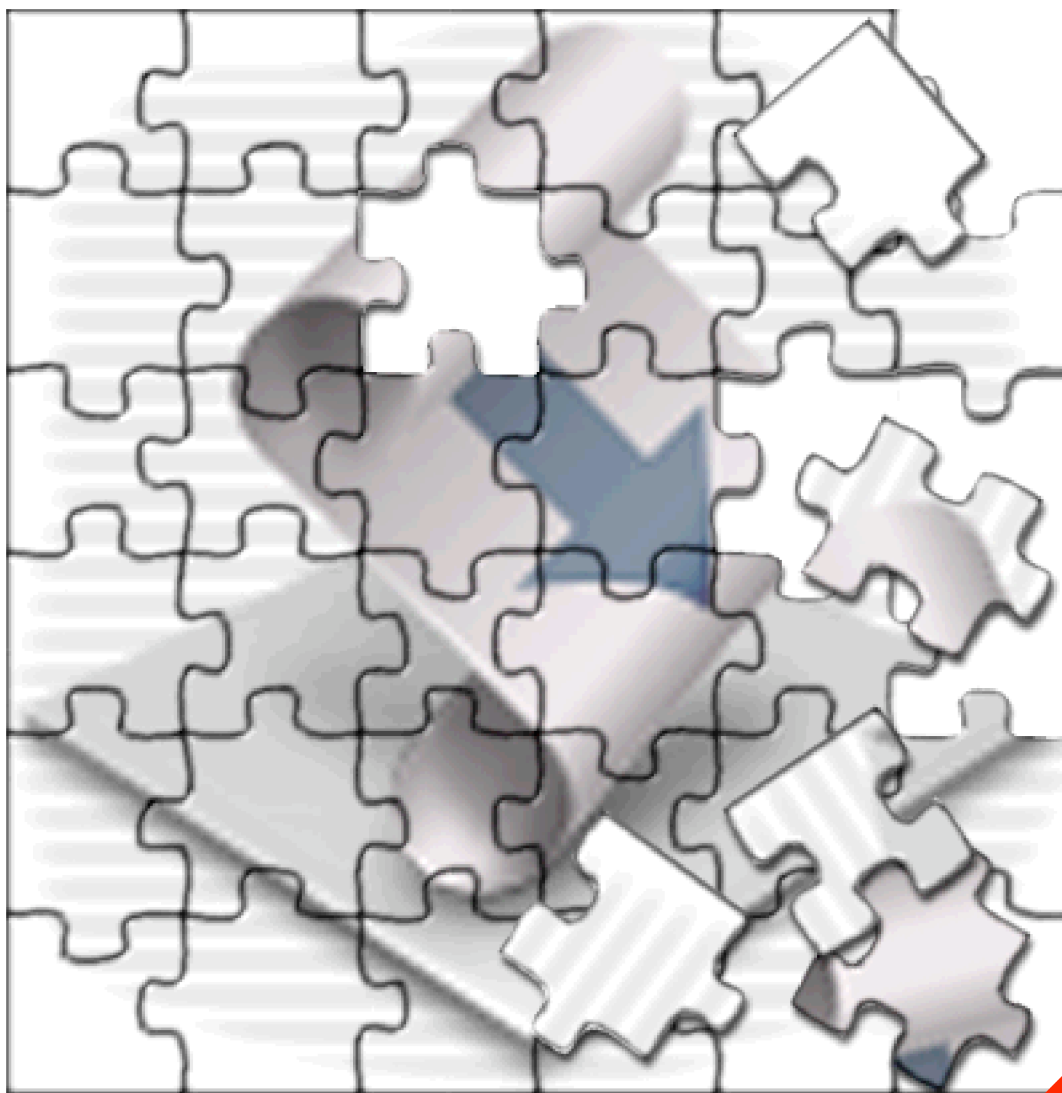


Берт Альтенбург

APPLESCRIPT ДЛЯ АБСОЛЮТНЫХ НОВИЧКОВ



Дизайн обложки: Питер Фишер
Перевод: Владимир Прохоренков

ПЕРВАЯ КНИГА
НА РУССКОМ ЯЗЫКЕ

ВСТУПЛЕНИЕ

AppleScript это революционная технология компании Apple делающая возможным общение между программами. Для примера, с AppleScript Вы можете:

- отправить электронное письмо через Mail и сохранить в базе данных;
- заставить программу редактирования иллюстраций сделать изменения в разрешении целой серии иллюстраций, изменить размер и послать результат на другой компьютер или через интернет;
- и многое, многое другое.

AppleScript, или просто скрипт, это серия инструкций записываемая языком скриптов. Этот язык очень близок для понимания на английском языке, делая AppleScript максимально легким как в изучении, так и просто в чтении и записи.

Несмотря на свою мощь, AppleScript тяжел для использования только в нескольких местах. Эффективность индустрии публикаций зависит от автоматизации процессов (PhotoShop, QuarkExpress, InDesign). Разработчики Filemaker Pro используют его для создания баз данных. Отдельно стоит упомянуть небольшие программы, такие как GraphicConverter, BBEdit, и даже огромный Word с поддержкой AppleScript. Вы можете использовать AppleScript для того чтобы похозяйничать в этих программах. Однако применение скриптов для этих программ не будет рассмотрено в данной книге. Для этого уже сейчас на издательском рынке есть соответствующие книги. Эти книги, как правило, очень мало рассматривают сам язык и очень быстро начнут показывать Вам как надо использовать AppleScript, но для этого требуются базовые знания по языку AppleScript. Цель этой книги, помочь Вам именно в этом.

Эта книга дает базовые знания и расширяет их. Наверняка Вы захотите это проверить на свежих версиях (см. Главу 15). Вторая книга по автоматизации программ с помощью AppleScript сейчас в стадии написания. Эта книга бесплатна, и Вы можете распространять ее среди пользователей Макинтош только бесплатно.

Как только Вы окунетесь в мир AppleScript, Вы будете использовать термин «AppleScript» в трех разных направлениях.

- Язык AppleScript: похожий на английский язык с возможностью писать инструкции для Вашего Макинтош;
- AppleScript: серия готовых инструкций, т.е. скриптов, написанных на языке AppleScript;
- И как часть операционной системы (Mac OS X), который читает и выполняет инструкции AppleScript.

В этой книге, если есть желание обратиться к одному из трех понятий:

- Язык AppleScript;
- Скрипт AppleScript;

- И AppleScript как компонент Mac OS X.

Изучение скриптов на базе AppleScript идеальный путь в мир программирования. Это не учитывая тех сложных приготовлений к программированию задачи, например как в языке Java, прежде чем программа начнет по-настоящему работать. При всем том, что язык AppleScript достаточно прост в изучении, что вполне возможно его изучение даже 10-ти летним ребенком, но при всей простоте язык мощный, что даже профессионалы в состоянии этим насладиться. Это оставляет и Вам огромную площадь для фантазии и роста. Вы можете писать даже свои собственные программы, что к сожалению в этой книге не рассматривается. Для создания настоящих программ Вам понадобится AppleScript Studio (прим.пер.: или Xcode если Вы используете Mac OS X Panther).

В чем различие между скриптами и программированием? Я хотел бы думать, что в простоте, если писать скрипт легко, нежели в программировании. Однако JavaScript не легок судя по моей книге, настолько, что сравнение не возможно по этой причине.

Как использовать эту книгу?

По мере того как Вы увидите, что некоторые абзацы выделены зеленым цветом, мы предлагаем читать Вам каждую главу как минимум дважды. Первое время пропускайте зеленый текст, а потом читая главу, читайте и зеленые абзацы, не пропуская их. Для большего эффекта изучения языка, повторите – что Вы выучили как стихотворение в школе. Используйте книгу в дороге, Вы сократите путь и освоите материал быстрее.

Эта книга содержит множество примеров. Для того, что разобраться и понять используются примеры, которые в тексте указаны в квадратных скобках, как это: [4]. Большинство сценариев состоят из двух и более строк текста. Время от времени будет использовать вторая цифра в квадратных скобках, которая указывает на строку кода, например: [4.3], где вторая цифра это третья строка в примере [4].

Вы не сможете научить лошадь скакать, если будете только читать эту книгу. Вы не сможете выучить AppleScript, если будете использовать эту книгу как чтение. Вы просто обязаны переключаться на программу Script Editor (см. Главу 2) и пробовать полученные знания.

Copyright (c) 2003 by Bert Altenburg

Атрибутика: Bert Altenburg, владелец лицензии на эту книгу, позволяет ее копировать, дорабатывать в любой последовательности.

Не для продажи: Владелец лицензии разрешает копирование, модификацию и дистрибуцию данной книги, без ее продажи. Изменение лицензии не возможно без участия правообладателя.

ГЛАВА 0

ПРЕЖДЕ ЧЕМ НАЧАТЬ

Я написал эту книгу для Вас. Так как она свободна в своем распространении, пожалуйста позвольте мне сказать пару слов чтобы прорекламирровать Макинтош. Каждый пользователь Макинтош может помочь улучшить их любимую платформу с меньшим усилием. Здесь так и будет

1. Очень эффективная AppleScript, которую легко получить приобретая Макинтош с дисками программного обеспечения. Есть, кстати, курсы по изучению языка AppleScript после которых Вы сможете сэкономить огромное количество денег и время для своей компании.
2. Весь этот мир недоступен пользователям PC, сколько бы они не смотрели на Макинтош. Вы можете носить майку с символикой Apple вне своего дома, чтобы показать свой мир. Если вы используете CPU Monitor (в папке Utilities Вашего компьютера), вы увидите какая мощь спрятана в Вашем Макинтош. Научные сотрудники Folding@home используют всю эту мощь в своей работе. Вы можете помочь в этом счете, загрузив бесплатно маленькую программу DC-клиента, которая будет считать небольшой код вычислений, а потом все это будет объединено в целую картину. Поможет ли все это Макинтош? Только наилучшим образом! Это безусловно повысит рейтинг платформы Макинтош. Так потребители других платформ увидят, что делают Макинтоши. Потом появятся клиенты DC для того, чтобы найти лекарство от болезней. И прочего. Для того, чтобы выбрать проект, который Вам понравится, сходите и выберите DC-клиента через сайт в интернете: www.aspenleaf.com/distributed/distrib-projects.html
Одной проблемой станет меньше.
3. Макинтош имеет одно из лучших средств для разработки программ. Вы просто обязаны научиться программировать лучшие программы. Справедливо будет помощь таким программистам, наладьте с ними обратную связь, чтобы сделать программы еще лучше. Посетите этот адрес в интернет, где красиво объяснят что и как надо делать, чтобы сообщить об ошибках в программах: <http://www.macinstruct.com/tutorials/crash/index.html>
4. Платите за программу, которую Вы используете. Это сможет доказать разработчикам, что платформа Макинтош перспективна.
5. Пожалуйста сообщите своим друзьям об этой книге, или прорекламируйте ее в четырех разных местах.

Ну все, если Вы уже загрузили себе клиента DC в фоновый режим, тогда начнем!

ГЛАВА 1

СКРИПТ ЭТО СЕРИЯ ИНСТРУКЦИЙ

AppleScript это часть операционной системы Макинтош довольно производительна с небольшим количеством прописных инструкций. Для примера, воспроизведем системный звук. Посмотрите сейчас на скрипт [1] воспроизводящий системный звук Макинтош.

[1]
`beep`

Здесь имеется только одно короткое слово, состоящее из одиночной команды или инструкции. Одна линия содержит инструкцию или задание на выполнение, избавляя от длинного описания всей процедуры. Теперь если выполнить этот скрипт, Ваш Макинтош издаст короткий системный звук. Если нужно выполнить серию из нескольких звуков, надо после инструкции указать количество таких звуков, цифра после команды укажет на количество таких звуков [2].

[2]
`beep 2`

Как Вы видите разница между скриптами [1] и [2], состоит только в дополнительной опции. Если Вы не установите цифру после системного сигнала, AppleScript воспроизведет один единственный сигнал. Также, цифра 1 это значение по умолчанию, которое ставить не обязательно.

Если Вы слышали системный звук на PC, то мы можем заставить AppleScript пообщаться с нами в духе Макинтош [3], используя следующее выражение:

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

Вы можете выбрать другой голос для исполнения этой фразы, например "Fred", "Trinoids", "Cellos", или "Zarvox" [4], чтобы заменить голос который стоит по умолчанию - "Victoria".

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

#Примечание: Изначально, AppleScript не чувствителен к регистру записываемого текста. Нет разницы между большими и прописными буквами. Однако, голоса которые мы можем использовать, например "Victoria", или "Zarvox" следует писать так как они указаны. Grrr.#

Как Вы видите, инструкции AppleScript похожи на английский язык, делая сценарий легким и понятным, тем более если Вы никогда не имели опыта с другими языками сценариев. Но пока сценарии [1-4] будут как потеха; они не очень полезны. Язык AppleScript имеет множество команд, которые возможно Вас впечатлят. AppleScript тем более удобный язык, потому что в состоянии связать Вас с другими программами. Но это будет работать если такие программы поддерживают язык сценариев. В результате чего Вы имеете расширенный набор команд от AppleScript компонента операционной системы Mac OS X, тем самым имея возможность расширить использование Ваших программ.

Некоторые программы на Макинтош более популярны, чем другие. Но одна наиболее часто используется пользователем Макинтош: Finder. Да, Finder это программа. Когда Вы включаете Макинтош, она запускается автоматически и все время работает. Finder позволяет Вам перемещать файлы, искать файлы на жестком диске, создавать папки, копировать или переименовывать и многое другое. Для примера, Если Вы очищаете Корзину, то Вы используете как раз Finder для этой задачи. Но раз Вы можете выполнить очистку Корзины мышкой или клавиатурой, то почему бы не сделать тоже самое с помощью AppleScript [5] вот так к примеру.

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

Как начальник, Вы должны сказать:

- кто должен выполнить задание и
- как она должна работать.

Бесполезно говорить, для примера, что PhotoShop что-то должен стереть положив в Корзину. PhotoShop не знает как это сделать. Так что, инструкция для выполнения этого действия должна относиться скорее к Finder.

Как в реальном мире, работу, которую приказал сделать начальник, Вы можете сделать так или иначе в отличии от Вашего Макинтош, который сделает то, что от него просят. Если Вы положили в корзину важные документы, то после исполнения сценария AppleScript [5], Вы потеряете их навсегда.

Первой позицией сценария [5.1] будет оператор **tell** который адресует выполнение дальнейшего сценария AppleScript к конкретной программе Mac OS X, где в данном случае будет программа Finder. AppleScript будет выполнять задание в данной программе Mac OS X до тех пор, пока не встретит оператор **end tell** [5.3]. В выше приведенном сценарии [5] мы указываем AppleScript послать сообщение Finder, чтобы он выполнил инструкцию по очистке Корзины. Вместе, такие линии:

```
tell application "xyz"

end tell
```

принято называть **tell block**. Инструкции выполняемой программы "xyz" будут находиться в **tell block** программы "xyz". Если следовать духу Макинтош, то язык AppleScript еще не полностью отвечает всем требованиям, даже в сравнении с другими языками сценариев или программирования и не следует избегать двух правил. Первое из которых гласит, что Вы обязаны заключать в двойные кавычки имя программы, это видно из строки [5.1].

И второе, это дать Finder достаточное количество инструкций. В примере [6] ниже, сценарий из двух элементов [6.2, 6.3] с заданием к Finder. Потому, что они оба должны быть выполнены Finder, так как заключены в **tell block** для программы Finder.

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

После очистки Корзины, Finder откроет окно и покажет содержимое жесткого диска.

Как видите, мы делаем с Finder все, что хотим. Мы можем сказать Finder чтобы он изменил размер окна, поменял его позицию на экране и конечно многое другое. Вы научитесь делать это, но позднее.

Мы сейчас создадим сценарий который содержит больше инструкций для Finder, и для AppleScript [7].

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

Первое, что выполнит Finder это строки инструкции [7.2, 7.3]. И только потом **beep** инструкцию [7.5] после выполнения AppleScript. Это все для эффекта, чтобы показать, что сценарий выполнен. Интересно, если положить инструкцию **beep** внутри всего задания (и получится, что мы выполняем инструкцию программы Finder, как компонента Mac OS X) внутри или снаружи **tell block** [8].

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

Пока Finder не знает команды `beep`, зато AppleScript компонент Mac OS X знает как это выполнить. Это дает возможность легко прочитать и понять сценарий. А также, Вы имеете первый `tell block` содержащий первый Finder - выполнимый элемент [8.2], далее элемент с командой `beep`, и наконец вторая часть `tell block` для последнего Finder - выполнимого элемента [8.4].

Запомните, пока команды выполнялись около AppleScript компонентов Mac OS X находясь в любом месте сценария, каждая инструкция для конкретной программы, например Finder, находясь внутри `tell block` конкретной программы. Следующий сценарий [9] содержит критическую ошибку(последний элемент [9.5]).

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

AppleScript компонент Mac OS X не знает какой открыть диск, и программа не сможет выполнить эту процедуру. Первая часть сценария (элементы [9.2-3] внутри `tell block`) будут выполнены, в то время как последний элемент [9.5] не выполним.

В выполняемом сценарии, единственная проблема, что все добавочные элементы не выполнимы [10].

