Python.

Всякое.

Обход контейнеров.

```
for element in [1, 2, 3]:
    print element
for element in (1, 2, 3):
    print element
for key in {'один':1, 'два':2 }:
    print key
for char in "123":
    print char
for line in open("myfile.txt"):
    print line
```

- оператор for вызывает метод iter() объекта-контейнера
- метод next()
- исключение StopIteration

Пример

```
>>> s = 'a6'
>>> it = iter(s)
>>> it
<iterator object at 0x00A1DB50>
>>> it.next()
'a’
>>> it.next()
‘6’
>>> it.next()
Traceback (most recent call last):
 File "<stdin>", line 1, in ?
        it.next()
StopIteration
```

Создание итератора.

```
class Reverse:
    "Итератор для прохождения циклом по последовательности в обратном
направлении"
    def __init__(self, data):
        self.data = data
        self.index = len(data)
    def __iter__(self):
        return self
    def next(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]

>>> for char in Reverse('спам'):
...     print char
...
м
а
п
с
```

Генераторы.

- Инструмент для создания итераторов.
- Каждый раз, при вызове next(), генератор возвращается к месту, где он был остановлен

- Автоматически создаются __iter__() и next()

Пример.

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]

>>> for char in reverse('гольф'):
...     print char
...
ф
ь
л
о
г
```

Выражения-генераторы

```
>>> sum(i*i for i in range(10))                 # сумма квадратов
285

>>> xvec = [10, 20, 30]
>>> yvec = [7, 5, 3]
>>> sum(x*y for x,y in zip(xvec, yvec))     # скалярное
произведение
260

>>> unique_words = set(word  for line in page  for word in
line.split())
```

Декораторы. Пример.

```
def makebold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

@makebold
def hello():
    return "hello world"

print hello() ## выведет <b>hello world</b>
```

Функция = объект

```
def shout(word="да"):
    return word.capitalize()+"!"

print shout()    # выведет: 'Да!’

scream = shout   # связывание

print scream()   # выведет: 'Да!’

del shout
try:
    print shout()
except NameError, e:
    print e           #выведет: "name 'shout' is not defined"

print scream()   # выведет: 'Да!'
```

Функция в функции

```
def talk():
    def whisper(word="да"):
        return word.lower()+"..."

    print whisper()

talk()

try:
    print whisper()
except NameError, e:
    print e
# выведет : "name 'whisper' is not defined"
```

Ссылки на функции

```
def getTalk(type="shout"):

    def shout(word="да"):
        return word.capitalize()+"!"

    def whisper(word="да”):
        return word.lower()+"..."

    if type == "shout":
        return shout
    else:
        return whisper

talk = getTalk()
print talk()
print getTalk("whisper")()
```

Функция как параметр

```
def doSomethingBefore(func):
    print " Do something before calling function."
    print func()

doSomethingBefore(scream)
```

Декораторы.

- Обертка, которая позволяет делать что-либо до или после вызова функции

- Функция, ожидающая другую функцию как параметр.

Примеры

- @decorator — просто синтаксический сахар
- порядок декорирования важен

Декораторы с параметрами

Пример

Декорирование функций vs. Декорирование методов

- все функции декорируются сразу же при импорте.

- Вы не можете «раздекорировать» функцию.

Использование.

- для расширения возможностей функций из сторонних библиотек
- расширения различных функций одним и тем же кодом, без повторного его переписывания каждый раз

- Property, staticmethod