Объектно-ориентированное программирование на python

Принципы ООП

- Все данные представляются объектами
- Программа является набором взаимодействующих объектов, посылающих друг другу сообщения
- Каждый объект имеет собственную часть памяти и может иметь в составе другие объекты
- Каждый объект имеет тип
- Объекты одного типа могут принимать одни и те же сообщения (и выполнять одни и те же действия)
- *Абстракция, инкапсуляция, полиморфизм, наследование*

Определение класса

class имя_класса (надкласс1, надкласс2, ...):

 # определения атрибутов и методов класса

У класса могут быть базовые (родительские) классы (надклассы), которые (если они есть) указываются в скобках после имени определяемого класса.

Минимально возможное определение класса выглядит так:

```
class A:
    pass
```

Методы классов

Определение метода m1 класса A:

```
class A:
    def m1(self, x):
        # блок кода метода
```

Объявление поля (атрибута/свойства) класса A:

```
class A:
        attr1 = 2 * 2
```

В Питоне класс не является чем-то статическим после определения, поэтому добавить атрибуты можно и после объявления класса:

```
class A:
    pass
def myMethod(self, x):
    return x * x
A.m1 = myMethod
A.attr1 = 2 * 2
```

Создание экземпляра класса (инстанцирование)

```
class Point:
    def __init__(self, x, y, z):
        self.coord = (x, y, z)
    def __repr__(self):
        return "Point(%s, %s, %s)" % self.coord

p = Point(0.0, 1.0, 0.0)
p
>> Point(0.0, 1.0, 0.0)
```

Конструктор и деструктор

```
class Line:
    def __init__(self, p1, p2):
        self.line = (p1, p2)
    def __del__(self):
        print "Удаляется линия %s - %s" % self.line

l = Line((0.0, 1.0), (0.0, 2.0))
del l
>> Удаляется линия (0.0, 1.0) - (0.0, 2.0)
```

- в python реализовано автоматическое управление памятью (garbage collector);
- необработанные в деструкторе исключения игнорируются.

ООП: Инкапсуляция

Методы и данные объекта доступны через его атрибуты:

- **_attribute** # метод не предназначен для использования вне методов класса (или вне функций и классов модуля), однако, атрибут все-таки доступен по этому имени
- **__attribute** # атрибут перестает быть доступен по этому имени
- **__init__** # атрибут доступен по своему имени, но его использование зарезервировано для специальных атрибутов, изменяющих поведение объекта

Отличие __attribute от private

_ИмяКласса__ИмяАтрибута → [экземпляр какого класса]__атрибут

```
class parent(object):
      def __init__(self):
          self.__f = 2
      def get(self):return self.__f
....
class child(parent):
      def __init__(self):
        self.__f = 1
        parent.__init__(self)
      def cget(self):return self.__f
....
c = child()
c.get()
>> 2
c.cget()
>> 1
c.__dict__
>> {'_child__f': 1, '_parent__f': 2}  # на самом деле у объекта "c" два
                                        разных атрибута
```

Управление доступом к атрибутам

- прямой доступ

```
print a.x
```

- с использованием специальных методов (getter, setter)

```
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, "Свойство x“)
print a.x
a.x = 5
```

ООП: Полиморфизм

доступ на этапе исполнения => в python все методы виртуальные

```
class Parent(object):
    def isParOrPChild(self) :
        return True
    def who(self) :
        return 'parent'
class Child(Parent):
    def who(self):
        return 'child'
x = Parent()
x.who(), x.isParOrPChild()
>> ('parent', True)
x = Child()
x.who(), x.isParOrPChild()
>> ('child', True)
```

Полиморфизм - 2

- Signature-oriented polymorfism
- Явно указав имя класса, можно обратиться к методу родителя (как впрочем и любого другого объекта).

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

- В общем случае для получения класса-предка применяется функция super.

