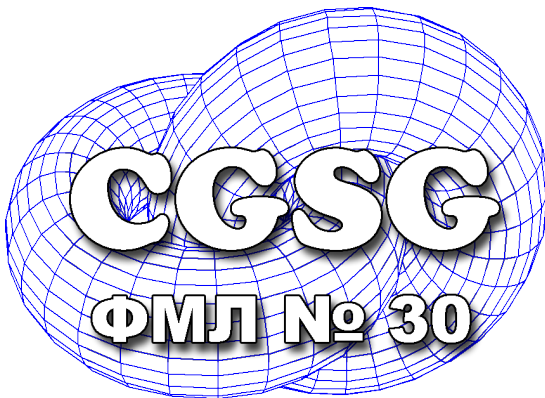


Язык программирования



Лекция 2. ООП, области видимости



Владимир Руцкий
<altsysrq@gmail.com>



План лекции

- Повторение
- Принципы ООП
- Области видимости

Ссылки

- Презентации будут доступны на сайте школы:
<http://school30.spb.ru/cgsg/python/>
- Рабочие версии презентаций, использованные материалы и примеры можно найти в моём git: <https://github.com/rutsky/python-course>
- Как скачать:
 - выберите файл
 - нажмите «view raw» или «raw»
 - скопируйте или сохраните предложенный файл
- Презентации:
 - 00_introduction.pdf
 - 01_oop_scopes.pdf
- Примеры и материалы в соответствующих директориях:
 - 00_introduction/...
 - 01_oop_scopes/...

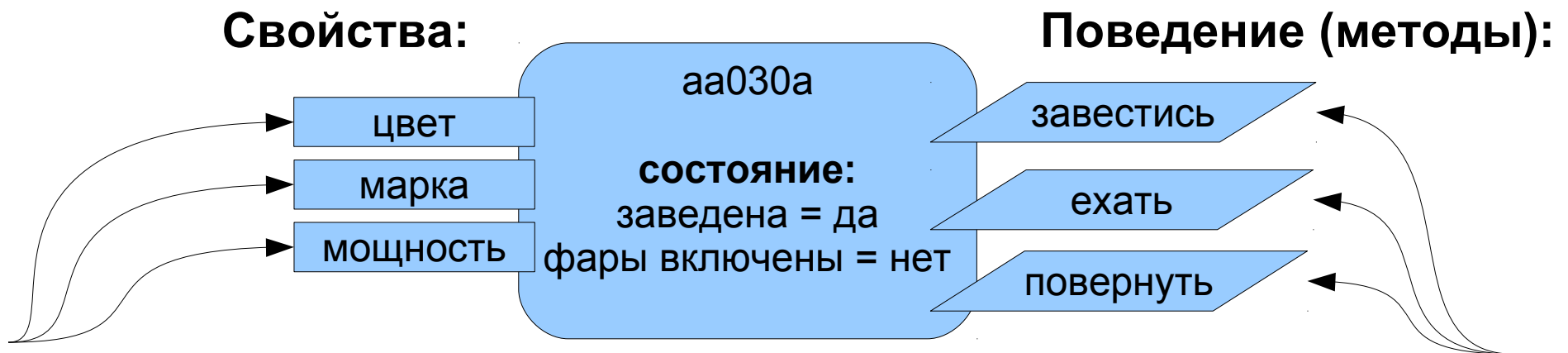
Повторение

Объектно ориентированное программирование (ООП)

- ООП - *парадигма* программирования - совокупность идей и понятий, определяющих стиль написания программ
- Основные понятия: *объект, класс, абстракция, наследование, инкапсуляция, полиморфизм*

Объекты (1/2)

- **Объект** - сущность, обладающая определённым состоянием, поведением и свойствами
 - Объект «автомобиль с номером aa030a» (конкретный)
 - **Внешний интерфейс** (доступен всем пользователям):
 - свойства: «цвет», «марка», «мощность двигателя», «количество мест»
 - поведение (функции): «завестись», «ехать», «повернуть», «включить фары»
 - **Внутреннее состояние** (доступно только объекту):
 - «заведена», «включены фары», «положение роторов», «напряжение на контурах»



Объекты (2/2)

- Объект «водитель Пётр» взаимодействует с объектом «автомобиль aa030a» посредством **внешнего интерфейса**
 - Пётр нажимает педали, крутит руль, получает информацию о цвете и марке автомобиля
- Внешний объект «Пётр» **не может влиять на внутреннее состояние** объекта «автомобиль aa030a»
 - Механизм изменения внутреннего состояния может быть различным у разных объектов-автомобилей
 - Прямое изменение внутреннего состояния внешними объектами скорее всего приведёт к поломке системы (объекта «автомобиль aa030a»)
 - Говорят, что объект **инкапсулирует** свои внутренние свойства
- Объекты состоят из **внешнего интерфейса** и **внутренней реализации**
 - Обеспечивает гибкость - возможность свободного изменения внутренней реализации без боязни что-то сломать
 - Обеспечивает консистентность (согласованность) - объект сам меняет своё внутреннее состояние и обеспечивает его корректность

