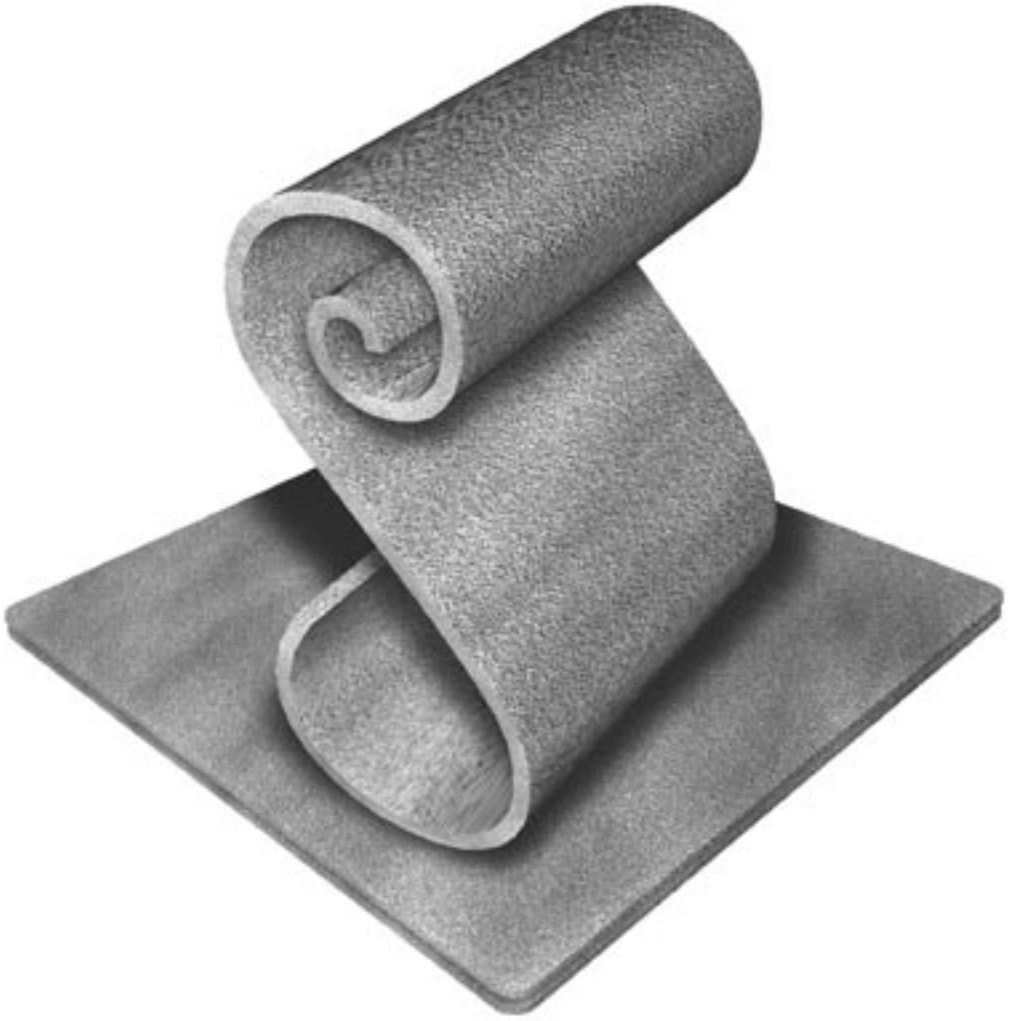
APPLESCRIPT ДЛЯ АБСОЛЮТНЫХ НОВИЧКОВ



Автор: Берт Алтенбург

Перевод: Кирилл Корчагин и Владимир Прохоренков

Верстка: Кирилл Корчагин

AppleScript – это революционная технология компании Apple, которая делает возможным общение программ между собой и с пользователем. AppleScript – это серия инструкций записываемая на языке AppleScript. Этот язык очень близок к английскому языку, что делает его одинаково легким для понимания, чтения и написания сценариев.

Книга будет полезна как начинающим, так и опытным пользователям Макинтош, которые хотят расширить свои возможности в управлении компьютером и повысить эффективность своей работы с помощью AppleScript.

СОДЕРЖАНИЕ

ВСТУПЛЕНИЕ

AppleScript – это революционная технология компании Apple, которая делает возможным общение программ между собой. Например, с AppleScript вы можете:

– получить электронное письмо в Mail и сохранить его в базе данных;
– заставить программу редактирования изображений сделать изменения в разрешении целой серии иллюстраций, изменить размер и послать полученные изображения на другой компьютер или разместить их на web-странице;
– и многое, многое другое.

AppleScript, далее просто сценарий или скрипт – это серия инструкций записываемая на языке AppleScript. Этот язык очень близок к английскому языку, что делает его одинаково легким для понимания, чтения и написания сценариев.

Несмотря на свою мощь, бывают случаи, где AppleScript тяжело использовать, но их очень мало. Настольные издательские системы зависят от AppleScript при автоматизации рабочих процессов (Adobe PhotoShop, QuarkXPress, Adobe InDesign). Разработчики FileMaker Pro используют его для создания баз данных. Кроме упомянутых программ, есть большое количество основных и второстепенных программ таких как GraphicConverter, BBEdit, и Microsoft Word – все они могут отвечать на команды AppleScript (скриптуемые). Это означает, что вы можете использовать AppleScript для того чтобы управлять этими программами напрямую. Однако, мы не будем фокусироваться на скриптуемых программах в данной книге. Для этого уже сейчас на рынке есть соответствующие книги. В таких книгах, как правило, очень мало рассматривается сам язык AppleScript, и обычно показывается как быстро добиться эффектных вещей, которые требуют среднего или хорошего знания AppleScript. Цель этой книги, помочь вам именно в освоении основ AppleScript.

Подразумевается также обновление и расширение этой книги. Наверняка вы захотите проверить наличие свежих версий (см. главу 15). Вторая книга о создании сценариев для разных программ с помощью AppleScript находится сейчас в стадии написания. Эта книга бесплатна, и вы можете поощрить автора и переводчиков тем, что привлечете внимание других пользователей Макинтош.

Как только вы окунетесь в мир AppleScript, вы заметите, что термин «AppleScript» используется с тремя различными понятиями:

- как язык похожий на английский язык, на котором записываются инструкции для Макинтош;
- AppleScript как серия готовых инструкций, т.е. сценариев, написанных на языке AppleScript;
- как часть операционной системы (Mac OS X), которая считывает и выполняет инструкции AppleScript.

В этой книге, при необходимости обращения к одному из этих понятий используется один из следующих терминов соответственно:

- язык AppleScript;
- AppleScript или сценарий;
- AppleScript (компонент Mac OS X).

Изучение сценариев AppleScript идеально подходит для вступления в мир программирования. С ним остается в стороне вся трудоемкая работа программиста свойственная компьютерным языкам, таким например как Java, которую необходимо проделывать даже при решении простейших задач. AppleScript достаточно прост, чтобы его мог изучить 10-и летний, но он также достаточно мощный, что даже профессионалы пользуются им. Это оставляет большой простор для вашей фантазии и вашего роста. Вы

можете даже создать с AppleScript программы, которые будут выглядеть и работать как коммерческие программы, обычно используемые вами на Маке, с кнопками, меню, иконками и прочим. Для их создания вам понадобится AppleScript Studio, бесплатно предоставляемый вашей любимой компьютерной компанией вместе с инструментарием разработчика (Developer Tools или XCode).

В чем различие между написанием сценариев и программированием? Мне нравится такая формулировка, что если пишется легко, то это сценарий, а если трудно – программирование. Однако, судя по моей книге, сценарии JavaScript не столь легки, так что это определение не совсем точно.

Как использовать эту книгу?

Как вы увидите, некоторые абзацы выделены зеленным цветом. Мы предлагаем читать вам каждую главу (как минимум) дважды. Сначала пропускайте зеленый текст, а потом, перечитайте главу, включая абзацы отмеченные зеленым цветом. Для большего эффекта вы повторите, что вы изучили, но также изучите некоторые интересные фичи, которые пропускались при первом прочтении. Используя книгу таким образом, вы сгладите неизбежные неровности изучения.

Эта книга содержит множество примеров. Для того, что бы точно указать какой используется пример, каждый сценарий обозначен цифрой в квадратных скобках, вот так: [4]. Большинство сценариев состоят из двух и более строк. Иногда, будет использоваться вторая цифра для ссылки на соответствующую строку. Например: [4.3] указывает на третью строку в сценарии [4].

Вы не сможете научиться скакать на лошади только с помощью чтения книги. Аналогично, вы не сможете изучить AppleScript, не работая за вашим Маком. Это электронная книга. Вам не удастся избежать переключения на программу Script Editor (см. главу 2), чтобы опробовать полученные знания.

ГЛАВА 0

ПРЕЖДЕ ЧЕМ НАЧАТЬ

Я написал эту книгу для вас. Так как она свободно распространяется, пожалуйста позвольте мне сказать пару слов чтобы порекламировать Макинтош. Каждый пользователь Макинтош может помочь в продвижении их любимой компьютерной платформы при небольшом усилии. Сейчас расскажу как.

1. Высокая эффективность вашей работы на Макинтош – это наиболее легкий путь убедить других людей считаться с платформой Макинтош. Так что держите руку на пульсе, посещайте Мак-сайты и читайте журналы посвященные Маку. Конечно, изучать AppleScript и применять его так же замечательно. А компании использующие AppleScript смогут сэкономить огромное количество денег и времени.

2. Покажите миру, что не все в нем используют PC, сделав Макинтош более заметным. Один из способов носить майку с маковской символикой, так вы и вне своего дома сделаете Мак более заметным. Если вы запустите CPU Monitor (в папке Utilities, которую вы найдете в папке Applications вашего компьютера), вы обратите внимание, что ваш Мак использует процессор на полную мощность только изредка. Ученые запустили несколько проектов распределенного вычисления (Distributed Computing – DC), такие как Folding@home, который использует эту простаивающую мощность. Скачайте маленькую бесплатную программу называемую DC-клиент, и запустите вычислительную рабочую единицу. DC-клиенты работают с самым низким приоритетом. Если какая-нибудь ваша программа потребует максимальную мощность от процессора, то DC-клиент немедленно освободит ее. Вы даже не почувствуете, что он работает. Поможет ли все эту Маку? Большинство DC-проектов ведут рейтинг на своих веб-сайтах о количестве рабочих единиц. Если вы присоединитесь к команде Мак (вы узнаете их имена в рейтингах), вы сможете помочь Мак-команде своим выбором и поднимите вверх их рейтинг. Таким образом, пользователи других компьютерных платформ увидят, что могут делать Маки и насколько хорошо. Есть DC-клиенты для множества разных направлений: математика, лечение болезней и многое другое. Для того, чтобы выбрать проект, который вам понравится, зайдите и выберите DC-клиент на сайте:

   www.aspenleaf.com/distributed/distrib-projects.html

   Есть одна проблема: это может затянуть.

3. Постарайтесь сделать программное обеспечение для платформы Макинтош лучше. Нет, вам не надо учиться программировать. Просто выработайте у себя привычку отправлять отзывы и пожелания (только вежливо) разработчикам програмного обеспечения, которое вы используете. Даже если вы попробовали какое-либо програмное обеспечение и оно вам не понравилось, то расскажите разработчику почему. Сообщая об ошибках предоставляйте описания всех ваших действий в тот момент, когда ошибка возникла. Чтобы получить великолепное и бесплатное мультимедиа-руководство на эту тему, посетите:

   http://www.macinstruct.com/tutorials/crash/index.html

4. Платите за програмное обеспечение, которое вы используете. Пока производство програмного обеспечения для Макинтош будет выгодным, разработчики будут создавать и поддерживать новое програмное обеспечение.

5. Пожалуйста, сообщите как минимум трем пользователям Макинтош об этой книге, и скажите где ее найти. Или прорекламируйте ее в четырех разных местах.

Ну все, пока вы в фоновом режиме скачиваете DC-клиент, давайте начнем!

ГЛАВА 1

СЦЕНАРИЙ – ЭТО СЕРИЯ ИНСТРУКЦИЙ

AppleScript как часть Mac OS может выполнять лишь ограниченное число задач. Для примера, воспроизведем системный звук. Посмотрите сейчас на сценарий [1], воспроизводящий системный звук Макинтош.

```
beep
```
[1]

Должно быть это самый короткий сценарий в мире, состоящий из единственной команды или инструкции. Одна строка содержит инструкцию, называемую оператором (statement), даже если эта строка длиной всего в одно слово. Теперь, если выполнить этот сценарий, Макинтош издаст системный звук.

Чтобы воспроизвести несколько раз системный звук, вы можете задать команду beep с числом, указывающим количество звуков, которое вы хотите услышать [2].

```
beep 2
```
[2]

Как вы можете увидеть сравнивая сценарии [1] и [2], дополнительная часть информации необязательна. Если вы не указываете число, то AppleScript предполагает, что вы хотите услышать только один сигнал. Итак, 1 – это значение по умолчанию.

Если вы считаете, что это слишком по-ПиСишному, то почему бы не позволить AppleScript общаться с нами в духе Макинтош [3], используя следующий оператор:

```
say "This is a spoken sentence."
```
[3]

Вы можете даже выбрать другой голос для произнесения этой фразы, например «Fred», «Trinoids», «Cellos», или «Zarvox» [4], чтобы изменить голос по умолчанию «Victoria».

```
say "This is a spoken sentence." using "Zarvox"
```
[4]

Примечание: в основном AppleScript не чувствителен к регистру символов текста. То есть, нет разницы будете ли вы писать прописными или строчными буквами. Однако, голоса, такие как «Victoria» или «Zarvox» следует писать соблюдая регистр.

Как вы видите, инструкции AppleScript похожи на английский язык. Это делает сценарий вполне читаемым и понимаемым, даже если вы ни разу не писали какие-либо сценарии. Но пока что сценарии [1–4] возможно пригодятся только для развлечения: пользы от них мало. Язык AppleScript имеет немного больше команд, но, возможно, не так много чтобы вас впечатляло их количество. Сила AppleScript в том, что он позволяет вам взаимодействовать с другими программами. Но эти программы должны поддерживать язык сценариев – должны быть скриптуемыми (scriptable). В результате чего в вашем распоряжении не только предоставленный, ограниченный набор команд от AppleScript (компонента Mac OS X), но также и громадное количество команд предоставляемое вашими программами.

Некоторые программы на Макинтош более популярны, чем другие. Но одна из них используется каждым пользователем Макинтош – Finder. Да, Finder это программа. Когда вы включаете Макинтош, он запускается автоматически и все время работает. Finder позволяет вам перемещать файлы, искать файлы на жестком диске, создавать папки, копировать или переименовывать их и многое другое. Например, если вы очищаете Trash, то для этого вы используете именно Finder. Но раз вы можете выполнить

операцию очистки корзины мышкой или клавиатурой, то почему бы не сделать то же самое с помощью AppleScript [5].

```
tell application "Finder"
    empty the trash
end tell
```
[5]

Как главный, вы должны сказать:

- кто должен выполнить задание и
- что должно быть выполнено.

Бесполезно говорить, например, программе Adobe PhotoShop что она должна очистить корзину. Программа PhotoShop не знает как это сделать. Поэтому, инструкция для выполнения очистки корзины должна быть направлена к Finder.

Как и в реальном мире, работу, которую приказал сделать руководитель, вы можете сделать бездумно, но ваш Макинтош куда более обязательный работник, и делает то, что ему говорят. Если в корзине лежали важные документы, то после выполнения сценария AppleScript [5], вы потеряете их навсегда.

Первый оператором сценария [5.1] – оператор «tell», которым мы просим AppleScript (компонент Mac OS X) перенаправить один или более операторов другой программе, здесь программе Finder. AppleScript (компонент Mac OS X) будет продолжать так делать до тех пор, пока не встретит оператор «end tell» [5.3]. В приведенном выше сценарии [5] мы указываем AppleScript послать сообщение программе Finder инструкцию очистки корзины и затем прекращаем говорить программе Finder, что делать. Взятые вместе, получатся такие строки:

```
tell application "xyz"

end tell
```

которые называются «блоком обращения» (tell block). Инструкции выполняемые программой «xyz» заключены в блок обращения программы «xyz». Кстати, несмотря на то что язык AppleScript не очень строг в системе обозначений, при сравнении с другими языками сценариев и особенно с языками программирования обнаруживается, что у него тоже есть некоторые правила. Одно из них гласит, что вы обязаны заключать в двойные кавычки имя программы, как в первом операторе [5.1]. (Изображение и соответственно компьютерное представление кавычек имеет большое значение, так здесь имеются в виду двойные прямые верхние кавычки ("). В тексте книги часто встречаются кавычки елочки: « и », которые в языке AppleScript также имеют большое значение и отличаются от прямых кавычек. – *прим. ред.*)

Еще можно дать Finder больше инструкций подряд. В примере ниже [6], два оператора [6.2, 6.3] предназначаются программе Finder. Так как оба оператора должны выполнятся программой Finder, они должны быть внутри блока обращения программы Finder.

```
tell application "Finder"
    empty the trash
    open the startup disk
end tell
```
[6]

После очистки корзины, Finder откроет окно, показывающее содержимое вашего жесткого диска.

Как видите, мы можем делать с Finder всё что захотим. Мы можем даже сказать Finder, чтобы он изменил размер окна, поменял его положение на экране и многое другое. Вы научитесь делать это позже.

А сейчас мы создадим сценарий, содержащий все инструкции, как операторы для Finder, так и для самого AppleScript (компонента Mac OS X) [7].

```
tell application "Finder"
    empty the trash
    open the startup disk                                    [7]
end tell
beep
```

Сначала, Finder получает пару инструкций [7.2, 7.3]. Затем инструкция «beep» [7.5] выполниться AppleScript'ом. Фактически, сигнал укажет, что сценарий выполнен.

