

Н. А. Прохоренко



Python

САМОЕ
НЕОБХОДИМОЕ



Базовый синтаксис языка

Объектно-ориентированное программирование

Работа с файлами и каталогами

Основы SQLite

Интерфейс доступа к базам данных SQLite и MySQL

Создание изображений с помощью библиотеки PIL

Получение данных из Интернета

Примеры и советы из практики



ВИДЕОКУРС

Николай Прохоренок

Python

САМОЕ
НЕОБХОДИМОЕ

Санкт-Петербург

«БХВ-Петербург»

2010

УДК 681.3.068+800.92Python

ББК 32.973.26-018.1

П84

Прохоренок Н. А.

П84 Python. Самое необходимое. — СПб.: БХВ-Петербург, 2011. — 416 с.: ил.
+ Видеокурс (на DVD)

ISBN 978-5-9775-0614-4

Описан базовый синтаксис языка Python: типы данных, операторы, условия, циклы, регулярные выражения, встроенные функции, объектно-ориентированное программирование, часто используемые модули стандартной библиотеки. Даны основы SQLite, описан интерфейс доступа к базам данных SQLite и MySQL. Рассмотрены работа с изображениями с помощью библиотеки PIL и получение данных из Интернета. Книга содержит более двухсот практических примеров, помогающих начать программировать на языке Python самостоятельно. Весь материал тщательно подобран, хорошо структурирован и компактно изложен, что позволяет использовать книгу как удобный справочник.

Прилагаемый DVD содержит листинги описанных в книге примеров и видеоролики.

Для программистов

УДК 681.3.068+800.92Python

ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Анна Кузьмина</i>
Компьютерная верстка	<i>Натальи Смирновой</i>
Корректор	<i>Наталья Першакова</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.07.10.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 33,54.

Тираж 1500 экз. Заказ №

"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

Оглавление

ВВЕДЕНИЕ	1
ГЛАВА 1. ПЕРВЫЕ ШАГИ	3
1.1. Установка Python.....	3
1.2. Первая программа на Python.....	9
1.3. Структура программы.....	11
1.4. Комментарии.....	15
1.5. Скрытые возможности IDLE.....	16
1.6. Вывод результатов работы программы.....	17
1.7. Ввод данных.....	19
1.8. Доступ к документации.....	21
ГЛАВА 2. ПЕРЕМЕННЫЕ	24
2.1. Именованние переменных.....	24
2.2. Типы данных.....	26
2.3. Инициализация переменных.....	29
2.4. Проверка типа данных.....	31
2.5. Преобразование типов данных.....	31
2.6. Удаление переменной.....	33
ГЛАВА 3. ОПЕРАТОРЫ PYTHON	34
3.1. Математические операторы.....	34
3.2. Двоичные операторы.....	36
3.3. Операторы для работы с последовательностями.....	37
3.4. Операторы присваивания.....	37
3.5. Приоритет выполнения операторов.....	38
ГЛАВА 4 . УСЛОВНЫЕ ОПЕРАТОРЫ И ЦИКЛЫ	40
4.1. Операторы сравнения.....	41
4.2. Оператор ветвления <i>if...else</i>	43
4.3. Цикл <i>for</i>	46

4.4. Функции <i>range()</i> , <i>xrange()</i> и <i>enumerate()</i>	48
4.5. Цикл <i>while</i>	50
4.6. Оператор <i>continue</i> . Переход на следующую итерацию цикла.....	52
4.7. Оператор <i>break</i> . Прерывание цикла.....	52
ГЛАВА 5. ЧИСЛА	54
5.1. Встроенные функции для работы с числами	55
5.2. Модуль <i>math</i> . Математические функции	57
5.3. Модуль <i>random</i> . Генерация случайных чисел	59
ГЛАВА 6. СТРОКИ	62
6.1. Создание строки	63
6.2. Специальные символы	66
6.3. Операции над строками	67
6.4. Форматирование строк.....	70
6.5. Метод <i>format()</i>	77
6.6. Функции и методы для работы со строками.....	80
6.7. Настройка локали и изменение регистра символов	84
6.8. Функции для работы с символами	86
6.9. Поиск и замена в строке.....	86
6.10. Проверка типа содержимого строки.....	90
6.11. Преобразование объекта в строку.....	93
6.12. Шифрование строк	94
6.13. Преобразование кодировок	94
ГЛАВА 7. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ	96
7.1. Синтаксис регулярных выражений.....	96
7.2. Поиск первого совпадения с шаблоном	105
7.3. Поиск всех совпадений с шаблоном.....	110
7.4. Замена в строке	111
7.5. Прочие функции и методы	113
ГЛАВА 8. СПИСКИ, КОРТЕЖИ И МНОЖЕСТВА.....	115
8.1. Создание списка	116
8.2. Операции над списками	119
8.3. Многомерные списки	121
8.4. Перебор элементов списка.....	122
8.5. Генераторы списков и выражения-генераторы	123
8.6. Перебор элементов списка без циклов	125
8.7. Добавление и удаление элементов списка	128
8.8. Поиск элемента в списке	130

8.9. Переворачивание и перемешивание списка.....	131
8.10. Выбор элементов случайным образом	132
8.11. Сортировка списка	133
8.12. Заполнение списка числами	135
8.13. Преобразование списка в строку.....	136
8.14. Кортежи	137
8.15. Множества.....	139
ГЛАВА 9. СЛОВАРИ	144
9.1. Создание словаря.....	144
9.2. Операции над словарями	147
9.3. Перебор элементов словаря.....	148
9.4. Методы для работы со словарями.....	149
ГЛАВА 10. РАБОТА С ДАТОЙ И ВРЕМЕНЕМ	152
10.1. Получение текущей даты и времени.....	152
10.2. Форматирование даты и времени	154
10.3. "Засыпание" скрипта	156
10.4. Модуль <i>datetime</i> . Манипуляции датой и временем.....	157
10.4.1. Класс <i>timedelta</i>	157
10.4.2. Класс <i>date</i>	159
10.4.3. Класс <i>time</i>	162
10.4.4. Класс <i>datetime</i>	164
10.5. Модуль <i>calendar</i> . Вывод календаря.....	168
10.5.1. Методы классов <i>TextCalendar</i> и <i>LocaleTextCalendar</i>	169
10.5.2. Методы классов <i>HTMLCalendar</i> и <i>LocaleHTMLCalendar</i>	171
10.5.3. Другие полезные функции	172
10.6. Измерение времени выполнения фрагментов кода.....	174
ГЛАВА 11. ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ	177
11.1. Создание функции и ее вызов	177
11.2. Расположение определений функций.....	180
11.3. Необязательные параметры и сопоставление по ключам	181
11.4. Переменное число параметров в функции.....	184
11.5. Анонимные функции.....	185
11.6. Функции-генераторы.....	186
11.7. Декораторы функций.....	187
11.8. Рекурсия. Вычисление факториала.....	189
11.9. Глобальные и локальные переменные.....	190

