Programming Python

Second Edition

Mark Lutz

O'REILLY®

Программирование на Python

Второе издание

Марк Лутц

Санкт-Петербург
2002

Марк Лутц

Программирование на Python, 2 издание

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*М. Деркачев*
Редактор	*А. Лосев*
Корректура	*С. Беляева*
Верстка	*Н. Гриценко*

Лутц М.
Программирование на Python. – Пер. с англ. – СПб: Символ-Плюс, 2002. – 1136 с., ил.
ISBN 5-93286-036-7

Python – это широко распространенный язык программирования, применяемый при решении многих важных задач, диапазон которых простирается от коммерческих сценариев установки Linux и программирования веб-приложений до анимации фильмов и создания спецэффектов. Он доступен на всех ведущих вычислительных платформах, в том числе на основных коммерческих версиях Unix, Linux, Windows и Mac OS. Кроме того, он является языком с открытым исходным кодом.

Второе издание самого известного бестселлера по Python, прорецензированное и одобренное Гвидо ван Россумом, создателем Python, представляет собой наиболее полный на сегодняшний день источник для серьезно программирующих на Python. Основное внимание здесь сосредоточено на практическом применении языка. Читатель обнаружит, что одна книга фактически содержит в себе четыре, которые глубоко освещают создание сценариев для Интернета, системное программирование, программирование GUI с использованием Tkinter и интеграцию с C. Кроме того, обсуждаются новые инструменты и приложения: Jython – версия Python, компилируемая в виде байт-кодов Java; расширения Active Scripting и COM; Zope – система веб-приложений с открытым исходным кодом; генераторы кода HTMLgen и SWIG; поддержка потоков; модули CGI и протоколов Интернета. В книге приводится большое количество примеров кода, которые вы сможете использовать при разработке на Python сложных приложений. Прилагается CD для платформ PC, Macintoch и Unix.

ISBN 5-93286-036-7
ISBN 0-596-00085-5 (англ)

Издательство «Символ-Плюс». 193148, Санкт-Петербург, ул. Пинегина, 4,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 18.09.2002. Формат 70x100 1/16 . Печать офсетная.
Объем 71 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Вступительное слово

Менее пяти лет назад я написал вступительное слово к первому изданию «Программирования на Python». За истекшее время книга изменилась примерно в той же степени, что и сам язык и сообщество Python! Мне больше не нужно защищать Python: статистика и события, приведенные Марком в предисловии, говорят сами за себя.

За последний год Python быстро развивался. Мы выпустили Python 2.0 – большой шаг вперед – с новыми возможностями, включенными в стандартную библиотеку, такими как поддержка Unicode и XML, и несколькими новыми синтаксическими конструкциями, в том числе улучшенным присвоением: теперь вместо `x = x+1` можно писать `x += 1`. Кое-кто считал, что это мелочь (представьте себе, однако, не `x`, а `dict[key]` или `list[index]`), но в целом это имело успех у тех пользователей, которые привыкли к использованию улучшенного присвоения в других языках.

Менее теплый прием получило расширение команды print, `print>>file` – сокращенная форма для вывода в файл, отличный от стандартного вывода. Лично я использую эту возможность Python 2.0 очень часто, но большинство высказавшихся по этому поводу считает ее отвратительной. Обсуждение данного простого расширения языка в телеконференции было одним из самых длинных в истории, исключая вечную тему «Python против Perl».

И это подсказывает следующую тему. (Нет, не противостояние Python и Perl. Предисловие – не место для стычек.) Я имею в виду скорость, с которой происходит развитие Python, – эту тему автор книги принимает близко к сердцу. Всякий раз, когда я ввожу в Python новую функцию, у Марка появляется еще одна седая прядь в волосах – еще одна глава устарела! Особое его беспокойство вызвало появление в Python 2.0 множества новых функций, добавленных как раз в то время, когда он работал над данным, вторым изданием книги. Что если в Python 2.1 появится столько же нововведений? Книга устареет в тот день, когда будет опубликована!

Марк, расслабься. Python продолжит свое развитие, но я обещаю, что те вещи, которые активно используются, сохранятся! Например, много беспокойства вызвал модуль для работы со строками. Сейчас, с появлением методов у объектов строк, этот модуль стал в значительной мере избыточным. Хотелось бы мне объявить его устаревшим и тем самым побудить программистов Python начать использовать вместо него методы строк. Но с учетом того, что большая часть существующего кода Python и даже многие стандартные модули используют модуль строк, такая перемена не может, очевидно, произойти очень быстро. Скорее всего, модуль строк мы сможем убрать не раньше выхода в свет Python 3000, и даже тогда, вероятно, этот модуль сохранится в библиотеке для поддержки обратной совместимости, чтобы иметь возможность использовать старый код.

Python 3000?! Да, это условное название интерпретатора Python следующего поколения. Название можно считать игрой слов с Windows 2000 или ассоциацией с «Mystery Science Theater 3000» – ТВ-шоу вполне в духе Python, ставшее культовым. Когда выйдет Python 3000? О-о-о-чень нескоро, хотя не обязательно придется ждать 3000-го года.

Первоначально Python 3000 должен был быть спроектирован и переписан заново. Это позволило бы мне осуществить несовместимые изменения, чтобы исправить те проблемы архитектуры языка, которые нельзя решить с сохранением обратной совместимости. В настоящее время, однако, планируется постепенное введение необходимых изменений в рамках продолжения текущей линии Python 2.x с ясным переходным периодом, в течение которого поддерживается обратная совместимость.

Возьмем, к примеру, деление целых чисел. Так же как и в C, в настоящее время в Python деление x/y для двух целочисленных аргументов имеет целочисленный результат. Иными словами, 1/2 дает 0! Для закоренелых программистов это естественно, но для новичков, составляющих все большую часть пользователей Python (количество которых растет экспоненциально), это служит беспрестанным источником недоразумений. С точки зрения вычислений более разумно, чтобы оператор / давал одно и то же значение независимо от типа его операндов: в конце концов, так поступают все числовые операторы. Но нельзя просто взять и изменить Python, чтобы 1/2 давало 0.5, потому что (как и в случае удаления модуля string) слишком большая часть существующего кода стала бы неработоспособной. Где же выход?

Решение – слишком сложное, чтобы описывать его здесь подробно, – охватит несколько последующих версий Python и состоит в постепенном усилении давления на программистов Python (сначала в документации, затем через предупреждения об использовании устаревших конструкций и, наконец, через сообщения об ошибках) с целью заставить их переработать свой код. Кстати, структура вывода предупредительных сообщений будет включена в версию Python 2.1. Прости, Марк!

Поэтому не ожидайте в скором времени объявления о выходе Python 3000. Вместо этого вы можете в один прекрасный день обнаружить, что *уже* используете Python 3000, только название его будет не таким, а скажем, чем-нибудь вроде Python 2.8.7. И большая часть того, что вы узнаете из этой книги, будет по-прежнему действовать! Тем не менее, в ссылках на Python 3000 не будет недостатка: просто помните, что это намеренно превозносимая химера в буквальном смысле слова. Вместо беспокойства по поводу Python 3000 лучше продолжайте использовать и изучать ту версию Python, которая у вас есть.

Хотелось бы сказать несколько слов о существующей в данное время модели разработки Python. До начала 2000 года разработчиков Python были сотни, но, в сущности, все нововведения проходили через мой почтовый ящик. Предлагая внести в Python изменение, вы должны были прислать мне по почте файл различий, который я применял к своей рабочей версии Python, и если изменение меня удовлетворяло, я вносил его в свое дерево исходного кода CVS. (CVS – это система управления версиями кода, которой посвящено несколько книг.) Сообщения о выявленных ошибках следовали тем же путем, за исключением того, что в итоге я выпускал заплатку. Очевидно, что с ростом числа предлагаемых изменений мой почтовый ящик стал узким местом процесса. Что было делать?

К счастью, Python – не единственный проект open source, в котором возникла эта проблема, и несколько умных людей в VA Linux предложили свое решение: SourceForge! Это динамичный веб-сайт с полным набором средств управления распределенными проектами: открытое хранилище CVS, почтовые списки рассылки (использующие Mailman, очень популярное приложение Python!), дискуссионные форумы, средства администрирования для ошибок и патчей и область для загрузки с сервера – все доступно для любого проекта open source, которому это может понадобиться.

В данное время у нас есть группа из 30 добровольцев с правами записи в SourceForge и почтовый список рассылки для разработчиков, в котором вдвое больше народа. Все привилегированные добровольцы поклялись на верность BDFL (Benevolent Dictator For Life – Великодушному Пожизненному Диктатору, то есть мне :-). Введение существенных новых функций регулируется с помощью облегченной системы предложений и обратной связи под названием Python Enhancement Proposals (PEPs). Наша система PEP оказалась столь успешной, что почти буквально была скопирована сообществом Tcl, когда оно совершило аналогичный переход с Cathedral на Bazaar.

Итак, с уверенностью в будущем Python я передаю слово Марку Лутцу. Отличная работа, Марк! Завершаю моей любимой цитатой из Monty Python: «Take it away, Eric, the orchestra leader»!

Гвидо ван Россум

Рестон, Вирджиния, январь 2001

Предисловие ко второму изданию

«А теперь нечто совершенно новое... опять»

Предыдущее издание этой книги стало одним из первых, представивших язык программирования Python. Второе издание является фактически новой книгой, посвященной более сложным темам Python, и должно послужить продолжением изложения основ языка в «Learning Python», дополненного справочными материалами из «Python Pocket Reference».

Это означает, что настоящее издание посвящено скорее возможностям *использования* Python, чем самому языку. Попутно исследуются идеи, лежащие в основе разработки программного обеспечения на Python; в действительности они обретают смысл только в контексте более крупных примеров, которые включены в данное издание. Но в целом, предполагается наличие у читателя хотя бы беглого знакомства с основами языка Python, которые помогут перейти к оставшейся части рассказа о нем.

В данном предисловии будут объяснены некоторые причины такой существенной переработки текста, более подробно описана структура издания и дан краткий обзор возможностей использования программ Python, имеющихся на прилагаемом компакт-диске. Однако вначале нам предстоит осуществить исторический экскурс.

Вехи распространения Python

Последние пять лет стали примечательными в мире Python. С тех пор как я написал первое издание данной книги в период между 1995 и 1996 годами, Python из новичка в семействе языков сценариев превратился в солидный и широко распространенный инструмент, используемый компаниями, разбросанными по всему свету. Хотя не всегда легко оценить популярность свободно распространяемого программного обеспечения с открытым кодом (*http://opensource.org*), такого как Python, имеющаяся статистика показывает экспоненциальный рост его популярности за последние пять лет. Вот некоторые из самых свежих признаков взрывного роста интереса к нему:

Книги

В 2001 году, когда пишется эта книга, в продаже имеется свыше десятка книг, посвященных Python, и еще почти столько же готовится к изданию (в 1995 книг не было). Некоторые из этих книг посвящены определенным областям (например, Windows), а некоторые изданы на немецком, французском и японском языках.

Пользователи

В 1999 году один из ведущих обозревателей индустрии программного обеспечения предположил исходя из различных статистических данных, что в мире насчитывается 300 000 пользователей Python. Другие оценки являются еще более оптимистичными. В начале 2000 года, например, уже действовал веб-сайт Python, с которого к концу года было произведено 500 000 новых загрузок интерпретатора (помимо того, что есть и другие носители дистрибутивов Python). Вероятно, на момент написания данной книги последняя цифра ближе к подлинной численности контингента пользователей.

Периодическая печать

Сейчас Python регулярно служит темой публикаций в периодических изданиях по программированию. В действительности после 1995 года создатель Python Гвидо ван Россум (Guido van Rossum) появился на обложке известных специальных журналов, например *Linux Journal* и *Dr. Dobb's Journal*; в публикации последнего создание Python стало основанием для награды за выдающиеся успехи в программировании.[1]

Приложения

Настоящие компании применяют Python для создания настоящих продуктов. Он был использован в построении спецэффектов для последнего фильма «Звездные войны» (Industrial Light & Magic), при выводе карт и каталогов в Интернете (Yahoo), в инсталляционной программе операционной системы Linux (Red Hat), при проверке микросхем и плат (Intel), в управлении дискуссионными форумами Интернета (Egroups), в сценариях сетевых игр (Origin), в интерфейсе к серверу CORBA (TCSI), в реализации инструментария веб-сайтов (Digital Creations' Zope), в сценариях для беспроводных устройств (Agilent) и во многих других местах.[2]

Телеконференции

Объем переписки в главной телеконференции по Python, *comp.lang.python*, также значительно вырос. Например, согласно eGroups (см. *http://www.egroups.com/group/python-list*), в январе 1994 года в этот список было подано 76 сообщений против 2678 в январе 2000 года – 35-кратное увеличение. В последующее время активность еще более возросла (4226 сообщений только за июнь 2000 – примерно 140 в сутки), и наблюдается постоянный рост с момента начала работы списка. К тому времени, когда вы будете читать это, цифры, относящиеся к числу пользователей и приведенные в данном предисловии, вероятно, возрастут. Но даже при настоящей интенсивности обмена форумы по Python достаточно загружены, чтобы полностью занять время того, кто готов полностью посвятить себя им.

Конференции

Сейчас ежегодно проводятся две конференции по Python, одну из которых организует O'Reilly & Associates. Число участников конференций, посвященных Python, примерно удваивалось каждый год. Кроме того, в Европе теперь проводится ежегодный «День Python».

Групповая терапия

Региональные группы пользователей Python стали возникать во многих местах в США и за границей, в том числе в Орегоне, Сан-Франциско, Вашингтоне, Италии, Корее и Англии. Такие группы работают над усовершенствованиями языка, проводят общественные мероприятия по Python и делают многое другое.

---

[1] Когда я писал эту книгу, *Linux Journal* напечатал также специальное приложение, посвященное Python, к майскому изданию 2000 года, на обложке которого, конечно, красовался голый мужчина, сидящий на улице перед компьютером вместо пианино. Если вам не понятно, почему это должно быть смешно, нужно посмотреть повторные показы телесериала «Monty Python», от которого получил свое название Python (считайте это первым предложенным упражнением). Я подробнее расскажу о последствиях, к которым привело название Python, в первой главе.

[2] Подробнее см. на *http://www.python.org*. Некоторые компании скрывают использование ими Python по соображениям конкуренции, хотя для многих это в итоге выявляется, когда происходит отказ на какой-либо из веб-страниц и в броузере появляется сообщение Python об ошибке. Обычно среди компаний, «засветившихся» подобным образом, упоминается Hewlett Packard.

Области применения

Развитие Python объединило как разработчиков для Microsoft Windows, с включением поддержки COM и Active Scripting, так и разработчиков Java благодаря новой, основанной на Java реализации языка – JPython (переименованного в «Jython»). Как будет показано в данной книге, появившаяся в языке поддержка COM позволяет сценариям Python являться как серверами компонентов, так и клиентами этих серверов; Active Scripting позволяет встраивать код Python в страницы HTML и выполнять его у клиента или на сервере, а JPython компилирует сценарии Python в код виртуальной машины Java, благодаря чему они могут выполняться в системах, поддерживающих Java, и беспрепятственно интегрировать библиотеки классов Java для использования в коде Python. В качестве инструментального средства, упрощающего создание сайтов, внимание вебмастеров и программистов CGI также привлек основанный на Python сервер веб-приложений Zope, описываемый в данной книге.

Поддержка

Что касается практической стороны, то коммерческая поддержка, консультации, готовые дистрибутивы и профессиональное обучение в настоящее время предоставляются многими фирмами. Например, интерпретатор Python можно получить на CD и в пакетах, продаваемых различными компаниями (в том числе Walnut Creek, «Dr. Dobb's Journal» и ActiveState); кроме того, Python обычно бесплатно поставляется в собранном виде во многих дистрибутивах операционной системы Linux.

Работа

Сейчас можно зарабатывать деньги в качестве Python-программиста (даже не прибегая к необходимости писать большие книги, содержащие интересные идеи). Когда писалась эта книга, на доске объявлений о работе на *http://www.python.org/Jobs.html* было перечислено около 60 компаний, которым требуются программисты Python в США или за рубежом. Поиск «Python» на популярных сайтах по трудоустройству дает еще более внушительные результаты: 285 связанных с Python рабочих мест на Monster.com и 369 на dice.com. Конечно, не обязательно менять работу, но приятно знать, что теперь можно зарабатывать на жизнь благодаря знанию языка, использование которого к тому же доставляет удовольствие.

Инструменты разработки

Python лег в основу многих проектов разработки инструментальных средств. Самыми заметными из них на момент написания книги являются проект Software Carpentry, в котором разрабатываются новые базовые программные инструментальные средства для Python; ActiveState, которая близка к выпуску продуктов разработки на Python для Windows и Linux[1]; и PythonWare, собирающаяся выпустить для Python интегрированную среду разработки и средство создания графического интерфейса пользователя.

Компиляторы

Когда писалось это предисловие, ActiveState объявила о создании нового компилятора Python для структуры Microsoft.NET и языковой среды C# – действительного компилятора Python и независимой реализации языка Python, в которой генерируются файлы DLL и EXE, можно разрабатывать код Python в Visual Studio и обеспечивается гладкая интеграция .NET со сценариями Python. Это будет третья реализация Python, наряду со стандартным Python, основанным на C, и системой Jpython, основанной на Java.

---

[1] ActiveState выпустила IDE Komodo, поддерживающий Python, для платформ Windows и Linux. – *Примеч. науч. ред.*

Так что же такое Python?

Если вам нужно исчерпывающее определение темы данной книги, воспользуйтесь таким:

Python является языком программирования общего назначения с открытым исходным кодом (open source), оптимизированным для качества, производительности, переносимости и интеграции. Им пользуются сотни тысяч разработчиков по всему миру в таких областях, как создание интернет-сценариев, системное программирование, проектирование пользовательских интерфейсов, настройка программных продуктов под пользователя и др.

Помимо всего остального Python поддерживает объектно-ориентированное программирование (ООП); обладает синтаксисом, который очень прост, легко читается и сопровождается; интегрируется с компонентами, написанными на языке C; обладает большим собранием уже запрограммированных интерфейсов и утилит. Несмотря на свое общее назначение, Python часто называют языком сценариев (*scripting language*), поскольку в нем просто использовать другие программные компоненты и управлять ими. Возможно, самым большим достоинством Python является просто то, что с его помощью разработка программного обеспечения становится более быстрой и приятной. Как это происходит, станет ясно из дальнейшего изложения.

Обучение

Python начал также привлекать внимание образовательных учреждений, многие из которых рассматривают его как «Pascal 2000-х годов» – язык, идеальный для обучения программированию благодаря его простоте и структуре. Отчасти это направление порождено проектом Гвидо ван Россума «Computer Programming for Everybody» (CP4E, компьютерное программирование для всех), который имеет целью сделать Python предпочтительным языком для начинающих программистов во всем мире. Будущее проекта CP4E пока остается неясным, но сформировалась группа особого интереса (SIG) к Python, которая должна заняться вопросами, связанными с образованием. Независимо от исхода той или иной инициативы, Python может сделать программирование более доступным для тех многочисленных людей, которым скоро наскучит щелкать по заранее запрограммированным ссылкам по мере их превращения из пользователей компьютеров в создателей сценариев.

Иными словами, 1995 год давно прошел. Большая часть приведенного перечня была невыполнима, когда было задумано первое издание этой книги. Естественно, этот список окажется устаревшим даже раньше, чем эта книга попадет на полки, но тем не менее, он показывает те вехи, которыми отмечено развитие Python за последние пять лет и которые ожидают нас в ближайшие годы. В качестве языка, оптимально подходящего к сегодняшним требованиям при создании программного обеспечения, Python, несомненно, еще ждет вершины своего успеха.

Для чего потребовалось это издание?

Одним из следствий растущей популярности Python стал наплыв новых пользователей, стилей программирования и приложений, из-за чего некоторые части первого издания данной книги потребовали обновления. Сам Python изменился незначительно, но важные расширения упростили различные стороны разработки на Python и заслуживают того, чтобы о них было рассказано.

Возможно, главной причиной этого издания является то, что изменилась «аудитория» Python. За последние пять лет Python превратился из развивающегося языка, которым интересуются преимущественно первопроходцы, в широко используемый программистами инструмент для решения повседневных задач. Данное издание переориентировано на эту новую аудиторию Python. Вы увидите, что теперь это более посвященная техническим деталям книга, в меньшей степени нацеленная на знакомство с языком и его популяризацию, чем на рассказ о том, как применять его к задачам программирования реального масштаба.

Объем изменений таков, что данное издание отчасти представляет совершенно новую книгу. Хочу выразить признательность тем читателям, которым понравилась первая книга; надеюсь, что тот же дух они обнаружат во втором издании. И хотя книга в значительной мере переработана, я попытался сохранить возможно большую часть материала и стиля первоначальной книги (в особенности шутки :-).

После выхода пять лет назад первого издания мне предоставилась возможность преподавать Python в США и за рубежом, и некоторые новые примеры отражают опыт, полученный в результате этого учебного процесса. Новые примеры областей применения отражают часто возникающие вопросы и интересы – как мои собственные, так и моих учеников. Преподавание Python практическим работникам, многие из которых теперь *вынуждены* использовать Python в своей работе, обусловило новый уровень связи с практикой, что вы заметите в выборе примеров и тем данного издания.

Другие примеры появились на свет как результат удовольствия, получаемого мной при программировании на Python. Да, удовольствия: я твердо уверен, что одним из самых больших неосязаемых достоинств Python является его способность вызывать радость программирования у новичков и снова вызывать эту радость у тех, кто годами трудился с использованием более требовательных инструментов. В данном издании будет продемонстрировано, что Python чрезвычайно облегчает применение более сложных, но полезных инструментов, таких как потоки, сокеты, GUI, веб-сайты и ООП – областей, которые могут показаться и утомительными и пугающими в традиционных компилируемых языках, таких как C и C++.

Честно говоря, даже после восьми лет пребывания в качестве добросовестного «питониста» (*Pythonista*)[1] мне все еще приятно заниматься программированием, когда я делаю это на Python. Python является чрезвычайно продуктивным языком, а рассмотрение его приложений доставляет в первую очередь эстетическое удовольствие. Надеюсь, что данное издание, так же как и предыдущее, продемонстрирует, как использовать преимущества Python в продуктивности, и отчасти передаст удовлетворение и восхищение, доставляемые таким средством быстрой разработки приложений, как Python.

Основные изменения, внесенные во второе издание

Конечно, лучше всего составить впечатление о какой-либо книге, прочтя ее. Но для тех, кто читал первое издание, в последующих нескольких разделах специально более подробно описывается, что нового внесено в данное издание.

---

[1] После выхода первого издания приверженцы Python стремились найти для себя название. Чаще всего встречается *Pythonista*, но горстка ветеранов упорно называют себя *Pythoneer* или еще как-нибудь. Pythonista имеет псевдовоинственный оттенок, что не обязательно отражает истинное лицо политики языка сценариев.

Издание обновлено в соответствии с Python 2.0

Данное издание обновлено в соответствии с Python 2.0, а материал по графическим интерфейсам пользователя (GUI) обновлен для Tk версий 8.0 и более поздних. Формально это обновление началось с Python 1.5.2, но все примеры перед публикацией были проверены под версией 2.0.

Для тех, кто любит мелочи: выпуск 2.0 был первым выпуском Python после перехода Гвидо в BeOpen, а 1.6 был последним выпуском у предыдущего работодателя Гвидо, CNRI. Перед самым завершением мной окончательного наброска книги и после выхода версии 2.0 Гвидо и основная команда разработчиков Python перешли из BeOpen в Digital Creations, родину Zope – инструментального набора создания веб-приложений, но это перемещение не зависит от выпусков Python (дополнительные подробности смотрите в главе 1 «Введение в Python»).

В версии 2.0 появились новые расширения языка, но 2.0 и 1.6 по содержанию сходны, и обновление только добавляет несколько функций. Примечательно, что большинство примеров из первого издания по-прежнему действует пять лет спустя с последними выпусками Python; для тех, которые не работали, потребовались незначительные исправления (например, формат вызова GUI и интерфейсы C API).

С другой стороны, хотя основы языка не сильно изменились с момента первого издания, в него был добавлен ряд новых конструкций, и мы их всех применим тут. В число этих новых характеристик Python входят пакеты модулей, классы исключений, псевдо-приватные аттрибуты классов, строки в Unicode, новый модуль регулярных выражений, новые функции Tkinter, такие как построитель таблиц, стандартные диалоги и меню верхнего уровня и т. д. В новом приложении приводится сводка всех важнейших изменений в Python между первым и вторым изданиями этой книги.

Помимо изменений в языке в данной книге представлены новые инструменты и приложения Python, появившиеся за последние годы. В их число входят интерфейс программирования IDLE, компилятор JPython (известный также как «Jython»), расширения Active Scripting и COM, структура создания веб-приложений Zope, серверные страницы Python (Python Server Pages – PSP), режим ограниченного исполнения, генераторы кода HTMLgen и SWIG, поддержка потоков, модули CGI и протоколов Интернета и многое другое (эти пять лет были активными). Такие приложения составляют самую суть второго издания.

Книга переориентирована на более подготовленную аудиторию

В данном издании программирование на Python представлено *усложненными примерами*. Чтобы стать профессионалом в Python, нужно решить две различные задачи: овладеть основами самого языка, а затем научиться применять его в приложениях. Решение второй (и более крупной) задачи в данном издании осуществляется путем представления библиотек Python, инструментальных средств и технологий программирования. Поскольку это совершенно иная задача, следует сказать здесь несколько слов о том, почему она была поставлена.

Поскольку во время появления первого издания этой книги других изданий по Python на горизонте заметно не было, оно было обращено сразу к очень широкой аудитории: и к новичкам и к гуру одновременно. С тех пор появилась другая книга O'Reilly, «Learning Python» («Изучаем Python»), которая была ориентирована на новичков, и был опубликован «Python Pocket Reference» («Карманный справочник по Python») для читателей, нуждающихся в кратком справочнике по Python. В результате ввод-

ный материал по основам языка и первоначальные справочные приложения были изъяты из этой книги.

«Изучаем Python» знакомит с основами языка – синтаксисом, типами данных и т. д. – с помощью намеренно упрощенных примеров. Многие считают эту книгу идеальной для изучения самого языка, но Python может стать еще интереснее, когда вы овладеете базовым синтаксисом и сможете писать простые примеры в интерактивном интерфейсе. Научившись разрезать список, вы весьма скоро захотите делать реальные вещи, например писать сценарии для сравнения каталогов файлов, отвечать на запросы пользователя через Интернет, выводить графические изображения в окнах, читать электронную почту и т. д. Повседневная работа состоит в основном в применении языка, а не в самом языке.

Данная книга «Программирование на Python» сконцентрирована на «всем остальном» в разработках на Python. В ней рассказывается о библиотеках и инструментарии, находящихся за рамками основного языка, которые приобретают первостепенное значение при написании реальных приложений. Она также рассматривает вопросы проектирования более крупных приложений, например повторное использование кода и ООП, которые можно проиллюстрировать только в контексте программ более реального масштаба. Иными словами, книга «Программирование на Python», особенно в этом новом издании, призвана подхватить изложение там, где оно закончилось в «Изучаем Python».

Поэтому, если данная книга покажется вам слишком сложной, я советую предварительно прочесть «Изучаем Python» и после овладения основами вернуться сюда за продолжением рассказа. Если только у вас нет значительного опыта в программировании, это издание лучше всего использовать в качестве второй своей книги по Python.

В книге раскрыты новые темы

Большинство изменений в данном издании было сделано для того, чтобы осветить новые темы. В нем появились новые главы и разделы, рассказывающие о сценариях в Интернете, сценариях CGI, интерфейсах операционных систем, генераторе интегрирующего кода SWIG, более сложных разделах Tkinter, генераторе веб-страниц HTMLgen, JPython, потоках, режиме ограниченного исполнения и многом другом. Весь объем можно уяснить, обратившись к оглавлению, а здесь перечислены некоторые из новых тем и структурных изменений, которые появились в данном издании:

Темы

Уделено значительно большее внимание Интернету, системному программированию, графическим интерфейсам Tkinter и интеграции с C. Можно утверждать, что на них теперь в основном сосредоточен данный текст. Например, появилось шесть новых глав по сценариям в Интернете, в которых рассказано об инструментах на стороне клиента, сценариях на стороне сервера, веб-сайтах и более сложных темах и системах, связанных с Интернетом. Четыре новые главы посвящены системным вопросам: потокам, обработке каталогов, запуску программ и т. д. Материал по GUI также вырос с одной главы до значительно более полного представления в четырех главах, охватывая теперь все графические элементы (в том числе текстовые и холст), а также новую поддержку таблиц, меню и диалогов.

Интеграция с C

Главы, посвященные расширениям и встраиванию C, дополнены новым материалом, охватывающим такие темы, как SWIG (способ, которым в настоящее время следует соединять Python с библиотеками C/C++), и представляющим новые примеры смешанного режима, такие как диспетчер функций обратного вызова (рас-

ширение и встраивание). Интеграция с C лежит в сердцевине многих систем Python, но примеры из этой области неизбежно сложны и включают в себя большие программы на C, интересные только пользователям C. Из уважения к читателям, которым не требуется интегрировать Python с C, этот материал теперь помещен отдельно в конце книги. Некоторые листинги кода на C также удалены, чтобы сократить объем книги: вместо них я предпочитал по возможности отсылать читателей к исходным файлам на C на прилагаемом CD.