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

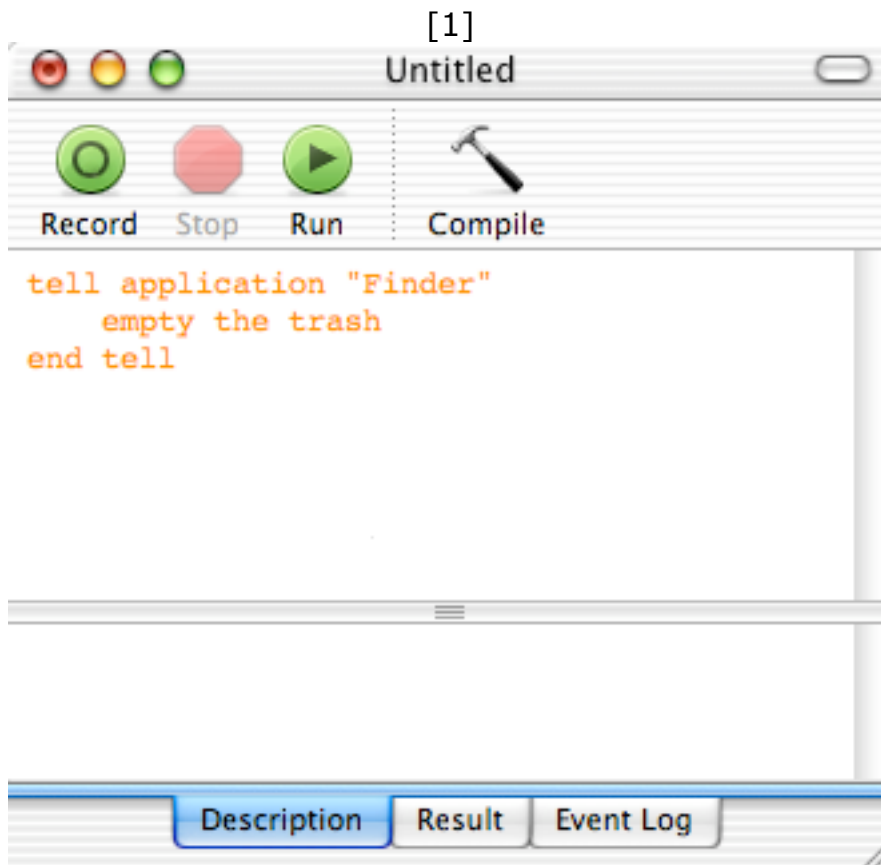
После очистки Корзины, AppleScript компонент Mac OS X остановится на строке [10.4] потому, что оно не адресовано Finder. Вы даже не услышите строки с сообщением [10.5], хотя в нем нет ничего не правильного или ошибочного (прим. Пер.: программа просто споткнется на первой ошибке и не будет выполнять все остальное).

ГЛАВА 2

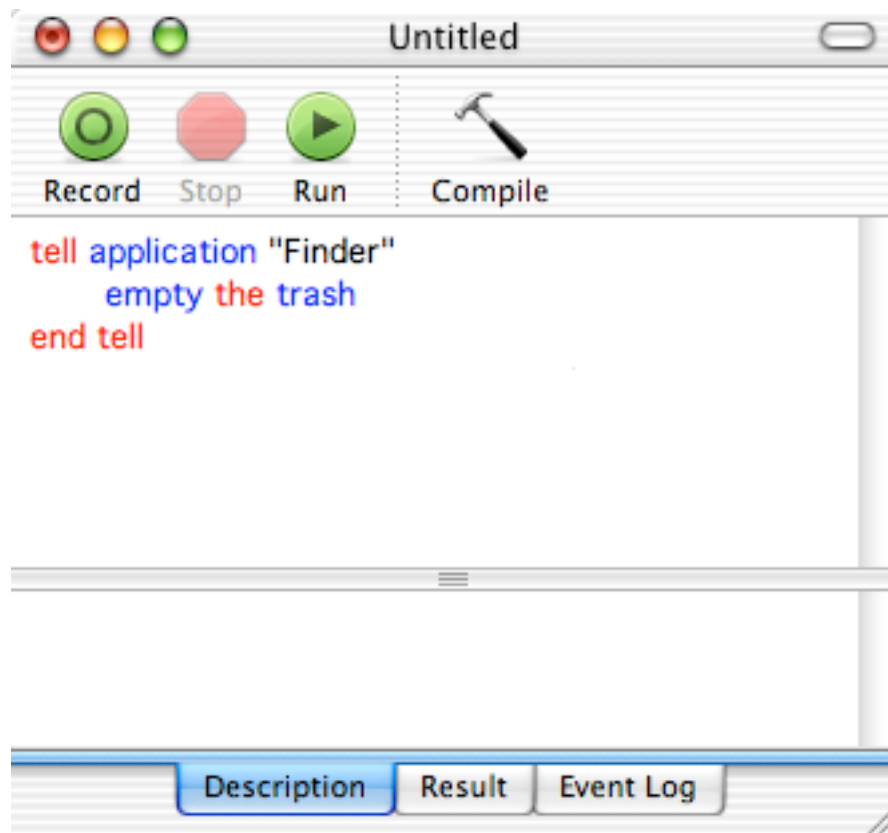
СОХРАНЕНИЕ И ВЫПОЛНЕНИЕ СЦЕНАРИЯ

Вы видели уже пару сценариев, и согласитесь, что они очень похожи на английский язык, делая сценарии легкими в понимании. Вы смогли выполнить сценарий – очистка Корзины – без использования мыши и клавиатуры. И Вы видели как Макинтош выполнил это для Вас.

Script Editor это программа в которой можно записать и выполнить программу. Вы можете найти Script Editor в папке AppleScript которая в свою очередь находится в папке Applications. После ее запуска Вы видите два поля. Верхнее поле для написания сценария [1].



Говоря в середине верхней части окна Вы видите кнопку Compile. Пока AppleScript очень похож на английский, язык AppleScript базируется на английском языке. "Эй Finder! Брось в мусор!" или "Эй Finder, очисти мою корзину!" это Finder конечно не поймет. Нужен процесс compilation (прим.Пер.: создание программы, умышлено не перевожу), AppleScript компонент Mac OS X проверяет созданный сценарий и проверяет правильность его написания. Если все правильно, то текст сценария становится цветным. Необработанный текст выглядит оранжевым цветом, но после обработки текста (Compile) зарезервированные слова станут синими и красными.



Если сценарий не удалось создать (compile) в следствии ошибки, Вы получите сообщение о критической ошибке, с указанием места ошибки. Попробуйте убрать одну из двойных кавычек в сценарии [2] и Вы увидите сообщение об ошибке сценария AppleScript.

[2]

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

Если все правильно, тогда нажмите кнопку Run, и скрипт выполнится. Добавим огня с помощью Script Editor, выберите любой из ранних сценариев и попробуйте выполнить его!

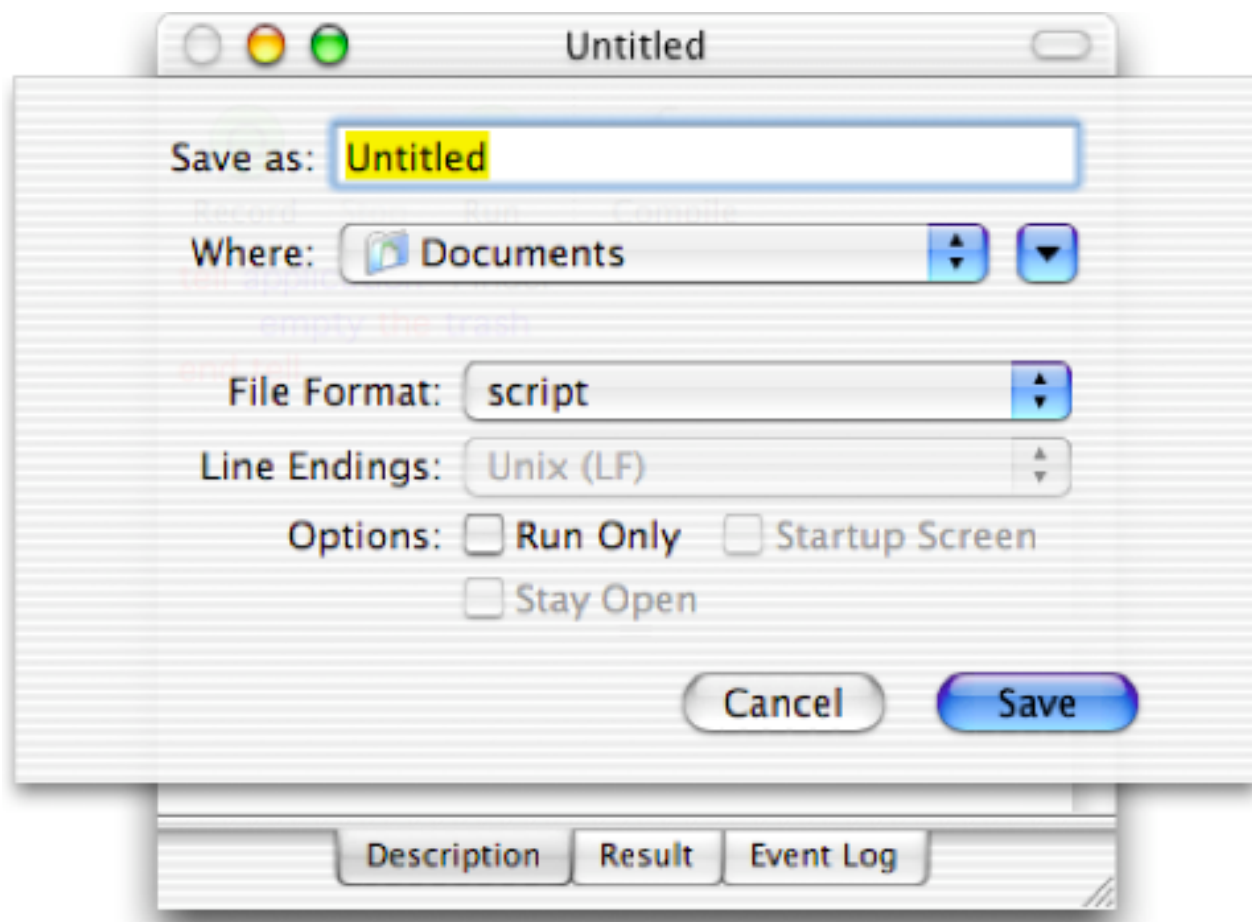
#Можно использовать клавишу Enter как «горячую клавишу» для окончательного создания сценария. Enter находится справа от клавиши Space (для ноутбуков) или в цифровой части настольного компьютера. Клавиша Return (около правой клавиши Shift) работает как переход на другую строку, и создает новую линию после текущей. Никогда не используйте клавишу Return для создания сценария. (прим.Пер.: мне кажется автор что-то напутал с положением клавиш, это Вам «задание на дом»). Можно сделать тоже самое если нажать на кнопку Compile после окончания сценария. Если Вы сразу нажмете кнопку Run, сначала проверяется правильность сценария, и только потом его выполнение. Заменить нажатие кнопки Run, можно заменить на нажатие Command-R.

Примечание: In reality, compiling involves more than just checking the syntax, but that is not your concern. #

Сохранение сценария

Для того, чтобы потом воспользоваться Вашим сценарием – сохраните его. Если сценарий не откомпилирован, Вы можете только сохранить сценарий как скрипт.

Если нет никаких проблем компиляции сценария, тогда в диалоговом окне можете не только сохранить сценарий, но и сохранить его как compiled script или как программу.



COMPILED SCRIPT: Если дважды щелкнуть на иконке compiled AppleScript,



откроется Script Editor и можно выполнить сценарий после нажатия на кнопку Run.

ПРОГРАММА: Если дважды щелкнуть на иконке AppleScript программы,



сценарий выполнится сразу. Причем Script Editor даже не откроется. Сценарий сохраненный как программа Вы можете вставлять в окно автозагрузки (в Вашем System Preferences). После того как Вы войдете в систему, такой сценарий выполнится автоматически. А если бы сохранили как скрипт, то выполнение программы не произошло бы, а вместо этого была запущена программа Script Editor, и загружен сценарий для дальнейшего выполнения.

#ПРЕДУПРЕЖДЕНИЕ: Установка в диалоговом окне может лишить возможности редактирования Вашего сценария, опция: run-only. Сделайте резервную копию вашего скрипта, прежде чем сделаете это, иначе не сможете отредактировать такой файл. #

ГЛАВА 3

ПРОСТОЙ СЦЕНАРИЙ (I)

В Главе 1 Вы уже познакомились со сценарием:

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

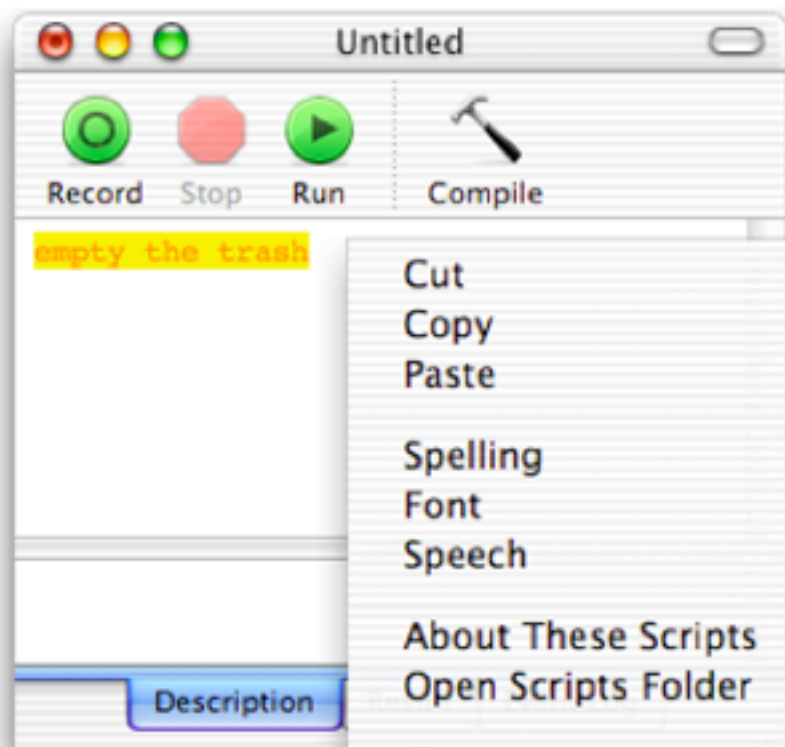
Как видите AppleScript и редактор Script Editor делают легким для чтения сценарий. В первой линии **tell block**, Вы можете заменить слово **application**, на такую короткую фразу:

```
tell app "xyz"
```

Во время создания сценария, Script Editor заменит **app** на **application**. Даже лучше, если Вы напечатаете по буквам и проверите имя программы **xyz**. Справедливо будет проверить (не принадлежит ли имя другой программе), например **pqr**. Если Вы создаете сценарий, AppleScript обеспечит Вас всем перечнем программ которые поддерживают язык сценариев. Вы можете выбрать программу и AppleScript заменит **pqr** на правильное имя программы, а потом запишет концовку блока **tell**. (прим.Пер.: проверил – **app** заменил на **application**, в кавычках заменил на программу из списка предложенных, а фразу **end tell** не добавил, а только указал, что это надо обязательно сделать)

Фактически, Script Editor позволяет Вам создавать **tell block** без полного набора слов, можно использовать контекстное меню. Его можно вызвать указав на тексте, нажать на клавишу control и потом щелкнуть мышью. Все это можно выполнить в два этапа:

- 1) Control и щелчок на верхнем поле Script Editor. Появится контекстное меню (см. Иллюстрацию на следующей странице), посмотрите в самом конце списка, Вы увидите меню с надписью **Tell Blocks**. После щелчка мышью появится дополнительное меню, выберите **Tell "Finder"**.
- 2) Если сценарий уже содержит одну или более директив **Finder** – например только **empty the trash** - но **tell block** отсутствует, выделите этот элемент и повторите шаг 1). Script Editor вокруг директивы **empty the trash** создаст законченный **tell block**, как в сценарии [1].



Cut
Copy
Paste

Spelling
Font
Speech

About These Scripts
Open Scripts Folder

- 📄 About these scripts...
- 📄 Choose
- 📄 Comment Tags
- 📄 TIDs
- 📁 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 ЦИФРАМИ

Школьный пример счета, заполните точки результатом:

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

В более старших классах, точки заменяют уже на переменные 'x' и 'y'. Теперь если посмотреть в прошлое, почему нас пугало простое изменение точек на переменные.

2 + 6 = x
y = 3 * 4

AppleScript использует переменные. Переменные это ничего больше чем удобное имя содержащее данные и никогда не могут писаться цифрами. Имя переменных иногда еще называются идентификаторами (identifiers), поскольку обозначают, идентифицируют данные. Теперь два примера [1] переменных AppleScript, где им присваиваются конкретное значение, используется для этого команда **set**.

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

Имена переменных, сами по себе, не имеют специального смысла в AppleScript, человек сам использует имена переменных, чтобы легче было читать и понимать прочитанное. Это облегчает поиск неисправности в сценарии (ошибки в сценарии и программах традиционно называют bugs (прим. Пер.: по-русски наверно правильнее называть «блохи» - вместо «жуки», а поиск таких ошибок – «ловлей блох»)). Поэтому, избегайте использовать такие переменные как 'x'. Для примера, переменная для ширины картинки может быть названа **pictureWidth** [2].

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

Пожалуйста, обратите внимание, что переменная состоит из одного слова (или одионого знака). После проверки синтаксиса (прим. Пер.: правильности написания выражения), имя переменной становится зеленым, так Вы сможете увидеть – не зарезервировано ли это слово в AppleScript, они показаны в голубом или красном цвете. А также, заметьте что данные (цифра '8' в сценарии [2]) будут черными.

#Пока Вы имеете полную свободу как выбрать имя для переменной, позднее будет несколько правил по их использованию. Пока я могу их перечислить по пальцам.

