# Основные стандартные модули Python

Вступление

Понятие модуля

Как написать  модуль

Модули в Python

Обзор стандартной библиотеки

Сервисы периода выполнения

Взаимодействие с операционной системой

Модуль XML

# Вступление

*Модуль* – файл, содержащий определения и другие инструкции языка Python. Имя файла образуется путем добавления к имени модуля суффикса(расширения) '.py'.

В пределах модуля его имя доступно глобальной переменной

__name__.

# Как написать свой модуль

Модуль оформляется в виде отдельного файла с исходным кодом.

```
#!/usr/bin/python                    # создали файл fibo.py
def fib(n):
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b


>>> import fibo
>>> fibo.fib(500) # 1 1 2 3 5 8 13 21 34 55 89 144
  233 377
```

# Программа на PYTHON

В программе на Python модуль представлен объектом-модулем, атрибутами которого являются имена, определенные в модуле:

```
#!/usr/bin/python
import datetime
d1 = datetime.date(2010, 03, 31)
```

В этом примере импортируется модуль datetime. В результате работы оператора import в текущем пространстве имен появляется объект с именем datetime.

# Модули расширения

Модули для использования в программах на языке Python по своему происхождению делятся на обычные (написанные на Python) и модули расширения, написанные на другом языке программирования (как правило, на C).

С точки зрения пользователя они могут отличаться разве что быстродействием. Бывает, что в стандартной библиотеке есть два варианта модуля: **на Python и на C.** Таковы, например, модули **pickle и cPickle**.

Обычно модули на Python в чем-то гибче, чем модули расширения.

# Пути к каталогам

Стандартные модули находятся в каталоге, где их может найти соответствующий интерпретатор языка. Пути к каталогам, в которых Python ищет модули, можно увидеть в значении переменной sys.path.

```
>>> sys.path
----------------------------------------
['', '/usr/lib/python25.zip', '/usr/lib/python2.5']
```

В последних версиях Python модули можно помещать и в zip архивы для более компактного хранения (по аналогии с jar архивами в Java).При запуске программы поиск модулей также идет в текущем каталоге. (Нужно внимательно называть собственные модули, чтобы не было конфликта имен состандартными или дополнительно установленными модулями.)

# Подключение к модулю

Подключение модуля к программе на Python осуществляется с помощью оператора import.
У него есть две формы: import и from-import:

```
>>> import fibo
>>> from fibo import fib, fib2
>>> from fibo import *
```

С помощью первой формы с текущей областью видимости связывается только имя, ссылающееся на объект модуля, а при использовании второй - указанные имена (или все имена, если применена *) объектов модуля связываются с текущей областью видимости.

# Пример

Если Вы собираетесь использовать функцию часто, то можно ее присвоить локальной переменной:

```
>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

# Пример

Этот пример показывает как возможно избежать конфликта имен, опередеенного в модуле, со встроенным именем:

```
import anydbm
dbopen = anydbm.open
```

# import AS

Начиная с версии 2.0 благодаря расширению синтаксиса инструкции import возможно импортировать модуль используя для него локальное имя, отличное от исходного:

```
import string as _string

from anydbm import open as dbopen
```

Следует заметить, что as не является зарезервированным словом, и Вы можете по-прежнему определять переменные с таким именем

# Функция dir()

Несмотря на то, что относительно легко найти и импортировать модуль, не так-то просто запомнить, что каждый модуль содержит. И вряд ли вам захочется всякий раз смотреть исходный код, чтобы это выяснить. К счастью, Python предоставляет способ определения содержимого модулей (и других объектов) с помощью встроенной функции dir().

Функция dir(), вероятно, наиболее известная из всех интроспективных механизмов Python. Т.е. это означает, что для любого объекта можно получить всю информацию о его внутренней структуре. Она возвращает отсортированный список имен атрибутов для любого переданного в нее объекта. Если ни один объект не указан, dir() возвращает имена в текущей области видимости.

# Пример

```
>>> a = [1, 2, 3, 4, 5]
>>> import fibo, sys
>>> fib = fibo.fib
>>> dir() #Имена в текущей области видимости
-----------------------------------------------
['__name__', 'a', 'fib', 'fibo', 'sys']
```

Обратите внимание, что перечисляются имена всех типов:

переменные, функции, модули и т.п.

Список возвращаемый функцией dir() не содержит имена встроенных функций и переменных, они опередены в стандартном модуле __builtin__

# Пример

```
>>> import anydbm
>>> dir(anydbm)        # атрибуты модуля anydbm
-----------------------------------------

['__builtins__', '__doc__', '__file__',
  '__name__', '_defaultmod', '_errors', '_mod',
  '_name', '_names', 'error', 'open']
```

# Модуль sys

Модуль sys содержит информацию о среде выполнения программы, об интерпретаторе Python. Далее будут представлены наиболее популярные объекты из этого модуля:

| | |
|---|---|
| exit([c]) | Выход из программы. Можно передать числовой код завершения. |
| argv | Список аргументов командной строки. |
| sys.argv[0] | содержит имя запущенной программы, а остальные параметры передаются из командной строки. |
| platform | Платформа, на которой работает интерпретатор. |
| stdin, stdout, stderr | Стандартный ввод, вывод, вывод ошибок. |
| version | Версия интерпретатора. |
| setrecursionlimit(limit) | Установка уровня максимальной вложенности рекурсивных вызовов. |
| exc_info() | Информация об обрабатываемом исключении. |

# Пример

