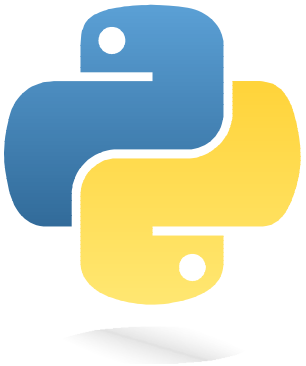
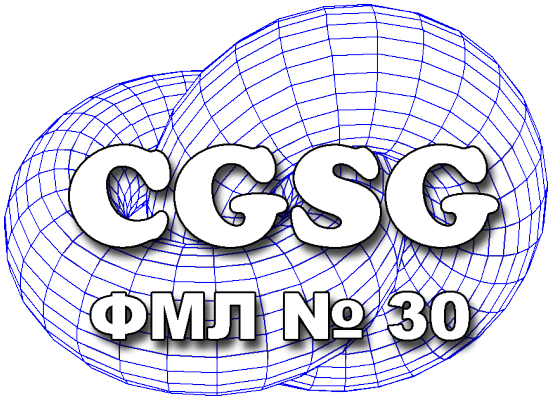
Язык программирования



python™

Лекция 3. Классы, модули



Владимир Руцкий
<altsysrq@gmail.com>

План лекции

- Классы
- Модули
- Практика

Классы в Python

```
>>> # Класс определяется с помощью оператора `class':
... # class ИмяКласса:
... #     выражение1
... #     выражение2
... #     ...
... class MyClass:
...     def f(self):
...         return 'Hello!'
...
>>> MyClass # Оператор class создал новый класс
<class __main__.MyClass at 0x10bce88>
>>> x = MyClass() # `вызов' класса --- создание экземпляра класса
>>> x       # - экземпляр (instance) класса MyClass
<__main__.MyClass instance at 0x11247e8>
>>> x.f()   # вызываем метод класса
'Hello!'
>>>
```

Классы в Python 2

```
>>> class User:
...     """Класс пользователя (это docstring, необязательный)"""
...     # Все методы класса принимают первым аргументом экземпляр класса `self'
...     def __init__(self, name):
...         # Конструктор класса --- этот метод вызывается при инициализации
...         # вновь созданного экземпляра класса.
...         # Первый аргумент `self' --- экземпляр класса (кого инициализируем)
...         self.my_name = name # --- записываем в атрибут экземпляра класса с
...                             # именем `car_name' значение переменной `name'
...     def hello(self):
...         # Определим метод класса hello()
...         return "Hello, my name is " + self.my_name + "!"
...
>>> # Создаём экземпляр класса (в конструктор передаётся name="Peter")
... user_instance = User("Peter")
>>> user_instance.my_name # члены экземпляра доступны через точку
'Peter'
>>> user_instance.hello()
'Hello, my name is Peter!'
>>>
```

Статические члены

```
>>> class User:
...     # Объявления в классе являются статическими, т.е. общими для всех
...     # экземпляров класса.
...     greeting = "Hello " # статический член
...     def __init__(self, name):
...         self.my_name = name
...     def hello(self):
...         return self.greeting + self.my_name
...
>>> peter = User("Peter")
>>> sam = User("Sam")
>>> peter.hello()
'Hello Peter'
>>> sam.hello()
'Hello Sam'
>>> # Изменим статический член User.greeting
... User.greeting = "Hi "
>>> peter.hello()
'Hi Peter'
>>> sam.hello()
'Hi Sam'
>>> def new_hello(self):
...     return "New hello() called with greeting `" + self.greeting + \
...         "' and name `" + self.my_name + "'"
...
>>> # Изменим статический член класса функцию hello():
... User.hello = new_hello
>>> peter.hello()
"New hello() called with greeting `Hi ' and name `Peter'"
>>> sam.hello()
"New hello() called with greeting `Hi ' and name `Sam'"
>>>
```