Основное правило, которого Вам следует придерживаться, это не должна быть команда AppleScript или любое зарезервированное слово. Для примера, 'set', 'say', 'to', и 'beep' слова которые служат для специальных целей в AppleScript. Лучший путь создания имен переменных это придумывать слова со смыслом, как 'pictureWidth', тогда Вы в безопасности. Имя переменной должно легко читаться, поэтому переменная созданная из нескольких слов – лучше выглядит, если каждое слово начинается с заглавной буквы.

Если Вы настаиваете – выучить пару правил, доберитесь до конца параграфа. Использовать цифры в начале переменной нельзя, а лучше всего начинать имя переменной с подчеркивания _, например: _pictureWidth #

Теперь мы научились присваивать переменной значение. AppleScript в состоянии выполнять базовые математические вычисления, поэтому нам будет достаточно чтобы предложить программе готовое решение, вместо того чтобы заставлять ее считать, например размер иллюстрации. Теперь сценарий [3] покажет это.

[3]

```
set pictureWidth to 8
set pictureHeight to 6
set pictureSurfaceArea to pictureWidth * pictureHeight
```

Используются для вычисления следующие символы, официально используемые операторы, для базовых вычислений.

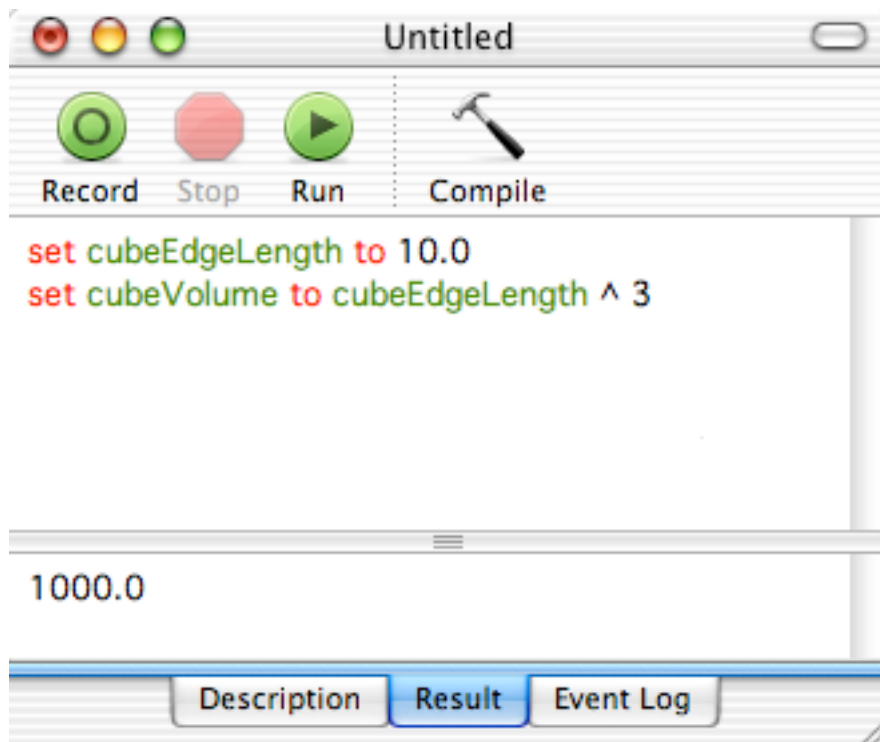
- + для сложения
- для вычитания
- / для деления
- * и для умножения

Степень записывается символом ^. Теперь сценарий [4] который высчитывает площадь куба.

[4]

```
set cubeEdgeLength to 10.0
set cubeVolume to cubeEdgeLength ^ 3
```

Если выполнить этот сценарий [4] в редакторе Script Editor, то результат можно будет увидеть в нижнем поле редактора. Если Вы не видите результат, то укажите курсором закладку Result среди горизонтальных закладок. Вы увидите результат вычислений. Если сценарий содержит только один элемент вычислений [4.1], то результат будет равен 10.0. Для всего сценария [4], результат равен 1000.0. Это результат переменной cubeEdgeLength ^ 3 после математической операции, который и виден на экране (иллюстрация на следующей странице).



Цифровые значения разделяются на несколько типов: целые значения и с плавающей точкой. Вы видите целое значение переменной в сценарии [1.1] и с плавающей точкой [1.2]. Целые используются для счета (прим. Пер.: для счета например овец, где овец нельзя посчитать половинками, только целиком), что мы будем делать далее, для того чтобы посчитать используя серию инструкций (см. Главу 13). Для простоты будем называть их «целыми», для примера, этот как счет в баскетболе. Значение может быть положительным или отрицательным, это можно увидеть на примере Вашего счета в банке.

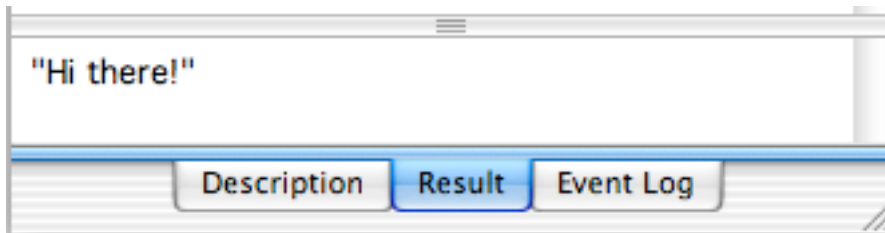
ГЛАВА 5

РАБОТА С ТЕКСТОМ

Переменные содержат не только цифры. Пришло время попробовать использовать текст для этих целей. Какой бы текст ни был, равным нулю или состоящим из одного знака, называться он будет – «строка». Строка заключается в двойные кавычки. Для примера, три строки [1], где переменным задекларированы строковые значения.

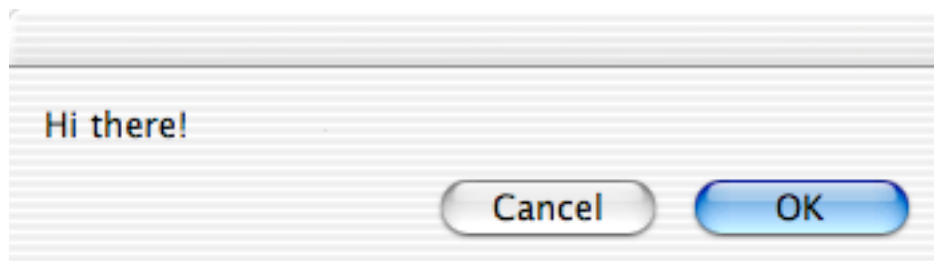
```
[1]
set emptyString to ""
set notEmptyContainsASpace to " "
set greeting to "Hi there!"
```

После выполнения сценария [1], результат можно увидеть в двойных кавычках. Так, после результата вычислений можно понять, что значение строка, а не цифра (цифры пишутся без кавычек, а строка только с кавычками).



После выполнения последнего результата, все переменные окрасятся в зеленый цвет [1.3] на экране.

В дополнение к полю Result, AppleScript предлагает очень удобную альтернативу: диалоговое окно. Выглядит это так:



Вы можете воспользоваться командой: `display dialog` с указанием данных (цифровое или строковое значения) для демонстрации результата. Такой диалог показан в сценарии [2].

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

Почему строковые значения имеют двойные кавычки? AppleScript может состоять только из очень ограниченной технологии, и читать Ваш скрипт, а потом интерпретировать его в инструкцию для выполнения, будет все еще трудно компьютеру. Поэтому, AppleScript полагается на ключи, чтобы понять заявленное в сценарии. Только по этой причине мы должны строковое значение помещать в двойные кавычки. Конечно, AppleScript без кавычек, может ошибиться в определении значения. Посмотрите следующий сценарий [6]:

[6]

```
set stringToBeDisplayed to "Hi there!"  
display dialog "stringToBeDisplayed"  
display dialog stringToBeDisplayed
```

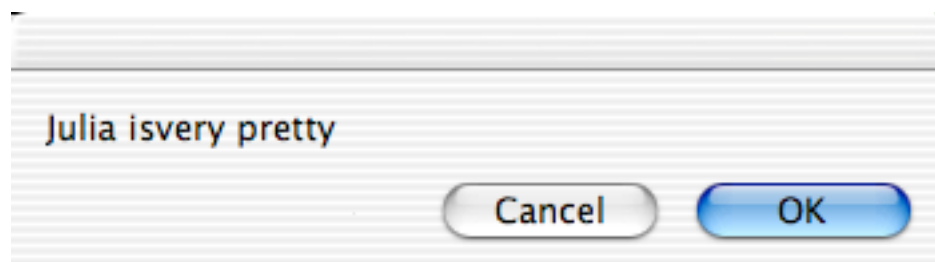
Выполните этот сценарий. Часть скрипта [6.2] покажет на экране сообщение как строку "stringToBeDisplayed", в то время как [6.3] покажет "Hi there!". Script Editor укажет цветом откомпилированные строки, тогда как строка со значением [6.3] **stringToBeDisplayed**, будет выглядеть зеленой, то в элементе строки [6.2] "stringToBeDisplayed" показана черным, это демонстрация переменной и строкового значения. Время от времени, цветастая раскраска поможет Вам быстрее «ловить блох» в сценарии.

После выполнения, AppleScript расшифрует значения в близком к английскому языку, чтобы легче понять написанное для Макинтош. Здесь другой пример, почему эти значения важны: если мы напишем "thirty" как цифровое значение, но с двойными скобками, т.е. как "30", тогда значение станет строковым. Указать тип данных – это очень важно, потому что некоторые операции можно выполнить только на конкретном типе данных. Для примера, Вы можете разделить 2 цифры, но Вы не можете сделать тоже со строковыми значениями. Как цифрами, строковыми нельзя манипулировать. Вы можете склеить строковые значения, как на операции ниже, используя амперсant [7].

[7]

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

В третьем элементе [7.3], мы наблюдаем три строковых значения, два из которых - переменные.



Пожалуйста заметьте, что количество пробелов между строковыми и амперсантом не зависит на результате переменной `resultingString`. После компиляции, Script Editor заменит лишние пробелы на один единственный, если Вы добавили больше чем один. Если Вы добавите один или несколько пробелов в слова, которые нужно вывести как предложения, то нужно это сделать внутри двойных кавычек. В части [7.3], присутствует пробел слева от слова " is", в то время как после буквы "s" слова "is", нет ничего.

Будет еще больше команд которые можно выполнить над строковыми. Некоторые из них требуют дополнительных знаний, поэтому мы встретимся с ними позже. Пока это не для новичка. Но мы можем дать Вам другой пример работы со строковыми. Вы можете узнать длину строки выражения [8].

[8]

```
set theLength to the length of "I am"
```

Если Вы выполните этот сценарий, то результатом будет 4. Также обратите внимание, что в значение включен и пробел, а не только буквы. Потому, что двойные кавычки дают команду о начале и конце строки. Вы думаете, что строка не может содержать двойные кавычки? Конечно, язык AppleScript воспринял бы это как конец строки. Тогда добавьте косую черту перед двойной кавычкой, и AppleScript не споткнется читая такую строку [9].

[9]

```
set exampleString to "She said: \"Hi, I_m Julia.\""
```

#Если Вам нужно вывести текст в кавычках, ваша строка должна содержать двойные кавычки после обратной черты. Предположим Вы хотите вывести такой текст:

```
blah blah \" blah.
```

Первое, Вы должны поставить косую черту до косой черты. AppleScript игнорирует специальный знак до следующей косой черты. Конечно, мы должны поставить двойные кавычки, иначе AppleScript думал бы, где конец строки? Следовательно двойные кавычки должны быть после косой черты, как мы видели раньше. Если все понятно, обратимся к сценарию [10].

[10]

```
display dialog "blah blah \\" blah"
```

Для Вашего удобства, я показал косую черту жирной, хотя Script Editor этого не делает. Надо еще сказать, что косая черта используется со специальными знаками, для специальных действий. Пример, `\\n` создает новую линию (аналогично Return), и `\\t` создает табуляцию (аналогично Tab).#

Как сказано выше, цифровые и строковые значения различны. Вы не можете вычесть из строки цифру 3, как на примере [11].

[11]

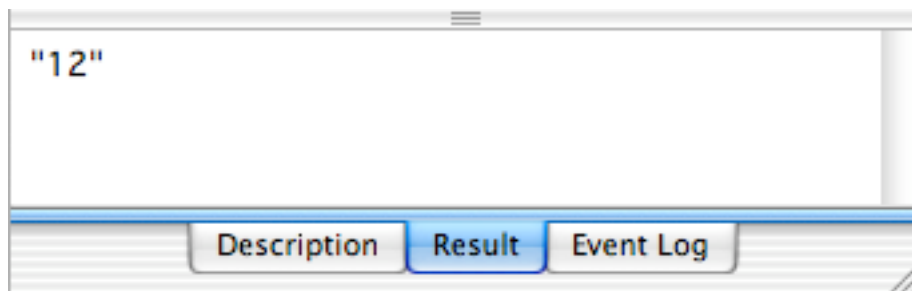
```
set nonsensical to "fifteen" - 3
```

Если выполнить сценарий [11], Вы поймете насколько дружелюбен язык AppleScript. Он практически пытается преобразовать строку в цифру. Если строка содержала бы "15", а не "fifteen" – тогда конверсия была бы возможной. Преобразование одного данного к другому вызвано принуждением. Вы можете принудить к этому как показано на примере [12]

[12]

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

В поле для результатов программы Script Editor виден результат последних вычислений, части сценария [12.2]. Тогда как переменная `coercedToNumber` является строкой, это указано двойными кавычками, но объявлена как цифровое значение, с помощью `as number`, а переменная `coercedToString` объявлена как строка, с помощью `as string` (см. Иллюстрацию результата ниже).



Для сценария [12.1], результат 15 был бы без двойных кавычек, что показало бы нам, что это цифра, а не строка.

ГЛАВА 6

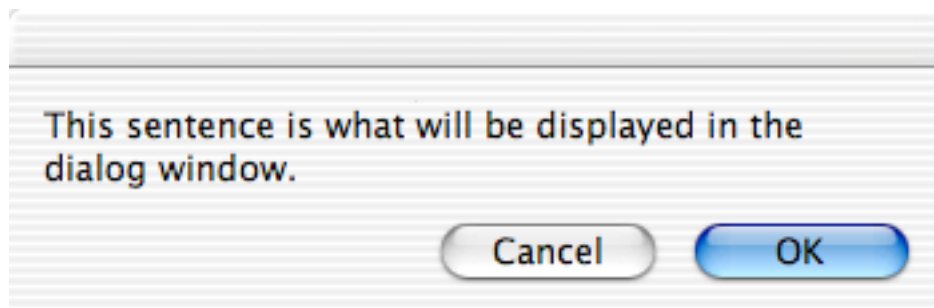
СПИСКИ

В предыдущих главах, Вы видели как создается простой сценарий для числовых операций и работой над строками. Сценарий пользователя представлен с помощью `display dialog` команды [1]. В этой и следующей главе мы будем использовать команду `display dialog` для изучения больших возможностей языка AppleScript.

[1]

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

Если выполнить сценарий [1], Вы увидите диалоговое окно, с кнопкой по умолчанию для выбора между: кнопками Cancel и OK.

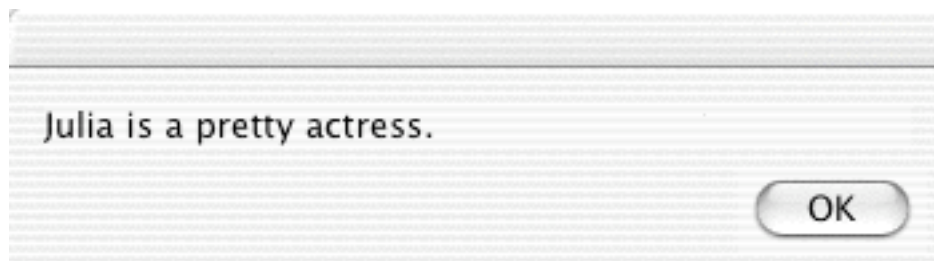


Кнопка Cancel отменяет дальнейшее выполнение сценария. Вышеуказанный сценарий, на самом деле, не выполняет никаких дополнительных действий и кнопка Cancel здесь просто лишняя. В диалоговом окне можно назначить кнопку по умолчанию, в нашем случае это кнопка OK [2.2].

[2]

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

Если выполнить данный сценарий, то мы не увидим кнопки Cancel, она отсутствует.



Если посмотреть вторую строку [2.2], то мы увидим список действий в этой строке: {"OK"}, заключенный в фигурных скобках. Зачем стоят эти фигурные скобки? Как

мы видели ранее, AppleScript использует для простоты понимания определенного типа элементы. Такие фигурные кавычки нужны AppleScript чтобы распознать в данных – список. Элемент списка [2.2] в данном примере, существует только один, строка {"OK"}. Если список состоит из нескольких элементов, такой список разделяется запятой [3] .

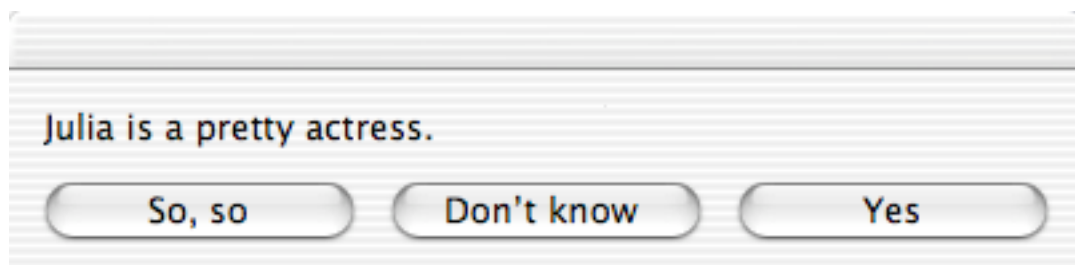
[3]

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

Список элементов в строке [3] состоит из 4 элементов: одна строка и три цифры. Создайте диалоговое окно с множеством диалоговых кнопок. Команда `display dialog` может состоять из одной, двух, трех кнопок (а скоро еще из кнопки по умолчанию). Чтобы создать диалоговое окно с тремя кнопками, нужно создать список из трех элементов [4.2].