Хотя более поздние главы основываются на материале более ранних глав, темы в данном издании освещаются достаточно независимо и объединяются в части книги. Благодаря этому не будет большой натяжкой считать, что в данном издании объединены в одно целое четыре или пять книг. Структура основных разделов книги подчеркивает ее концентрацию на темах приложения:

Два замечания. Не дайте заглавиям ввести вас в заблуждение: хотя большинство разделов относится к темам приложений, все же попутно рассматриваются характеристики и общие идеи проектирования языка Python в контексте практических задач. И второе: читатели, использующие Python в качестве самостоятельного инструмента, могут благополучно пропустить главы, связанные с интеграцией, хотя все же рекомендуется бегло взглянуть на них. Программирование на C не дает такого удовольствия и легкости, как программирование на Python. Однако поскольку интеграция играет центральную роль в использовании Python в качестве языка сценариев, некоторое понимание ее может оказаться полезным независимо от того, занимаетесь ли вы интеграцией, написанием сценариев или тем и другим вместе.

Читатели первого издания обратят внимание на то, что материал по большей части является новым, причем даже главы, сохранившие свое название, сильно обновились. Заметно отсутствие в данном издании первоначального краткого обзора, минисправочника и учебного приложения, а также полное отсутствие прежней части II, что отражает новую направленность книги и предполагаемую аудиторию.

Книга более ориентирована на примеры

Эта книга в значительной мере состоит из примеров. В настоящем издании старые примеры расширены, чтобы придать им большую реалистичность (например, PyForm и PyCalc); кроме того, всюду добавлены новые примеры. В число основных примеров входят:

PyEdit

Объект и программа редактора текстовых файлов на Python/Tk.

PyView

Слайд-шоу для фотоизображений и файлов заметок.

PyDraw

Графический редактор для рисования и перемещения объектов изображений.

PyTree

Программа для изображения древовидных структур данных.

PyClock

Графический элемент для изображения аналоговых и цифровых часов на Python/Tk.

PyToe

Графическая программа игры в крестики-нолики с искусственным интеллектом.

PyForm

Броузер для таблиц постоянных объектов.

PyCalc

Графический элемент калькулятора на Python/Tk.

PyMail

Почтовый клиент с поддержкой POP и SMTP на Python/Tk.

PyFtp

Простой GUI для пересылки файлов на Python/Tk.

PyErrata

Размещаемая в Сети система сообщений об ошибках.

PyMailCgi

Размещаемый в Сети интерфейс электронной почты.

Есть также новые примеры интеграции с C в смешанном режиме (например, регистрация функции обратного вызова (callback) и работа с объектом класса), примеры SWIG (с «теневыми» классами для C++ и без них), дополнительные примеры Интернета (сценарии отправки и получения по FTP, примеры NNTP и HTTP, средства работы с электронной почтой и новые примеры модулей `socket` и `select`), много новых примеров потоков в Python и новое освещение JPython, HTMLgen, Zope, Active Scripting, COM и интерфейсов Python к базам данных. Многие новые примеры являются довольно продвинутыми, впрочем, как и весь текст.

Кроме того, прежний API для встраивания в C кода на Python (называемый теперь *ppembed*) расширен с целью поддержки предварительной компиляции строк в байт-коды, а первоначальный пример калькулятора (имеющий теперь название *PyCalc*) усилен и стал поддерживать ввод с клавиатуры, буфер команд, цвета и другие функции.

На практике примеры в новой книге, распространяемые на прилагаемом к данному изданию CD-ROM, *сами по себе* являются довольно сложной системой программ на Python, входящие в которую примеры структурно изменены в ряде важных направлений:

Дерево примеров

Весь дистрибутив примеров организован в виде одного большого *пакета модулей* Python, что облегчает импорт из других каталогов и устраняет конфликт имен с другим кодом Python, установленным на компьютере. Использование путей каталогов в командах import (вместо сложного PYTHONPATH) облегчает также установление происхождения модулей. Более того, теперь нужно добавить в путь поиска PYTHONPATH только один каталог для всего дерева примеров в книге: каталог, содержащий корневой каталог примеров *PP2E*. Для использования примеров, приведенных в книге, в собственных приложениях нужно просто импортировать их через корень пакета *PP2E* (например, `from PP2E.Launcher import which`).

Имена файлов примеров

Имена модулей стали значительно менее таинственными. Я давно перестал обращать внимание на совместимость с форматом DOS 8.3 и использую более описательные имена. Были также исправлены некоторые старые имена файлов, полностью записанные в верхнем регистре – последнее наследие MS-DOS.

Названия примеров

В метках листингов примеров теперь указывается полный путь к файлу исходного кода примера, что помогает найти его в дистрибутиве примеров. Например, файл с исходным кодом примера, имя которого указано как *Example N-M: PP2E\Internet\Ftp\sousa.py*, указывает на файл *sousa.py* в подкаталоге *PP2E\Internet\Ftp* каталога дистрибутива примеров.[1]

Командные строки примеров

Аналогично, когда показана командная строка, введенная после приглашения, например `C:\...\PP2E\System\Streams>`, она в действительности должна быть введе-

[1] «Каталог дистрибутива примеров» является каталогом, содержащим каталог верхнего уровня *PP2E* дерева примеров книги. На CD это самый верхний каталог *Examples*; если вы скопировали примеры на диск, это то место, куда вы скопировали (или распаковали) корневой каталог *PP2E*. Большинство этих примеров можно непосредственно запускать с CD, но они должны быть скопированы на жесткий диск для внесения в них изменений и чтобы позволить Python сохранять файлы компилированного байт-кода *.pyc*, допускающие более быстрый начальный запуск.

на так, чтобы указывать на подкаталог *PP2E\System\Streams* в вашем дереве примеров. Пользователям Unix и Linux: пожалуйста, имейте в виду / при встрече с \ в путях к файлам (официальное извинение по этому поводу выражено в следующем разделе).

Средства запуска примеров

Поскольку приятно иметь возможность сразу щелкнуть куда нужно мышкой, есть новые самоконфигурирующиеся программы запуска демонстрационных примеров (описываемые ниже в этом предисловии в разделе «Вкратце»), позволяющие быстро взглянуть на сценарии Python в действии с минимальной предварительной настройкой. Обычно можно запускать их прямо с CD, не устанавливая каких-либо переменных окружения.

Книга более нейтральна в отношении используемых платформ

За исключением все тех же примеров с интеграцией с C, большинство программ в этом издании было разработано на моем портативном компьютере под Windows 98 с прицелом на переносимость под Linux и другие платформы. В действительности некоторые примеры порождены моим желанием создать на Python переносимые эквиваленты некоторых инструментов, отсутствующих в Windows (например, программ для разбивки файлов на части). Когда программы показываются в действии, это обычно делается на Windows; на платформе Red Hat Linux 6.x они демонстрируются только при использовании специфических для Unix интерфейсов.

Это вовсе не политическое заявление: Linux мне тоже нравится. Это в основном следствие того, что я писал эту книгу с помощью MS Word; когда время ограничено, удобнее запускать сценарии на той же платформе, что и издательские средства, чтобы не перегружаться слишком часто в Linux. К счастью, поскольку Python стал теперь в большой степени переносим между Windows и Linux, используемая операционная система меньше заботит разработчиков Python, чем было ранее. Python, его библиотеки и система GUI Tkinter в настоящее время прекрасно работают на обеих платформах.

Однако поскольку я не занимаюсь политикой, то постарался сделать примеры возможно более независимыми от платформы и попутно указать на специфические для каждой платформы проблемы. Вообще говоря, большинство сценариев должно работать на обычных платформах Python без внесения изменений. Например, все примеры GUI были проверены под Windows (98, 95) и Linux (KDE, Gnome), а большинство примеров командной строки и потоков были разработаны в Windows, но работают и в Linux. Поскольку системные интерфейсы Python обычно строятся в расчете на переносимость, сделать это легче, чем может показаться.

В то же время, эта книга при необходимости погружается в специфические для платформ темы. Заново изложены многие специальные темы Windows: Active Scripting, COM, варианты запуска программ и т. д. Читатели, работающие под Linux и Unix, найдут и материал, относящийся к их платформе: ветвление, конвейеры и тому подобное. Заново изложены также способы редактирования и запуска программ Python на большинстве главных платформ.

Единственное место, где читатели смогут усмотреть уклон в сторону определенной платформы, – это примеры интеграции Python с C. Для простоты детали компиляции программ на C, о которых говорится в данной книге, все еще имеют некоторый уклон в сторону Unix/Linux. Во всяком случае этому можно найти разумное объяснение: Linux не только поставляется с бесплатными компиляторами C, но вокруг этого языка выросла вся его среда разработки. Код расширений на C, приведенных в этой книге,

Но все равно это не справочное руководство

Обратите, пожалуйста, внимание на то, что это издание, как и первое, все же скорее *учебник*, чем справочное руководство (несмотря на название, сходное с популярным справочником по Perl). Эта книга должна учить, а не документировать. С помощью ее оглавления и предметного указателя можно найти конкретные детали, и новая структура облегчает это. Но все же данное издание задумано как книга, которую используют вместе со справочниками по Python, а не вместо них. Поскольку руководства по Python бесплатны, написаны хорошо, доступны в Сети и часто редактируются, было бы безумием тратить место на копирование их содержания. Исчерпывающий список всех инструментов, имеющихся в системе Python, можно найти в других книгах (например, изданной O'Reilly «Python Pocket Reference») или стандартных руководствах на веб-сайте Python либо на компакт-диске, прилагаемом к данной книге.

будет работать и под Windows, но может потребоваться использование различных процедур сборки программ в зависимости от используемого в Windows компилятора. Издательство O'Reilly опубликовало замечательную книгу «Python Programming on Win32», в которой освещены специфические для Windows темы Python вроде этой, которая должна помочь справиться с некоторыми наблюдаемыми здесь расхождениями. Если вы занимаетесь программированием под Windows, то найдете все относящиеся к Windows подробности, которые здесь пропущены.

Использование примеров и демонстрационных программ

Хочу здесь кратко описать, как использовать имеющиеся в книге примеры. В целом, однако, посмотрите, пожалуйста, следующие текстовые файлы в каталоге дистрибутива примеров, в которых вы найдете дополнительные сведения:

- *README-root.txt* – замечания о структуре пакета
- *PP2E\README-PP2E.txt* – общие замечания об использовании
- *PP2E\Config\setup-pp.bat* – конфигурация для Windows
- *PP2E\Config\setup-pp.csh* – конфигурация для Unix и Linux

Из этих файлов наиболее информативен *README-PP2E.txt*, а в каталоге *PP2E\Config* содержатся все файлы примеров конфигурации. Здесь дается обзор, но в перечисленных файлах находится полное описание.

Вкратце

Если вы сразу хотите посмотреть некоторые примеры Python, поступите так:

1. Установите Python с компакт-диска, прилагаемого к книге, если он еще не установлен на вашем компьютере. В Windows щелкните на имени самоустанавливающейся программы на CD и сделайте установку по умолчанию (отвечайте «yes» или «next» на все приглашения). Для других систем посмотрите файл README (заархивированный дистрибутив исходного кода на CD можно использовать для локальной сборки Python).
2. Запустите один из следующих *самоконфигурирующихся сценариев*, располагающихся в каталоге CD верхнего уровня Examples\PP2E. Щелкните по соответству-

ющим пиктограммам в проводнике или запустите из системной подсказки (например, окна консоли DOS или Linux Xterm) посредством командной строки формата `python имя-сценария` (может потребоваться указать полный путь к `python`, если он не включен в систему):

- *Launch_PyDemos.pyw* – главная инструментальная панель для запуска демонстрационных программ Python/Tk
- *Launch_PyGadgets_bar.pyw* – инструментальная панель для запуска утилит Python/Tk
- *Launch_PyGadgets.py* – запускает стандартные утилиты Python/Tk
- *LaunchBrowser.py* – открывает указатель веб-примеров в веб-броузере

Сценарии `Launch_*` запускают программы Python переносимым образом[1] и требуют только наличия установленного Python: для их запуска не требуется предварительной настройки переменных окружения или изменений в прилагаемых файлах настройки *PP2E\Config*. `LaunchBrowser` сможет работать, если найдет на вашей машине веб-броузер, даже если у вас нет соединения с Интернетом (хотя некоторые интернет-примеры работают неполным образом в отсутствие работающего соединения).

Если отказаться от установки Python, все же можно запустить несколько веб-демонстраций Python, отправив броузер на *http://starship.python.net/~lutz/PyInternetDemos.html*. Так как эти примеры предназначены для выполнения сценариев на сервере, лучше всего они работают при запуске на этом сайте, а не с CD, прилагаемого к книге.

Подробности

Для лучшей организации новых примеров я поместил в каталог верхнего уровня дистрибутива примеров *PP2E* программу запуска демонстраций, *PyDemos.pyw*. На рис. П.1 показана PyDemos в действии под Windows после нажатия нескольких кнопок. Панель запуска появляется в левой части экрана; с ее помощью можно щелчком мыши запустить большинство графических примеров, приведенных в книге. Если на вашей машине удастся обнаружить броузер, то с помощью панели запуска демонстраций можно также запускать основные примеры для Интернета (смотрите описание программы запуска далее).

Помимо запуска демонстрационных программ исходный код PyDemos дает указатели на основные примеры в дистрибутиве; детали ищите в исходном коде программы. В каталоге верхнего уровня примеров вы также обнаружите сценарии для автоматической компиляции под Linux примеров интеграции Python и C, которые служат указателями для основных примеров на C.

Я также включил программу верхнего уровня под названием *PyGadgets.py* и родственную ей *PyGadgets_bar.pyw*, чтобы запускать для реального использования, а не для демонстрации некоторые наиболее полезные примеры GUI из книги (в большинстве своем часто используемые мной программы; переконфигурируйте их, если захотите). На рис. П.2 показано, как выглядит в Windows `PyGadgets_bar`, а также некоторые утилиты, которые могут запускаться его кнопками. Все эти программы представлены в данной книге и включены в дистрибутив примеров. Для большинства из

[1] Все демо- и запускающие сценарии написаны переносимым образом, но на момент написания книги известно, что они работают только под Windows 95/98 и Linux; для других платформ могут потребоваться незначительные изменения. Приношу извинения, если вы используете платформу, которую я не смог проверить: Tk работает под Windows, X11 и на Macs; сам Python работает всюду – от карманных PDA до мэйнфреймов; и мой вклад в написание этой книги был не столь велик, как вы могли подумать.

Рис. П.1. Запускающая программа PyDemos со всплывающими окнами и демонстрационными программами (фотография Гвидо перепечатана с разрешения Dr. Dobb's Journal)

них требуется Python с поддержкой Tkinter, но это и есть стандартная конфигурация для запуска материалов с CD этой книги под Windows.

Для непосредственного запуска файлов, перечисленных в предыдущем параграфе, требуется настроить путь поиска модулей Python (подсказку ищите в файлах `PP2E/Config/setup*`). Но если вы хотите запустить собрание демонстрационных программ Python из данной книги и не хотите утруждать себя предварительной настройкой окружения, просто выполните сценарии утилит в каталоге *PP2E*: *Launch_PyDemos.pyw*, *Launch_PyGadgets.py* и *Launch_ PyGadgets_bar.pyw*.

Эти сценарии для запуска программ, написанные на Python, предполагают, что Python уже установлен, и автоматически найдут интерпретатор языка и дистрибутив примеров из книги, а также настроят пути к модулям Python и системные пути поиска так, как это требуется для запуска демонстрационных программ. Вероятно, вы сможете запустить эти сценарии, просто щелкая по их именам в проводнике, а также сможете запустить их непосредственно с прилагаемого CD-ROM. Дополнительные сведения можно найти в комментариях в начале *Launcher.py* (или прочесть об этих сценариях в главе 4 «Более крупные системные примеры, часть 1»).

Многие из примеров для Интернета, использующих броузеры, можно найти в сети на *http://starship.python.net/~lutz/PyInternetDemos.html*, где можно проверить их работу. Поскольку эти примеры выполняются в броузере, их можно посмотреть, даже если на машине не установлен Python (или поддержка Tk для Python).

Программа PyDemos также пытается запустить броузер с веб-страницами основных примеров путем выполнения сценария *LaunchBrowser.py* в корневом каталоге примеров. Этот сценарий обычно успешно находит на вашей машине броузер, которым можно воспользоваться; если поиск окажется неудачным, смотрите дополнительные

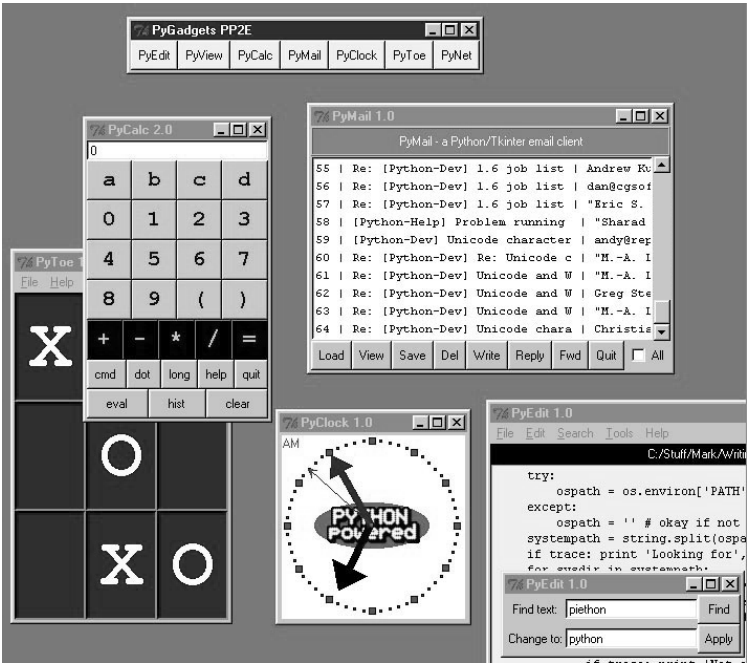

Рис. П.2. Панель запуска утилит PyGadgets вместе с программами

сведения в сценарии. Если `LaunchBrowser` сможет найти на вашей машине броузер, некоторые кнопки автоматически выведут веб-страницы независимо от наличия действующего соединения с Интернетом (если соединения нет, в броузере будут показаны локальные файлы). На рис. П.3 показано, как выглядит страница PyInternetDemos в Internet Explorer под Windows.

Особенно интересно, что ссылка *getfile.html* на этой странице позволяет просмотреть исходный код любого другого файла на сайте книги – код HTML, CGI-сценарии Python и т. д.; подробности смотрите в главе 12. Подведем итоги; вот что вы найдете в каталоге верхнего уровня *PP2E* дистрибутива примеров книги:

PyDemos.pyw

Панель кнопок для запуска основных примеров GUI и Интернета.

PyGadgets.py

Запуск программ в недемонстрационном режиме для обычного использования.

PyGadgets_bar.pyw

Панель кнопок для запуска PyGadgets по требованию.

*Launch_*.py**

Запускает программы PyDemos и PyGadgets с помощью *Launcher.py*, автоматически конфигурирующего пути поиска (запускайте их, чтобы бегло взглянуть).

Launcher.py

Используется для запуска программ без настройки окружения: находит Python, устанавливает PYTHONPATH, запускает программы Python.

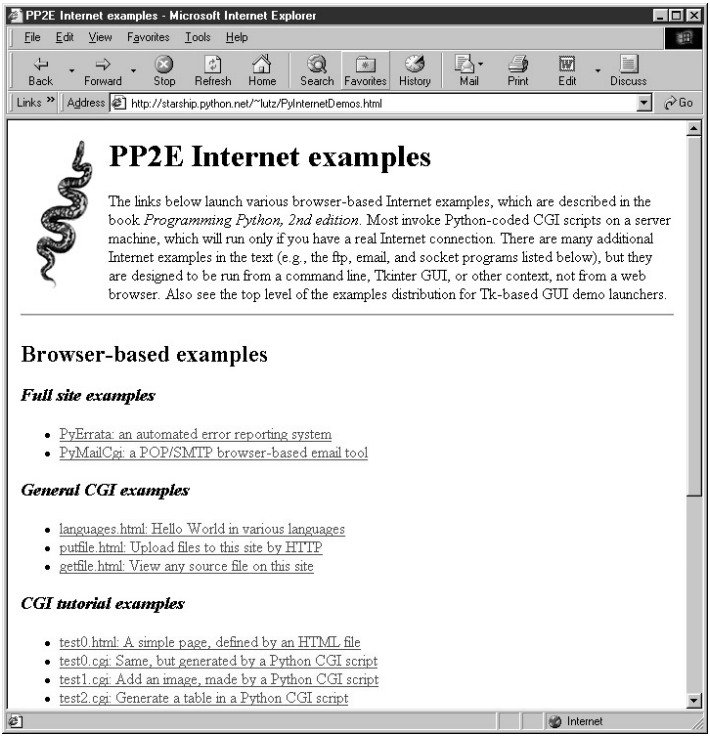

Рис. П.3. Веб-страница PyInternetDemos

LaunchBrowser.py

Открывает веб-страницы примеров в автоматически найденном броузере, загружая их из Сети или из локальных файлов; при непосредственном запуске открывает страницу указателя PyInternetDemos.

Существуют также подкаталоги для примеров из каждой основной группы тем, рассматриваемых в книге.

Кроме того, каталог верхнего уровня *PP2E\PyTools* содержит написанные на Python утилиты командной строки для преобразования символов перевода строки во всех текстовых файлах примеров в формат DOS или Unix (полезно использовать, если они странно выглядят в вашем текстовом редакторе), превращения всех файлов примеров в доступные для записи (полезны, если вы перетаскиваете файлы с прилагаемого CD), удаления старых файлов байт-кода *.pyc* в дереве каталогов и другие. Дополнительные сведения, касающиеся всех вопросов, связанных с примерами, ищите в файле *README-PP2E.txt*.

Где все это находится

Дистрибутив примеров книги находится на сопровождающем ее компакт-диске. Детали использования описаны в файле README на верхнем уровне каталогов CD. Для беглого ознакомления можно просмотреть корневой каталог примеров на компакт-диске в своем любимом файловом проводнике.

Помимо примеров из книги CD содержит также различные относящиеся к Python пакеты, в том числе полную *программу самоустановки* под Windows с поддержкой Python и Tk (для установки дважды щелкните по ней и ответьте «yes» на все приглашения), полный *дистрибутив исходного кода* Python (распакуйте и откомпилируйте на своей машине) и набор *стандартной документации* Python в формате HTML (щелкните для просмотра в своем веб-броузере).

Дополнительные пакеты открытого исходного кода, например последние версии (на момент публикации) генератора кода SWIG и Jpython, тоже помещены на CD, но вы всегда можете найти новейшие версии Python и других пакетов на веб-сайте Python *http://www.python.org*.

Использованные в книге типографские обозначения

В данной книге принято следующее использование шрифтов:

Курсив

Используется в именах файлов и каталогов, URL, командах, для выделения новых терминов при первом знакомстве и в некоторых комментариях в фрагментах кода.

`Моноширинный`

Используется в листингах кода и для обозначения модулей, методов, опций, классов, функций, утверждений, программ, объектов и тегов HTML.

`Моноширинный полужирный`

Используется в приводимом коде для показа данных, вводимых пользователем.

`Моноширинный курсив`

Используется для обозначения заменяемого пользователем текста.

Изображение совы обозначает замечание, относящееся к близлежащему тексту.

Изображение индюшки обозначает предупреждение, относящееся к близлежащему тексту.

Где можно найти обновления

Как и прежде, обновления, исправления и дополнения для данной книги поддерживаются на веб-сайте автора *http://www.rmi.net/~lutz*. Найдите на этой странице ссылку на второе издание, чтобы получить всю дополнительную информацию, касающуюся данной версии книги. Как и для первого издания, я буду вести на этом сайте журнал регистрации изменений, производимых в Python, который следует рассматривать как дополнение к данной книге.

Начиная с этого издания я открываю в Сети систему сообщения пользователями об ошибках, найденных в книге, на сайте:

http://starship.python.net/~lutz/PyErrata/pyerrata.html

Там вы найдете формы для сообщений о проблемах, возникших с книгой, и комментариев, а также сможете просматривать базу данных сообщений, отсортированную по различным ключам. По умолчанию сообщения хранятся в базе данных общего доступа, но при желании можно направлять их закрытым письмом. Система PyErrata тоже написана на Python и составила пример, вошедший в данную книгу; смотрите главу 14. Рис. П.4 показывает внешний вид корневой страницы PyErrata.

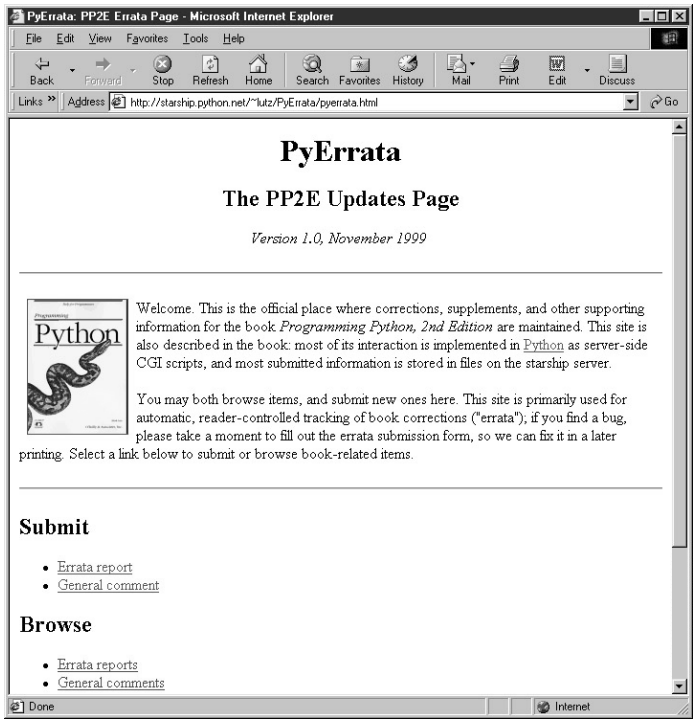

Рис. П.4. Сайт PyErrata вносимых в книгу исправлений

Если какие-либо из этих адресов окажутся недействующими, то получить доступ к этим страницам можно также на веб-сайте издательства O'Reilly *http://www.oreilly.com*.[1] Я по-прежнему буду рад получать прямые письма читателей, но надеюсь, что PyErrata усовершенствует процесс передачи сообщений об ошибках.

Связь с O'Reilly

Можно также обратиться с комментариями и вопросами относительно данной книги к издателю:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)

[1] На веб-сайте O'Reilly тоже есть система приема сообщений об ошибках, и оба эти списка в совокупности следует рассматривать как официальное заключение о найденных ошибках и внесенных изменениях.

(707) 829-0515 (international/local)
(707) 829-0104 (fax)

В O'Reilly есть веб-страница, посвященная данной книге, на которой приведены ошибки, примеры и любая дополнительная информация. Эта страница находится на

http://www.oreilly.com/catalog/python2/

Для комментариев или технических вопросов по данной книге пишите по адресу:

bookquestions@oreilly.com

Дополнительную информацию о книгах, конференциях, программном обеспечении, центрах ресурсов и сети O'Reilly Network можно найти на веб-сайте O'Reilly на

http://www.oreilly.com

Благодарности

Помимо людей, упомянутых в первом издании, я хотел бы дополнительно выразить признательность тем, кто так или иначе помогал мне во время данного проекта второго издания:

- Редактору первого издания Фрэнку Уиллисону (Frank Willison) за то, что он просмотрел эту редакцию и продвигал Python как в O'Reilly, так и за его пределами. Следующему редактору данной книги Лоре Левин (Laura Lewin), перенявшей у него эстафету.
- Создателю Python Гвидо ван Россуму (Guido van Rossum), благодаря которому работа над книгой снова стала удовольствием.
- Участникам рецензирования раннего проекта данного издания: Эрику Рэймонду (Eric Raymond), Марку Хэммонду (Mark Hammond), Дэвиду Эшеру (David Ascher), Тиму Петерсу (Tim Peters) и Дэйву Бизли (Dave Beazley).
- Тиму О'Рейлли (Tim O'Reilly) и служащим O'Reilly & Associates как за выход этой книги, так и за поддержку программного обеспечения с открытым кодом в целом.
- Сообществу Python вообще за прилежание, напряженную работу и юмор – в прежние годы и сейчас. Мы далеко продвинулись, но вспомним строчку из 1970-х: «Вы еще ничего не видите».
- Студентам многочисленных курсов Python, которым я преподавал, а также многочисленным читателям, не пожалевшим времени, чтобы послать мне комментарии по поводу первого издания: ваше мнение помогло определить характер данной новой редакции.

Наконец, несколько личных выражений благодарности. Моим детям, Майклу, Саманте и Роксане, за целеустремленность. Если они представляют свое поколение, то будущее нашего вида находится в хороших руках. Вам придется извинить меня за гордыню, но с такими детьми, как мои, невозможно чувствовать иначе.

А самая большая благодарность Лайзе, матери этих изумительных детей. Я в самом большом долгу перед ней за все – от терпеливого отношения к моим уходам от реальности, когда я пишу книги вроде этой, до того, что она спасла меня от тюрьмы в годы нашей молодости. Что бы ни таило нам будущее, я благодарен судьбе, соединившей нас два десятилетия назад.

Марк Лутц
Ноябрь 2000
Где-то в Колорадо

«Когда Билли пойдет вниз, с ним это произойдет быстро»

За последние пять лет наблюдался также подъем движения за программное обеспечение с открытым кодом (open source) – это программы, бесплатно распространяемые вместе с полным исходным кодом, обычно являющиеся результатом работы многих разработчиков, сотрудничающих друг с другом в свободной манере. В эту категорию попадают Python, операционная система Linux и многие другие инструменты, например Perl и веб-сервер Apache. Частично благодаря вызову, брошенному им преобладанию на рынке мегакомпаний, движение программного обеспечения с открытым кодом быстро и глубоко проникло в общество.

Позвольте рассказать о событии, недавно подчеркнувшем степень влияния на меня этого движения. Чтобы вам стал понятен этот рассказ, следует сообщить, что я писал эту книгу в небольшом городке в Колорадо, о котором нельзя сказать, что он находился на переднем крае технологических нововведений. Сказать образнее, это одно из тех мест, которые называют «ковбойский городок».

