Python. Лекция 2.
Основные стандартные модули Python.

Одним из важных преимуществ языка Python является наличие большой библиотеки модулей и пакетов, входящих в стандартную поставку. Как говорят, к Python "приложены батарейки".

Понятие модуля

Перед тем как приступить к изучению модулей стандартной библиотеки, необходимо определить то, что в Python называется **модулем**.

В соответствии с модульным подходом к программированию большая задача разбивается на несколько более мелких, каждую из которых (в идеале) решает отдельный модуль. В разных методологиях даются различные ограничения на размер модулей, однако при построении модульной структуры программы важнее составить такую композицию модулей, которая позволила бы свести к минимуму связи между ними. Набор классов и функций, имеющий множество связей между своими элементами, было бы логично расположить в одном модуле. Есть и еще одно полезное замечание: модули должно быть легче использовать, чем написать заново. Это значит, что модуль должен иметь удобный **интерфейс**: набор функций, классов и констант, который он предлагает своим пользователям.

В языке Python набор модулей, посвященных одной проблеме, можно поместить в **пакет**. Хорошим примером такого пакета является пакет `xml`, в котором собраны модули для различных аспектов обработки XML.

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

```
>>> import datetime
>>> d1 = datetime.date(2004, 11, 20)
```

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

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.

Модули в Python

Модуль оформляется в виде отдельного файла с исходным кодом. 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/local/lib/python23.zip',
```

```
'/usr/local/lib/python2.3',
'/usr/local/lib/python2.3/plat-linux2',
'/usr/local/lib/python2.3/lib-tk',
'/usr/local/lib/python2.3/lib-dynload',
'/usr/local/lib/python2.3/site-packages']
```

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 со стандартными или дополнительно установленными модулями.)

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

```
import os
import pre as re
from sys import argv, environ
from string 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. При импорте можно изменить имя, с которым объект будет связан, с помощью `as`. В первом случае пространство имен модуля остается в отдельном имени и для доступа к конкретному имени из модуля нужно применять точку. Во втором случае имена используются так, как если бы они были определены в текущем модуле:

```
os.system("dir")
digits = re.compile("\d+")
print argv[0], environ
```

Повторный импорт модуля происходит гораздо быстрее, так как модули кэшируются интерпретатором. Загруженный модуль можно загрузить еще раз (например, если модуль изменился на диске) с помощью функции `reload()`:

```
import mymodule
. . .
reload(mymodule)
```

Однако в этом случае все объекты, являющиеся экземплярами классов из старого варианта модуля, не изменят своего поведения.

При работе с модулями есть и другие тонкости. Например, сам процесс импорта модуля можно переопределить. Подробнее об этом можно узнать в оригинальной документации.

Встроенные функции

В среде Python без дополнительных операций импорта доступно более сотни встроенных объектов, в основном, функций и исключений. Для удобства функции условно разделены по категориям:

1. Функции преобразования типов и классы: `coerce`, `str`, `repr`, `int`, `list`, `tuple`, `long`, `float`, `complex`, `dict`, `super`, `file`, `bool`, `object`
2. Числовые и строковые функции: `abs`, `divmod`, `ord`, `pow`, `len`, `chr`, `unichr`, `hex`, `oct`, `cmp`, `round`, `unicode`
3. Функции обработки данных: `apply`, `map`, `filter`, `reduce`, `zip`, `range`, `xrange`, `max`, `min`, `iter`, `enumerate`, `sum`
4. Функции определения свойств: `hash`, `id`, `callable`, `issubclass`, `isinstance`, `type`
5. Функции для доступа к внутренним структурам: `locals`, `globals`, `vars`, `intern`, `dir`
6. Функции компиляции и исполнения: `eval`, `execfile`, `reload`, `__import__`, `compile`
7. Функции ввода-вывода: `input`, `raw_input`, `open`
8. Функции для работы с атрибутами: `getattr`, `setattr`, `delattr`, `hasattr`
9. Функции-"украшатели" методов классов: `staticmethod`, `classmethod`, `property`
10. Прочие функции: `buffer`, `slice`

Совет:

Уточнить назначение функции, ее аргументов и результата можно в интерактивной сессии интерпретатора Python:

```
>>> help(len)
Help on built-in function len:
len(...)
  len(object) -> integer
  Return the number of items of a sequence or mapping.
```

Или так:

```
>>> print len.__doc__
len(object) -> integer
Return the number of items of a sequence or mapping.
```

Функции преобразования типов и классы

Функции и классы из этой категории служат для преобразования типов данных. В старых версиях Python для преобразования к нужному типу использовалась одноименная функция. В новых версиях Python роль таких функций играют имена встроенных классов (однако семантика не изменилась). Для понимания сути достаточно небольшого примера:

```
>>> int(23.5)
23
>>> float('12.345')
12.345000000000001
>>> dict([('a', 2), ('b', 3)])
{'a': 2, 'b': 3}
>>> object
<type 'object'>
>>> class MyObject(object):
...   pass
...
```

Числовые и строковые функции

Функции работают с числовыми или строковыми аргументами. В следующей таблице даны описания этих функций.