ГЛАВА 12. МОДУЛИ И ПАКЕТЫ	194
12.1. Инструкция <i>import</i>	194
12.2. Инструкция <i>from</i>	198
12.3. Пути поиска модулей	200
12.4. Повторная загрузка модулей	202
12.5. Пакеты	202
ГЛАВА 13. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ.....	207
13.1. Определение класса и создание экземпляра класса	207
13.2. Методы <code>__init__()</code> и <code>__del__()</code>	210
13.3. Наследование	211
13.4. Множественное наследование	212
13.5. Классы нового стиля	214
13.6. Специальные методы	215
13.7. Перегрузка операторов.....	218
13.8. Статические методы и методы класса	221
13.9. Абстрактные методы	222
13.10. Ограничение доступа к идентификаторам внутри класса	223
13.11. Свойства класса	225
ГЛАВА 14. ОБРАБОТКА ИСКЛЮЧЕНИЙ.....	227
14.1. Инструкция <i>try...except...else...finally</i>	228
14.2. Инструкция <i>with...as</i>	233
14.3. Классы встроенных исключений	235
14.4. Пользовательские исключения	237
ГЛАВА 15. РАБОТА С ФАЙЛАМИ И КАТАЛОГАМИ	241
15.1. Открытие файла	241
15.2. Методы для работы с файлами.....	246
15.3. Доступ к файлам с помощью модуля <i>os</i>	252
15.4. Модуль <i>StringIO</i>	254
15.5. Права доступа к файлам и каталогам	257
15.6. Функции для манипулирования файлами	259
15.7. Преобразование пути к файлу или каталогу	263
15.8. Перенаправление ввода/вывода	265
15.9. Сохранение объектов в файл	268
15.10. Функции для работы с каталогами	271
ГЛАВА 16. ОСНОВЫ SQLITE	275
16.1. Создание базы данных	276
16.2. Создание таблицы.....	277

16.3. Вставка записей	284
16.4. Обновление и удаление записей	286
16.5. Изменение свойств таблицы	287
16.6. Выбор записей	288
16.7. Выбор записей из нескольких таблиц	291
16.8. Условия в инструкции <i>WHERE</i>	293
16.9. Индексы	296
16.10. Вложенные запросы	299
16.11. Транзакции	300
16.12. Удаление таблицы и базы данных	302
ГЛАВА 17. ДОСТУП К БАЗЕ ДАННЫХ SQLITE ИЗ PYTHON.....	303
17.1. Создание и открытие базы данных	304
17.2. Выполнение запроса.....	305
17.3. Обработка результата запроса	309
17.4. Управление транзакциями	314
17.5. Создание пользовательской сортировки	315
17.6. Поиск без учета регистра символов.....	316
17.7. Создание агрегатных функций	318
17.8. Преобразование типов данных.....	319
17.9. Сохранение в таблице даты и времени.....	323
17.10. Обработка исключений.....	324
ГЛАВА 18. ДОСТУП К БАЗЕ ДАННЫХ MYSQL	328
18.1. Модуль <i>MySQLdb</i>	329
18.1.1. Подключение к базе данных.....	329
18.1.2. Выполнение запроса.....	332
18.1.3. Обработка результата запроса	336
18.2. Модуль <i>PyODBC</i>	339
18.2.1. Подключение к базе данных.....	340
18.2.2. Выполнение запроса.....	341
18.2.3. Обработка результата запроса	343
ГЛАВА 19. БИБЛИОТЕКА PIL. РАБОТА С ИЗОБРАЖЕНИЯМИ	347
19.1. Загрузка готового изображения	347
19.2. Создание нового изображения	350
19.3. Получение информации об изображении	350
19.4. Манипулирование изображением	351
19.5. Рисование линий и фигур	355
19.6. Модуль <i>aggdraw</i>	357
19.7. Вывод текста на изображение	362
19.8. Создание скриншотов	363

Глава 20. Взаимодействие с Интернетом	365
20.1. Разбор URL-адреса	365
20.2. Кодирование и декодирование строки запроса	368
20.3. Преобразование относительной ссылки в абсолютную	372
20.4. Разбор HTML-эквивалентов	373
20.5. Обмен данными по протоколу HTTP	374
20.6. Обмен данными с помощью модуля <i>urllib2</i>	379
20.7. Определение кодировки.....	382
ЗАКЛЮЧЕНИЕ.....	385
Приложение 1. Отличия PYTHON 3 от PYTHON 2.....	389
Приложение 2. Описание DVD.....	395
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ	399

Введение

Добро пожаловать в мир Python!

Python — это интерпретируемый объектно-ориентированный язык программирования высокого уровня, предназначенный для самого широкого круга задач. С его помощью можно обрабатывать различные данные, создавать изображения, работать с базами данных, разрабатывать Web-сайты и приложения с графическим интерфейсом. Python является кроссплатформенным языком, позволяющим создавать программы, которые будут работать во всех операционных системах. В этой книге мы рассмотрим базовые возможности Python 2.6 применительно к операционной системе Windows.

Согласно официальной версии название языка произошло вовсе не от змеи. Создатель языка Гвидо ван Россум (Guido van Rossum) назвал свое творение в честь британского комедийного телешоу BBC "Monty Python's Flying Circus". Поэтому более правильно будет "Пайтон". Тем не менее, многие считают, что для русского человека более привычно называть язык "Питон". Так или иначе, в этой книге мы будем придерживаться традиционного написания слова Python на английском языке.

Программа на языке Python представляет собой обычный текстовый файл с расширением `py` (консольная программа) или `pyw` (программа с графическим интерфейсом). Все инструкции из этого файла выполняются интерпретатором построчно. Для ускорения работы при первом импорте модуля создается промежуточный байт-код и сохраняется в одноименном файле с расширением `pyc`. При последующих запусках, если модуль не был изменен, исполняется именно байт-код. Для выполнения низкоуровневых операций и задач, требующих высокой скорости работы, можно написать модуль на языке C, скомпилировать его, а затем подключить к основной программе.

Python является объектно-ориентированным языком. Это означает, что практически все данные являются объектами, даже сами типы данных. В переменной всегда сохраняется только ссылка на объект, а не сам объект. Например, можно создать функцию, сохранить ссылку на нее в переменной, а затем вызвать функцию через эту переменную. Данное обстоятельство делает язык Python идеальным инструментом для создания программ, использующих функции обратного вызова, например при разработке графического интерфейса. Тот факт, что язык является объектно-ориентированным, отнюдь не означает, что ООП-стиль программирования является обязательным. На языке Python можно писать программы как в ООП-стиле, так и в процедурном стиле. Какой стиль использовать — зависит от конкретной ситуации и предпочтений программиста.

Python — самый стильный язык программирования в мире, не допускающий двоякого написания кода. Например, в языке Perl существует зависимость от контекста и множественность синтаксиса. Часто два программиста, пишущих на Perl, просто не понимают код друг друга. В Python код можно написать только одним способом. В нем отсутствуют лишние конструкции. Все программисты должны придерживаться стандарта, описанного в документе <http://python.org/dev/peps/pep-0008/>. Более читаемого кода нет ни в одном другом языке программирования.