Интересно то, что неважно где вы расположите инструкцию «beep» (и любую другую инструкцию понятную AppleScript (компоненту Mac OS X)) внутри или снаружи блока обращения [8].

```
tell application "Finder"
    empty the trash
    beep                                                     [8]
    open the startup disk
end tell
```

Хотя Finder не знает команды «beep», AppleScript (компонент Mac OS X) знает как на неё реагировать. Это делает сценарий и читаемым, и понятным. Другими словами, у вас получится в начале блок обращения, содержащий первый оператор Finder [8.2], потом оператор состоящий из команды «beep», и наконец, вторая часть блока обращения с последним оператором Finder [8.4].

Помните, пока команды понятны AppleScript (компоненту Mac OS X) они могут находится в любом месте сценария. Каждая инструкция предназначенная для какой-либо конкретной программы, такой как Finder, должна быть внутри блока обращения этой программы. Следующий сценарий [9] содержит критическую ошибку (последний оператор [9.5]).

```
tell application "Finder"
    empty the trash
    beep                                                     [9]
end tell
open the startup disk
```

AppleScript (компонент Mac OS X) не знает как открыть диск, и не способен сам отыскать программу, которая сможет это сделать. Первая часть сценария (операторы [9.2–9.3] внутри блока обращения) будет прекрасно выполнена, но последний оператор [9.5] невыполним.

В процессе выполнения сценария, если возникнет хотя бы одна ошибка, то какие-либо последующие операторы выполняться не будут [10].

```
tell application "Finder"
    empty the trash
end tell                                                     [10]
open the startup disk
say "I emptied the trash and opened the startup disk for you" using "Victoria"
```

После очистки корзины, AppleScript (компонент Mac OS X) остановится на операторе [10.4], который должен был бы адресоваться Finder'у. Вы не услышите фразу произносимую оператором [10.5], хотя в нем нет ничего неправильного.

ГЛАВА 2

СОХРАНЕНИЕ И ВЫПОЛНЕНИЕ СЦЕНАРИЯ

Вы уже видели пару сценариев, и согласитесь, что они очень похожи на английский язык, что делает их легкими при чтении и понимании. Вы смогли бы выполнить команды сценария – подобные очистке корзины – сами, используя мышь и клавиатуру. Давайте посмотрим как Мак сможет сделать это за вас.

Script Editor – это программа, где вы можете набрать сценарий и выполнить его. Вы найдете Script Editor в папке AppleScript, которая в свою очередь находится в папке Applications. После ее запуска, вы увидите два поля. Верхнее поле служит для ввода текста сценария [1].

[1]

Рядом с серединой верхней линейки инструментов вы увидите кнопку «Compile». Несмотря на то, что AppleScript может походить на английский, язык AppleScript все же очень далек от разговорного английского. «Yo Finder! Dump my garbage» (Эй Finder! Вынеси мой мусор!) или «Hey Finder! Clean out the bin» (эй Finder, очисти мою корзину) – это совсем не то чего ожидает Finder. В процессе называемом «компиляция» (compilation) AppleScript (компонент Mac OS X) выполняет проверку: все ли будет понятно в ходе вашего сценария. Если так, то сценарий форматируется в приятный, разноцветный текст. Неоткомпилированный текст отображается оранжевым цветом, а после компиляции зарезервированные слова показываются красным и синим. Стиль и цвет отображаемого текста можно установить в настройках параметров Script Editor.

Если сценарий не компилируется вследствие ошибки, вы увидите непонятное сообщение показывающее, что в сценарии что-то неправильно. Попробуйте убрать одну из кавычек в сценарии [2], и вы сами увидите как AppleScript (компонент Mac OS X) перестанет вас понимать.

```
say "I'm learning AppleScript the easy way!" using "Zarvox"
```
[2]

Если все правильно, то вы можете нажать кнопку Run, и ваш скрипт выполнится. Сейчас запустите Script Editor, выберите один из виденных вами ранее сценариев и попробуйте выполнить его!

Можно нажать клавишу Enter как клавиатурное сокращение для компиляции сценария. Enter находится справа от клавиши Space (для ноутбуков) или в цифровой части (у настольных Маков). Клавиша Return (около правой клавиши Shift) работает так как вы и ожидаете и создает новую строку после текущей. Вы не можете использовать клавишу Return для компиляции сценария.

Нет особой надобности нажимать кнопку Compile перед каждым запуском сценария. Если вы нажмете кнопку Run, синтаксис сценария проверится, и, если все в порядке, сценарий сразу же запустится.

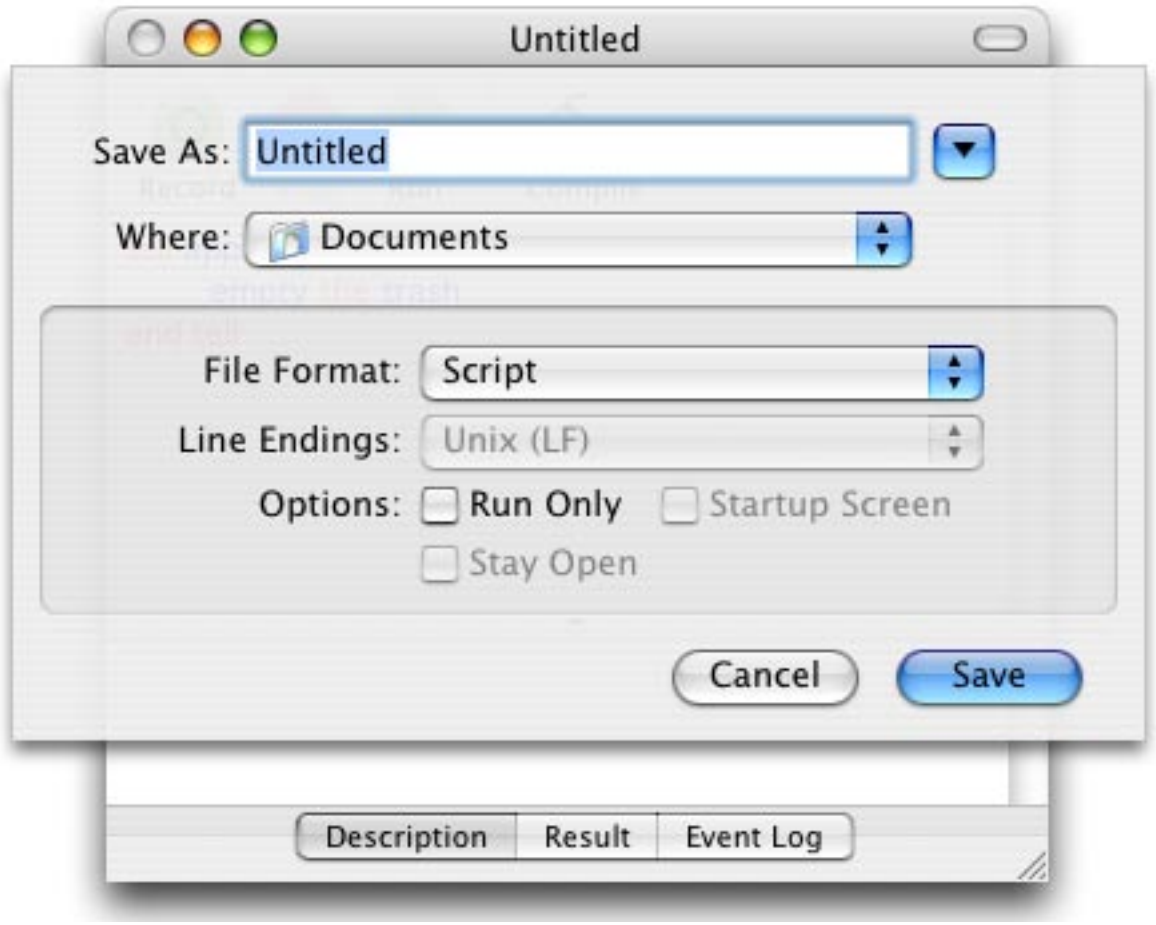
Заменить нажатие кнопки Run, можно нажатием комбинации клавиш Command-R.

Примечание: В действительности, компиляция включает в себя больше, чем просто проверку синтаксиса, но это не должно вас беспокоить.

Сохранение сценария

Есть несколько вариантов сохранения вашего сценария. Если сценарий не был успешно откомпилирован, вы можете сохранить сценарий только как простой текст.

Если проблем в процессе компиляции не было, выплывет показанное ниже диалоговое окно, и вы можете сохранить ваш текст как откомпилированный сценарий (compiled script) или как программу (application).



ОТКОМПИЛИРОВАННЫЙ СЦЕНАРИЙ: если вы дважды щелкните по иконке сохраненного, откомпилированного сценария AppleScript,

откроется Script Editor и вы сможете выполнить сценарий нажатием кнопки Run.

ПРОГРАММА: если вы дважды щелкните по иконке сценария, сохраненного как AppleScript программа,

сценарий сразу же запуститься на выполнение. Причем, Script Editor не откроется. Сценарий, сохраненный как программа, можно использовать как элемент автозагрузки (в System Preferences). После входа в систему, ваш Мак будет выполнять задачи указанные в вашем сценарии. Если вам нужно отредактировать сценарий, сохраненный как программа, запустите Script Editor, и откройте сценарий командой меню Open из меню File.

ПРЕДУПРЕЖДЕНИЕ: установка соответствующей галочки в диалоговом окне «Save», позволяет сохранить ваш скрипт доступным только для чтения (run-only). Убедитесь, что у вас есть резервная копия вашего сценария, потому что сохраненный только для чтения сценарий нельзя открыть и отредактировать снова.

ГЛАВА 3

УПРОЩЕНИЕ СОЗДАНИЯ СЦЕНАРИЕВ (I)

В главе 1 вы уже познакомились со сценарием:

```
tell application "Finder"
    empty the trash
end tell
```
[1]

Давайте посмотрим, как Script Editor старается помочь вам при создании сценариев.

В первой строке блока обращения, вместо того, чтобы набирать слово «application» полностью, вы можете написать:

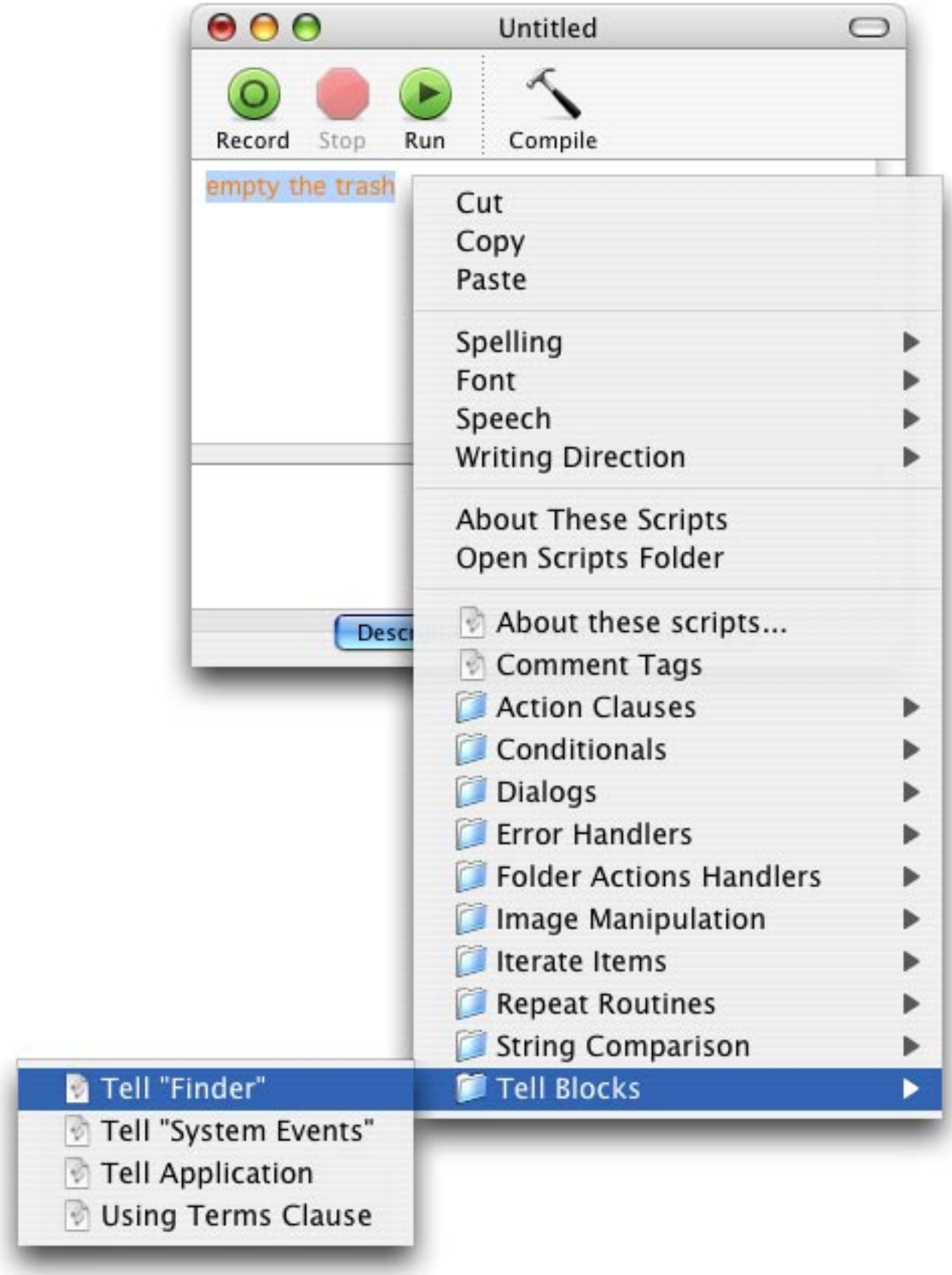
```
tell app "xyz"
```

Во время компиляции, Script Editor продолжит слово «app» до «application». Мало того, вам так же не надо набирать или знать как пишется имя приложения «xyz». Просто напишите что-нибудь (предложите имя не похожее на название какого-либо приложения), например «pqr». Когда вы будете компилировать сценарий, AppleScript предложит вам список всех скриптуемых приложений на вашем Маке. Вам останется только выбрать подходящее приложение, и AppleScript заменит «pqr» правильным именем приложения, и в сущности, завершит написание оператора tell за вас.

В действительности, Script Editor позволяет вам создавать блок обращения без набора текста, используя контекстное меню. Это такой тип меню, которое появляется если удерживая клавишу control щелкнуть мышью. Этот фокус можно проделать двумя способами:

1) Control и щелчок на верхнем поле Script Editor. Появится меню (см. иллюстрацию на следующей странице), и в конце этого меню вы увидите элемент меню с надписью «Tell Blocks». После щелчка мышью по нему, появится подменю, выберите «Tell "Finder"».

2) Если ваш сценарий уже содержит один или более операторов Finder – такие как «empty the trash» – которые еще не заключены в блок обращения, выделите оператор(ы), и повторите шаг 1. Вы можете увидеть это в действии на картинке ниже. Ваши операторы автоматически будут заключены в блок обращения.


Untitled
Record
Stop
Run
Compile
empty the trash
Cut
Copy
Paste
Spelling
Font
Speech
Writing Direction
About These Scripts
Open Scripts Folder
About these scripts...
Comment Tags
Action Clauses
Conditionals
Dialogs
Error Handlers
Folder Actions Handlers
Image Manipulation
Iterate Items
Repeat Routines
String Comparison
Tell Blocks
Tell "Finder"
Tell "System Events"
Tell Application
Using Terms Clause

ГЛАВА 4

РАБОТА С ЧИСЛАМИ

В начальной школе вы должно быть делали вычисления, заменяя точки результатом:

2 + 6 = ...
... = 3 * 4

В средней школе, точки вышли из употребления, и всюду были расставлены переменные называемые «x» и «y». Оглядываясь назад, вы можете удивиться тому, что люди так пугались этого небольшого изменения в системе обозначений.

2 + 6 = x
y = 3 * 4

AppleScript тоже использует переменные. Переменные (variables) – это ничего более, чем удобное имя, определяющее некоторую особую часть данных, таких как число. Имена переменных часто называют «идентификаторами» (identifiers), поскольку они обозначают, идентифицируют данные. Здесь имеется два примера [1] с операторами AppleScript, где переменным присваиваются некоторые значения с помощью команды «set».

```
set x to 25
set y to 4321.234
```
[1]

Покуда имена переменных, сами по себе, в AppleScript не имеют специального значения, для нас, людей, описательные имена переменных могут сделать сценарий более легким для чтения и потому более понятным. Это большой плюс: в случае если вам надо отыскать ошибку в вашем сценарии (ошибки в сценариях и программах традиционно называют «блохи», «баги» (bugs)). Следовательно, избегайте использования непонятных имен переменных, таких как «x». Например, переменная для обозначения ширины изображения может быть названа «pictureWidth» [2].

```
set pictureWidth to 8
```
[2]

Пожалуйста, обратите внимание, что имя переменной состоит из одного слова (или, в крайнем случае, из одного символа). После проверки синтаксиса, имя переменной отобразится зеленным цветом, так что вы сможете сразу увидеть, что это не зарезервированное слово AppleScript, которое выделяется голубым или красным цветом. А также, заметьте, что данные (такие как число «8» в сценарии [2]) окрашены черными цветом.

Пока вы вполне свободны в выборе имен переменных, но есть несколько правил, которым они должны соответствовать. Я могу их все вам перечислить, но это будет неинтересно. Основное правило, которому вы должны следовать, то что имя вашей переменной не может быть командой AppleScript или каким-любо зарезервированным словом. Например, «set», «say», «to», и «beep» – слова, которые имеют специальное значение в AppleScript. Составляя имена переменных из словосочетаний, подобных «pictureWidth», вы всегда можете быть уверены в них. Чтобы поддерживать читаемость имен переменных, рекомендуется использовать в них прописные символы.