`abs(x)`	Модуль числа `x`. Результат: `|x|`.
`divmod(x, y)`	Частное и остаток от деления. Результат: (частное, остаток).
`pow(x, y[, m])`	Возведение `x` в степень `y` по модулю `m`. Результат: `x**y % m`.
`round(n[, z])`	Округление чисел до заданного знака после (или до) точки.
`ord(s)`	Функция возвращает код (или Unicode) заданного ей символа в односимвольной строке.
`chr(n)`	Возвращает строку с символом с заданным кодом.
`len(s)`	Возвращает число элементов последовательности или отображения.
`oct(n)`, `hex(n)`	Функции возвращают строку с восьмеричным или шестнадцатеричным представлением целого числа `n`.
`cmp(x, y)`	Сравнение двух значений. Результат: отрицательный, ноль или положительный, в зависимости от результата сравнения.
`unichr(n)`	Возвращает односимвольную Unicode-строку с символом с кодом `n`.
`unicode(s [, encoding[, errors]])`	Создает Unicode-объект, соответствующий строке `s` в заданной кодировке encoding. Ошибки кодирования обрабатываются в соответствии с errors, который может принимать значения: `'strict'` (строгое преобразование), `'replace'` (с заменой несуществующих символов) или `'ignore'` (игнорировать несуществующие символы). По умолчанию: `encoding='utf-8'`, `errors='strict'`.

Следующий пример строит таблицу кодировки кириллических букв в Unicode:

```
print "Таблица Unicode (русские буквы)".center(18*4)
i = 0
for c in "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"\
         "абвгдежзийклмнопрстуфхцчшщъыьэюя":
  u = unicode(c, 'koi8-r')
  print "%3i: %1s %s" % (ord(u), c, `u`),
  i += 1
  if i % 4 == 0:
    print
```

Функции обработки данных

Эти функции подробнее будут рассмотрены в лекции по функциональному программированию. Пример с функциями `range()` и `enumerate()`:

```
>>> for i, c in enumerate("ABC"):
...    print i, c
...
0 A
1 B
2 C
>>> print range(4, 20, 2)
[4, 6, 8, 10, 12, 14, 16, 18]
```

Функции определения свойств

Эти функции обеспечивают доступ к некоторым встроенным атрибутам объектов и другим свойствам. Следующий пример показывает некоторые из этих функций:

```
>>> s = "abcde"
>>> s1 = "abcde"
>>> s2 = "ab" + "cde"
>>> print "hash:", hash(s), hash(s1), hash(s2)
hash: -1332677140 -1332677140 -1332677140
>>> print "id:", id(s), id(s1), id(s2)
id: 1076618592 1076618592 1076618656
```

Здесь, можно увидеть, что для одного и того же строкового литерала `"abcde"` получается один и тот же объект, тогда как для одинаковых по значению объектов вполне можно получить разные объекты.

Функции для доступа к внутренним структурам

В современной реализации языка Python глобальные и локальные переменные доступны в виде словаря благодаря функциям `globals()` и `locals()`. Правда, записывать что-либо в эти словари не рекомендуется.

Функция `vars()` возвращает таблицу локальных имен некоторого объекта (если параметр не задан, она возвращает то же, что и `locals()`). Обычно используется в качестве словаря для операции форматирования:

```
a = 1
b = 2
c = 3
print "%(a)s + %(b)s = %(c)s" % vars()
```

Функции компиляции и исполнения

Функция `reload()` уже рассматривалась, а из остальных функций этой категории особого внимания заслуживает `eval()`. Как следует из названия, эта функция вычисляет переданное ей выражение. В примере ниже вычисляется выражение, которое строится динамически:

```
a = 2
b = 3
for op in "+-*/%":
  e = "a " + op + " b"
  print e, "->", eval(e)
```

У функции `eval()` кроме подлежащего вычислению выражения есть еще два параметра - с их помощью можно задать глобальное и локальное пространства имен, из которых будут разрешаться имена выражения. Пример выше, переписанный для использования с собственным словарем имен в качестве глобального пространства имен:

```
for op in "+-*/%":
  e = "a " + op + " b"
  print e, "->", eval(e, {'a': 2, 'b': 3})
```

Функцией `eval()` легко злоупотребить. Нужно стараться использовать ее только тогда, когда без нее не обойтись. Из соображений безопасности не следует применять `eval()` для аргумента, в котором присутствует непроверенный ввод от пользователя.