Наследование классов

```
>>> # В Python есть множественное наследоваение классов:
... # class ИмяКласса(ИмяБазовогоКласса1, ИмяБазовогоКласса2, ...):
... #     выражение1
... #     выражение2
... #     ...
... class UserWithSurname(User): # Наследуем свойства класса User
...     def __init__(self, name, surname):
...         # Вызываем конструктор базового класса с необходимыми
...         # аргументами
...         User.__init__(self, name) # он присвоит в self.my_name = name
...         self.my_surname = surname
...     def hello(self): # hello() переопределяется
...         return self.greeting + self.my_name + " " + self.my_surname
...     def old_hello(self):
...         # Явно вызываем метод базового класса
...         return User.hello(self)
...
>>> peter = UserWithSurname("Peter", "Ivanov")
>>> peter.hello()
'Hello Peter Ivanov'
>>> peter.old_hello()
'Hello Peter'
>>>
```

Кратко о дополнительных возможностях классов

- Члены класса, не являющиеся частью публичного интерфейса, принято именовать с подчеркивания: «self._some_internal_list»
- Члены класса, начинающиеся с двух подчеркиваний «self.__my_var» будут преобразованы в «self._classname__my_var»
- В классах можно определять специальные методы для перегрузки операций: __call__(), __str__(), __add__(), __mul__(), ...

Модули

Объявление функций, классов и переменных можно выделить в отдельный модуль, **user.py**:

```
1 # -*- coding: utf-8 -*-
2
3 class User:
4     """Класс пользователя"""
5     greeting = "Hello "
6     def __init__(self, name):
7         self.name = name
8     def hello(self):
9         return self.greeting + self.name
```

Затем использовать их «импортировав» модуль

```
1 >>> import user
2 >>> dir(user)
3 ['User', '__builtins__', '__doc__', '__file__', '__name__', '__package__']
4 >>> peter = user.User("Peter")
5 >>> peter.hello()
6 'Hello Peter'
7 >>>
```

Модули 2

- Можно импортировать отдельные объекты модуля:

```
from user import User
peter = User(«Peter»)
```

- Можно переименовывать импортируемые объекты:

```
from user import User as UserClass
peter = UserClass(«Peter»)
```

- или

```
import user as user_module
peter = user_module.User(«Peter»)
```

- Можно импортировать несколько модулей:

```
import user, math
print math.cos(math.pi)
```

- или объектов:

```
from math import cos, pi
print cos(pi)
```

Пакеты

- Модули можно объединять в пакеты:

```
sound/                          Пакет верхнего уровня
    __init__.py                 Инициализация пакета работы со звуком (sound)
    formats/                    Подпакет для конвертирования форматов файлов
        __init__.py
        wavread.py              (чтение wav)
        wavwrite.py             (запись wav)
    effects/                    Подпакет для звуковых эффектов
        __init__.py
        echo.py                 ( эхо )
        surround.py             ( окружение )
        reverse.py              ( обращение )
```

- Использование:

```
import sound.effects.echo
sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
```

- или

```
from sound.effects import echo
echo.echofilter(input, output, delay=0.7, atten=4)
```

Практика 1

1. Запустить интерпретатор
2. Вывести «Hello, world!»
3. Вычислить выражение

$$\sqrt{12}\left(1-\frac{1}{3\cdot 3^{1}}+\frac{1}{5\cdot 3^{2}}-\frac{1}{7\cdot 3^{3}}+\frac{1}{9\cdot 3^{4}}\right)$$

4. Создать переменную «a» со значением 2
5. Присвоить в «a» «a в степени 512»
6. Создать строку «s», со значением из «a»
7. Найти длину «s»
8. Найти символ в 101-й позиции (индексируя с нуля)
9. Найти подстроку со 101-й позиции по 110-ю (110ю не включать)

Практика 1. Ответы

1. Запустить интерпретатор
2. Вывести «Hello, world!»
3. Вычислить выражение

$$\sqrt{12}\left(1-\frac{1}{3\cdot 3^{1}}+\frac{1}{5\cdot 3^{2}}-\frac{1}{7\cdot 3^{3}}+\frac{1}{9\cdot 3^{4}}\right)$$

4. Создать переменную «a» со значением 2
5. Присвоить в «a» «a в степени 512»
6. Создать строку «s», со значением из «a»
7. Найти длину «s»
8. Найти символ в 101-й позиции (индексируя с нуля)
9. Найти подстроку со 101-й позиции по 110-ю (110ю не включать)

