Роман Сузи

bhv®

www.bhv.ru
www.bhv.kiev.ua

Python

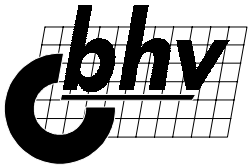
- Библиотеки Python Image Library и Numeric Python
- Работа с базами данных
- Среда разработки Web-приложений Zope

Наиболее полное руководство

В ПОДЛИННИКЕ

Роман Сузи

Python



Дюссельдорф ◆ Киев ◆ Москва ◆ Санкт-Петербург

УДК 681.3.06

Описываются синтаксис и семантика типов данных, операций, конструкций и операторов языка Python; функции, специальные методы классов и исключения. Рассматриваются расширения языка: библиотека Numeric Python, позволяющая эффективно работать с массивами числовых данных, и библиотека Python Image Library, предназначенная для обработки растровых изображений. Обсуждаются работа с базами данных, возможности разработки графического интерфейса пользователя, интеграция программного продукта с модулями на языках SQL, Tcl/Tk, C, C++ и Java. Особое внимание уделено средствам Web-программирования.

Для программистов и Web-разработчиков

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Руководитель проекта	*Анатолий Адаменко*
Зав. редакцией	*Анна Кузьмина*
Редактор	*Семен Штейнер*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Сузи Р. А.
Python. — СПб.: БХВ-Петербург, 2002. — 768 с.: ил.

ISBN 5-94157-097-X

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 17.12.01.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 61,92.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.1.953.П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН.
199034, Санкт-Петербург, 9 линия, 12.

Содержание

Глава 1

Интерактивный Python

В этой главе мы познакомимся с Python — новым интересным языком программирования. Он задумывался для тех, кто хочет с легкостью и изяществом решать даже трудные задачи, и подходит для тех, кто хочет научиться программировать.

1.1. Запускаем Python

Для диалога с Python нам необходимо прежде всего запустить интерпретатор Python. Упрощенно, *интерпретатор* — это такая программа, которая последовательно, одну за другой, читает команды языка программирования (они называются операторами), и переводит их в машинные команды, которые исполняет процессор компьютера.

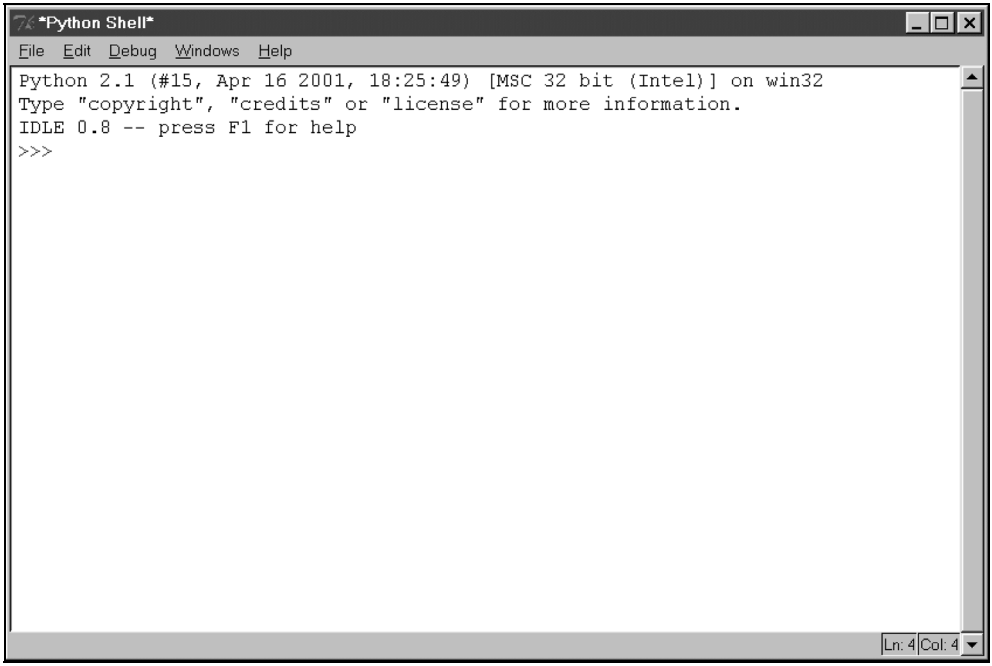


Рис. 1.1. Первый запуск IDLE

В системе Windows интерпретатор обычно запускается вместе с графической оболочкой IDLE (рис. 1.1) из **Программы | Python | IDLE (Python GUI)**, но можно запустить только интерпретатор, выбрав команду **Программы | Python | Python (Command line)**. И то, и другое даст нам возможность вводить команды, которые Python будет исполнять.

В системе UNIX и Linux также можно запустить Python просто командой `python`, а можно запустить IDLE (если вы любите работать в X Window) — командой `idle`. (В UNIX режим командной строки гораздо богаче возможностями редактирования, чем в Windows).