Функции ввода-вывода

Функции `input()` и `raw_input()` используются для ввода со стандартного ввода. В серьезных программах их лучше не применять. Функция `open()` служит для открытия файла по имени для чтения, записи или изменения. В следующем примере файл открывается для чтения:

```
f = open("file.txt", "r", 1)
for line in f:
  . . .
f.close()
```

Функция принимает три аргумента: имя файла (путь к файлу), режим открытия (" `r` " - чтение, " `w` " - запись, " `a` " - добавление или " `w+` ", " `a+` ", " `r+` " - изменение. Также может прибавляться " `t` ", что обозначает текстовый файл. Это имеет значение только на платформе Windows). Третий аргумент указывает режим буферизации: `0` - без буферизации, `1` - построчная буферизация, больше `1` - буфер указанного размера в байтах.

В новых версиях Python функция `open()` является синонимом для `file()`.

Функции для работы с атрибутами

У объектов в языке Python могут быть атрибуты (в терминологии языка C++ - члены-данные и члены-функции). Следующие две программы эквивалентны:

```
# первая программа:
class A:
  pass
a = A()
a.attr = 1
try:
  print a.attr
except:
  print None
del a.attr

# вторая программа:
class A:
  pass
a = A()
setattr(a, 'attr', 1)
if hasattr(a, 'attr'):
  print getattr(a, 'attr')
else:
  print None
delattr(a, 'attr')
```

Функции-"украшатели" методов классов

Эти функции будут рассмотрены в лекции, посвященной ООП.

Обзор стандартной библиотеки

Модули стандартной библиотеки можно условно разбить на группы по тематике.

1. Сервисы периода выполнения. Модули: `sys`, `atexit`, `copy`, `traceback`, `math`, `cmath`, `random`, `time`, `calendar`, `datetime`, `sets`, `array`, `struct`, `itertools`, `locale`, `gettext`.
2. Поддержка цикла разработки. Модули: `pdb`, `hotshot`, `profile`, `unittest`, `pydoc`. Пакеты `docutils`, `distutils`.
3. Взаимодействие с ОС (файлы, процессы). Модули: `os`, `os.path`, `getopt`, `glob`, `popen2`, `shutil`, `select`, `signal`, `stat`, `tempfile`.
4. Обработка текстов. Модули: `string`, `re`, `StringIO`, `codecs`, `difflib`, `mmap`, `sgmllib`, `htmllib`, `htmlentitydefs`. Пакет `xml`.
5. Многопоточные вычисления. Модули: `threading`, `thread`, `Queue`.
6. Хранение данных. Архивация. Модули: `pickle`, `shelve`, `anydbm`, `gdbm`, `gzip`, `zlib`, `zipfile`, `bz2`, `csv`, `tarfile`.
7. Платформо-зависимые модули. Для UNIX: `commands`, `pwd`, `grp`, `fcntl`, `resource`, `termios`, `readline`, `rlcompleter`. Для Windows: `msvcrt`, `_winreg`, `winsound`.
8. Поддержка сети. Протоколы Интернет. Модули: `cgi`, `Cookie`, `urllib`, `urlparse`, `httplib`, `smtplib`, `poplib`, `telnetlib`, `socket`, `asyncore`. Примеры серверов: `SocketServer`, `BaseHTTPServer`, `xmlrpclib`, `asynchat`.
9. Поддержка Internet. Форматы данных. Модули: `quopri`, `uu`, `base64`, `binhex`, `binascii`, `rfc822`, `mimetools`, `MimeWriter`, `multifile`, `mailbox`. Пакет `email`.
10. Python о себе. Модули: `parser`, `symbol`, `token`, `keyword`, `inspect`, `tokenize`, `pyclbr`, `py_compile`, `compileall`, `dis`, `compiler`.
11. Графический интерфейс. Модуль `Tkinter`.

Примечание:

Очень часто модули содержат один или несколько классов, с помощью которых создается объект нужного типа, а затем речь идет уже не об именах из модуля, а об атрибутах этого объекта. И наоборот, некоторые модули содержат лишь функции, достаточно общие для того, чтобы работать над произвольными объектами (либо достаточно большой категорией объектов).

Сервисы периода выполнения

Модуль sys

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

`exit([c])`	Выход из программы. Можно передать числовой код завершения: `0` в случае успешного завершения, другие числа при аварийном завершении программы.
`argv`	Список аргументов командной строки. Обычно `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. Открытые файловые объекты.

`version`	Версия интерпретатора.
`setrecursionlimit(limit)`	Установка уровня максимальной вложенности рекурсивных вызовов.
`exc_info()`	Информация об обрабатываемом исключении.

Модуль copy

Этот модуль содержит функции для копирования объектов. Вначале предлагается к рассмотрению "парадокс", который вводит в замешательство новичков в Python:

```
lst1 = [0, 0, 0]
lst = [lst1] * 3
print lst
lst[0][1] = 1
print lst
```

В результате получается, возможно, не то, что ожидалось:

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[[0, 1, 0], [0, 1, 0], [0, 1, 0]]
```

Дело в том, что список `lst` содержит ссылки на один и тот же список! Для того чтобы действительно размножить список, необходимо применить функцию `copy()` из модуля `copy`:

```
from copy import copy
lst1 = [0, 0, 0]
lst = [copy(lst1) for i in range(3)]
print lst
lst[0][1] = 1
print lst
```

Теперь результат тот, который ожидался:

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[[0, 1, 0], [0, 0, 0], [0, 0, 0]]
```

В модуле copy есть еще и функция `deepcopy()` для глубокого копирования, при которой объекты копируются на всю возможную глубину, рекурсивно.