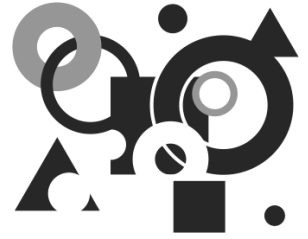
Синтаксис языка Python вызывает много нареканий у программистов, знакомых с другими языками. На первый взгляд может показаться, что отсутствие ограничительных символов (фигурных скобок или конструкции `begin...end`) для выделения блоков и обязательная вставка пробелов впереди инструкций может приводить к ошибкам. Однако это только первое и неправильное впечатление. Хороший стиль программирования в любом языке обязывает выделять инструкции внутри блока одинаковым количеством пробелов. В этой ситуации ограничительные символы просто являются лишними. Бытует мнение, что программа будет по-разному смотреться в разных редакторах. Это неверно. Согласно стандарту для выделения блоков необходимо использовать *четыре пробела*. Четыре пробела в любом редакторе будут смотреться одинаково. Если в другом языке вас не приучили к хорошему стилю программирования, то язык Python быстро это исправит. Если количество пробелов внутри блока будет разным, то интерпретатор выведет сообщение о фатальной ошибке, и программа будет остановлена. Таким образом, язык Python приучает программистов писать красивый и понятный код.

Так как программа на языке Python представляет собой обычный текстовый файл, его можно редактировать с помощью любого текстового редактора, например, с помощью Notepad++. Однако лучше воспользоваться специализированными редакторами, которые не только подсвечивают код, но и выводят различные подсказки и позволяют отладить программу. Таких редакторов очень много, например PyScripter, PythonWin, UliPad, Eclipse + PyDev, Netbeans и др. Полный список редакторов расположен на странице <http://wiki.python.org/moin/PythonEditors>. На всем протяжении этой книги мы будем пользоваться редактором IDLE, который входит в состав стандартной библиотеки Python в Windows. Этот редактор идеально подходит для изучения языка Python.

Ну что, приступим к изучению Python? Язык достоин того, чтобы его знал каждый программист! Но не забывайте, что книги по программированию нужно не только читать, но и выполнять все примеры, а также экспериментировать, изменяя что-нибудь в примерах. Все листинги из книги, а также видеоролики вы найдете на прилагаемом к книге DVD.

Ваши замечания и пожелания можно оставить в гостевой книге или на форуме моего сайта <http://wwwadmin.ru/>. Все замеченные опечатки прошу присылать по адресу mail@bhv.ru.

Желаю приятного прочтения и надеюсь, что эта книга станет верным спутником в вашей повседневной жизни.



Первые шаги

1.1. Установка Python

Прежде чем начать изучение основ языка, необходимо установить на компьютер интерпретатор Python.

1. Для загрузки дистрибутива переходим на страницу <http://python.org/download/> и скачиваем файл `python-2.6.5.msi`. Затем запускаем программу установки с помощью двойного щелчка на значке файла.
2. В открывшемся окне (рис. 1.1) устанавливаем переключатель **Install for all users** (Установить для всех пользователей) и нажимаем кнопку **Next**.
3. На следующем шаге (рис. 1.2) предлагается выбрать каталог для установки. Оставляем каталог по умолчанию (`C:\Python26\`) и нажимаем кнопку **Next**.
4. В следующем диалоговом окне (рис. 1.3) можно выбрать компоненты, которые следует установить. По умолчанию устанавливаются все компоненты и прописывается ассоциация с файловыми расширениями `py`, `pyw` и др. В этом случае запускать программы можно с помощью двойного щелчка на значке файла. Оставляем выбранными все компоненты и нажимаем кнопку **Next**.
5. После завершения установки будет выведено окно, изображенное на рис. 1.4. Нажимаем кнопку **Finish** для выхода из программы установки.

В результате установки исходные файлы интерпретатора будут скопированы в папку `C:\Python26`. В этой папке расположены два исполняемых файла: `python.exe` и `pythonw.exe`. Файл `python.exe` предназначен для выполнения консольных приложений. Именно эта программа запускается при двойном щелчке на значке файла с расширением `py`. Файл `pythonw.exe` используется для запуска оконных приложений. В этом случае окно консоли выводиться не будет. Эта программа запускается при двойном щелчке на значке файла с расширением `pyw`. В этой книге мы будем запускать программы только с помощью файла `python.exe`.

Если сделать двойной щелчок на файле `python.exe`, то запустится интерактивная оболочка в окне консоли (рис. 1.5). Символы `>>>` в этом окне означают приглашение для ввода выражений на языке Python. Если после этих символов ввести, например, `2 + 2` и нажать клавишу `<Enter>`, то на следующей строке сразу будет выведен результат выполнения, а затем опять приглашение для ввода нового выражения.

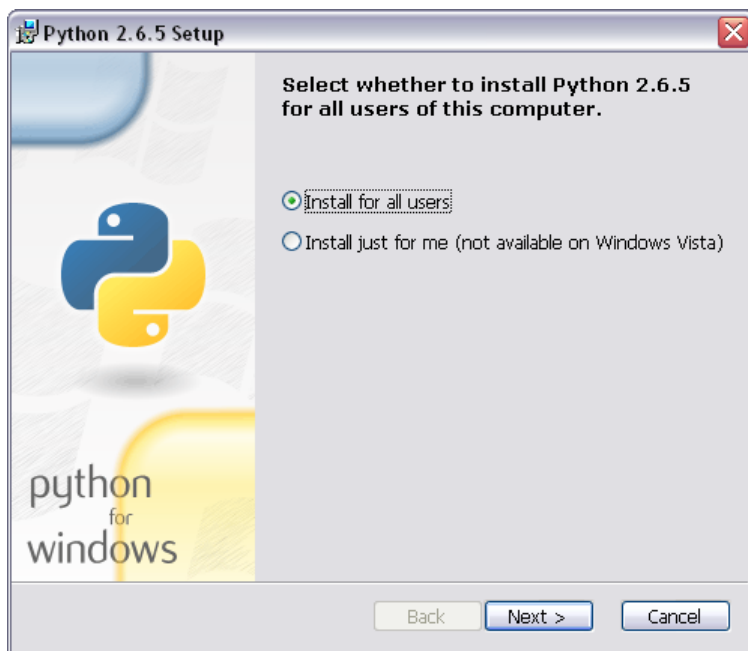


Рис. 1.1. Установка Python. Шаг 1

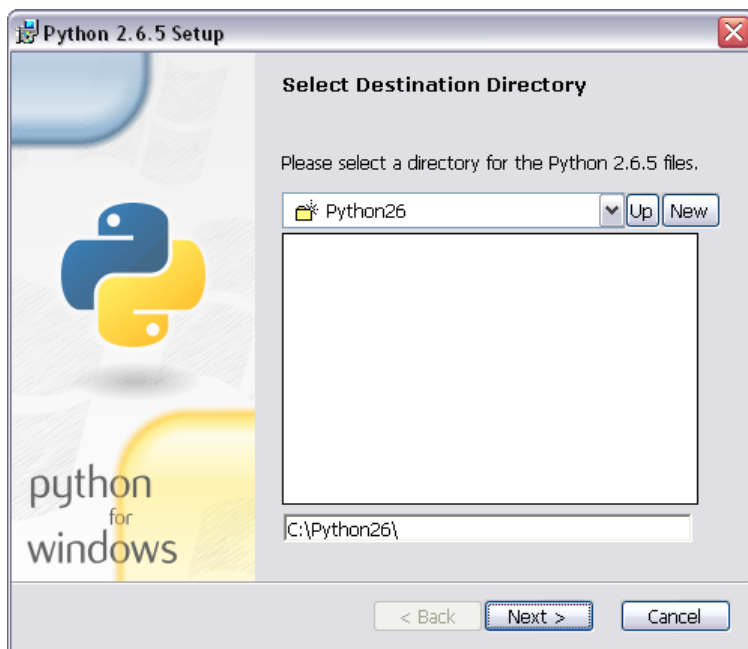


Рис. 1.2. Установка Python. Шаг 2



Рис. 1.3. Установка Python. Шаг 3



Рис. 1.4. Установка Python. Шаг 4

Таким образом, это окно можно использовать в качестве калькулятора, а также для изучения языка. Открыть такое же окно можно с помощью пункта **Python (command line)** в меню **Пуск | Программы | Python 2.6**.

