Advanced Python. Functions, generators.

Функции

- Терминология: процедуры, функции, методы.
- Анонимные функции (lambda expressions)

Именованные функции

Именованные функции:

```
def function_name(arguments):
   statements*
```

Используйте единую схему именования (лучше всего ту, которая принята в Python)

Анонимные функции

Объявление:

```
lambda arguments: expression
```

Эквивалент именованной функции:

```
def <noname>(arguments):
    return expression
```

Возврат значений

Возврат значения:

```
def function_name(arguments):
    ...
    return expression
```

Возврат нескольких значений:

```
def function_name(arguments):
    ...
    return a, b
```

Возврат значений

Функции могут возвращать более одного значения, в этом случае они упаковываются в кортеж (tuple):

```
def fn(arg):
    return True, “Success”

status, message = fn()
```

Возврат значений

Функции, не возвращающие значения:

```
def procedure1():
  return

def procedure2():
  2 * 2
```

Функции, не возвращающие значения, неявно возвращают None

Возврат значений

Функции могут содержать более одного return

```
def fn(arg):
  if arg > 0:
    return 'positive'
  elif arg < 0:
    return 'negative'
  else:
    return
```

Функция без тела

Ключевое слово **pass** используется как пустое, no-op выражение для сохранения синтаксической структуры кода.

```
def void_function():
  pass
```

Вызов функции

```
def add(a, b):
  return a + b
```

Вызов:

```
add(2, 3)
```

Присваивание возвращенного значения переменной:

```
result = add(2, 3)
```

Аргументы функции

Функция без аргументов:

```
def some_func():
  statements
```

Позиционные аргументы:

```
def some_func(a, b, c):
  statements
```

Аргументы функции

Значения аргументов по умолчанию:

```
def some_func(a, b=2, c=3):
   statements
```

Аргументы функции

Список аргументов:

```
def some_func(*args):
  for a in args:
    do_something(a)
```

Аргументы функции

Keyword аргументы (словарь аргументов):

```
def some_func(**kwargs):
  for key, val in kwargs:
    print(key, val)
```

Аргументы функции

Все виды аргументов можно комбинировать, сохраняя правильный порядок:

```
def f(a, b, c=2, *args, **kwargs):
    ...
```

Передача аргументов

- При передаче аргументов копии объектов не создаются
- Передавая в неизвестную функцию изменяемый объект - делайте копию, если вам нужно его исходное содержимое!
- Если изменяете в своей функции mutable аргументы - документируйте (еще лучше - возвращайте копию)!

Docstrings

- Первым выражением после объявления функции может быть строка, предназначенная для документирования функции
- В docstrings всегда рекомендуется использовать multiline строки (вместо ‘ или “ используются тройные кавычки ’’’ или ”””).
- PEP 257 - рекомендации по написанию docstrings
- Docstring хранится в специальном атрибуте __doc__

Области видимости

- Каждая функция создает свою локальную область видимости (scope)
- globals(), locals()
- Инструкция global дает возможность доступа к глобальной области видимости
- Правило LEGB:

L - local, локальная область видимости
E - enclosing, окружающая область видимости
G - global, глобальный контекст
B - built-in, имена встроенных объектов

Вложенные функции

Функции могут определены внутри других функций.

```
def adder(a):
    def inner(b):
        return a + b
    return inner
```

Декораторы

Декоратор - функция, принимающая декорируемую функцию в качестве аргумента, и возвращающая функцию.

```
def source_func(*args):
    ...
result_func = decorator(source_func)
```

С декоратором:

```
@decorator
def source_func(*args):
    ...
```

Декораторы

Декоратор может быть параметризованным, в этом случае это функция, принимающая параметры в качестве аргументов, и возвращающая функцию-декоратор (которая принимает декорируемую функцию в качестве аргумента, и возвращает функцию).

```
@bounded(0, 100)
def func(*args):
    ...
```

Функции - генераторы

В Python существует специальный синтаксис для создания итераторов: generator functions, функции-генераторы.

Для реализации генераторов вместо **return** для возврата значений используется ключевое слово **yield**

```
def desc(n):
  while n > 0:
    yield n
    n -= 1
```

Функции - генераторы

- При вызове генератора возвращается объект - генератор, из которого можно выбрать значения, вызывая метод __next__()
- … или встроенную функцию next()
- … или перебрав все значения в цикле **for**
- Когда значения закончатся, будет выброшено исключение StopIteration

Функции - генераторы

return можно использовать для принудительного завершения генератора, при этом будет выброшено исключение **StopIteration**

```
def desc(n):
  while True:
    yield n
    if n <= 0:
      return   # StopIteration
    n -= 1
```

Функции - генераторы

В генератор также можно отправить значение с помощью функции **send()**. Для получения значения нужно использовать расширенную форму выражения **yield**

```
def acc():
  a = 0
  while True:
    n = yield a
    if n is None:
      n = 1
    a += n
```

Рекурсия

- Рекурсия - вызов функции из нее же самой
- в Python лучше не использовать
- sys.{get,set}recursionlimit()

```
def factorial(n):
    if n <= 0:
        return 1
    else:
        return n * factorial(n-1)
```