Модули math и cmath

В этих модулях собраны математические функции для действительных и комплексных аргументов. Это те же функции, что используются в языке C. В таблице ниже даны функции модуля math. Там, где аргумент обозначен буквой z, аналогичная функция определена и в модуле cmath.

Функция или константа	Описание
`acos(z)`	арккосинус `z`
`asin(z)`	арксинус `z`
`atan(z)`	арктангенс `z`

atan2(y,x)	atan(y/x)
ceil(x)	наименьшее целое, большее или равное x
cos(z)	косинус z
cosh(x)	гиперболический косинус x
e	константа e
exp(z)	экспонента (то есть, e**z)
fabs(x)	абсолютное значение x
floor(x)	наибольшее целое, меньшее или равное x
fmod(x,y)	остаток от деления x на y
frexp(x)	возвращает мантиссу и порядок x как пару (m, i), где m - число с плавающей точкой, а i - целое, такое, что x = m * 2.**i. Если 0, возвращает (0,0), иначе 0.5 <= abs(m) < 1.0
hypot(x,y)	sqrt(x*x + y*y)
ldexp(m,i)	m * (2**i)
log(z)	натуральный логарифм z
log10(z)	десятичный логарифм z
modf(x)	возвращает пару (y,q) - целую и дробную часть x. Обе части имеют знак исходного числа
pi	константа пи
pow(x,y)	x**y
sin(z)	синус z
sinh(z)	гиперболический синус z
sqrt(z)	корень квадратный от z
tan(z)	тангенс z
tanh(z)	гиперболический тангенс z

Модуль random

Этот модуль генерирует псевдослучайные числа для нескольких различных распределений. Наиболее используемые функции:

random()	Генерирует псевдослучайное число из полуоткрытого диапазона [0.0, 1.0).
choice(s)	Выбирает случайный элемент из последовательности s.
shuffle(s)	Размешивает элементы изменчивой последовательности s на месте.
randrange([start,] stop[, step])	Выдает случайное целое число из диапазона range(start, stop, step). Аналогично choice(range(start, stop, step)).
normalvariate(mu, sigma)	Выдает число из последовательности нормально распределенных псевдослучайных чисел. Здесь mu - среднее, sigma - среднеквадратическое отклонение (sigma > 0)

Остальные функции и их параметры можно уточнить по документации. Следует отметить, что в модуле есть функция seed(n), которая позволяет установить генератор случайных чисел в некоторое состояние. Например, если возникнет необходимость многократного использования одной и той же последовательности псевдослучайных чисел.

Модуль time

Этот модуль дает функции для получения текущего времени и преобразования форматов времени.

Модуль sets

Модуль реализует тип данных для множеств. Следующий пример показывает, как использовать этот модуль. Следует заметить, что в Python 2.4 и старше тип `set` стал встроенным, и вместо `sets.Set` можно использовать `set`:

```
import sets
A = sets.Set([1, 2, 3])
B = sets.Set([2, 3, 4])
print A | B, A & B, A - B, A ^ B
for i in A:
  if i in B:
    print i,
```

В результате будет выведено:

```
Set([1, 2, 3, 4]) Set([2, 3]) Set([1]) Set([1, 4])
2 3
```

Модули array и struct

Эти модули реализуют низкоуровневый массив и структуру данных. Основное их назначение - разбор двоичных форматов данных.

Модуль itertools

Этот модуль содержит набор функций для работы с **итераторами**. Итераторы позволяют работать с данными последовательно, как если бы они получались в цикле. Альтернативный подход - использование списков для хранения промежуточных результатов - требует подчас большого количества памяти, тогда как использование итераторов позволяет получать значения на момент, когда они действительно требуются для дальнейших вычислений. Итераторы будут рассмотрены более подробно в лекции по функциональному программированию.

Модуль locale

Модуль `locale` применяется для работы с культурной средой. В конкретной культурной среде могут использоваться свои правила для написания чисел, валют, времени и даты и т.п. Следующий пример выводит дату сначала в культурной среде "C", а затем на русском языке:

```
import time, locale
locale.setlocale(locale.LC_ALL, None)
print time.strftime("%d %B %Y", time.localtime (time.time()))
locale.setlocale(locale.LC_ALL, "ru_RU.KOI8-R")
print time.strftime("%d %B %Y", time.localtime (time.time()))
```

В результате:

```
18 November 2004
18 Ноября 2004
```

Модуль gettext

При интернационализации программы важно не только предусмотреть возможность использования нескольких культурных сред, но и перевод сообщений и меню программы на соответствующий язык. Модуль `gettext` позволяет упростить этот процесс достаточно стандартным способом. Основные сообщения программы пишутся на английском языке. А переводы строк, отмеченных в программе специальным образом, даются в виде отдельных файлов, по одному на каждый язык (или культурную среду). Уточнить нюансы использования `gettext` можно по документации к Python.