[4]

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



#Вы заметили, что ни одна кнопка в диалоговом окне не выделена [2.2, 4.2] Мы теперь побеспокоимся об этом, чтобы наша программа была удобна для пользователя. Подсветим кнопку по умолчанию [5].

[5]

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

Вместо использования имени кнопки, Вы можете использовать номер этой кнопки, по тому порядку в котором она стоит [6], т.е. третья в данном списке.

[6]

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

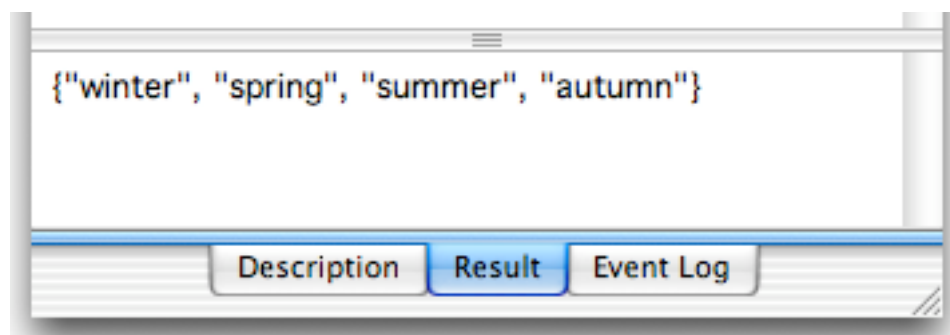
В следующей главе Вы научитесь определять – какая кнопка была в результате нажата. А пока мы продолжим изучать списки. Списки можно использовать для

того, чтобы использовать серию данных. Вы научитесь как редактировать список и как извлекать данные из списка. Очень легко добавить данные к началу и концу списка. Для того, чтобы добавить данные к списку мы используем амперсant, как мы их используем для строковых значений.

[7]

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

В примере [7.4] мы создали список из четырех элементов. И в результате мы получим список из четырех строковых элементов.



Обратите внимание, что `addToBegin` и `addToEnd` имена переменных [7.1-2], выбраны для того, чтобы помочь Вам понять сценарий и они не делают ничего, кроме того, что хранят определенные данные. Зеленый цвет указывает, что эти имена являются переменными.

Вы можете определить каждую деталь в списке номером. Первым номером будет "winter", вторым "summer" и т.д. Это позволит Вам изменять значения в списке или изменять список по номеру. Смотрите пример[8].

[8]

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

Команда `get` покажет последний элемент [8.3], она покажет последнее значение переменной `myList`. Это значение переменной `myList` будет {"winter", "spring"}. Обратите внимание на вторую строку сценария [8.2], где меняется значение второго элемента в списке [9] из первой строки списка [8] (прим. Пер.: Запутались? Второй элемент списка это "summer", который меняется на "spring").

[9]

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


Первый элемент [9.1] элегантно написан, практически на английском языке. Это учтенное значение можно использовать до десяти. Потом только так - **item 11**, и так далее. Альтернатива этому, можно описать как **11th item**, т.е. как в примере [9.2]. К последнему в списке можно обратиться как **last item** [10].

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

Поэтому, Вы избавляете себя знанием, сколько значений в списке, каково его последнее значение (или установить ему другое значение, не зная каким оно идет по счету).

AppleScript позволяет Вам устанавливать значение путем отсчитывания в обратном направлении, т.е. справа налево. С этого конца, используйте отрицательные числа, где -1 будет последним значением, значение -2 вторым с конца, ну и так далее. Сценарий [11] делает тоже самое, что и сценарий [10], только по другому.

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

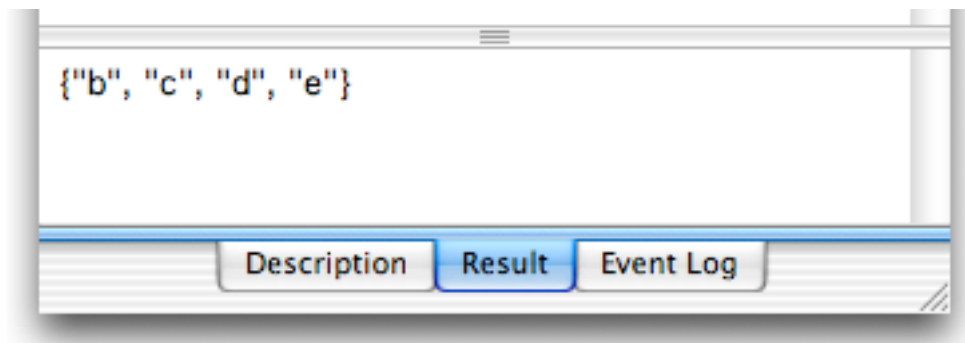
Теперь Вы знаете как создать список, как добавить к нему значения, и как их заменить на другие данные. Вы также умеете получать одиночное значение из списка. Но Вы еще будете желать создать большие списки, чем ранее [12].

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

Элемент [12.2], можно написать **thru** которое интерпретируется как **through**, если захотите. Если Вы измените порядок в номерах [13.2] в определенном ряду [13].

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

Тогда результат будет [13] будет такой же, что и в сценарии [12].



(прим. Пер.: нет никакой разницы считать значения с 2 до 5 или от 5 до 2, значение будет тоже самое, даже порядок данных не изменится)

Чтобы изменить порядок данных используется команда `reverse` [14].

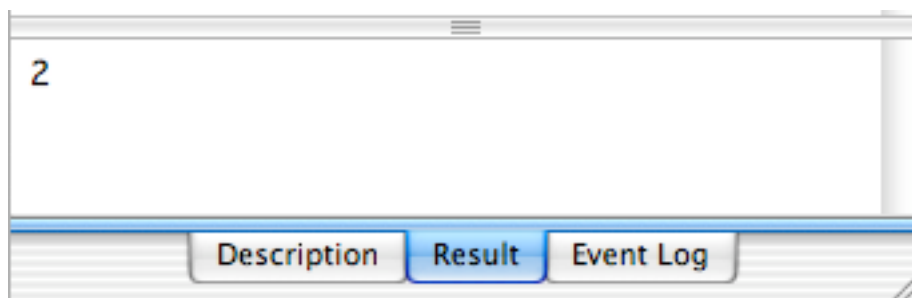
[14]

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

Время от времени, Вы захотите узнать – а насколько велик мой список? Ответ прост [15], используя один из двух вариантов строк.

[15]

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



И наконец, данные можно получить случайным образом [16].

[16]

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

Какой будет результат? Как получится – случайным образом.

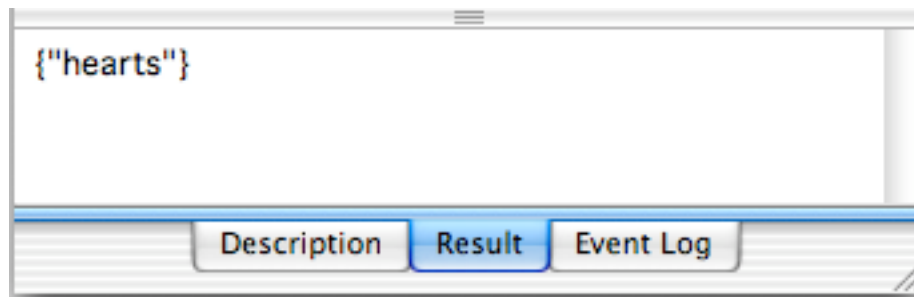
Я знаю, что эта глава длиннее остальных, но мы специально обсуждаем, как делать определенные действия со списком и строковыми. Мы обсуждали как работать со строковыми в главе 5, а сейчас обсуждаем списки, но мы их не обсудили до конца, так что держитесь или передохните пока.

В предыдущих главах Вы учили как преобразовать один тип данных в другой. Сейчас мы увидим как изменить строковое значение в цифровое и наоборот [17].

[17]

```
set cardType to "hearts"  
set stringAsList to cardType as list
```

После этого мы получим одиночное значение как строковое (см. Иллюстрацию на следующей странице):



Общась с обоими списками и строковыми значениями, как следствие будет важная мера безопасности, которую продемонстрирую здесь. По мере того как Вы вспомните как амперсant использовался для объединения строковых. Необходимо использовать амперсant чтобы добавить строковое значение к списку [18]?

[18]

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

Тип данных переменной theResult зависит от типа данных, которые можно понять в выражении [18.3]. Потому, что выражение начинается с переменной myList которое является списком. Попробуйте проверить поле Result. Оно показывает тип данных в фигурных скобках. Если мы обратим внимание, то увидим порядок следования переменных myList и myString, и запишем [19.3]:

[19]

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

результатом будет строковое значение. Сценарий выполняется без указания на ошибку, идет автоматическое преобразование значения в строковое. Для того, чтобы не получить случайное значение, сделайте следующее [20].

[20]

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

В части [20.3], строковая "b" присваивается переменной myString, которая далее преобразуется в список. Результатом преобразования myString в список, будет theResult в сценарии [20.3]. Для того, чтобы добавить один или несколько элементов к списку, без конкатенации, сделайте следующее, что быстрее: #

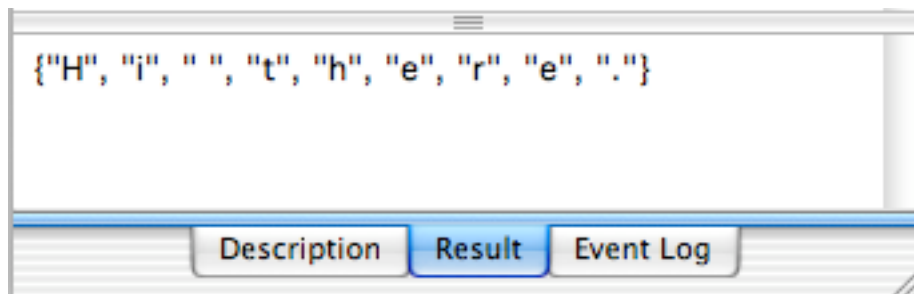
```
set myList to {1, 2, 3, 4}  
set the end of myList to 5  
get myList
```

Если без принуждения сделать из строки список, а также возможно создание списка, каждый элемент которого будет содержать строковые значения [21]. (Примечание: Пока используется `item` вместо работающей `character`, но оно противоречит Mac OS X).

[21]

```
set itemized to every character of "Hi there."
```

Результат – список, который видно в поле Result:



Можно использовать одиночные строковые значения (прим. Пер.: просто буквы, например: «а», «б» и т.д., в том числе и знаки препинания), чтобы разрезать предложение на слова. Это можно сделать используя разделители между слов. Вы определяете эти разделители, которые могут быть между словами. Слова войдут в окончательный тип – список. Для того, чтобы разрезать предложение на слова, мы будем использовать разделитель – пробел между словами. Смотрите сценарий [22]. Хороший сценарный стиль AppleScript требует от Вас заменять текстовые разделители [22.3], чтобы можно было вернуться к первоначальной версии предложения [22.5].

[22]

```
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
```

Обратите особое внимание на часть сценария [22.4], где видно `text item`, а не `just item`. Это будет часто совершаемой ошибкой. Учите на зубок `text item text item text item`. Это понятно?

Очень легко соединить список в строковое значение [23].

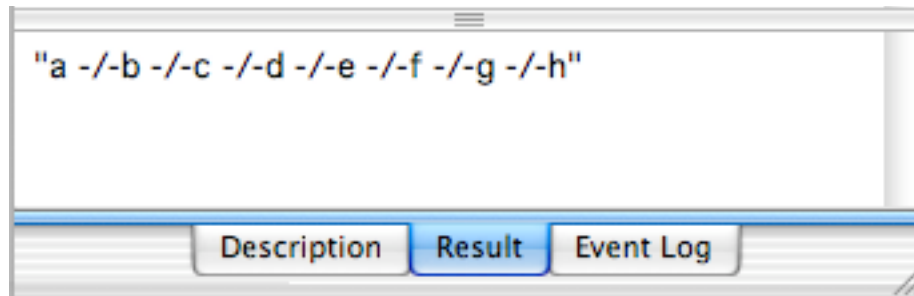
[23]

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

Если определить знак или серию знаков в строке и выбрать из списка, то вы должны установить такие знаки разграничители в тексте [24].

[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
```



Складываем выученное в сценарии [22] и сценарии [24]. Теперь у Вас есть возможность найти разграничители в тексте и заменить их на строку.

#Причина, почему нужно восстановить разграничители в тексте, ранее установленные в сценарии AppleScript, кроется в том, что другие сценаристы могут быть менее тщательны в разработке сценария. Их сценарий может содержать другие знаки разграничители в тексте AppleScript. Изменение Вашего сценария сохранено в Mac OS X как части AppleScript чтобы потом Ваш скрипт закончить. #

Возразите мне: Есть несколько типов данных, таких как цифры, строки и списки. Каждый тип данных имеет своих собственных операторов, которые Вы можете использовать для выполнения своих целей. Есть множество операторов для выполнения операций над списками. В этой главе, посвященной спискам, мы учились работать со списками или с двумя строками, которые мы не обсуждали ранее.

ГЛАВА 7

ЗАПИСИ

В предыдущей главе, мы использовали команду `display dialog` для вывода на экрана результатов работы списка. Теперь мы можем попробовать поработать с другим типом данных.

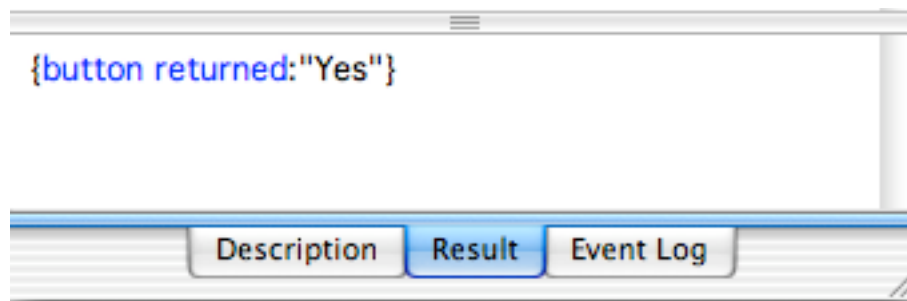
Команда `display dialog` работает с конкретными данными, цифрами, строками и списками, используя для диалога кнопки, помните еще? [1.2].

```
[1]
set stringToBeDisplayed to "Julia is a pretty actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don_t know", "Yes"}
```

Но как мы узнаем, какая кнопка была нажата? Проверьте сами [2]

```
[2]
set stringToBeDisplayed to "Julia is a pretty actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Don_t know", "Yes"}
```

Когда Вы выполните сценарий [2], Вы сможете увидеть результат в окне Result. После того, как Вы нажмете кнопку, Вы сразу увидите результат:



Этот результат содержит другой тип данных, называется – запись. Можно подумать, что это список, из-за фигурных скобок, но этот результат состоит из двух частей, разделенных двоеточием. Каждый элемент в записи называется свойством, а также содержит имя и значение. Первая часть значения это метка (прим. Пер.: в оригинале – label, может некоторым, кто привык использовать англоязычный интерфейс Mac OS X, так будет привычнее), или имя кнопки; как здесь: `{button returned: "Yes"}`, а также вторая часть истинное значение свойства (здесь значение это строка "Yes"). Потому, что вторая часть значения, которую редактор Script Editor показывает черным.

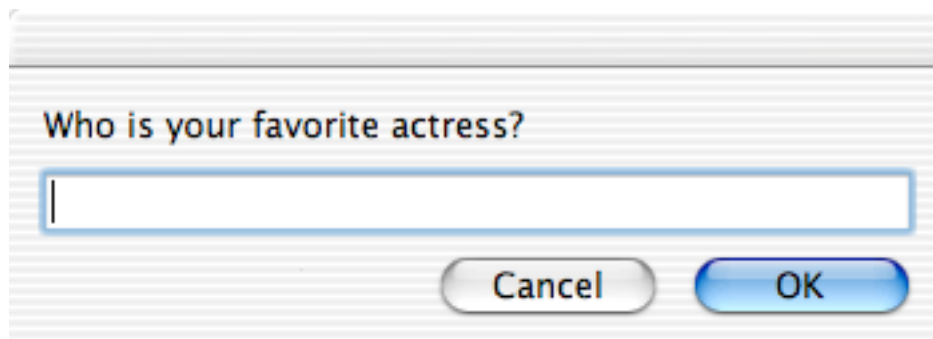
Когда определена нажатая кнопка, мы можем узнать значение свойства имея метку `button returned` [3.3].

[3]

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

В части [3.3] мы специально вставили переменную `theButtonPressed`, где значение равно метке `theButtonPressed` в записи `tempVar`. Ай, да мы! Мы теперь знаем какая кнопка будет нажата.

Диалоговое окно ограничено размером и показывает цифры или строки, очень короткие. И не показывает списки и записи. И совершенно наоборот, показывает результат который может быть любым типом данных. Однако, `display dialog` может сделать что-то очень полезное: он может позволить пользователю вписать цифровые или строковые данные, когда Ваш скрипт выполняется.

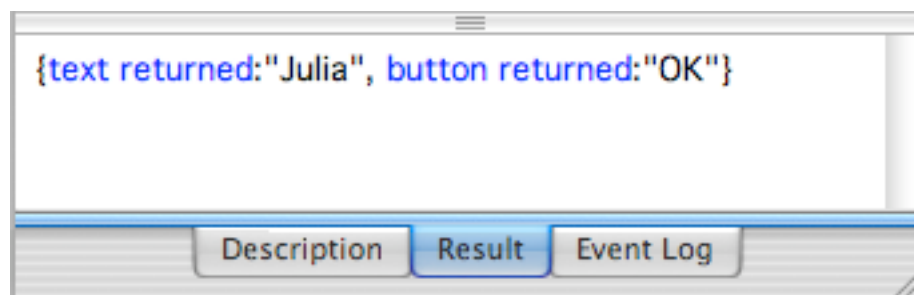


Которое создает пустое окно для ввода. Вы можете ввести любой ответ (прим. Пер.: имеется ввиду - `default answer`), например пустую строку "" [4].

[4]

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

Если выполнить сценарий [4], результат которого запись из двух значений, т.е. два имени / значения, запись – которую можно посмотреть в поле Result (в зависимости от написанного сценария):



Обратите внимание, что значение списка разделено запятой. AppleScript не

покажет запись как список, из-за двоеточий. В противном случае список, где Вы можете запомнить ту часть информации, как фактически данные, которые можно извлечь из записи метки и это очень легко. Извлечь имя `textEntered`, которое мы можем получить от значения помеченного как `text returned` [5.2]

[5]

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

#Текстовое значение возвращается как строковое, но часть текста, будет цифрой, если пользователь ввел число. Например, если ввести число 30, значение переменной `textEntered` не 30, а строковая "30". Если вы хотите сделать вычисление, то Вам повезло! AppleScript автоматически преобразует данные [6.3]. Но Вы не должны включать [7.1] для принуждения действия после [6.2].

[6]

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

[7]