Если вы настаиваете на изучении еще пары правил, дочитайте этот параграф. Кроме букв, допустимо использовать ещё и цифры, но имя переменной нельзя начинать с цифры. Также допускается символ подчеркивания _.

Теперь, когда мы научились присваивать переменной значение, мы можем производить вычисления. AppleScript в состоянии выполнять основные математические операции, поэтому нет нужды обращаться к специальным программам, которые могли бы произвести вычисления, чтобы определить площадь изображения. Вот сценарий [3], который делает именно это.

```
set pictureWidth to 8
set pictureHeight to 6                                                    [3]
set pictureSurfaceArea to pictureWidth * pictureHeight
```

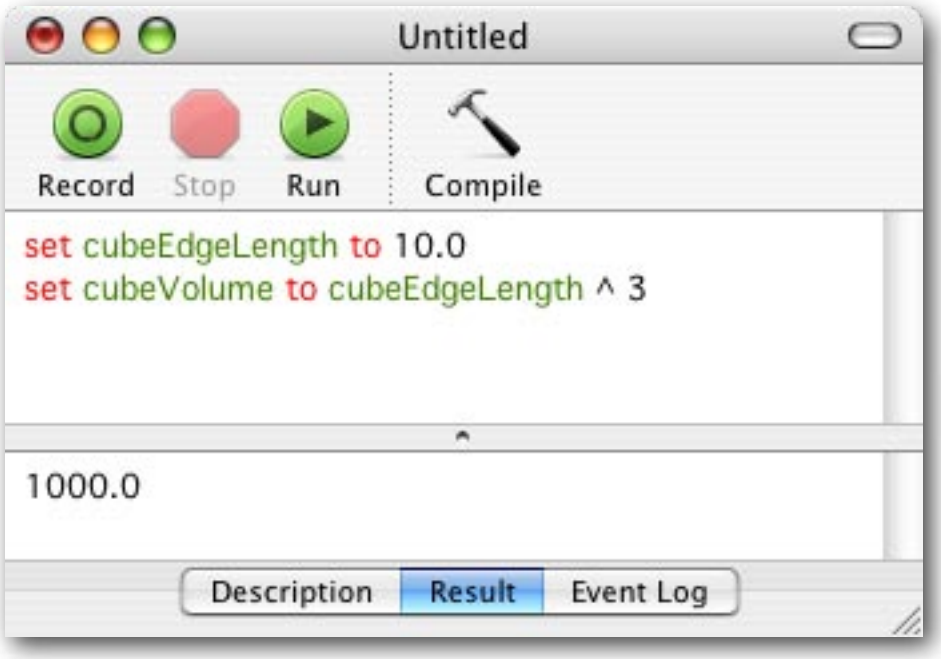
Используйте следующие символы, официально известные как операторы (operators), для выполнения основных математических вычислений.

+ для сложения
– для вычитания
/ для деления
* для умножения

Степень записывается символом ^. Сценарий [4], который вычисляет объем куба.

```
set cubeEdgeLength to 10.0
set cubeVolume to cubeEdgeLength ^ 3                                      [4]
```

Если выполнить этот сценарий [4] в Script Editor, то результат отобразится в нижнем поле редактора. Если вы не видите результат, переместите горизонтальную линию повыше от закладок. Поле результата (Result) покажет результат последнего выполненного оператора. Если ваш сценарий содержит только один оператор [4.1], то в поле результата будет показано «10.0». Для всего сценария [4], результат будет «1000.0». То есть, выражение «cubeEdgeLength ^ 3» будет посчитано, и будет показан результат.



Числа делятся на два типа: целые и дробные. Вы можете увидеть пример каждого из них в операторах [1.1] и [1.2] соответственно. Целые (integers) используются для счета, в котором мы будем что-нибудь подсчитывать, когда будем повторять серию инструкций определенное количество раз (см. главу 13). Вам знакомы дробные числа или вещественные (real) числа, например как средняя скорость болида в гонках Formula-1. Кстати, оба и целые, и вещественные числа могут быть отрицательными, как вы можете знать по вашему счету в банке.

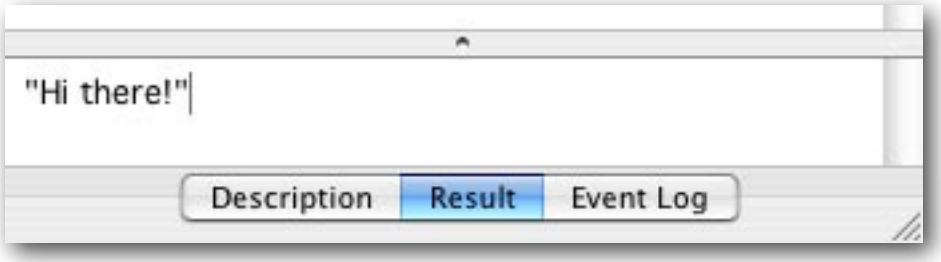
ГЛАВА 5

РАБОТА С ТЕКСТОМ

Переменные служат не только для хранения чисел. Они так же хорошо могут использоваться и для хранения текста. Фрагмент текста, даже если его длина равна нулю или одному знаку, называться строкой (string). Строка должна размещаться между двойными кавычками. В трех примерах [1] переменным с наглядными именами присваиваются соответствующие строковые значения.

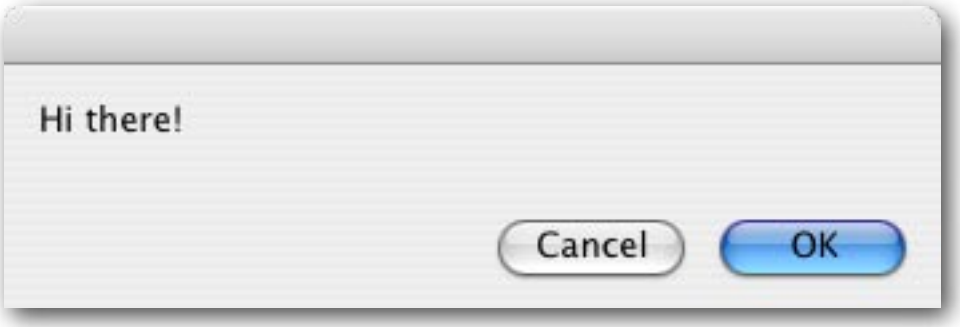
```
set emptyString to ""
set notEmtpyContainsASpace to " "
set greeting to "Hi there!"
```
[1]

После выполнения сценария [1], поле результата покажет строку в двойных кавычках. Так что, поле результата информирует вас не только о значении, но и о типе данных (числа без кавычек; строки в кавычках).



Поскольку поле результата показывает результат последнего выполненного оператора, отобразится только строка содержащая приветствие [1.3].

В дополнение к полю результата, AppleScript предлагает удобную альтернативу для связи с миром: диалоговое окно. Выглядит это так:



Его можно вызвать командой «display dialog», после которой следуют данные (число или строка), которые вы хотите показать. Диалог на рисунке выше был создан следующим сценарием [2].

```
display dialog "Hi there!"
```
[2]

Почему строковые значения имеют двойные кавычки? AppleScript может содержать только очень ограниченный запас слов, а читая ваш скрипт и расшифровывая что инструкция, а что нет, он сильно загрузит компьютер. Поэтому, AppleScript полагается на ключи, которые помогают ему понять значение каждого элемента в операторе сценария. По этой причине мы должны строковое значение помещать в двойные кавычки. Иначе, AppleScript может перепутать строку с именем переменной. Проверьте следующий сценарий [6]:

```
set stringToBeDisplayed to "Hi there!"
display dialog "stringToBeDisplayed"
display dialog stringToBeDisplayed
```
[6]

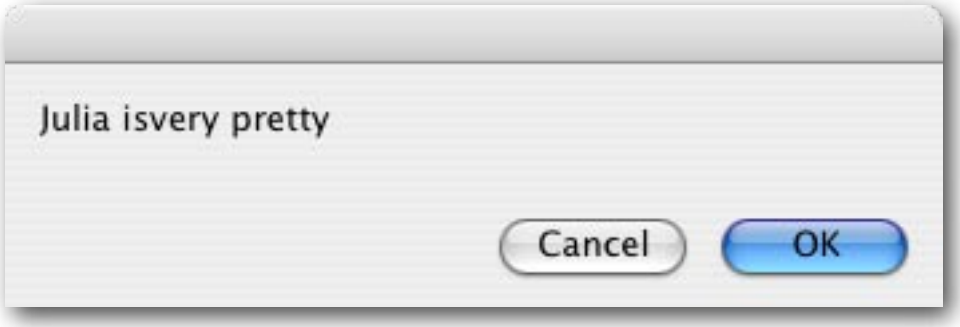
Запустите этот сценарий и вы сами увидите, что произойдет. Оператор [6.2] покажет текст «stringToBeDisplayed», тогда как оператор [6.3] отобразит «Hi there!». Так как Script Editor показывает откомпилированный сценарий различными цветами, легко увидеть, что в операторе [6.3] «stringToBeDisplayed» является именем переменной, и потому окрашена зеленым цветом, в операторе [6.2] «stringToBeDisplayed» показана черным цветом, который указывает, что это слово есть данные (строка). Порой, форматирование с цветом будет помогать вам быстрее отыскивать ошибки.

Как было сказано выше, AppleScript'у нужны ключи для расшифровки англоязычного сценария во что-нибудь понятное Макинтошу. Тут приведен другой пример, почему эти ключи важны: если мы напишем «thirty» как число в двойных кавычках, т.е. как "30", это будет уже не число, а строка. Различать типы данных – очень важно, потому что некоторые операции могут быть выполнены только с конкретным типом данных. Например, вы можете разделить два числа, но вы не сможете разделить одну строку на другую. Давайте взглянем на пару операций, которые можно выполнять над строками.

Подобно числам (и избирателям), строками можно манипулировать. Вы можете склеить строки вместе используя амперсанд, такая операция называется конкатенация (объединение) [7].

```
set nameOfActress to "Julia"
set actressRating to "very pretty"
set resultingString to nameOfActress & " is" & actressRating
display dialog resultingString
```
[7]

В третьем операторе [7.3], мы объединяем три строки, две из которых представлены переменными.



Пожалуйста, обратите внимание, что количество пробелов между строкой и амперсандом не влияет на результирующую строку, содержащуюся в переменной resultingString. После компиляции, Script Editor уменьшит число пробелов до одного, если вы поставили более чем один. Если вам нужен один или более пробелов, чтобы разделить слова в показываемом предложении, вам надо поставить их внутри двойных кавычек строки. В опраторе [7.3], кроме пробела слева от слова «is», должен быть еще один пробел, следующий за буквой «s» слова «is».

Есть еще команды для работы со строками. Некоторые из них требуют дополнительных знаний, мы встретимся с ними в следующих главах, так что на данный момент оставим их в стороне. Но мы можем дать вам пример одной команды относящейся к строкам. Вы можете узнать длину строки [8].

```
set theLength to the length of "I am"
```
[8]

Если вы запустите этот сценарий, то результат, показанный в поле результата, будет – 4. Так что запомните, что при вычислении длины строки, также считаются и пробелы.

Так как двойные кавычки используются для обозначения начала и конца строки, вам может показаться, что строка не может содержать двойные кавычки. Конечно, язык AppleScript предлагает обходной путь,

называемый «эскейпинг» (escaping). Просто добавьте обратную косую черту перед двойной кавычкой, и AppleScript больше не будет пытаться интерпретировать двойную кавычку как конец строки [9].

```
set exampleString to "She said: \"Hi, I'm Julia.\""
```
[9]

Если вы поразмышляете над этим, то обнаружите, что проблема остается в тех редких случаях, когда строка должна содержать двойную кавычку следующую за обратной косой чертой. Предположим вы хотите вывести следующий текст:

blah blah \" blah.

Сперва, мы поставим обратную косую черту перед косой чертой. Это означает, что AppleScript будет игнорировать любое специальное значение следующего символа, то есть значение второй обратной косой черты. Конечно, нам все еще надо уберечь нашу двойную кавычку, иначе AppleScript сочтет, что наша строка закончилась в этом месте. Следовательно, двойной кавычке должна предшествовать обратная косая черта, подобно тому как мы видели раньше. Собрав всё это вместе, мы прийдем к следующему оператору [10].

```
display dialog "blah blah \\\" blah"
```
[10]

Для удобства, я показал эскейп-символы (обратные косые черты) жирными, хотя Script Editor этого не делает. Будьте осторожны с обратной косой чертой, потому что она может иметь особое значение и перед некоторыми другими символами. Например, \n обозначает новую строку (аналогично Return – возврат каретки), и \t обозначает табуляцию (Tab).

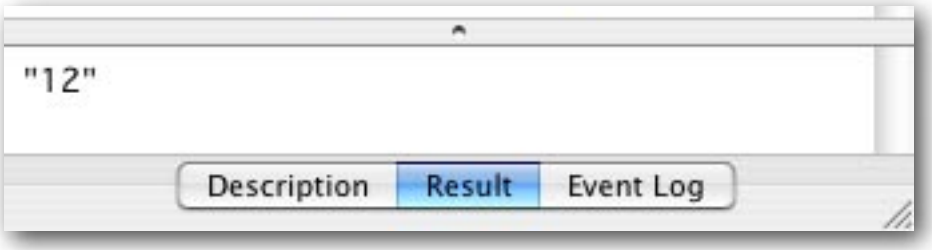
Как сказано выше, числа и строки – различные типы данных. Вы не можете вычесть число три из строки [11].

```
set nonsensical to "fifteen" - 3
```
[11]

Если вы попытаетесь выполнить сценарий [11], вы поймете насколько дружественен язык AppleScript. Он действительно попытается конвертировать строку в число. Если строка была бы "15", вместо "fifteen", эта конвертация сработала бы. Конвертирование одного типа данных в другой называется преобразованием (coercion). Вы можете произвести преобразование как показано в следующих двух примерах [12]

```
set coercedToNumber to "15" as number
set coercedToString to 12 as string
```
[12]

В поле для результатов программы Script Editor виден результат выполения последнего оператора [12.2]. Данные содержащиеся в «coercedToString» – являются строкой, так как двойные кавычки ясно указывают на это (см. рисунок ниже).



Для сценария заканчивающегося первым оператором [12.1], поле результата показало бы 15 без двойных кавычек, указывая что результат число, а не строка.

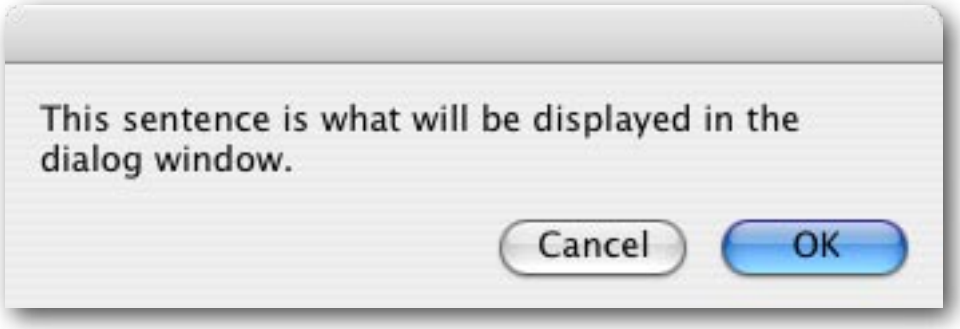
ГЛАВА 6

СПИСКИ

В предыдущих главах, вы увидели как писать очень простые сценарии для выполнения основных вычислений и операций над строками. Пользователю сценария можно предоставить результат командой «display dialog» [1]. В этой и следующей главах мы будем использовать команду «display dialog» чтобы лучше изучить различные аспекты языка AppleScript.

```
display dialog "This sentence is what will be displayed in the dialog window."      [1]
```

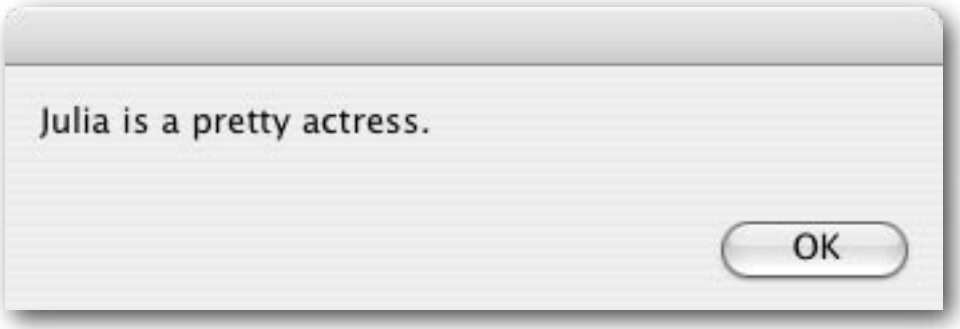
Если выполнить сценарий [1], вы увидите диалоговое окно, которое по умолчанию имеет две кнопки: Cancel и OK.



Кнопка Cancel останавливает дальнейшее выполнение сценария. Так как вышеописанный сценарий не имеет каких-либо других операторов, кнопка Cancel здесь просто лишняя. Давайте изучим это поближе и определим кнопки диалогового окна сами. Команда «display dialog» позволяет задать список кнопок. В нашем случае, нам нужна лишь одна, кнопка, которую мы хотим увидеть как «OK» [2.2].

```
set stringToBeDisplayed to "Julia is a pretty actress."                    [2]
display dialog stringToBeDisplayed buttons {"OK"}
```

Если вы теперь запустите сценарий, то вы увидите, что кнопка Cancel исчезла.