Поддержка цикла разработки

Модули этого раздела помогают поддерживать документацию, производить регрессионное тестирование, отлаживать и профилировать программы на Python, а также обслуживают распространение готовых программ, создавая среду для конфигурирования и установки пакетов.

В качестве иллюстрации можно предположить, что создается модуль для вычисления простых чисел по алгоритму "решето Эратосфена". Модуль будет находиться в файле `Sieve.py` и состоять из одной функции `primes(N)`, которая в результате своей работы дает все простые (не имеющие натуральных делителей кроме себя и единицы) числа от `2` до `N`:

```
import sets
import math
"""Модуль для вычисления простых чисел от 2 до N """
def primes(N):
  """Возвращает все простые от 2 до N"""
  sieve = sets.Set(range(2, N))
  for i in range(2, math.sqrt(N)):
    if i in sieve:
      sieve -= sets.Set(range(2*i, N, i))
  return sieve
```

Модуль pdb

Модуль `pdb` предоставляет функции отладчика с интерфейсом - командной строкой. Сессия отладки вышеприведенного модуля могла бы быть такой:

```
>>> import pdb
>>> pdb.runcall(Sieve.primes, 100)
> /home/rnd/workup/intuit-python/examples/Sieve.py(15)primes()
-> sieve = sets.Set(range(2, N))
(Pdb) l
 10     import sets
 11     import math
 12     """Модуль для вычисления простых чисел от 2 до N """
 13     def primes(N):
 14       """Возвращает все простые от 2 до N"""
 15  ->   sieve = sets.Set(range(2, N))
 16       for i in range(2, int(math.sqrt(N))):
 17         if i in sieve:
 18           sieve -= sets.Set(range(2*i, N, i))
 19       return sieve
 20
```

```
(Pdb) n
> /home/rnd/workup/intuit-python/examples/Sieve.py(16)primes()
-> for i in range(2, int(math.sqrt(N))):
(Pdb) n
> /home/rnd/workup/intuit-python/examples/Sieve.py(17)primes()
-> if i in sieve:
(Pdb) n
> /home/rnd/workup/intuit-python/examples/Sieve.py(18)primes()
-> sieve -= sets.Set(range(2*i, N, i))
(Pdb) n
> /home/rnd/workup/intuit-python/examples/Sieve.py(16)primes()
-> for i in range(2, int(math.sqrt(N))):
(Pdb) p sieve
Set([2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35,
37, 39,
41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75,
77, 79,
81, 83, 85, 87, 89, 91, 93, 95, 97, 99])
(Pdb) n
> /home/rnd/workup/intuit-python/examples/Sieve.py(17)primes()
-> if i in sieve:
(Pdb) n
> /home/rnd/workup/intuit-python/examples/Sieve.py(18)primes()
-> sieve -= sets.Set(range(2*i, N, i))
(Pdb) n
> /home/rnd/workup/intuit-python/examples/Sieve.py(16)primes()
-> for i in range(2, int(math.sqrt(N))):
(Pdb) p sieve
Set([2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37, 41, 43, 47, 49,
53, 55, 59, 61, 65, 67, 71, 73, 77, 79, 83, 85, 89, 91, 95, 97])
```

Модуль profile

С помощью **профайлера** разработчики программного обеспечения могут узнать, сколько времени занимает исполнение различных функций и методов.

Продолжая пример с решетом Эратосфена, стоит посмотреть, как тратится процессорное время при вызове функции `primes()`:

```
>>> profile.run("Sieve.primes(100000)")
        709 function calls in 1.320 CPU seconds

  Ordered by: standard name

  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1    0.010    0.010    1.320    1.320 <string>:1(?)
       1    0.140    0.140    1.310    1.310 Sieve.py:13(primes)
       1          0.000          0.000          1.320          1.320
profile:0(Sieve.primes(100000))
       0    0.000             0.000          profile:0(profiler)
      65    0.000    0.000    0.000    0.000 sets.py:119(__iter__)
     314    0.000    0.000    0.000    0.000 sets.py:292(__contains__)
      65          0.000          0.000          0.000          0.000
sets.py:339(_binary_sanity_check)
      66    0.630    0.010    0.630    0.010 sets.py:356(_update)
      66    0.000    0.000    0.630    0.010 sets.py:425(__init__)
      65    0.010    0.000    0.540    0.008 sets.py:489(__isub__)
      65          0.530          0.008          0.530          0.008
sets.py:495(difference_update)
```

Здесь `ncalls` - количество вызовов функции или метода, `tottime` - полное время выполнения кода функции (без времени нахождения в вызываемых функциях), `percall` - тоже, в пересчете на один вызов, `cumtime` - аккумулированное время нахождения в

функции, вместе со всеми вызываемыми функциями. В последнем столбце приведено имя файла, номер строки с функцией или методом и его имя.