Я зашел в небольшую местную книжную лавку в поисках последнего номера *Linux Journal*. После недолгих поисков я нашел требуемое и направился к кассе. За прилавком стояли двое служащих, внешний вид которых более соответствовал занятию родео, чем нахождению за прилавком этого заведения. Старший из них был седым, с усами и с задубелой кожей человека, привыкшего к жизни на ранчо. Оба носили обязательные бейсбольные кепки. Вылитые ковбои.

Когда я положил журнал, старший из служащих на некоторое время поднял глаза и сказал, типично по-ковбойски растягивая слова: «Угу, Линукс? Вот что я скажу: когда Билли пойдет вниз, с ним это произойдет быстро!» Разумеется, речь шла о широко комментировавшемся соревновании между Linux и Microsoft Windows Билла Гейтса, усилившемся благодаря движению open source.

В другое время эти двое могли бы рассуждать о коровах и револьверах, сидя за чашкой крепкого кофе. Однако каким-то странным образом они стали страстными защитниками Linux – операционной системы с открытым исходным кодом. Подобрав отвалившуюся челюсть, я вступил с ними в оживленный разговор о Linux, Microsoft, Python и всяких открытых вещах. Можно сказать, мы славно поболтали.

Я не хочу отдавать предпочтение одной операционной системе перед другой: у каждой из них есть свои преимущества, а Python одинаково хорошо работает на той и другой платформе (в действительности вошедшие в книгу примеры разрабатывались на обеих системах). Но меня поразило, что мысль, которую разработчики программного обеспечения часто считают само собой разумеющейся, имела такое глубокое влияние на средних американцев. Это вселяет в меня большие надежды: если технология действительно должна повысить качество жизни в следующем тысячелетии, нам нужно побольше таких ковбоев.

9

Более крупные примеры GUI

«Как улучшить мышеловку»

В этой главе мы продолжаем изучать создание графических интерфейсов пользователей с помощью Python и его стандартной библиотеки Tkinter путем представления ряда реальных программ GUI. В трех предшествующих главах мы познакомились с основами программирования с Tkinter и обследовали базовый набор графических элементов (*widgets*) – классов Python, которые генерируют инструменты на экране компьютера и могут реагировать на события, вызываемые пользователем, например щелчки мышью. В данной главе мы займемся тем, что будем создавать более полезные GUI, соединяя эти графические элементы вместе. Нами будут изучены:

- Развитые технологии кодирования GUI
- *PyEdit* – программа текстового редактора
- *PyView* – программа слайд-шоу графических изображений
- *PyDraw* – графический редактор
- *PyClock* – графические часы
- *PyToe* – и даже простая игра в качестве развлечения[1]

Как и в главе 4 «Более крупные системные примеры, часть 1» и в главе 5 «Более крупные системные примеры, часть 2», я выбрал примеры этой главы из собственной библиотеки программ Python, которыми я действительно пользуюсь. Например, GUI текстового редактора и часов, с которыми мы здесь познакомимся, служат рабочими лошадками, изо дня в день используемыми мной на моих машинах. Так как они написаны на Python и Tkinter, то без изменений работают на машинах Windows и Linux и должны работать также на Mac.

А так как это сценарии на чистом Python, их дальнейшее развитие целиком зависит от пользователей – освоившись с интерфейсами Tkinter, изменить или улучшить поведение таких программ редактированием их кода Python легко. Хотя некоторые из этих примеров аналогичны коммерческим программам (например, PyEdit напоминает Windows Notepad), переносимость и почти безграничная настраиваемость сценариев Python дают явное преимущество.

---

[1] У всех более крупных примеров этой книги в начале имени стоит «Py». Это соглашение, принятое в мире Python. Если порыться на *http://www.python.org*, то можно найти другое свободно распространяемое программное обеспечение, следующее этой схеме: PyApache (интерфейс Python к веб-серверу Apache), PySol (игра в солитер на Python/Tkinter) и многие другие. Я не знаю, с кого началась эта схема, но она оказалась достаточно тонким способом рекламы языка программирования для всего мира программного обеспечения с открытым кодом. Если Питонист слишком прямолинеен – это не Питонист!

Примеры в других главах

Далее в этой книге мы встретим другие программы GUI на базе Tkinter, представляющие удобные интерфейсы для конкретных областей приложений. В частности, в последующих главах появятся такие более крупные примеры GUI:

- *PyMail* – клиент электронной почты в главе 11 «Создание сценариев, выполняемых на стороне клиента»
- *PyForm* – средство просмотра таблиц постоянных объектов в главе 16 «Базы данных и постоянное хранение данных»
- *PyTree* – средство просмотра древовидных структур данных в главе 17 «Структуры данных»
- *PyCalc* – калькулятор в главе 18 «Текст и язык»

Большинством из этих программ я постоянно пользуюсь. Так как библиотеки GUI являются инструментами общего назначения, не много найдется областей приложений, которые не выиграли бы от простого в использовании, простого в программировании и хорошо переносимого интерфейса, код которого написан на Python с Tkinter.

Помимо примеров, приведенных в этой книге, для Python существуют инструментальные наборы GUI более высокого уровня, например система PMW, упомянутая в главе 6 «Графические интерфейсы пользователя». Такие системы, основываясь на Tkinter, предоставляют составные компоненты, например блокнот или элементы с закладками. Позднее мы встретимся также с программами, которые строят интерфейсы пользователя в веб-броузерах, а не Tkinter. Но за исключением простых, основанных на веб интерфейсов, GUI на базе Tkinter могут оказаться необходимой характеристикой почти каждой создаваемой на Python программы.

Стратегия данной главы

Как и все главы этой книги, посвященные исследованию конкретного случая, данная глава в значительной мере является «обучением на примере»; текст большинства программ приведен с минимумом подробностей. Попутно я буду отмечать новые функции Tkinter, появляющиеся в каждом примере, но я также буду предполагать, что вы самостоятельно изучите детали по приведенному исходному коду и комментариям. Легкость чтения Python становится существенным достоинством для программистов (и авторов), особенно когда сложность программ достигает такого уровня, как в этой главе.

Наконец, я хочу напомнить, что все перечисленные выше крупные программы можно запускать из GUI панелей запуска PyDemos и PyGadgets, с которыми мы встретились в конце прошлой главы. Я попытаюсь передать их поведение на снимках экранов, которые будут приведены, но GUI по своей природе являются системами, управляемыми событиями, и реальный запуск их для того, чтобы почувствовать характер взаимодействия с пользователем, ничем не заменишь. Поэтому панели запуска являются фактическим дополнением к материалу данной главы. Они могут выполняться на большинстве платформ и обеспечивают легкость запуска (см. файл *README-PP2E.txt*). Запускайте их и сразу начинайте щелкать мышью, если еще не сделали этого.

Более сложные приемы написания кода GUI

Если вы прочли главу 8 «Обзор Tkinter, часть 2», то знаете, что код, строящий нетривиальные GUI, может достичь большого размера, если каждый графический элемент строить вручную. Приходится не только вручную связывать все графические элемен-

ты, но нужно помнить десятки параметров, которые должны быть установлены. При такой стратегии программирование GUI часто становится упражнением по вводу с клавиатуры или операциям вырезания и вставки в текстовом редакторе.

GuiMixin: использование общих методов «подмешиваемых» классов

Вместо выполнения всех действий вручную правильно было бы создать оболочку или автоматизировать в возможно большей степени процесс построения GUI. Одним из решений является создание функций, обеспечивающих типичные конфигурации графических элементов; например можно было бы определить функцию кнопки, работающую с деталями настройки и поддерживающую большинство рисуемых кнопок.

Другим способом может быть реализация общих методов в классе и наследование их в необходимых случаях. Такие классы обычно называют *подмешиваемыми* (*mixin*), потому что их методы «подмешиваются» в другие классы. Такие классы оформляют полезные во многих случаях инструменты в виде методов. Идея близка к импортированию модулей, однако подмешиваемые классы могут обращаться к конкретному экземпляру, *self*, используя состояние конкретного объекта и унаследованные методы. Пример 9.1 демонстрирует, как это делается.

Пример 9.1. PP2E\Gui\Tools\guimixin.py

```
##############################################################################################
# "подмешиваемый" класс для других фреймов: общие методы для готовых диалогов, порождения
# программ и т. д.; должен смешиваться с классом, производным от Frame, ради его метода quit
##############################################################################################

from Tkinter import *
from tkMessageBox import *
from tkFileDialog import *
from ScrolledText import ScrolledText
from PP2E.launchmodes import PortableLauncher, System

class GuiMixin:
    def infobox(self, title, text, *args):              # использовать стандартные диалоги
        return showinfo(title, text)                    # *args для обратной совместимости
    def errorbox(self, text):
        showerror('Error!', text)
    def question(self, title, text, *args):
        return askyesno(title, text)

    def notdone(self):
        showerror('Not implemented', 'Option not available')
    def quit(self):
        ans = self.question('Verify quit', 'Are you sure you want to quit?')
        if ans == 1:
            Frame.quit(self)                            # quit не рекурсивен!
    def help(self):
        self.infobox('RTFM', 'See figure 1...')         # переопределите его более подходящим

    def selectOpenFile(self, file="", dir="."):         # использовать стандартные диалоги
        return askopenfilename(initialdir=dir, initialfile=file)
    def selectSaveFile(self, file="", dir="."):
        return asksaveasfilename(initialfile=file, initialdir=dir)

    def clone(self):
        new = Toplevel()                    # создать новую версию
```

```
        myclass = self.__class__          # объект экземпляра (самого низшего) класса
        myclass(new)                      # прикрепить/выполнить экземпляр к новому окну

    def spawn(self, pycmdline, wait=0):
        if not wait:
            PortableLauncher(pycmdline, pycmdline)()     # запустить программу Python
        else:
            System(pycmdline, pycmdline)()               # ждать ее завершения

    def browser(self, filename):
        new  = Toplevel()                                # создать новое окно
        text = ScrolledText(new, height=30, width=90)    # Text с полосой прокрутки
        text.config(font=('courier', 10, 'normal'))      # моноширинный шрифт
        text.pack()
        new.title("Text Viewer")                         # атрибуты менеджера окон
        new.iconname("browser")
        text.insert('0.0', open(filename, 'r').read() )  # вставить текст файла

if __name__ == '__main__':
    class TestMixin(GuiMixin, Frame):      # автономное тестирование
        def __init__(self, parent=None):
            Frame.__init__(self, parent)
            self.pack()
            Button(self, text='quit',  command=self.quit).pack(fill=X)
            Button(self, text='help',  command=self.help).pack(fill=X)
            Button(self, text='clone', command=self.clone).pack(fill=X)
    TestMixin().mainloop()
```

Хотя пример 9.1 ориентирован на GUI, в действительности он иллюстрирует идеи архитектуры. Класс `GuiMixin` реализует частые операции со стандартными интерфейсами, которые безразличны к возможным изменениям в реализации. На самом деле реализации некоторых методов этого класса действительно изменились – в промежутке между первым и вторым изданиями этой книги вызовы `Dialog` в старом стиле были заменены новыми вызовами стандартных диалогов Tk. Так как интерфейс этого класса скрывает такие детали, не потребовалось изменять его клиенты, чтобы они использовали новую технологию диалогов.

В данном виде `GuiMixin` предоставляет методы для стандартных диалогов, клонирования окон, порождения программ, просмотра текстовых файлов и т. д. Позднее, если обнаружится, что одни и те же методы приходится кодировать многократно, их можно добавить в такой подмешиваемый класс, и они станут немедленно доступны всюду, куда он импортируется и внедряется. Более того, методы `GuiMixin` можно наследовать и использовать в существующем виде либо переопределять в подклассе.

Здесь нужно отметить несколько вещей:

- Метод `quit` выполняет отчасти ту же задачу, что и кнопка многократного использования `Quitter` в предыдущих главах. Так как в подмешиваемых классах могут определяться большие библиотеки многократно используемых методов, они могут предоставлять более мощный способ упаковки многократно используемых компонентов, чем отдельные классы. Если правильно упаковать подмешиваемый класс, то он может дать значительно больше, чем обратный вызов одной кнопки.
- Метод `clone` создает новый экземпляр самого узкого класса, который смешивается с `GuiMixin`, в новом окне верхнего уровня (`self.__class__` является объектом класса, из которого был создан экземпляр). Он открывает новый независимый экземпляр окна.
- Метод `browser` открывает в новом окне объект стандартной библиотеки `ScrolledText` и заполняет его текстом файла, который нужно просмотреть. В предыдущей главе

мы написали собственный ScrolledText, который может потребоваться в данном случае, если класс стандартной библиотеки когда-либо будет объявлен устаревшим (но не стоит делать на это ставки).

- Метод spawn запускает командную строку программы Python как новый процесс и либо ждет его завершения, либо нет (в зависимости от аргумента wait). Этот метод прост потому, что детали запуска скрыты в модуле launchmodes, показанном в конце главы 3 «Системные средства параллельного выполнения». GuiMixin поощряет и использует на практике хорошие привычки повторного использования кода.

Класс GuiMixin предназначен служить библиотекой многократно используемых инструментальных методов и сам по себе, в сущности, бесполезен. В действительности для использования его нужно смешивать с классом, основанным на Frame: quit предполагает, что он смешивается с Frame, а clone предполагает, что он смешивается с классом графического элемента. Чтобы удовлетворить этим ограничениям, находящийся в конце код для самотестирования объединяет GuiMixin с графическим элементом Frame. На рис. 9.1 показана картина, которая возникает при самотестировании после того, как дважды нажата кнопка «clone» и затем «help» в одном из трех экземпляров.

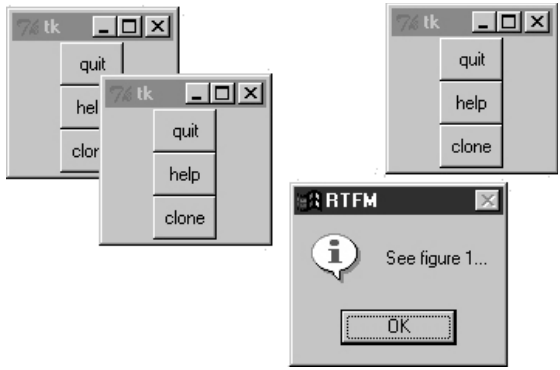

Рис. 9.1. Код самотестирования GuiMixin в действии

Мы снова встретимся с использованием этого класса в качестве mixin в последующих примерах – в конце концов, в этом весь смысл повторного использования кода.

GuiMaker: автоматизация создания меню и панелей инструментов

Подмешиваемый класс из последнего раздела облегчает выполнение стандартных задач, но не решает проблем сложности связывания в таких графических элементах, как меню и панели инструментов. Конечно, при наличии инструмента структурирования GUI, который генерировал бы код Python, проблем бы не было. Мы бы проектировали графические элементы интерактивно, нажимали кнопку и заполняли пустые места в обработчике обратного вызова.

Но пока может сгодиться и подход, основанный на программировании. Хотелось бы иметь возможность наследовать некоторый класс, который сам выполняет всю черновую работу, если задать шаблон для меню и панелей инструментов в окне. Вот один из способов, которыми это можно сделать, использующий деревья простых объектов. Класс примера 9.2 интерпретирует представление меню и панелей инструментов в виде структуры данных и автоматически строит все графические элементы.

Пример 9.2. PP2E\Gui\Tools\guimaker.py

```
##############################################################################################
# Расширенный Frame, автоматически создающий оконные меню и панели инструментов.
# GuiMakerFrameMenu предназначен для встроенных компонентов (делает меню на основе фреймов).
# GuiMakerWindowMenu предназначен для окон верхнего уровня (делает оконные меню Tk8.0).
# Пример формата дерева структуры см. в коде самотестирования (и PyEdit).
##############################################################################################

import sys
from Tkinter import *                           # классы графических элементов
from types   import *                           # константы типов

class GuiMaker(Frame):
    menuBar    = []                             # значения по умолчанию; изменять
    toolBar    = []                             # при создании подклассов
    helpButton = 1                              # устанавливать в start(), если нужен self

    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        self.pack(expand=YES, fill=BOTH)        # растягиваемый фрейм
        self.start()                            # в подклассе: установить menu/toolBar
        self.makeMenuBar()                      # панель меню строится здесь
        self.makeToolBar()                      # панель инструментов строится здесь
        self.makeWidgets()                      # в подклассе: добавить середину

    def makeMenuBar(self):
        """
        сделать панель меню вверху (меню Tk8.0 внизу)
        expand=no, fill=x, чтобы ширина не менялась
        """
        menubar = Frame(self, relief=RAISED, bd=2)
        menubar.pack(side=TOP, fill=X)

        for (name, key, items) in self.menuBar:
            mbutton  = Menubutton(menubar, text=name, underline=key)
            mbutton.pack(side=LEFT)
            pulldown = Menu(mbutton)
            self.addMenuItems(pulldown, items)
            mbutton.config(menu=pulldown)

        if self.helpButton:
            Button(menubar, text    = 'Help',
                            cursor  = 'gumby',
                            relief  = FLAT,
                            command = self.help).pack(side=RIGHT)

    def addMenuItems(self, menu, items):
        for item in items:                      # просмотр списка вложенных элементов
            if item == 'separator':             # строка: добавить разделитель
                menu.add_separator({})
            elif type(item) == ListType:        # список: отключенных элементов
                for num in item:
                    menu.entryconfig(num, state=DISABLED)
            elif type(item[2]) != ListType:
                menu.add_command(label     = item[0],           # команда:
                                 underline = item[1],           # добавить команду
                                 command   = item[2])           # что вызвать
            else:
                pullover = Menu(menu)
                self.addMenuItems(pullover, item[2])            # подсписок:
```

```
                menu.add_cascade(label     = item[0],          # создать подменю
                                 underline = item[1],          # добавить каскад
                                 menu      = pullover)

    def makeToolBar(self):
        """
        если нужно, сделать внизу панель кнопок
        expand=no, fill=x, чтобы ширина не изменялась
        """
        if self.toolBar:
            toolbar = Frame(self, cursor='hand2', relief=SUNKEN, bd=2)
            toolbar.pack(side=BOTTOM, fill=X)
            for (name, action, where) in self.toolBar:
                Button(toolbar, text=name, command=action).pack(where)

    def makeWidgets(self):
        """
        'средняя' часть создается последней, чтобы меню/панель инструментов
        всегда были вверху/внизу и обрезались последними;
        переопределяйте: упаковывайте середину к любому краю;
        для grid: располагайте среднюю часть в упаковываемом фрейме
        """
        name = Label(self,
                     width=40, height=10,
                     relief=SUNKEN, bg='white',
                     text   = self.__class__.__name__,
                     cursor = 'crosshair')
        name.pack(expand=YES, fill=BOTH, side=TOP)

    def help(self):
        """
        переопределить в подклассе
        """
        from tkMessageBox import showinfo
        showinfo('Help', 'Sorry, no help for ' + self.__class__.__name__)

    def start(self): pass  # переопределить в подклассе

################################################################################
# Для панели меню главного окна Tk 8.0 вместо фрейма
################################################################################

GuiMakerFrameMenu = GuiMaker                  # используется для меню встраиваемых компонентов

class GuiMakerWindowMenu(GuiMaker):           # используется для меню окон верхнего уровня
    def makeMenuBar(self):
        menubar = Menu(self.master)
        self.master.config(menu=menubar)

        for (name, key, items) in self.menuBar:
            pulldown = Menu(menubar)
            self.addMenuItems(pulldown, items)
            menubar.add_cascade(label=name, underline=key, menu=pulldown)

        if self.helpButton:
            if sys.platform[:3] == 'win':
                menubar.add_command(label='Help', command=self.help)
            else:
                pulldown = Menu(menubar)      # для linux необходимо настоящее ниспадающее меню
                pulldown.add_command(label='About', command=self.help)
                menubar.add_cascade(label='Help', menu=pulldown)
```

```
##############################################################################
# Самотестирование при автономном выполнении: 'python guimaker.py'
##############################################################################

if __name__ == '__main__':
    from guimixin import GuiMixin            # встроить метод help

    menuBar = [
        ('File', 0,
            [('Open',  0, lambda:0),         # lambda:0 - пустая операция
             ('Quit',  0, sys.exit)]),       # здесь sys, а не self
        ('Edit', 0,
            [('Cut',   0, lambda:0),
             ('Paste', 0, lambda:0)]) ]
    toolBar = [('Quit', sys.exit, {'side': LEFT})]

    class TestAppFrameMenu(GuiMixin, GuiMakerFrameMenu):
        def start(self):
            self.menuBar = menuBar
            self.toolBar = toolBar
    class TestAppWindowMenu(GuiMixin, GuiMakerWindowMenu):
        def start(self):
            self.menuBar = menuBar
            self.toolBar = toolBar
    class TestAppWindowMenuBasic(GuiMakerWindowMenu):
        def start(self):
            self.menuBar = menuBar
            self.toolBar = toolBar            # help из guimaker, а не из guimixin

    root = Tk()
    TestAppFrameMenu(Toplevel())
    TestAppWindowMenu(Toplevel())
    TestAppWindowMenuBasic(root)
    root.mainloop()
```

Чтобы понять этот модуль, нужно быть знакомым с основами создания меню, изложенными в предыдущей главе. Тогда код ясен: класс `GuiMaker` обходит структуры меню и панели инструментов и попутно создает соответствующие графические элементы. В код самотестирования включен простой пример структур данных, описывающих меню и панели инструментов:

Шаблоны меню

Списки и вложенные подсписки троек (*метка, подчеркивание_для_быстрого_вызова, обработчик*). Если `обработчик` является подсписком, а не функцией или методом, предполагается, что это каскадное подменю.

Шаблоны панелей инструментов

Список троек (*метка, обработчик, параметры_упаковки*). Параметры упаковки записываются как словарь параметров, передаваемых методу графического элемента `pack` (он принимает словари, но можно преобразовать словарь в аргументы – ключевые слова, передав его в качестве третьего аргумента в `apply`).

Протоколы подклассов

Помимо структур меню и панелей инструментов клиенты этого класса могут вмешиваться и изменять реализованные в нем протоколы методов и геометрии:

Атрибуты шаблона

Предполагается, что клиенты этого класса устанавливают атрибуты `menuBar` и `toolBar` в каком-то месте цепочки наследования к моменту завершения метода `start`.

Инициализация

Метод `start` может переопределяться для динамического создания шаблонов меню и панели инструментов (поскольку будет доступен `self`); `start` служит также местом осуществления общей инициализации – конструктор `__init__` класса `GuiMixin` должен быть выполнен, а не переопределен.

Добавление графических элементов

Метод `makeWidgets` может быть переопределен так, чтобы создавать среднюю часть окна – место в приложении между панелями меню и инструментов. По умолчанию `makeWidgets` помещает в середине метку с именем самого узкого класса, но по сути это абстрактный метод и предполагается его конкретизация.

Протокол упаковки

В конкретном методе `makeWidgets` клиенты могут прикреплять графические элементы из средней части к любому краю «self» (`Frame`), так как панели меню и инструментов уже захватили верх и низ контейнера к моменту выполнения `makeWidgets`. Если составляющие среднюю часть элементы упаковываются, она не обязательно должна быть вложенным фреймом. Панели меню и инструментов автоматически упаковываются первыми, чтобы быть обрезанными в последнюю очередь при сжатии окна.

Протокол сетки

В средней части расположение элементов может осуществляться по сетке, если эта сетка помещена во вложенный `Frame`, который упаковывается в родительский `self`. (Помните, что на каждом уровне контейнеров можно применять `grid` или `pack`, но не оба вместе, а `self` является `Frame` с уже упакованными панелями ко времени вызова `makeWidgets`.) Так как `Frame GuiMaker` упаковывает себя в родительском контейнере, по аналогичным причинам его нельзя непосредственно встраивать в контейнер с элементами, располагаемыми по сетке, – для использования его в таком контексте добавьте промежуточный `Frame` с сеткой.

Классы GuiMaker

В ответ на выполнение условий по протоколам и шаблонам `GuiMaker` клиентские подклассы получают `Frame`, который умеет автоматически строить свои меню и панели инструментов по структурам данных шаблона. Если вы смотрели примеры меню из предыдущей главы, то можете понять, что это большое приобретение в смысле уменьшения объема кода. `GuiMaker` также достаточно сообразителен и может экспортировать интерфейсы меню обоих стилей, с которыми мы встречались в предыдущей главе:

- `GuiMakerWindowMenu` реализует меню окон верхнего уровня в стиле Tk 8.0, применяемые для меню, связываемых с самостоятельными программами, и всплывающих окон.
- `GuiMakerFrameMenu` реализует альтернативные меню, основанные на `Frame/Menubutton`, применяемые для меню объектов, встраиваемых в качестве компонентов более крупных GUI.

Оба класса строят панели инструментов, экспортируют те же протоколы и ожидают те же структуры шаблонов; они отличаются только способом обработки шаблонов меню. В действительности один из них является подклассом другого, специализируя метод создания меню – два стиля отличаются только обработкой меню верхнего уровня (`Menu` с каскадами `Menu` вместо `Frame` с `Menubuttons`).

Самотестирование GuiMaker

Как и в GuiMixin, при выполнении примера 9.2 в качестве программы верхнего уровня запускается логика самотестирования, находящаяся в конце файла. На рис. 9.2 показаны получаемые при этом окна. Возникают три окна, представляющие классы TestApp кода самотестирования. У всех трех есть меню и панель инструментов, параметры которых заданы в структурах данных шаблонов, создаваемых кодом самотестирования: выпадающие меню File и Edit, а также кнопка панели инструментов Quit и стандартная кнопка меню Help. На снимке меню File одного из окон оторвано, а меню Edit другого окна раскрыто.

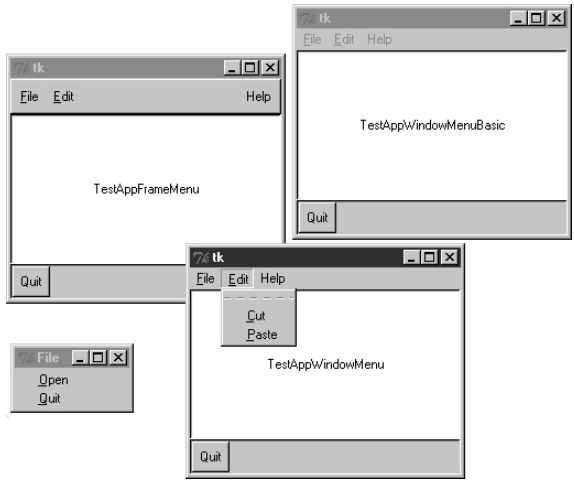

Рис. 9.2. Самотестирование GuiMaker в работе

Связи между суперклассами установлены в коде так, что два из трех окон получают обработчик обратного вызова help от GuiMixin; TestAppWindowMenuBasic получает его от GuiMaker. Обратите внимание на то, что важен порядок, в котором соединяются эти два класса: так как метод quit определен в обоих классах GuiMixin и Frame, класс, от которого мы хотим его получить, нужно указать первым в строке заголовка смешанного класса, поскольку при множественном наследовании поиск производится слева направо. Чтобы выбирались методы GuiMixin, его обычно следует указывать перед суперклассом, производным от реальных графических элементов.

Более практическое применение GuiMaker найдет в таких примерах, как PyEdit далее в этой главе. В следующем модуле показан другой способ использования шаблонов GuiMaker для построения усложненного интерфейса.

BigGui: программа демонстрации клиента

Рассмотрим программу, в которой лучшим образом используются только что написанные нами два класса автоматизации. В модуле примера 9.3 класс Hello является наследником как GuiMixin, так и GuiMaker. GuiMaker обеспечивает связь с графическим элементом Frame и логику построения меню/панели инструментов. GuiMixin обеспечивает дополнительные методы стандартного поведения. В действительности Hello служит образцом еще одного расширения элемента Frame, поскольку является производным от GuiMaker. Чтобы бесплатно получить меню и панель инструментов, он просто следует протоколам, определенным в GuiMaker, – устанавливает атрибуты menuBar и toolBar в методе start и переопределяет makeWidgets, помещая в середину метку.