1. python.exe
2. print "Hello, world!"
3. 12**0.5 * (1 - 1.0 / (3 * 3) + 1.0 / (5 * 3**2) - 1.0 / (7 * 3**3) + 1.0 / (9 * 3**4)) # 3.142604745663084 6
4. a = 2
5. a = a ** 512
6. s = str(a)
7. len(s) # 155
8. s[101] # '6'
9. s[101:110] # '690031858'

Практика 2

1. Присвоить в «a» список из элементов ["a", "b", "c"]
2. Присвоить в «b» список из 5 повторений «a», должно получиться ["a", "b", "c", "a", "b", "c", ..., "a", "b", "c"]
3. Вывести каждый третий элемент «b», должно получиться ['a', 'a', ..., 'a']
4. Вычислить длину списка из вопроса 3.
5. Создайте список «d» из [0, ..., 9] с помощью функции range
6. Добавьте в конец «b» элемент 'end' с помощью метода списка .append()
7. Добавьте в начало «b» элемент 'begin' с помощью метода списка .insert(index, elem)
8. Замените середину «b» на часть «d» без крайних элементов так, чтобы получился список ['begin', 1, 2, 3, 4, 5, 6, 7, 8, 'end']

Практика 2. Ответы

1. Присвоить в «a» список из элементов ["a", "b", "c"]
2. Присвоить в «b» список из 5 повторений «a», должно получиться ["a", "b", "c", "a", "b", "c", ..., "a", "b", "c"]
3. Вывести каждый третий элемент «b», должно получиться ['a', 'a', ..., 'a']
4. Вычислить длину списка из вопроса 3.
5. Создайте список «d» из [0, ..., 9] с помощью функции range
6. Добавьте в конец «b» элемент 'end' с помощью метода списка .append()
7. Добавьте в начало «b» элемент 'begin' с помощью метода списка .insert(index, elem)
8. Замените середину «b» на часть «d» без крайних элементов так, чтобы получился список ['begin', 1, 2, 3, 4, 5, 6, 7, 8, 'end']

```
1.a = ["a", "b", "c"]
2.b = a * 5
3.b[::3]
4.len(b[::3]) # 5
5.d = range(10)
6.b.append("end")
7.b.insert(0, 'begin')
8.b[1:-1] = d[1:-1]
```

List comprehension

- List comprehension - способ преобразования последовательности
  - [выражение **for** переменная **in** послед.]
  - [выражение **for** переменная **in** послед. **if** условие]
- Если использовать круглые скобки, то будет генерироваться кортеж:
  - (выражение **for** переменная **in** послед.)
- Примеры:
  - Вывести квадраты чисел от 0 до 9:

    ```
    >>> [x**2 for x in range(10)]
    [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
    ```
  - Вывести квадраты четных чисел от 0 до 9:

    ```
    >>> [x**2 for x in range(10) if x % 2 == 0]
    [0, 4, 16, 36, 64]
    ```

Работа со списками

- **zip**() - от англ. «zip» - «застёжка молния» - соединить два списка, как зубцы молнии
  - zip(посл.1, посл.2, ...) -> список кортежей

    >>> **zip**([1, 2, 3], ['a', 'b', 'c'])

    [(1, 'a'), (2, 'b'), (3, 'c')]
- **map**(функция, посл.) - применить функцию к каждому элементу последовательности

  >>> map(len, ['a', 'abcd', 'qqq'])

  [1, 4, 3]

Форматированный вывод

- Подробное описание: http://docs.python.org/library/string.html#formatspec
- У строки есть метод **.format()**, позволяющий создать форматированную строку из текущей

```
>>> "Hello, {0}".format("Peter")
'Hello, Peter'
>>> "Hello, {0} {1}. Or maybe {1} {0}?".format("John", "Smith")
'Hello, John Smith. Or maybe Smith John?'
>>> "Hello, {} {} {}".format(1, 2, 3)
'Hello, 1 2 3'
>>> "Object: {0} - size is {size}, weight is {w}".format("potato", size=10, w="0.1 kg")
'Object: potato - size is 10, weight is 0.1 kg'
>>> "int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".format(30)
'int: 30;  hex: 1e;  oct: 36;  bin: 11110'
>>> "{:10.3}".format(3.1415926653589793)
'      3.14'
>>>
```