```
set ageEntered to ageEntered as number
```

Представьте, что пользователь введет не цифру 30. То есть, пользователь ввел "тридцать" или "30 лет". AppleScript не знает таких данных, и скрипт выполнится не правильно. Потому, что Вы этого не предвидели, что вводимые данные могут быть совершенно различными. В главе 10 Вы этому научитесь. #

Вы не создаете запись, чтобы установить переменную имя/значение [8].

[8]

```
set personalData to {age:30}
```

Обратите внимание, что цвет `age` зеленый, т.е. определено Вами. Данные напротив, показаны черным цветом.

#Вместе с выполнением сценариев [3, 5], обратите внимание на значение метки `button returned` и `text returned`. Вы можете это увидеть синим цветом, а само значение в два цвета. Если Вы создаете запись, Вы не можете использовать два или больше слов для определения значения [9.1]. Метка (или имя) состоит из одного слова. [9.2]

[9]

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

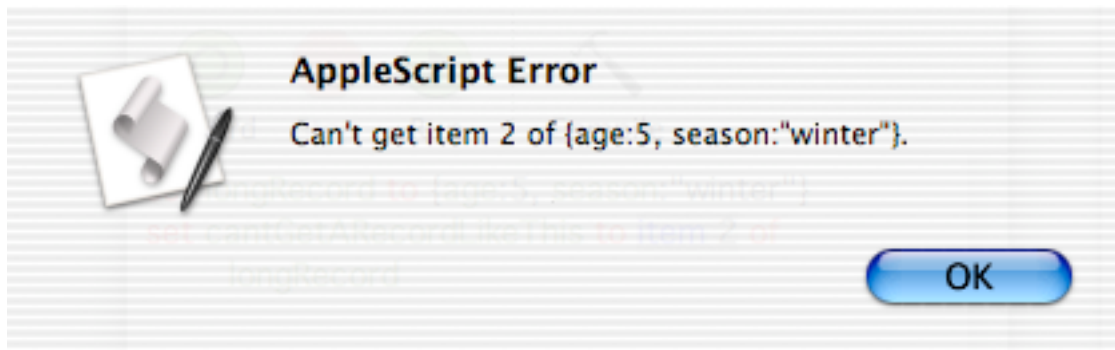
#

Вы можете объединить записи в списки, но прежде: Если записи, которые Вы объединяете имеют конкретные метки свойств, то результат, надеюсь, Вы отличите.

Пожалуйста не вызывайте свойства записи `item`, потому что даже следующий сценарий [10] проходит синтаксическую проверку, AppleScript не позволит Вам определить как пару запись имя/значение как деталь [10.2].

[10]

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



Вы можете подсчитать больше свойств как в записи ниже [11]:

[11]

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

Тогда создайте новую запись содержащую значение другой записи, создайте запись как показано ниже [12].

[12]

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

Результатом будет новая запись `{age:5}`. Если записать в сценарий [12] больше значений, но Вам будет трудно его сначала прочитать[13].

[13]

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

К несчастью не возможно обозначить некоторые пары имя/значение в этой записи. Тогда, Вы не можете создать список всех значений меток. Подобно тому, как изменить метки в записи. Если Вы хотите использовать другую метку, Вы должны создать новую запись, как в сценарии [13].

#Когда закончится эта глава с одним гадким чувством. Где скрипт не должен заканчиваться сюрпризом [14].

[14]

```
set firstValue to 30
set rememberFirstValue to firstValue -- копия сохраняется как 'rememberFirstValue'
set firstValue to 73 -- Изменяет значение в переменную
get rememberFirstValue -- Запрашиваем значение 'rememberFirstValue'.
```

Результат 30. Для записи (и списков!) это значение совершенно разное, хороший результат - это свобода от ошибок. Проверьте это из сценария [15]

[15]

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

Результатом будет {age:73}! ! ! Команда set никогда не создаст копию, если переменная содержит запись или список. Чтобы создать копию данных, Вы можете использовать команду copy [16].

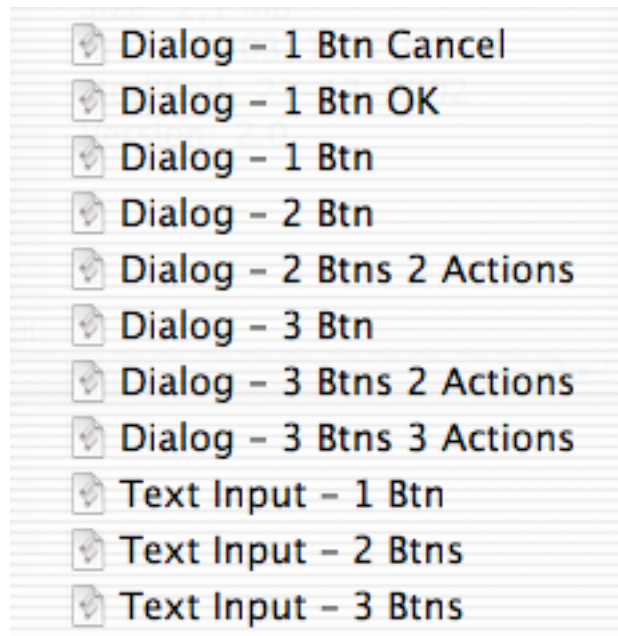
[16]

```
set personalData to {age:30}
copy personalData to rememberPersonalData
set age of personalData to 73
get rememberPersonalData
#
```

ГЛАВА 8

ПРОСТОЙ СЦЕНАРИЙ (II)

В предыдущей главе мы видели варианты простого диалогового окна. Если Вы запомнили все варианты, безусловно Вы мой гость. Если Вам весь этот путь кажется легким, запомните еще Control-щелчок в верхнем поле редактора Script Editor чтобы вызвать контекстное меню. Первым в меню стоит Dialogs. Щелкните и Вы увидите следующее выпадающее меню:



Где, Btn стандартная кнопка. Цифра обозначает - сколько использовано кнопок ответа. Пока не буду объяснять меню с пометкой Actions. Вы это поймете как мы доберемся до главы 10. Последнее дерево меню содержит ввод текста (Text Input). Редактор Script Editor, напишет следующее:

```
set temp to
```

Нет ничего после команды **to**. Сейчас, Control-Щелчок справа после пробела от **to**, и выберите Text Input - 2 Btns. Это создаст диалоговое окно, где пользователь в дальнейшем сможет ввести данные, как мы уже говорили в главе 7 когда мы обсуждали результат **text_returned**.

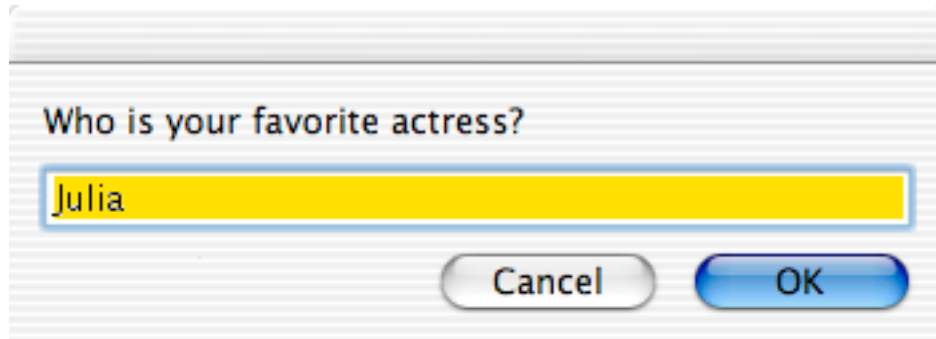
Если Вы хотите конечно, **display dialog** может создать ответ по умолчанию, и пользователь в сценарии не изменит ничего самостоятельно. Если такой поступок оправдан, то он делает сценарий более дружелюбным пользователю [1.1]. И Вы конечно знаете, что пользователи Макинтош очень любят это. Но если ответ по умолчанию не оправдан для значения, то он все еще дает ответ о предполагаемом типе данных.

[1]

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

Если выполнить сценарий [1], Вы увидите следующее диалоговое окно. Если Вы согласны – нажмите Enter.

Конечно, Вы можете написать другое имя.



Если кратко, нет необходимости запоминать все варианты реинкарнации команды `display dialog`. Редактор Script Editor создает его для Вас, и Вы можете добавить их к Вашему сценарию, без набора текста на клавиатуре.

ГЛАВА 9

НЕТ КОММЕНТАРИЕВ? ПЛОХО!

Несколько причин, делают язык AppleScripts легким для чтения, написания и понимания. Одна из них, это близкое родство с натуральны английским языком. Однако, другими причинами могут стать описательный способ имен переменных. В этой главе мы обсудим другой важный фактор.

Пока мы использовали короткие сценарии для примеров. Сценарий AppleScripts который в последствии напишите Вы - будет более длинным. Основной момент, когда Вы пишете сценарий - определение того, что он должен сделать, но также правильно его задокументировать. Позднее, если Вы увидите сценарий и захотите его отредактировать, значительно легче будет - если он будет с комментариями. Вы уверены в действиях сценария, если он имеет комментарии. Можно убедить Вас, что время потраченное на комментарии - будет в последствии оправдано. Также это поможет разобраться другому человеку, если он взял прочитать Ваш сценарий.

Чтобы создать комментарий, начните с двух знаков тире.

```
-- Этот комментарий ПОСЛЕ компиляции (проверки синтаксиса), серый.  
-- И это комментарий  
-- Этот комментарий только в одну линию
```

В плохие дни, можно использовать комментарий в несколько строк, который стоит заключить в круглые скобки (* *)

```
(* Этот комментарий может быть больше чем две линии.*)
```

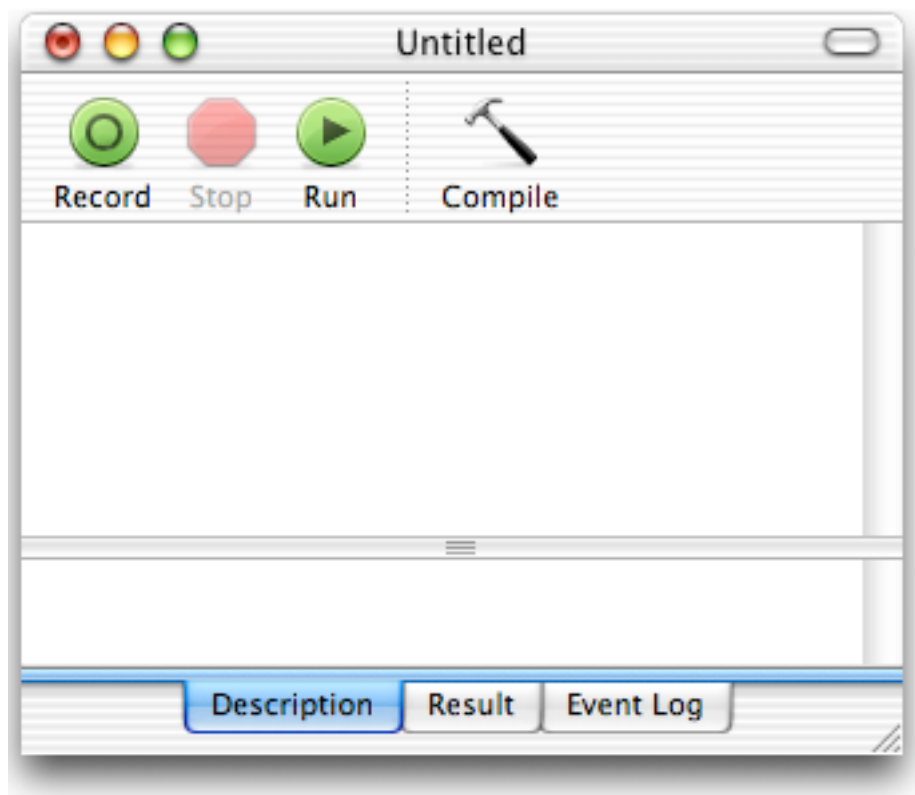
С появлением контекстного меню, можно рекомендовать больше. Вы должны использовать два тире для комментариев, в то время как в контекстном меню есть шаблон для комментариев.

Устанавливая комментарий типа скобок (* *), Вы можете таким способом временно вывести из строя часть сценария, для того чтобы проверить работу других частей. Этот способ позволяет Вам избегать проблем и ошибок. Если часть сценария не закрыта комментариями, например, значение части переменных, Вы можете выполнить часть сценария, чтобы выяснить результат части сценария.

С помощью контекстного меню редактора Script Editor сделать это просто. Выделите часть сценария, который требуется вывести из строя, например с помощью мышки. Теперь Control-Щелчок над этой частью, и выберите из меню Comment Tags. Теперь выделенный текст сценария будет выделен в комментарий. Повторное выделение этой части сценария и повторное выполнение этой

процедуры приведет к тому, что комментарий с данной области будет снят.

В редакторе Script Editor, удобно использовать табуляцию после той части сценария который стоит закомментировать. А в нижнем поле редактора Вы можете написать общий комментарий к Вашему сценарию.



Важность комментариев трудно недооценить. Все сценарии в этой книге можно закомментировать, но мы не делаем это, только потому, что вокруг сценария есть объяснения.

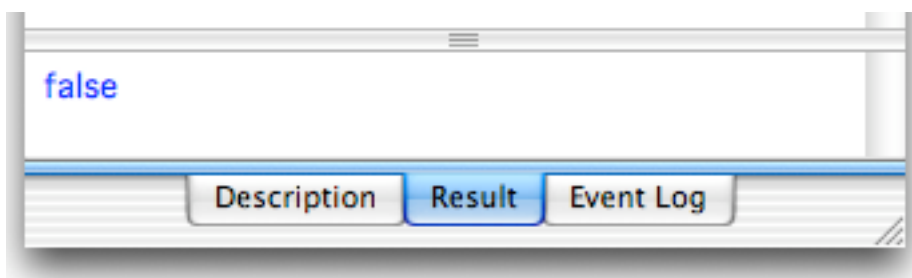
ГЛАВА 10

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ

Иногда, Вам захочется выполнить сценарий, серию действий, если есть определенное соответствие. Наберем в редакторе следующее [1] и выполним этот сценарий.

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

Результат будет выглядеть вот так:



AppleScript оценивает возможное в сценарии [1] и определяет как **false**, т.е. определяет как неправильное. Если Вы введете $30 = 30$, соответственно результатом будет **true**.

Есть возможность у AppleScript сравнить два значения при использовании **if...then** выражения [2] выполнить выражение только если условие соответствует. Выражение **if...then** вызвано условным заявлением. Обратите внимание, что выражение заканчивается связкой **end if**. Дополнительно мы к этому вернемся позже.

