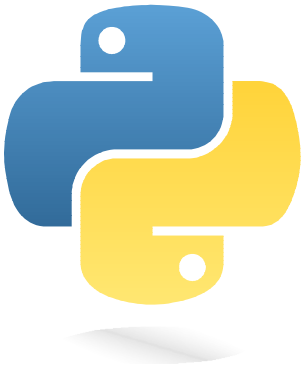
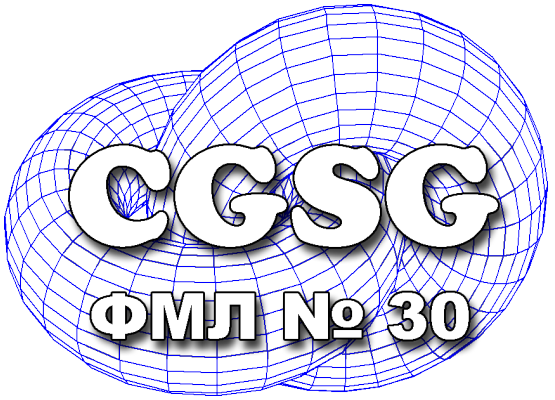
Язык программирования



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