Примечание:

"Странные" имена, например, `__iter__`, `__contains__` и `__isub__` - имена методов, реализующих итерацию по элементам, проверку принадлежности элемента (`in`) и операцию `-=`. Метод `__init__` - конструктор объекта (в данном случае - множества).

Модуль unittest

При разработке программного обеспечения рекомендуется применять так называемые **регрессионные испытания**. Для каждого модуля составляется набор тестов, по возможности таким образом, чтобы проверялись не только типичные вычисления, но и "крайние", вырожденные случаи, чтобы испытания затронули каждую ветку алгоритма хотя бы один раз. Тест для данного модуля (написанный сразу после того, как определен интерфейс модуля) находится в файле `test_Sieve.py`:

```
# file: test_Sieve.py
import Sieve, sets
import unittest

class TestSieve(unittest.TestCase):

    def setUp(self):
        pass

    def testone(self):
        primes = Sieve.primes(1)
        self.assertEqual(primes, sets.Set())

    def test100(self):
        primes = Sieve.primes(100)
        self.assert_(primes == sets.Set([2, 3, 5, 7, 11, 13, 17, 19, 23,
29, 31, 37, 41, 43, 47,
                    53, 59, 61, 67, 71, 73, 79, 83, 89, 97]))

if __name__ == '__main__':
    unittest.main()
```

Тестовый модуль состоит из определения класса, унаследованного от класса `unittest.TestCase`, в котором описывается подготовка к испытаниям (метод `setUp`) и сами испытания -- методы, начинающиеся на `test`. В данном случае таких испытаний всего два: в первом испытывается случай `N=1`, а во втором -- `N=100`.

Запуск тестов производится выполнением функции `unittest.main()`. Вот как выглядят успешные испытания:

```
$ python test_Sieve.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
```

В процессе разработки перед каждым выпуском все модули прогоняются через регрессионные испытания, чтобы обнаружить, не были ли внесены ошибки. Однако никакие тесты в общем случае не могут гарантировать безошибочности сложной программы. При дополнении модулей тесты также могут быть дополнены, чтобы отразить изменения в проекте.

Кстати, сам Python и его стандартная библиотека имеют тесты для каждого модуля - они находятся в каталоге `test` в месте, где развернуты файлы поставки Python, и являются частью пакета `test`.

Модуль pydoc

Успех проекта зависит не только от обеспечения эффективного и качественного кода, но и от качества документации. Утилита `pydoc` аналогична команде `man` в Unix:

```
$ pydoc Sieve
Help on module Sieve:

NAME
    Sieve - Модуль для вычисления простых чисел от 2 до N

FILE
    Sieve.py

FUNCTIONS
    primes(N)
        Возвращает все простые от 2 до N
```

Эта страница помощи появилась благодаря тому, что были написаны строки документации - как ко всему модулю, так и к функции `primes(N)`.

Стоит попробовать запустить `pydoc` следующей командой:

```
pydoc -p 8088
```

И направить браузер на URL http://127.0.0.1:8088/ - можно получить документацию по модулям Python в виде красивого web-сайта.

Узнать другие возможности `pydoc` можно, подав команду `pydoc pydoc`.

Пакет docutils

Этот пакет и набор утилит пока что не входит в стандартную поставку Python, однако о нем нужно знать тем, кто хочет быстро готовить документацию (руководства пользователя и т.п.) для своих модулей. Этот пакет использует специальный язык разметки (ReStructuredText), из которого потом легко получается документация в виде HTML, LaTeX и в других форматах. Текст в формате RST легко читать и в исходном виде. С этим инструментом можно познакомиться на http://docutils.sourceforge.net

Пакет distutils

Данный пакет предоставляет стандартный путь для распространения собственных Python-пакетов. Достаточно написать небольшой конфигурационный файл `setup.py`, использующий `distutils`, и файл с перечислением файлов проекта `MANIFEST.in`, чтобы пользователи пакета смогли его установить командой

```
python setup.py install
```

Тонкости работы с `distutils` можно изучить по документации.

Взаимодействие с операционной системой

Различные операционные системы имеют свои особенности. Здесь рассматривается основной модуль этой категории, функции которого работают на многих операционных системах.