Как показано во втором операторе [2.2], список содержит строку "OK", помещенную между фигурными скобками. Откуда здесь эти фигурные скобки? Как мы видели ранее, AppleScript'у нужны ключи, помогающие ему понять значение каждого элемента в операторе сценария. Чтобы помочь AppleScript распознать список (list), тоже необходим ключ, состоящий из фигурных скобок.

Список в операторе [2.2] содержит только один элемент, строку "OK". Если список содержит больше элементов, они разделяются запятыми [3] .

```
set exampleList to {213.1, 6, "Julia, the actress", 8.5}
```
[3]

Список в операторе [3] содержит 4 элемента: одна строка и три числа. Давайте вернемся к команде «display dialog» и создадим диалоговое окно с несколькими кнопками. Команда AppleScript «display dialog» позволяет задать одну, две или три кнопки, с (уже совсем скоро) текстом на ваш собственный выбор. Так, для того чтобы создать диалоговое окно с тремя кнопками, мы определяем список с тремя элементами [4.2].

```
set stringToBeDisplayed to "Julia is a pretty actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don't know", "Yes"}
```
[4]

Вы заметили, что нет выделенных кнопок, когда вы их определяете сами [2.2, 4.2] Это значит, что человек, который запустит сценарий, не сможет нажать клавишу Enter, чтобы убрать диалоговое окно. Так как этот человек Мак-юзер, который ценит дружественность интерфейса, то мы теперь позаботимся о выделенной кнопке [5].

```
set stringToBeDisplayed to "Julia is a pretty actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don't know", "Yes"} default button "Yes"
```
[5]

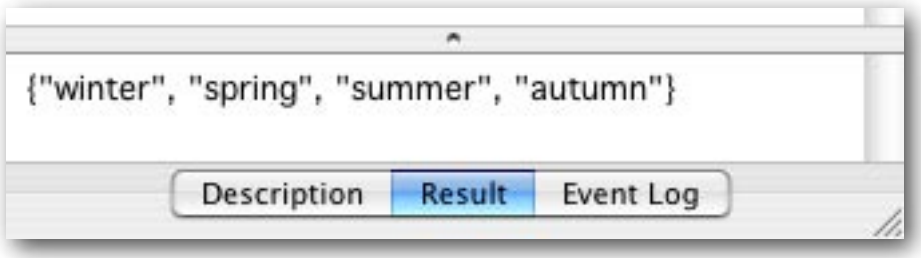
Вместо написания имени кнопки выделяемой из всех, вы можете указать номер кнопки [6], т.е. третий элемент в данном списке.

```
set stringToBeDisplayed to "Julia is a pretty actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don't know", "Yes"} default button 3
```
[6]

В следующей главе вы научитесь определять какая кнопка была нажата. А пока мы продолжим изучать списки. Списки могут быть использованы для хранения серий данных. Поэтому вам нужно будет знать как редактировать список и извлекать данные из списка. Самое простое – добавлять данные в начало или конец списка. Чтобы добавить элементы к списку мы используем амперсанд, подобно тому как было в строках.

```
set addToBegin to {"winter"}
set addToEnd to {"summer", "autumn"}
set currentList to {"spring"}
set modifiedList to addToBegin & currentList & addToEnd
```
[7]

В операторе [7.4] мы создали список состоящий из четырех элементов. Поле результата покажет фигурные скобки – характерная особенность типа данных «list» (список).



Обратите внимание, что «addToBegin» и «addToEnd» просто имена переменных [7.1–7.2], выбраны для того, чтобы помочь вам понять сценарий и они не делают ничего, кроме того что ссылаются на данные. Зеленый цвет точно указывает их принадлежность к именам переменных.

Вы можете обратится к каждому элементу (item) в списке по номеру. Самый левый элемент является элементом 1, следующий – 2 и т.д. Это позволяет вам получить соответствующее значение из списка, или изменить значение (такое как строка или число). Вот пример [8].

```
set myList to {"winter", "summer"}
set item 2 of myList to "spring"                                    [8]
get myList
```

Команда «get» в последнем операторе [8.3] позволяет нам показать значение переменной myList в поле результата. Это поле покажет значение переменной myList как {"winter", "spring"}.

Сфокусируйте внимание на втором операторе [8.2], такого же результата можно добиться любым из следующих двух операторов в сценарии [9], как у второго оператора в сценарии [8].

```
set the second item of myList to "spring"
set the 2nd item of myList to "spring"                              [9]
```

Первый оператор [9.1] изящно демонстрирует англоязычную натуру AppleScript. Эта словесная нумерация работает до десятого элемента. После десятого, вы должны прибегнуть к «item 11», «item 12» и т.д. В качестве альтернативы, вы можете написать «11th item» и т.д. аналогично [9.2]. Кроме ссылки по словесно нумерованному списку элементов, есть также и «last» item (последний элемент) [10].

```
set myList to {"winter", "summer"}
set valueOfLastItem to the last item of myList                      [10]
```

Так что, вам не надо помнить какой длины список, чтобы получить значение последнего элемента списка (или установить его на другое значение).

AppleScript позволяет вам обращаться к элементам путем отсчитывания их в обратном направлении, т.е. справа налево. Чтобы отсчитывать с конца, используйте отрицательные числа, где -1 будет последним значением, значение -2 – предпоследним, и т.д. Сценарий [11] выдаст в точности такой же результат, что и сценарий [10].

```
set myList to {"winter", "summer"}
set valueOfLastItem to item -1 of myList                            [11]
```

Теперь вы знаете, как создать список, как добавить элементов к нему и как изменить значения элементов списка. Вы также знаете как получать одиночное значение из списка. Вы, возможно, еще хотите узнать как создать список из части другого списка [12].

```
set myList to {"a", "b", "c", "d", "e", "f", "g", "h"}              [12]
set shortList to items 2 through 5 of myList
```

В операторе [12.2], вы можете использовать «thru» вместо «through». Если вы измените порядок номеров элементов [13.2] при указании диапазона, ваш список не будет перевернут [13].

```
set myList to {"a", "b", "c", "d", "e", "f", "g", "h"}              [13]
set shortList to items 5 through 2 of myList
```

Так что результат сценария [13] будет в точности такой же, что и у сценария [12].

Чтобы перевернуть порядок элементов в списке, в вашем распоряжении имеется команда «reverse» [14].

```
set reversedList to reverse of {3, 2, 1}
```
[14]

Иногда, вам надо будет знать какой длины ваш список. Ответ легко получить [15], используя один из двух следующих операторов.

```
set theListLength to the length of {"first", "last"}
set theListLength to the count of {"first", "last"}
```
[15]

И наконец, вы можете обратиться к случайному элементу [16].

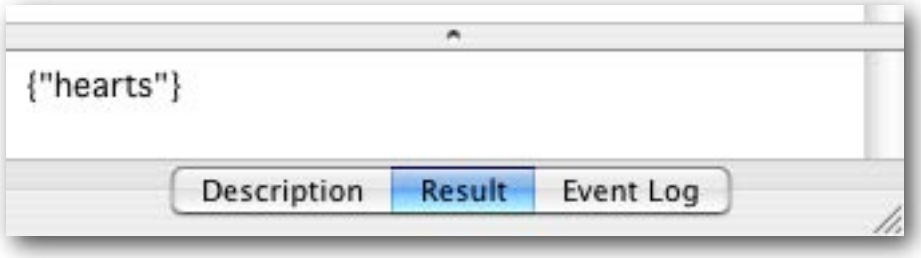
```
set x to some item of {"hearts", "clubs", "spades", "diamonds"}
```
[16]

Я знаю, эта глава длинная, но мы действительно должны обсудить некоторые вопросы, имеющие место быть со строками и списками. Мы не обсуждали их в главе 5, где рассказывалось о строках, потому что мы не говорили еще о списках. Так что держитесь или сделайте сначала небольшой перерыв.

В предыдущих главах вы узнали, что возможно преобразовать один тип данных в другой. Сейчас мы покажем вам как превратить строку (или число) в список [17].

```
set cardType to "hearts"
set stringAsList to cardType as list
```
[17]

После преобразования строки в список, в поле результата будет список содержащий один единственный элемент – строку.



Когда приходиться работать со списками и строками вместе, преобразование является важной мерой безопасности, как ниже будет показано.

Как вы помните, амперсанд использовался для объединения строк. Что случится если вы используете амперсанд для добавления строки к списку [18]?

```
set myList to {"a"}
set myString to "b"                                                    [18]
set theResult to myList & myString
```

Тип данных переменной theResult зависит от типа данных, который встретится первым в вычисляемом выражении [18.3]. Так как выражение начинается с переменной myList, которое есть список, результат тоже будет списком. Попробуйте сделать это сами и проверить поле результата. Оно откроет вам тип данных результата, показав фигурные скобки. Если мы изменим порядок следования имен переменных myList и myString, и запишем [19.3]:

```
set myList to {"a"}
set myString to "b"                                                    [19]
set theResult to myString & myList
```

то результатом будет строка. Этот вариант проходит, но при этом не говорится, что если вы не обратите особого внимания на такие изменения, то они легко приведут к ошибкам в вашем сценарии. Чтобы предотвратить появления чего-либо неожиданного, сделайте преобразование [20].

```
set myList to {"a"}
set myString to "b"                                                    [20]
set theResult to (myString as list) & myList
```

В операторе [20.3], строка "b", содержащаяся в переменной myString, преобразуется в список {"b"}, независимо от типа данных содержащегося в «myString». Так как переменная «myString», теперь содержит список, theResult тоже будет списком [20.3].

Для того, чтобы добавить один или несколько элементов к списку, без конкатенации, вы можете написать:

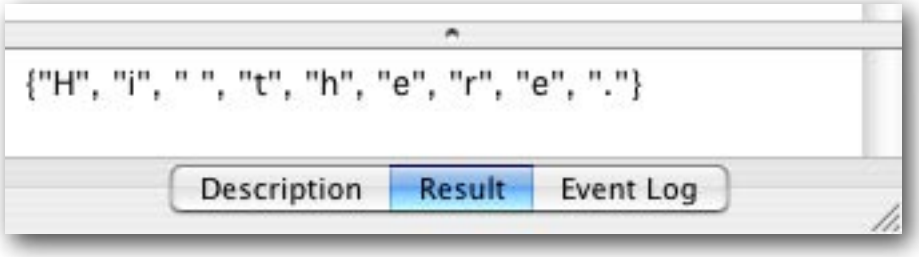
```
set myList to {1, 2, 3, 4}
set the end of myList to 5
get myList
```

Так, поверьте, будет быстрее.

Кроме преобразования, превращающего строку в список, можно создать список, содержащий все символы строки [21]. (Примечание: хотя использование «item» вместо «character» будет работать, этого не рекомендуется делать под Mac OS X).

```
set itemized to every character of "Hi there."      [21]
```

Полученный список, видимый в поле результата, похож на:



Вместо того, чтобы разбивать фразу на одиночные символы, вы скорее захотите поделить предложение на слова. Это можно сделать используя AppleScript's text item delimiters (разделители элементов слов AppleScript'а. Вы указываете символ, который будет служить как разделитель элементов, которые в итоге составят список. Для того, чтобы разрезать предложение на слова, разделителем должен быть пробел. Посмотрите сценарий [22]. Хорошим тоном при написании сценариев считается следующее: если вы изменили AppleScript's text item delimiters [22.3], то измените их назад сразу после использования [22.5].

```
set myString to "Hi there."
set oldDelimiters to AppleScript's text item delimiters
set AppleScript's text item delimiters to " "
set myList to every text item of myString
set AppleScript's text item delimiters to oldDelimiters
get myList
```
[22]

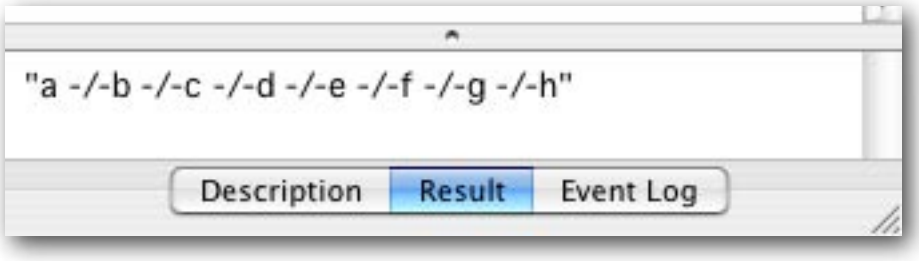
Обратите особое внимание на оператор [22.4], где говорится «text item», а не просто «item». Это часто совершаемая ошибка. text item text item text item. Запомнили?

Легко преобразовать список в строку [23].

```
set myList to {"a", "b", "c", "d", "e", "f", "g", "h"}
set myList to myList as string
```
[23]

Если вам нужно, чтобы в строке отдельный символ или серия символов разделяли исходные элементы списка, вы должны установить AppleScript's text item delimiters соответствующим образом [24].

```
set myList to {"a", "b", "c", "d", "e", "f", "g", "h"}
set oldDelimiters to AppleScript's text item delimiters
set AppleScript's text item delimiters to " -/-"
set myList to myList as string
set AppleScript's text item delimiters to oldDelimiters
get myList
```
[24]



Объединяем выученное в сценарии [22] и сценарии [24]. Теперь у вас есть возможность выполнять операции поиска и замены в тексте.

Причина, почему вы должны восстанавливать предыдущее значение AppleScript's text item delimiters, кроется в том, что другие авторы сценариев могут быть менее осторожны. Их сценарий может предполагать использование особого значения AppleScript's text item delimiters. Изменение вносимое вашим сценарием запоминается в AppleScript (компоненте MacOS X) даже после завершения вашего сценария.

Подведем итог: есть несколько типов данных, такие как число, строка и список. Каждый тип данных имеет свои собственные операторы, которые вы можете использовать для выполнения различных операций с вашими данными. Много разных операторов служит для манипуляций списками. В этой главе, посвященной спискам, мы так же узнали немного нового о строках.

ГЛАВА 7

ЗАПИСИ

В предыдущей главе, мы использовали команду «display dialog» для ознакомления с новым типом данных – списком. Теперь мы ее используем для изучения другого типа данных.

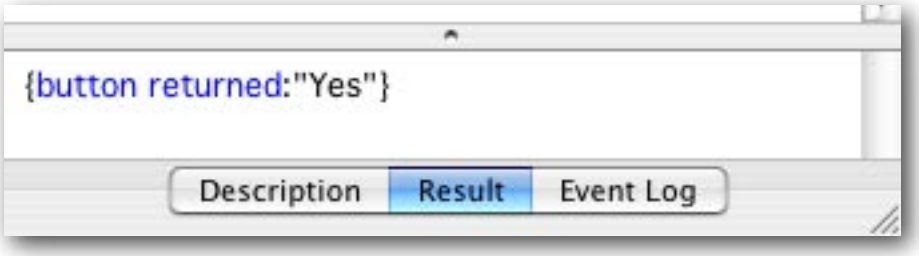
Команда «display dialog» позволяет вам задать число кнопок, предоставив список от одной до трех строк [1.2].

```
set stringToBeDisplayed to "Julia  is a pretty  actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don't know", "Yes"}          [1]
```

Но как узнать, какая кнопка была нажата? Посмотрите сценарий [2]

```
set stringToBeDisplayed to "Julia  is a pretty  actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Don't know", "Yes"}    [2]
```

Когда вы выполните сценарий [2], вы увидите результат в поле результата. В зависимости от того какую вы нажали кнопку, он будет выглядеть подобно этому:



Этот результат представляет собою уже другой тип данных, называемый записью (record). Подобно списку, он заключается в фигурные скобки, но в отличие от списка каждый элемент состоит из двух частей, разделенных двоеточием. Элемент записи называется свойством (property), а также часто называется как пара имя/значение. Первая часть свойства это метка (label) или имя (в нашем примере «button returned»), а вторая часть – само значение этого свойства (здесь значение – строка "Yes"). Так как вторая часть является значением, то редактор Script Editor показывает её черным.

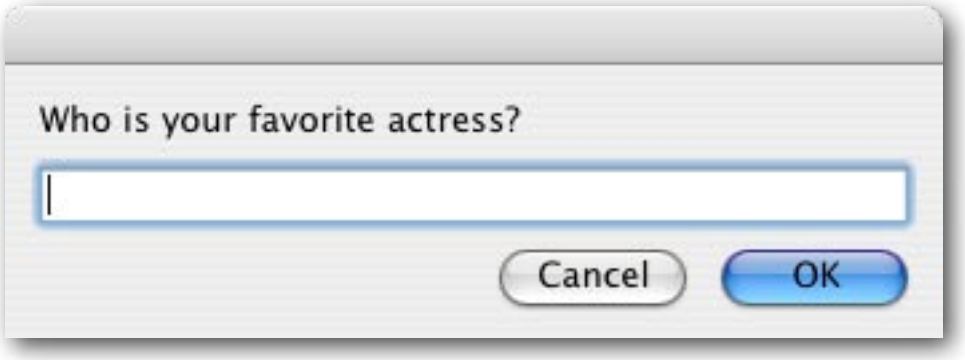
Чтобы найти которая из кнопок была нажата, все что мы должны сделать, это потребовать значение свойства имеющего метку «button returned» [3.3].

```
set stringToBeDisplayed to "Julia  is a pretty  actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Don't know", "Yes"}    [3]
set theButtonPressed to button returned of tempVar
display dialog "You pressed the following button " & theButtonPressed
```

В операторе [3.3] мы указываем, что хотим установить переменную «theButtonPressed», на значение содержащееся в свойстве с меткой «button returned» записи «tempVar». И мы это делаем! Теперь мы знаем какая кнопка диалогового окна была нажата.

