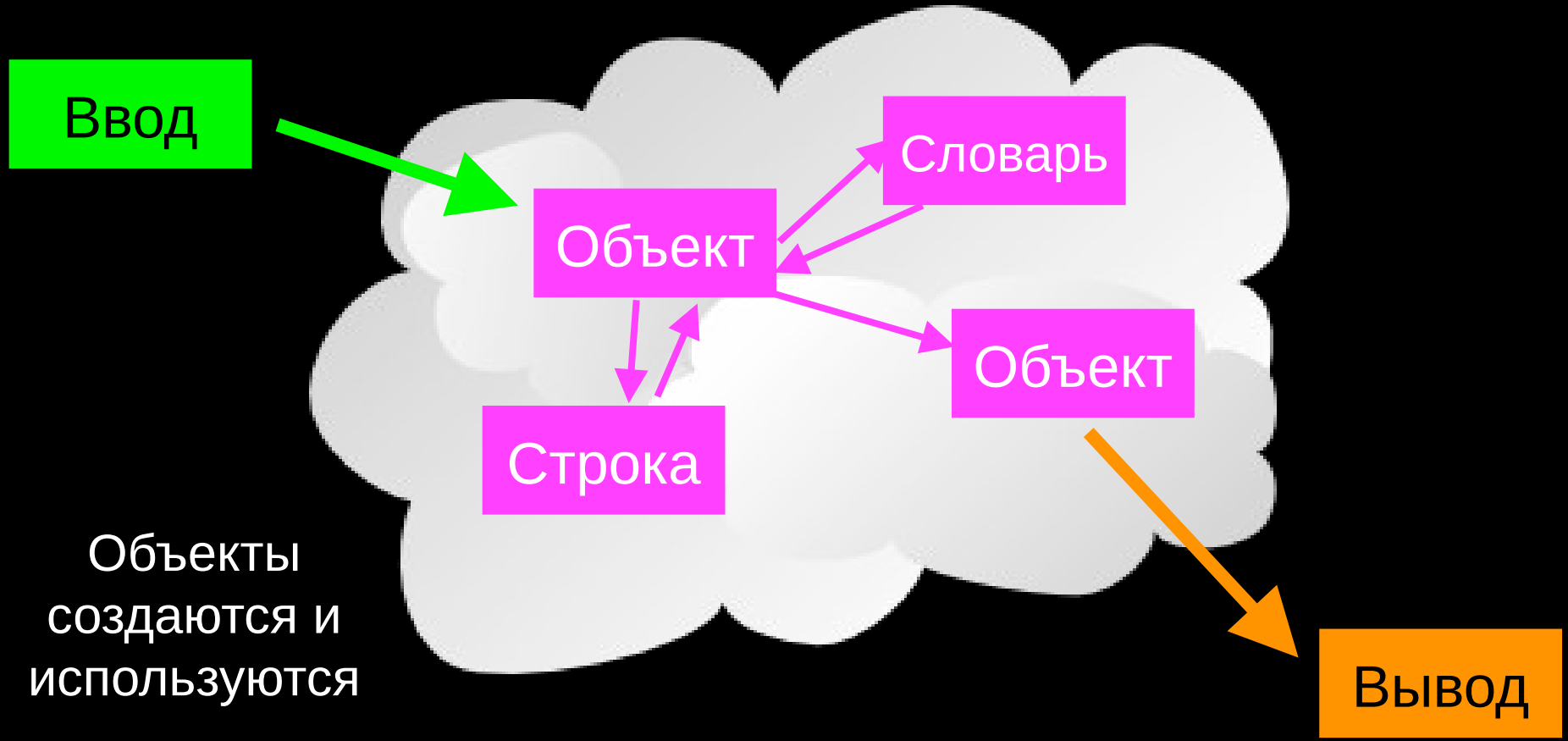
Этаж в Европе? 0
Этаж в США 1

Объектно-ориентированное программирование

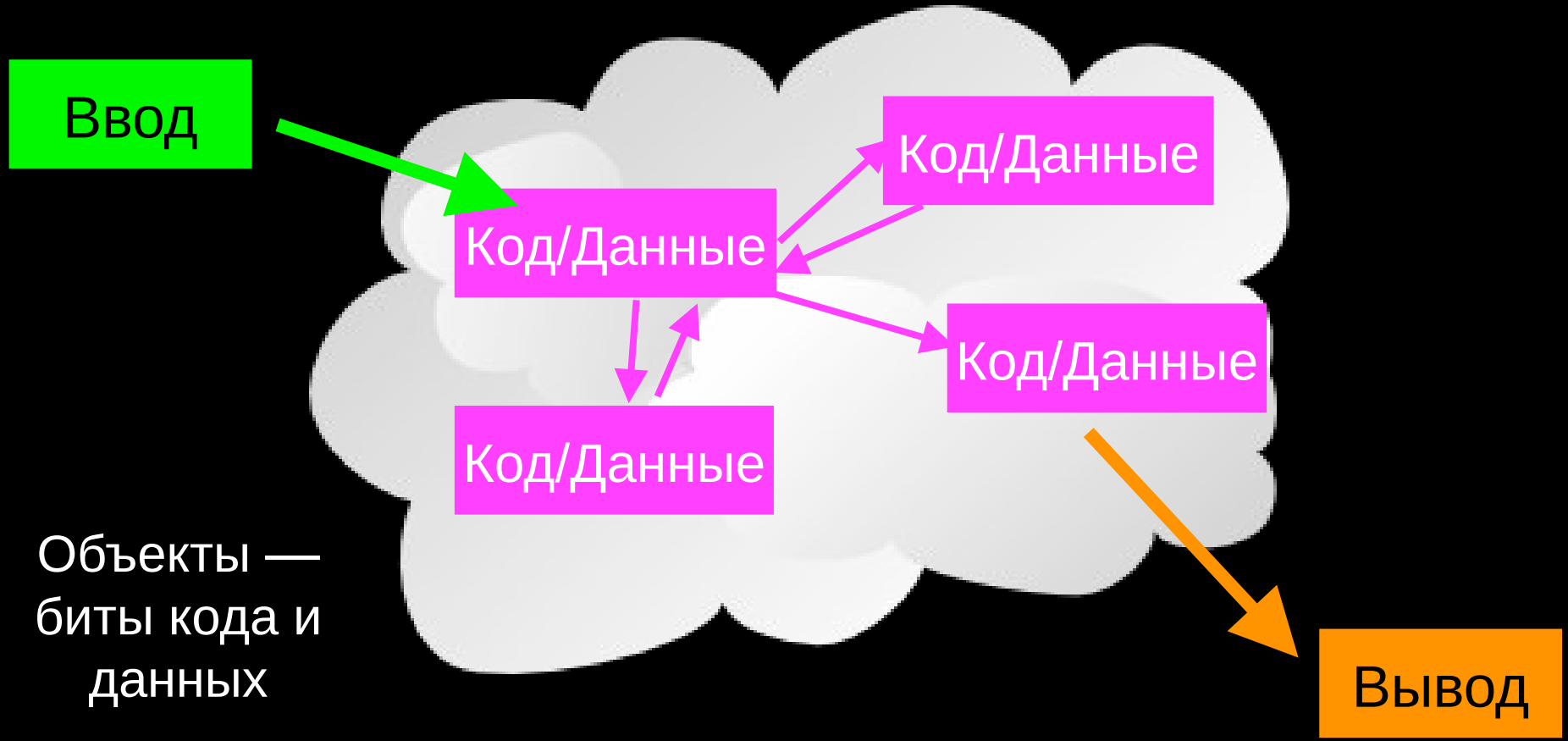
- Программа состоит из множества взаимодействующих объектов
- Вместо того, чтобы быть «цельной программой», каждый объект — словно маленький «остров» внутри программы, совместно работающий вместе с другими объектами
- Программа состоит из одного или нескольких объектов, работающих вместе и использующих возможности друг друга