Вместо интерактивной оболочки для изучения языка, а также создания и редактирования файлов с программой, лучше воспользоваться редактором IDLE, который входит в состав установленных компонентов. Для запуска редактора в меню **Пуск | Программы | Python 2.6** выбираем пункт **IDLE (Python GUI)**. В результате откроется окно **Python Shell** (рис. 1.6), которое выполняет все функции интерактивной оболочки, но дополнительно производит подсветку синтаксиса, выводит подсказки и др. Именно этим редактором мы будем пользоваться на протяжении всей книги. Более подробно редактор IDLE мы рассмотрим немного позже.

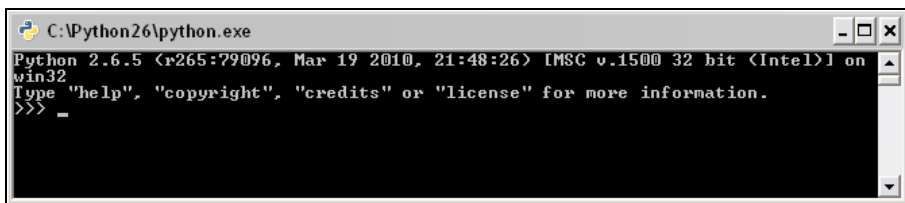


Рис. 1.5. Интерактивная оболочка

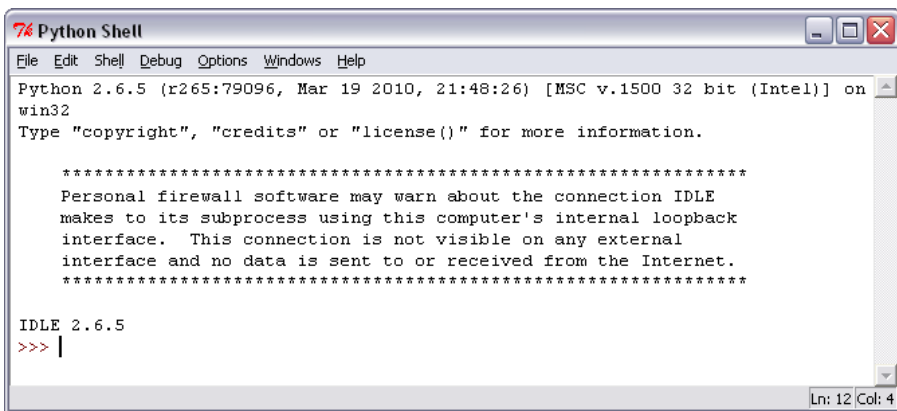


Рис. 1.6. Окно Python Shell редактора IDLE

Версии языка Python выпускаются с завидной регулярностью, но, к сожалению, сторонние разработчики не успевают за такой скоростью и не так часто обновляют свои модули. Поэтому приходится при наличии версии Python 3 использовать на практике версию Python 2.6. В некоторых случаях сторонние модули даже не поддерживают версию 2.6. Как же быть, если установлена версия 2.6, а необходимо запустить модуль для версии 2.5 или хочется попробовать возможности версии 3.1? В этом случае удалять версию 2.6 с компьютера не нужно. Все программы установки позволяют выбрать устанавливаемые компоненты. Так, например, существу-

ет возможность задать ассоциацию с файловым расширением. И вот этот компонент необходимо просто отключить при установке.

В качестве примера мы установим на компьютер еще две версии: 2.5.5 и 3.1.2, но вместо программ установки с сайта <http://python.org/> выберем альтернативный дистрибутив от компании ActiveState. Переходим на страницу <http://www.activestate.com/activepython/downloads/> и скачиваем файлы ActivePython-2.5.5.7-win32-x86.msi и ActivePython-3.1.2.3-win32-x86.msi. Последовательность запуска этих программ имеет значение, т. к. при установке в контекстное меню добавляется пункт **Edit with Pythonwin**. С помощью этого пункта запускается редактор PythonWin, который можно использовать вместо IDLE. Так вот из контекстного меню будет открываться версия PythonWin, которая была установлена последней. Поэтому сначала лучше установить версию 2.5, а затем версию 3.1. Установку программ производим в каталоги по умолчанию (C:\Python25\ и C:\Python31\). Обратите особое внимание: при установке в окне **Custom Setup** (рис. 1.7) необходимо отключить компонент **Register as Default Python** (рис. 1.8). Не забудьте это сделать при установке каждой версии, иначе текущей версией будет не Python 2.6.

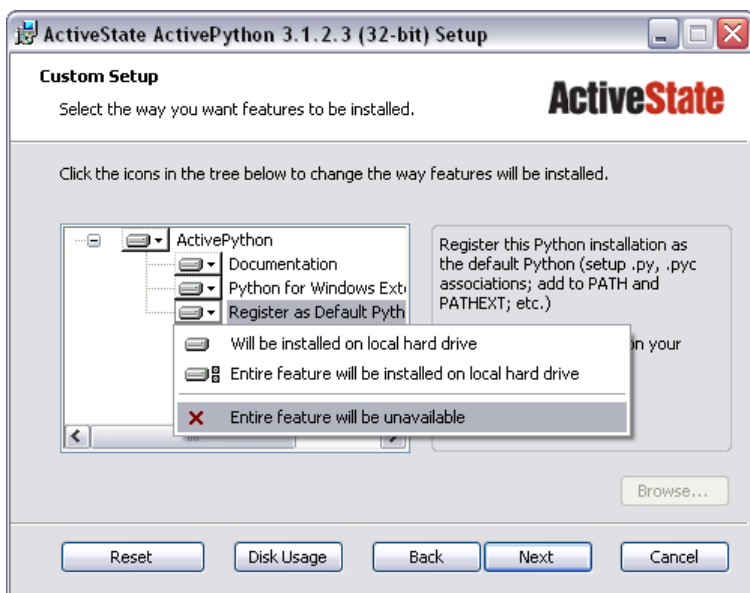


Рис. 1.7. Окно Custom Setup

В состав ActivePython кроме редактора PythonWin входит также редактор IDLE. Однако ни в одном меню нет пункта, с помощью которого можно запустить редактор IDLE. Чтобы это исправить, создадим два файла: IDLE25.bat и IDLE31.bat. Содержимое файла IDLE25.bat:

```
@echo off
start C:\Python25\pythonw.exe C:\Python25\Lib\idlelib\idle.pyw
```


Содержимое файла IDLE31.bat:

```
@echo off
start C:\Python31\pythonw.exe C:\Python31\Lib\idlelib\idle.pyw
```