Классы

- **Класс** - совокупность объектов (экземпляров **класса**), объединённых общими свойствами и поведением
 - Класс «Автомобили» - совокупность объектов, имеющих
 - поведения: «завестись», «ехать», «повернуть»
 - свойства: «марка», «цвет», «количество мест»
 - Класс «Велосипеды»:
 - поведение: «сесть на велосипед», «ехать», «повернуть», «поднять велосипед»
 - свойства: «марка», «цвет», «масса»
 - Класс «Транспортные средства»:
 - поведения: «ехать», «повернуть»
- Класс «Транспортные средства» содержит в себе классы «Автомобили» и «Велосипеды»
- Классы «Автомобили» и «Велосипеды» **наследуют свойства и поведение** класса «Транспортные средства»

родительский
(базовый)
класс

Транспортные средства

Автомобили

Велосипеды

дочерние (производные)
классы (наследники)

Любой экземпляр класса «Автомобиль» является также экземпляром класса «Транспортное средство»

Абстракция и полиморфизм

- **Абстрагирование** - выделение значимых свойств, опуская незначимые
 - «Транспортное средство» - абстракция, такого объекта не существует
- При наследовании реализация метода может быть изменена - **полиморфизм**
 - Рассмотрим класс «Автомобиль Лада Калина»
 - Создадим **производный** от класса «Автомобиль Лада Калина» класс «Автомобиль Лада Калина с двигателем от Ford», в котором изменим **внутреннюю реализацию** методов «завестись» и «поехать» для двигателя от Ford
 - Новые автомобили, экземпляры «Автомобиль Лада Калина с двигателем от Ford», поддерживают интерфейс класса «Автомобиль Лада Калина», но имеют **изменённую (полиморфную)** реализацию

Области видимости Python (1/2)

```
1 >>> # Все переменные в Python определяются в каком-то модуле. Создаваемые в интерактивной консоли переменные кладутся в модуль __main__.
2 ... # Для поиска по имени переменной объекта, на который она ссылается, Python использует специальные словари --- области видимости.
3 ... # Переменные, определяемые на уровне модуля, добавляются в глобальную область видимости. Переменные, определяемые внутри функции или
4 ... # класса сохраняются в локальную область видимости. В любой строке кода можно получить ссылки на локальную и глобальную область
5 ... # видимости с помощью функций locals() и globals():
6 ... globals() # вернёт словарь, отображающий имена переменных в объекты, на которые они ссылаются
7 {'__builtins__': <module '__builtin__' (built-in)>, '__name__': '__main__', '__doc__': None, '__package__': None}
8 >>> __name__ # имя текущего модуля
9 '__main__'
10 >>> # При указании в коде имени переменной, Python ищет имя переменной в областях видимости в следующем порядке: в локальной, в глобальной
11 ... # и во встроенной в Python областях видимости.
12 ... # Имя функции 'globals' находится во встроенной области видимости, а имя '__name__' - в глобальной
13 ... # Т.о. доступ к __name__ эквивалентен в этом случае:
14 ... globals()['__name__']
15 '__main__'
16 >>> # При написания кода на уровне модуля, локальная область видимости совпадает с глобальной:
17 ... locals() is globals()
18 True
19 >>> # Определим переменную. При этом Python добавит в текущую локальную область видимости пару с ключом 'имя переменной' и значением объектом
20 ... a = "test"
21 >>> locals()
22 {'__builtins__': <module '__builtin__' (built-in)>, '__name__': '__main__', '__doc__': None, 'a': 'test', '__package__': None}
23 >>> # Удалим имя переменной (реальный объект удалится при сборке мусора, но об этом потом):
24 ... del a
25 >>> locals()
26 {'__builtins__': <module '__builtin__' (built-in)>, '__name__': '__main__', '__doc__': None, '__package__': None}
27 >>>
```

Области видимости Python (2/2)

```
1 >>> # Оператор `def` создаёт объект-функцию и добавляет её в локальную область видимости:
2 ... def f(arg):
3 ...     v = 30
4 ...     return locals(), globals() # вернём локальную и глобальную о.в. для данной строки
5 ...
6 >>> # Имя `f` занеслось в локальную о.в.
7 ... locals()
8 {'f': <function f at 0x24a2848>, '__builtins__': <module '__builtin__' (built-in)>, '__package__': None, '__name__': '__main__', '__doc__': None}
9 >>> f("aaa")[0] # locals() из внутреннейности f()
10 {'arg': 'aaa', 'v': 30}
11 >>> # globals() из f() совпадает с globals из данного модуля
12 ... globals() is f("bbb")[1]
13 True
14 >>> # При каждом вызове функции, для тела функции создаётся новая локальная о.в.
15 ... # Т.к. всё, что делает оператор `def` это создание нового объекта и помещение в локальную
16 ... # область видимости, то можно делать вложенные функции:
17 ... def f1(arg):
18 ...     s1 = "f1() local variable"
19 ...     def f2(arg2):
20 ...         return arg2, s1, some_glb_var # arg2 найдётся в локальной о.в., s1 во внешней локальной,
21 ...         # а some_glb_var будет искаться в глобальной о.в.
22 ...     return f2(30)
23 ...
24 >>> f1("test")
25 Traceback (most recent call last):
26   File "<stdin>", line 1, in <module>
27   File "<stdin>", line 9, in f1
28   File "<stdin>", line 7, in f2
29 NameError: global name 'some_glb_var' is not defined
30 >>> some_glb_var = "global variable"
31 >>> f1("test")
32 (30, 'f1() local variable', 'global variable')
33 >>>
```