

Python: classes

budnyjj@pirates.by

- ▶ ООП
 - ▶ Инкапсуляция
 - ▶ Наследование
 - ▶ Полиморфизм
- ▶ Работа с исключениями

Объект — сущность, состоящая из

- ▶ данных (полей, атрибутов) и
- ▶ методов обработки этих данных (методов объекта).

Пример объекта

Кроме прочих своих достоинств, кот

- ▶ демонстрирует характерное поведение,
- ▶ реагирует на сообщения,
- ▶ наделён унаследованными реакциями,
- ▶ управляет своим, вполне независимым, внутренним состоянием.¹



¹Roger King, My cat is object-oriented

Объектно-ориентированный подход хорош там, где проект подразумевает долгосрочное развитие, состоит из большого количества библиотек и внутренних связей.

Объявление класса [dataClass.py]

```
1 class exampleDataClass:
2     """A data example class"""
3
4     cls_var = "cls_data"
5
6     def __init__(self):
7         self.obj_var = "obj_data"
8
9     def f(self):
10         return self.obj_var
11
```

Атрибуты объекта [dataExample1.py]

```
1 import dataClass as dc
2
3 o1 = dc.exampleDataClass();
4 o2 = dc.exampleDataClass();
5
6 print "obj_var is object data member:"
7 print "BEFORE:"
8 print "o1.obj_var: ", o1.obj_var
9 print "o2.obj_var:", o2.obj_var
10
11 o1.obj_var = "new_object_data"
12
13 print "AFTER:"
14 print "o1.obj_var:", o1.obj_var
15 print "o2.obj_var:", o2.obj_var
16
```

```
class exampleDataClass:
    cls_var = "cls_data"

    def __init__(self):
        self.obj_var =
            "obj_data"

    def f(self):
        return self.obj_var
```

Атрибуты класса [dataExample2.py]

```
1 import dataClass as dc
2
3 o1 = dc.exampleDataClass();
4 o2 = dc.exampleDataClass();
5
6 print "Data is object data member:"
7 print "BEFORE:"
8 print "o1.cls_var:", o1.cls_var
9 print "o2.cls_var:", o2.cls_var
10
11 dc.exampleDataClass.cls_var = \
12     "new_object_data"
13
14 print "AFTER:"
15 print "o1.cls_var: ", o1.cls_var
16 print "o2.cls_var:", o2.cls_var
17
```

```
class exampleDataClass:
    cls_var = "cls_data"

    def __init__(self):
        self.obj_var =
            "obj_data"

    def f(self):
        return self.obj_var
```


Изменение атрибутов объекта

```
getattr(object, name)  
setattr(object, name, value)  
delattr(object, name)  
hasattr(object, name)
```

```
var = object.name  
object.name = value  
del object.name
```

implemented by calling `getattr(object, name)`
and seeing whether it raises an exception or not

Изменение атрибутов объекта [dataExample3.py]

```
1 import dataClass as dc
2
3 o = dc.exampleDataClass();
4 print dir(o);
5
6 del o.obj_var
7 o.new_data = "some_data"
8 o.f2 = lambda x: x * 3
9
10 print dir(o);
11 print o.new_data
12 print o.f2("M")
13
```

```
class exampleDataClass:
    cls_var = "cls_data"

    def __init__(self):
        self.obj_var =
            "obj_data"

    def f(self):
        return self.obj_var
```

Инкапсуляция

```
class Simple:
    """ Simple class with private attribute """
    def __init__(self, count, str):
        self.__private_attr = 20
        print self.__private_attr

s = Simple(1, '22')
print s.__private_attr
```

Наследование

Синтаксис:

```
class Derived(Base):
```

```
    pass
```

```
class Derived(module_name.Base):
```

```
    pass
```

```
class Derived(Base1, Base2, Base3):
```

```
    pass
```

Особенности:

- ▶ Все методы — виртуальные;
- ▶ Вызов метода базового класса: `Base.method()`;
- ▶ Порядок поиска атрибута:
 1. Derived,
 2. Base1, затем рекурсивно в базовых классах,
 3. Base2, затем рекурсивно в базовых классах ...