Диалоговое окно ограничено тем, что показывает только числа и (короткие) строки. Оно не может показывать списки и записи. А вот поле результата показывает все типы данных, как мы и видели до сих

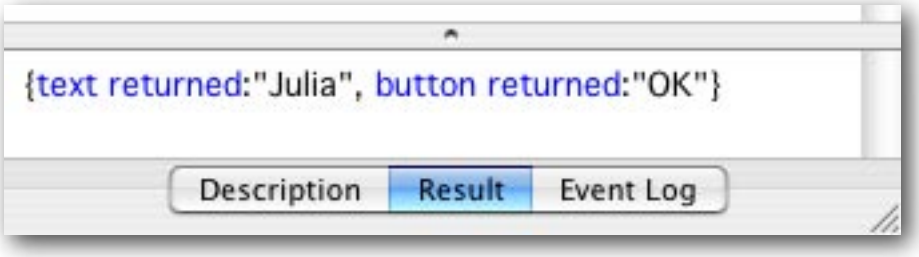
пор. Не смотря на это, «display dialog» может сделать кое-что очень полезное: он позволяет пользователю ввести числа или текст, которые ваш сценарий может потом обработать.



Чтобы появилась строка ввода, вы должны задать ответ по умолчанию (default answer) как строку, например как пустую строку "" [4].

```
set temp to display dialog "Who is your favorite actress?" default answer ""          [4]
```

Если вы запустите сценарий [4], результатом будет запись из двух значений, т.е. две пары имя/значение, такая запись может выглядеть как эта в поле результата (в зависимости от того, что ввел пользователь):



Обратите внимание, подобно элементам списка, свойства разделены запятой. AppleScript не спутает запись со списком, потому что там есть двоеточия. В отличие от списка, где вам надо помнить какой порядковый номер элемента какую часть информации содержит, фактически данные могут быть извлечены из записи указанием метки свойства, что куда проще. Чтобы извлечь имя актрисы, все что мы должны сделать, это потребовать значение свойства имеющего метку «text returned» [5.2].

```
set temp to display dialog "Who is your favorite actress?" default answer ""          [5]
set textEntered to text returned of temp
```

Пожалуйста, обратите внимание, что значение text returned будет строкой, и всегда будет строкой, даже если пользователь ввел число. Например, если пользователь ввел 30, значение переменной textEntered будет не 30, а "30". Если вы захотите выполнить вычисления с введенными данными, то вам повезло. AppleScript попытается преобразовать строку в число автоматически [6.3]. Так что, вам не обязательно включать опреатор преобразования [7.1] после оператора [6.2].

```
set temp to display dialog "What is your age?" default answer ""
set ageEntered to text returned of temp                                              [6]
set ageInMonths to ageEntered * 12
display dialog "Your age in months is" & ageInMonths


set ageEntered to ageEntered as number                                               [7]
```

Преобразование будет работать если пользователь ввел число, такое как 30. Но если пользователь ввел «тридцать» или «30 лет», AppleScript не сможет выполнить преобразование, и в сценарии возникнет сбой. Так как нет надежды на то, что пользователь сделает именно все так, как было задумано вами, вам придется научиться писать сценарии, которые будут учитывать разные случаи поведения пользователя. В главе 10 вы узнаете как разрешить такую задачу.

Вы можете создать свою собственную запись, присвоив переменной пару имя/значение [8].

```
set personalData to {age:30}
```
[8]

Обратите внимание, что цвет свойства «age» зеленый, т.е. описанный, определенный вами. Как обычно, данные показаны черным.

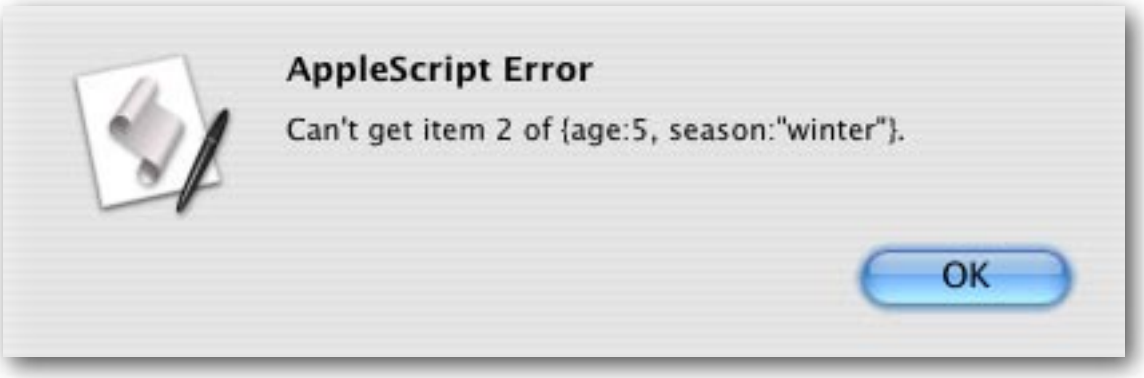
Возвращаясь к сценариям [3, 5], пожалуйста, обратите внимание, что метка свойства «button returned» и «text returned» являются зарезервированными словами AppleScript. Вы видите, их цвет синий, но их еще отличает и то, что они состоят из двух слов. Если вы создаете свои записи, вам не позволено использовать два или более слов для идентификации свойства [9.1]. Метка (или имя) свойства должна быть одним словом. [9.2]

```
set improperlyNamedProperty to {my property: "This is not correct"}
set properlyNamedProperty to {myproperty:"This  is  correct"}
```
[9]

Вы можете объединять записи как списки, но будьте осторожны: если записи, которые вы объединяете, имеют одинаковые метки свойств, то результат может отличаться от ожидаемого вами.

Пожалуйста, не называйте свойства элементами (items) записи, потому что, хоть следующий сценарий [10] и проходит синтаксическую проверку, AppleScript не позволяет ссылаться на пары имя/значение в записи как на элемент [10.2].

```
set longRecord to {age:5, season:"winter"}
set cantGetARecordLikeThis to item 2 of longRecord
```
[10]



Тем не менее, вы можете подсчитать сколько свойств в записи [11]:

```
set longRecord to {age:5, season:"winter"}
set theNoOfProperties to the count of longRecord
```
[11]

Чтобы создать новую запись, содержащую свойство другой записи, вы должны создать запись так как показано ниже [12].

```
set longRecord to {age:5, season:"winter"}
set temp to the age of longRecord
set newRecord to {age:temp}
```
[12]

Поле результата покажет новую запись, которая будет {age:5}. Возможно написать сценарий [12] более кратко, но может быть вы поначалу найдете его более тяжело читаемым [13].

```
set longRecord to {age:5, season:"winter"}                    [13]
set newRecord to {age:age of longRecord}
```

К несчастью, невозможно определить которая пара имя/значение присутствует в записи. Вот почему вы не сможете создать список из меток всех свойств. Так же, невозможно изменить метки в записи. Если вы хотите использовать другую метку, вы должны создать запись заново, как показано в сценарии [13].

Давайте закончим эту главу одним неприятным подводным камнем. Вот сценарий, в котором нет каких-либо сюрпризов [14].

```
set firstValue to 30
set rememberFirstValue to firstValue -- копия сохраняется в «rememberFirstValue».   [14]
set firstValue to 73 -- изменяем значение исходной переменной.
get rememberFirstValue -- запрашиваем значение «rememberFirstValue».
```

Результат – 30. Для записей (и списков!) этот порядок действий совершенно другой, что в результате может привести к очень тяжелому отлову ошибок. Посмотрите на сценарий [15]

```
set personalData to {age:30}
set rememberPersonalData to personalData                      [15]
set age of personalData to 73
get rememberPersonalData
```

Результат будет {age:73}! ! ! Команда set никогда не создаст копию, если переменная содержит запись или список. Чтобы быть уверенным в том, что данные скопировались, вы должны использовать команду copy [16].

```
set personalData to {age:30}
copy personalData to rememberPersonalData                     [16]
set age of personalData to 73
get rememberPersonalData
```

(Команда set, вместо копирования записи или списка, устанавливает значение переменной на ссылку на эту запись или список. Это очень полезно при обработке больших записей и списков и может сэкономить много вычислительных ресурсов – прим. авт.).

ГЛАВА 8

УПРОЩЕНИЕ СОЗДАНИЯ СЦЕНАРИЕВ (II)

В предыдущей главе мы видели рассмотрели способы отображения диалогового окна. Если вам кажется, что вы запомнили все опции, милости просим. Если вы предпочитаете более легкий способ, просто запомните, что клик мышкой с нажатым control в верхнем поле редактора Script Editor, вызовет контекстное меню. Одним из первых элементов меню идет «Dialogs». Щелчок на нем, и перед вами раскроется следующее подменю:

Где, слово «Btn» означает кнопка (button). Цифра перед ним обозначает число кнопок. Пока забудьте об элементах меню со словом «Actions». Вы освоите их позже, когда мы достигнем главы 10.

Последние три элемента меню создают диалоги, которые позволяют пользователю вводить текст. В редакторе Script Editor, напишите следующее:

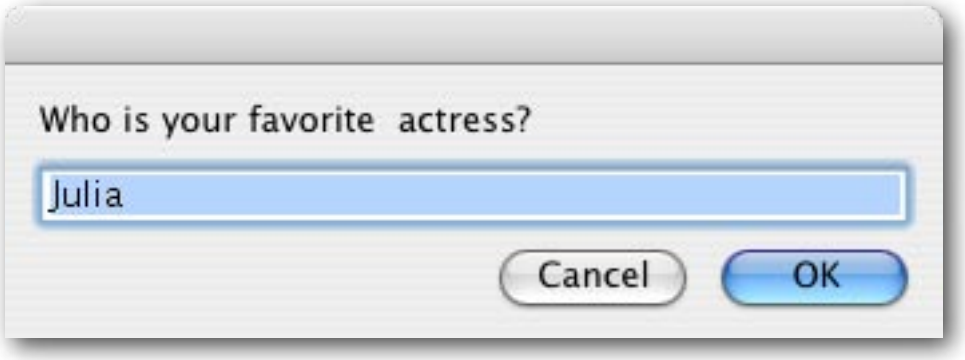
```
set temp to
```

Убедитесь в наличии пробела после «to». Теперь, кликните с нажатым control справа после пробела, и выберите «Text Input – 2 Btns». Будет создан оператор необходимый, чтобы предоставить пользователю поле, в которое он или она смогут ввести данные, как обсуждалось в главе 7, и где говорилось о свойстве «text returned».

Если вы хотите, display dialog может предоставить ответ по умолчанию (default answer), и пользователь сценария может изменить его на что-нибудь еще, при необходимости. Ваш сценарий будет более дружественным пользователю, если вы предложите ответ по умолчанию похожий на вероятный ответ пользователя [1.1]. И вы, конечно, знаете, что пользователи Макинтош очень любят дружественный интерфейс. Даже если ответ по умолчанию не годится как ответ, то он все еще дает подсказку о том какой тип ответа ожидается.

```
set temp to display dialog "Who is your favorite  actress?" default answer "Julia"     [1]
```

Запустите сценарий [1], и вам будет представлено следующее диалоговое окно. Если вы согласны, просто нажмите Enter. В противном случае, вы можете написать другое имя.



Короче, нет необходимости запоминать точные команды для всех возможных воплощений команды «display dialog». Редактор Script Editor сделает это за вас, и вы сможете добавлять их к вашему сценарию, не набирая текста.

ГЛАВА 9

НЕТ КОММЕНТАРИЕВ? НЕДОПУСТИМО!

Есть несколько факторов, которые помогают сделать AppleScripts более легким в чтении, написании и поддержке. Одни находятся за пределами вашего влияния, например такое как родство этого языка с натуральным английским языком. Зато другие факторы всегда в вашем распоряжении, как например, использование описательных имен переменных. В этой главе мы обсудим еще один важный фактор.

Пока мы ограничивались примерами в несколько строк длиной, но сценарии, которые вы напишите будут становиться все длиннее и длиннее. Исключительно важно при написании сценариев не просто побеспокоиться о том, чтобы ваш сценарий действительно делал то что вы хотите, но и чтобы он был правильно продокументирован. Позднее, если вы некоторое время не будете видеть перед собой сценарий, и вдруг захотите его отредактировать, вам действительно сильно понадобятся комментарии, чтобы помочь сразу понять какая часть скрипта за что отвечала и почему у этих частей такой порядок следования. Весьма благоразумно потратить некоторое время на комментирование вашего сценария. Мы можем вас уверить, что неоднократно потраченное время окупиться в будущем. Также, если вы используете ваш сценарий совместно с кем-нибудь еще, другой человек будет в состоянии быстро переделать сценарий, если он снабжен комментариями.

Чтобы создать комментарий, начните его с двух дефисов (минусов).

```
-- Это комментарий
```

После компиляции (проверки синтаксиса), комментарий показывается серым цветом.

```
-- И это комментарий
-- Этот комментарий занимает больше чем
        две строки
```

Иногда, многострочные комментарии заключаются между скобками (* *)

```
(* Этот комментарий растянут
на две строки.*)
```

С появлением контекстного меню, этого уже не надо делать вручную.

Заключая часть вашего сценария между кавычками (* *), вы можете временно отключить («закомментировать») часть сценария, чтобы посмотреть продолжит ли он работать так как положено. Это позволяет отлавливать ошибки. Если закомментированная часть сценария должна приводить к какому-то результату, например, к присваиванию значения некоторой переменной, включите временную строку, где устанавливается переменная на значение соответствующее для тестирования оставшегося сценария.

Контекстное меню редактора Script Editor позволяет легко что-нибудь закомментировать. Если вы хотите закомментировать часть вашего сценария, выделите эту часть, например протягиванием мыши. Затем щелкните с нажатым control в верхнем поле и выберите «Comment Tags». Попробуйте добавить или удалить теги комментария в выбранном тексте. Первое выполнение процедуры «Comment Tags» заключит выделенный текст в теги комментария, при втором – в выделенном тексте они будут удалены. Если ваши пояснения были помещены после двух дефисов, то эта процедура их не затронет.

В Script Editor, щелкните на закладку «Description» (описание). В раскрывшемся нижнем поле вы можете поместить общее пояснение о назначении вашего сценария. Дополнительно, в качестве альтернативы, начните верхнюю часть вашего сценария с пояснения (используя двойной дефис).

Важность комментариев трудно переоценить. Сценарии в этой книге не содержат так много комментариев, как бывает обычно, потому что сценарии в книге итак уже окружены со всех сторон пояснениями.

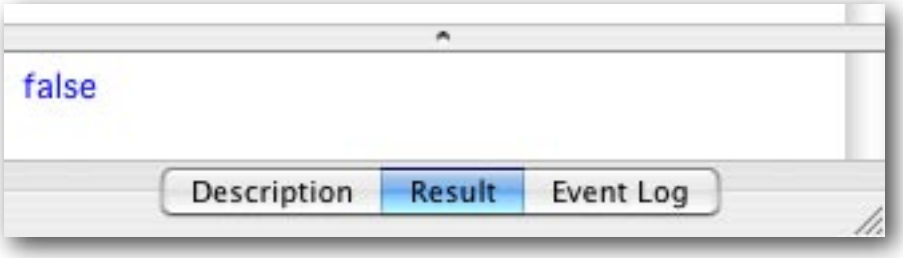
ГЛАВА 10

УСЛОВНЫЕ ОПЕРАТОРЫ

Иногда необходимо, чтобы сценарий выполнил некоторую серию действий, только если выполняется определенное условие. Наберите следующую строку [1] в Script Editor и нажмите Run.

```
73 = 30
```
[1]

Вот что вы увидите в поле результата:



AppleScript вычисляет сравнение в сценарии [1] и заключает, что это false (ложь), т.е. не правда. Если вы введете что-то подобное «30 = 30», вы увидите результат «true».

Эта способность AppleScript сравнивать два значения используется в операторе «if...then» (если... то) [2] для того, чтобы выполнить операторы только в том случае, если условие соблюдается. Оператор «if...then» называется условным оператором. Обратите внимание, что этот оператор требует наличия оператора «end if» (окончание если) в конце. Но об этом позже.

```
if true then
    -- действия выполнятся
end if
```
[2]

```
if false then
    -- эти действия не выполнятся
end if
```

Итак, все что нам нужно сделать, это заменить слово «true» в операторе [2.1] сравнением. Если это сравнение окажется верным («true»), то операторы в строке [2.2] будут выполнены.

```
set ageEntered to 73
set myAge to 30
if ageEntered is myAge then
    display dialog "You are as old as I am."
end if
```
[3]

Если сравнение ageEntered is myAge [3.3] было бы истинным, то было бы показано диалоговое окно. С указанными выше значениями переменных в первых двух операторах, вы не увидите этого окна [3.4].

Если нужно определить больше инструкций, которые должны будут выполниться при верном условии, они все их надлежит расположить внутри блока оператора «if...then...end if» [4].

```
set ageEntered to 73
set myAge to 30
if ageEntered is myAge then
    display dialog "You are as old as I am."                    [4]
    beep
end if
say "This sentence is spoken anyway."
```

Вы заметили сходство между блоком обращения и оператором «if...then»? В обоих случаях необходимо, чтобы оператор содержал слово «end». «end if» позволяет AppleScript определить какие операторы будут выполняться если сравнение вернет «true». В сценарии [4], вы услышите последний оператор [4.7] независимо от того будет условие [4.3] выполняться или нет.

Мы можем также предложить альтернативный набор инструкций в том случае, если условие не выполниться, используя оператор «if...then...else» [5]

```
set ageEntered to 73
set myAge to 30
if ageEntered = myAge then
    display dialog "You are as old as I am."                    [5]
else
    display dialog "You are not as old as I am." -- [5.6]
end if
```

Здесь, диалог оператора [5.6] будет отображен, потому что сравнение в операторе [5.3] выдаст «false» (ложь). Вы можете выполнять очень много разных сравнений. Эти операторы сравнения подходят и для других типов данных.

Кроме знака равно в операторе [5.3], в вашем распоряжении следующие операторы сравнения для чисел.