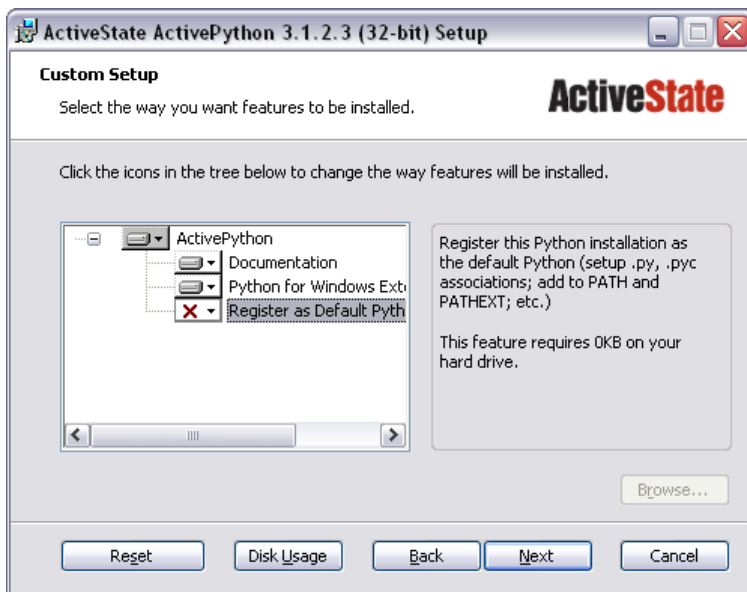


Рис. 1.8. Компонент **Register as Default Python** отключен

С помощью двойного щелчка на этих файлах можно запускать IDLE для версий Python 2.5 и Python 3.1. Чтобы запустить редактор в версии Python 2.6, в меню **Пуск | Программы | Python 2.6** выбираем пункт **IDLE (Python GUI)**.

Теперь рассмотрим запуск программы с помощью разных версий Python. По умолчанию при двойном щелчке на значке файла запускается Python 2.6. Чтобы запустить с помощью другой версии, щелкаем правой кнопкой мыши на значке файла с программой и в контекстном меню выбираем пункт **Открыть с помощью...** При первом запуске в списке будет только программа python.exe. Чтобы добавить другую версию, щелкаем на пункте **Выбрать программу...**, в открывшемся окне нажимаем кнопку **Обзор** и выбираем программу python25.exe из папки C:\Python25. Затем добавляем программу python31.exe из папки C:\Python31. В результате список в контекстном меню будет выглядеть так, как показано на рис. 1.9. Выбирая нужную версию из списка, можно производить запуск программы с помощью разных версий Python.

Для проверки установки создайте файл test.py с помощью любого текстового редактора, например Блокнота. Содержимое файла приведено в листинге 1.1.

Листинг 1.1. Проверка установки

```
import sys
print (tuple(sys.version_info))
```

```
try:
    raw_input()          # Python 2
except NameError:
    input()             # Python 3
```

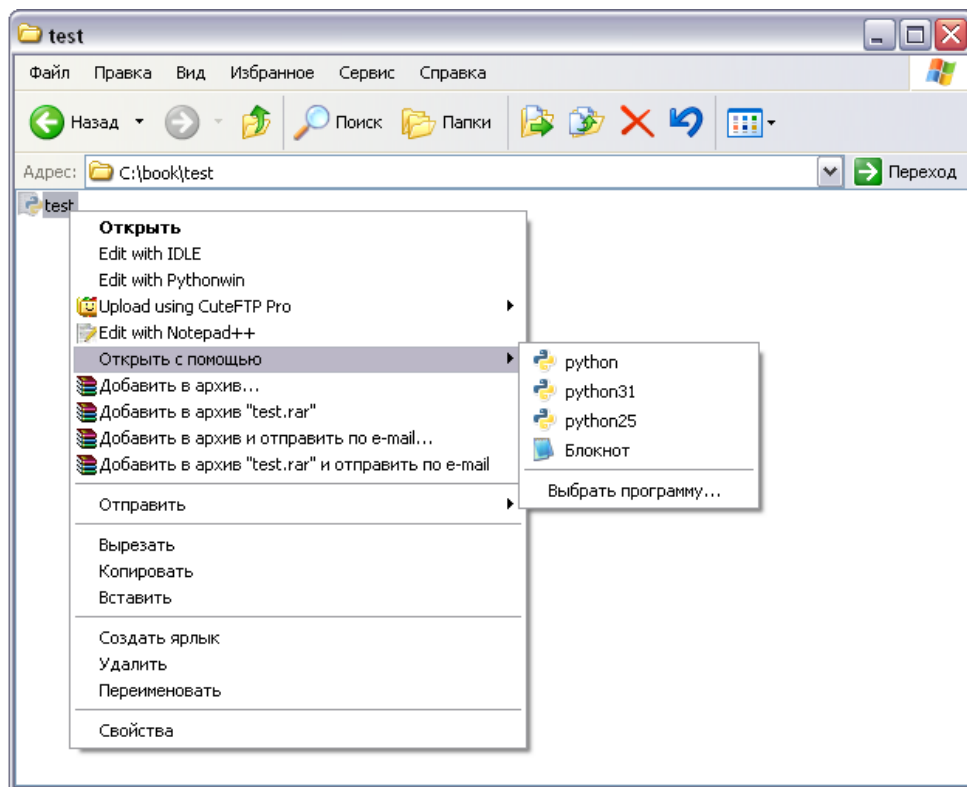


Рис. 1.9. Варианты запуска программы разными версиями Python

Затем запустите программу с помощью двойного щелчка на значке файла. Если результат выполнения — $(2, 6, 5, 'final', 0)$, то установка прошла нормально, а если $(2, 5, 5, 'final', 0)$ или $(3, 1, 2, 'final', 0)$, то вы не отключили компонент **Register as Default Python**. Для изучения материала этой книги по умолчанию должна быть версия Python 2.6.5.

1.2. Первая программа на Python

При изучении языков программирования принято начинать с программы, выводящей надпись "Привет, мир!". Не будем нарушать традицию и продемонстрируем, как это будет выглядеть на Python (листинг 1.2).

Листинг 1.2. Первая программа на Python

```
# Выводим надпись с помощью оператора print
print "Привет, мир!"
```

Для запуска программы в меню **Пуск** выбираем пункт **Программы | Python 2.6 | IDLE (Python GUI)**. В результате откроется окно **Python Shell**, в котором символы `>>>` означают приглашение ввести команду. После ввода команды нажимаем клавишу `<Enter>`. На следующей строке сразу отобразится результат, а далее приглашение для ввода новой команды. Последовательность выполнения нашей программы показана в листинге 1.3.

Листинг 1.3. Последовательность выполнения программы в окне Python Shell

```
>>> # Выводим надпись с помощью оператора print
>>> print "Привет, мир!"
Привет, мир!
>>>
```

ПРИМЕЧАНИЕ

Символы `>>>` вводить не нужно, они вставляются автоматически.

Для создания файла с программой в меню **File** выбираем пункт **New Window**. В открывшемся окне набираем код из листинга 1.2, а затем сохраняем его под именем `test.py`, выбрав пункт меню **File | Save As**. При этом редактор выведет запрос, в какой кодировке сохранить файл, и предложит вставить строку:

```
# -*- coding: cp1251 -*-
```

Соглашаемся и нажимаем кнопку **Edit my file**. В результате строка будет вставлена в самое начало программы, а файл будет сохранен в кодировке `Windows-1251`.

Запустить программу на выполнение можно, выбрав пункт меню **Run | Run Module** или нажав клавишу `<F5>`. Результат выполнения программы будет отображен в окне **Python Shell**. По умолчанию в этом окне используется кодировка `Windows-1251`, поэтому русские буквы отображаются корректно.