Объект

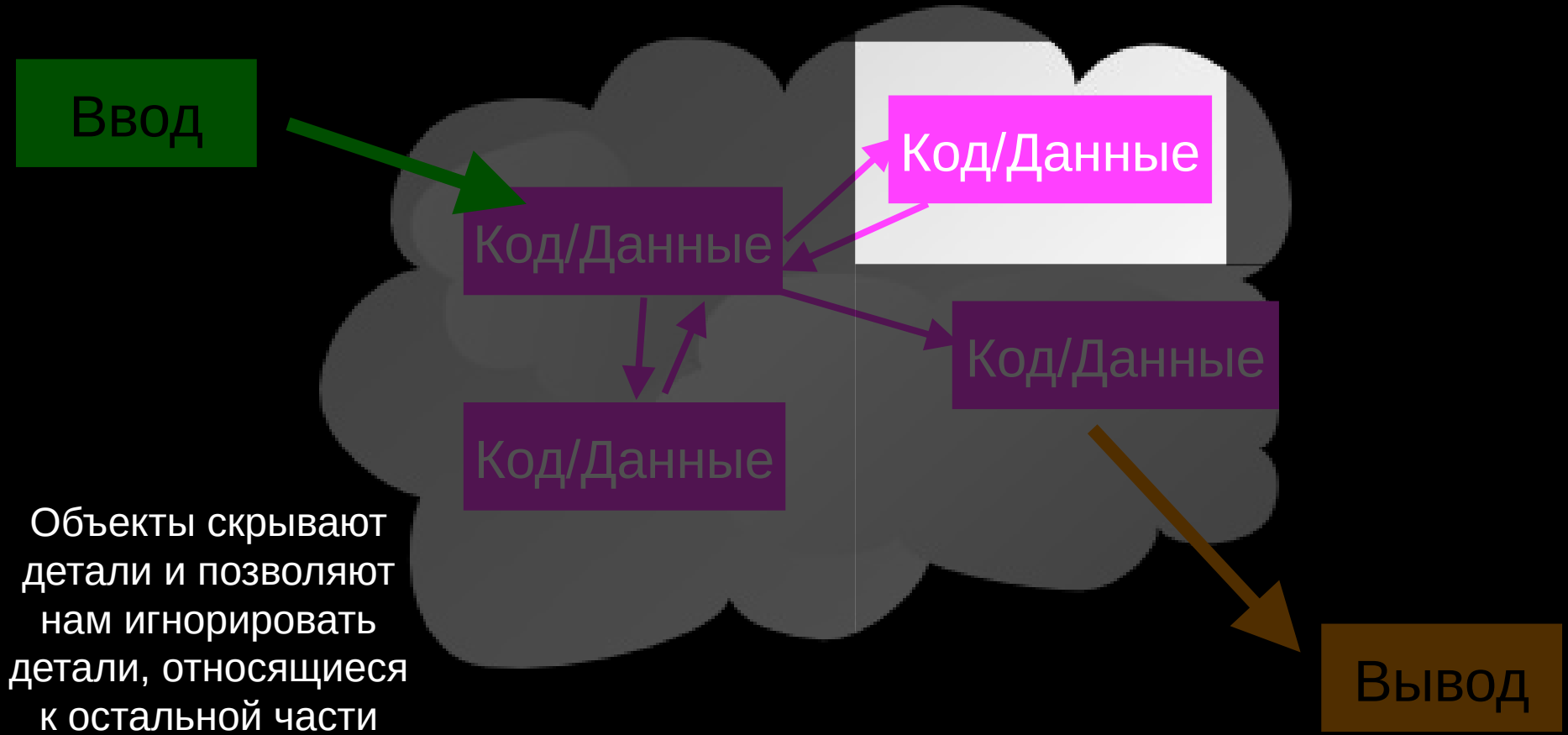
- Объект — это сущность, содержащая Код и Данные
- Ключевой аспект объектного подхода — разделить проблему на маленькие понятные части («Разделяй и властвуй»)
- Ограничения, свойственные объектам, позволяют игнорировать ненужные детали
- Все это время мы использовали объекты: Строковые, Числовые, Словари, Списки…