Пример 9.3. PP2E\Gui\Tools\BigGui\big_gui.py

```
#!/usr/bin/python
###########################################################
# реализация gui - объединяет maker, mixin и данный
###########################################################

import sys, os, string
from Tkinter import *                          # классы графических элементов
from PP2E.Gui.Tools.guimixin import *          # внедряемые методы
from PP2E.Gui.Tools.guimaker import *          # фрейм плюс построение меню/панели инструментов
from find_demo_dir import findDemoDir          # поиск демопрограмм Python

class Hello(GuiMixin, GuiMakerWindowMenu):     # или GuiMakerFrameMenu
    def start(self):
        self.hellos = 0
        self.master.title("GuiMaker Demo")
        self.master.iconname("GuiMaker")

        self.menuBar = [                               # дерево: 3 выпадающих меню
          ('File', 0,                                  # (выпадающее меню)
              [('New...',  0, self.notdone),           # [список элементов меню]
               ('Open...', 0, self.fileOpen),
               ('Quit',    0, self.quit)]              # метка,подчеркивание,действие
          ),

          ('Edit', 0,
              [('Cut',  -1, self.notdone),             # без подчеркивания|действия
               ('Paste',-1, self.notdone),             # lambda:0 тоже можно
               'separator',                            # добавить разделитель
               ('Stuff', -1,
                   [('Clone', -1, self.clone),         # каскадное подменю
                    ('More',  -1, self.more)]
               ),
               ('Delete', -1, lambda:0),
               [5]]                                    # отключить 'delete'
          ),

          ('Play', 0,
              [('Hello',     0, self.greeting),
               ('Popup...',  0, self.dialog),
               ('Demos', 0,
                  [('Hanoi', 0,
                       lambda x=self:
                        x.spawn(findDemoDir() + '\guido\hanoi.py', wait=0)),
                   ('Pong',  0,
                       lambda x=self:
                       x.spawn(findDemoDir() + '\matt\pong-demo-1.py', wait=0)),
                   ('Other...', -1, self.pickDemo)]
               )]
          )]

        self.toolBar = [
          ('Quit',  self.quit,     {'side': RIGHT}),       # добавить 3 кнопки
          ('Hello', self.greeting, {'side': LEFT}),
          ('Popup', self.dialog,   {'side': LEFT, 'expand':YES}) ]

    def makeWidgets(self):                             # переопределить имеющийся по умолчанию
        middle = Label(self, text='Hello maker world!', width=40, height=10,
                       cursor='pencil', bg='white', relief=SUNKEN)
        middle.pack(expand=YES, fill=BOTH)
```

```
    def greeting(self):
        self.hellos = self.hellos + 1
        if self.hellos % 3:
            print "hi"
        else:
            self.infobox("Three", 'HELLO!')    # при каждом третьем нажатии

    def dialog(self):
        button = self.question('OOPS!',
                               'You typed "rm*" ... continue?',
                               'questhead', ('yes', 'no', 'help'))
        [lambda:0, self.quit, self.help][button]()

    def fileOpen(self):
        pick = self.selectOpenFile(file='big_gui.py')
        if pick:
            self.browser(pick)     # просмотр файла исходного кода или иного

    def more(self):
        new = Toplevel()
        Label(new,  text='A new non-modal window').pack()
        Button(new, text='Quit', command=self.quit).pack(side=LEFT)
        Button(new, text='More', command=self.more).pack(side=RIGHT)

    def pickDemo(self):
        pick = self.selectOpenFile(dir=findDemoDir()+'\guido')
        if pick:
            self.spawn(pick, wait=0)    # запустить любую программу Python

if __name__ == '__main__':  Hello().mainloop()   # создать, запустить
```

Этот сценарий размещает довольно большую структуру из меню и панели инструментов, которую мы вскоре увидим. Он также добавляет собственные методы обратного вызова, которые выводят сообщения в `stdout`, показывают средства просмотра текстовых файлов и новые окна и запускают программы. Однако многие обратные вызовы не делают ничего, кроме запуска метода `notDone`, унаследованного от `GuiMixin`: этот код предназначен в основном для демонстрации `GuiMaker` и `GuiMixin`.

Сценарий `big_gui` является почти законченной программой, но не совсем: он нуждается во вспомогательном модуле для поиска готовых демонстрационных программ, поставляемых вместе с полным дистрибутивом исходного кода Python. (Эти демонстрационные программы не входят в собрание примеров данной книги.) Дистрибутив исходного кода Python может быть распакован в любом месте машины.

По этой причине неизвестно, где располагается каталог с демонстрационными программами (и есть ли он вообще). Но вместо того чтобы предложить начинающим изменить исходный код сценария, задав в нем соответствующий путь, выполняется поиск демонстрационного каталога с помощью инструмента `guessLocation` из модуля `Launcher`, с которым мы познакомились в конце главы 4 (см. пример 9.4). Если вы забыли, как это действует, перелистайте книгу назад (хотя прелесть повторного использования кода заключается в том, что часто можно позволить себе забывать).

Пример 9.4. PP2E\Gui\Tools\BigGui\find_demo_dir.py

```
################################################################################
# поиск демонстрационных программ, распространяемых с дистрибутивом исходного кода Python;
# PATH и PP2EHOME здесь не помогут, т. к. эти программы не входят в стандартную установку
# или дерево данной книги
################################################################################
```

```
import os, string, PP2E.Launcher
demoDir  = None
myTryDir = ''

#sourceDir = r'C:\Stuff\Etc\Python-ddj-cd\distributions'
#myTryDir  = sourceDir + r'\Python-1.5.2\Demo\tkinter'

def findDemoDir():
    global demoDir
    if not demoDir:                          # поиск только при первом обращении
        if os.path.exists(myTryDir):         # использовать каталог, заданный в коде, или искать
            demoDir = myTryDir               # сохранить в глобальной переменной
                                             # для следующего обращения
        else:
            print 'Searching for standard demos on your machine...'
            path = PP2E.Launcher.guessLocation('hanoi.py')
            if path:
                demoDir = string.join(string.split(path, os.sep)[:-2], os.sep)
                print 'Using demo dir:', demoDir
    assert demoDir, 'Where is your demo directory?'
    return demoDir
```

При выполнении `big_gui` в качестве программы верхнего уровня создается окно с четырьмя выпадающими меню вверху и панелью инструментов с тремя кнопками внизу, которое показано на рис. 9.3 вместе с некоторыми всплывающими окнами, созданными его обратными вызовами. В меню есть разделители, отключенные элементы и каскадные подменю в полном соответствии с шаблоном `menuBar`.

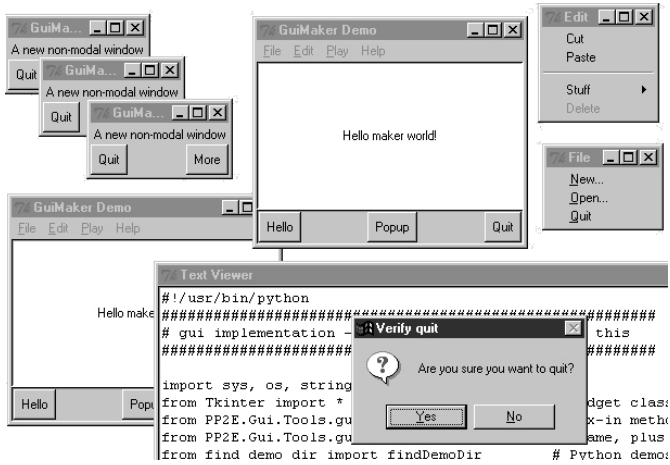

Рис. 9.3. big_gui с несколькими всплывающими окнами

Рис. 9.4 снова показывает окно этого сценария после того, как через выпадающее меню Play были запущены два независимо выполняющиеся экземпляра сценария *hanoi.py*, поставляемого с дистрибутивом исходного кода Python и написанного создателем Python Гвидо ван Россумом. Эта демонстрационная программа содержит простую анимацию решения головоломки «ханойская башня» – классической рекурсивной задачи, часто встречающейся в контрольных опросах по информатике (если вы никогда о ней не слышали, я пощажу вас от изложения деталей).

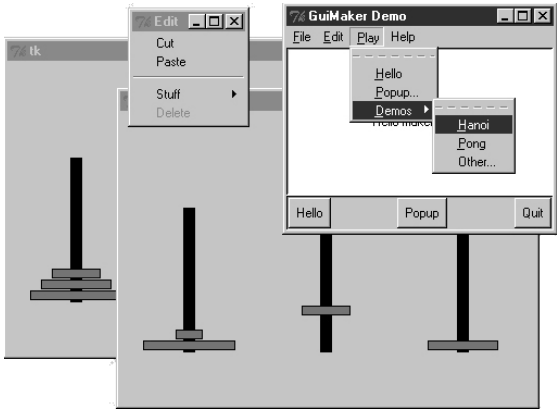

Рис. 9.4. big_gui с порожденными демонстрационными программами hanoi в процессе выполнения

Чтобы обнаружить эту демонстрационную программу, сценарий обыскивает деревья каталогов машины, расположенные в стандартных местах; на моей машине его удалось найти, только прибегнув в последнюю очередь к обходу всего жесткого диска *C:*:

```
C:\...\PP2E\Gui\Tools\BigGui>python big_gui.py
Searching for standard demos on your machine...
Searching for hanoi.py in C:\Program Files\Python
Searching for hanoi.py in C:\PP2ndEd\examples\PP2E\Gui\Tools\BigGui
Searching for hanoi.py in C:\Program Files
Searching for hanoi.py in C:\
Using demo dir: C:\PP2ndEd\cdrom\Python1.5.2\SourceDistribution\Unpacked\Python-
1.5.2\Demo\tkinter
C:\PP2ndEd\cdrom\Python1.5.2\SourceDistribution\Unpacked\Python-1.5.2\Demo\tkint
er\guido\hanoi.py
```

Этот поиск длится примерно 20 секунд на моем 650-мегагерцовом переносном компьютере под Windows, но выполняется только при первом выборе одной из этих демонстрационных программ – после успешного поиска модуль `find_demo_dir` сохраняет имя каталога в глобальной переменной, чтобы его можно было немедленно извлечь при следующем вызове демонстрационной программы. Если нужно выполнить демонстрационные программы из других каталогов (например, программ этой книги в дереве *PP2E*), выберите в меню Play вариант Other, который вызовет стандартный диалог выбора файла, и найдите файл нужной программы.

Наконец, хочу заметить, что `GuiMaker` можно перепроектировать так, чтобы в нем использовались деревья вложенных экземпляров классов, умеющих применять себя к строящемуся дереву графических элементов Tkinter, а не ветвление по типам элементов в структурах данных шаблона. Однако ввиду недостатка места мы отнесем это расширение к области предлагаемых в данном издании самостоятельных упражнений.

ShellGui: добавление GUI к инструментам командной строки

Чтобы явственнее продемонстрировать практическую пользу таких конструкций, как класс `GuiMixin`, требуется более реальное приложение. Вот одно из них. В главе 4 мы познакомились с простыми сценариями для упаковки и распаковки текстовых файлов (см. раздел «Упаковка и распаковка файлов»). Там, если помните, сценарий

packapp.py конкатенировал несколько текстовых файлов в один, а *unpackapp.py* извлекал исходные файлы из объединенного файла.

Эти сценарии запускались с помощью вводимых вручную командных строк, не самых сложных, которые могут встретиться, но достаточно непростых, чтобы легко забыть их. Вместо того чтобы требовать от пользователей таких утилит вводить загадочные команды в оболочке, почему бы для запуска этих программ не создать простой в использовании интерфейс Tkinter? И если мы ставим такую задачу, почему бы вообще не обобщить идею запуска утилит командной строки из GUI для обеспечения поддержки и тех утилит, которые появятся в будущем?

Общий экран инструментов оболочки

Примеры с 9.5 по 9.8 представляют одну конкретную реализацию этих абстрактных размышлений. Так как я хотел, чтобы это был инструмент общего назначения, способный запустить любую программу командной строки, его конструкция разбита на модули, которые становятся все более специфическими для приложений по мере более глубокого вхождения в иерархию программного обеспечения. На самом верху все является общим настолько, насколько это возможно, как показано в примере 9.5.

Пример 9.5. PP2E\Gui\ShellGui\shellgui.py.py

```
#!/usr/local/bin/python
################################################################################
# запуск утилит; использует шаблоны guimaker, стандартный диалог завершения guimixin;
# это лишь библиотека классов: вывести gui можно сценарием mytools;
################################################################################

from Tkinter import *                               # получить графические элементы
from PP2E.Gui.Tools.guimixin import GuiMixin        # получить quit, not done
from PP2E.Gui.Tools.guimaker import *               # построитель меню/панелей инструментов

class ShellGui(GuiMixin, GuiMakerWindowMenu):       # фрейм + конструктор + mixin
    def start(self):                                # для компонентов использовать GuiMaker
        self.setMenuBar()
        self.setToolBar()
        self.master.title("Shell Tools Listbox")
        self.master.iconname("Shell Tools")

    def handleList(self, event):                    # при двойном щелчке по окну списка
        label = self.listbox.get(ACTIVE)            # получить выбранный текст
        self.runCommand(label)                      # и вызвать действие

    def makeWidgets(self):                          # поместить в середину окно списка
        sbar = Scrollbar(self)                      # связать sbar со списком
        list = Listbox(self, bg='white')            # или использовать Tour.ScrolledList
        sbar.config(command=list.yview)
        list.config(yscrollcommand=sbar.set)
        sbar.pack(side=RIGHT, fill=Y)                   # упаковать первым - обрезать последним
        list.pack(side=LEFT, expand=YES, fill=BOTH)     # первым обрезается список
        for (label, action) in self.fetchCommands():    # добавить в окно списка
            list.insert(END, label)                     # и меню/панель инструментов
        list.bind('<Double-1>', self.handleList)        # установить обработчик события
        self.listbox = list

    def forToolBar(self, label):                    # поместить на панель инструментов?
        return 1                                    # по умолчанию - все

    def setToolBar(self):
        self.toolBar = []
```

```
        for (label, action) in self.fetchCommands():
            if self.forToolBar(label):
                self.toolBar.append((label, action, {'side': LEFT}))
        self.toolBar.append(('Quit', self.quit, {'side': RIGHT}))

    def setMenuBar(self):
        toolEntries  = []
        self.menuBar = [
            ('File',  0, [('Quit', -1, self.quit)]),     # имя выпадающего меню
            ('Tools', 0, toolEntries)                    # список элементов меню
            ]                                            # метка, подчеркивание
                                                         # для быстрого вызова, действие
        for (label, action) in self.fetchCommands():
            toolEntries.append((label, -1, action))      # добавить приложения элементами меню

##########################################################################
# делегирование в шаблонные, специфичные для типов, подклассы,
# которые делегируют в подклассы, специфичные для набора утилит
##########################################################################

class ListMenuGui(ShellGui):
    def fetchCommands(self):              # подкласс: установка 'MyMenu'
        return self.myMenu                # список вида (метка, функция)
    def runCommand(self, cmd):
        for (label, action) in self.myMenu:
            if label == cmd: action()

class DictMenuGui(ShellGui):
    def fetchCommands(self):   return self.myMenu.items()
    def runCommand(self, cmd): self.myMenu[cmd]()
```

Класс `ShellGui`, находящийся в этом модуле, умеет с помощью интерфейсов `GuiMaker` и `GuiMixin` строить окно для выбора, которое выводит имена утилит в меню, прокручиваемом списке и панели инструментов. Он также предоставляет переопределяемый метод `forToolBar`, позволяющий подклассам указывать, какие утилиты должны добавляться в панель инструментов, а какие – нет (на панели инструментов может быстро не оказаться свободного места). Однако он умышленно оставлен в неведении относительно имен утилит, которые должны быть выведены в указанных местах, и действий, которые должны быть выполнены при выборе имен утилит.

Вместо этого `ShellGui` использует находящиеся в этом файле подклассы `ListMenuGui` и `DictMenuGui`, чтобы получить список имен утилит через метод `fetchCommands` и управлять действиями по именам через метод `runCommand`. Эти два подкласса в действительности лишь предоставляют интерфейс к специфическим для приложения наборам утилит, представленным как списки или словари; они по-прежнему не знают, какие имена утилит в действительности появляются в GUI. Это также сделано умышленно: так как отображаемые наборы утилит определяются подклассами более низкого уровня, с помощью `ShellGui` можно отображать различные наборы утилит.

Классы наборов утилит, специфичные для приложений

Чтобы получать фактические наборы утилит, нужно спуститься вниз на один уровень. Модуль примера 9.6 определяет подклассы двух специфических по типу классов `ShellGui`, чтобы предоставить имеющиеся утилиты в формате как списка, так и словаря (обычно требуется только один из них, но модуль служит для иллюстрации обоих). Это также тот модуль, который действительно *выполняется* для запуска GUI – модуль `shellgui` служит только библиотекой классов.

Пример 9.6. PP2E\Gui\ShellGui\mytools.py

```
#!/usr/local/bin/python
from shellgui import *                  # специфичные по типу интерфейсы оболочки
from packdlg  import runPackDialog      # диалоги для ввода данных
from unpkdlg  import runUnpackDialog    # оба выполняют классы приложений

class TextPak1(ListMenuGui):
    def __init__(self):
        self.myMenu = [('Pack',    runPackDialog),
                       ('Unpack',  runUnpackDialog),    # простые функции
                       ('Mtool',   self.notdone)]       # метод из guimixin
        ListMenuGui.__init__(self)

    def forToolBar(self, label):
        return label in ['Pack', 'Unpack']

class TextPak2(DictMenuGui):
    def __init__(self):
        self.myMenu = {'Pack':    runPackDialog,        # или использовать входные
                       'Unpack':  runUnpackDialog,    # данные вместо диалогов
                       'Mtool':   self.notdone}
        DictMenuGui.__init__(self)

if __name__ == '__main__':                             # код самопроверки...
    from sys import argv                               # 'menugui.py list|^'
    if len(argv) > 1 and argv[1] == 'list':
        print 'list test'
        TextPak1().mainloop()
    else:
        print 'dict test'
        TextPak2().mainloop()
```

Классы в этом модуле специфичны для конкретных наборов утилит; чтобы выводить другой набор имен утилит, нужно написать новый подкласс и выполнить его. Разделение логики приложения на такие отдельные подклассы и модули повышает возможность повторного использования программного обеспечения.

На рис. 9.5 показано главное окно ShellGui, создаваемое при запуске под Windows сценария mytools с классом структуры меню, основывающемся на словаре, а также оторванные меню, демонстрирующие свое содержание. Меню этого окна и его панель

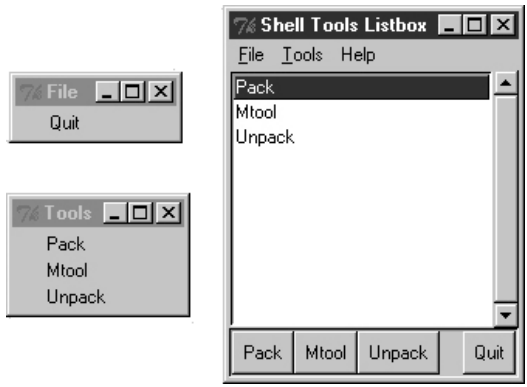

Рис. 9.5. Элементы mytools в окне ShellGui

инструментов построил GuiMaker, а кнопки Quit и Help и выбор пунктов меню запускают методы quit и help, унаследованные от GuiMixin через суперклассы модуля ShellGui. Вы начинаете понимать, почему в этой книге столь часто проповедуется повторное использование кода?

Добавление GUI клиента к командным строкам

Однако действия в обратных вызовах, задаваемые классами предыдущего модуля, обычно должны выполнять нечто GUI-ориентированное. Так как миром исходных сценариев упаковки и распаковки файлов являются текстовые потоки, их нужно заключить в кодовые оболочки, принимающие входные параметры от пользователей, настроенных на работу с GUI.

Модуль примера 9.7 использует технику пользовательских модальных диалогов, изученную нами в главе 7 «Обзор Tkinter, часть 1», чтобы вывести экран для получения параметров сценария упаковки. Его функция runPackDialog служит фактическим обработчиком обратного вызова, запускаемым при выборе имен утилит в главном окне ShellGui.

Пример 9.7. PP2E\Gui\ShellGui\packdlg.py.

```
# добавлены диалоги выбора файлов, проверка пустых полей; можно использовать сетку

import string
from glob import glob                                   # расширение имен файлов
from Tkinter import *                                   # графические элементы для gui
from tkFileDialog import *                              # диалог выбора файла
from PP2E.System.App.Clients.packapp import PackApp     # упаковывающий класс

def runPackDialog():
    s1, s2 = StringVar(), StringVar()          # запуск класса как функции
    PackDialog(s1, s2)                         # всплывающий диалог: устанавливает s1/s2
    output, patterns = s1.get(), s2.get()      # 'ok' или wm-destroy
    if output != "" and patterns != "":
        patterns = string.split(patterns)
        filenames = []
        for sublist in map(glob, patterns):    # расширение вручную
            filenames = filenames + sublist    # автоматически в командной строке Unix
        print 'PackApp:', output, filenames
        app = PackApp(ofile=output)            # запуск с переадресацией вывода
        app.args = filenames                   # переустановка списка аргументов
        app.main()                             # в gui тоже должны показываться сообщения

class PackDialog(Toplevel):
    def __init__(self, target1, target2):
        Toplevel.__init__(self)                # новое окно верхнего уровня
        self.title('Enter Pack Parameters')    # 2 фрейма и кнопка

        f1 = Frame(self)
        l1 = Label(f1,  text='Output file?', relief=RIDGE, width=15)
        e1 = Entry(f1,  relief=SUNKEN)
        b1 = Button(f1, text='browse...')
        f1.pack(fill=X)
        l1.pack(side=LEFT)
        e1.pack(side=LEFT, expand=YES, fill=X)
        b1.pack(side=RIGHT)
        b1.config(command= (lambda x=target1: x.set(askopenfilename())) )

        f2 = Frame(self)
        l2 = Label(f2,  text='Files to pack?', relief=RIDGE, width=15)
```

```
        e2 = Entry(f2,  relief=SUNKEN)
        b2 = Button(f2, text='browse...')
        f2.pack(fill=X)
        l2.pack(side=LEFT)
        e2.pack(side=LEFT, expand=YES, fill=X)
        b2.pack(side=RIGHT)
        b2.config(command=
                 (lambda x=target2: x.set(x.get() +' '+ askopenfilename())) )

        Button(self, text='OK', command=self.destroy).pack()
        e1.config(textvariable=target1)
        e2.config(textvariable=target2)

        self.grab_set()          # сделать модальным: захват мыши,
        self.focus_set()         # фокус клавиатуры, ожидание...
        self.wait_window()       # до закрытия; иначе сразу возврат

if __name__ == '__main__':
    root = Tk()
    Button(root, text='pop', command=runPackDialog).pack(fill=X)
    Button(root, text='bye', command=root.quit).pack(fill=X)
    root.mainloop()
```

При выполнении этого сценария создается форма для ввода, показанная на рис. 9.6. Пользователь может ввести имена входных и выходных файлов с клавиатуры или нажать кнопку «browse...» для открытия стандартных диалогов выбора файлов. Можно вводить и шаблоны имен файлов – вызов `glob.glob` вручную расширяет шаблоны имен файлов и отфильтровывает имена несуществующих файлов. Командная строка Unix осуществляет такое расширение шаблонов автоматически при запуске `PackApp` из оболочки, в отличие от Windows (см. подробности в главе 2 «Системные инструменты»).

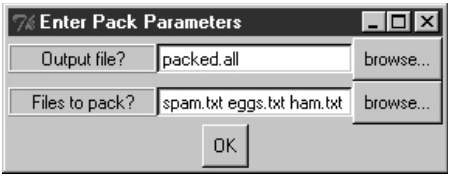

Рис. 9.6. Форма ввода packdlg

Когда форма заполнена и послана с помощью кнопки OK, параметры передаются экземпляру созданного нами в главе 4 класса `PackApp`, чтобы выполнить конкатенацию файлов. GUI для сценария распаковки проще, так как в нем только одно поле ввода – для имени упакованного файла. Сценарий примера 9.8 создает окно формы ввода, показанное на рис. 9.7.

Пример 9.8. PP2E\Gui\ShellGui\unpkdlg.py

```
# добавлен диалог выбора файла, лучше обрабатывает отмену

from Tkinter import *                                   # классы графических элементов
from tkFileDialog import *                              # диалог открытия файла
from PP2E.System.App.Clients.unpackapp import UnpackApp     # класс распаковщика

def runUnpackDialog():
    input = UnpackDialog().input                        # получить входные данные из GUI
    if input != '':                                     # работа с файлами вне gui
        print 'UnpackApp:', input
        app = UnpackApp(ifile=input)                    # запустить с входным файлом
```

```
        app.main()                                          # выполнить класс app

class UnpackDialog(Toplevel):
    def __init__(self):                                     # функция тоже работает
        Toplevel.__init__(self)                             # корневое окно
        self.input = ''                                     # метка и поле ввода
        self.title('Enter Unpack Parameters')
        Label(self, text='input file?', relief=RIDGE, width=11).pack(side=LEFT)
        e = Entry(self, relief=SUNKEN)
        b = Button(self, text='browse...')
        e.bind('<Key-Return>', self.gotit)
        b.config(command=(lambda x=e: x.insert(0, askopenfilename())))
        b.pack(side=RIGHT)
        e.pack(side=LEFT, expand=YES, fill=X)
        self.entry = e
        self.grab_set()                                 # сделать модальным
        self.focus_set()
        self.wait_window()                              # до закрытия по return->gotit
    def gotit(self, event):                             # по клавише return: event.widget==Entry
        self.input = self.entry.get()                   # получить текст, сохранить в self
        self.destroy()                                  # убить окно, но экземпляр сохраняется

if __name__ == "__main__":
    Button(None, text='pop', command=runUnpackDialog).pack()
    mainloop()
```

Кнопка «browse...» на рис. 9.7 показывает диалог выбора файла так же, как форма `packdlg`. Вместо кнопки OK этот диалог привязывает событие нажатия клавиши Enter к закрытию окна и завершению паузы, вызванной ожиданием в модальном состоянии; при этом имя файла передается экземпляру класса `UnpackApp` из главы 4 для выполнения реальной процедуры сканирования файла.

Рис. 9.7. Форма ввода unpkdlg

Все это действует так, как обещано, – благодаря такой доступности утилит командной строки в графическом виде они становятся значительно более привлекательными для пользователей, привычным образом жизни которых является GUI. Все же есть два аспекта такой конструкции, которые представляются требующими усовершенствования.

Во-первых, оба диалога для ввода содержат специальный код для передачи особого внешнего вида, однако кажется возможным существенно упростить их путем импорта стандартного модуля для построения форм. Мы встречались с обобщенным кодом для построения форм в главах 7 и 8 и будем встречаться с ним и далее; см. также указания по обобщению создания форм в модуле *form.py* в главе 10 «Сетевые сценарии».

Во-вторых, в тот момент, когда пользователь передает данные, введенные в той или иной диалоговой форме, след GUI теряется – сообщения `PackApp` и `UnpackApp` по-прежнему поступают в окно консоли `stdout`:

```
C:\...\PP2E\Gui\ShellGui\test>python ..\mytools.py
dict test
PackApp: packed.all ['spam.txt', 'eggs.txt', 'ham.txt']
packing: spam.txt
packing: eggs.txt
packing: ham.txt
```

```
UnpackApp: packed.all
creating: spam.txt
creating: eggs.txt
creating: ham.txt
```

Лучшим решением будет *переадресация* stdout в объект, который выводит полученный текст в окно GUI. О том, как это сделать, можно прочесть в следующем разделе.

GuiStreams: перенаправление потоков данных элементам GUI

Сценарий в примере 9.9 организует отображение входных и выходных потоков во всплывающие окна приложения GUI способом, во многом напоминающим то, как мы поступали со строками в темах, связанных с переадресацией потоков в главе 2. Несмотря на то что этот модуль служит лишь начальным прототипом и сам нуждается в усовершенствовании (например, каждый запрос строки ввода появляется в новом диалоговом окне ввода), он демонстрирует идею в целом.

Объекты этого модуля GuiOutput и GuiInput определяют методы, позволяющие им маскироваться под файлы в любом интерфейсе, предполагающем файл. Как известно из главы 2, это осуществляется с помощью средств обработки стандартных потоков, таких как print, raw_input, и явных вызовов read и write. Функция redirectedGuiFunc из этого модуля с помощью такой «plug-and-play»-совместимости позволяет выполнять любую функцию так, что ее стандартные входной и выходной потоки целиком отображаются во всплывающие окна, а не в окно консоли (или туда, куда эти потоки отображались бы в обычном случае).

Пример 9.9. PP2E\Gui\Tools\guiStreams.py

```
##############################################################################################
# начальная реализация классов, похожих на файлы, используемых для переадресации
# входных и выходных потоков в экраны GUI; входные данные поступают из стандартного
# диалога (единый интерфейс output+input или постоянное поле Entry
# для ввода было бы лучше); кроме того, неверно берутся строки в запросах чтения
# при количестве байтов > len(строки); см. также guiStreamsTools.py;
##############################################################################################
from Tkinter import *
from ScrolledText import ScrolledText
from tkSimpleDialog import askstring

class GuiOutput:
    def __init__(self, parent=None):
        self.text = None
        if parent: self.popupnow(parent)          # вывод сейчас или при первой записи
    def popupnow(self, parent=None):              # сейчас в родительском, Toplevel потом
        if self.text: return
        self.text = ScrolledText(parent or Toplevel())
        self.text.config(font=('courier', 9, 'normal'))
        self.text.pack()
    def write(self, text):
        self.popupnow()
        self.text.insert(END, str(text))
        self.text.see(END)
        self.text.update()
    def writelines(self, lines):                  # в строках уже есть '\n'
        for line in lines: self.write(line)       # или map(self.write, lines)

class GuiInput:
    def __init__(self):
```

```
        self.buff = ''
    def inputLine(self):
        line = askstring('GuiInput', 'Enter input line + <crlf> (cancel=eof)')
        if line == None:
            return ''                       # всплывающий диалог для каждой строки
        else:                               # кнопка cancel означает eof
            return line + '\n'              # иначе добавить маркер конца строки
    def read(self, bytes=None):
        if not self.buff:
            self.buff = self.inputLine()
        if bytes:                                   # читать по счетчику байтов
            text = self.buff[:bytes]                # не захватывает строки
            self.buff = self.buff[bytes:]
        else:
            text = ''                               # читать все до eof
            line = self.buff
            while line:
                text = text + line
                line = self.inputLine()             # до cancel=eof=''
        return text
    def readline(self):
        text = self.buff or self.inputLine()        # эмуляция методов чтения файлов
        self.buff = ''
        return text
    def readlines(self):
        lines = []                                  # читать все строки
        while 1:
            next = self.readline()
            if not next: break
            lines.append(next)
        return lines

def redirectedGuiFunc(func, *pargs, **kargs):
    import sys
    saveStreams = sys.stdin, sys.stdout         # отображение потоков func во всплывающие окна
    sys.stdin   = GuiInput()                    # показывает нужный диалог
    sys.stdout  = GuiOutput()                   # новое окно вывода для каждого вызова
    sys.stderr  = sys.stdout
    result = apply(func, pargs, kargs)          # это блокирующий вызов func
    sys.stdin, sys.stdout = saveStreams
    return result

def redirectedGUIhellCmd(command):
    import os
    input  = os.popen(command, 'r')
    output = GuiOutput()
    def reader(input, output):                  # показать стандартный вывод
        while 1:                                # команды оболочки в новом
            line = input.readline()             # элементе текстового окна
            if not line: break
            output.write(line)
    reader(input, output)

if __name__ == '__main__':
    import string
    def makeUpper():                            # использовать стандартные потоки
        while 1:
            try:
                line = raw_input('Line? ')
```

```
            except:
                break
            print string.upper(line)
        print 'end of file'

    def makeLower(input, output):                # использовать явные файлы
        while 1:
            line = input.readline()
            if not line: break
            output.write(string.lower(line))
        print 'end of file'

    root = Tk()
    Button(root, text='test streams',
           command=lambda: redirectedGuiFunc(makeUpper)).pack(fill=X)
    Button(root, text='test files  ',
           command=lambda: makeLower(GuiInput(), GuiOutput()) ).pack(fill=X)
    Button(root, text='test popen  ',
           command=lambda: redirectedGUIhellCmd('dir *')).pack(fill=X)
    root.mainloop()
```

Согласно этому коду `GuiOutput` либо прикрепляет `ScrolledText` к родительскому контейнеру, либо выводит новое окно верхнего уровня, которое должно служить контейнером, при первом обращении к записи. `GuiInput` выводит новый стандартный диалог ввода каждый раз, когда запрос чтения требует новую входную строку. Ни одна из этих схем не является идеальной во всех ситуациях (ввод лучше было бы отобразить на более долговременный графический элемент), но они доказывают правильность общей идеи. На рис. 9.8 показана картина, создаваемая кодом самотестирования этого сценария, после перехвата вывода команды оболочки *dir* (слева) и двух интерактивных проверок цикла (окно с приглашениями «Line?» и заглавными буквами представляет тест потоков `makeUpper`). Диалог для ввода выведен для нового теста файлов `makeLower`.