Для чисел

=	is, is equal to	есть, равно
>	is greater then	больше чем
<	is less then	меньше чем
>=	is greater then or equal to	больше или равно
<=	is less then or equal to	меньше или равно

Вторая колонка не просто показывает расшифровку символов первой колонки, вы также можете записывать сравнения в их устной форме! [6]

```
if a is greater than b then
    display dialog "a is larger"                                [6]
end if
```

Если набрать >= и выполнить проверку синтаксиса, AppleScript автоматически конвертирует этот оператор сравнения в служебный символ ≥. Аналогично, оператор сравнения «меньше или равно» <= будет заменен на ≤. Вы должны набирать символы в правильном порядке, иначе AppleScript будет выражать недовольство. Так что, он не всегда настолько дружественен к пользователю, как мог бы быть. (Чтобы ввести с клавиатуры символы ≤ и ≥, удерживайте option и нажмите знак меньше или больше соответственно. – прим. пер.)

Отрицательные формулировки также допустимы, например: «is not equal than» (не равно). Если набрать /=, оно автоматически будет переформатировано при компиляции в ≠, который для краткости читается как «is not» (Чтобы ввести с клавиатуры символ ≠, удерживайте option и нажмите знак равно. – прим. пер.)

В главе 7 мы столкнулись со следующим сценарием [7], который позволил нам определить, какая кнопка была нажата.

```
set stringToBeDisplayed to "Julia is a pretty actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Yes"}          [7]
set theButtonPressed to button returned of tempVar
```

Используя оператор «if...then», мы можем выполнить желаемые действия [8].

```
set stringToBeDisplayed to "Julia is a pretty actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Yes"}
set theButtonPressed to button returned of tempVar
if theButtonPressed is "Yes" then                                                    [8]
    say "I agree entirely!"
else
    say "Didn't you see the movie 'Pretty Woman'?"
end if
beep
```

Если пользователь нажимает кнопку «Yes», тогда фраза оператора [8.5] проговаривается. В любом случае, сценарий продолжится и мы услышим сигнал beep последнего оператора.

В предыдущем сценарии, значения строк должны быть идентичны. Однако, AppleScript в состоянии сделать намного больше, чем это. Здесь три примера [9]. Обратите внимание, что «end if» не требуются, если выполняемая инструкция находится на той же строке, что и оператор «if...then».

```
set textString to "Julia is a beautiful actress."
if textString begins with "Julia" then display dialog "The first word is Julia"      [9]
if textString does not start with "Julia" then beep
if textString contains "beau" then set myVar to 5
```

Ниже дан обзор операторов сравнения для строк.

Для строк	
begins with (или starts with)	начинается с
ends with	оканчивается на
is equal to	равно
comes before	идет перед
comes after	идет после
is in	находится внутри
contains	содержит

Можно также сформулировать их как негативные.

does not start with	не начинается с
does not contain	не содержит
is not in	не находится внутри
и т.д.	

Если написать «doesn't», то оно автоматически переформатируется на «does not».

Если написать «does not begin with», то выражение автоматически переформатируется на «does not start with».

Операторы сравнения «comes before» и «comes after» работают с алфавитным порядком. Так что, результатом следующего оператора будет сигнал beep [10].

```
if "Steve" comes after "Jobs" then
    beep
end if
```
[10]

Когда сравниваются строки, то регистр может быть важным для вас. Просто дайте знать об этом AppleScript'у. И конечно, когда закончите сравнение, не забудьте это указание выключить [11].

```
set string1 to "j"
set string2 to "Steve Jobs"
considering case
    if string1 is in string2 then
          display dialog "String2 contains a \"j\""
    else
          display dialog "String2 does not contain a \"j\""
    end if
end considering
```
[11]

По умолчанию, пробельное пространство (пробел, переход на другую строку/return, табуляция) учитывается. Если вы этого не хотите, наберите «ignoring white space» (игнорировать пробельное пространство), как продемонстрировано в сценарии [12]. Пожалуйста, обратите внимание, что вы должны добавить оператор «end ignoring» или «end considering».

```
set string1 to "Stev e Jobs"
set string2 to "Steve Jobs"
ignoring white space
    if string1 = string2 then beep
end ignoring
```
[12]

Вы можете также заставить AppleScript игнорировать пунктуацию или диакритические знаки.

Для типа данных list, доступно меньше операторов сравнения, чем для строк, но с другой стороны, они такие же. Так что, вам не придется заучивать больше операторов сравнения, чем необходимо.

Для списков	
begins with	начинается с
ends with	оканчивается на
contains	содержит
is equal to	равно
is in	находится внутри

Часто бывает, что сравниваются индивидуальные элементы списка (или разных списков). Давайте взглянем на практический пример. Посмотрите на сценарий [8]. Что если у нас три кнопки в нашем диалоге? С помощью операторов вложенных в «if...then» [13], все опции диалога будут учтены.

```
set stringToBeDisplayed to "Julia is a pretty actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Who?", "Yes"}
set theButtonPressed to button returned of tempVar
if theButtonPressed is "Yes" then
    say "I agree entirely!"
    beep
else
    if theButtonPressed is "Who?" then
          say "Didn't you see the movie 'Pretty Woman'?" -- [4.8]
    else
          say "I don't agree with you."
    end if
end if
```
[13]

Как видите, отступы облегчают чтение сценария, но это, тем не менее, немножко неудобно для чтения. Плюс к этому, когда много текста легко можно забыть или поставить не туда оператор «end if», в результате чего ваш сценарий не скомпилируется. Контекстное меню редактора Script Editor спасает! Вспомнили элементы меню Dialog со словом Actions?

Щелкните одну с 3-я кнопками (3 Btns) и 3-я действиями (Actions) и вам будет представлен полный набор необходимых операторов. Вам просто нужно заполнить значениями список [14.1] и впечатать необходимые действия.

```
display dialog "" buttons {"", "", ""} default button 3
set the button_pressed to the button returned of the result
if the button_pressed is "" then
    -- здесь идут действия для первой кнопки
else if the button_pressed is "" then                                    [14]
    -- здесь идут действия для второй кнопки
else
    -- здесь идут действия для третьей кнопки
end if
```

Первые две строки, возможно, нуждаются в объяснении. Как видите, оператор [14.1] отличается от оператора [8.2], в котором мы не присваиваем какой-либо переменной результирующую запись команды display dialog. Просто, подобно полю результата, автоматически показывающего результат последнего выполненного оператора, результат предыдущего оператора будет доступен вам, если вы сошлетесь на него используя ключевое слово «result». При расположении оператора if на той же строке что и «else» [14.5], разделитель «end if» не требуется. В итоге, этот сценарий легче читается.

Для записей количество операторов сравнения еще более ограничено, чем для списков.

Для записей	
contains	содержит
is equal to (или =)	равно

```
set x to {name:"Julia", occupation:"actress"}
if x contains {name:"Julia"} then display dialog "Yes"                   [15]
```

Обратили внимание, что в операторе [15.1] одна метка показана синим цветом, а другая зеленым? Таким образом вы предупреждены, что одна метка является ключевым словом. Несмотря на то, что AppleScript не запрещает вам использовать метки свойств идентичных зарезервированным ключевым словам, таких как «to», «set», «string» и т.д., это может позже привести вас к проблемам, так что остерегайтесь использования зарезервированных ключевых слов, и помните подсказки в главе 4 об именах переменных.

Ограниченное количество операторов сравнения для записей не проблема, потому что вы будете сравнивать не полные записи, а только индивидуальные значения свойств этих записей [16].

```
set aRecord to {name:"Julia", occupation:"actress"}
if name of aRecord is "Julia" then display dialog "OK"                    [16]
```

Как мы видели в первой части этой главы, если сравнивать два значения (каков бы ни был их тип данных), вы на самом деле пытаетесь увидеть будет ли выражение сравнения истиной или ложью (true или false). И действительно, они представлены специальным типом данных, называемом – Boolean (булевой, логический тип). Переменные хранящие значение этого типа могут иметь только первое или второе значение: true или false. Для чисел, у вас есть операторы «+» и «–». Если вы используете такой оператор, результат будет числом. Для булевых операторов, мы имеем «and», «or» и «not». Результат этих операций опять же будет Boolean.

Простейшее из тех трех – «not». Например, «not true» будет ложь, и «not false» будет истина.

```
set x to not true -- Значит, переменной x присвоить ложь                  [17]
```

Как продемонстрировано в следующем сценарии [18], для оператора «and», оба x и y должны быть истиной, чтобы z было равно true.

```
set x to true
set y to true                                                             [18]
set z to (x and y) -- z равно true
```

И наоборот, для оператора «or», будет достаточно, чтобы хоть одно из x и y было true [19].

```
set x to true
set y to false                                                            [19]
set z to (x or y) -- z равно true
```

Зачем я все это говорю вам? Ну, при некоторых обстоятельствах вам понадобиться выполнить набор инструкций только если несколько условий выполнится. Следующий сценарий [20] вызывает диалог только последнего оператора.

```
set x to 5
set y to 7
set z to "Julia"
if x = 5 and y = 6 then display dialog "Both conditions met."             [20]
if x = 5 or z = "actress" then display dialog "At least one condition met."
```

В операторе [20.4] первое сравнение истинно, а второе – нет. Для «and» требуется, чтобы бы оба операнда были истинными, так что диалоговое окно не появляется.

В операторе [20.5], «or» будет счастлив, если хоть одно из условий выполниться.

В сценариях, в операторах подобных [20.4, 20.5] вам скорее всего захочется использовать скобки, так как это улучшает читаемость (и возможно надежность).

ГЛАВА 11

ПЫТАЕМСЯ ИЗБЕЖАТЬ СБОЕВ

Во всех сценариях, которые мы обсуждали до сих пор предполагалось, что если AppleScript в процессе выполнения столкнется с ошибкой, то выполнение сценария завершится.

```
beep
set x to 1 / 0                                                    [1]
say "You will never hear this!" -- вы этого никогда не услышите
```

Если вы проверите синтаксис сценария [1], AppleScript не сообщит о каких-либо проблемах. Однако, если запустить сценарий, вы не услышите фразы [1.3], потому что выполнение сценария остановится на операторе [1.2], даже несмотря на правильность оператора [1.3].

Конечно, завершение сценария не обязательно именно то, чего вы ожидаете. Например, если ваш сценарий требует наличия папки с файлами для работы, и папка была удалена, вам может понадобиться дать пользователю альтернативу, чтобы выбрать другую папку.

Когда пишется сценарий, вы должны определять операторы, которые чувствительны к проблемам в процессе выполнения. Заключите эти операторы в блок try ... end try как продемонстрировано здесь [2].

```
try
    beep
    set x to 1 / 0
    say "You will never hear this!"                               [2]
end try
say "The error does not stop this sentence being spoken"
-- ошибка не остановит произнесение этой фразы
```

Теперь, если запустить сценарий, вы услышите вторую фразу [2.6], так как AppleScript продолжит выполнение сценария после оператора «end try» [2.4].

В главе 7, в которой обсуждались записи, мы столкнулись со следующим сценарием [3]

```
set temp to display dialog "What is your age in years?" default answer ""
set ageEntered to text returned of temp                           [3]
set ageInMonths to ageEntered * 12
display dialog "Your age in months is " & ageInMonths
```

Проблема в этом сценарии в том, что пользователь введет что-либо еще вместо числа, и в сценарии будет сбой. Мы можем избежать прерывание работы сценария, и обеспечить пользователя полезной обратной связью [4].

```
set temp to display dialog "What is your age in years?" default answer ""
set ageEntered to text returned of temp
try
    -- Сначала проверим, что пользователь ввел число
    set ageEntered to ageEntered as number
    set ageInMonths to ageEntered * 12
    display dialog "Your age in months is " & ageInMonths         [4]
on error
    -- Если это не число, то ввод должно быть является текстом.
    display dialog "Instead of a number, like 30, you entered text."
    -- В отличие от числа, подобного 30, вы ввели текст.
end try
```

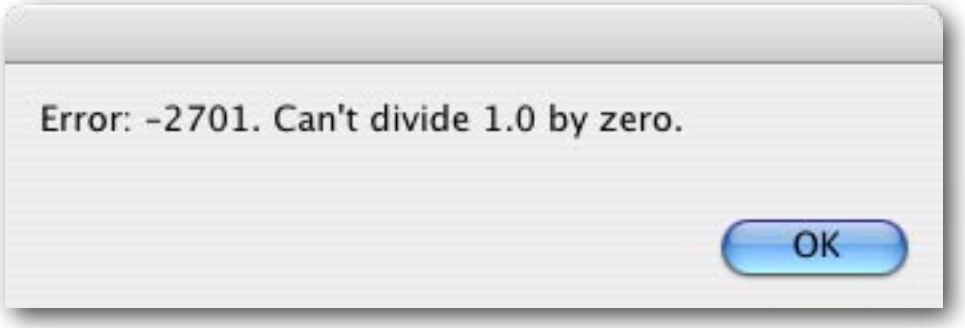
Теперь, если пользователь введет не число, он будет обеспечен обратной связью. Единственное неудобство в том, что пользователь должен запустить сценарий еще раз. В главе 13 мы решим и эту проблему.

В Script Editor очень легко заключать текст в блоки try. Просто выберите один или несколько операторов, которые вы хотите включить в блок try, и через контекстное меню, выберите нужный пункт в подменю «Error Handlers», такой например как использован в сценарии [5].

Когда сценарий переключится на часть «on error» блока try, у AppleScript будет для вас пара полезных вещей: номер ошибки, и заодно описание проблемы (зачастую оно зашифровано).

```
try
    set x to 1 / 0
on error the error_message number the error_number
    display dialog "Error: " & the error_number & ". " & the error_message ¬
          buttons {"OK"} default button 1
end try
```
[5]

Если вы поместите имя переменной после «on error», описание проблемы будет присвоено этой переменной. Если имени переменной предшествует «number», то этой переменной будет присвоен номер ошибки. В операторе [5.3] мы имеем оба варианта. Здесь диалог выглядит так.



ГЛАВА 12

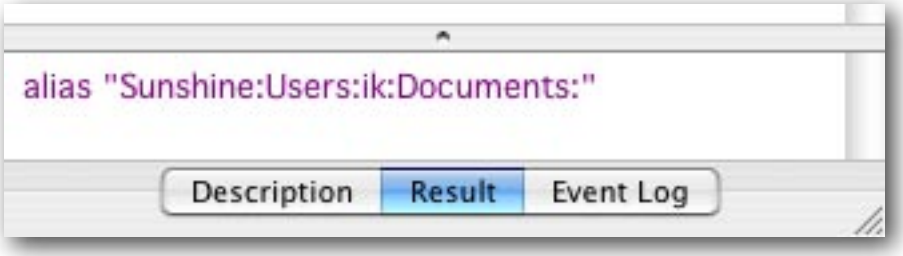
ПУТИ К ФАЙЛАМ, ПАПКАМ И ПРИЛОЖЕНИЯМ

Давайте посмотрим как организованы папки и файлы на вашем жестком диске. Если вы откроете окно Finder в виде колонок, оно будет выглядеть примерно так.

Тут жесткий диск, который содержит папки, приложения и файлы (на рисунке выше файлы и приложения не показаны). Все эти элементы организованы в виде иерархической структуры. Это позволяет нам определять размещение файла (или папки, или приложения) через путь. Давайте посмотрим как такой путь выглядит [1].

```
choose folder
```
[1]

Вот что поле результата покажет, если я выберу мою папку Documents.



Вы можете видеть путь, который в основном пишется так:

hard disk name:folder name:subfolder name:subfolder name:

(имя жесткого диска:имя папки:имя подпапки:имя подпапки:)

Проследите этот путь на картинке выше (снимок окна Finder) до папки «Documents». Затем, переведите свое внимание на картинку с полем Result. Пожалуйста, обратите внимание, что в путях двоеточия используются как разделители. Вот почему двоеточия не допустимы в именах файлов и папок. Попро-

буйте создать на рабочем столе новую папку и дать ей имя содержащее двоеточие. Наконец, вы можете увидеть, как путь оканчивается двоеточием, которое показывает, что путь относиться к папке.

Используем путь, чтобы приказать Finder'у открыть папку [2].

```
tell application "Finder"
    open folder "Macintosh HD:users:ik:Documents"                    [2]
end tell
```

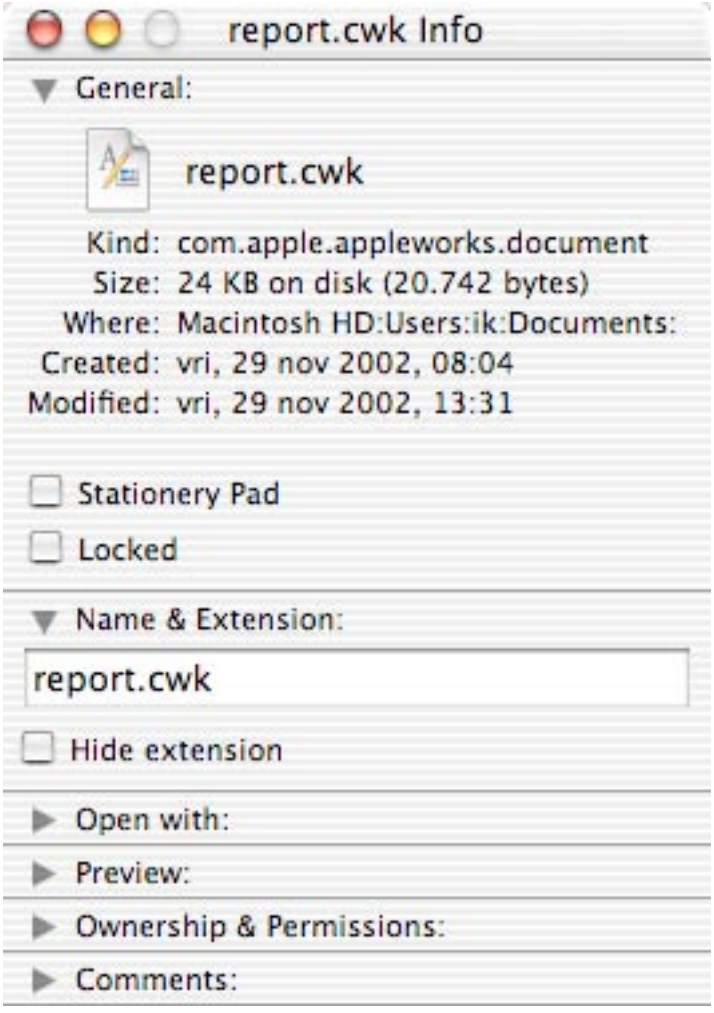
Как вы сможете увидеть, когда запустите сценарий [2] (не забудьте заменить «ik» на ваш собственное имя в системе), AppleScript будет весьма снисходителен, и не важно укажите ли вы двоеточие в конце пути или нет. Однако, пожалуйста, обратите внимание, что имя жесткого диска должно быть написано строго с учетом регистра. Ай-ай-ай, Apple!