- ▶ `type(object);`
- ▶ `isinstance(object, classinfo);`
- ▶ `issubclass(class, classinfo).`

Встроенные методы классов

- ▶ `__name__`
- ▶ `__module__`
- ▶ `__dict__`
- ▶ `__bases__`
- ▶ `__doc__`

Встроенные методы объектов

- ▶ `__dict__`
- ▶ `__class__`
- ▶ `__init__`
- ▶ `__del__`
- ▶ `__cmp__`
- ▶ `__hash__`
- ▶ `__getattr__`
- ▶ `__setattr__`
- ▶ `__delattr__`
- ▶ `__call__`

Эмуляция последовательностей

- ▶ `__len__`
- ▶ `__getitem__`
- ▶ `__setitem__`
- ▶ `__delitem__`
- ▶ `__getslice__`
- ▶ `__setslice__`
- ▶ `__delslice__`
- ▶ `__contains__`

```
import logging

class LoggingDict(dict):
    def __setitem__(self, key, value):
        logging.info(
            "Setting {k} to {v}".\
            format(k=key, v=value))
        return super(LoggingDict, self).\
            __setitem__(key, value)

logging.basicConfig(level=logging.INFO)

ld = LoggingDict()
ld["a"] = 123
```


Приведение к базовым типам

- ▶ `__repr__`
- ▶ `__str__`
- ▶ `__oct__`
- ▶ `__hex__`
- ▶ `__complex__`
- ▶ `__int__`
- ▶ `__long__`
- ▶ `__float__`

Пример перегрузки операторов

```
class SignableMatrix:
    def __init__(self, array=[]):
        self.array = dc(array)
        dim_size = len(self.array)
        self.__row_signs = [False for i in range(dim_size)]
        self.__col_signs = [False for i in range(dim_size)]

    def __repr__(self):
        repr_str = "Matrix: [\n"
        for row in self.array:
            repr_str += "    {}\n".format(row)
        repr_str += "]\n"
        repr_str += "Row signs:\n    {}".format(self.__row_signs)
        repr_str += "Column signs:\n    {}".format(self.__col_signs)

        return repr_str

init_graph = [[1, 2, 3], [1, 2, 3], [1, 2, 3]]
init_matrix = SignableMatrix(init_graph)

print init_matrix
```


Остальные магические методы

- ▶ <http://www.rafekettler.com/magicmethods.html>
- ▶ <https://docs.python.org/2/reference/datamodel.html#special-method-names>

Декораторы классов

- ▶ `@staticmethod`
- ▶ `@classmethod`
- ▶ `@property`
- ▶ `@setter`
- ▶ `@deleter`

Обработка исключительных ситуаций — механизм языков программирования, предназначенный для описания реакции программы на возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей отработки программой её базового алгоритма. ²

²http://en.wikipedia.org/wiki/Exception_handling 

Как ловить исключение

```
for arg in sys.argv[1:]:  
    try:  
        f = open(arg, 'r')  
    except IOError:  
        print 'cannot open', arg  
    else:  
        print arg, 'has', len(f.readlines()), 'lines'  
        f.close()
```

Как сгенерировать исключение

```
try:
    raise Exception('spam', 'eggs')
except Exception as inst:
    print type(inst)
    print inst.args
    print inst
    x, y = inst.args
    print 'x =', x
    print 'y =', y
```

Безопасное деление

```
def divide(x, y):  
    try:  
        result = x / y  
    except ZeroDivisionError:  
        print "division by zero!"  
    else:  
        print "result is", result  
    finally:  
        print "executing finally clause"
```


Стандартные исключения

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StandardError
    +-- Warning

+-- Warning
    +-- DeprecationWarning
    +-- PendingDeprecationWarning
    +-- RuntimeWarning
    +-- SyntaxWarning
    +-- UserWarning
    +-- FutureWarning
    +-- ImportWarning
    +-- UnicodeWarning
    +-- BytesWarning

+-- StandardError
    +-- BufferError
    +-- ArithmeticError
    |   +-- FloatingPointError
    |   +-- OverflowError
    |   +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- EnvironmentError
    |   +-- IOError
    |   +-- OSError ...
    +-- EOFError
    +-- ImportError
    +-- LookupError
    |   +-- IndexError
    |   +-- KeyError
    +-- MemoryError
    +-- NameError
    |   +-- UnboundLocalError
    +-- ReferenceError
    +-- RuntimeError
    |   +-- NotImplementedError
    +-- SyntaxError
    |   +-- IndentationError
    |   +-- TabError
    +-- SystemError
    +-- TypeError
    +-- ValueError
        +-- UnicodeError ...
```

- ▶ <https://docs.python.org/2/tutorial/classes.html>
- ▶ <https://docs.python.org/2/library/exceptions.html>
- ▶ <http://www.rafekettler.com/magicmethods.html>
- ▶ <https://docs.python.org/2/reference/datamodel.html#special-method-names>
- ▶ http://www.ibm.com/developerworks/ru/library/l-python_part_6/
- ▶ https://www.ibm.com/developerworks/ru/library/l-python_part_7/

The End

- ▶ Вопросы и предложения;
- ▶ Планы на май.