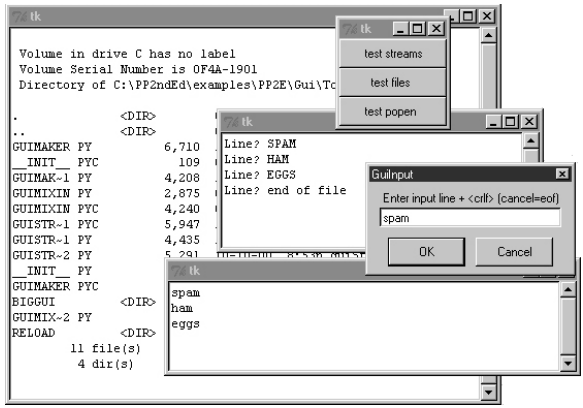

Рис. 9.8. guiStreams направляет потоки во всплывающие окна

Использование переадресации со сценариями упаковки

Теперь, чтобы использовать эти инструменты переадресации для отображения вывода сценария командной строки обратно в GUI, просто выполним вызовы и командные строки через две переадресованные функции этого модуля. Пример 9.10 показывает один из способов создания оболочки для операции упаковки, благодаря которой вывод операции оказывается во всплывающем окне вместо консоли.

Пример 9.10. PP2E\Gui\ShellGui\packdlg-redirect.py

```
# оболочка сценария командной строки в виде утилиты GUI,
# переадресующей его вывод во всплывающее окно

from Tkinter import *
from packdlg import runPackDialog
from PP2E.Gui.Tools.GUIstreams import redirectedGuiFunc

def runPackDialog_Wrapped():
    redirectedGuiFunc(runPackDialog)    # оболочка для всего обработчика обратного вызова

if __name__ == '__main__':
    root = Tk()
    Button(root, text='pop', command=runPackDialog_Wrapped).pack(fill=X)
    root.mainloop()
```

Можете проверить работу этого сценария, вызвав его непосредственно, без привлечения окна `ShellGui`. На рис. 9.9 показано получающееся окно `stdout` после закрытия входного диалога упаковки. Окно появляется, как только сценарий создаст вывод, и предоставляет пользователю несколько более дружественный GUI, чем при отлове сообщений в консоли. Аналогичный код можно написать, чтобы диалог параметров распаковки тоже направлял свой вывод во всплывающее окно.[1] На самом деле с помощью такой техники можно направить во всплывающее окно вывод любого вызова функции или командной строки; как обычно, идея совместимых интерфейсов объектов в значительной мере обусловливает гибкость Python.

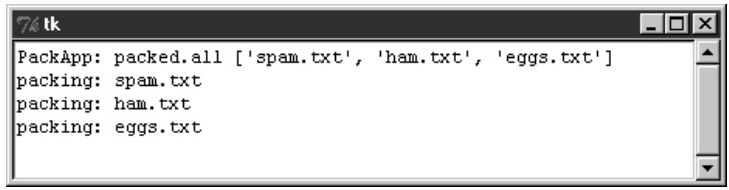

Рис. 9.9. Направление вывода сценария во всплывающее окно GUI

Динамическая перегрузка обработчиков обратного вызова GUI

Стоит рассмотреть здесь еще один последний прием программирования GUI. Функция Python `reload` позволяет динамически изменять и перегружать модули программы, не останавливая ее. Например, можно вызвать текстовый редактор, изменить отдельные части системы во время ее выполнения и увидеть, как проявляются эти изменения сразу после перегрузки измененного модуля.

Это мощная функция, особенно при разработке программ, перезапуск которых мог бы занять длительное время. Программы, которые подключаются к базам данных или сетевым серверам, инициализируют большие объекты или проходят долгую последовательность шагов, чтобы снова запустить обратный вызов, являются первыми кандидатами на использование `reload`. Эта функция может существенно сократить время разработки.

---

[1] Эти два примера довольно уникальны; так как применяемый в них суперкласс `App` сохраняет стандартные потоки в собственных атрибутах во время создания объекта, нужно запускать вызовы переадресующей оболочки GUI как можно раньше, чтобы `App` нашел переадресованные потоки GUI в `sys`, когда будет сохранять их локально. Большинство других сценариев не столь ловки в том, что касается внутренней переадресации потоков.

Однако особенностью GUI является то, что по причине регистрации обработчиков обратного вызова как *ссылок на объекты*, а не имен модулей и объектов, перегрузка функций обработчиков обратного вызова не действует после регистрации обратных вызовов. Операция Python `reload` действует путем изменения содержимого объекта модуля по месту. Однако из-за того что Tkinter непосредственно запоминает указатель на зарегистрированный объект обработчика, ему неведомо о перегрузке модуля, из которого происходит обработчик. Это означает, что Tkinter по-прежнему будет ссылаться на старые объекты модуля, даже если модуль изменен и перегружен.

Это тонкий момент, но нужно только запомнить, что для динамической перегрузки функций обработчиков обратного вызова требуется выполнить особые действия. Необходимо не только явно потребовать перегрузки измененных модулей, но и обеспечить некоторый косвенный слой, маршрутизирующий обратные вызовы от зарегистрированных объектов в модули, чтобы перегрузка возымела эффект.

Например, сценарий примера 9.11 делает дополнительную работу, косвенно переправляя обратные вызовы функциям в явно перегруженном модуле. Обработчики обратного вызова, зарегистрированные Tkinter, являются объектами методов, которые всего лишь осуществляют перегрузку и снова отправляют вызов. Так как доступ к действительным функциям обработчиков обратных вызовов происходит через объект модуля, перегрузка этого модуля приводит к обращению к последним версиям этих функций.

Пример 9.11. PP2E\Gui\Tools\Reload\rad.py

```
from Tkinter import *
import actions              # получить начальные обработчики обратных вызовов

class Hello(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.make_widgets()

    def make_widgets(self):
        Button(self, text='message1', command=self.message1).pack(side=LEFT)
        Button(self, text='message2', command=self.message2).pack(side=RIGHT)

    def message1(self):
        reload(actions)         # необходимо перегрузить перед вызовом модуль actions
        actions.message1()      # теперь нажатие кнопки запускает новую версию

    def message2(self):
        reload(actions)         # изменения actions.py действуют благодаря перегрузке
        actions.message2(self)  # вызвать последнюю версию; передать self

    def method1(self):
        print 'exposed method...'       # вызывается из функции actions

Hello().mainloop()
```

При выполнении этого сценария создается окно с двумя кнопками, запускающими `message1` и `message2`. Пример 9.12 содержит фактический код обработчика обратного вызова. Его функции получают аргумент `self`, который снова возвращает доступ к объекту класса `Hello`, как если бы это были реальные методы. Можно многократно изменять этот файл во время выполнения сценария `rad`; каждое такое действие изменяет поведение GUI при нажатии кнопки.

Пример 9.12. PP2E\Gui\Tools\Reload\actions.py

```
# обработчики обратного вызова: перегружаются при каждом запуске
```

```
def message1():                  # изменить себя
    print 'spamSpamSPAM'         # можно было построить диалог...

def message2(self):
    print 'Ni! Ni!'              # изменить себя
    self.method1()               # доступ к экземпляру 'Hello'...
```

Попробуйте запустить `rad` и редактировать сообщения, которые выводит `actions`, в другом окне. Вы должны увидеть, как при нажатии кнопок GUI в окно консоли `stdout` выводятся новые сообщения. Данный пример намеренно сделан простым, чтобы проиллюстрировать идею, но на практике перегружаемые таким способом действия могут создавать всплывающие диалоги, новые окна верхнего уровня и т. д. Перегрузка кода, создающего такие окна, позволяет динамически изменять их внешний вид.

Есть и другие способы изменить GUI во время его выполнения. Например, в главе 8 мы видели, что внешний вид в любой момент можно изменить, вызвав метод графического элемента `config`, а графические элементы можно динамически добавлять и удалять с экрана такими методами, как `pack_forget` и `pack` (и родственными им для менеджера окон `grid`). Кроме того, передача нового значения параметра `command=action` в метод `config` может на лету установить в качестве обработчика обратного вызова новый объект действия; при наличии соответствующего кода поддержки это может оказаться реальной альтернативой использованной выше обходной схеме повышения эффективности перегрузки в GUI.

Примеры законченных программ

В оставшейся части главы представлен ряд законченных программ GUI как примеров того, чего можно достичь с помощью Python и Tkinter. Так как я уже показывал интерфейсы, применяемые в этих сценариях, этот раздел состоит в основном из снимков экранов, листингов программ и нескольких абзацев, описывающих некоторые наиболее важные аспекты этих программ. Иными словами, это раздел для самостоятельного изучения: читайте исходный код, выполняйте примеры на своем компьютере и обращайтесь к предшествующим главам по вопросам, связанным с приведенным кодом. Многим из этих сценариев сопутствуют не приведенные здесь альтернативные или экспериментальные реализации на прилагаемом к книге CD; см. дополнительные программные примеры на CD.

PyEdit: программа/объект текстового редактора

За последние десятилетия мне пришлось набирать текст во многих программах. Большинство из них были закрытыми системами (мне приходилось довольствоваться теми решениями, которые были сделаны их разработчиками), и многие работали только на одной платформе. Представленная в этом разделе программа PyEdit более удачна в обоих отношениях: она реализует полноценный графический редактор текста примерно в 470 строках переносимого кода Python (включая пробельные символы и комментарии). Несмотря на свой размер, PyEdit оказался достаточно мощным и надежным, чтобы послужить основным инструментом для создания кода большинства примеров, приведенных в этой книге.

PyEdit поддерживает все обычные операции редактирования текста с помощью мыши и клавиатуры: удаление и вставка, поиск и замена, открытие и сохранение и т. д. Но в действительности PyEdit представляет собой нечто большее, чем просто текстовый редактор, – он разработан так, чтобы использоваться как программа и библиотечный компонент, и может выполняться в различном качестве:

Автономный режим

В качестве *автономной* программы текстового редактора, с возможностью передачи имени редактируемого файла в командной строке. В этом режиме PyEdit сходен с другими утилитами редактирования текста (например, Notepad в Windows), но, кроме того, предоставляет развитые функции, например запуск программы Python, код которой редактируется, изменение шрифта и цвета и т. д. Более важно то, что благодаря написанию PyEdit на Python его легко настраивать и переносимым образом выполнять в Windows, X Windows и Macintosh.

Всплывающий режим

Внутри нового всплывающего окна, позволяя программе одновременно выводить произвольное количество экземпляров. Поскольку информация о состоянии хранится в атрибутах экземпляра класса, каждый созданный объект PyEdit действует независимо. В этом режиме и в следующем PyEdit служит библиотечным объектом, используемым другими сценариями, а не готовым приложением.

Встроенный режим

В качестве *прикрепляемого* компонента, предоставляющего графический элемент редактирования текста для других GUI. Будучи прикреплен, PyEdit использует меню, основанное на фрейме, и может отключать некоторые опции меню. Например, PyView (далее в этой главе) использует PyEdit во встроенном режиме в качестве редактора подписей для фотографий, а PyMail (в главе 11) прикрепляет его и получает бесплатный редактор текста электронных писем.

Может показаться, что такое поведение с разными режимами трудно реализовать, но на самом деле режимы PyEdit по большей части являются побочными продуктами написания GUI с применением основанной на классах техники, рассматривавшейся на протяжении последних трех глав.

Запуск PyEdit

У PyEdit есть масса функций, и лучший способ понять, как он работает, – поработать с ним самостоятельно. Его можно запустить как файл *textEditor.pyw* или через панели запуска демонстрационных программ PyDemo и PyGadget, которые описаны в предыдущей главе (сами запускающие программы находятся на верхнем уровне дерева каталогов примеров книги). Получить представление о его интерфейсах можно из рис. 9.10, показывающему вид главного окна по умолчанию после открытия файла с исходным кодом PyEdit.

Главную часть этого окна составляет графический элемент `Text`, и если вы прочли его описание в предыдущей главе, то вам должны быть знакомы операции редактирования текста, выполняемые PyEdit. В нем используются метки, теги и индексы текста и реализованы операции удаления и вставки через системный буфер, благодаря чему PyEdit может обмениваться данными с другими приложениями. Для поддержки перемещения по произвольным файлам с элементом `Text` взаимно связаны вертикальная и горизонтальная полосы прокрутки.

Меню и панели инструментов PyEdit должны показаться вам знакомыми – он строит главное окно, используя минимальный объем кода и надлежащие схемы действий при обрезании и расширении, путем внедрения класса `GuiMaker`, с которым мы познакомились ранее в этой главе. Панель инструментов внизу окна содержит кнопки для сокращенного вызова операций, которым я пользуюсь чаще всего; если ваши вкусы не совпадают с моими, просто измените в исходном коде список для панели инструментов, чтобы в ней были те кнопки, которые вам нужны (в конце концов, это Python). Как обычно, в меню Tkinter для быстрого вызова элементов меню можно ис-

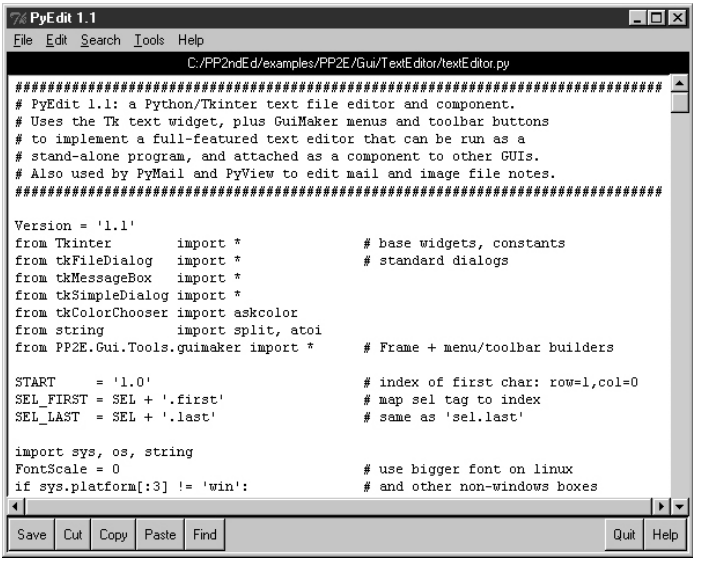

Рис. 9.10. Главное окно PyEdit, редактирующего самого себя

пользовать клавиши сокращенного набора – следует нажать <Alt> и все подчеркнутые клавиши на пути к нужному действию.

PyEdit выводит ряд модальных и немодальных диалогов, стандартных и пользовательских. Рис. 9.11 показывает пользовательский диалог замены и стандартный диалог для вывода статистики файла.

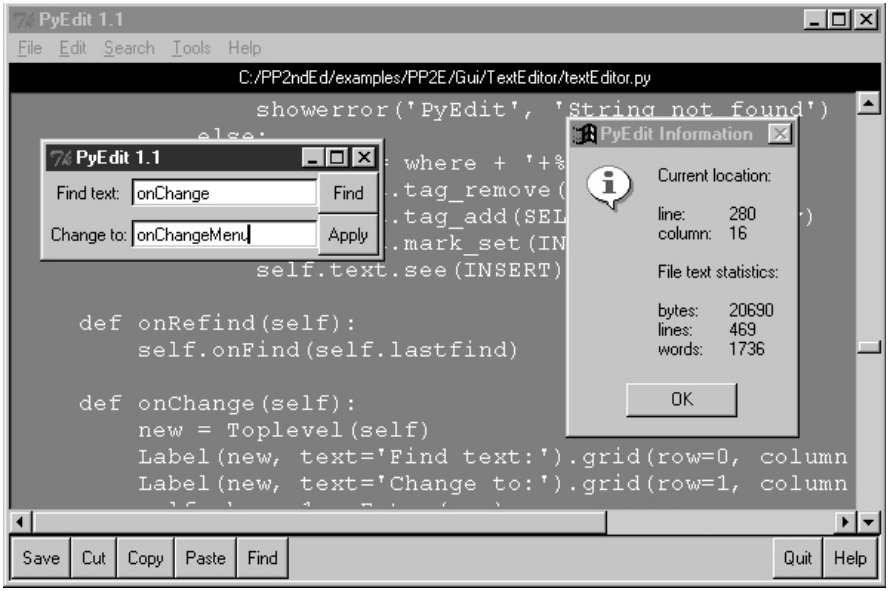

Рис. 9.11. PyEdit с цветом, шрифтами и некоторыми всплывающими диалогами

Здесь для главного окна установлены новые цвета переднего плана и фона (с помощью стандартного диалога выбора цвета), а для текста выбран новый шрифт из имеющегося в сценарии готового списка, который пользователи могут изменять в соответствии со своими предпочтениями (в конце концов, это Python). В PyEdit стандартные диалоги открытия и сохранения файла используют интерфейсы на основе объектов, которые запоминают каталог, выбиравшийся последним, и устраняют необходимость каждый раз заново находить к нему путь.

Одной из специфических особенностей PyEdit является возможность запуска программы Python, код которой редактируется. Это тоже не так сложно, как может показаться: поскольку в Python есть встроенные функции для компиляции и выполнения строк кода, а также запуска программ, PyEdit должен лишь выполнить соответствующие вызовы. В частности, легко написать на Python простенький интерпретатор Python (хотя осуществить обработку многострочных команд несколько сложнее), как показано в примере 9.13.

Пример 9.13. PP2E\Gui\TextEditor\simpleshell.py

```
namespace= {}
while 1:
    try:
        line = raw_input('>>> ')      # только однострочные предложения
    except EOFError:
        break
    else:
        exec line in namespace        # или eval() и print результата
```

В зависимости от настроек пользователя PyEdit либо делает что-нибудь аналогичное для выполнения кода, выбираемого из текстового элемента, либо использует модуль `launchmodes`, который мы написали в конце главы 3, чтобы выполнить файл с кодом как независимую программу. В обеих схемах могут быть использованы различные варианты, которые можно настроить по своему вкусу (в конце концов, это Python). Детали смотрите в методе `onRunCode` или просто отредактируйте и выполните свой собственный код Python.

На рис. 9.12 показаны четыре независимо выполняющиеся экземпляра PyEdit с различными цветовыми схемами, размерами и шрифтами. На этом рисунке видны также два оторванных меню PyEdit (в правом нижнем углу) и всплывающее окно PyEdit help (правый верхний угол). Фоном окон редактирования служат оттенки желтого, голубого, фиолетового и оранжевого; для установки желаемого цвета выберите в меню Tools элемент Pick.

Так как все эти четыре сеанса PyEdit редактируют исходный код Python, их содержимое можно выполнить через пункт Run Code выпадающего меню Tools. Код, запускаемый из файлов, выполняется независимым образом; стандартные потоки кода, выполняемого не из файла (например, полученные из самого графического элемента Text), отображаются в окно консоли сеанса PyEdit. Это никоим образом не является IDE (интегрированной средой разработки) – я добавил это только потому, что мне показалось это полезным. Очень удобно запускать редактируемый код, не разыскивая его в дереве каталогов.

Одно предостережение, прежде чем я натравлю вас на исходный код: в данной версии PyEdit еще нет кнопки «Отмена». Сам я не использую такого режима, и удаление и вставку можно легко отменить после того, как они сделаны (вставьте текст из буфера в прежнее место или удалите вставленный выделенный текст). Добавление обобщенной опции отмены действия составило бы хорошее упражнение для тех, кого это интересует. Возможный подход состоит в том, чтобы породить подклассы `TextEditor` или

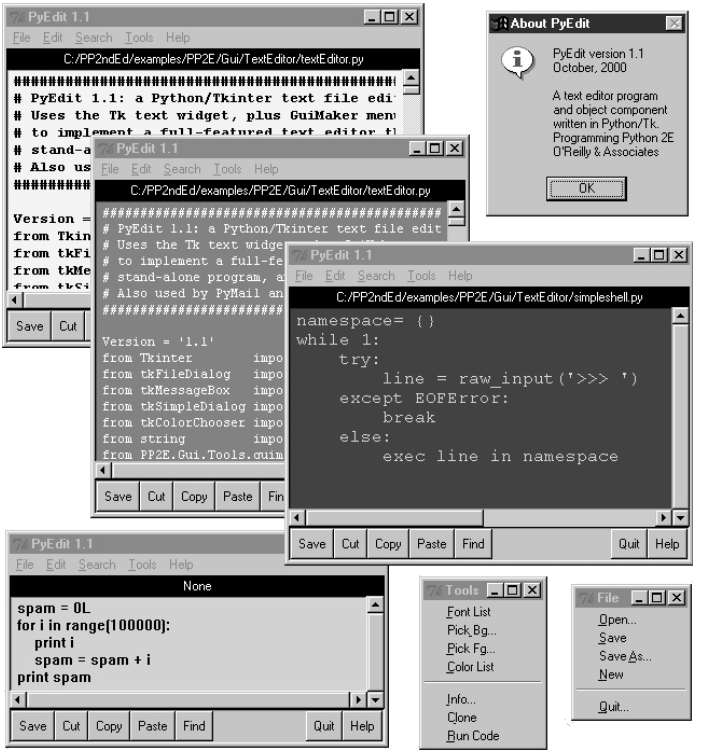

Рис. 9.12. Несколько одновременных сеансов PyEdit

самого класса Tkinter `Text`. Такой класс записывал бы операции с текстом в списке ограниченной длины и запускал вызовы для возвращения из каждой зарегистрированной операции по требованию и при обращении. С его помощью можно было бы также сделать PyEdit умнее, чтобы он знал, когда нужно запрашивать о сохранении файла перед выходом. В результате добавления «отмены» как внешнего класса существующий код PyEdit не потребовалось бы организовывать так, чтобы он отслеживал все, что делается с текстом. В конце концов, это Python.

Исходный код PyEdit

Программа в примере 9.14 состоит всего из двух исходных файлов – *.pyw*, который можно выполнять в Windows, чтобы избежать появления всплывающих окон консоли потоков DOS, и главного *.py*, который можно выполнять или импортировать. Нам требуются оба файла, так как PyEdit одновременно является сценарием и библиотекой, а файлы *.pyw* можно только запускать, но не импортировать (см. главу 2, если вы забыли, что это означает).

Пример 9.14. PP2E\Gui\TextEditor\textEditor.pyw

```
############################################################################
# выполнение PyEdit без окна консоли DOS для os.system в Windows;
# в настоящее время файлы ".pyw" нельзя импортировать как модули;
# если нужно, чтобы файл был программой, запускаемой без окна
# консоли dos в Windows, и мог импортироваться, используйте
```

```
# ".py" для главного файла и импортируйте код .py из ".pyw";
# execfile('textEditor.py') не работает при вызове из другого
# каталога, т. к. текущий каталог - тот, из которого я запущен;
########################################################################

import textEditor             # получить файл .py (или .pyc)
textEditor.main()             # запуск с точки входа верхнего уровня
```

Модуль примера 9.15 представляет собой реализацию PyEdit; главные классы для запуска и встраивания объекта PyEdit находятся в конце файла. Во время экспериментов с PyEdit изучайте этот листинг, чтобы разобраться в функциях и технологиях.