Модуль os

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

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. Например, с помощью переменных окружения web-сервер передает некоторые параметры в CGI-сценарий. В следующем примере можно получить переменную окружения PATH:

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

Большая группа функций посвящена работе с файлами и каталогами. Ниже приводятся только те, которые доступны как в Unix, так и в Windows.

access(path, flags)	Проверка доступности файла или каталога с именем path. Режим запрашиваемого доступа указывается значением flags, составленных комбинацией (побитовым ИЛИ) флагов os.F_OK (файл существует), os.R_OK (из файла можно читать), os.W_OK (в файл можно писать) и os.X_OK (файл можно исполнять, каталог можно просматривать).
chdir(path)	Делает path текущим рабочим каталогом.
getcwd()	Текущий рабочий каталог.
chmod(path, mode)	Устанавливает режим доступа к path в значение mode. Режим доступа можно получить, скомбинировав флаги (см. ниже). Следует заметить, что chmod() не дополняет действующий режим, а устанавливает его заново.
listdir(dir)	Возвращает список файлов в каталоге dir. В список не входят специальные значения "." и "..".
mkdir(path[, mode])	Создает каталог path. По умолчанию режим mode равен 0777, то есть: S_IRWXU|S_IRWXG|S_IRWXO, если пользоваться константами модуля stat.
makedirs(path[, mode])	Аналог mkdir(), создающий все необходимые каталоги, если они не существуют. Возбуждает исключение, когда последний каталог уже существует.
remove(path), unlink(path)	Удаляет файл path. Для удаления каталогов используются rmdir() и removedirs().
rmdir(path)	Удаляет пустой каталог path.
removedirs(path)	Удаляет path до первого непустого каталога. В случае если самый последний вложенный подкаталог в указанном пути - не пустой,

	возбуждается исключение `OSError`.
`rename(src, dst)`	Переименовывает файл или каталог `src` в `dst`.
`renames(src, dst)`	Аналог `rename()`, создающий все необходимые каталоги для пути `dst` и удаляющий пустые каталоги пути `src`.
`stat(path)`	Возвращает информацию о `path` в виде не менее чем десятиэлементного кортежа. Для доступа к элементам кортежа можно использовать константы из модуля `stat`, например `stat.ST_MTIME` (время последней модификации файла).
`utime(path, times)`	Устанавливает значения времен последней модификации (`mtime`) и доступа к файлу (`atime`). Если `times` равен `None`, в качестве времен берется текущее время. В других случаях `times` рассматривается как двухэлементный кортеж (`atime`, `mtime`). Для получения `atime` и `mtime` некоторого файла можно использовать `stat()` совместно с константами модуля `stat`.

Для работы с процессами модуль `os` предлагает следующие функции (здесь упомянуты только некоторые, доступные как в Unix, так и в Windows):

`abort()`	Вызывает для текущего процесса сигнал `SIGABRT`.
`system(cmd)`	Выполняет командную строку `cmd` в отдельной оболочке, аналогично вызову `system` библиотеки языка C. Возвращаемое значение зависит от используемой платформы.
`times()`	Возвращает кортеж из пяти элементов, содержащий время в секундах работы процесса, ОС (по обслуживанию процесса), дочерних процессов, ОС для дочерних процессов, а также время от фиксированного момента в прошлом (например, от момента запуска системы).
`getloadavg()`	Возвращает кортеж из трех значений, соответствующих занятости процессора за последние 1, 5 и 15 минут.

Модуль stat

В этом модуле описаны константы, которые можно использовать как индексы к кортежам, применяемым функциями `os.stat()` и `os.chmod()` (а также некоторыми другими). Их можно уточнить в документации по Python.

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.

Обработка текстов

Модули этой категории будут подробно рассмотрены в отдельной лекции.

Многопоточные вычисления

Модули этой категории станут предметом рассмотрения отдельной лекции.

Хранение данных. Архивация

К этой категории отнесены модули, которые работают с внешними хранилищами данных.

Модуль pickle

Процесс записи объекта в виде последовательности байтов называется **сериализацией**. Для того чтобы сохранить объект во внешней памяти или передать его по каналам связи, его нужно вначале сериализовать.

Модуль `pickle` позволяет сериализовывать объекты и сохранять их в строке или файле. Следующие объекты могут быть сериализованы:

- встроенные типы: `None`, числа, строки (обычные и Unicode).
- списки, кортежи и словари, содержащие только сериализуемые объекты.
- функции, определенные на уровне модуля (сохраняется имя, но не реализация!).
- встроенные функции.
- классы, определенные на уровне модуля.
- объекты классов, `__dict__` или `__setstate__()` которые являются сериализуемыми.

Типичный вариант использования модуля приведен ниже.