```
[2]
if true then
    -- здесь выполняем действия
end if

if false then
    -- если эти действия не выполнены
end if
```

Уловно, нужно заменить слово **true** в выражении [2.1] для сравнения. Если при сравнении выражение **true**, то выражение в строке [2.2] выполнены.

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

Сравните выражение `ageEntered is myAge` [3.3] будет ли оно справедливым, это показало бы диалоговое окно. С вышеуказанными выражениями переменных в первых двух выражениях, Вы не увидите диалогового окна [3.4].

Если будет больше инструкций, то должно быть выполнено условие соответствия, что они все должны быть в пределах выражения `if...then...end if` в блоке выражения [4].

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

Вы узнаете сходство в блоке выражения `if...then`. В обоих выражениях есть конец слова `end` в выражении. `end if` позволяет AppleScript определить истинное выражение, если сравнение выражения приведет к результату сравнения – `true`. В сценарии [4], Вы услышите последнее заявление [4.7] соответствует ли условию [4.3] или нет.

Мы можем выполнить альтернативное действие, чтобы определить истинность выражения, с использованием выражения `if...then...else` [5]

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

Здесь, в диалоговом окне [5.6] мы увидим результат, если результат выражения [5.3] приведет к `false`. Будет больше возможностей, которые можно выполнить. Здесь операторы сравнения для различных типов данных.

Отдельно от знака равенство [5.3], следуя оперяторам сравнения для цифровых данных в нашем выражении.

Для цифровых данных

=	равно (или будет равно)
>	больше чем
<	меньше чем
>=	больше или равно
<=	меньше или равно

Вторая колонка просто показывает объяснения для первой колонки, но Вы можете использовать это, чтобы уточнить выражение! [6] (прим. Пер.: вряд ли русскоязычные пользователи воспользуются словарным выражением, типа "больше чем - **is greater than**", наверно для нас удобнее использовать математические символы - ">")

[6]

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

Если есть выражение >= и проверить его синтаксис, AppleScript автоматически конвертирует выражение в официально выбранный символ ≥. Соответственно, выражение 'меньше или равно' определяемое оператором <= будет заменено на официальный символ ≤. Вы должны напечатать символы правильно, иначе это приведет к ошибке AppleScript. Вы всегда можете использовать более дружелюбный подход.

Отрицательные формулировки, например: **is not greater than** (прим. Пер.: не равно, в данном случае). Если написать /=, то автоматически будет заменено на символ ≠, который для краткости читается **is not** (прим. Пер.: не равно)

В главе 7 мы использовали следующий сценарий [7], который помог нам разобраться, какая кнопка была нажата.

[7]

```
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
```

Использование выражения **if...then**, позволяет сделать нам действие [8].

[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  
    say "I agree entirely!"  
else  
    say "Didn't you see the movie 'Pretty Woman'?"  
end if  
beep
```

Если пользователь нажмет кнопку "Yes", тогда истинно выражение [8.5], которое нам об этом само скажет. В любом случае, сценарий продолжится и мы услышим сигнал **beep**.

В предыдущем сценарии, значение в строках одинаково. Однако, AppleScript в состоянии сделать большее, чем это. Например сценарий [9]. Здесь использование концовки **end if** не требуется заканчивать выражение **if...then**, так как они находятся на одной линии.

[9]

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

Теперь рассмотрим операторы для строковых.

Операторы для строковых

begins with (или starts with)	– начать с...
ends with	– закончить с...
is equal to	– равно
comes before	– будет до или если ранее
comes after	– будет позже или если позже этого
is in	– находится внутри
contains	– содержит

(прим. Пер.: здесь после тире я дал расшифровку, потому что эти выражения итак понятны англоязычным, а нам просто стоит это запомнить наизусть и понимать, что будет происходить, если это использовать со строковыми выражениями... далее то же самое, после тире расшифровка выражения)

Для отрицательных действий:

does not start with	– не начинается с...
does not contain	– не содержит ...

is not in
etc.

– находится вне ...
– и т.д. (прим. Пер.: это не перевод,
для англоязычных и так понятно, что дальше)

Если написать **doesn't**, т.е. сокращенная форма **doesn it**, то автоматически редактор исправит на выражение **does not**. А если написать **does not begin with**, будет заменено на выражение **does not start with**. (прим. Пер.: последнее выражение легче понять так – **если не начинается с...** автоматически будет переведено – **если не начнется с...**)

Выражения **comes before** и **comes after** работают учитывая азбучное правило сортировки. Также, после выражения будет сигнал **beep** [10].

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

#Если сравнить строки, то регистр будет учитываться. Это маленький сигнал для AppleScript. Конечно, можно это требование отключить [11].

```
[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
```

По умолчанию, пробелы (пробелы, переходя на другую строку, табуляция) все это учитывайте при действиях. Если Вы не используете **ignoring white space**, как в примере [12]. Обратите внимание, что Вы должны включить выражение **end ignoring** или **end considering** выражения.

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

Вы можете также заставить AppleScript игнорировать пунктуацию, и не критические ошибки. #

Для списковых выражений, значительно меньше операторов, чем для строковых, они другие, но по смыслу тоже самое. Так, что Вы не выучите больше операторов чем необходимо.

Для списков

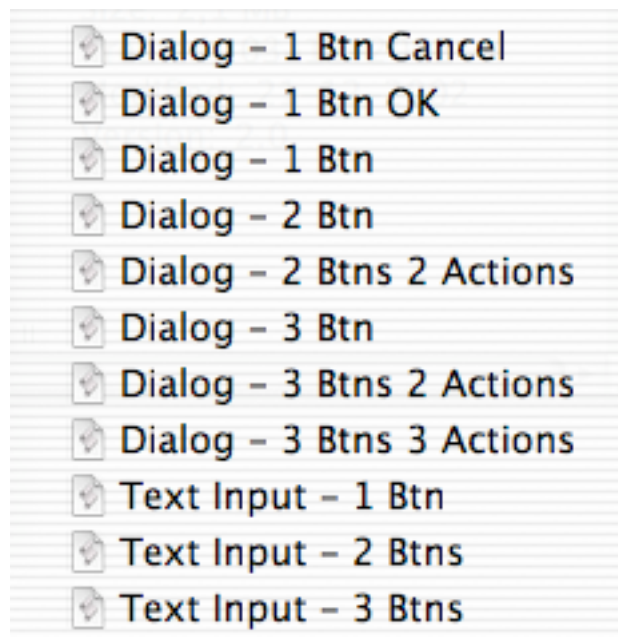
begins with	– начинается с ...
ends with	– кончается с ...
contains	– содержит ...
is equal to	– равно ...
is in	– находится внутри выражения ...

Часто, Вы сравниваете индивидуальные списки (или часть списков). Необходимо рассмотреть практический пример. Смотрите сценарий [8]. Где используется три диалоговых кнопки? Вложенность **if...then** выражения [13], для всех выражений верно.

[13]

```
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
```

#Как видите, вложенность позволяет прочитать, но по прежнему трудно. Плюс, можно забывать или не использовать end if выражение, если результат сценария компилируется. В редакторе Script Editor контекстное меню помогает! Вызовите меню Dialog и выберите Actions, может так?



Попробуйте вариант с тремя кнопками, 3 Actions и получите полное выражение. Справедливо выполнить вне списка [14.1] и заполнить самостоятельно дальше.

[14]

```
display dialog "" buttons { "", "", "" } default button 3
set the button_pressed to the button returned of the result
if the button_pressed is "" then
    -- реакция на 1 кнопку
else if the button_pressed is "" then
    -- здесь на вторую
else
    -- и на 3 кнопку здесь
end if
```

Первые две линии, возможно, нуждаются в объяснении. Поскольку Вы можете видеть из сценария [15.1] отличаются от выражений [8.2] мы используем set для определения переменных записи, результат виден из команды display dialog. Точно так же поле Result автоматически покажет результат выражения. Если выражение if на той же линии с else [14.5], то выражение end if не требуется. Теперь, сценарий легко можно прочитать. #

Для записей еще меньше операторов, чем для цифр, строк и списков.

Для записей

Contains	– содержит ...
is equal to -- or =	– равно ...

[15]

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

Вы заметили, что внутри выражения [15.1] одна метка показана синим цветом, а другая зеленым? Это Вам предупреждение, что одна из меток будет ключевым словом. Пока AppleScript не останавливает вас от использования меток идентичных ключевым словам, таких как to, set, string и т.д., это может Вас беспокоить иногда получением сообщений об ошибке, тогда избегайте использования ключевых слов в именах переменных, смотрите главу 4.#

Ограниченным числом операторов сравнения для записей не будет для Вас ограничением потому, что Вы не будете сравнивать полные записи, а индивидуальные свойства записей [16].

[16]

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

По мере того, как мы видели в первой части этой главы, две части, если сравнивать два значения (их тип данных), Вы пытаетесь увидеть выражение как true или false. Результат, называется логическим типом – Boolean. Переменные могут иметь только два значения: true или false (прим. Пер.: в подлиннике написано – true и false, правильное все же будет использовать частицу "или", потому, что может быть только одно значение результата действий). Для цифровых данных, Вы используете операторы + и -. Результатом такого действия будет цифровое значение. Для операторов Boolean, мы используем and, or и not. Результат этих операций будет все равно логическим – Boolean.

not простейшее в структуре. not true это не верно – false, и not false это истинно – true.

[17]

```
set x to not true -- So, x is false
```

Для демонстрации рассмотрим сценарий [18], для оператора and, для всех x and y результатом будет z, что в данном случае равно true.

[18]

```
set x to true  
set y to true  
set z to (x and y) -- z как видите - равно true
```

И наоборот, для оператора **or**, будет достаточно одно из **x** и **y** равным **true** [19].

[19]

```
set x to true
set y to false
set z to (x or y) -- z is true
```

Зачем я все это говорю? Ну, в некоторых обстоятельствах Вы хотите комплект инструкций которые выполняются только если несколько условий соответствует. Следующий сценарий [20] вызывает **display dialog** только для последнего выражения.

[20]

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

В выражении [20.4] первое выражение **true** (прим. Пер.: что то **y** не очень то и равен 6, скорее 7 и чтобы проверить это, написал сценарий, я теперь тоже немного умею писать сценарии:

```
set x to 5
set y to 7
if x = 5 and y = 6 then
    display dialog "true"
else
    display dialog "false"
end if
```

и результат конечно **false**), а во втором **false**. **and** требует, чтобы было **true**, но диалоговое окно не появляется (прим. Пер.: то о чем мы и говорили с Вами). В выражении [20.5], **or** принесло бы точное соответствие (прим. Пер.: ну что еще раз проверим? Давайте:

```
set x to 5
set y to 7
if x = 5 or y = 6 then
    display dialog "true"
else
    display dialog "false"
end if
```

вот теперь результат **true**).

В выражениях сценариев [20.4, 20.5] Вы захотите использовать скобки, так как это увеличит читаемость выражения (и по возможности надежность).

ГЛАВА 11

КАК ОБОЙТИСЬ БЕЗ ОШИБОК

Во всех обсуждаемых до сих пор сценариях мы обсуждали до сих пор, если AppleScript столкнется с ошибкой во время выполнения сценария, то он должен остановиться.

```
[1]
beep
set x to 1 / 0
say "You will never hear this!"
```

Если Вы проверите синтаксис данного сценария [1], AppleScript не укажет на проблему. Тогда как, при выполнении сценария, Вы не получите сообщения [1.3] потому, что сценарий не дойдет до выполнения [1.3], а остановится на части [1.2].

Конечно, остановка сценария не очень приятна для Вас. Например, если Ваш сценарий выполняет работу с папками или файлами, и папка уничтожается, Вы можете выбрать еще вариант - чтобы пользователь сам выбрал папку.

Пока записываете сценарий, Вы должны соотнести уровень ошибок для выполняемого сценария. Заключите эти заявления в блок **try ... end** как продемонстрировано здесь [2].

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

Теперь, если выполнить сценарий, вы услышите предложение [2.6], AppleScript продолжит выполнение после заявления **end try** [2.4].

В главе 7, мы рассматривали записи, и столкнулись с таким сценарием [3]

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

Проблема этого сценария в том, что пользователь мог ввести отличное от цифровых данных значение, тогда сценарий не мог быть выполнен. Мы можем

избежать остановки сценария, и обеспечить пользователя обратной связью со сценарием [4].

[4]

```
set temp to display dialog "What is your age in years?" default answer ""
set ageEntered to text returned of temp
try
    -- Сначала проверим, ввел ли пользователь число
    set ageEntered to ageEntered as number
    set ageInMonths to ageEntered * 12
    display dialog "Your age in months is " & ageInMonths
on error
    -- Если не число, тогда принять как текст и сообщить об ошибке
    display dialog "Instead of a number, like 30 , you entered text."
end try
```

Теперь, если пользователь введет не число, он увидит сообщение об этом. Единственное неудобство, что пользователь должен выполнить попытку еще раз. В главе 13 мы решим эту проблему.

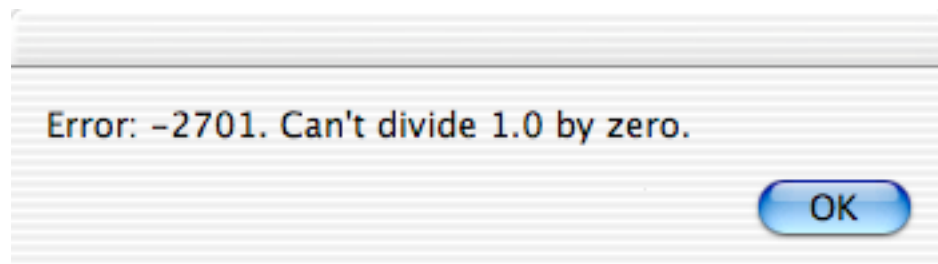
#Редактор Script Editor создает очень простые блоки try. Выберите одно или несколько строк кода чтобы включить в блок try, и через контекстное меню выберите try block чтобы заключить в блок попыток [17].

Если сценарий возвращает ошибку (on error), перед блоком try, AppleScript будет выполнять повторы операций: возвращая число ошибки, и будет выполнять сценарий пока не решится проблема.

[17]

```
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
```

Если Ваша переменная находится после `on error`, объяснение проблемы заключено внутри переменной. Если имя Вашей переменной содержит число, то номер ошибки будет положен внутрь этой переменной. В части [17.3] Вы все это увидите. Здесь диалог выглядит так.

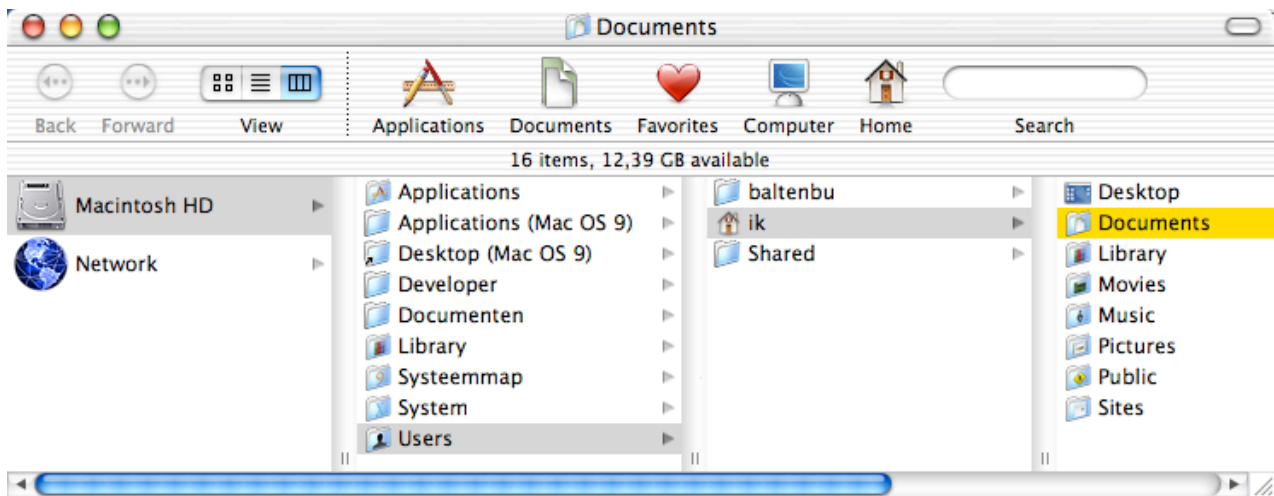


#

ГЛАВА 12

ПУТИ К ФАЙЛАМ, ПАПКАМ И ПРОГРАММАМ

Теперь стоит обратить внимание на то, как папки и файлы организованы на Вашем жестком диске. Если Вы откроете окно Finder в виде column view, что представлено здесь.

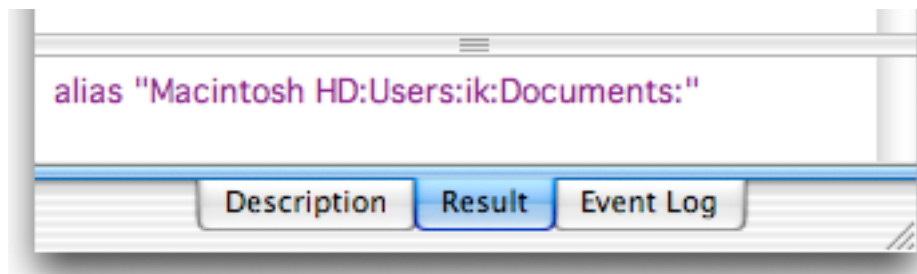


Этот хард диск, содержит папки, программы и файлы (на иллюстрации отсутствуют файлы и программы). Все элементы организованы иерархической структурой. Это удобно для определения положения файлов на жестком диске (или папок, или программ). Внимательно рассмотрите путь как в примере[1].

[1]

[choose folder](#)

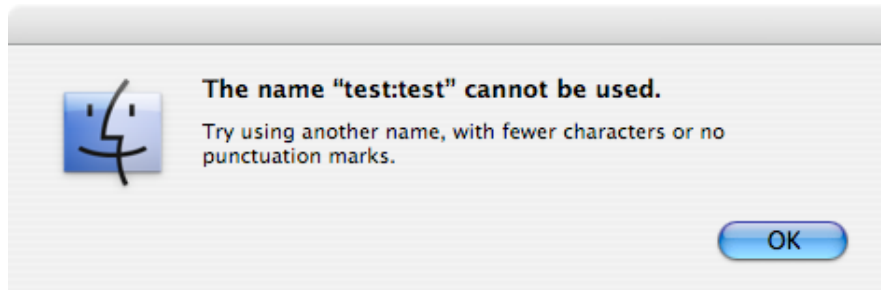
Можно посмотреть результат в окне Result, если выбрать например мою папку Documents.