Моя папка «Documents» содержит файл AppleWorks под именем «report». Давайте откроем этот файл [3].

```
tell application "Finder"
    open file "Macintosh HD:users:ik:Documents:report.cwk"           [3]
end tell
```

Первое замечание к этому оператору [3.2], он определяет файл («file»), не папку («folder»), потому что мы обращаемся к файлу. Второе замечание, в конце имени пути в имени файла должно быть определено расширение файла. Здесь расширение «cwk», производное от оригинального имени AppleWorks, которое раньше было ClarisWorks.

В Mac OS X расширение файла скрыто по умолчанию. Вы можете найти расширение, выбрав файл и нажав Command-i, чтобы увидеть паспорт этого файла.



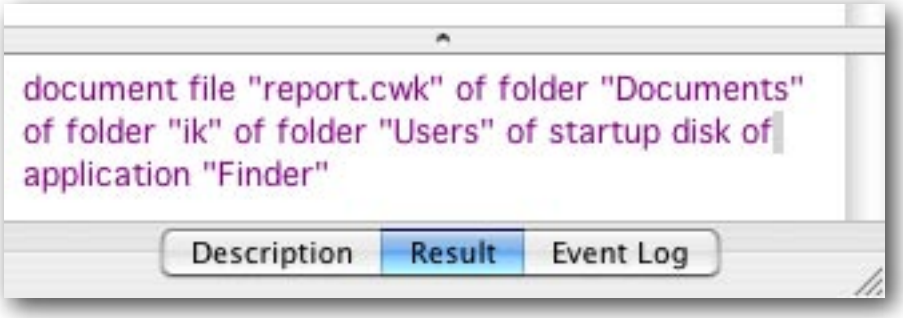
Вместо этого, вы можете предпочесть, чтобы расширения файлов были видимыми всегда. Чтобы это сделать, установите соответствующие параметры Finder. Выберите элемент меню «Finder» программы «Finder». В появившемся меню выберите «Preferences» (Параметры).

В окне, которое появится, выберите раздел «Advanced» (Дополнительно) поставьте галочку в чек-боксе «Show all file extensions» (Показывать расширения всех файлов).

Если вы хотите сохранить путь к файлу «report.cwk» в переменной, вы можете сделать это так [4]

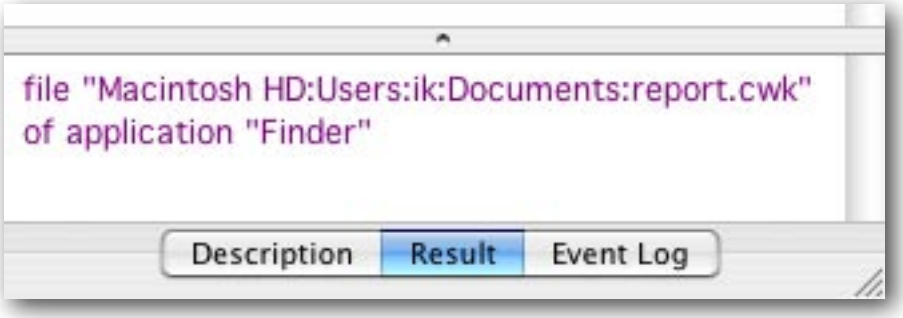
```
tell application "Finder"
    set thePath to file "Macintosh HD:Users:ik:Documents:report.cwk"
end tell
```
[4]

Однако, если запустить данный сценарий, результат не будет в таком же формате как вы видели выше. Теперь, он будет выглядеть так:



Такой стиль немного сложнее для чтения, особенно с длинными путями, и хуже всего то, что этот формат распознает только Finder. Вам может больше понравиться, или даже понадобиться, более абстрактное представление пути, использующее двоеточия. Чтобы заставить Finder вернуть его в таком виде, используйте команду «a reference to» (дать ссылку на) [5].

```
tell application "Finder"
    set thePath to a reference to file "Macintosh HD:Users:ik:Documents:report.cwk"     [5]
end tell
```



Обратите внимание, поле результата говорит, что мы имеем file (файл). Пожалуйста, запустите сценарий [6] и выберите тот же файл report.cwk.

```
choose file                                                                          [6]
```

Теперь результат будет выглядеть вот так:

Оно содержит «alias», вместо «file»! Чтобы объяснить различие в AppleScript, давайте сначала обсудим, что такое псевдоним (alias), к которому вы уже привыкли в работе с Finder.

Предположим, что я создал псевдоним на моем рабочем столе на файл «report.cwk», который находится в моей папке «Documents». Если я перемещу файл «report.cwk» из папки «Documents» в другое место, двойной щелчок по псевдониму будет всё так же открывать этот файл. Замечательно! Я могу даже переименовать исходный файл «report.cwk» во что-нибудь другое, например в «funny_story.cwk». Это потому что псевдоним не хранит информацию о размещении (и имени) файла «report.cwk», такую как:

```
"Macintosh HD:users:Documents:report.cwk"
```

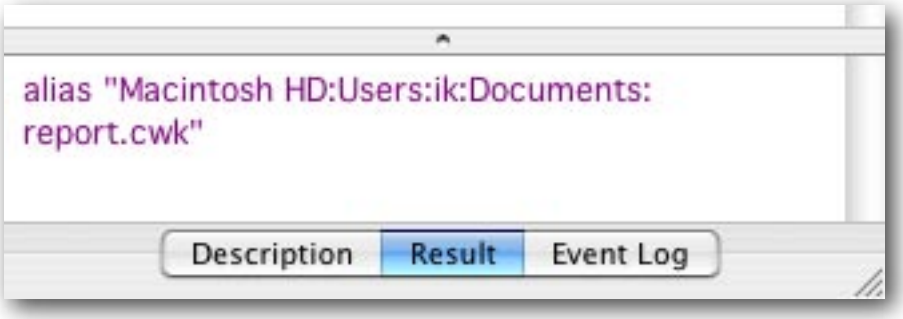
а хранит особый вид идентификатора. Finder обслуживает базу данных этих идентификаторов вместе с текущим месторасположением соответствующих файлов (а также папок, и приложений). Так что, если я перемещу файл «report.cwk», его уникальный идентификатор остается тем же, но Finder внесет изменения во внутреннюю базу данных, отражающие новое местоположение файла. Когда я дважды щелкаю на псевдониме, содержащийся в нем идентификатор используется Finder'ом для поиска файла, соответствующего этому идентификатору, и после Finder открывает нужный файл.

Чтобы создать сценарий, который не прервется в случае, если файл (или папка), на который файл ссылается, был удален или переименован, наш сценарий должен содержать идентификатор (ID) файла (или папки), вместо «жестко-заданного» пути. AppleScript позволяет это сделать [7].

```
set thePath to alias "Macintosh HD:Users:ik:Documents:report.cwk"          [7]
```

Очень важное замечание по сценарию [7], который ссылается на оригинальный файл, находящийся в папке «Documents», а не на какой-нибудь созданный пользователем псевдоним этого файла, подобный тому, что я предполагал создать на моем рабочем столе. В операторе сценария [7], ключевое слово «alias» показывающее что, после компиляции (т.е. проверка синтаксиса), сценарий запомнит идентификатор файла и, во время исполнения, не будет спрашивать Finder о файле в месте указанным с помощью пути, а будет основываться на идентификаторе.

Если вы запустите сценарий [7], и проверите результат в поле результата, то увидите следующее:



Вы не увидите сам идентификатор. Однако, слово «alias» впереди пути подскажет вам, что сценарий внутри себя работает с идентификатором, а не с «жестко-заданным» путем. Теперь, когда сценарий был скомпилирован, переместите файл «report.cwk» в другое место и запустите сценарий еще раз. Я перенес файл в папку «Miscellaneous» внутри моей папки «Documents». Даже если сам сценарий, и в особенности путь в сценарии [7], не изменяли, результат будет таким

```
alias "Macintosh HD:Users:ik:Documents:Miscellaneous:report.cwk"
```

Попытайтесь самостоятельно (еще раз повторю, ваш путь будет выглядеть по другому, потому что ваш логин скорее всего не ik и возможно потому что вы используете другие имена папок)! Если вы сохраните сценарий как скомпилированный сценарий (compiled script) или как программу (AppleScript application), идентификатор сохранится, и в следующий раз, когда вы запустите сценарий, он будет выполнен безупречно даже если вы переместили или переименовали исходный файл. Пусть для вас не будет сюрпризом то, что в сценарии произойдет сбой, если файл удалить.

Подведем итоги. У вас есть два способа, чтобы обращаться с путями в сценарии. Вы можете либо указывать местонахождение файла (жестко задавать, используя file "path here"), либо использовать ключевое слово «alias», чтобы сделать сценарий нечувствительным к перемещениям и переименованиям файла/папки/приложения после компиляции сценария.

Не смотря на то, что часто используется способ с «alias», во время компиляции сценария файл, на который вы ссылаетесь, должен быть там, где вы указали. А также, вы не сможете передать сценарий

кому-нибудь еще в виде приложения AppleScript, потому что даже если файл с таким же именем присутствует в аналогичном месте на компьютере этого человека, файл все-равно будет иметь другой идентификатор.

Примечание по сценарию [7]. Лично я был приятно удивлен тем, что всё это работает без блока обращения Finder. Все-таки внутренняя база данных хранящая идентификаторы и местоположения файлов управляется приложением Finder. Потому что только Finder может предоставить нам такую информацию, и AppleScript (компонент MacOS X) знает это, AppleScript по-видимому спрашивает Finder об идентификаторе закулисно или обращается к базе данных напрямую. Это поведение все-таки исключение. Например, открытие файла с использованием пути требует блока обращения (см. сценарий [3]). Это потому что не только Finder может открыть некоторый файл, но также программа, которая этот файл создала, и возможно другая программа тоже. Например, изображение в формате JPEG, может быть открыто программами PhotoShop, Preview и Safary. Блок обращения необходим для точного указания программы, которой будет открываться файл.

Теперь, когда мы знаем как адресовать файлы и папки, мы можем перемещать их всюду или копировать их [8].

```
tell application "Finder"
    move file "Macintosh HD:Users:ik:Documents:report.cwk" to the trash          [8]
end tell
```

ГЛАВА 13

ЦИКЛЫ

Во всех сценариях, которые мы рассматривали ранее, каждый оператор выполнялся только один раз. Иногда, требуется выполнить один или несколько операторов несколько раз. Для реализации этого AppleScript предлагает несколько разных способов.

Если вам известно, какое число раз должен повториться оператор (или группа операторов), вы можете указать его включив это число в оператор repeat сценария [1]. Число должно быть целым, потому что вы не можете повторить какую-либо операцию, скажем, 2,7 раза.

```
repeat 2 times
    say "Julia is a beautiful actress"                    [1]
end repeat
say "This sentence is spoken only once"
```

Аналогично тому, что мы видели для операторов «tell», «try» и «if...then», оператор «end repeat» обязателен, чтобы показать AppleScript'у какие операторы принадлежат к группе повторяющихся операторов.

Вместо прямого указания числа, вы можете использовать переменную [2].

```
set repetitions to 2
repeat repetitions times
    -- Здесь следуют повторяемые операторы                [2]
end repeat
```

В сценарии [3] более похожая на действительность модификация сценария [2]. Когда сценарий [3] запущен, пользователю предлагается ввести число в диалоговое окно. Поскольку все что будет введено в итоге окажется в «text returned», мы должны конвертировать введенное значение в целое число. Такую конвертацию будет невозможно выполнить, если пользователь введет текст или действительное (дробное) число. Так что, мы должны принять некоторые меры предосторожности.

```
-- Пользователю предлагается задать количество раз произнесения текстовой строки
set textToDisplay to "How often has the sentence to be repeated?"
-- Число «2» будет показано, чтобы намекнуть пользователю, какой тип данных ожидается
display dialog textToDisplay default answer "2"
set valueEntered to text returned of the result
try
    -- valueEntered это строка (не целое), которое может выглядеть так: "2".
    -- Здесь мы попытаемся преобразовать введенное пользователем значение в целое. Если это не получиться, то оператор try предотвратит прерывание сценария.
    set valueEntered to valueEntered as integer           [3]
end try
-- Если valueEntered будет соответствующего класса (т.е. целым) мы сможем выполнить блок цикла. Если нет, мы покажем диалоговое окно.
```

```
if class of valueEntered is integer then
    -- Блок цикла выполнится, если не произойдет сбоя при преобразовании
    repeat valueEntered times
          say "Julia is a beautiful actress"
    end repeat
else
    display dialog "You did not enter a (valid) number."
end if
```

После оператора else [3.21], пользователя сценария упрекнут, что он не использовал подсказку о том как сделать правильно, и пользователь должен будет заново запустить сценарий. Такая манера поведения программы не особенно то приветствуется в среде Мак-сообщества. Вот два альтернативных оператора repeat [4, 5], которые позволят вам повторять операции до тех пор пока условие не выполнится.

```
set conditionMet to false
repeat while conditionMet is false
    -- если (некоторая проверка даст в результате истину),
    -- то выполните следующий оператор                                    [4]
    --   set conditionMet в true
end repeat

set conditionMet to false
repeat until conditionMet is true
    -- если (некоторая проверка даст в результате истину),
    -- то выполните следующий оператор                                    [5]
    --   set conditionMet в true
end repeat
```

Давайте используем оператор repeat сценария [4] для того чтобы устранить неудобство сценария [3]. Мы будем повторять запрос до тех пор пока введенное значение не преобразуется в целое. В случае удачного преобразования, мы установим переменную «correctEntry» на true, в результате чего петля цикла будет покинута, и произнесение фразы сценария будет выполнено [6]. Если значение введенное пользователем невозможно будет преобразовать в целое, то мы дадим пользователю детализированную обратную связь.

```
set correctEntry to false
repeat while correctEntry is false
-- Пользователю предлагается задать количество раз произнесения текстовой
строки
    set textToDisplay to "How often has the sentence to be repeated?"
    -- Число «2» будет показано, чтобы намекнуть пользователю, какой тип
данных ожидается
    display dialog textToDisplay default answer "2"
    set valueEntered to text returned of the result
    try
          -- valueEntered всегда строка.
          --  Здесь мы попытаемся преобразовать введенное пользователем
значение в целое. Если этого не случится,
мы перепрыгнем в секцию on error
          set valueEntered to valueEntered as integer
          -- Установка correctEntry на true, закончит цикл
          set correctEntry to true
```

```
    on error
        -- Создадим детализированную обратную связь                    [6]
        try
            -- Сначала мы проверим, вдруг пользователь ввел дробь
            set valueEntered to valueEntered as number
            display dialog "You entered a fractional number instead of an integer."
        on error
            -- Если это не число, введенное должно быть текст
            display dialog "Instead of an integer, like 9, you entered text."
        end try
    -- Так как значение correctEntry осталось ложным, цикл продолжается
    end try
end repeat

-- Сценарий может выполнить следующие операторы, только если correctEntry
будет истинным. correctEntry будет истинным только если valueEntered удачно
преобразуется в целое
repeat valueEntered times
    say "Julia is a beautiful actress"
end repeat
```

Пожалуйста, обратите внимание на то, где расположен оператор «set correctEntry to true», это очень важно. Его положение должно быть:

– внутри (первого) блока try; и
– после оператора, который может привести к ошибке (попытка преобразования – потенциально опасный оператор).

Иначе, correctEntry станет истинным независимо от того требуемое ли действие (успешное преобразование в целое) выполнится.

Несмотря на то, что сценарий [3] может выполнить заданное действие (произнести фразу указанное количество раз) в точности как сценарий [6], он не является ни дружественным для пользователя, ни надежным. Т.е. в нем произойдет сбой, если пользователь введет неверные данные, и он не сможет обеспечить адекватную реакцию, если пользователь сделает ошибку. Дополнительно, сценарий [6] можно улучшить путем введения ограничения на значение valueEntered (например, используя фрагмент [7]), чтобы предотвратить произнесение фразы 10,000 раз. С другой стороны, может и сценарий [6] вполне удовлетворит ваши потребности.

```
if valueEntered > 5 then
    set valueEntered to 5                                               [7]
end if
```

Если вам нужен действительно надежный сценарий, вы должны его хорошенько протестировать. Попробуйте ввести текст, дробные числа, очень большие числа и т.п., чтобы проверить как сценарий себя поведет. Единственное, что не предусмотрено сценарием [6], это тот случай, когда пользователь введет отрицательное число. Вы можете тоже перевести его в положительное число, или просто сообщить пользователю, что ожидается положительное число. Интересно, что в сценарии [6] не произойдет сбоя, если ввести отрицательное число. Проверьте это сами изменив сценарий [1].