Сохранение:

```
import pickle, time
mydata = ("abc", 12, [1, 2, 3])
output_file = open("mydata.dat", "w")
p = pickle.Pickler(output_file)
p.dump(mydata)
output_file.close()
```

Восстановление:

```
import pickle
input_file = open("mydata.dat", "r")
mydata = pickle.load(input_file)
print mydata
input_file.close()
```

Модуль shelve

Для хранения объектов в родном для Python формате можно применять полку (shelve). По своему интерфейсу полка ничем не отличается от словаря. Следующий пример показывает, как использовать полку:

```
import shelve
data = ("abc", 12)      # - данные (объект)
key = "key"             # - ключ (строка)
filename = "polka.dat"  # - имя файла для хранения полки
```

```
d = shelve.open(filename)       # открытие полки
d[key] = data           # сохранить данные под ключом key
                        # (удаляет старое значение, если оно было)
data = d[key]           # загрузить значение по ключу
len(d)                  # получить количество объектов на полке
d.sync()                # запись изменений в БД на диске
del d[key]              # удалить ключ и значение
flag = d.has_key(key)   # проверка наличия ключа
lst = d.keys()          # список ключей
d.close()               # закрытие полки
```

Модули anydbm и gdbm

Для внешнего хранения данных можно использовать примитивные базы данных, содержащие пары ключ-значение. В Python имеется несколько модулей для работы с такими базами: `bsddb`, `gdbm`, `dbhash` и т.п. Модуль `anydbm` выбирает один из имеющихся хэшей, поэтому его можно применять для чтения ряда форматов (`any` - любой).

Доступ к хэшу из Python мало отличается от доступа к словарю. Разница лишь в том, что хэш еще нужно открыть для создания, чтения или записи, а затем закрыть. Кроме того, при записи хэш блокируется, чтобы не испортить данные.

Модуль csv

Формат CSV (comma separated values - значения, разделенные запятыми) достаточно популярен для обмена данными между электронными таблицами и базами данных. Следующий ниже пример посвящен записи в CSV-файл и чтению из него:

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

# Запись в файл:
f = file("my.csv", "w")
writer = csv.writer(f)
for row in mydata:
    writer.writerow(row)
f.close()

# Чтение из файла:
reader = csv.reader(file("my.csv"))
for row in reader:
    print row
```

Платформо-зависимые модули

Эта категория модулей имеет применение только для конкретных операционных систем и семейств операционных систем. Довольно большое число модулей в стандартной поставке Python посвящено трем платформам: Unix, Windows и Macintosh.

При создании переносимых приложений использовать платформо-зависимые модули можно только при условии реализации альтернативных веток алгоритма, либо с отказом от свойств, которые доступны не на всех платформах. Так, под Windows не работает достаточно обычная для Unix функция `os.fork()`, поэтому при создании переносимых приложений нужно использовать другие средства для распараллеленных вычислений, например, многопоточность.

В документации по языку обычно отмечено, для каких платформ доступен тот или иной модуль или даже отдельная функция.

Поддержка сети. Протоколы Интернет

Почти все модули из этой категории, обслуживающие клиентскую часть протокола, построены по одному и тому же принципу: из модуля необходим только класс, объект которого содержит информацию о соединении с сервером, а методы реализуют взаимодействие с сервером по соответствующему протоколу. Таким образом, чем сложнее протокол, тем больше методов и других деталей требуется для реализации клиента.

Примеры серверов используются по другому принципу. В модуле с реализацией сервера описан базовый класс, из которого пользователь модуля должен наследовать свой класс, реализующий требуемую функциональность. Правда, иногда замещать нужно всего один или два метода.

Этому вопросу будет посвящена отдельная лекция.

Поддержка Internet. Форматы данных

В стандартной библиотеке Python имеются разноуровневые модули для работы с различными форматами, применяющимися для кодирования данных в сети Интернет и тому подобных приложениях.

Сегодня наиболее мощным инструментом для обработки сообщений в формате RFC 2822 является пакет email. С его помощью можно как разбирать сообщения в удобном для программной обработки виде, так и формировать сообщение на основе данных о полях и основном содержимом (включая вложения).

Python о себе

Язык Python является рефлективным языком, в котором можно "заглянуть" глубоко в собственные внутренние структуры кода и данных. Модули этой категории дают возможность прикоснуться к внутреннему устройству Python. Более подробно об этом рассказывается в отдельной лекции.

Графический интерфейс

Почти все современные приложения имеют графический интерфейс пользователя. Такие приложения можно создавать и на языке Python. В стандартной поставке имеется модуль Tkinter, который есть не что иное, как интерфейс к языку Tcl/Tk, на котором можно описывать графический интерфейс.

Следует отметить, что существуют и другие пакеты для программирования графического интерфейса: wxPython (основан на wxWindows), PyGTK и т.д. Среди этих пакетов в основном такие, которые работают на одной платформе (реже - на двух).

Помимо возможностей программного описания графического интерфейса, для Python есть несколько коммерческих и некоммерческих **построителей графического интерфейса** (GUI builders), однако в данном курсе они не рассматриваются.

Заключение

В этой лекции говорилось о встроенных функциях языка Python и модулях его стандартной библиотеки. Некоторые направления будут рассмотрены более подробно в следующих лекциях. Python имеет настолько обширную стандартную библиотеку, что в

рамках одной лекции можно только сделать ее краткий обзор, подкрепив небольшими примерами наиболее типичные идиомы при использовании модулей.