Python.

Объектно-ориентированное программирование

Повтор про области видимости

В каждый момент существует 3 области видимости:

- Локальная
- Средняя (глобальные имена модуля)
- Внешняя (встроенные имена)

Глобальным переменным невозможно прямо присвоить значения внутри функций, хотя ссылки на них могут использоваться.

Принципы ООП

Какие принципы?

Принципы ООП

- *Абстракция*
- *Инкапсуляция*
- *Полиморфизм*
- *Наследование*

Определение класса

```
class ИмяКласса (класс1, класс2, ...):
    <- выражение 1 ->
    # определения атрибутов и методов класса

class A:
    pass
```

Объекты-классы

- Ссылки на атрибуты
- Создание экземпляра

```
class MyClass:
    """  Class example """
    attr = 12345
    def foo(self):
        return ‘Hello world’

>>> MyClass.attr
>>> MyClass.foo
```

Bound and unbound

```
class A:
    def foo(self):
        pass


>>> A.foo
<unbound method A.foo>
>>> a = A()
>>> a.foo
<bound method A.foo of <__main__.A
instance at 0x10e33aa70>>
```

Методы и атрибуты

```
class Student:
    city = “St. Petersburg”
    def __init__(self, name, year):
        self.name = name
        self.year = year
    def print_info(self) :
        print self.name, "is on the", self.year,
"-th year”
```

Методы и атрибуты

```
>>> vasya = Student("Vasya", 5)
>>> vasya.print_info()
Vasya is on the 5-th year
>>> Student.city
 “St. Petersburg”
>>> vasya.city
 “St. Petersburg”
>>> Student.print_info(vasya)
Vasya is on the 5-th year
```

Методы и атрибуты

```
>>> vasya.city = “Moskow”
>>> print vasya.city
“Moskow”
>>> print Student.city
“St. Petersburg”
>>>Student.city = “Volgograd”
>>> print Student.city
“Volgograd”
```

Инициализатор и деструктор

- __init__(self)
- __del__(self)
- необработанные в деструкторе исключения игнорируются.

Self

- self – текущий экземпляр класса

- Экземпляры классов нет необходимости удалять явно, так как удаление происходит автоматически, когда на них больше нет ссылок.

Методы и функции

```
 def print_info(self) :
        print self.name, "is on the", self.year,
"-th year”


class Student:
    info = print_info
    def __init__(self, name, year):
        self.name=name
        self.year=year
```

Private and public

```
class Student:
    city = “St. Petersburg”
    def __init__(self, name, year):
        self.name=name
        self.year=year
    def __print (self) :
        print self.name, "is on the", self.year, "-th year”

>>> vasya = Student(“Vasya”, 5)
>>> vasya.__print
AttributeError: Student instance has no attribute
'__print'
```

Private

- не менее двух символов подчеркивания в начале
- не более одного символа подчеркивания в конце

Semi private

- **_attribute** – атрибут не предназначен для использования вне методов класса, однако, атрибут все-таки доступен по этому имени

Статические переменные

```
class Counter:
    count = 0
    def __init__(self) :
        self.__class__.count += 1

>>> print Counter.count
0
>>> c = Counter()
>>> print c.count, Counter.count
1 1
>>> d = Counter()
>>> print c.count, d.count, Counter.count
2 2 2
```

Наследование

```
class Person :
    def __init__(self, name) :
        self.name=name
    def print_info(self) :
        print self.name
```

```
class Student (Person) :
    def __init__(self, gr, n) :
        Person.__init__(self, n)
        self.group=gr
    def print_info(self) :
        print self.name, self.group
```

```
>>> p = Person("Petya")
>>> p. print_info()
Petya
>>> s=Student(23, "Vasya")
>>> s. print_info()
Vasya 23
```

Наследование

- Все методы – виртуальные

- Явное указание имени класса для доступа к методу родителя

  Person.__init__(self, n)

Функция super

```
class Child(Parent):
  def __init__(self):
    super(Child, self).__init__(self)
```

Множественное наследование

```
class  First:
   def print_name(self) :
      print ”First"

class Second :
   def print_name(self) :
      print "class2”
```

```
class Child(First,  Second) :
   def print_parent_name(self) :
      self.print_name()

>>> child=Child()
>>> child.print_parent_name()
First
```

Специальные атрибуты классов

- __name__ Имя класса.
- __module__ Имя модуля, в котором класс определен.
- __doc__ Строка документации класса или None, если она не определенна.
- __bases__ Кортеж базовых классов в порядке их следования в списке базовых классов.
- __dict__ Словарь атрибутов класса.

Чисто виртуальные методы

```
class PureVirtual(object):
    def pure(self):
        raise NotImplementedError('Method
PureVirtual.pure is pure virtual')

>>> PureVirtual().pure()
Traceback (most recent call last):
...
NotImplementedError: Method
PureVirtual.pure is pure virtual
```

Имитация встроенных типов

```
class Add:
    def __call__(self, x, y):
        return x + y


>>> add = Add()
>>> add(3, 4)    #add.__call__(3, 4)
7
```

New and old-style classes

```
class OldStyleClass:
    pass                # старый класс

class NewStyleClass(object):
    pass                # новый класс
```

В версии **Python3.x** поддержка «старых» классов была удалена.

New and old-style classes

Old-style:

x.__class__ == class of x

type(x) == <type 'instance'>

New-style:

x.__class__ == type(x)

property

```
class A(object):
    def __init__(self, x):
        self._x = x
    def getx(self):
        return self._x
    def setx(self, value):
        self._x = value
    def delx(self):
        del self._x
    x = property(getx, setx, delx, “x property doc“)
```

Статические методы

```
class StaticExample(object):
    @staticmethod
    def test(x):
        return x == 0
```

Не передается self

```
>>> example = StaticExample()
>>> example.test(1)
False
>>> StaticExample.test(1)
False
```

isinstance & issubclass

- isinstance(obj, int) == True,

если obj.__class__ является int

- issubclass(bool, int) == True

(класс bool является наследником int)

О разном

- Можно наследоваться от встроенных типов
- Атрибуты-данные переопределяют атрибуты-методы с тем же именем

Полиморфизм

```
class Based:
    def __init__(self, n) :
        self.numb = n
    def out (self ):
        print self.numb

class One(Based) :
    def multi(self,m) :
        self.numb *= m
```

```
class Two(Based) :
    def inlist (self) :
        self.inlist = list(str(self.numb))
    def out(self) :
        i = 0
        while i < len (self.inlist) :
            print (self.inlist[i])
            i += 1
```

```
>>> obj1 = One(45)
>>> obj2 = Two('abc')
>>> obj1.multi(2)
>>> obj1.out()
>>> obj2.inlist()
>>> obj2.out()
```

```
90
a
b
c
```

See in next episode.

- Исключения, Итераторы, Генераторы
- Мета-программирование
- Модули
- Сериализация