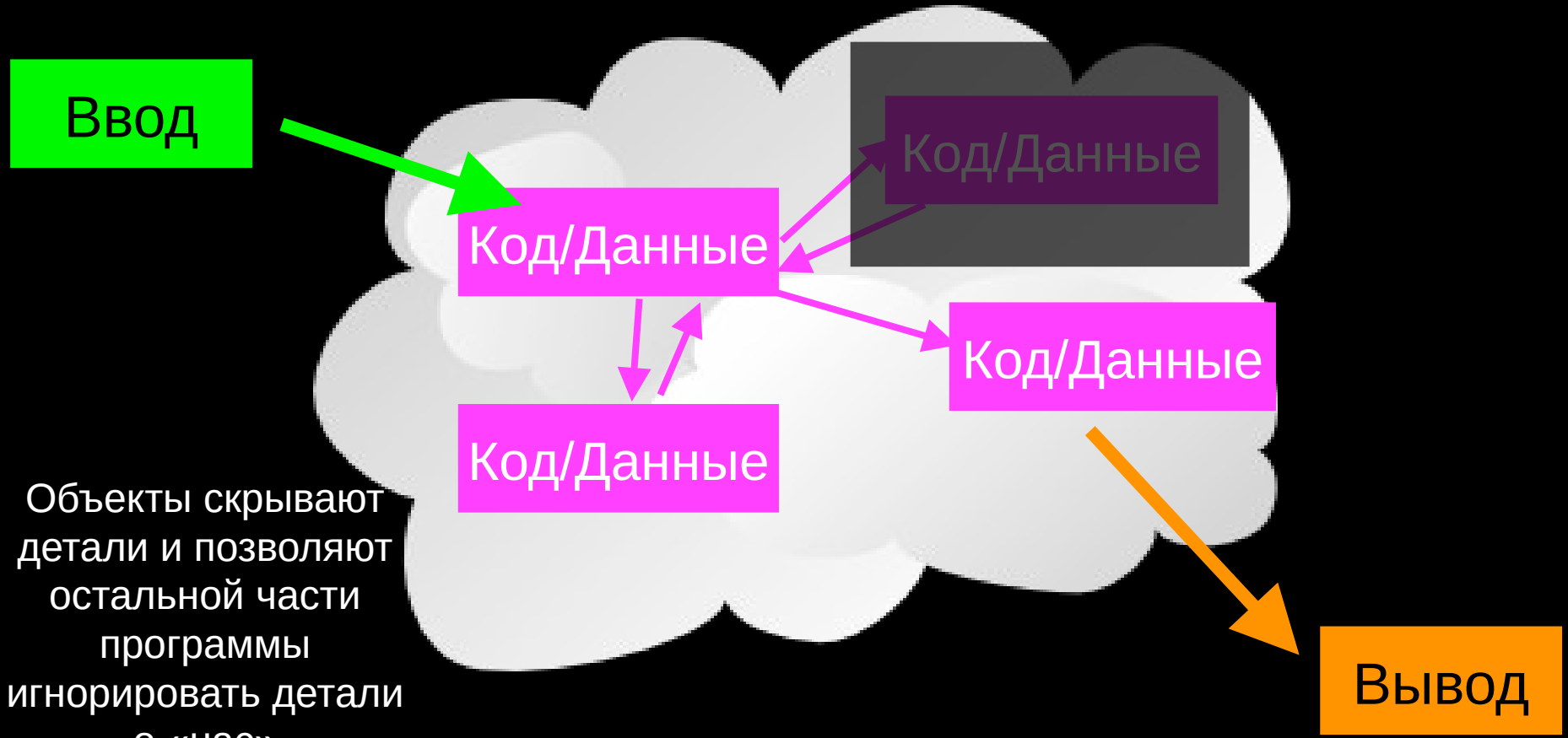
Объекты создаются и используются



Объекты — биты кода и данных



Объекты скрывают детали и позволяют нам игнорировать детали, относящиеся к остальной части программы



Объекты скрывают детали и позволяют остальной части программы игнорировать детали о «нас»

Определения

- Класс — это шаблон
- Метод или Сообщение — определенные возможности класса
- Поле класса или Атрибут — данные внутри класса
- Объект или Экземпляр — конкретный экземпляр класса

Терминология: Класс

Описывает абстрактные свойства объекта, включая характеристики (атрибуты, поля или свойства), и поведение объекта (то, что объект может делать или методы, операции и дополнительные возможности). Можно сказать, что класс — это шаблон, который описывает природу чего-либо. Например, класс «Собака» будет содержать черты, присущие всем собакам, такие как порода или цвет шерсти (характеристики), а также способность лаять и сидеть (поведение).

https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование

Терминология: Экземпляр

Можно создать экземпляр класса или конкретного объекта. Экземпляр — это фактический объект, созданный во время выполнения. На жаргоне программистов объект Лесси — это экземпляр класса «Собака». Набор значений атрибутов конкретного объекта называется состоянием. Объект состоит из состояния и поведения, которые определены в классе объекта.