```
>>> sys.platform
```

---

'linux2'

```
>>> sys.version
```

---

'2.5.2 (r252:60911, Jan 20 2010, 21:48:48) \n[GCC 4.2.4 (Ubuntu
    4.2.4-1ubuntu3)]'

```
>>> sys.maxint
```

---

2147483647

# Модуль OS

Различные операционные системы имеют свои особенности. Здесь рассматривается основной модуль этой категории, функции которого работают на многих операционных системах. Разделители каталогов и другие связанные с этим обозначения доступны в виде констант.

| Константа | Что обозначает |
|---|---|
| os.curdir | Текущий каталог |
| os.pardir | Родительский каталог |
| os.sep | Разделитель элементов пути |
| os.altsep | Другой разделитель элементов пути |
| os.pathsep | Разделитель путей в списке путей |
| os.defpath | Список путей по умолчанию |
| os.linesep | Признак окончания строки |

.

# Модуль OS

Программа на Python работает в операционной системе в виде отдельного процесса.
Функции модуля os дают доступ к различным значениям, относящимся к процессу и к среде, в которой он исполняется.
Одним из важных объектов, доступных из модуля **os**, является словарь переменных окружения environ.
В следующем примере можно получить переменную окружения PATH:

```
import os
PATH = os.environ['PATH']
```

# Модуль tempfile

Программе иногда требуется создать временный файл, который после выполнения некоторых действий больше не нужен. Для этих целей можно использовать функцию TemporaryFile, которая возвращает файловый объект, готовый к записи и чтению.В следующем примере создается временный файл, куда записываются данные и затем читаются:

```
import tempfile
f = tempfile.TemporaryFile()
f.write("0"*100)          # записывается сто символов 0
f.seek(0)                 # уст. указатель на начало файла

print len(f.read())       # читается до конца файла и вычисляется длина
```

Как и следовало ожидать, в результате будет выведено 100. Временный файл будет удален, как только будут удалены все ссылки на его объект.

# Работа с несколькими файлами

Для упрощения работы с несколькими файлами можно использовать модуль fileinput. Он позволяет обработать в одном цикле строки всех указанных в командной строке файлов:

```
import fileinput

for line in fileinput.input():

    process(line)
```

В случае, когда файлов не задано, обрабатывается стандартный ввод.

# Модуль csv

Формат CSV (comma separated values - значения, разделенные запятыми) достаточно популярен для обмена данными между электронными таблицами и базами данных.

```
mydata = [(1, 2, 3), (1, 3, 4)]

import csv

f = file("my.csv", "w") # Запись в файл

writer = csv.writer(f)

for row in mydata:

    writer.writerow(row)

f.close()

reader = csv.reader(file("my.csv")) # Чтение из файла

for row in reader:

    print row
```

---

```
['1', '2', '3']

['1', '3', '4']
```

# Import XML.SAX

**SAX** (Simple API for XML, простой программный интерфейс для XML). Работа SAX заключается в чтении источников данных (input source) XML-анализаторами (XML-reader) и генерации последовательности событий (events), которые обрабатываются объектами-обработчиками (handlers). SAX дает последовательный доступ к XML-документу.

# Листинг XML

**Простейший пример XML файла**

```
<?xml version = "1.0" encoding = "iso-8859-1"?>
<persons>
    <person name = "Nikita" surname = "Timofeev" age = "23" />
    <person name = "Tatiana" surname = "Trofimova" age = "25"/>
</persons>
```

# парсинг XML

```
#!/usr/bin/python
import xml.sax
import xml.sax.handler
class MyHandler(xml.sax.handler.ContentHandler):
    def __init__(self):
        self.persons = []

    def startElement(self, name, attributes):
        if name == "person":
            self.name = attributes["name"]
            self.surname = attributes["surname"]
            self.age = attributes["age"]
```

# продолжение

```
def endElement(self, name):
    if name == "person":
        self.persons.append({'name': self.name, 'surname':
          self.surname, 'age': self.age})


parser = xml.sax.make_parser()
handler = MyHandler()
parser.setContentHandler(handler)
parser.parse("example.xml")
print handler.persons
```

---

```
[{'age': u'23', 'surname': u'Timofeev', 'name':
  u'Nikita'}, {'age': u'25', 'surname': u'Trofimova',
  'name': u'Tatiana'}]
```

# создание XML

```
#!/usr/bin/python

import sys

from xml.sax.saxutils import XMLGenerator

s = [{'age': u'23', 'surname': u'Timofeev', 'name':
  u'Nikita'}, {'age': u'25', 'surname': u'Trofimova',
  'name': u'Tatiana'}]

g = XMLGenerator(sys.stdout)

g.startDocument()

g.startElement(”persons”, {})

for s in persons:

    g.startElement(”person”, s)

    g.endElement(”person”)

g.endElement(”persons”)

g.endDocument()
```

# Вопросы?

Спасибо за внимание!