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

Имитация чисто виртуальных методов NotImplementedException

```
class abstobj(object):
    def abstmeth(self):
        raise NotImplementedError('Method
    abstobj.abstmeth is pure virtual')
abstobj().abstmeth()
>> Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "<stdin>", line 2, in method
NotImplementedError: Method abstobj.abstmeth is pure
    virtual
```

Управление иерархией наследования

Изменение атрибута **__class__**:

```
c = child()
c.val = 10
c.who()
>> 'child‘
c.__class__ = parent
c.who()
>> 'parent‘
c.val
>> 10
```

Имитация встроенных типов

```
>>> class Point:
...     def __init__(self, x = 0, y = 0):
...         self.x = x
...         self.y = y
...     def __add__(self, point):
...         return Point( self.x+point.x, self.y+point.y )
...     def __repr__(self):
...         return str(self.x) + " " + str(self.y);
>>> a = Point(1,2)
>>> b = Point(3,4)
>>> print(a+b)
4 6
```

Отношения между классами

Наследование и множественное наследование

```
class Par1(object):
    def name1(self): return 'Par1'
class Par2(object):
    def name2(self): return 'Par2'
class Child(Par1, Par2): # создадим класс,
        # наследующий Par1, Par2 (и,
        # опосредованно, object)
    pass
x = Child()
x.name1(), x.name2()
>> 'Par1','Par2'
```

«Новые» и «классические» классы

В версиях до 2.2:

- невозможно наследовать встроенные классы и классы из модулей расширения;
- свойства (property) не выделялись явно;
- etc

Начиная с 2.2:
две объектные модели: «классические» типы и «новые»

«Новые» и «классические» классы - 2

Создание «нового» и «старого» класса:

```
class OldStyleClass:pass # класс "старого" типа
class NewStyleClass(object):pass # и "нового"
```

Все стандартные классы — классы «нового» типа.

В версии **Python3000** поддержка «старых» классов была удалена.

Методы

Синтаксис

- аналогичен функциям,
- первый параметр self.

```
class MyClass(object):
    def mymethod(self, x):
        return x == self._x
```

Мультиметоды

Мультиметод - механизм, позволяющий выбрать одну из нескольких функций в зависимости от динамических типов или значений аргументов.

Пример:

```
import operator as op
print op.add(2, 2), op.add(2.0, 2), op.add(2, 2.0), op.add(2j, 2)
>> 4 4.0 4.0 (2+2j)
```

Мультиметоды - 2

Пример определенных пользователем мультиметодов

```
@multimethod(Asteroid, Asteroid)
def collide(a, b):
    """Behavior when asteroid hits asteroid"""
    # определено новое поведение
@multimethod(Asteroid, Spaceship)
def collide(a, b):
    """Behavior when asteroid hits spaceship"""
    # определено новое поведение

# определяем другие мультиметоды..
```

Домашнее задание

Матрицы:

- Создать класс матриц 3x3
- Создать класс трехмерных векторов.
- Создать наследника класса матриц – класс матриц поворота вокруг оси Z.
- Перегрузить операции:
  - Сложения матриц
  - Умножения матриц на число
  - Умножения матрицы на вектор
  - Умножения матрицы на другую матрицу
  - Взятия обратной матрицы (операцию ~)

Домашнее задание

Машинки:

Написать базовый класс Car, в котором определен метод who, выводящий на экран фразу «Привет я машинко»

Создать два производных класса DriftCar и Bulldozer, выводящих при вызове метода who фразы «Я супер машинка» и «А я бульдозер :(», соответственно.

Создать объекты всех вышеперечисленных типов и вызвать у них метод who().

Домашнее задание

База данных:

Задан файл успеваемости студентов следующего вида:

4

Вася Пупкин: 3 3 3 5 2 1

Петя Васечкин: 2 3 1

Александро Дель-Пьеро: 4 1 5

Робин ван Перси: 2 2 2

Требуется создать класс студента, хранящий набор оценок.
У данного класса должна быть возможность:

- выводить себя с помощью команды print в формате указанном выше.
- Считать средний балл
- Добавлять новые оценки
- Конструктор на вход должен принимать имя Студента

Домашнее задание

Также требуется написать класс базы данных. Конструктор этого класса на вход принимает имя файла, после чего загружает всю информацию из файла, создавая набор объектов типа Студент.

Должна быть предусмотрена возможность добавления, удаления студентов, а также возможность вывода всей базы данных в файл.

Должен быть написан метод, позволяющий находить студента с наивысшим среедним баллом.