Запустить программу можно также с помощью двойного щелчка мыши на значке файла. В этом случае результат выполнения будет отображен в консоли `Windows`. В консоли используется кодировка `cp866`, по этой причине файл необходимо сохранить именно в этой кодировке, иначе русские буквы будут искажены. Кроме того, следует учитывать, что после вывода результата окно консоли сразу закрывается. Чтобы предотвратить закрытие окна, необходимо добавить вызов функции `raw_input()`, которая будет ожидать нажатия клавиши `<Enter>` и не позволит окну сразу закрыться. С учетом всего вышесказанного наша программа будет выглядеть так, как показано в листинге 1.4.

Листинг 1.4. Программа для запуска с помощью двойного щелчка мыши

```
# -*- coding: cp866 -*-
print "Привет, мир!"           # Выводим строку
raw_input()                   # Ожидаем нажатия клавиши <Enter>
```

ПРИМЕЧАНИЕ

Если до функции `raw_input()` возникнет ошибка, то сообщение о ней будет выведено в консоль, но сама консоль после этого сразу будет закрыта, и вы не сможете прочитать сообщение об ошибке.

Если необходимо, чтобы русские буквы правильно отображались в консоли и в окне **Python Shell**, то вместо обычных строк следует использовать Unicode-строки. В этом случае при выводе Unicode-строка будет автоматически преобразована в обычную строку в кодировке терминала. Для создания Unicode-строки можно воспользоваться модификатором `u`, который указывается перед открывающей кавычкой, или функцией `unicode()`. Пример использования Unicode-строк приведен в листинге 1.5.

Листинг 1.5. Пример использования Unicode-строк

```
# -*- coding: cp1251 -*-
print u"Привет, мир!"         # Модификатор u
print unicode("Привет, мир!", "cp1251") # Функция unicode()
raw_input()
```

Чтобы отредактировать уже созданный файл, щелкаем правой кнопкой мыши на значке файла и из контекстного меню выбираем пункт **Edit with IDLE**. Так как программа на языке Python представляет собой обычный текстовый файл, сохраненный с расширением `py`, его можно редактировать с помощью других программ, например, Notepad++. Можно также воспользоваться специализированными редакторами, скажем, PyScripter.

При открытии файла с помощью пункта **Edit with IDLE** по умолчанию открываются сразу два окна — окно с текстом программы и окно **Python Shell**. На мой взгляд, это не очень удобно. Чтобы открывалось только одно окно с текстом программы в меню **Options**, выбираем пункт **Configure IDLE**. В открывшемся окне на вкладке **General** устанавливаем флажок **Open Edit Window**.

1.3. Структура программы

Как вы уже знаете, программа на языке Python представляет собой обычный текстовый файл с выражениями. Каждое выражение располагается на отдельной строке. Если выражение не является вложенным, то оно должно начинаться с начала строки, иначе будет выведено сообщение об ошибке (листинг 1.6).

Листинг 1.6. Исключение `IndentationError`

```
>>> import sys

File "<pyshell#0>", line 1
    import sys
    ^
IndentationError: unexpected indent
>>>
```

В этом случае перед оператором `import` расположен один лишний пробел, который привел к выводу сообщения об ошибке.

Если программа предназначена для исполнения в операционной системе UNIX, то в первой строке необходимо дополнительно указать путь к интерпретатору Python:

```
#!/usr/bin/python
```

На некоторых серверах путь к интерпретатору выглядит по-другому:

```
#!/usr/local/bin/python
```

Иногда можно не указывать точный путь к интерпретатору, а передать название языка программе `env`:

```
#!/usr/bin/env python
```

В этом случае программа `env` произведет поиск интерпретатора Python в соответствии с настройками путей поиска.

Помимо указания пути к интерпретатору Python для CGI-программ необходимо, чтобы был установлен бит на выполнение в правах доступа к файлу. Кроме того, следует помнить, что перевод строки в операционной системе Windows состоит из последовательности двух символов — `\r` (перевод каретки) и `\n` (перевод строки). В операционной системе UNIX перевод строки осуществляется только одним символом `\n`. Если загрузить файл программы по протоколу FTP в бинарном режиме, то символ `\r` вызовет фатальную ошибку. По этой причине файлы по протоколу FTP следует загружать только в текстовом режиме (режим `ASCII`). В этом режиме символ `\r` будет удален автоматически. После загрузки файла следует установить права на выполнение. Для исполнения скриптов на Python устанавливаем права в 755 (`-rwxr-xr-x`). Изменить права доступа позволяет практически любой FTP-клиент.

Во второй строке (для ОС Windows в первой строке) следует указать кодировку. Для кодировки Windows-1251 строка будет выглядеть так:

```
# -*- coding: cp1251 -*-
```

Редактор IDLE учитывает указанную кодировку и автоматически производит перекодирование при сохранении файла. Получить полный список поддерживаемых кодировок и их псевдонимы позволяет код, приведенный в листинге 1.7.

Листинг 1.7. Вывод списка поддерживаемых кодировок

```
# -*- coding: cp1251 -*-
import encodings.aliases
arr = encodings.aliases.aliases
keys = arr.keys()
keys.sort()
for key in keys:
    print "%s => %s" % (key, arr[key])
```

Во многих языках программирования (например, в PHP, Perl и др.) каждое выражение должно завершаться точкой с запятой. В языке Python в конце выражения также можно поставить точку с запятой, но это не обязательно. В отличие от языка JavaScript, где рекомендуется завершать выражения точкой с запятой, в языке Python точку с запятой ставить *не рекомендуется*. Концом выражения является конец строки. Тем не менее, если необходимо разместить несколько выражений на одной строке, то точку с запятой *следует указать* (листинг 1.8).

Листинг 1.8. Несколько выражений на одной строке

```
# -*- coding: cp1251 -*-
x = 5; y = 10; z = x + y # Три выражения на одной строке
print z
```

Еще одной отличительной особенностью языка Python является отсутствие ограничительных символов для выделения выражений внутри блока. Например, в языке PHP выражения внутри цикла `while` выделяются фигурными скобками:

```
$i = 1;
while ($i<11) {
    echo $i . "\n";
    $i++;
}
echo "Конец программы";
```

В языке Python тот же код будет выглядеть по-другому (листинг 1.9).

Листинг 1.9. Выделение выражений внутри блока

```
# -*- coding: cp1251 -*-
i = 1
while i<11:
    print i
    i += 1
print "Конец программы"
```

Обратите внимание, что перед всеми выражениями внутри блока расположено одинаковое количество пробелов. Таким образом в языке Python выделяются блоки. Выражения, перед которыми расположено одинаковое количество пробелов, являются телом блока. В нашем примере два выражения выполняются десять раз. Концом блока является выражение, перед которым расположено меньшее количество пробелов. В нашем случае это оператор `print`, который выводит строку "Конец программы". Если количество пробелов внутри блока будет разным, то интерпретатор выведет сообщение о фатальной ошибке и программа будет остановлена. Таким образом, язык Python приучает программистов писать красивый и понятный код.

ПРИМЕЧАНИЕ

В языке Python принято использовать четыре пробела для выделения выражений внутри блока.