Термины «Объект» и «Экземпляр» взаимозаменяемы.

https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование

Терминология: Метод

Способности объекта. С точки зрения языка, методы — это глаголы. Лесси, будучи Собакой, имеет способность лаять. Так что лай() — один из методов Лесси. У нее могут быть и другие методы, например, сидеть(), есть(), идти() или защищать_хозяина(). В программе использование метода обычно влияет только на один конкретный объект; лаять умеют все Собаки, но вам нужно чтобы лаяла какая-то одна.

Термины «Метод» и «Сообщение» взаимозаменяемы.

https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование

Некоторые объекты Пайтон

```
>>> x = 'abc'
>>> type(x)
<class 'str'>
>>> type(2.5)
<class 'float'>
>>> type(2)
<class 'int'>
>>> y = list()
>>> type(y)
<class 'list'>
>>> z = dict()
>>> type(z)
<class 'dict'>
```

```
>>> dir(x)
[ … 'capitalize', 'casefold', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find',
'format', … 'lower', 'lstrip', 'maketrans',
'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
>>> dir(y)
[… 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
>>> dir(z)
[…, 'clear', 'copy', 'fromkeys', 'get', 'items',
'keys', 'pop', 'popitem', 'setdefault', 'update',
'values']
```

Образец класса

«class» — зарезервированное слово

Каждый объект PartyAnimal содержит некоторое количество кода

Указывает объекту «an» запустить внутри него код party()

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print("Сейчас ",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

Шаблон создания объектов PartyAnimal

Каждый объект PartyAnimal содержит некоторое количество данных

Создает объект PartyAnimal и хранит его в переменной «an»

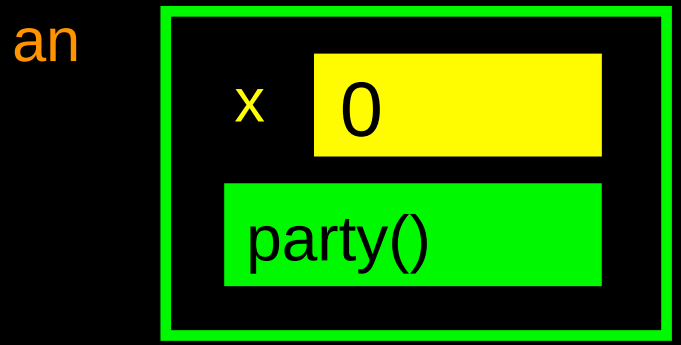
PartyAnimal.party(an)

```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print("Сейчас ",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

```
$ python party1.py
```

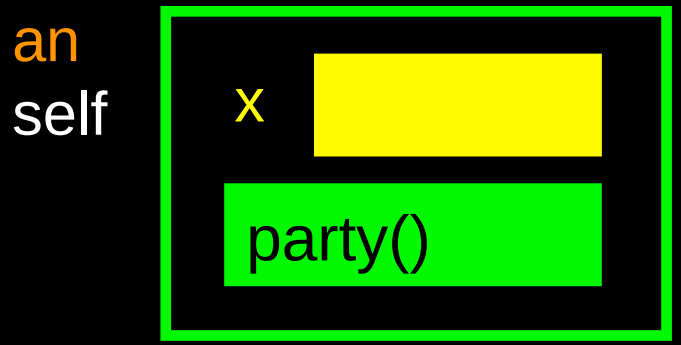
```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print("Сейчас ",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

```
$ python party1.py
```



```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print("Сейчас ",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

```
$ python party1.py
Сейчас 1
Сейчас 2
Сейчас 3
```



PartyAnimal.party(an)

Забавы с dir() и type()

«Ботанский» способ нахождения возможностей объекта

- Функция dir() возвращает список возможностей объекта
- Игнорируйте те, что с подчеркиванием, это специальные методы Пайтон
- Операции, которые объект может производить
- Похоже на функцию type(), которая сообщает нам некую *информацию* о переменной