Базовое представление пути к папке будет выглядеть так:

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

Теперь примените это правило к верхнему изображению (снимок экрана окна Finder) к папке Documents. После этого обратите внимание на поле Result. Обратите внимание, что двоеточия используются как разделители. Вот почему нельзя использовать двоеточия в именах папок и файлов. Попробуйте создать на Вашем desktop папку содержащую двоеточие. (прим. Пер.: ну давайте проверим:



как видите ничего у меня не получилось, я попытался создать имя для папки test:test и это привело к ошибке). Теперь, Вы можете увидеть, что путь кончается двоеточием, которое показывает, что путь ссылается к папке.

Будем использовать путь через **tell** к программе Finder чтобы открыть папку [2].

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

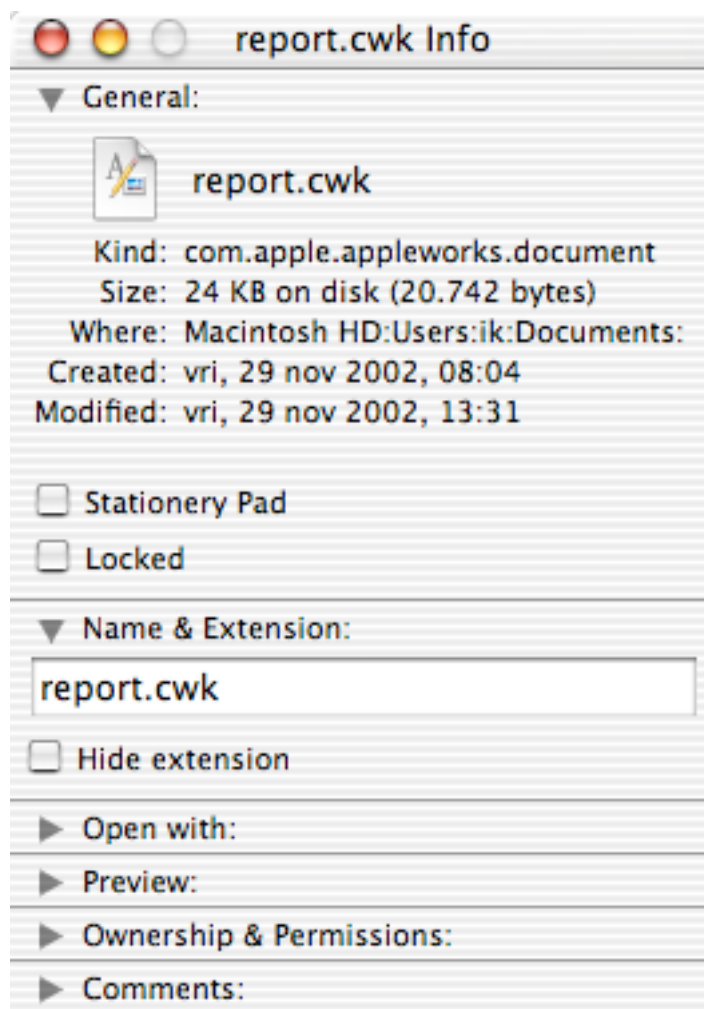
Посмотрите как выполняется сценарий [2] (замените **ik** на Ваше имя в системе), AppleScript разрешает не указывать двоеточие в конце пути, если Вы вдруг забудете его поставить. Однако, обратите внимание, что имя папки на жеством диске написано строго с учетом регистра. Чур, чур, Apple!

Моя папка Documents содержит документ AppleWorks под именем **report**. Надо открыть этот файл [3].

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

Первое примечание по сценарию [3.2] содержит **file**, а не **folder**, потому, что обращаемся к файлу. Вторая вещь, на которую стоит обратить внимание, это расширение файла, которое является неотъемлемой частью имени файла. Здесь расширение **cwk**, обозначающее принадлежность к программе AppleWorks, которое раньше было ClarisWorks.

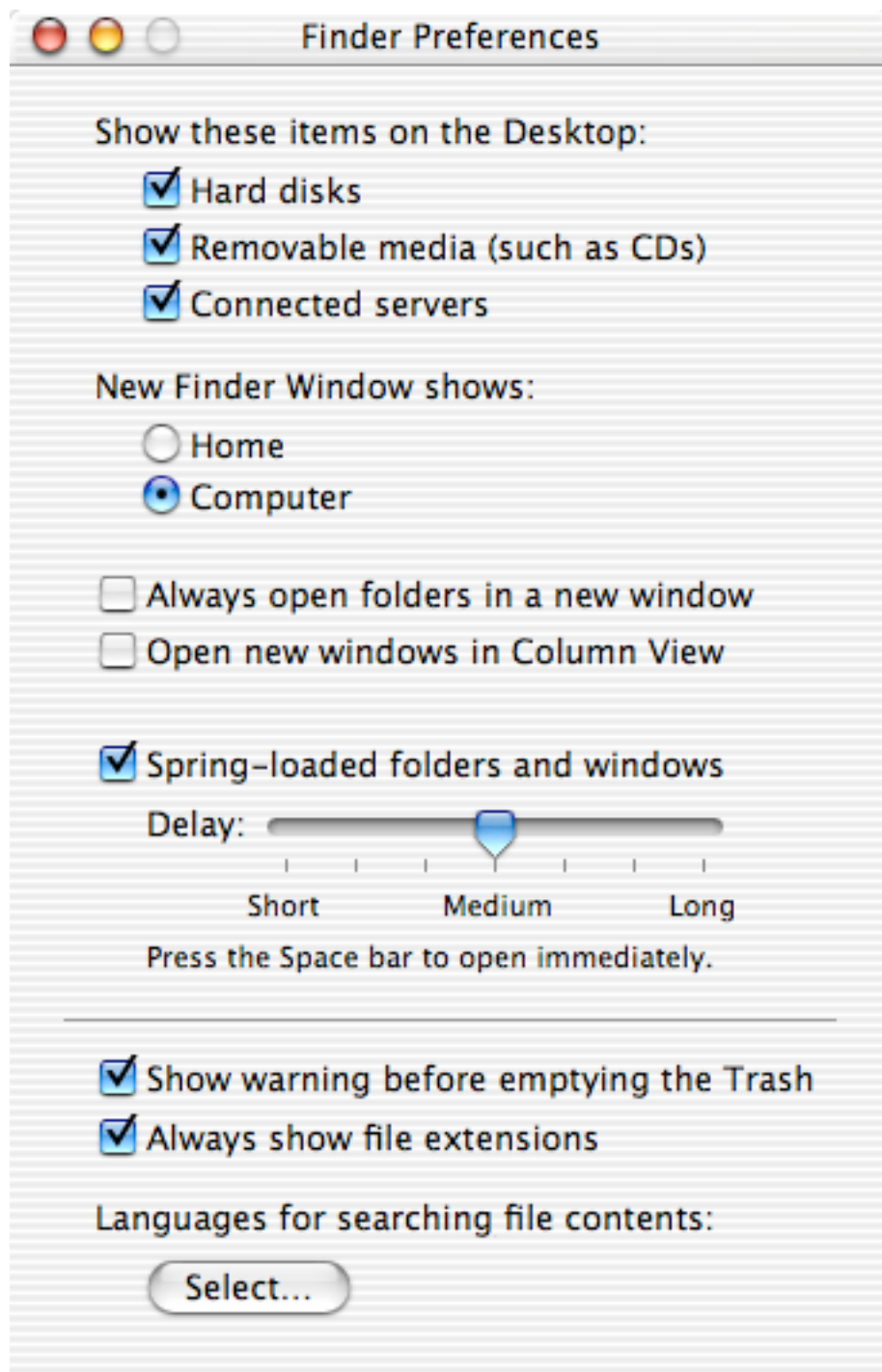
#В Mac OS X расширение файла или папки скрыто по умолчанию. Вы это можете увидеть если воспользуетесь командой Command-I (См. на следующей странице).



Вместо этого Вы можете сделать настройку в системе, чтобы она показывала имя файлов всегда. Вы можете это сделать обратившись Finder preferences меню. Щелкните через Finder меню. Выберите Preferences.



В появившемся окне, ищите внизу Always show file extensions.



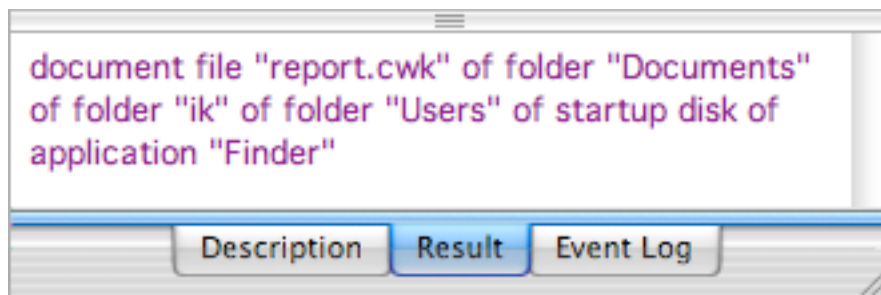
#

Если Вы хотите сохранить путь к файлу `report.cwk`, используйте переменную, вложив в нее путь к файлу, как здесь [4]

[4]

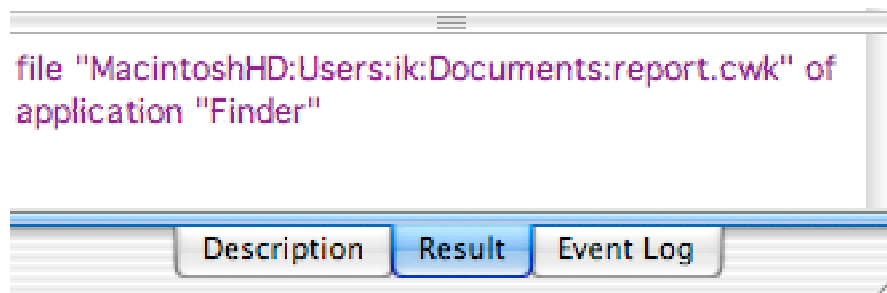
```
tell application "Finder"
    set thePath to file "Macintosh HD:Users:ik:Documents:report.cwk"
end tell
```

Если выполнить данный сценарий, то в окне Result можно увидеть, например вот это:



Это сложно прочитать, особенно для более длинных путей. Вы можете это сократить если использовать команду **a reference to** [5].

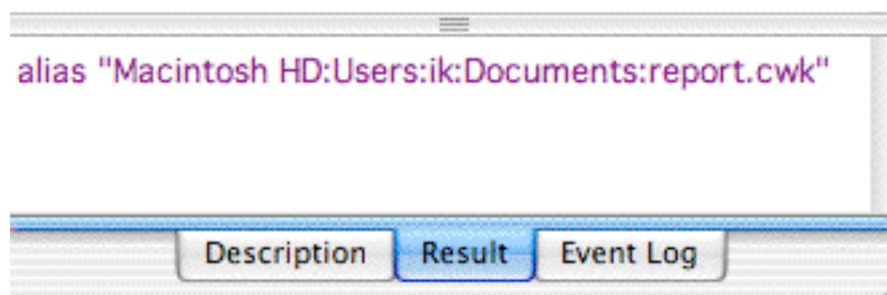
```
[5]
tell application "Finder"
    set thePath to a reference to file "Macintosh HD:Users:
                                   ik:Documents:report.cwk"
end tell
```



Обратите внимание на результат, где находится наш файл. Выполните сценарий [6] выбрав тот же файл report.cwk.

```
[6]
choose file
```

Теперь результат будет выглядеть вот так:



Вы видите слово **alias**, вместо **file**! Это совсем другой подход AppleScript, поспорьте

пока со мной, пока не разберетесь как работает Finder.

Предположите, что я создаю алиас на моем desktop файла [report.cwk](#) который находится в папке [Documents](#). Теперь, если мы перенесем файл [report.cwk](#) из папки [Documents](#) в другое место, двойным щелчком откроем этот файл. Замечательно! Я переименую оригинальное имя [report.cwk](#) например на [funny_story.cwk](#). Если это алиас, то он не хранит путь (и имя) в файле [report.cwk](#) как:

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

но хранит как [ID](#). Finder использует базу данных [ID](#) вместе с текущим местоположением файлов (и папок, и программ для этого дела). Так, если я перенесу файл [report.cwk](#), его уникальное [ID](#) остается тем же, потому, что Finder изменяет внутри базы данных всю основную информацию о новом местоположении. Если я дважды щелкну на алиасе, [ID](#) содержит путь и Finder это будет использовать в поисках местоположения файла, и Finder откроет правильный файл.

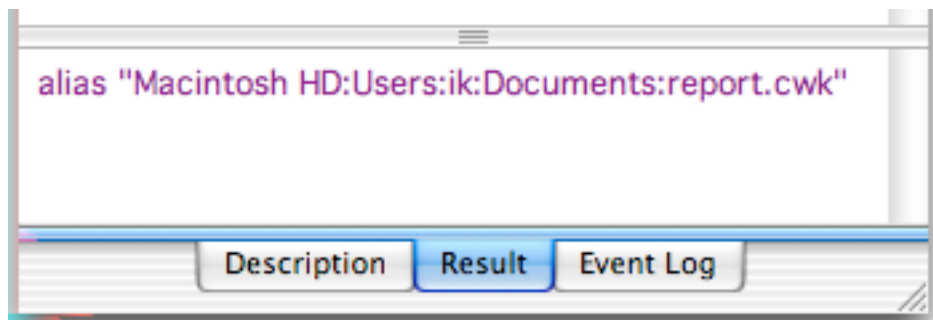
Сценарий не меняется если изменено имя файла (или папки), скрипт будет искать уникальный [ID](#) файла (или папки). AppleScript позволяет для этого сделать [7].

[7]

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

Очень важное замечание по сценарию [7] которое ссылается на первоначальный путь в папке Documents, а не на то что создал пользователь, когда переименовал файл, как я сделал, перенеся файл на мой desktop. В сценарии [7], алиас используется как ключевое слово, после компиляции (т.е. проверка синтаксиса), сценарий запоминает уникальный [ID](#) файла и, во время исполнения, спрашивая у Finder первоначальное местоположение файла, путь который базируется на [ID](#).

Если Вы выполните сценарий [7], и проверите результат, то увидите следующее:



Так, Вы ничего не получите, видите только собственный [ID](#). Однако, слово [alias](#) говорит Вам, что сценарий берет путь из [ID](#), и не работает с "жестким" путем. Теперь, когда сценарий выполнен, перенесите файл [report.cwk](#) в другое место на

диске и выполните сценарий еще раз. Я перенес файл в папку [Miscellaneous](#) внутри папки [Documents](#). Даже если сценарий, и путь в сценарии [7], не изменяли, результат такой:

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

Попытайтесь самостоятельно (снова, Ваш путь будет указан по другому, в пути будет использован Ваш [login](#) а не мой [ik](#) и возможно потому, что Вы можете по другому называть Ваши папки)! Если Вы сохраните сценарий, а потом сохраните как compiled script или как AppleScript application, ID сохранится, и следующий раз, когда Вы выполните сценарий, то оно все равно будет обращаться к оригинальному файлу. Может это Вас удивит, но если файл стереть, то вот тогда сценарий не будет выполняться.

Подведем итог, у Вас есть два варианта, чтобы общаться с путями в сценарии. Вы можете указывать специфическое местонахождение файла (жестко прописать, используя строку типа "путь здесь"), или используйте [alias](#) как ключевое слово для выполнения сценария, это поможет Вам застраховаться от перемещения и переименования файла/папки/программы после компиляции сценария.

Пока используется [alias](#) как вариант выполнения сценария, то файл должен иметь неизменяемый путь до конца компиляции сценария. А также, Вы не можете использовать сценарий пока не создадите, например программу AppleScript, потому что файл с таким именем существует в определенном месте на Вашем компьютере, у файла будет разное ID.

#Пора сделать примечание по сценарию [7]. Лично, я был приятно удивлен как это работает без Finder "tell block". Потому что внутренняя база данных хранит ID и местоположение файла, и как это поддерживается Finder. Finder единственная программа которая предоставит нам эту информацию, и часть системы AppleScript знает это, AppleScript явно просит Finder дать ID напрямую из базы данных. Это исключительно, однако. Например, открытый и используемый путь к файлу требует наличия tell block (См. сценарий [3]). Это не справедливо, потому что Finder мог открыть нам этот путь, и по возможности другая программа тоже. Опять же для примера, изображение в jpeg формате, может быть открыто программой PhotoShop или Вашим браузером. tell block необходим для того, чтобы показать какая программа будет открывать файл. #

Теперь когда мы знаем адрес файлов и папок, мы можем перенести их в другое место или скопировать [8].

[8]

```
tell application "Finder"
```

```
    move file "Macintosh HD:Users:ik:Documents:report.cwk" to the trash
```

```
end tell
```

ГЛАВА 13

ЦИКЛИЧЕСКИЕ ПОВТОРЫ (ЦИКЛЫ)

Во всех сценариях, которые мы рассматривали ранее, каждый пример выполнялся один раз. Иногда, требуется выполнить блок задач по несколько раз. AppleScript предлагает несколько вариантов, чтобы это выполнить.

Если Вы знаете точно число повторов (или группы повторов) которые должны быть выполнены, Вы это можете определить путем включения количества повторов сценария или его части [1]. Количество повторов может быть только целым числом, потому что Вы не можете повторить, например сказанное слово - только 2.7 раза.

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

Это видно из выражения **tell**, **try** и **if...then**, а также выражения **end repeat** для принудительного выполнения AppleScript которому принадлежит выполнить группу повторов.