Пример 9.15. PP2E\Gui\TextEditor\textEditor.py

```
##############################################################################
# PyEdit 1.1: текстовый редактор и компонент на Python/Tkinter.
# Использует элемент текст из Tk, меню и кнопки GuiMaker для реализации
# полнофункционального редактора, который выполняется как самостоятельная
# программа или прикрепляется к другим GUI как компонент. Используется также
# в PyMail и PyView для редактирования почты и примечаний к графическому файлу.
##############################################################################

Version = '1.1'
from Tkinter        import *             # графические элементы, константы
from tkFileDialog   import *             # стандартные диалоги
from tkMessageBox   import *
from tkSimpleDialog import *
from tkColorChooser import askcolor
from string         import split, atoi
from PP2E.Gui.Tools.guimaker import *    # фрейм + построители меню/панелей инструментов
START     = '1.0'                        # индекс первого символа: row=1,col=0
SEL_FIRST = SEL + '.first'               # отобразить тег sel в индекс
SEL_LAST  = SEL + '.last'                # то же, что 'sel.last'

import sys, os, string
FontScale = 0                            # в Linux и других не-windows системах
if sys.platform[:3] != 'win':            # использовать более крупный шрифт
  FontScale = 3

class TextEditor:                   # смешать с классом фрейма, имеющим меню/панель инструментов
  startfiledir = '.'
  ftypes = [('All files',    '*'),             # для диалога открытия файла
        ('Text files',   '.txt'),             # настраивать в подклассе или
        ('Python files', '.py')]              # устанавливать в каждом экземпляре

  colors = [{'fg':'black',    'bg':'white'},  # список выбираемых цветов
        {'fg':'yellow',   'bg':'black'},      # первый элемент выбирается по умолчанию
        {'fg':'white',    'bg':'blue'},
        {'fg':'black',    'bg':'beige'},      # переделать по-своему или
        {'fg':'yellow',   'bg':'purple'},     # использовать выбор PickBg/Fg
        {'fg':'black',    'bg':'brown'},
        {'fg':'lightgreen', 'bg':'darkgreen'},
        {'fg':'darkblue',   'bg':'orange'},
        {'fg':'orange',   'bg':'darkblue'}]

  fonts  = [('courier',  9+FontScale, 'normal'), # нейтральные в отношении
        ('courier',   12+FontScale, 'normal'),   # платформы шрифты
        ('courier',   10+FontScale, 'bold'),     # (семейство,размер,стиль)
        ('courier',   10+FontScale, 'italic'),   # или вывод окна списка
        ('times',   10+FontScale, 'normal'),     # сделать крупнее в linux
```

```
      ('helvetica', 10+FontScale, 'normal'),
      ('arial',   10+FontScale, 'normal'),
      ('system',  10+FontScale, 'normal'),
      ('courier',   20+FontScale, 'normal')]

  def __init__(self, loadFirst=''):
    if not isinstance(self, GuiMaker):
      raise TypeError, 'TextEditor needs a GuiMaker mixin'
    self.setFileName(None)
    self.lastfind   = None
    self.openDialog = None
    self.saveDialog = None
    self.text.focus()              # иначе придется щелкнуть
    if loadFirst:
      self.onOpen(loadFirst)

  def start(self):                 # запускается из GuiMaker.__init__
    self.menuBar = [               # настройка меню/панели инструментов
      ('File', 0,
         [('Open...',  0, self.onOpen),
          ('Save',     0, self.onSave),
          ('Save As...', 5, self.onSaveAs),
          ('New',    0, self.onNew),
          'separator',
          ('Quit...',  0, self.onQuit)]
      ),
      ('Edit', 0,
         [('Cut',    0, self.onCut),
          ('Copy',     1, self.onCopy),
          ('Paste',    0, self.onPaste),
          'separator',
          ('Delete',   0, self.onDelete),
          ('Select All', 0, self.onSelectAll)]
      ),
      ('Search', 0,
         [('Goto...',  0, self.onGoto),
          ('Find...',  0, self.onFind),
          ('Refind',   0, self.onRefind),
          ('Change...',  0, self.onChange)]
      ),
      ('Tools', 0,
         [('Font List',   0, self.onFontList),
          ('Pick Bg...',  4, self.onPickBg),
          ('Pick Fg...',  0, self.onPickFg),
          ('Color List',  0, self.onColorList),
         'separator',
          ('Info...',  0, self.onInfo),
          ('Clone',    1, self.onClone),
          ('Run Code',   0, self.onRunCode)]
      )]
    self.toolBar = [
      ('Save',  self.onSave,   {'side': LEFT}),
      ('Cut',   self.onCut,  {'side': LEFT}),
      ('Copy',  self.onCopy,   {'side': LEFT}),
      ('Paste', self.onPaste,  {'side': LEFT}),
      ('Find',  self.onRefind, {'side': LEFT}),
      ('Help',  self.help,   {'side': RIGHT}),
      ('Quit',  self.onQuit,   {'side': RIGHT})]
```

```
def makeWidgets(self):                          # запускается из GuiMaker.__init__
  name = Label(self, bg='black', fg='white')  # добавить под меню и над панелью инструментов
  name.pack(side=TOP, fill=X)                 # меню/панели инструментов пакуются
  vbar  = Scrollbar(self)
  hbar  = Scrollbar(self, orient='horizontal')
  text  = Text(self, padx=5, wrap='none')

  vbar.pack(side=RIGHT,  fill=Y)
  hbar.pack(side=BOTTOM, fill=X)                # упаковать text последним
  text.pack(side=TOP,  fill=BOTH, expand=YES)  # иначе обрежутся полосы прокрутки
  text.config(yscrollcommand=vbar.set)         # вызывать vbar.set при перемещении по тексту
  text.config(xscrollcommand=hbar.set)
  vbar.config(command=text.yview)              # вызывать text.yview при прокрутке
  hbar.config(command=text.xview)              # или hbar['command']=text.xview

  text.config(font=self.fonts[0],
        bg=self.colors[0]['bg'], fg=self.colors[0]['fg'])
  self.text = text
  self.filelabel = name

######################
# Команды меню Edit
######################

def onCopy(self):                           # получить текст, выделенный мышью и т. п.
  if not self.text.tag_ranges(SEL):         # сохранить в общем буфере
    showerror('PyEdit', 'No text selected')
  else:
    text = self.text.get(SEL_FIRST, SEL_LAST)
    self.clipboard_clear()
    self.clipboard_append(text)

def onDelete(self):                         # удалить выделенный текст без сохранения
  if not self.text.tag_ranges(SEL):
    showerror('PyEdit', 'No text selected')
  else:
    self.text.delete(SEL_FIRST, SEL_LAST)

def onCut(self):
  if not self.text.tag_ranges(SEL):
    showerror('PyEdit', 'No text selected')
  else:
    self.onCopy()                           # сохранить и удалить выделенный текст
    self.onDelete()

def onPaste(self):
  try:
    text = self.selection_get(selection='CLIPBOARD')
  except TclError:
    showerror('PyEdit', 'Nothing to paste')
    return
  self.text.insert(INSERT, text)            # добавить в текущем положении курсора
  self.text.tag_remove(SEL, '1.0', END)
  self.text.tag_add(SEL, INSERT+'-%dc' % len(text), INSERT)
  self.text.see(INSERT)                     # выделить, чтобы можно было удалить

def onSelectAll(self):
  self.text.tag_add(SEL, '1.0', END+'-1c')  # выделить весь текст
  self.text.mark_set(INSERT, '1.0')         # переместить точку вставки вверх
  self.text.see(INSERT)                     # прокрутка до верха
```

```
######################
# Команды меню Tools
######################

def onFontList(self):
  self.fonts.append(self.fonts[0])          # выбрать следующий шрифт из списка
  del self.fonts[0]                         # изменить размер текстовой области
  self.text.config(font=self.fonts[0])

def onColorList(self):
  self.colors.append(self.colors[0])        # выбрать следующий цвет из списка
  del self.colors[0]                        # текущий сместить в конец
  self.text.config(fg=self.colors[0]['fg'], bg=self.colors[0]['bg'])

def onPickFg(self):
  self.pickColor('fg')                      # добавлено 10/02/00
def onPickBg(self):                         # выбрать произвольный цвет
  self.pickColor('bg')                      # в стандартном диалоге цвета
def pickColor(self, part):                  # это очень просто
  (triple, hexstr) = askcolor()
  if hexstr:
    apply(self.text.config, (), {part: hexstr})

def onInfo(self):
  text  = self.getAllText()                 # добавлено 5.3.00 за 15 мин.
  bytes = len(text)                         # за words принимается все, что
  lines = len(string.split(text, '\n'))     # разделяется пробельными символами
  words = len(string.split(text))
  index = self.text.index(INSERT)
  where = tuple(string.split(index, '.'))
  showinfo('PyEdit Information',
       'Current location:\n\n' +
       'line:\t%s\ncolumn:\t%s\n\n' % where +
       'File text statistics:\n\n' +
       'bytes:\t%d\nlines:\t%d\nwords:\t%d\n' % (bytes, lines, words))

def onClone(self):
  new = Toplevel()                    # новое окно редактора в том же процессе
  myclass = self.__class__            # объект класса экземпляра (самый нижний)
  myclass(new)                        # прикрепить/выполнить экземпляр моего класса

def onRunCode(self, parallelmode=1):
  """
  выполнение редактируемого кода Python - это не IDE, но удобно;
  пытается работать в каталоге файла, а не cwd (может быть корнем pp2e);
  вводит и добавляет аргументы командной строки для файлов сценариев;
  для кода stdin/out/err = стартовому окну редактора, если оно есть;
  но parallelmode открывает окно dos для i/o;
  """
  from PP2E.launchmodes import System, Start, Fork
  filemode = 0
  thefile  = str(self.getFileName())
  cmdargs  = askstring('PyEdit', 'Commandline arguments?') or ''
  if os.path.exists(thefile):
    filemode = askyesno('PyEdit', 'Run from file?')
  if not filemode:                                  # выполнение как строки текста
    namespace = {'__name__': '__main__'}            # выполнение на верхнем уровне
    sys.argv = [thefile] + string.split(cmdargs)    # можно использовать потоки
    exec self.getAllText() + '\n' in namespace      # исключительные ситуации игнорируются
  elif askyesno('PyEdit', 'Text saved in file?'):
```

```
    mycwd = os.getcwd()                                 # cwd может быть корнем
    os.chdir(os.path.dirname(thefile) or mycwd)         # cd по имени файла
    thecmd  = thefile + ' ' + cmdargs
    if not parallelmode:                                # выполнение как файла
      System(thecmd, thecmd)()                          # блокировать редактор editor
    else:
      if sys.platform[:3] == 'win':                     # породить параллельно
        Start(thecmd, thecmd)()                         # или с помощью os.spawnv
      else:
        Fork(thecmd, thecmd)()                          # породить параллельно
    os.chdir(mycwd)

########################
# Команды меню Search
########################

def onGoto(self):
  line = askinteger('PyEdit', 'Enter line number')
  self.text.update()
  self.text.focus()
  if line is not None:
    maxindex = self.text.index(END+'-1c')
    maxline  = atoi(split(maxindex, '.')[0])
    if line > 0 and line <= maxline:
      self.text.mark_set(INSERT, '%d.0' % line)         # идти на строку
      self.text.tag_remove(SEL, '1.0', END)             # удалить выделенное
      self.text.tag_add(SEL, INSERT, 'insert + 1l')     # выделить строку
      self.text.see(INSERT)                             # прокрутка до строки
    else:
      showerror('PyEdit', 'Bad line number')

def onFind(self, lastkey=None):
  key = lastkey or askstring('PyEdit', 'Enter search string')
  self.text.update()
  self.text.focus()
  self.lastfind = key
  if key:
    where = self.text.search(key, INSERT, END)    # не переносить в начало
    if not where:
      showerror('PyEdit', 'String not found')
    else:
      pastkey = where + '+%dc' % len(key)           # индекс после ключа
      self.text.tag_remove(SEL, '1.0', END)         # удалить выделения
      self.text.tag_add(SEL, where, pastkey)        # выделить ключ
      self.text.mark_set(INSERT, pastkey)           # для следующего поиска
      self.text.see(where)                          # прокрутить экран

def onRefind(self):
  self.onFind(self.lastfind)

def onChange(self):
  new = Toplevel(self)
  Label(new, text='Find text:').grid(row=0, column=0)
  Label(new, text='Change to:').grid(row=1, column=0)
  self.change1 = Entry(new)
  self.change2 = Entry(new)
  self.change1.grid(row=0, column=1, sticky=EW)
  self.change2.grid(row=1, column=1, sticky=EW)
  Button(new, text='Find',
```

```
        command=self.onDoFind).grid(row=0, column=2, sticky=EW)
    Button(new, text='Apply',
        command=self.onDoChange).grid(row=1, column=2, sticky=EW)
    new.columnconfigure(1, weight=1)                # расширяемые поля ввода

  def onDoFind(self):
    self.onFind(self.change1.get())                 # найти то, что в окне замены

  def onDoChange(self):
    if self.text.tag_ranges(SEL):                   # сначала найти
      self.text.delete(SEL_FIRST, SEL_LAST)         # применить замену
      self.text.insert(INSERT, self.change2.get())  # удаляет, если пусто
      self.text.see(INSERT)
      self.onFind(self.change1.get())               # переход к следующему
      self.text.update()                            # заставить обновиться

  ######################
  # Команды меню File
  ######################

  def my_askopenfilename(self):       # объекты помнят каталог/файл последнего результата
    if not self.openDialog:
       self.openDialog = Open(initialdir=self.startfiledir,
                   filetypes=self.ftypes)
    return self.openDialog.show()

  def my_asksaveasfilename(self):     # объекты помнят каталог/файл последнего результата
    if not self.saveDialog:
       self.saveDialog = SaveAs(initialdir=self.startfiledir,
                   filetypes=self.ftypes)
    return self.saveDialog.show()

  def onOpen(self, loadFirst=''):
    doit = self.isEmpty() or askyesno('PyEdit', 'Disgard text?')
    if doit:
      file = loadFirst or self.my_askopenfilename()
      if file:
        try:
          text = open(file, 'r').read()
        except:
          showerror('PyEdit', 'Could not open file ' + file)
        else:
          self.setAllText(text)
          self.setFileName(file)

  def onSave(self):
    self.onSaveAs(self.currfile)     # может оказаться None

  def onSaveAs(self, forcefile=None):
    file = forcefile or self.my_asksaveasfilename()
    if file:
      text = self.getAllText()
      try:
        open(file, 'w').write(text)
      except:
        showerror('PyEdit', 'Could not write file ' + file)
      else:
        self.setFileName(file)       # может быть заново создаваемым

  def onNew(self):
    doit = self.isEmpty() or askyesno('PyEdit', 'Disgard text?')
```

```
    if doit:
      self.setFileName(None)
      self.clearAllText()

  def onQuit(self):
    if askyesno('PyEdit', 'Really quit PyEdit?'):
      self.quit()                  # Frame.quit через GuiMaker

  ####################################
  # Прочие, полезные вне этого класса
  ####################################

  def isEmpty(self):
    return not self.getAllText()

  def getAllText(self):
    return self.text.get('1.0', END+'-1c')  # извлечь текст как строку

  def setAllText(self, text):
    self.text.delete('1.0', END)      # записать текстовую строку в элемент
    self.text.insert(END, text)       # или '1.0'
    self.text.mark_set(INSERT, '1.0') # переместить точку ввода в начало
    self.text.see(INSERT)             # прокрутка туда, где точка вставки

  def clearAllText(self):
    self.text.delete('1.0', END)      # удалить текст элемента

  def getFileName(self):
    return self.currfile

  def setFileName(self, name):
    self.currfile = name  # for save
    self.filelabel.config(text=str(name))

  def help(self):
    showinfo('About PyEdit',
         'PyEdit version %s\nOctober, 2000\n\n'
         'A text editor program\nand object component\n'
         'written in Python/Tk.\nProgramming Python 2E\n'
         "O'Reilly & Associates" % Version)

##################################################################
# готовые к употреблению классы редактора соединить
# с подклассом Frame, который строит меню/панели инструментов
##################################################################

# когда редактор является владельцем окна
class TextEditorMain(TextEditor, GuiMakerWindowMenu):    # добавить конструктор
                                                         # меню/панели инструментов
  def __init__(self, parent=None, loadFirst=''):         # когда заполняет все окно
    GuiMaker.__init__(self, parent)                      # использовать меню главного окна
    TextEditor.__init__(self, loadFirst)                 # у self есть фрейм GuiMaker
    self.master.title('PyEdit ' + Version)               # заголовок в автономной версии
    self.master.iconname('PyEdit')                       # перехват кнопки wm delete
    self.master.protocol('WM_DELETE_WINDOW', self.onQuit)

class TextEditorMainPopup(TextEditor, GuiMakerWindowMenu):
  def __init__(self, parent=None, loadFirst=''):
    self.popup = Toplevel(parent)                        # создать свое окно
    GuiMaker.__init__(self, self.popup)                  # использовать меню главного окна
    TextEditor.__init__(self, loadFirst)
    assert self.master == self.popup
```

```
    self.popup.title('PyEdit ' + Version)
    self.popup.iconname('PyEdit')
  def quit(self):
    self.popup.destroy()                                  # убить только это окно

# если встраивается в другое окно
class TextEditorComponent(TextEditor, GuiMakerFrameMenu):
  def __init__(self, parent=None, loadFirst=''):          # меню на основе Frame
    GuiMaker.__init__(self, parent)                       # все меню, кнопки
    TextEditor.__init__(self, loadFirst)                  # GuiMaker инициируется первым

class TextEditorComponentMinimal(TextEditor, GuiMakerFrameMenu):
  def __init__(self, parent=None, loadFirst='', deleteFile=1):
    self.deleteFile = deleteFile
    GuiMaker.__init__(self, parent)
    TextEditor.__init__(self, loadFirst)
  def start(self):
    TextEditor.start(self)                                # вызов запуска GuiMaker
    for i in range(len(self.toolBar)):                    # убрать quit из панели инструментов
      if self.toolBar[i][0] == 'Quit':                    # убрать пункты меню file
        del self.toolBar[i]; break                        # или просто отключить
    if self.deleteFile:
      for i in range(len(self.menuBar)):
        if self.menuBar[i][0] == 'File':
          del self.menuBar[i]; break
    else:
      for (name, key, items) in self.menuBar:
        if name == 'File':
          items.append([1,2,3,4,6])

# выполнение самостоятельной программы
def testPopup():
  # см. проверку компонент в PyView и PyMail
  root = Tk()
  TextEditorMainPopup(root)
  TextEditorMainPopup(root)
  Button(root, text='More', command=TextEditorMainPopup).pack(fill=X)
  Button(root, text='Quit', command=root.quit).pack(fill=X)
  root.mainloop()

def main():                             # можно ввести с клавиатуры или щелкнуть
  try:                                  # либо ассоциировано имя файла в Windows
    fname = sys.argv[1]                 # arg = необязательное имя файла
  except IndexError:
    fname = None
  TextEditorMain(loadFirst=fname).pack(expand=YES, fill=BOTH)
  mainloop()

if __name__ == '__main__':              # если выполняется как сценарий
  #testPopup()
  main()                                # выполнять .pyw, чтобы не было окна dos
```

PyView: слайд-шоу для графики и заметок

Лучше один раз увидеть, чем потратить тысячу слов, но их понадобится значительно меньше, чтобы вывести картинку с помощью Python. В следующей программе, PyView, реализован простой алгоритм слайд-шоу с помощью переносимого кода Python/Tkinter.

Выполнение PyView

В PyView соединились многие из тем, изучавшихся в последней главе: последовательность показа регулируется событиями `after`, графические объекты выводятся на холсте, размер которого автоматически изменяется, и т. д. В главном окне программы на холсте выводится фотография; пользователь может открыть и просматривать ее непосредственно или запустить режим показа слайдов, в котором выводятся фотографии, случайным образом выбранные из каталога, через регулярные промежутки времени, задаваемые с помощью ползунка.

По умолчанию показ слайдов PyView производится для каталога с графикой с прилагаемого CD (хотя кнопка Open позволяет загружать графику из любых каталогов). Чтобы посмотреть другую группу фотографий, передайте имя каталога в качестве первого аргумента командной строки или измените имя каталога по умолчанию в самом сценарии. Показ слайдов здесь не воспроизвести, но главное окно привести можно. На рис. 9.13 изображено главное окно PyView по умолчанию.

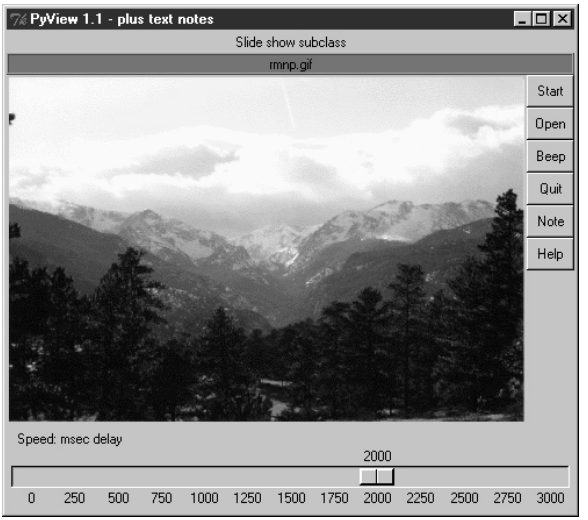

Рис. 9.13. PyView без текста

На рисунке этого не видно, но в действительности на метке вверху окна черным по красному выведен путь к отображаемому файлу. Сейчас переместите ползунок до конца к отметке «0», что задает отсутствие задержки между сменой фотографий, и щелкните по кнопке Start, чтобы начать очень быстрый показ слайдов. Если ваш компьютер обладает хотя бы таким же быстродействием, как мой, то фотографии будут мелькать слишком быстро, чтобы их можно было где-либо применить, кроме как в рекламе, действующей на подсознание. Демонстрируемые фотографии загружаются при начальном запуске, чтобы сохранить ссылки на них (помните, что объекты с изображениями нужно удерживать). Но скорость, с которой в Python большие GIF-файлы выводятся в окно, впечатляет, а то и просто восхищает.

Во время показа слайдов кнопка GUI Start изменяется на Stop (изменяется ее атрибут текста с помощью метода элемента `config`). Рис. 9.14 показывает, что находится на экране после нажатия в подходящий момент кнопки Stop.

Кроме того, у каждой фотографии может быть свой файл «примечаний», который автоматически открывается вместе с изображением. С помощью этой функции можно

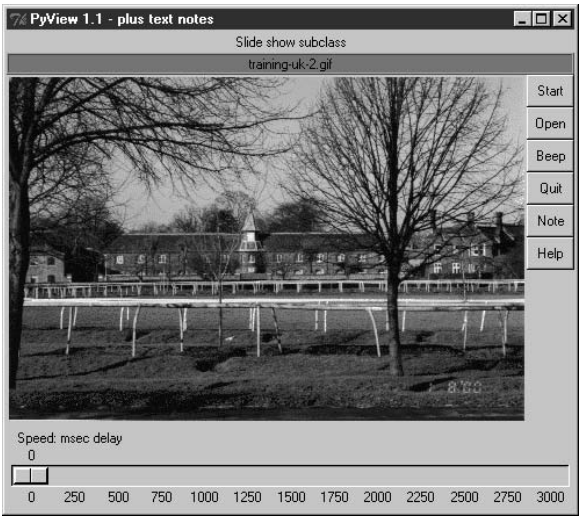

Рис. 9.14. PyView после остановки показа слайдов

записывать основные данные о фотографии. Нажмите кнопку Note, чтобы открыть дополнительный набор графических элементов, с помощью которых можно просматривать и изменять файл примечаний, связанный с просматриваемой в данный момент фотографией. Этот дополнительный набор элементов должен показаться вам знакомым – это текстовый редактор PyEdit из предыдущего раздела, прикрепленный к PyView в качестве средства просмотра и редактирования примечаний к фотографиям. На рис. 9.15 показан PyView вместе с открытым прикрепленным к нему компонентом PyEdit для редактирования примечаний.

В результате получается очень большое окно, которое обычно лучше просматривать развернутым на весь экран. Однако главное, на что нужно обратить внимание, это правый нижний угол экрана над ползунком – там находится прикрепленный объект PyEdit, выполняющий тот же самый код, который приведен в предыдущем разделе. Так как PyEdit реализован в виде класса GUI, подобным же образом можно повторно использовать его в любом GUI, которому требуется интерфейс для редактирования текста. При встраивании таким способом меню PyEdit основываются на фрейме (он не владеет окном в целом), текстовое содержимое запоминается и выбирается непосредственно, а некоторые возможности автономного режима опущены (например, не стало выпадающего меню File).

Средство просмотра примечаний появляется только при нажатии кнопки Note и удаляется при повторном ее нажатии; для того чтобы показать или скрыть фрейм просмотра примечаний, PyView пользуется методами графических элементов `pack` и `pack_forget`, с которыми мы познакомились в конце предыдущей главы. Окно автоматически расширяется, чтобы разместить средство просмотра примечаний, когда оно пакуется и отображается. Можно открыть файл примечаний во всплывающем окне PyEdit, но PyView встраивает редактор, чтобы сохранить прямую зрительную ассоциацию. Мы еще встретимся с таким встраиванием PyEdit внутри другого GUI, когда будем рассматривать PyMail в главе 11.

Предупреждение: в таком виде PyView поддерживает те же графические форматы, что и объект `PhotoImage` библиотеки Tkinter, поэтому по умолчанию он ищет файлы GIF. Улучшить положение можно, установив расширение PIL для просмотра JPEG

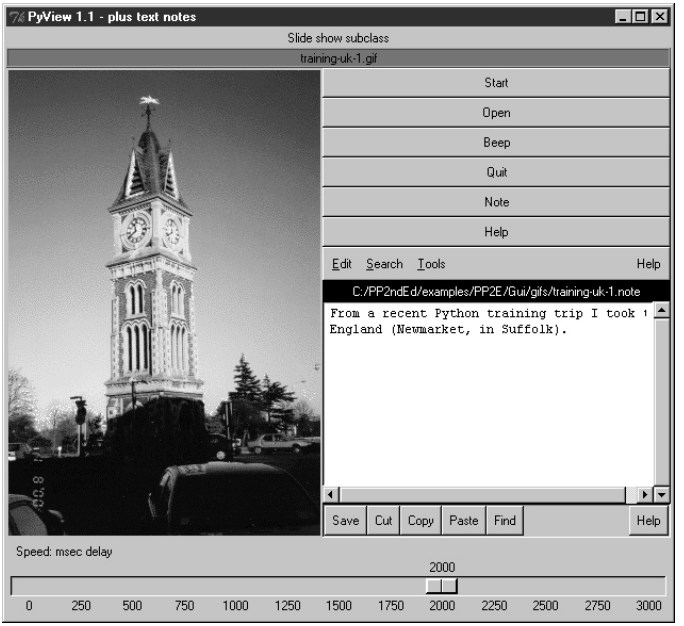

Рис. 9.15. PyView с примечаниями

(и многих других форматов). Поскольку сегодня PIL является факультативным расширением, он не включен в данную версию PyView. Подробнее о PIL и графических форматах сказано в конце главы 7.

Исходный код PyView

Поскольку программа PyView реализовывалась поэтапно, чтобы понять, как она в действительности работает, нужно изучить объединение, состоящее из двух файлов и классов. В одном файле реализован класс, предоставляющий основные функции показа слайдов, а в другом реализован класс, расширяющий исходный и добавляющий новые функции поверх базового поведения. Начнем с класса расширения: пример 9.16 добавляет ряд функций в импортируемый базовый класс показа слайдов – редактирование примечаний, ползунок для задания задержки, метка файла и т. д. Это тот файл, который фактически выполняется для запуска PyView.

Пример 9.16. PP2E\Gui\SlideShow\slideShowPlus.py

```
##############################################################################################
# SlideShowPlus: добавить файлы примечаний с помощью прикрепленного объекта PyEdit, ползунок
# для установки интервала задержки показа и метку с именем отображаемого в данный момент файла;
##############################################################################################

import os, string
from Tkinter import *
from PP2E.Gui.TextEditor.textEditor import *
from slideShow import SlideShow
#from slideShow_threads import SlideShow

class SlideShowPlus(SlideShow):
    def __init__(self, parent, picdir, editclass, msecs=2000):
```

```
        self.msecs = msecs
        self.editclass = editclass
        SlideShow.__init__(self, parent=parent, picdir=picdir, msecs=msecs)
    def makeWidgets(self):
        self.name = Label(self, text='None', bg='red', relief=RIDGE)
        self.name.pack(fill=X)
        SlideShow.makeWidgets(self)
        Button(self, text='Note', command=self.onNote).pack(fill=X)
        Button(self, text='Help', command=self.onHelp).pack(fill=X)
        s = Scale(label='Speed: msec delay', command=self.onScale,
                  from_=0, to=3000, resolution=50, showvalue=YES,
                  length=400, tickinterval=250, orient='horizontal')
        s.pack(side=BOTTOM, fill=X)
        s.set(self.msecs)
        if self.editclass == TextEditorMain:                # теперь сделать редактор
            self.editor = self.editclass(self.master)       # нужен корень для меню
        else:
            self.editor = self.editclass(self)              # встроенный или всплывающий
        self.editor.pack_forget()                           # первоначально скрыть редактор
        self.editorUp = self.image = None
    def onStart(self):
        SlideShow.onStart(self)
        self.config(cursor='watch')
    def onStop(self):
        SlideShow.onStop(self)
        self.config(cursor='hand2')
    def onOpen(self):
        SlideShow.onOpen(self)
        if self.image:
            self.name.config(text=os.path.split(self.image[0])[1])
        self.config(cursor='crosshair')
        self.switchNote()
    def quit(self):
        self.saveNote()
        SlideShow.quit(self)
    def drawNext(self):
        SlideShow.drawNext(self)
        if self.image:
            self.name.config(text=os.path.split(self.image[0])[1])
        self.loadNote()
    def onScale(self, value):
        self.msecs = string.atoi(value)
    def onNote(self):
        if self.editorUp:                   # если редактор уже открыт
            #self.saveNote()                # сохранить текст, скрыть редактор
            self.editor.pack_forget()
            self.editorUp = 0
        else:
            self.editor.pack(side=TOP)      # либо показать/упаковать редактор
            self.editorUp = 1               # и загрузить текст примечания к фото
            self.loadNote()
    def switchNote(self):
        if self.editorUp:
            self.saveNote()         # сохранить примечание к тексту текущего изображения
            self.loadNote()         # загрузить примечание для нового изображения
    def saveNote(self):
        if self.editorUp:
            currfile = self.editor.getFileName()     # или self.editor.onSave()
```

```
            currtext = self.editor.getAllText()      # но текст может быть пуст
            if currfile and currtext:
                try:
                    open(currfile, 'w').write(currtext)
                except:
                    pass # нормально, если запускается не с cd
    def loadNote(self):
        if self.image and self.editorUp:
            root, ext = os.path.splitext(self.image[0])
            notefile  = root + '.note'
            self.editor.setFileName(notefile)
            try:
                self.editor.setAllText(open(notefile).read())
            except:
                self.editor.clearAllText()
    def onHelp(self):
        showinfo('About PyView',
                 'PyView version 1.1\nJuly, 1999\n'
                 'An image slide show\nProgramming Python 2E')

if __name__ == '__main__':
    import sys
    picdir = '../gifs'
    if len(sys.argv) >= 2:
        picdir = sys.argv[1]

    editstyle = TextEditorComponentMinimal
    if len(sys.argv) == 3:
        try:
            editstyle = [TextEditorMain,
                         TextEditorMainPopup,
                         TextEditorComponent,
                         TextEditorComponentMinimal][string.atoi(sys.argv[2])]
        except: pass

    root = Tk()
    root.title('PyView 1.1 - plus text notes')
    Label(root, text="Slide show subclass").pack()
    SlideShowPlus(parent=root, picdir=picdir, editclass=editstyle)
    root.mainloop()
```

Базовые функции, расширяемые `SlideShowPlus`**, находятся в примере 9.17. Это было первоначальной реализацией показа слайдов; она открывает изображения, выводит изображения и организует показ слайдов. Можно запустить ее самостоятельно, но не будет получено таких развитых функций, как примечания и ползунки, добавляемые в подклассе** `SlideShowPlus`**.**

Пример 9.17. PP2E\Gui\SlideShow\slideShow.py

```
##############################################################
# SlideShow: простой показ слайдов на Python/Tkinter;
# базовый набор функций можно расширять в подклассах;
##############################################################

from Tkinter import *
from glob import glob
from tkMessageBox import askyesno
from tkFileDialog import askopenfilename
import random
Width, Height = 450, 450
```

```
imageTypes = [('Gif files', '.gif'),    # для диалога открытия файла
              ('Ppm files', '.ppm'),    # плюс jpg с патчем Tk,
              ('Pgm files', '.pgm'),    # плюс растровые с помощью BitmapImage
              ('All files', '*')]

class SlideShow(Frame):
    def __init__(self, parent=None, picdir='.', msecs=3000, **args):
        Frame.__init__(self, parent, args)
        self.makeWidgets()
        self.pack(expand=YES, fill=BOTH)
        self.opens = picdir
        files = []
        for label, ext in imageTypes[:-1]:
            files = files + glob('%s/*%s' % (picdir, ext))
        self.images = map(lambda x: (x, PhotoImage(file=x)), files)
        self.msecs  = msecs
        self.beep   = 1
        self.drawn  = None
    def makeWidgets(self):
        self.canvas = Canvas(self, bg='white', height=Height, width=Width)
        self.canvas.pack(side=LEFT, fill=BOTH, expand=YES)
        self.onoff = Button(self, text='Start', command=self.onStart)
        self.onoff.pack(fill=X)
        Button(self, text='Open',  command=self.onOpen).pack(fill=X)
        Button(self, text='Beep',  command=self.onBeep).pack(fill=X)
        Button(self, text='Quit',  command=self.onQuit).pack(fill=X)
    def onStart(self):
        self.loop = 1
        self.onoff.config(text='Stop', command=self.onStop)
        self.canvas.config(height=Height, width=Width)
        self.onTimer()
    def onStop(self):
        self.loop = 0
        self.onoff.config(text='Start', command=self.onStart)
    def onOpen(self):
        self.onStop()
        name = askopenfilename(initialdir=self.opens, filetypes=imageTypes)
        if name:
            if self.drawn: self.canvas.delete(self.drawn)
            img = PhotoImage(file=name)
            self.canvas.config(height=img.height(), width=img.width())
            self.drawn = self.canvas.create_image(2, 2, image=img, anchor=NW)
            self.image = name, img
    def onQuit(self):
        self.onStop()
        self.update()
        if askyesno('PyView', 'Really quit now?'):
            self.quit()
    def onBeep(self):
        self.beep = self.beep ^ 1
    def onTimer(self):
        if self.loop:
            self.drawNext()
            self.after(self.msecs, self.onTimer)
    def drawNext(self):
        if self.drawn: self.canvas.delete(self.drawn)
        name, img  = random.choice(self.images)
        self.drawn = self.canvas.create_image(2, 2, image=img, anchor=NW)
```

```
            self.image = name, img
            if self.beep: self.bell()
            self.canvas.update()

    if __name__ == '__main__':
        import sys
        if len(sys.argv) == 2:
            picdir = sys.argv[1]
        else:
            picdir = '../gifs'
        root = Tk()
        root.title('PyView 1.0')
        root.iconname('PyView')
        Label(root, text="Python Slide Show Viewer").pack()
        SlideShow(root, picdir=picdir, bd=3, relief=SUNKEN)
        root.mainloop()
```

Чтобы дать лучшее представление о том, что реализовано этим базовым классом, на рис. 9.16 показано, как он выглядит при самостоятельном выполнении (в действительности это два экземпляра, выполняемые самостоятельно) сценарием slideShow_frames, который находится на прилагаемом CD.

