Классы

- Класс это объединение данных и функций, работающих над ними
- Данные должны быть взаимосвязанны, а функции формировать API, достаточное для полноценной работы над данными ~~без прямого доступа к ним~~
- Язык обеспечивает автоматизацию многих задач по поддержке ООП

Пример класса

```
1       class Simple(ParentClass):
2           "class␣documentation"
3
4           def method(self, val):
5               "method␣documentation"
6               self.some_field = val
7               return val ** 0.3
```

Нет инкапсуляции

- "Принцип открытого кимоно" == "мы не знаем как совместить инкапсуляцию с остальными возможностями языка"
- Есть property, "скрытые поля" (но они предназначенны для другого)
- В отличии от Java всегда можно изменить поле на свойство с сохранением совместимости
- Можно реализовать любой вид инкапсуляции динамической
- Документирование API vs чтение заголовков
- Ничего из этого не получило какого-нить распространения в питоне
- Реализуя сокрытие полей объекта вы создаете проблемы многим библиотекам python

__init__

- __init__ конструктор, вызывается при создании экземпляра класса
-

```
1    class A(object):
2        def __init__(self, val):
3            self.var = val
4            print "A inited with value", val
5
6    a = A(1) # A inited with value 1
7    print a.var # 1
```

__del__

- __del__ Должен вызываться перед удалением объекта (и, обычно, вызывается достаточно предсказуемо)
- Если нет циклических ссылок
- Или объект не попал во фрейм, где произошла ошибка
- Использовать with вместо надежд на __del__
- def __del__(self):...

__new__

- __new__ аналог перегрузки new в C++
- Классовый метод(автоматически), вызываемый для создания нового экземпляра объекта, который затем будет проинициализирован с помощью __init__. classmethod использовать не надо.
- Получает те-же параметры, что и __init__.

```
1     class X(object):
2         def __new__(cls, val):
3             print "{}.__new__({!r})".format(cls.__name__, val)
4             return object.__new__(cls, val)
5
6         def __init__(self, val):
7             print "{}.__init__({!r})".format(self.__class__.__nar
8
9     X(1)
10    #X.__new__(1)
11    #X.__init__(1)
```

__new__

- __new__ может вернуть объект другого типа
- Наверное, не самая лучшая идея

```
1   class A(object):
2       def __init__(self, x):
3           pass
4
5   class B(object):
6       def __init__(self, x, y):
7           pass
8       def __new__(cls, x, y=None):
9           if y is None:
10              return A(x)
11          else:
12              return super(B, cls).__new__(cls, x, y)
13
14  print B(1, 2) #<__main__.B at 0x...>
15  print B(1) # <__main__.A at 0x...>
```

Наследование TBD

Полиморфизм

- Все методы - виртуальные.

```
1     class A(object):
2         def some_method(self):
3             print "A.some_meth"
4
5     class B(object):
6         def some_method(self):
7             print "B.some_meth"
8
9     b = B(1)
10    # B inited with value 1
11    # A inited with value 1
12    b.double_var() # B double called
```

super

```
1   class A(object):
2       def __init__(self, val):
3           self.var = val
4           print "A inited with value", val
5       def double_var(self):
6           self.var *= 2
7
8   class B(A):
9       def __init__(self, val):
10          print "B inited with value", val
11          A.__init__(self, val)
12      def double_var(self):
13          self.var *= 2
14          print "B double called"
15
16  b = B(1)
17  # B inited with value 1
18  # A inited with value 1
19  b.double_var()
20  # B double called
```

super

```
1       class A(object):
2           def draw(self, pt):
3               some_action()
4
5       class B(A):
6           def draw(self, pt):
7               #A.draw(self, pt)
8               super(B, self).draw(pt)
```

super uncovered TBD

Скрытие полей

- __xxxx - "скрытые поля и методы" - переименовываются для избежания пересечения имен
- A.__xxx -> A._A__xxx
- Не для инкапсуляции

Связанные и не связанные методы

- a = A(), a.b(1) == A.b(a, 1)
- A.b несвязанный метод. A.b(1) - ошибка, Первым параметром должен идти экземпляр класса A. (A.b)(a, 1) - ok
- a.b - связанный метод, эквивалентен функции.

```
1       a = A()
2       t = a.b
3       t(1) # a.b(1)
```

Сортировка и сравнение DSU

Классовые и статические методы

- classmethod превращяет метод в классовый, первым параметром вместо экземпляра такой метод получает класс и может быть вызван от класса
- staticmethod превращяет метод в статический - обычная функция

```
1     class A(object):
2         val = 12
3         @classmethod
4         def meth1(cls, x):
5             return x + cls.val
6         @staticmethod
7         def meth2(x, y):
8             return x + y
9
10    A.meth1(1) == 13
11    A().meth2(1,2) == 3
12
13    class B(A):
```

```
14          val = 13
15
16      B.meth1(1) == 14
```

Специальные методы - int

- __add__(self, obj) # self + obj
- __radd__(self, obj) # obj + self
- __iadd__(self, obj) # self += obj
- __int__(self) # int(self)

Работа некоторых встроенных функций (протоколы встроенных функций)

- int(x) == x.__int__()
- str(x) == x.__str__()
- repr(x) == x.__repr__()
- len(x) == x.__len__()
- iter(x) == x.__iter__()
- next(x) == x.next() O_o
- hex, oct, hash

Специальные методы - контейнер

- x.__getitem__(index) #x[index]
- x.__setitem__(index, val) # x[index] = val
- x.__delitem__(index) #del x[index]
- ...

Специальные методы - доступ к атрибутам

- x.__getattribute__(name)
- x.__getattr__(name)
- x.__setattr__(name, val)
- x.__delattr__(name)
- getattr(x, name[, val])
- setattr(x, name, val)
- delattr(x, name)