Сразу после запуска Python нас приветствует, например, так:

```
Python 2.1 (#1, Apr 21 2001, 13:46:58)
[GCC egcs-2.91.66 19990314/Linux (egcs-1.1.2 release)] on linux2
Type "copyright", "credits" or "license" for more information.
>>>
```

Три знака `">"` — *приглашение ко вводу*. Мы можем вводить наши команды.

1.2. Диалог с Python

Наш Python готов к диалогу, о чем говорит `>>>`. Для начала спросим его о погоде на сегодня:

```
>>> Какая сегодня погода?
  File "<stdin>", line 1
    Какая сегодня погода?
                ^
SyntaxError: invalid syntax
>>>
```

К сожалению, уважаемый Python не понимает русского языка, и нам придется учить его язык. Но что же он нам сказал, указав на нашу ошибку? `SyntaxError` — синтаксическая ошибка, а более конкретно `invalid syntax` — неверный синтаксис. Высказавшись, Python снова предложил продолжить общение.

```
>>> 2+2
4
```

Вот это ему понятно. Пробуем дальше (`print` означает "печатай"):

```
>>> print "Сегодня хороший день!"
Сегодня хороший день!
>>> print "Сегодня хороший день!", 1, "мая!"
Сегодня хороший день! 1 мая!
```

```
>>> print "Завтра будет", 1+1, "мая!"
Завтра будет 2 мая!
>>> print "Послезавтра будет", 1+ "мая!"
Послезавтра будет
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: unsupported operand types for +
>>>
```

Кажется, мы увлеклись и забыли, что компьютер — вещь точная, и требует четкости при работе с ним. На этот раз произошла ошибка типа `TypeError`, потому что нельзя складывать числа и строки, как это случайно произошло в диалоге выше. Ничего страшного, мы исправим ошибку и продолжим общение:

```
>>> print "Послезавтра будет", 1+2, "мая!"
Послезавтра будет 3 мая!
>>>
```

Кстати, строки с текстом тоже можно "складывать":

```
>>> print "Послезавтра" + "будет" + "май!"
Послезавтрабудетмай!
>>> print "Послезавтра " + "будет " + "май!"
Послезавтра будет май!
```

А вот арифметические операции в кавычках исполняться, естественно, не будут:

```
>>> print "2+2"
2+2
```

Python все равно, использованы ли кавычки или апострофы. Кроме того, для записи строк текста можно применять утроенные кавычки или апострофы:

```
>>> print """Текст можно писать и в таких кавычках, """ 'и в таких.'
Текст можно писать и в таких кавычках, и в таких.
>>> print "Но не в таких:", `2+2`
Но не в таких: 4
```

Надеемся, что читатель неплохо провел время в легкой беседе с интерпретатором Python и готов к более серьезному общению в следующем разделе.

Упражнение

Сложите яблоки с апельсинами.

1.3. Python как калькулятор

В этом разделе мы научимся использовать Python в качестве калькулятора. Запустите Python (*см. разд. 1.1*, если вы забыли, как это делается) и за дело.

Python умеет производить простейшие арифметические операции: сложение, вычитание, умножение, деление и даже получение остатка от деления. Причем делать он это может как с целыми числами, так и с десятичными дробями, а также с комплексными числами. В записываемых числовых примерах (их грамотно называть *выражениями*) можно расставлять скобки так, как нужно для правильных вычислений. В отличие от многих калькуляторов, Python знает, что умножение, деление и получение остатка от деления имеет бóльший приоритет, чем сложение или вычитание. Давайте посмотрим, как Python справится со следующими выражениями:

```
>>> 2+3*4
14
>>> 2*3+4
10
>>> (2+3)*4
20
>>> 10/2
5
>>> 10/3
3
>>> 12.48/2
6.24
>>> 13%5
3
```

В первых примерах все ясно. Операция умножения обычно обозначается в языках программирования как `"*"`, деление — как `"/"`, а операция получения остатка от деления — как `"%"`. Вы, наверное, заметили, что десятичные дроби пишутся через точку (`12.48`), несмотря на то, что в России принято писать через запятую. Запись дробей с точкой традиционна для многих языков и систем программирования. Кроме того, результат `10/3` равен `3`. Дело в том, что и `10`, и `3` — целые числа, и потому Python осуществляет деление нацело. Для того чтобы разделить числа "обычным" образом, нужно записать хотя бы одно из них как десятичную дробь — с точкой:

```
>>> 10.0/3
3.33333333333
>>> 11./3
```

```
3.66666666667
>>> 10/3.
3.33333333333
```

Теперь можно решать примеры любой сложности:

```
>>> (12.3 + 12)/(12 - (78.3+5*6))
-0.252336448598
>>> _ + 5
4.7476635514
```

Одиночное подчеркивание "_" является специальным именем для обозначения результата предыдущего вычисления.