Если блок состоит из одного выражения, то допустимо разместить его на одной строке с основной инструкцией. Например, код

```
for i in range(1, 11):
    print i
print "Конец программы"
```

можно записать так:

```
for i in range(1, 11): print i
print "Конец программы"
```

Если выражение является слишком длинным, то его можно перенести на следующую строку следующими способами:

- в конце строки разместить символ `\`. После этого символа должен следовать символ перевода строки. Другие символы (в том числе и комментарии) недопустимы. Пример:

```
x = 15 + 20 \
    + 30
print x
```

- поместить выражение внутри круглых скобок. Это лучший способ, т. к. любое выражение можно разместить внутри круглых скобок. Пример:

```
x = (15 + 20 # Это комментарий
    + 30)
print x
```

- определение списка и словаря можно разместить на нескольких строках, т. к. используются квадратные и фигурные скобки соответственно. Пример определения списка:

```
arr = [15, 20, # Это комментарий
      30]
print arr
```

Пример определения словаря:

```
arr = {"x": 15, "y": 20, # Это комментарий
      "z": 30}
print arr
```

1.4. Комментарии

Комментарии предназначены для вставки пояснений в текст скрипта, и интерпретатор полностью их игнорирует. Внутри комментария может располагаться любой текст, включая выражения, которые выполнять не следует. Помните, комментарии нужны программисту, а не интерпретатору Python. Вставка комментариев в код позволит через некоторое время быстро вспомнить предназначение фрагмента кода.

В языке Python присутствует только *однострочный комментарий*. Он начинается с символа #:

```
# Это комментарий
```

Однострочный комментарий может начинаться не только с начала строки, но и располагаться после выражения. Например, после инструкции вывести надпись "Привет, мир!":

```
print "Привет, мир!" # Выводим надпись с помощью оператора print
```

Если символ комментария разместить перед выражением, то оно не будет выполнено:

```
# print "Привет, мир!" Это выражение выполнено не будет
```

Если символ # расположен внутри кавычек или апострофов, то он не является символом комментария:

```
print "# Это НЕ комментарий"
```

Так как в языке Python нет многострочного комментария, то часто комментируемый фрагмент размещают внутри утроенных кавычек (или утроенных апострофов):

```
"""
Это выражение не будет выполнено
print "Привет, мир!"
"""
```

Следует заметить, что этот фрагмент кода не игнорируется интерпретатором, т. к. он не является комментарием. В результате выполнения фрагмента будет создан объект строкового типа. Тем не менее, выражения внутри утроенных кавычек выполнены не будут, т. к. все выражения будут считаться простым текстом. Такие строки являются строками документирования, а не комментариями.

1.5. Скрытые возможности IDLE

На всем протяжении этой книги в качестве редактора мы будем использовать IDLE. По этой причине рассмотрим некоторые возможности этой среды разработки.

Как вы уже знаете, в окне **Python Shell** символы `>>>` означают приглашение ввести команду. После ввода команды нажимаем клавишу `<Enter>`. На следующей строке сразу отобразится результат, а далее — приглашение для ввода новой команды. При вводе многострочной команды после нажатия клавиши `<Enter>` редактор автоматически вставит отступ и будет ожидать дальнейшего ввода. Чтобы сообщить редактору о конце ввода команды необходимо дважды нажать клавишу `<Enter>`. Пример:

```
>>> for n in range(1, 3):
    print n

1
2
>>>
```

В предыдущем разделе мы выводили строку "Привет, мир!" с помощью оператора `print`. В окне **Python Shell** это делать не обязательно. Например, мы можем просто ввести строку и нажать клавишу `<Enter>` для получения результата:

```
>>> "Hello, world!"
'Hello, world!'
>>>
```

Однако русские буквы таким образом отображаться не будут. Пример:

```
>>> "Привет, мир!"
'\xcf\x0\xe8\xe2\xe5\xf2, \xec\xe8\xf0!'
>>>
```

Обратите также внимание, что строки выводятся в апострофах. Этого не произойдет, если выводить строку с помощью оператора `print`:

```
>>> print "Привет, мир!"
Привет, мир!
>>>
```

Учитывая возможность получить результат сразу после ввода команды, окно **Python Shell** можно использовать для изучения команд, а также в качестве многофункционального калькулятора. Пример:

```
>>> 12 * 32 + 54
438
>>>
```

Результат вычисления последнего выражения сохраняется в переменной `_` (одно подчеркивание). Это позволяет производить дальнейшие расчеты без ввода предыдущего результата. Вместо него достаточно ввести символ подчеркивания.

Пример:

```
>>> 125 * 3          # Умножение
375
>>> _ + 50          # Сложение. Эквивалентно 375 + 50
425
>>> _ / 5           # Деление. Эквивалентно 425 / 5
85
>>>
```

При вводе команды можно воспользоваться комбинацией клавиш `<Ctrl>+<Пробел>`. В результате будет отображен список, из которого можно выбрать нужный идентификатор. Если при открытом списке вводить буквы, то показываться будут идентификаторы, начинающиеся с этих букв. Выбирать идентификатор необходимо с помощью клавиш `<↑>` и `<↓>`. После выбора не следует нажимать клавишу `<Enter>`, иначе это приведет к выполнению выражения. Просто вводите выражение дальше, а список закроется. Такой же список будет автоматически появляться (с некоторой задержкой) при обращении к свойствам объекта или атрибутам модулей после ввода точки. Для автоматического завершения идентификатора после ввода первых букв можно воспользоваться комбинацией клавиш `<Alt>+</>`. При каждом последующем нажатии этой комбинации будет вставляться следующий идентификатор. Эти две комбинации клавиш очень удобны, если вы забыли, как пишется слово, или хотите, чтобы редактор закончил его за вас.

При необходимости повторно выполнить ранее введенное выражение его придется набирать заново. Можно, конечно, скопировать выражение, а затем вставить, но как вы можете сами убедиться, в контекстном меню нет пунктов **Copy** (Копировать) и **Paste** (Вставить). Они расположены в меню **Edit**. Постоянно выбирать пункты из этого меню очень неудобно. Одним из решений проблемы является использование комбинации клавиш быстрого доступа — `<Ctrl>+<C>` (копировать) и `<Ctrl>+<V>` (вставить). Комбинации стандартны для Windows, и вы наверняка их уже использовали ранее. Но опять-таки, прежде чем скопировать выражение, его предварительно необходимо выделить. Редактор IDLE избавляет нас от лишних действий и предоставляет комбинацию клавиш `<Alt>+<N>` для вставки первого введенного выражения, а также комбинацию `<Alt>+<P>` для вставки последнего выражения. Каждое последующее нажатие этих клавиш будет вставлять следующее (или предыдущее) выражение. Для еще более быстрого повторного ввода выражения следует предварительно ввести его первые буквы. В этом случае перебираться будут только выражения, начинающиеся в этих буквах.