```
>>> y = list()
>>> type(y)
<class 'list'>
>>> dir(y)
['__add__', '__class__',
'__contains__', '__delattr__',
'__delitem__', '__delslice__',
'__doc__', … '__setitem__',
'__setslice__', '__str__',
'append', 'clear', 'copy',
'count', 'extend', 'index',
'insert', 'pop', 'remove',
'reverse', 'sort']
>>>
```

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print("Сейчас ",self.x)

an = PartyAnimal()

print("Тип ", type(an))
print("Dir ", dir(an))
```

Можем использовать dir(), чтобы узнать «возможности» созданного нами класса.

```
$ python party3.py
Тип <class '__main__.PartyAnimal'>
Dir  ['__class__', ... 'party', 'x']
```

Пробуем применить dir() к строке

```
>>> x = 'Hello there'
>>> dir(x)
['__add__', '__class__', '__contains__', '__delattr__',
'__doc__', '__eq__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__getslice__', '__gt__',
'__hash__', '__init__', '__le__', '__len__', '__lt__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__str__',
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith',
'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit',
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

Жизненный цикл объекта

https://ru.wikipedia.org/wiki/Конструктор_(объектно-ориентированное_программирование)

Жизненный цикл объекта

- Объекты создаются, используются и уничтожаются
- У нас есть специальные блоки кода (методы), которые вызываются:
  - В момент создания (конструктор);
  - В момент уничтожения (деструктор).
- Конструкторы используются очень часто
- Деструкторы используются редко

Конструктор

Основная задача конструктора — установить значения переменных экземпляра, чтобы при создании объекта они имели правильные начальные значения.

```
class PartyAnimal:
   x = 0

   def __init__(self):
     print('Я создан')

   def party(self) :
     self.x = self.x + 1
     print('Сейчас',self.x)

   def __del__(self):
     print('Я уничтожен', self.x)

an = PartyAnimal()
an.party()
an.party()
an = 42
print('an содержит',an)
```

```
$ python party4.py
Я создан
Сейчас 1
Сейчас 2
Я уничтожен 2
an содержит 42
```

Использование конструктора и деструктора не является обязательным. Конструктор обычно используется для настройки переменных.
Деструктор используется редко.

Конструктор

В объектно-ориентированном программировании конструктор класса — это специальный блок инструкций, вызываемый при создании объекта

https://ru.wikipedia.org/wiki/Конструктор_(объектно-ориентированное_программирование)

Множество экземпляров

- Можно создать много объектов, класс — это шаблон для объекта
- Каждый отдельный объект может храниться в собственной переменной
- Мы называем это множественными экземплярами класса
- Каждый экземпляр имеет собственную копию переменных экземпляра

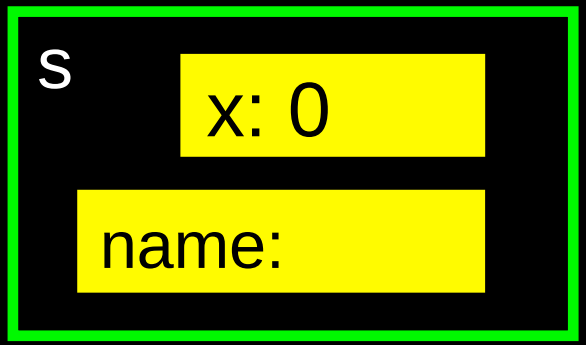
```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

Конструкторы могут иметь дополнительные параметры. Их можно использовать для настройки переменных экземпляра для конкретного экземпляра класса (т.е., для конкретного объекта).

party5.py

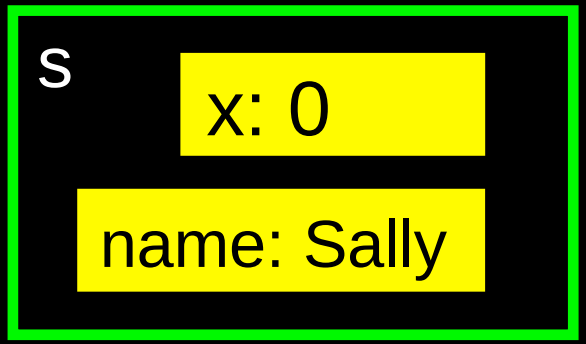
```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

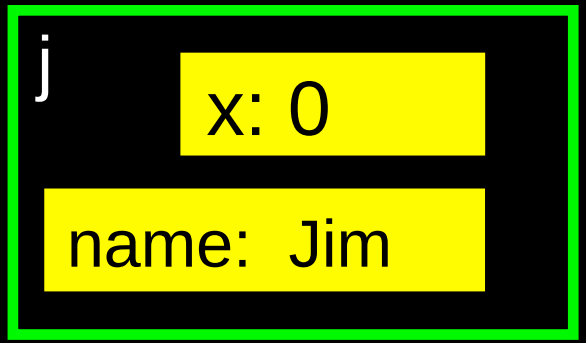
```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```



```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

У нас есть два независимых экземпляра