> ***Упражнение***
>
> Составьте выражение на Python, которое вычисляет стоимость покупки яблок, мандаринов и бананов, если известны их вес и цена за килограмм.

Следующий пример показывает, что некоторые операции над большими числами приводят к переполнениям, о которых Python тут же заявляет:

```
>>> 1000000000*1000000000
Traceback (innermost last):
  File "<stdin>", line 1, in ?
OverflowError: integer multiplication
```

Это было сказано для того, чтобы, пользуясь имеющейся в Python операцией возведения в степень (обозначается `**`), вы не удивлялись возникающим переполнениям:

```
print "1 килобайт =", 2**10, "байт"
1 килобайт = 1024 байт
print "1 мегабайт =", 1024 ** 2, "байт"
1 мегабайт = 1048576 байт
print "1 гигабайт =", 1024 ** 3, "байт"
1 гигабайт = 1073741824 байт
>>> 2**100
Traceback (innermost last):
  File "<stdin>", line 1, in ?
OverflowError: integer pow()
```

> ***Упражнение***
>
> В какую наибольшую степень можно возвести число 2 без переполнения?

Python может стать "инженерным" калькулятором, если перед вычислениями подать специальную команду, загружающую в память модуль с различными математическими функциями *(см. разд. 9.4.1)*. Ниже мы будем использовать `sqrt` (квадратный корень), `sin` (синус), `cos` (косинус), `atan` (арктангенс).

```
>>> from math import *
>>> sqrt(16)
4.0
>>> sqrt(18.49)
4.3
>>> sin(3)
0.14112000806
>>> sin(1.22)**2 + cos(1.22)**2
1.0
```

Упражнение

Поработав с функцией `fabs()`, попытайтесь установить, что она делает.

Работая с обычными калькуляторами, бывает очень обидно, что с таким трудом вычисленные значения нужно стирать перед выполнением других вычислений. В Python результатам вычислений можно давать *имена*:

```
>>> from math import *
>>> x = sin(0.5) + atan(2.3)
>>> y = 7
>>> x*y
11.480661674
```

В этом примере Python, вычислив `sin(0.5) + atan(2.3)`, дает полученному результату имя `x`. Аналогично происходит и с `y`, хотя там уже никаких сложных вычислений нет. Говорят, что (переменной) `y` присваивается значение `7`. А дальше мы просто используем имена переменных в вычисляемых выражениях наравне с числами.

Кстати, все тот же модуль `math` содержит имена, соответствующие приближенным значениям известных констант π и e:

```
>>> from math import *
>>> print "pi =", pi
pi = 3.14159265359
>>> print "e =", e
e = 2.71828182846
```

Так что площадь круга радиуса `r` можно вычислить так:

```
>>> from math import *
>>> r = 3
>>> print "S =", pi * r**2
S = 28.2743338823
```

> ***Упражнение***
>
> Воспользуйтесь интерпретатором Python для вычисления по какой-либо известной вам формуле.

За исключением примера с переполнением, в наших вычислениях участвовали сравнительно небольшие числа. Оказывается, для представления очень больших или очень близких нулю чисел существует свой вид записи — *запись в экспоненциальном формате*. В такой записи записываются лишь действительные числа (числа "с точкой"):

```
>>> 0.0001
0.0001
>>> 0.00001
1e-05
>>> 10.0**-6
1e-06
>>> 100000000000.
100000000000.
>>> 1000000000000.
1e+12
>>> 10.0**13
1e+13
>>> 3.14e5
314000.0
>>> 3.14e-5
3.14e-05
>>> 3.14e-3
0.00314
```

Секрет записи раскрывается просто: число получается в результате умножения *мантиссы* (стоит перед `"e"`) на 10 в степени, указанной после `"e"`. Указанное после `"e"` число называется *порядком*.

> ***Упражнение***
>
> Как в Python запишется миллион миллиардов?

Если же нас интересуют большие целые числа, мы можем использовать целые числа с произвольной точностью. В Python они (скромно) названы *длинными целыми*. Запись длинного целого отличается от записи обычного целого только добавлением "L" на конце:

```
>>> print "1 терабайт =", 1024L ** 4, "байт"
1 терабайт = 1099511627776L байт
>>> print "1 петабайт =", 1024L ** 5, "байт"
1 петабайт = 1125899906842624L байт
```

Числа с произвольной точностью ограничены лишь размером памяти, доступной интерпретатору Python, и степенью вашего терпения при ожидании результатов вычислений.

Упражнение

Вычислите с помощью целых чисел произвольной точности число 2 в десятитысячной степени. А в стотысячной?

Кстати, выполняющиеся вычисления можно прервать с помощью комбинации клавиш <Ctrl>+<C> (Linux) или <Ctrl>+<Break> (Windows). Прервемся и мы, чтобы в следующей главе заняться настоящими программами.