1.6. Вывод результатов работы программы

Вывести результаты работы программы можно с помощью оператора `print`. Мы уже использовали его ранее для вывода строки "Привет, мир!":

```
print "Привет, мир!"
```

Оператор `print` преобразует любой объект в строку и посылает ее в стандартный вывод `stdout`. После строки оператор автоматически добавляет символ перевода строки:

```
print "Строка 1"  
print "Строка 2"
```

Результат:

```
Строка 1  
Строка 2
```

Если необходимо вывести результат на той же строке, то в операторе `print` данные указываются через запятую:

```
print "Строка 1", "Строка 2"
```

Результат:

```
Строка 1 Строка 2
```

Как видно из примера, между выводимыми строками автоматически вставляется пробел. Чтобы этого избежать следует использовать конкатенацию строк или лучше всего форматирование:

```
print "Строка 1" + "Строка 2"           # Конкатенация строк  
# Выведет: Строка 1Строка 2  
print '%s%s' % ("Строка 1", "Строка 2") # Форматирование  
# Выведет: Строка 1Строка 2
```

Чтобы избежать добавления символа перевода строки следует последним символом указать запятую. Пример:

```
print "Строка 1", "Строка 2",  
print "Строка 3"  
# Выведет: Строка 1 Строка 2 Строка 3
```

Если, наоборот, необходимо вставить символ перевода строки, то оператор `print` указывается без параметров. Пример:

```
for n in range(1, 5):  
    print n,  
print  
print "Это текст на новой строке"
```

Результат выполнения:

```
1 2 3 4  
Это текст на новой строке
```

В этом примере мы использовали цикл `for`, который позволяет последовательно перебирать элементы. На каждой итерации цикла переменной `n` присваивается новое число, которое мы выводим с помощью оператора `print`, расположенного на следующей строке. Обратите внимание, что перед оператором мы добавили четыре пробела. Таким образом в языке Python выделяются *блоки*. Выражения, перед которыми расположено одинаковое количество пробелов, являются *телом цикла*. Все эти выражения выполняются определенное количество раз. Концом блока является выражение, перед которым расположено меньшее количество пробелов. В нашем

случае это оператор `print` без параметров, который вставляет символ переноса строки.

Если необходимо вывести большой блок текста, то его следует разместить между утроенными кавычками или утроенными апострофами. В этом случае текст сохраняет свое форматирование. Пример:

```
print """Строка 1
Строка 2
Строка 3"""
```

В результате выполнения этого примера мы получим три строки:

```
Строка 1
Строка 2
Строка 3
```

В окне **Python Shell** редактора IDLE необязательно использовать оператор `print`. Однако следует учитывать, что результат вывода будет отличаться:

```
>>> print "Строка"
Строка
>>> "Строка"
'\xd1\xf2\xf0\xee\xea\xe0'
```

Как видно из примера, все русские буквы были заменены соответствующими кодами, а сама строка расположена внутри апострофов. Это происходит потому, что результат автоматически обрабатывается с помощью функции `repr()`. Функция `repr()` возвращает строковое представление объекта. Выведем строку, используя оператор `print` и функцию `repr()`:

```
>>> print repr("Строка")
'\xd1\xf2\xf0\xee\xea\xe0'
```

Для вывода результатов работы программы вместо оператора `print` можно использовать метод `write()` объекта `sys.stdout`:

```
import sys # Подключаем модуль sys
sys.stdout.write("Строка") # Выводим строку
```

В первой строке, с помощью инструкции `import`, мы подключаем модуль `sys`, в котором объявлен объект. Далее с помощью метода `write()` выводим строку. Следует заметить, что метод не вставляет символ перевода строки. Поэтому при необходимости следует добавить его самим с помощью символа `\n`:

```
import sys
sys.stdout.write("Строка 1\n")
sys.stdout.write("Строка 2")
```

1.7. Ввод данных

Для ввода данных предназначены две функции:

- `raw_input([<Сообщение>])` — получает данные со стандартного ввода `stdin`.

Для примера переделаем нашу первую программу так, чтобы она здоровалась не со всем миром, а только с нами:

```
# -*- coding: cp866 -*-
n = raw_input("Введите ваше имя: ")
print "Привет,", n
raw_input("Нажмите <Enter> для закрытия окна")
```

Вводим код и сохраняем файл, например, под названием `test.py`, а затем запускаем программу на выполнение с помощью двойного щелчка на значке файла. Откроется черное окно, в котором будет надпись "Введите ваше имя: ". Вводим свое имя, например "Николай", и нажимаем клавишу <Enter>. В результате будет выведено приветствие "Привет, Николай". Чтобы окно сразу не закрылось, повторно вызываем функцию `raw_input()`. В этом случае окно не закроется, пока не будет нажата клавиша <Enter>:

□ `input([<Сообщение>])` — получает данные со стандартного ввода `stdin` и обрабатывает их с помощью функции `eval()`. Вызов функции `input()` эквивалентен следующему коду:

```
eval(raw_input([<Сообщение>]))
```

ВНИМАНИЕ!

Функция `eval()` выполнит любую введенную инструкцию. Никогда не используйте функцию `input()`, если не доверяете пользователю.

Передать данные можно в командной строке после названия файла. Такие данные доступны через список `argv` модуля `sys`. Первый элемент списка `argv` будет содержать название файла, а последующие элементы — переданные данные. В качестве примера создадим файл `test.py` в папке `C:\book`. Содержимое файла приведено в листинге 1.10.

Листинг 1.10. Получение данных из командной строки

```
# -*- coding: cp866 -*-
import sys
arr = sys.argv[:]
for n in arr:
    print n
```

Теперь запустим программу на выполнение с помощью командной строки и передадим данные. Запускаем командную строку. Для этого в меню **Пуск** выбираем пункт **Выполнить**. В открывшемся окне набираем команду `cmd` и нажимаем кнопку **ОК**. Откроется черное окно, в котором будет приглашение для ввода команд. Переходим в папку `C:\book`. Для этого набираем команду:

```
cd C:\book
```

В командной строке должно быть приглашение:

```
C:\book>
```

Для запуска нашей программы вводим команду:

```
C:\Python26\python.exe test.py -uNik -p123
```

В этой команде мы передаем название файла (test.py) и некоторые данные (-uNik -p123). Результат выполнения программы будет выглядеть так:

```
test.py
-uNik
-p123
```

1.8. Доступ к документации

Вместе с установкой интерпретатора Python на компьютер автоматически устанавливается документация в формате CHM. Чтобы отобразить документацию, в меню **Пуск** выбираем пункт **Программы | Python 2.6 | Python Manuals**.

Если в меню **Пуск** выбрать пункт **Программы | Python 2.6 | Module Docs**, то откроется окно **pydoc** (рис. 1.10). С помощью этого инструмента можно получить дополнительную информацию, которая расположена внутри модулей. Чтобы получить список всех модулей, установленных на компьютере, оставляем текстовое поле пустым и нажимаем кнопку **open browser**. В результате в окне Web-браузера, используемого в системе по умолчанию, отобразится список всех модулей. Каждое название модуля является ссылкой, при переходе по которой доступна документация по конкретному модулю. Если в окне **pydoc** в текстовом поле ввести какое-либо название и нажать клавишу <Enter>, то будет отображен список совпадений. Выделяем пункт в списке и нажимаем кнопку **go to selected**. Результат будет отображен в окне Web-браузера.

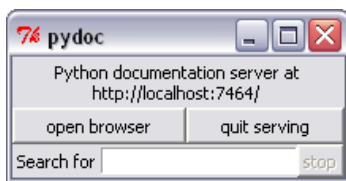


Рис. 1.10. Окно pydoc

В окне **Python Shell** редактора IDLE также можно отобразить документацию. Для этого предназначена функция `help()`. В качестве примера отобразим документацию по встроенной функции `raw_input()`:

```
>>> help(raw_input)
```

Результат выполнения:

```
Help on built-in function raw_input in module __builtin__:
```

```
raw_input(...)
    raw_input([prompt]) -> string
```