```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

```
Sally constructed
Jim constructed
Sally party count 1
Jim party count 1
Sally party count 2
```

Наследование

http://www.ibiblio.org/g2swap/byteofpython/read/inheritance.html

Наследование

- Создавая новый класс, мы можем использовать уже существующий и унаследовать все возможности этого класса, а затем добавить в наш новый класс что-нибудь еще
- Еще одна форма хранения и повторного использования
- Написав один раз, можно многократно использовать снова
- Новый класс (потомок) обладает всеми свойствами и методами родительского класса (родитель), а затем, если необходимо, их можно добавить или переопределить

Терминология: Наследование

«Подкласс» - производная от класса, которая наследует атрибуты и поведение своего родителя (суперкласс), и может вводить собственные методы и атрибуты.

https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование

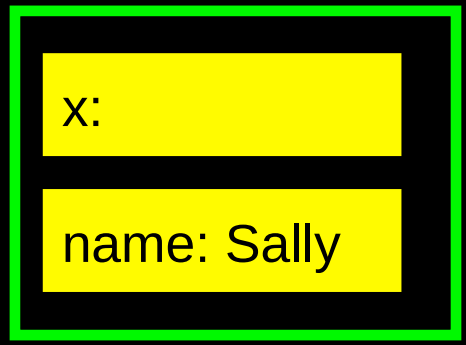
```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name,"constructed")

   def party(self) :
     self.x = self.x + 1
     print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

FootballFan — подкласс суперкласса PartyAnimal. В нем есть все возможности родительского класса PartyAnimal, плюс немного больше.

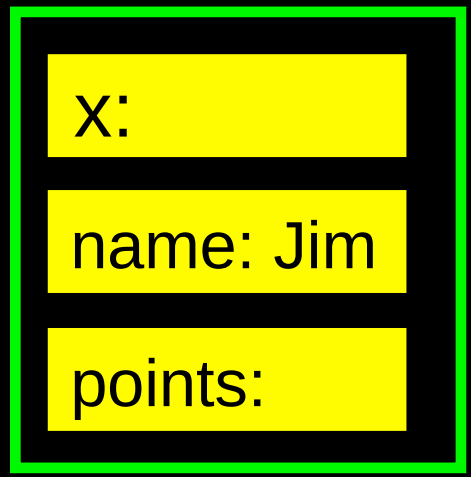
```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name,"constructed")

   def party(self) :
     self.x = self.x + 1
     print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```



```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name,"constructed")

   def party(self) :
     self.x = self.x + 1
     print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```



Определения

- Класс — шаблон
- Атрибут — переменная внутри класса
- Метод — функция внутри класса
- Объект — отдельный экземпляр класса
- Конструктор — код, который выполняется при создании объекта
- Наследование — возможность расширить класс для создания нового класса

Резюме

- Объектно-ориентированное программирование представляет собой очень структурированный подход к повторному использованию кода
- Мы можем сгруппировать вместе данные и функциональность, и создать множество независимых экземпляров класса

Авторы / Благодарности

… Insert new Contributors and Translations here

Исходная разработка: Чарльз Северанс, Школа Информации Мичиганского Университета.

Перевод выполнила Фомкина Виолетта.

… Добавьте сюда новых авторов и переводчиков

...

Дополнительная информация

- Форма для печенья «Снеговик» от Didriks, лицензия CC (Creative Commons) https://www.flickr.com/photos/dinnerseries/23570475099

- Фотография из телесериала «Лесси». Лесси наблюдает как Джефф (Томми Реттиг) возится с велосипедом https://en.wikipedia.org/wiki/Lassie#/media/File:Lassie_and_Tommy_Rettig_1956.JPG