Операторы repeat сценариев [4] и [5] легко могут быть использованы для любой цели. Вы можете выполнять цикл, чтобы убедиться в том, что:

– пользователь выбрал файл или папку,
– слово присутствует в файле,
– в программе выделен нужный объект,
– и т.п.

В противоположность основному применению операторов repeat в сценариях [4] и [5], сценарии [1] и [2] предназначены для числовых условий. Для этого существует еще несколько операторов repeat.

```
repeat with counter from 1 to 5
    say "I drank " & counter & " bottles of coke."                    [8]
end repeat
```

Как вы видите, вы можете использовать переменную оператора [8.1], т.е. «counter», в вашем сценарии. Однако, вы не сможете изменить значение переменной внутри блока repeat.

```
repeat with counter from 1 to 5
    say "I drank " & counter & " bottles of coke."                    [9]
    set counter to counter + 1
end repeat
```

Если вы запустите сценарий, ни одной произносимой фразы не будет пропущено, и все бутылки – с 1 по 5 – будут выпиты.

В операторе [9.1], шаг размером равным 1 используется по умолчанию. Если вы хотите другой размер шага, вы можете его задать – [10.1].

```
repeat with counter from 1 to 5 by 2
    say "I drank " & counter & " bottles of coke."                    [10]
end repeat
```

В сценарии [10] используется шаг с размером равным 2. Вы услышите предложение три раза (при значениях счетчика равных 1, 3 и 5).

Если у вас есть список, и каждый элемент должен быть использован в какой либо операции, вы можете подсчитать число элементов в списке, и выполнить цикл repeat как в сценарии [11].

```
tell application "Finder"
    set refToParentFolder to alias "Macintosh HD:Users:ik:Documents:"
    set listOfFolders to every folder of refToParentFolder            [11]
    set noOfFolders to the count of listOfFolders
    repeat with counter from 1 to noOfFolders
          -- Здесь действия
    end repeat
end tell
```

Однако, в AppleScript есть элегантная альтернатива, демонстрация которой дается ниже. Сценарий [12] позволяет вам определить количество вложенных папок в выбранной пользователем папке. Затем оператор repeat создает список имен всех имеющихся папок.

```
set folderSelected to choose folder "Select a folder"
-- Чтобы отыскать все имеющиеся папки в выбранной папке, мы просим Finder
дать нам ответ.
-- Внимание: «every folder» НЕ включает папки, находящиеся внутри других
папок. Это просто папки, которые вы увидите как если бы открыли папку в
Finder.
tell application "Finder"
    set listOfFolders to every folder of folderSelected
end tell
```

```
-- Результат будет списком ссылок на папки (путей), которые могут быть обработаны вне блока обращения Finder.
-- Закоментируйте все последующие операторы и используйте поле результата, чтобы увидеть, что список «listOfFolders» содержит нечто вот такое: folder "reports" of folder "Documents" of folder "ik" of folder "Users" of startup disk of application "Finder".
-- Только Finder и AppleScript (компонент Mac OS X) умеют обращаться с такими ссылками.

-- Имена папок будут сохранены в новом списке, который создается здесь
set theList to {}
repeat with aFolder in listOfFolders
    -- Мы имеем ссылки на папки, и так как ссылка содержат имя папки, AppleScript (компонент Mac OS X) может добыть имя не обращаясь к Finder.
    -- Если же вы хотите получить другое свойство папки, например размер (size, в байтах) папки, придется столкнуться с Finder (и также понадобиться блок обращения для следующего оператора).
    set temp to the name of aFolder
    -- Здесь мы добавляем имя в список.
    set theList to theList & temp
end repeat
```

ГЛАВА 14

ОБРАБОТЧИКИ ВСЕГО

AppleScript по природе англоподобный язык, и это помогает сделать сценарий легким в чтении и написании. Мы видели также, что часть ответственности за легкость восприятия сценариев лежит и на вас. Например, это видно, когда вы даете переменной описательное имя, и обеспечиваете ваш сценарий полезными комментариями. AppleScript поможет сохранить читаемость ваших сценариев при помощи «обработчиков» (handlers, также распространено название «подпрограммы»). Предположим, что у вас есть некоторый набор операторов, который встречается несколько раз в вашем сценарии. Если в нем вдруг окажется ошибка, вам придется исправлять каждый из них. AppleScript предлагает возможность группировать такие операторы и давать этой группе имя. Если вы вызовите этот набор по имени, то он будет выполнен .

Здесь показано, как определить обработчик [1].

```
on warning()
    display dialog "Don't do that!" buttons {"OK"} default button "OK"          [1]
end warning
```

Чтобы его использовать, ваш сценарий должен вызвать обработчик, вот так [2].

```
warning()                                                                       [2]
```

Не имеет значения где определен обработчик в сценарии до или после вызова обработчика.

Обработчик в сценарии [1] совсем негибкий. Было бы хорошо, если бы могли сказать обработчику какой текст показывать. Угадайте, для чего был разработан этот обработчик [3].

```
on warning(textToDisplay)
    display dialog textToDisplay buttons {"OK"} default button "OK"             [3]
end warning
warning("Don't do that!")
warning("Go fishing!")
```

В операторе [3.1], переменная «textToDisplay» принимает значение передаваемое с вызовом обработчика (в операторах [3.4] и [3.5], каждое из которых содержит значение заключенное в скобки, и это значение передается обработчику). Когда сценарий [3] будет выполнятся, будут показаны последовательно два диалоговых окна.

Вместо указания данных, когда вызывается обработчик, вы можете использовать переменную [4].

```
on warning(textToDisplay)
    display dialog textToDisplay buttons {"OK"} default button "OK"             [4]
end warning
set someText to some item of {"Don't do that!", "Go fishing!"}
warning(someText)
```

Обратите внимание, что имя переменной используемой, когда вызывается обработчик [4.5] отличается от имени в определении обработчика [4.1]. Теперь, вам не нужно знать (искать) имя переменной используемое обработчиком. Конечно, вам надо знать какой тип данных ожидает обработчик.

Вы можете не только передавать информацию обработчику, но и он также может возвращать информацию.

```
on circleArea(radius)
    set area to pi * (radius ^ 2)
end circleArea
set areaCalculated to circleArea(3)
```
[5]

Обработчик «circleArea()» выполняет вычисление pi*3^2 и автоматически возвращает результат. Автоматическое возвращение результата следует за последним выполняемым оператором обработчика, как будто бы используется команда «get». Чтобы гарантировать, что будет возвращен нужный вам результат, даже если он получен где-то среди серии операторов [6.3], используйте ключевое слово «return» [6].

```
on older(a)
    if a > 30 then
          return "older"
    end if
    return "not older"
end older
set theAge to older(73)
```
[6]

Если оператор сравнения [6.2] истинный, результат возвращает оператор [6.3], хоть он и не самый последний в обработчике.

Вы не ограничены передачей одной единственной переменной [7.7] обработчику.

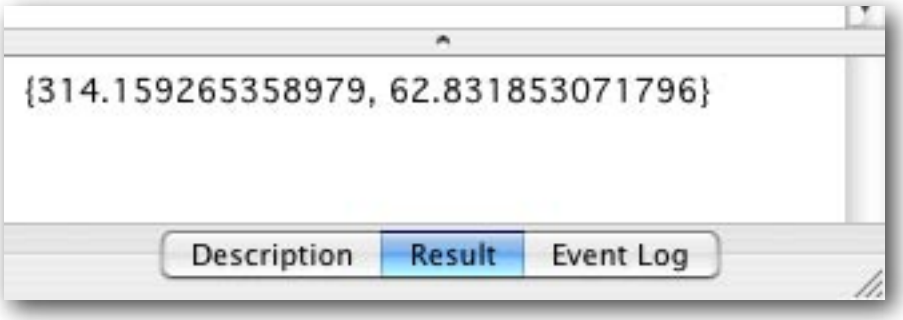
```
on largest(a, b)
    if a > b then
          return a
    end if
    return b
end largest
set theLargest to largest(5, 3)
```
[7]

Этот сценарий возвращает большее из двух значений. Обратите внимание, что в операторе [7.1] две переменные обеспечивают доступ к значениям. Соответственно, вызывая обработчик, необходимо передать две переменные [7.7]. В реальных сценариях, по крайней мере одно из этих значений будет передано в виде переменной.

Также можно возвращать более чем одно значение. Для этого нужно возвратить данные как список [8.4].

```
on circleCalculations(radius)
    set area to pi * (radius ^ 2)
    set circumference to 2 * pi * radius
    return {area, circumference}
    set testVar to 3
end circleCalculations
set circleProperties to circleCalculations(10)
```
[8]

Сценарий выше возвращает список содержащий оба значения: площадь и длину окружности, по заданному радиусу.



Написав сценарий как серию обработчиков и нескольких операторов, которые будут составлять тело вашего сценария, можно сохранить много времени и нервов благодаря одному важному свойству обработчиков. Здесь я задам вам вопрос (на который я сам для себя уже ответил): что если сценарий использует имя переменной, которое уже присутствует для других целей в обработчике? Не волнуйтесь, они не перепутаются. Для доказательства этого, выполним следующий сценарий [9].

```
set x to 5
on example()
    set a to 5 * x
end example
set y to example()
```
[9]

Вы получите следующую ошибку:

Внутри обработчика, переменная x не определена. Если вы хотите, чтобы значение x [9.1] было известно внутри обработчика, вы должны передать её, как продемонстрировано в сценарии [4] .

И наоборот, изменение переменной внутри обработчика не имеет эффекта вне обработчика. Вот доказательство [10].

```
set x to 5
on example()
    set x to 6
end example
example()
get x
```
[10]

В поле результата видно, что x равно 5. Так что, взаимодействия между сценарием и обработчиком нет, за исключением значений определенным образом передаваемых обработчику и возвращаемых обработчиком. Я должен сказать, что есть еще способ, как сделать переменную «x» оператора [9.1] или [10.1] функциональной в обработчике без передачи ее. Однако, это делает сценарий более трудным для чтения и отладки. Но это также дает большое преимущество обработчикам, о котором рассказано в следующем параграфе.

Если вам интересно узнать как создать переменную, которая действительна внутри и снаружи обработчика, здесь показано как это сделать: поставьте перед именем переменной ключевое слово «global» (глобально). Рекомендуется сделать это в первой строке(ах) вашего сценария, так вам легче будет найти переменные, которые ведут себя как глобальные.

```
global x

set x to 5
on example()
    set x to 6
end example
example()
get x
```

Теперь, в поле результатов показано, что x равна 6. Из вышесказанного следует, что переменные по умолчанию являются локальными .

Пока я не могу сказать, что мне нравится определять переменные глобально, по причинам объясняемым выше, но есть один тип глобальных переменных, который очень удобен. Они имеют другое название «property» (свойство).

```
property x : 1

on example()
    set x to x + 1
end example
example()
get x
```

Выполните указанный сценарий несколько раз и внимательно присмотритесь к результату. Значение свойства запоминается между несколькими запусками сценария. Вы заметили, что свойство действительно как внутри так и снаружи обработчиков, другими словами, оно ведет себя как глобальная переменная.

Отдельно от описанных выше важных преимуществ (читаемость и т.д.), есть еще одно дополнительное большое преимущество – это то, что вы можете повторно использовать обработчики в других сценариях, которые вы создаете. Поскольку обработчик был использован и протестирован для одного сценария, вы можете быть точно уверены, что он будет работать и в других сценариях. Это сократит время при разработке новых сценариев. Не думайте, что вы сможете скопировать операторы, которые выполнены не в форме обработчика, из одного сценария в другой так же легко. Вы должны будете пройти весь сценарий, чтобы вычислить что надо скопировать, на это уходит много времени. В дополнение, вы рискуете скопировать операторы неполностью и/или операторы, которые будут неверно работать в новом сценарии. Поверьте мне, лучше использовать обработчики.

Замечательно, теперь вы можете ощутить всю эффективность повторного использования обработчиков в других сценариях. Но что если вдруг вы найдете ошибку в обработчике или подумаете о последующем расширении его возможностей? Вы должны будете изменить все сценарии. В AppleScript есть решение этой проблемы – это команда «load script». Все что вы должны сделать:

1) сохранить один или более обработчиков как скомпилированный сценарии (compiled script);
2) включить в сценарий оператор нужного обработчика.

```
set aVariableName to (load script "path here")
```

Чтобы использовать обработчик компилированного сценария, вы должны использовать блок обращения.

```
tell aVariableName
    handlerName()
end tell
```

У обработчиков есть одна специфическая особенность, если вы используете блок обращения, без дополнительного указания обработчик окажется неопределенным внутри блока обращения. Если запустить сценарий [11],

```
on showProgress()
    display dialog "Job done!"
end showProgress
```

[11]

```
tell application "Finder"
    empty the trash
    showProgress()
end tell
```

то будет показана такая ошибка:

Этого легко избежать. Просто укажите, что обработчик принадлежит самому сценарию [12.7].

```
on showProgress()
    display dialog "Job done!"
end showProgress
```

[12]

```
tell application "Finder"
    empty the trash
    showProgress() of me
end tell
```

(AppleScript пытается отправить команду «showProgress()» непосредственно программе «Finder», который о ней ничего не знает, а знает о ней только сценарий. Существует также синоним my – «my showProgress()». – прим. ред.)

Это обязательно только для обработчиков, не для переменных [13], как видно на примере ниже.

```
set x to 4
tell application "Finder"
    set x to 5
end tell
get x
```

[13]

Результат будет равен 5.

ГЛАВА 15

ИСТОЧНИКИ ДОПОЛНИТЕЛЬНОЙ ИНФОРМАЦИИ

Основное назначение этой книги научить вас основам AppleScript. Если вы прочтете эту книгу дважды, и попробуете примеры со своими собственными вариациями, вы будете готовы научится писать сценарии для приложений, которые важны вам самим.

Важное дополнение для тех, кто хочет начать писать свои собственные сценарии: проверьте, может быть кто-нибудь уже написал, то что вам нужно! Так что, вы сохраните свое время просто поискав такой же сценарий, и найдя его, можете изменить под собственные нужды. Где вам искать эти сценарии? В Интернете, конечно. Сначала, посетите сайт Apple

http://www.apple.com/applescript

на котором много полезных ссылок.

Я очень рекомендую сделать также закладку на

http://www.AppleScriptSourcebook.com/

и конечно

http://macscripter.net/

На сайте Macscripter есть форум, куда вы можете отправить свои вопросы. Опытные члены сайта так же как и штат макскриптера сделают все, чтобы помочь вам. Да, мы же говорим здесь о Мак-сообществе. На сайте также имеется большая серия ссылок и на другие сайты и источники информации.

Я надеюсь, вам понравилась эта книга и язык сценариев AppleScript. Ох, и, пожалуйста, не забудьте главу 0.

Берт Алтенбург

ГЛАВА 16

РУССКОЯЗЫЧНЫЕ РЕСУРСЫ

На русском языке не так много статей и сайтов, но они есть. Эта книга тоже часть развития русскоязычных ресурсов по AppleScript. Итак, небольшой обзор сайтов.

Пока основной сайт по языку AppleScript в России. Есть форум, немного сценариев, есть ссылки на все возможные ресурсы в сети на русском языке посвященные AppleScript.

http://www.yezhe.ru/applescript/

Домашняя страничка Эдуарда Мисюка. Вот что он сам пишет о своем сайте: «здесь Вы сможете найти что-то обо мне, о том, чем я занимаюсь или занимался. Своему возникновению эти страницы обязаны моей работе в ДТП, долгой и стойкой любви к фирме Apple и ее продуктам, а также моему последнему увлечению – AppleScript.»

http://homepage.mac.com/e_s_m/index_ru.html

Макинтош и образование. На этом сайте вы найдете раздел посвященный AppleScript и краткий справочник. Автор – Михаил Крекин.

http://macedu.narod.ru/index.html

ГЛАВА 17

ПРИМЕЧАНИЕ К РУССКОМУ ПЕРЕВОДУ

До сих пор я не встречал ничего на русском языке по AppleScript. Были краткие введения в различных книгах, например: Девид Пог, «Mac OS X – Основное руководство», или книга Волкова «Mac OS X, UNIX – для всех». Но это были краткие упоминания по языку программирования… извините, языку сценариев AppleScript. И абсолютно нечего стесняться, что вы теперь прочитав эту книгу можете написать простейший скрипт. Это тоже программирование, но программирование – которое может сильно облегчить жизнь в рутинной и однообразной работе.

Прошу заранее извинить меня, за не очень хороший, как мне кажется перевод, но мною двигало не столько знание английского (со словарем), сколько огромное желание перевести наконец то, что уже давно должно быть на наших столах.

Хочется поблагодарить автора книги Берта Альтенбурга, который не только написал эту книгу, но и очень быстро откликнулся на мою просьбу и дал разрешение перевести эту книгу на русский язык! Она самая первая!

Special Thanks, Bert Altenburg!

Еще хотелось бы напомнить, что и в нашей стране есть информационные источники по языку AppleScript, которых к сожалению не так много, но слава Богу они есть. Рекомендую для начала сходить на сайты указанные в главе 16

Извините, больше не знаю, куда и обратиться за дополнительной информацией. Но у этой книги есть преимущество – она электронная, а следовательно можно ее дополнить информацией, а еще лучше… даже страшно подумать – написать продолжение… Ну, что есть богатыри?

Владимир Прохоренков

ПРИМЕЧАНИЕ КО ВТОРОМУ ИЗДАНИЮ

По вышеописанным причинам, потому что желание изучать язык есть, а писать и переводить мы не умеем, вышло это второе издание. Пусть даже у нас AppleScript не столь популярен, но все же хочется, что бы было с чего начать русскоговорящему пользователю. И я надеюсь на то, что второе издание станет куда более понятным и качественным. Я постарался не только более точно перевести, но и согласовать книгу с современными программами, а также попытаться написать ее на «русском» языке.

Кирилл Корчагин (Aha-pupok)

Для заметок

Для заметок