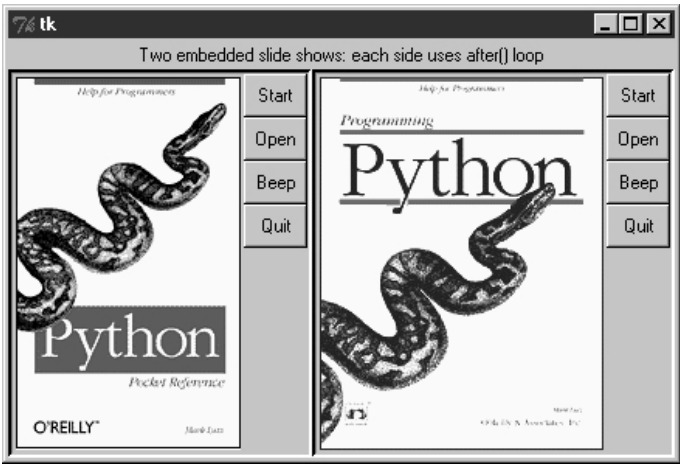

Рис. 9.16. Два прикрепленных объекта SlideShow

Простой сценарий slideShow_frames прикрепляет два экземпляра SlideShow к одному окну, что возможно благодаря записи информации о состоянии в переменные экземпляра класса, а не в глобальные переменные. Сценарий slideShow_toplevels, также имеющийся на CD, прикрепляет два SlideShow к двум всплывающим окнам верхнего уровня. В обоих случаях показ слайдов происходит независимо, но управляется событиями after, генерируемыми одним и тем же циклом событий в одном процессе.

PyDraw: рисование и перемещение графики

В предыдущей главе мы познакомились с простой техникой анимации в Tkinter (см. версии canvasDraw в обзоре). Приводимая здесь программа PyDraw, основываясь на тех же идеях, реализует на Python более богатые функциональные возможности. В ней появились новые режимы рисования trails и scribble, заливка объекта и фона,

встраивание фотографий и другое. Кроме того, реализованы приемы перемещения объектов и анимации – нарисованные объекты можно перемещать по холсту щелчками и перетаскиванием, а любой нарисованный объект можно плавно переместить через экран в место, указанное щелчком мыши.

Выполнение PyDraw

PyDraw, по существу, представляет собой холст Tkinter с многочисленными привязками событий клавиатуры и мыши, которые дают возможность пользователю осуществлять стандартные операции рисования. Ни по каким меркам это нельзя назвать графической программой профессионального уровня, но поразвлечься с ней можно. На самом деле даже нужно, поскольку такой носитель, как книга, не позволяет передать такие вещи, как движущийся объект. Запустите PyDraw из какой-либо панели запуска программ (или прямо файл *movingpics.py* из примера 9.18). Нажмите клавишу <?> и посмотрите подсказку по всем имеющимся командам (или прочтите строку help в листинге).

На рис. 9.17 показано окно PyDraw после изображения на холсте нескольких объектов. Чтобы переместить какой-либо из показанных объектов, щелкните по нему средней кнопкой мыши и перетаскивайте курсором либо щелкните средней кнопкой по объекту, а затем правой кнопкой в том месте, куда вы хотите его переместить. В последнем случае PyDraw осуществляет анимацию, постепенно перемещая объект в указанное место. Попробуйте сделать это с находящейся вверху фотографией создателя Python Гвидо ван Россума и вы увидите известный демонстрационный ролик «Движущийся Гвидо» (не сомневайтесь, у него тоже есть чувство юмора).

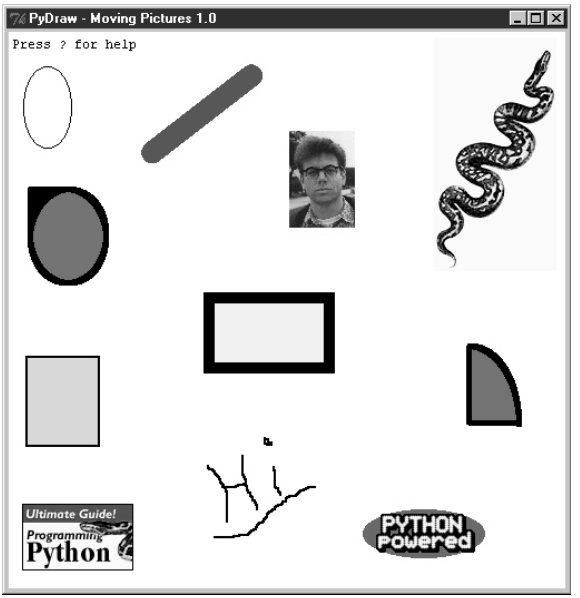

Рис. 9.17. PyDraw с рисованными объектами, готовыми двинуться

Для вставки фотографий нажмите <P>, для рисования фигур нажмите левую кнопку и вытягивайте их. Пользователям Windows: щелчок средней кнопкой обычно равносилен нажатию двух кнопок одновременно, но может потребоваться настроить это в панели управления. Помимо событий мыши есть еще 17 команд клавиш для редакти-

рования рисунков, но о них здесь не будет рассказано. Требуется некоторое время, чтобы освоиться со всеми командами клавиатуры и мыши, после чего вы тоже сможете создавать бессмысленные электронные художества, как на рис. 9.18.

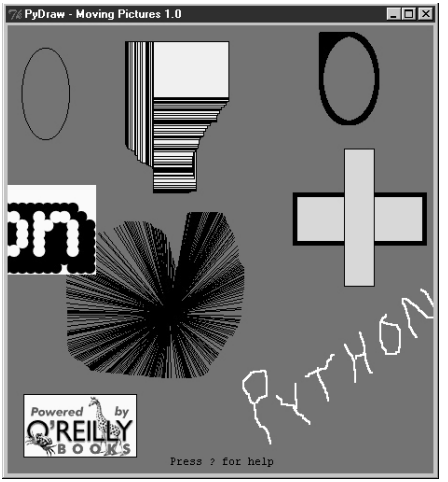

Рис. 9.18. PyDraw после некоторых игр с ним

Исходный код PyDraw

Как и PyEdit, PyDraw размещается в одном файле. За главным модулем, показанным в примере 9.18, приведены два расширения, изменяющие реализацию перемещения.

Пример 9.18. PP2E\Gui\MovingPics\movingpics.py

```
############################################################################
# PyDraw: простая программа рисования на холсте и перемещения/анимации объектов
# перемещение объектов осуществляется в циклах time.sleep, поэтому одновременно
# может происходить только одно такое перемещение; оно плавное и быстрое,
# но смотрите здесь же другие приемы с использованием .after и потоков;
############################################################################

helpstr = """--PyDraw version 1.0--
Mouse commands:
  Left        = Set target spot
  Left+Move   = Draw new object
  Double Left = Clear all objects
  Right       = Move current object
  Middle      = Select closest object
  Middle+Move = Drag current object

Keyboard commands:
  w=Pick border width  c=Pick color
  u=Pick move unit     s=Pick move delay
  o=Draw ovals         r=Draw rectangles
  l=Draw lines         a=Draw arcs
  d=Delete object      1=Raise object
  2=Lower object       f=Fill object
  b=Fill background    p=Add photo
  z=Save postscript    x=Pick pen modes
```

```
  ?=Help               other=clear text
"""

import time, sys
from Tkinter import *
from tkFileDialog import *
from tkMessageBox import *
PicDir = '../gifs'

if sys.platform[:3] == 'win':
    HelpFont = ('courier', 9, 'normal')
else:
    HelpFont = ('courier', 12, 'normal')

pickDelays = [0.01, 0.025, 0.05, 0.10, 0.25, 0.0, 0.001, 0.005]
pickUnits  = [1, 2, 4, 6, 8, 10, 12]
pickWidths = [1, 2, 5, 10, 20]
pickFills  = [None,'white','blue','red','black','yellow','green','purple']
pickPens   = ['elastic', 'scribble', 'trails']

class MovingPics:
    def __init__(self, parent=None):
        canvas = Canvas(parent, width=500, height=500, bg= 'white')
        canvas.pack(expand=YES, fill=BOTH)
        canvas.bind('<ButtonPress-1>',  self.onStart)
        canvas.bind('<B1-Motion>',      self.onGrow)
        canvas.bind('<Double-1>',       self.onClear)
        canvas.bind('<ButtonPress-3>',  self.onMove)
        canvas.bind('<Button-2>',       self.onSelect)
        canvas.bind('<B2-Motion>',      self.onDrag)
        parent.bind('<KeyPress>',       self.onOptions)
        self.createMethod = Canvas.create_oval
        self.canvas = canvas
        self.moving = []
        self.images = []
        self.object = None
        self.where  = None
        self.scribbleMode = 0
        parent.title('PyDraw - Moving Pictures 1.0')
        parent.protocol('WM_DELETE_WINDOW', self.onQuit)
        self.realquit = parent.quit
        self.textInfo = self.canvas.create_text(
                                5, 5, anchor=NW,
                                font=HelpFont,
                                text='Press ? for help')
    def onStart(self, event):
        self.where  = event
        self.object = None
    def onGrow(self, event):
        canvas = event.widget
        if self.object and pickPens[0] == 'elastic':
            canvas.delete(self.object)
        self.object = self.createMethod(canvas,
                               self.where.x, self.where.y,    # начало
                               event.x, event.y,              # конец
                               fill=pickFills[0], width=pickWidths[0])
        if pickPens[0] == 'scribble':
            self.where = event              # следующий раз отсюда
    def onClear(self, event):
```

```
        if self.moving: return              # ok, если движется
        event.widget.delete('all')          # с помощью тега all
        self.images = []
        self.textInfo = self.canvas.create_text(
                                5, 5, anchor=NW,
                                font=HelpFont,
                                text='Press ? for help')
    def plotMoves(self, event):
        diffX  = event.x - self.where.x             # планирование перемещений анимации
        diffY  = event.y - self.where.y             # по горизонтали, затем по вертикали
        reptX  = abs(diffX) / pickUnits[0]          # приращение в каждом шаге, число шагов
        reptY  = abs(diffY) / pickUnits[0]          # от предыдущего до щелчка
        incrX  = pickUnits[0] * ((diffX > 0) or -1)
        incrY  = pickUnits[0] * ((diffY > 0) or -1)
        return incrX, reptX, incrY, reptY
    def onMove(self, event):
        traceEvent('onMove', event, 0)      # переместить текущий объект в место щелчка
        object = self.object                # игнорировать во время движения некоторые операции
        if object and object not in self.moving:
            msecs  = int(pickDelays[0] * 1000)
            parms  = 'Delay=%d msec, Units=%d' % (msecs, pickUnits[0])
            self.setTextInfo(parms)
            self.moving.append(object)
            canvas = event.widget
            incrX, reptX, incrY, reptY = self.plotMoves(event)
            for i in range(reptX):
                canvas.move(object, incrX, 0)
                canvas.update()
                time.sleep(pickDelays[0])
            for i in range(reptY):
                canvas.move(object, 0, incrY)
                canvas.update()                     # update выполняет другие операции
                time.sleep(pickDelays[0])           # sleep до следующего перемещения
            self.moving.remove(object)
            if self.object == object: self.where  = event
    def onSelect(self, event):
        self.where  = event
        self.object = self.canvas.find_closest(event.x, event.y)[0]   # набор
    def onDrag(self, event):
        diffX = event.x - self.where.x              # ok, если объект в движении,
        diffY = event.y - self.where.y              # сбрасывает его с курса
        self.canvas.move(self.object, diffX, diffY)
        self.where = event
    def onOptions(self, event):
        keymap = {
            'w': lambda self: self.changeOption(pickWidths, 'Pen Width'),
            'c': lambda self: self.changeOption(pickFills,  'Color'),
            'u': lambda self: self.changeOption(pickUnits,  'Move Unit'),
            's': lambda self: self.changeOption(pickDelays, 'Move Delay'),
            'x': lambda self: self.changeOption(pickPens,   'Pen Mode'),
            'o': lambda self: self.changeDraw(Canvas.create_oval,      'Oval'),
            'r': lambda self: self.changeDraw(Canvas.create_rectangle, 'Rect'),
            'l': lambda self: self.changeDraw(Canvas.create_line,      'Line'),
            'a': lambda self: self.changeDraw(Canvas.create_arc,       'Arc'),
            'd': MovingPics.deleteObject,
            '1': MovingPics.raiseObject,
            '2': MovingPics.lowerObject,        # если только 1 схема вызова,
            'f': MovingPics.fillObject,         # использовать объекты несвязанных методов,
```

```
        'b': MovingPics.fillBackground,     # иначе лямбде передается self
        'p': MovingPics.addPhotoItem,
        'z': MovingPics.savePostscript,
        '?': MovingPics.help}
    try:
        keymap[event.char](self)
    except KeyError:
        self.setTextInfo('Press ? for help')
def changeDraw(self, method, name):
    self.createMethod = method              # несвязанный метод Canvas
    self.setTextInfo('Draw Object=' + name)
def changeOption(self, list, name):
    list.append(list[0])
    del list[0]
    self.setTextInfo('%s=%s' % (name, list[0]))
def deleteObject(self):
    if self.object != self.textInfo:        # ok, если объект в движении,
        self.canvas.delete(self.object)     # удаляет, но движение продолжается
        self.object = None
def raiseObject(self):
    if self.object:                         # ok, если объект в движении
        self.canvas.tkraise(self.object)    # возбуждает во время движения
def lowerObject(self):
    if self.object:
        self.canvas.lower(self.object)
def fillObject(self):
    if self.object:
        type = self.canvas.type(self.object)
        if type == 'image':
            pass
        elif type == 'text':
            self.canvas.itemconfig(self.object, fill=pickFills[0])
        else:
            self.canvas.itemconfig(self.object,
                           fill=pickFills[0], width=pickWidths[0])
def fillBackground(self):
    self.canvas.config(bg=pickFills[0])
def addPhotoItem(self):
    if not self.where: return
    filetypes=[('Gif files', '.gif'), ('All files', '*')]
    file = askopenfilename(initialdir=PicDir, filetypes=filetypes)
    if file:
        image = PhotoImage(file=file)                       # загрузка образа
        self.images.append(image)                           # сохранить ссылку
        self.object = self.canvas.create_image(             # поместить на холст
                            self.where.x, self.where.y,     # в предыдущую точку
                            image=image, anchor=NW)
def savePostscript(self):
    file = asksaveasfilename()
    if file:
        self.canvas.postscript(file=file)     # сохранить холст в файле
def help(self):
    self.setTextInfo(helpstr)
    #showinfo('PyDraw', helpstr)
def setTextInfo(self, text):
    self.canvas.dchars(self.textInfo, 0, END)
    self.canvas.insert(self.textInfo, 0, text)
    self.canvas.tkraise(self.textInfo)
```

```
    def onQuit(self):
        if self.moving:
            self.setTextInfo("Can't quit while move in progress")
        else:
           self.realquit()  # стандартное wm delete: сообщение об ошибке, если идет перемещение
def traceEvent(label, event, fullTrace=1):
    print label
    if fullTrace:
        for key in dir(event): print key, '=>', getattr(event, key)

if __name__ == '__main__':
    from sys import argv                    # если выполняется этот файл
    if len(argv) == 2: PicDir = argv[1]     # '..' не действует при запуске из другого места
    root = Tk()                             # создать и запустить объект MovingPics
    MovingPics(root)
    root.mainloop()
```

Так же как в примерах `canvasDraw` **из предыдущей главы, можно добавить поддержку одновременного перемещения более чем одного объекта с помощью событий планируемых обратных вызовов** `after` **или потоков. Пример 9.19 показывает подкласс** `MovingPics`**, в котором проведены изменения, необходимые для параллельного перемещения с помощью событий** `after`**. Запустите этот файл непосредственно и увидите разницу; я бы мог попытаться схватить на снимке вид нескольких движущихся объектов, но вряд ли мне это удалось бы.**

Пример 9.19. PP2E\Gui\MovingPics\movingpics_after.py

```
##############################################################################
# PyDraw-after: простая программа рисования и перемещения/анимации объектов
# перемещение объектов осуществляется с помощью планируемых событий .after,
# позволяющих перемещать одновременно несколько объектов, не прибегая к потокам;
# движение выполняется параллельно, но медленнее, чем в версии time.sleep;
# см. также canvasDraw в обзоре: строит и передает сразу весь список incX/incY:
# здесь было бы allmoves = [(incrX, 0)] * reptX) + ([(0, incrY)] * reptY)
##############################################################################

from movingpics import *

class MovingPicsAfter(MovingPics):
    def doMoves(self, delay, objectId, incrX, reptX, incrY, reptY):
        if reptX:
            self.canvas.move(objectId, incrX, 0)
            reptX = reptX - 1
        else:
            self.canvas.move(objectId, 0, incrY)
            reptY = reptY - 1
        if not (reptX or reptY):
            self.moving.remove(objectId)
        else:
            self.canvas.after(delay,
                self.doMoves, delay, objectId, incrX, reptX, incrY, reptY)
    def onMove(self, event):
        traceEvent('onMove', event, 0)
        object = self.object      # переместить текущий объект в точку щелчка
        if object:
            msecs  = int(pickDelays[0] * 1000)
            parms  = 'Delay=%d msec, Units=%d' % (msecs, pickUnits[0])
            self.setTextInfo(parms)
            self.moving.append(object)
```

```
            incrX, reptX, incrY, reptY = self.plotMoves(event)
            self.doMoves(msecs, object, incrX, reptX, incrY, reptY)
            self.where = event

if __name__ == '__main__':
    from sys import argv                        # когда выполняется этот файл
    if len(argv) == 2:
        import movingpics                       # глобальная переменная не этого модуля
        movingpics.PicDir = argv[1]             # и при использовании from* не получится
    root = Tk()                                 # импортировать имена объектов
    MovingPicsAfter(root)
    root.mainloop()
```

Теперь, когда происходит одно или несколько перемещений, можно начать еще одно, щелкнув средней кнопкой по другому объекту и щелкнув правой кнопкой в том месте, куда вы хотите его переместить. Перемещение начинается немедленно, даже если есть другие движущиеся объекты. Запланированные события `after` всех объектов помещаются в одну и ту же очередь цикла событий; Tkinter отправляет их как можно быстрее после срабатывания таймера. Если выполнить этот модуль подкласса непосредственно, то можно заметить, что перемещение не такое плавное и быстрое, как первоначально, но несколько перемещений могут происходить одновременно.

В примере 9.20 показывается, как достичь параллельности с помощью потоков. Такая процедура действует, но как отмечалось в предыдущей главе, обновление GUI в порожденных потоках является, вообще говоря, опасным делом. На моей машине перемещение в этом сценарии с потоками происходит рывками в сравнении с первоначальной версией, что отражает накладные расходы, связанные с переключением интерпретатора (и ЦП) между несколькими потоками.

Пример 9.20. PP2E\Gui\MovingPics\movingpics_threads.py

```
##############################################################################
# перемещение объектов с помощью потоков; должно работать в Windows, если не вызывать
# в потоках canvas.update() (иначе – завершение с фатальными ошибками, нарисованные объекты
# сразу начинают двигаться и т. п.); хотя бы некоторые методы холста в Tkinter должны быть
# безопасными для использования в потоках; менее плавно, чем time.sleep, и опасно
# в целом: лучше всего в потоках обновлять глобальные переменные, не трогая GUI;
##############################################################################

import thread, time, sys, random
from Tkinter import Tk, mainloop
from movingpics import MovingPics, pickUnits, pickDelays

class MovingPicsThreaded(MovingPics):
    def __init__(self, parent=None):
        MovingPics.__init__(self, parent)
        self.mutex = thread.allocate_lock()
        import sys
        #sys.setcheckinterval(0)         # переключение контекста после каждой операции
    def onMove(self, event):            # виртуальной машины не помогает
        object = self.object
        if object and object not in self.moving:
            msecs  = int(pickDelays[0] * 1000)
            parms  = 'Delay=%d msec, Units=%d' % (msecs, pickUnits[0])
            self.setTextInfo(parms)
            #self.mutex.acquire()
            self.moving.append(object)
            #self.mutex.release()
            thread.start_new_thread(self.doMove, (object, event))
```

```
    def doMove(self, object, event):
        canvas = event.widget
        incrX, reptX, incrY, reptY = self.plotMoves(event)
        for i in range(reptX):
            canvas.move(object, incrX, 0)
            # canvas.update()
            time.sleep(pickDelays[0])         # может измениться
        for i in range(reptY):
            canvas.move(object, 0, incrY)
            # canvas.update()                 # update выполняет другие операции
            time.sleep(pickDelays[0])         # спать до следующего перемещения
        #self.mutex.acquire()
        self.moving.remove(object)
        if self.object == object: self.where  = event
        #self.mutex.release()

if __name__ == '__main__':
    root = Tk()
    MovingPicsThreaded(root)
    mainloop()
```

PyClock: графический элемент аналоговых/цифровых часов

Изучая новый интерфейс компьютера, я всегда вначале отыскиваю часы. Я столько времени неотрывно нахожусь за компьютером, что мне совершенно невозможно следить за временем, если не видеть его прямо перед собой на экране (и даже тогда это проблематично). Следующая программа, PyClock, реализует такой графический элемент часов на Python. Она не очень отличается от тех часов, которые вы привыкли видеть в системах X Windows. Но так как ее код написан на Python, то она легко перенастраиваема и переносима между Windows, X Windows и Macintosh, как и весь код этой главы. В дополнение к развитым технологиям GUI этот пример демонстрирует использование модулей Python `math` и `time`.

Краткий урок геометрии

Прежде чем показывать вам PyClock, немного предыстории и признаний. Ну-ка, ответьте: как поставить точки на окружности? Эта задача, а также форматы времени и события оказываются основными при создании графических элементов часов. Чтобы нарисовать аналоговый циферблат часов на элементе холста, необходимо уметь нарисовать круг – сам циферблат состоит из точек окружности, а секундная, минутная и часовая стрелки представляют собой линии, проведенные из центра в точки окружности. Цифровые часы нарисовать проще, но смотреть на них неинтересно.

Теперь признание: начав писать PyClock, я не знал ответа на первый вопрос предыдущего параграфа. Я совершенно забыл математику построения точек окружности (как и большинство профессиональных программистов, у которых я этим интересовался). Бывает. Такие знания, не будучи востребованными в течение нескольких десятилетий, могут попасть в уборку мусора. В конце концов мне удалось смахнуть пыль с нескольких нейронов, длины которых оказалось достаточно, чтобы запрограммировать необходимые для построения действия, но блеснуть умом мне не удалось.

Если вы в таком же положении, то я покажу вам один способ простой записи на Python формул построения точек, но для глубоких занятий геометрией места здесь нет. Прежде чем взяться за более сложную задачу реализации часов, я написал сценарий

plotterGui, приведенный в примере 9.21, чтобы сосредоточиться только на логике построения круга.

Логика круга заключена в функции point – она строит координаты (X,Y) точки окружности по ее относительному номеру, общему количеству точек, помещаемых на окружности, и радиусу окружности (расстоянию между центром окружности и ее точками). Сначала вычисляется угол точки от вершины путем деления 360 на количество рисуемых точек и умножения на номер точки. Напомню, что весь круг составляет 360 градусов (например, если на окружности рисуется 4 точки, то каждая отстоит от предыдущей на 90 градусов, или 360/4). Стандартный модуль Python math предоставляет все необходимые для дальнейшего константы и функции – *pi*, *sine* и *cosine*. В действительности математика тут не слишком непонятная, если достаточно внимательно ее рассмотреть (возможно, вместе со старым учебником геометрии). На прилагаемом к книге CD можно найти альтернативные способы кодирования обработки чисел.[1]

Даже если вас не интересует математика, посмотрите в сценарии на функцию circle. Получив координаты (X,Y) точки окружности, возвращенные point, она чертит линию из центра окружности в точку и маленький прямоугольник вокруг самой точки, что несколько напоминает стрелки и отметки аналоговых часов. На холсте используются теги, чтобы удалять нарисованные объекты перед каждым построением.

Пример 9.21. PP2E\Gui\Clock\plotterGui.py

```
# рисование кругов (как я делал это в старших классах)

import math, sys
from Tkinter import *

def point(tick, range, radius):
    angle = tick * (360.0 / range)
    radiansPerDegree = math.pi / 180
    pointX = int( round( radius * math.sin(angle * radiansPerDegree) ))
    pointY = int( round( radius * math.cos(angle * radiansPerDegree) ))
    return (pointX, pointY)

def circle(points, radius, centerX, centerY, slow=0):
    canvas.delete('lines')
    canvas.delete('points')
    for i in range(points):
        x, y = point(i+1, points, radius-4)
        scaledX, scaledY = (x + centerX), (centerY - y)
        canvas.create_line(centerX, centerY, scaledX, scaledY, tag='lines')
        canvas.create_rectangle(scaledX-2, scaledY-2,
                                scaledX+2, scaledY+2,
                                           fill='red', tag='points')
        if slow: canvas.update()

def plotter():
    circle(scaleVar.get(), (Width / 2), originX, originY, checkVar.get())
```

[1] Если вы достаточно много занимаетесь обработкой чисел и разобрались в этом параграфе, вас, возможно, заинтересует NumPy – расширение Python для численного программирования. Там есть такие объекты, как векторы и сложные математические операции, превращающие Python в инструмент научного программирования. Это расширение эффективно используется многими организациями, в том числе Ливерморской национальной лабораторией. NumPy нужно получить и установить отдельно; смотрите ссылки на сайте Python. Кроме того, в Python есть встроенный тип комплексных чисел для инженерных расчетов; подробности смотрите в руководстве по библиотеке.

```
def makewidgets():
    global canvas, scaleVar, checkVar
    canvas = Canvas(width=Width, height=Width)
    canvas.pack(side=TOP)
    scaleVar = IntVar()
    checkVar = IntVar()
    scale = Scale(label='Points on circle', variable=scaleVar, from_=1, to=360)
    scale.pack(side=LEFT)
    Checkbutton(text='Slow mode', variable=checkVar).pack(side=LEFT)
    Button(text='Plot', command=plotter).pack(side=LEFT, padx=50)

if __name__ == '__main__':
    Width = 500                                          # ширина, высота по умолчанию
    if len(sys.argv) == 2: Width = int(sys.argv[1])  # ширина в командной строке?
    originX = originY = Width / 2                        # то же, что радиус
    makewidgets()                                        # в корне Tk по умолчанию
    mainloop()
```

По умолчанию ширина круга составит 500 пикселов, если не задать ширину в командной строке. Получив число точек на окружности, этот сценарий размечает окружность против часовой стрелки при каждом нажатии Plot, вычерчивая прямые из центра к маленьким прямоугольникам в точках на окружности. Переместите ползунок, чтобы нарисовать другое число точек, и щелкните по флажку, чтобы рисование происходило достаточно медленно и можно было заметить очередность вычерчивания линий и точек (при этом сценарий выполняет `update` для обновления экрана после каждого вычерчивания линии). Рис. 9.19 показывает результат нанесения 120 отметок при установке ширины круга в командной строке, равной 400; если потребовать 60 или 12 точек на окружности, сходство с часовым циферблатом становится более заметным.

Дополнительную помощь могут оказать содержащиеся на CD текстовые версии этого сценария графики, которые выводят координаты точек окружности в поток `stdout`, а

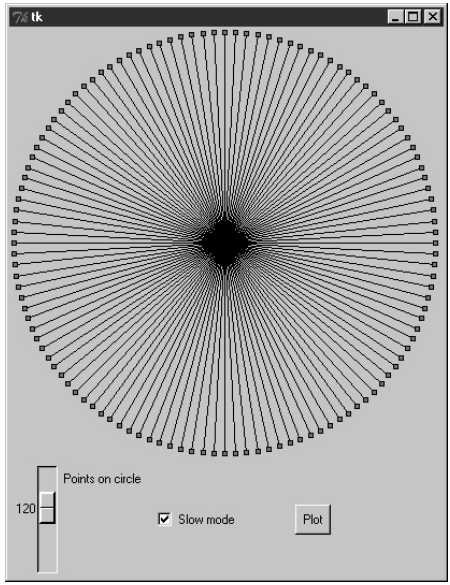

Рис. 9.19. plotterGui в действии

не отображают в GUI. Смотрите сценарии plotterText в каталоге часов. Вот что они показывают для 4 и 12 точек на окружности шириной 400 точек; формат вывода:

```
pointnumber : angle = (Xcoordinate, Ycoordinate)
```

Предполагается, что центр круга имеет координаты (0,0):

```
----------
1 : 90.0 = (200, 0)
2 : 180.0 = (0, -200)
3 : 270.0 = (-200, 0)
4 : 360.0 = (0, 200)
----------
1 : 30.0 = (100, 173)
2 : 60.0 = (173, 100)
3 : 90.0 = (200, 0)
4 : 120.0 = (173, -100)
5 : 150.0 = (100, -173)
6 : 180.0 = (0, -200)
7 : 210.0 = (-100, -173)
8 : 240.0 = (-173, -100)
9 : 270.0 = (-200, 0)
10 : 300.0 = (-173, 100)
11 : 330.0 = (-100, 173)
12 : 360.0 = (0, 200)
----------
```

Чтобы понять, как эти точки отображаются на холст, запомните сначала, что ширина и высота круга всегда одинаковы – радиус × 2. Поскольку координаты холста Tkinter (X,Y) начинаются с (0,0) в левом верхнем углу, построитель GUI должен сместить центр окружности в точку с координатами (ширина/2, ширина/2) – начальную точку, из которой вычерчиваются прямые. Например, в круге 400 на 400 центр холста будет в (200,200). Прямая в точку с углом 90 градусов с правой стороны окружности проходит из (200,200) в (400,200) – результат добавления координат точки (200,0), полученной для данного радиуса и угла. Линия вниз под углом 180 градусов проходит из (200,200) в (200,400) после учета выведенной точки (0,–200).

Этот алгоритм построения точек, применяемый в plotterGui, а также несколько констант масштабирования лежат в сердце аналогового изображения PyClock. Если все же вам кажется, что это слишком сложно, предлагаю сначала сосредоточиться на реализации *цифрового* экрана сценария PyClock. Аналоговые геометрические построения в действительности лишь расширяют механизм отсчета времени, использующийся в обоих режимах отображения. В действительности сами часы имеют структуру общего объекта Frame, одинаковым образом посылающего *встроенным* объектам цифрового и аналогового отображения события изменения времени и размеров. Аналоговый дисплей – это прикрепленный Canvas, умеющий рисовать окружности, а цифровой объект – просто прикрепленный Frame, метки которого показывают время.

Выполнение PyClock

За исключением части, касающейся построения окружностей, код PyClock простой. Он рисует циферблат для показа текущего времени и с помощью методов after будит себя 10 раз в секунду, проверяя, не перевалило ли системное время на следующую секунду. Если да, то перерисовываются секундная, минутная и часовая стрелки, чтобы показать новое время (либо изменяется текст меток цифрового дисплея). На языке создания GUI это означает, что аналоговое изображение выводится на холсте, перерисовывается при изменении размеров окна и при запросе изменяется на цифровой формат.

В PyClock также применяется стандартный модуль Python `time`, чтобы получать и преобразовывать системные данные времени, необходимые для часов. Если описать кратко, то метод `onTimer` получает системное время через `time.time`, встроенное средство, возвращающее число с плавающей точкой, выражающее количество секунд, прошедших с начала «эпохи» – точки начала отсчета времени на вашем компьютере. Затем это время преобразуется в набор, состоящий из значений часов, минут и секунд, с помощью вызова `time.localtime`; дополнительные подробности, относящиеся к работе с временем, можно найти в самом сценарии и руководстве по библиотеке Python.

Проверка системного времени 10 раз в секунду может показаться чрезмерной, но она гарантирует перемещение секундной стрелки вовремя и без рывков и скачков (события `after` синхронизируются не очень точно) и не влечет существенного отвлечения мощности ЦП на тех машинах, которыми я пользуюсь.[1] В Linux и в Windows PyClock незначительно расходует ресурсы процессора – в основном при обновлении экрана в аналоговом режиме, а не в событиях `after`. Чтобы минимизировать обновления экрана, PyClock перерисовывает часовые стрелки только при скачках секунд; отметки на циферблате перерисовываются только при начальном запуске и изменении размеров окна. На рис. 9.20 показан начальный формат вывода PyClock по умолчанию, когда *clock.py* запускается непосредственно.

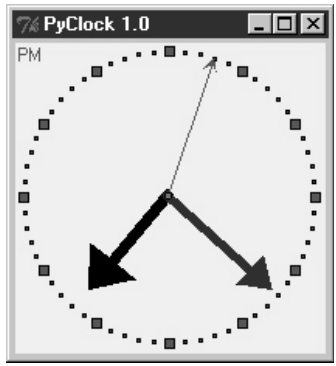

Рис. 9.20. Аналоговый экран PyClock по умолчанию

Линии, представляющие стрелки часов, действительно имеют форму стрелок, что определяется параметрами объекта линии `arrow` и `arrowshape`. Параметр `arrow` может принимать значения «first», «last», «none» или «both»; параметр `arrowshape` задается как набор чисел, задающих длину стрелки, касающейся линии, ее общую длину и ее ширину.

Как и PyView, PyClock динамически удаляет и перерисовывает части изображения по требованию (то есть в ответ на связанные события) с помощью методов `pack_forget` и `pack`. Щелчок по часам левой кнопкой мыши изменяет формат вывода на цифровой путем удаления графических элементов аналогового режима и вывода цифрового интерфейса; в результате получается более простой интерфейс, показанный на рис. 9.21.

Такая цифровая форма вывода полезна, если вы хотите сберечь драгоценное место на экране и уменьшить использование ЦП (расходы на обновление экрана очень малы).

---

[1] Что касается производительности, то я запускал по нескольку часов на всех проверяемых машинах – от Pentium III 650 МГц до «старого» Pentium I 200 МГц – и не обнаружил ухудшения работы каких-либо из выполнявшихся часов. Например, сценарий PyDemos запускает шесть часов, выполняющихся в одном процессе, и во всех них обновление времени происходит плавно. Возможно, так же обстоит дело и на более старых машинах, но мои настолько покрылись пылью, что полезных измерений на них уже не произвести.

Щелчок по часам левой кнопкой снова переводит их в аналоговый режим отображения. При запуске сценария строятся оба отображения – аналоговое и цифровое, но в каждый данный момент упаковано только одно из них.

Рис. 9.21. PyClock стал цифровым

Щелчок по часам правой кнопкой мыши в любом из режимов отображения вызывает появление или исчезновение прикрепленной метки, показывающей текущую дату в простом текстовом формате. На рис. 9.22 показан PyClock, выполняющийся в аналоговом режиме, с меткой даты и размещенным по центру фотографическим изображением.

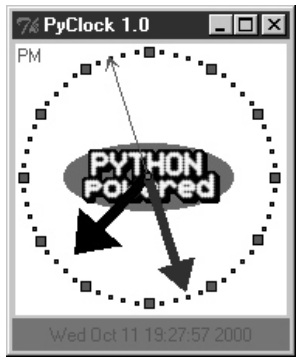

Рис. 9.22. Расширенное изображение PyClock с графикой

Изображение в центре добавлено путем передачи конструктору объекта PyClock объекта конфигурации с надлежащими установками. В действительности почти все особенности изображения можно настроить с помощью атрибутов объектов конфигурации PyClock – цвет стрелок, цвет меток, центральное изображение и начальный размер.

Так как в аналоговом режиме PyClock фигуры вручную рисуются на холсте, необходимо самостоятельно обрабатывать события *изменения размеров* окна: когда окно уменьшается или увеличивается, нужно перерисовывать циферблат часов в соответствии с новыми размерами окна. Для перехвата изменений размеров окна сценарий регистрирует событие `<Configure>` с помощью `bind`; удивительно, но это не событие менеджера верхнего окна, как для кнопки close. Если расширить PyClock, то циферблат увеличится вместе с окном, – попробуйте расширить, сжать и развернуть окно часов во весь экран на своем компьютере. Так как циферблат строится в квадратной системе координат, PyClock всегда расширяется в равном отношении по вертикали и горизонтали; если просто сделать окно только шире или выше, циферблат не изменится.

Наконец, подобно PyEdit, можно запускать PyClock автономно или прикрепленным и встроенным в другой GUI, в котором требуется вывести текущее время. Для облегчения запуска сконфигурированных некоторым образом часов существует вспомогательный модуль `clockStyles`, предоставляющий ряд объектов конфигурации, которые можно импортировать, расширять, создавая подклассы, и передавать в конструктор часов. На рис. 9.23 показано несколько заранее подготовленных стилей и размеров часов в действии, ведущих синхронный отсчет времени.

Во всех этих часах 10 раз в секунду проверяется изменение системного времени с использованием событий `after`. При выполнении в качестве окон верхнего уровня в од-

ном и том же процессе все они получают событие таймера из одного и того же цикла событий. При запуске в качестве независимых программ у каждой из них имеется собственный цикл событий. В том и другом случае их секундные стрелки раз в секунду дружно перемещаются.

Рис. 9.23. Несколько готовых стилей часов (фото Гвидо перепечатано с разрешения «Dr. Dobb's Journal»)

Исходный код PyClock

Весь исходный код PyClock располагается в одном файле, за исключением объектов предварительно закодированных стилей конфигурации. Если посмотреть в конец кода примера 9.22, то можно заметить, что объект часов можно создать либо передав объект конфигурации, либо задав параметры конфигурации аргументами командной строки (и тогда сценарий просто сам построит объект конфигурации). Вообще говоря, можно непосредственно выполнить этот файл для запуска часов, импортировать его и создать объекты с использованием объектов конфигурации, чтобы часы выглядели более индивидуально, или импортировать и прикрепить его объекты к другому GUI. Например, PyGadgets выполняет этот файл с параметрами командной строки для индивидуальной настройки внешнего вида.

Пример 9.22. PP2E\Gui\Clock\clock.py

```
##############################################################################
# PyClock: GUI часов с аналоговым и числовым режимами отображения,
# всплывающей меткой даты, графикой циферблата, изменением размеров и т. д.
# может выполняться самостоятельно или встроенным (прикрепленным) в другой GUI.
##############################################################################
```

```
from Tkinter import *
import math, time, string

##############################################################################
# Дополнительные конфигурирующие классы
##############################################################################

class ClockConfig:
    # defaults--override in instance or subclass
    size = 200                                      # ширина=высота
    bg, fg = 'beige', 'brown'                       # цвета циферблата, меток
    hh, mh, sh, cog = 'black', 'navy', 'blue', 'red'  # стрелок, центра
    picture = None                                  # файл картинки для циферблата

class PhotoClockConfig(ClockConfig):
    # пример конфигурации
    size    = 320
    picture = '../gifs/ora-pp.gif'
    bg, hh, mh = 'white', 'blue', 'orange'

##############################################################################
# Объект цифрового отображения
##############################################################################

class DigitalDisplay(Frame):
    def __init__(self, parent, cfg):
        Frame.__init__(self, parent)
        self.hour = Label(self)
        self.mins = Label(self)
        self.secs = Label(self)
        self.ampm = Label(self)
        for label in self.hour, self.mins, self.secs, self.ampm:
            label.config(bd=4, relief=SUNKEN, bg=cfg.bg, fg=cfg.fg)
            label.pack(side=LEFT)

    def onUpdate(self, hour, mins, secs, ampm, cfg):
        mins = string.zfill(str(mins), 2)
        self.hour.config(text=str(hour), width=4)
        self.mins.config(text=str(mins), width=4)
        self.secs.config(text=str(secs), width=4)
        self.ampm.config(text=str(ampm), width=4)

    def onResize(self, newWidth, newHeight, cfg):
        pass  # перерисовывать нечего


##############################################################################
# Объект аналогового отображения
##############################################################################

class AnalogDisplay(Canvas):
    def __init__(self, parent, cfg):
        Canvas.__init__(self, parent,
                        width=cfg.size, height=cfg.size, bg=cfg.bg)
        self.drawClockface(cfg)
        self.hourHand = self.minsHand = self.secsHand = self.cog = None

    def drawClockface(self, cfg):                       # при запуске и изменении размеров
        if cfg.picture:                                 # рисовать овал, картинку
            try:
                self.image = PhotoImage(file=cfg.picture)          # фон
```

```
            except:
                self.image = BitmapImage(file=cfg.picture)         # сохранить ссылку
            imgx = (cfg.size - self.image.width())  / 2            # центрировать
            imgy = (cfg.size - self.image.height()) / 2
            self.create_image(imgx+1, imgy+1,  anchor=NW, image=self.image)
        originX =  originY = radius = cfg.size/2
        for i in range(60):
            x, y = self.point(i, 60, radius-6, originX, originY)
            self.create_rectangle(x-1, y-1, x+1, y+1, fill=cfg.fg)  # минуты
        for i in range(12):
            x, y = self.point(i, 12, radius-6, originX, originY)
            self.create_rectangle(x-3, y-3, x+3, y+3, fill=cfg.fg)  # часы
        self.ampm = self.create_text(3, 3, anchor=NW, fill=cfg.fg)

    def point(self, tick, units, radius, originX, originY):
        angle = tick * (360.0 / units)
        radiansPerDegree = math.pi / 180
        pointX = int( round( radius * math.sin(angle * radiansPerDegree) ))
        pointY = int( round( radius * math.cos(angle * radiansPerDegree) ))
        return (pointX + originX+1), (originY+1 - pointY)

    def onUpdate(self, hour, mins, secs, ampm, cfg):        # по обратному вызову таймера
        if self.cog:                                        # перерисовать стрелки, центр
            self.delete(self.cog)
            self.delete(self.hourHand)
            self.delete(self.minsHand)
            self.delete(self.secsHand)
        originX = originY = radius = cfg.size/2
        hour = hour + (mins / 60.0)
        hx, hy = self.point(hour, 12, (radius * .80), originX, originY)
        mx, my = self.point(mins, 60, (radius * .90), originX, originY)
        sx, sy = self.point(secs, 60, (radius * .95), originX, originY)
        self.hourHand = self.create_line(originX, originY, hx, hy,
                            width=(cfg.size * .04),
                            arrow='last', arrowshape=(25,25,15), fill=cfg.hh)
        self.minsHand = self.create_line(originX, originY, mx, my,
                            width=(cfg.size * .03),
                            arrow='last', arrowshape=(20,20,10), fill=cfg.mh)
        self.secsHand = self.create_line(originX, originY, sx, sy,
                            width=1,
                            arrow='last', arrowshape=(5,10,5), fill=cfg.sh)
        cogsz = cfg.size * .01
        self.cog = self.create_oval(originX-cogsz, originY+cogsz,
                                    originX+cogsz, originY-cogsz, fill=cfg.cog)
        self.dchars(self.ampm, 0, END)
        self.insert(self.ampm, END, ampm)

    def onResize(self, newWidth, newHeight, cfg):
        newSize = min(newWidth, newHeight)
        #print 'analog onResize', cfg.size+4, newSize
        if newSize != cfg.size+4:
            cfg.size = newSize-4
            self.delete('all')
            self.drawClockface(cfg)  # onUpdate called next

################################################################################
# Составной объект часов
################################################################################
```

```
ChecksPerSec = 10  # second change timer

class Clock(Frame):
    def __init__(self, config=ClockConfig, parent=None):
        Frame.__init__(self, parent)
        self.cfg = config
        self.makeWidgets(parent)                    # дочерние пакуются,
        self.labelOn = 0                            # но клиенты могут делать pack или grid
        self.display = self.digitalDisplay
        self.lastSec = -1
        self.onSwitchMode(None)
        self.onTimer()

    def makeWidgets(self, parent):
        self.digitalDisplay = DigitalDisplay(self, self.cfg)
        self.analogDisplay  = AnalogDisplay(self,  self.cfg)
        self.dateLabel      = Label(self, bd=3, bg='red', fg='blue')
        parent.bind('<ButtonPress-1>', self.onSwitchMode)
        parent.bind('<ButtonPress-3>', self.onToggleLabel)
        parent.bind('<Configure>',     self.onResize)

    def onSwitchMode(self, event):
        self.display.pack_forget()
        if self.display == self.analogDisplay:
            self.display = self.digitalDisplay
        else:
            self.display = self.analogDisplay
        self.display.pack(side=TOP, expand=YES, fill=BOTH)

    def onToggleLabel(self, event):
        self.labelOn = self.labelOn + 1
        if self.labelOn % 2:
            self.dateLabel.pack(side=BOTTOM, fill=X)
        else:
            self.dateLabel.pack_forget()
        self.update()

    def onResize(self, event):
        if event.widget == self.display:
            self.display.onResize(event.width, event.height, self.cfg)

    def onTimer(self):
        secsSinceEpoch = time.time()
        timeTuple      = time.localtime(secsSinceEpoch)
        hour, min, sec = timeTuple[3:6]
        if sec != self.lastSec:
            self.lastSec = sec
            ampm = ((hour >= 12) and 'PM') or 'AM'              # 0...23
            hour = (hour % 12) or 12                            # 12..11
            self.display.onUpdate(hour, min, sec, ampm, self.cfg)
            self.dateLabel.config(text=time.ctime(secsSinceEpoch))
        self.after(1000 / ChecksPerSec, self.onTimer)   # выполнять N раз в секунду

################################################################################
# Автономные часы
################################################################################

class ClockWindow(Clock):
    def __init__(self, config=ClockConfig, parent=None, name=''):
        Clock.__init__(self, config, parent)
```

```
        self.pack(expand=YES, fill=BOTH)
        title = 'PyClock 1.0'
        if name: title = title + ' - ' + name
        self.master.title(title)               # master=родитель или по умолчанию
        self.master.protocol('WM_DELETE_WINDOW', self.quit)

##############################################################################
# Выполнение программы
##############################################################################

if __name__ == '__main__':
    def getOptions(config, argv):
        for attr in dir(ClockConfig):             # заполнить объект конфигурации по умолчанию
            try:                                  # из аргументов командной строки "-attr val"
                ix = argv.index('-' + attr)
            except:
                continue
            else:
                if ix in range(1, len(argv)-1):
                    if type(getattr(ClockConfig, attr)) == type(0):
                        setattr(config, attr, int(argv[ix+1]))
                    else:
                        setattr(config, attr, argv[ix+1])
    import sys
    config = ClockConfig()
    #config = PhotoClockConfig()
    if len(sys.argv) >= 2:
        getOptions(config, sys.argv)         # clock.py -size n -bg 'blue'...
    myclock = ClockWindow(config, Tk())      # при автономном выполнении родителем
                                             # служит корень Tk
    myclock.mainloop()
```

И наконец, в примере 9.23 показан модуль, фактически выполняемый из запускающего сценария PyDemos, – в нем предопределен ряд стилей часов и производится запуск одновременно шести часов, прикрепляемых к новым окнам верхнего уровня для создания демонстрационного эффекта (хотя на практике обычно даже мне достаточно иметь на экране одни часы).[1]

Пример 9.23. PP2E\Gui\Clock\clockStyles.py

```
from clock import *
from Tkinter import mainloop

gifdir = '../gifs/'
if __name__ == '__main__':
    from sys import argv
    if len(argv) > 1:
        gifdir = argv[1] + '/'

class PPClockBig(PhotoClockConfig):
    picture, bg, fg = gifdir + 'ora-pp.gif', 'navy', 'green'

class PPClockSmall(ClockConfig):
    size    = 175
    picture = gifdir + 'ora-pp.gif'
```

[1] **Учтите, что указанные в этом сценарии графические файлы могут отсутствовать на CD из соображений соблюдения авторских прав. Можете в отместку вспомнить здесь свой любимый анекдот про юриста.**

```
    bg, fg, hh, mh = 'white', 'red', 'blue', 'orange'

class GilliganClock(ClockConfig):
    size    = 550
    picture = gifdir + 'gilligan.gif'
    bg, fg, hh, mh = 'black', 'white', 'green', 'yellow'

class GuidoClock(GilliganClock):
    size = 400
    picture = gifdir + 'guido_ddj.gif'
    bg = 'navy'

class GuidoClockSmall(GuidoClock):
    size, fg = 278, 'black'

class OusterhoutClock(ClockConfig):
    size, picture = 200, gifdir + 'ousterhout-new.gif'
    bg, fg, hh    = 'black', 'gold', 'brown'

class GreyClock(ClockConfig):
    bg, fg, hh, mh, sh = 'grey', 'black', 'black', 'black', 'white'

class PinkClock(ClockConfig):
    bg, fg, hh, mh, sh = 'pink', 'yellow', 'purple', 'orange', 'yellow'

class PythonPoweredClock(ClockConfig):
    bg, size, picture = 'white', 175, gifdir + 'pythonPowered.gif'

if __name__ == '__main__':
    for configClass in [
        ClockConfig,
        PPClockBig,
        #PPClockSmall,
        GuidoClockSmall,
        #GilliganClock,
        OusterhoutClock,
        #GreyClock,
        PinkClock,
        PythonPoweredClock
    ]:
        ClockWindow(configClass, Toplevel(), configClass.__name__)
    Button(text='Quit Clocks', command='exit').pack()
    mainloop()
```

PyToe: графический элемент игры в крестики-нолики

И наконец, в завершение главы немного развлечений. В нашем последнем примере, *PyToe*, с помощью Python реализована программа игры в крестики-нолики («tic-tac-toe») с искусственным интеллектом. Большинству читателей, вероятно, знакома эта игра, поэтому я не стану останавливаться на ее подробностях. Если вкратце, то игроки поочередно ставят свои метки в клетках игровой доски, пытаясь занять целиком строку, колонку или диагональ. Победителем является тот, кому удалось первым это сделать.

В PyToe позиции на доске помечаются щелчком мыши, а одним из игроков является программа Python. Сама игровая доска выводится с помощью простого GUI Tkinter; по умолчанию PyToe строит доску размером 3 на 3 (стандартная настройка игры), но можно настроиться на игру произвольного размера N на N.

Когда приходит очередь компьютера сделать ход, с помощью алгоритмов искусственного интеллекта (AI) оцениваются возможные ходы и ведется поиск в дереве этих ходов и возможных ответов на них. Это довольно простая задача для игровых программ, а эвристики, применяемые для выбора ходов, несовершенны. Все же PyToe обычно достаточно сообразителен, чтобы найти выигрыш на несколько ходов раньше, чем пользователь.

Выполнение PyToe

GUI PyToe реализован в виде фрейма с упакованными метками и привязкой щелчков мыши к этим меткам для перехвата ходов пользователя. Текст метки устанавливается равным метке игрока после каждого хода компьютера или пользователя. Класс `GuiMaker`, который мы кодировали ранее в этой главе, тоже использован здесь для создания простой панели меню в верхней части окна (но панель инструментов внизу не выводится, так как PyToe оставляет ее дескриптор пустым). По умолчанию пользователь ставит метки «X», а PyToe ставит «O». На рис. 9.24 показано, как PyToe готов победить меня одним из двух способов.

Рис. 9.25 показывает всплывающий диалог подсказки PyToe, в котором приведены параметры конфигурации командной строки. Есть возможность задать цвет и размер для меток игровой доски, игрока, делающего первый ход, метку пользователя («X» или «O»), размер доски (переопределяющий 3 на 3 по умолчанию) и стратегию выбора хода для компьютера (например, «Minimax» выполняет поиск выигрышей и поражений в дереве ходов, а «Expert1» и «Expert2» используют статические эвристические функции оценки).

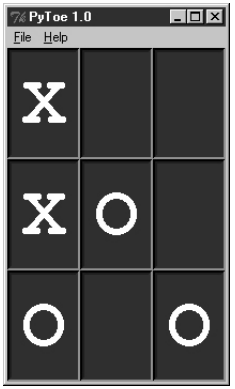

Рис. 9.24. PyToe обдумывает путь к победе

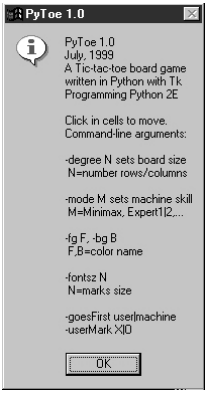

Рис. 9.25. Окно подсказки PyToe с параметрами командной строки

Используемая в PyToe технология ИИ интенсивно использует ЦП, и одни стратегии выбора хода компьютером требуют большего времени, чем другие, но скорость зависит в основном от скорости компьютера. Задержка, связанная с выбором хода, составляет на моей машине (650 МГц) доли секунды при игре 3 на 3 для любого задания стратегии выбора хода «-mode».

На рис. 9.26 показана альтернативная конфигурация PyToe сразу после того, как программа выиграла у меня. Несмотря отобранные для этой книги картинки, при некоторых параметрах выбора хода мне все же удается иногда выигрывать. При большей доске и более сложной игре алгоритм выбора хода PyToe становится еще более эффективным.

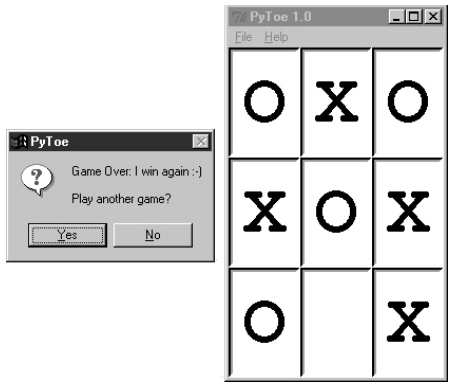

Рис. 9.26. Вариант размещения

Исходный код PyToe (на CD)

PyToe является крупной системой, предполагающей наличие некоторой подготовки в ИИ, а в отношении GUI, в сущности, не демонстрирует ничего нового. Отчасти по этой причине, но в основном из-за того, что я уже исчерпал предел страниц, отведенных на эту главу, исходный код не будет здесь приводиться, а обратиться за ним следует к прилагаемому CD. За деталями реализации PyToe обратитесь, пожалуйста, к следующим двум файлам из дистрибутива примеров:

- *PP2E\Ai\TicTacToe\tictactoe.py*, сценарий оболочки верхнего уровня
- *PP2E\Ai\TicTacToe\tictactoe_lists.py*, суть реализации

Могу вам посоветовать обратить внимание на то, что наибольшую трудность составляет структура данных используемой для представления состояния игровой доски. Если вы разберетесь, каким образом моделируется доска, то остальная часть кода станет вполне понятна.

Например, в варианте, основанном на списках, для представления состояния доски используется список списков, а также простой словарь из графических элементов полей ввода для GUI, индексируемый координатами игровой доски. Очистка доски после игры заключается просто в очистке исходных структур данных, как показано в следующем отрывке кода из указанных выше примеров:

```
def clearBoard(self):
    for row, col in self.label.keys():
        self.board[row][col] = Empty
        self.label[(row, col)].config(text=' ')
```

Аналогично выбор хода, по крайней мере, в случайном режиме, заключается в том, чтобы найти пустую ячейку в массиве, представляющем доску, и записать метку машины там и в GUI (degree означает размер доски):

```
def machineMove(self):
    row, col = self.pickMove()
    self.board[row][col] = self.machineMark
    self.label[(row, col)].config(text=self.machineMark)

def pickMove(self):
    empties = []
    for row in self.degree:
```

```
            for col in self.degree:
                if self.board[row][col] == Empty:
                    empties.append((row, col))
        return random.choice(empties)
```

Наконец, проверка состояния завершения игры сводится к просмотру строк, колонок и диагоналей по следующей схеме:

```
    def checkDraw(self, board=None):
        board = board or self.board
        for row in board:
            if Empty in row:
                return 0
        return 1 # не пусто: ничья или выигрыш

    def checkWin(self, mark, board=None):
        board = board or self.board
        for row in board:
            if row.count(mark) == self.degree:      # проверка по горизонтали
                return 1
        for col in range(self.degree):
            for row in board:                       # проверка по вертикали
                if row[col] != mark:
                    break
            else:
                return 1
        for row in range(self.degree):              # проверить диаг1
            col = row                               # строка == колонка
            if board[row][col] != mark: break
        else:
            return 1
        for row in range(self.degree):              # проверить диаг2
            col = (self.degree-1) - row             # row+col = degree-1
            if board[row][col] != mark: break
        else:
            return 1

    def checkFinish(self):
        if self.checkWin(self.userMark):
            outcome = "You've won!"
        elif self.checkWin(self.machineMark):
            outcome = 'I win again :-)'
        elif self.checkDraw():
            outcome = 'Looks like a draw'
```

Другой код выбора хода в основном просто проводит другие виды анализа структуры данных игровой доски или генерирует новые состояния доски для поиска в дереве ходов и ответных ходов.

В том же каталоге находятся родственные файлы, реализующие альтернативные схемы поиска и оценки ходов, различные представления доски и т. д. За дополнительными сведениями об оценке игры и поиске в целом обратитесь к учебникам по ИИ. Это интересный материал, но слишком сложный, чтобы можно было достаточным образом осветить его в данной книге.

Что дальше

На этом завершается часть данной книги, посвященная GUI, но рассказ о GUI здесь не кончается. Если хотите лучше изучить GUI, посмотрите примеры Tkinter, которые будут встречаться дальше в книге и описаны в начале этой главы. PyMail, PyCalc, PyForm и PyTree – все они представляют собой дополнительные конкретные примеры GUI. В следующей части книги мы узнаем также, как создавать интерфейсы пользователя, выполняемые в веб-броузерах, – совершенно другая идея, но еще один вариант конструкции простых интерфейсов.

Помните также, что даже если вы не найдете в этой книге примера GUI, который выглядит очень похожим на тот, который вам нужно запрограммировать, то все конструктивные элементы вам уже знакомы. Создание более крупного GUI для вашего приложения в действительности заключается в иерархическом расположении составляющих его графических элементов, представленных в этой части книги.

Например, сложный экран может быть составлен в виде совокупности радиокнопок, окон списков, ползунков, текстовых полей, меню и т. д., располагаемых во фреймах или на сетке для получения требуемого внешнего вида. Сложный графический интерфейс может быть дополнен всплывающими окнами верхнего уровня, а также независимо выполняемыми программами GUI, связь с которыми поддерживается через такие механизмы IPC, как каналы, сигналы и сокеты.

Кроме того, более крупные компоненты GUI могут быть реализованы в виде классов Python, прикрепляемых или расширяемых всюду, где требуется аналогичный инструмент интерфейса (важным примером этого служит PyEdit). Немного творчества, и с помощью набора графических элементов Tkinter и Python можно создавать практически неограниченное число структур.

Помимо данной книги посмотрите также раздел, посвященный документации и книгам, на сайте Python *http://www.python.org*. Можно было бы поместить сюда тексты, относящиеся к Tkinter, но, надо полагать, предлагаемое в этом разделе расширится за то время, пока актуальна данная книга. Наконец, если вас заразит Tkinter, еще раз хочу порекомендовать загрузить пакеты, о которых было сказано в главе 6, особенно PMW и PIL, и поэкспериментировать с ними. Оба они наращивают мощь арсенала Tkinter, позволяя создавать более сложные GUI при минимуме кодирования.