Вместо того, чтобы определять количество повторов, Вы можете использовать переменную [2].

```
[2]
set repetitions to 2
repeat repetitions times
    -- Statements to repeat here
end repeat
```

Здесь [3] более дружелюбный сценарий, чем [2]. Пока сценарий [3] выполняется, пользователь может вписать в диалоговом окне количество повторов. Мы должны вписать целое число, или преобразовать значение в цифровое. Это нужно если пользователь вписал текстовое, например, значение. Для этого, мы должны принять меры к предосторожности.

```
[3]
-- Пользователь может ввести текстовую строку
set textToDisplay to "How often has the sentence to be repeated?"
-- Цифра '2' это намек для пользователя, какой тип данных ввести
display dialog textToDisplay default answer "2"
set valueEntered to text returned of the result
```

```

try
    -- valueEntered это строка (а не целое число), пока видим: "2"
    -- Ждем целого числа. Если нет, прекратить
    set valueEntered to valueEntered as integer
end try
-- Если valueEntered не целое число, повторим блок
if class of valueEntered is integer then
    -- Повторный блок
    repeat valueEntered times
        say "Julia is a beautiful actress"
    end repeat
else
    display dialog "You did not enter a (valid) number."
end if

```

После заявления else [3.21], пользователь получит уведомление об ошибке и должен повторить ввод данных заново. Два других повтора [4, 5] будет выполняться пока условие не станет правильным.

```

[4]
set conditionMet to false
repeat while conditionMet is false
    -- если выражение верно
    -- установить conditionMet в true
end repeat

```

```

[5]
set conditionMet to false
repeat until conditionMet is true
    -- если выражение верно
    -- установить conditionMet в true
end repeat

```

Запретите повторение сценария [4] для того чтобы пересечь неудобство сценария [3]. Мы повторим запрос до тех пор пока значение не будет целым числом. В случае удачи, мы установим переменную correctEntry в значение true, в результате чего повторы сценария прекратятся, и будет продолжено выполнение остатка сценария [6]. Если значение введенное пользователем можно преобразовать в целое число, то мы сообщим об этом пользователю.

```

[6]
set correctEntry to false
repeat while correctEntry is false
    -- Покажем сообщение
    set textToDisplay to "How often has the sentence to be repeated?"
    -- Цифра '2' это намек на тип данных
    display dialog textToDisplay default answer "2"

```

```

set valueEntered to text returned of the result
try
    -- Переменная valueEntered всегда строковая.
    -- Здесь заставим ввести целое число
    set valueEntered to valueEntered as integer
    -- Установим correctEntry в true, чтобы закончить повтор
    set correctEntry to true
on error
    -- Покажем обратную связь
    try
        -- Ждем пользователя, его ввод
        set valueEntered to valueEntered as number
        display dialog "You entered a fractional number instead
                        of an integer."
    on error
        -- Если это не число, выведем строку с объяснением
        display dialog "Instead of an integer, like 9 , you
                        entered text."
    end try
    -- Пока значение correctEntry является false, продолжаем
end try
end repeat

-- Сценарий может сделать все правильно, если correctEntry равно true
CorrectEntry только true, если valueEntered целое число
repeat valueEntered times
    say "Julia is a beautiful actress"
end repeat

```

Обратите внимание на положение `set correctEntry to true`, это очень важно. Оно должно быть:

- внутри (первой) попытки (try block); и
- после заявления может привести к ошибке (здесь принудительно)

Иначе, переменная `correctEntry` нужно установить в значение `true` (для успешной концовки сценария).

#Пока сценарий [3] может выполнить заданное действие (говоря по другому - количество раз) при выполнении сценария [6], Т.е. он может вывести ошибку, если пользователь введет неправильные данные, и нельзя обеспечить правильную, обратную связь с пользователем, если произойдет ошибка. Однако, сценарий [6] можно расширить путем введения ограничений в значение `valueEntered` (например использованием фрагмента сценария [7]), для того, чтобы ограничить количество повторов до 10,000 раз. С другой стороны, сценарий [6] может быть достаточным для Ваших целей.

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

Если Вы сделаете нормально работающий сценарий, Вы должны его хорошо протестировать. То с помощью ввода текста, то с помощью цифр с плавающей точкой, непонятными данными и прочим. Тогда сценарий будет по настоящему рабочим. Единственное, что предусматривает сценарий [6] только если пользователь введет отрицательное значение. Вы можете перевести его или в положительное значение, или сообщить об этом пользователю. Интересно, в сценарии [6] что не произойдет ошибки, если введено отрицательное значение. Проверьте это в сценарии [1].

#

Если повторить сценарий [4] и [5] для любой цели. Вы можете повторить цикл для любых целей, если:

- пользователь введет файл или папку,
- слово присутствует в файле
- и т.д.

И наоборот, для общего значения повторить сценарий [4] и [5], сценарии [1] и [2] для цифровых значений. Будет больше пар для такого выполнения.

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

По мере того, как Вы видите, можно использовать переменную сценария [8.1], т.е. **counter**, внутри сценария. Однако, Вы можете не менять переменную для других значений в блоке повтора.

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

Если выполнить сценарий, ничто не будет сказано и все бутылки с 1 по 5 будут уничтожены.

В части [9.1], используется шаг 1 по умолчанию. Для другого значения используйте [10.1].

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

В сценарии [10], шаг размером 2, и Вы услышите предложение три раза (для значений переменных 1, 3 и 5).

Если у Вас список, и каждая деталь должна быть использована мимо темы или быть подвергнуто некоторому действию, Вы можете подсчитать цифры в списке, и выполнить циклы в сценарии [11].

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

Однако, в AppleScript есть элегантная альтернатива, демонстрация которой ниже. Сценарий [12] позволяет Вам обусловить количество папок в папке выбранных пользователем. После повтора можно создать список имен во всех папках.

```
[12]
set folderSelected to choose folder "Select a folder"
-- Ищем папки со всеми вложениями, мы узнаем от Finder.
-- Внимание: "every folder" НЕ содержит папок внутри других папок.
tell application "Finder"
    set listOfFolders to every folder of folderSelected
end tell
-- Результат список (путей)
-- Вне комментария все заявления используют:
    список listOfFolders содержит – папку reports внутри Documents
    внутри "ik" папки "Users"
    на начальном диске Finder.
-- Только Finder и AppleScript компонент Mac OS X могут общаться
    с такими заявлениями.

-- Имя папки будет сохранено в новом списке, созданных здесь
set theList to {}
repeat with aFolder in listOfFolders
    -- У нас есть доступ к папкам, и поэтому доступ содержит имя папки,
        AppleScript компонент Mac OS X может получить имя и без Finder
    -- Если Вы хотите найти другое свойство папок, т.е. размер (в байтах),
```

то обращение к Finder просто необходимо (используйте tell block)

```
set temp to the name of aFolder
```

```
-- Здесь мы добавим имя в список
```

```
set theList to theList & temp
```

```
end repeat
```


ГЛАВА 14

ПОДВЕДЕМ ИТОГИ (ОБРАБОТЧИК)

AppleScript по природе ангоподобный язык, делает сценарий легко создаваемым и читаемым. Мы это все с Вами посмотрели ранее. Например, как выбрать имена переменных и как создать комментарии к сценарию. AppleScript может помочь Вам содержать сценарии в порядке благодаря **обработчику событий**. Представьте, что Вы имеете такой же комплект заявлений или еще больше в Вашем сценарии. Если будет вызвана ошибка, то Вы должны исправить ее в каждом из таких мест в сценарии. AppleScript предлагает собрать такие заявления и дать этой группе имя. Если Вы вызовете имя, то будет вызвано точное выполнение.

Здесь, как включить обработчик [1].

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

При использовании, Ваш сценарий вызовет обработчик, как здесь [2].

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

Оно не имеет значения, определен ли обработчик в Вашем сценарии до или после выражения.

Обработчик в сценарии [1] будет непреклонен. Было бы славно, если бы могли сказать обработчику - покажи текст на экране. Угадайте, это обработчик был сделан для [3].

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

В части [3.1], переменная **textToDisplay** определяет значение когда обработчик получает (в части [3.4] и [3.5], каждое значение содержит значение между скобками, которое передает дальше обработчику). Когда сценарий [3] выполняется, два окна диалога показываются последовательно.

Вместо определения данных, вызывается обработчик, Вы можете использовать переменную вместо [4].

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

Обратите внимание, что имя переменной используется для вызова обработчика [4.5] отличает от другого вызова обработчика [4.1]. Так, Вы знаете (смотрите выше), что используется имя переменной для обработчика. Конечно, Вы знаете, что тип данных обработчик определит.

Вы не только можете передать информацию дальше обработчику, но они могут также возвращать информацию.

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

Обработчик `circleArea()` вычисляет значение $\pi * 3^2$ и автоматически возвращает результат. Однако, использование команды `get`, возвращает автоматически для последнего значения выполняемого только обработчиком. Главное обеспечить то, что Вы получаете как результат, который Вы хотите получить если создать в серии части [6.3], используйте ключевое слово `return` [6].

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

В сравниваемой части [6.2] равна `true`, часть [6.3], пока не последняя часть в обработчике, возвращает результат.

Вы не ограничены единственным значением переменной [7.7] к обработчику.

```
[7]
on largest(a, b)
    if a > b then
        return a
    end if
```

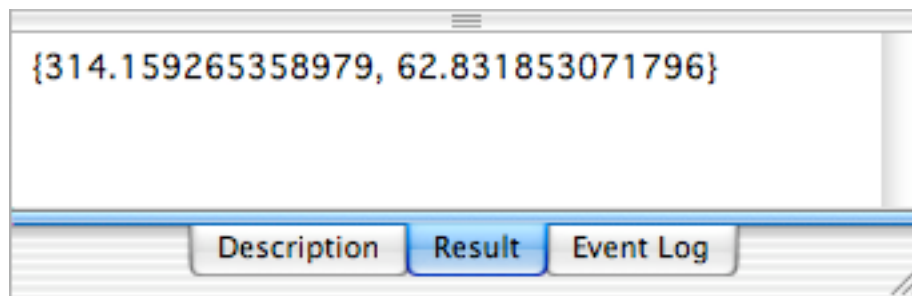
```
        return b
    end largest
    set theLargest to largest(5, 3)
```

Этот сценарий возвращает большее из двух значений. Обратите внимание, что в выражении [7.1] две переменные определяются на большее значение. Соответственно, вызывая обработчик, две переменные необходимо поставить [7.7]. В настоящем сценарии, по крайней мере эти значения нужно вписывать как переменные.

Также по возможности получить больше чем одно значение. В этот конец, нужно вернуть данные как список [8.4].

```
[8]
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)
```

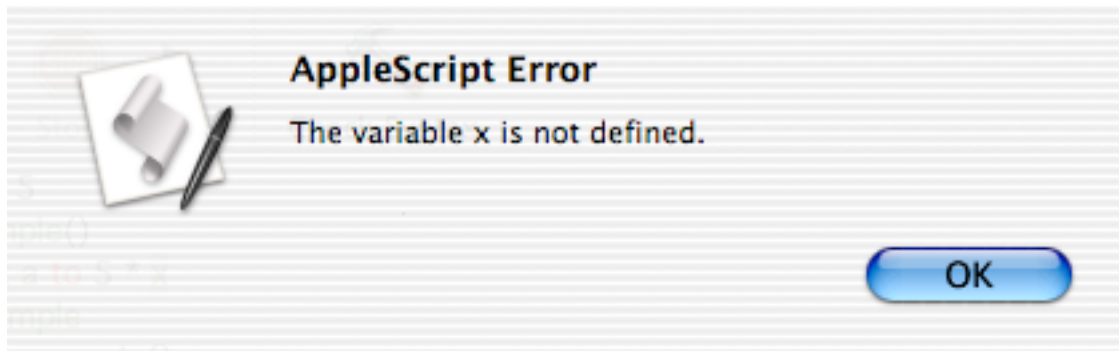
Вышеуказанный сценарий возвращает список значений для окружности круга с определенным радиусом.



Написать Ваш сценарий как серию обработчиков и пары значений делают выше тела сценария, можно конечно сохранить Ваше беспокойство и время благодаря важному свойству обработчика. Здесь вопрос для Вас (и ответ для себя): Если Ваш сценарий использует имя переменной с уже присутствующим для разных целей в обработчике? Не беспокоит, это не взаимодействие (прим. Пер.: можно перевести как интерфейс). Для доказательства этого, выполним пункт сценария [9].

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

Это вызовет следующую ошибку:



Внутри обработчик, переменная **x** не определена. Если Вы хотите значение **x** [9.1] определить внутри обработчика, Вы можете пойти дальше, как продемонстрировано в сценарии [4] .

И наоборот, изменяя переменную внутри обработчика не имеет никакого значения на обработчик. Здесь доказательство [10].

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

В поле результатов видно **x** равное 5. Так, не будет взаимодействия между сценарием и обработчиком, которое внутри, за исключением точно переданной дальше обработчику и возвращенное обработчиком. Я должен сказать, как вариант, сделать переменную **x** как значение [9.1] или [10.1] функционально в обработчике без его прохождения дальше. Однако, это делает сценарий трудным для чтения и отладки. В дополнение, оно негативно отражается на больших возможностях обработчика, обсужденных в следующем параграфе.

#

Если Вы любознательны как создать переменную, то действительно - все внутренние и внешние обработчики, здесь будут показаны – как сделать это: Задайте переменной значение **global** (прим. Пер.: **global** означает, доступен везде, а не только в текущем сценарии, доступен везде в программе, и не важно где переменная была объявлена). Рекомендуется это сделать в первых линиях сценария, потому что потом будет легче найти - где объявлена эта переменная.

```
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, переменные выполнены как местного/локального значения.

Пока я не сказал: "Я оформил определенные переменные как глобальные, потому, что причины описанные здесь, являются одним типом глобальных переменных, которые всегда под рукой. Они приходят с специфическим именем property.

```
property x : 1

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

Выполните указанный сценарий несколько раз и обратите внимание на результат. Значение свойства сохранено между несколькими выполнениями сценария. Вы заметили из сценария, где property значимо как внутри так и снаружи обработчика, в других словах, она выступает как глобальная переменная.

#

Отдельно от вышесказанного, из важных преимуществ (читаемость и т.д.), дополнительно большое преимущество, что Вы можете повторно использовать обработчики в других сценариях, которые создадите. Потому, что обработчик был использован и протестирован для другого сценария, Вы можете быть уверены, что переменная будет работать в других сценариях. Это сокращает время на разработку нового сценария. Вы можете скопировать это заявление из другого сценария, для простоты. Вы можете обратиться к сценарию, в котором переменная прописана как глобальная. В дополнение, Вы берете риск на себя, что Вы копируете в незаконченном или не рабочей части сценария в Вашем новом сценарии. Поверьте мне, обработчики поймут.

#

Замечательно, потому, что Вы можете теперь насладиться эффективно работающим другим сценарием. Но если Вы сталкиваетесь с ошибкой в обработчике или думаете о расширении дальше возможностей? Вы измените все Ваши сценарии. AppleScript имеет возможность решить эти проблемы, используйте load script команду. Все что Вы должны сделать:

- 1) Сохранить один или несколько обработчиков в выполняемом сценарии.
- 2) Включить в сценарий обработчик заявлений.

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

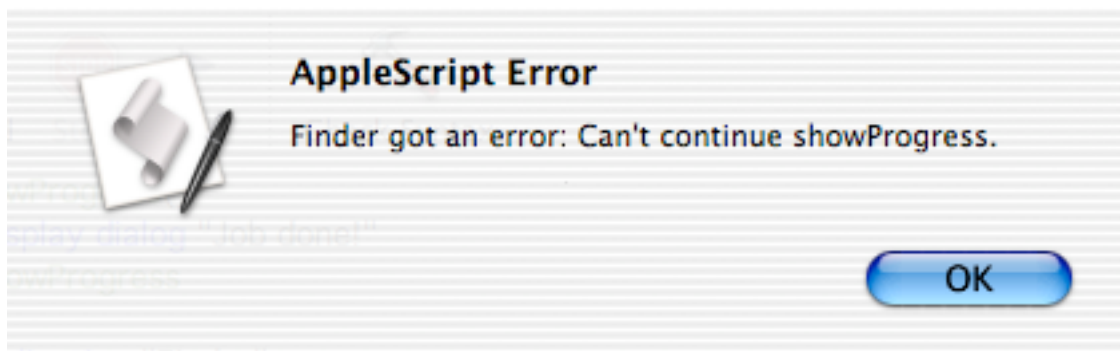
To use a handler of the compiled script, you must use a tell block.

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

Один специфический аспект обработчиков, что если Вы используете tell block, без дополнительного измерения, обработчик скажет что нужно внутри блока определять. Если выполнить сценарий [11]

```
[11]  
on showProgress()  
    display dialog "Job done!"  
end showProgress  
  
tell application "Finder"  
    empty the trash  
    showProgress()  
end tell
```

то будет показана такая ошибка:



Лечение просто. Покажите обработчику в самом сценарии [12.7].

```
[12]  
on showProgress()  
    display dialog "Job done!"  
end showProgress  
  
tell application "Finder"  
    empty the trash  
    showProgress() of me  
end tell
```

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

```
[13]
set x to 4
tell application "Finder"
    set x to 5
end tell
get x
```

Результатом будет 5.

ГЛАВА 15

ИСТОЧНИКИ ДОПОЛНИТЕЛЬНОЙ ИНФОРМАЦИИ

Эта книга дает только базовые знания по языку AppleScript. Если Вы пропустили через себя эту книгу дважды, и готовы создавать свои собственные сценарии и программы, то Вам понадобится дополнительная информация.

Важное примечание для тех, кто хочет создавать свои программы и свои сценарии: Сделайте это!

Но может такой сценарий уже сделан? Тогда сохраните Ваше время, тем более множество сценариев открыто для просмотра и редактирования. Где найти готовый скрипт? В интернете, конечно. Сделайте первый визит на сайт компании Apple

<http://www.apple.com/applescript>

здесь много различных ссылок. Я очень рекомендую сделать на них закладку

<http://www.AppleScriptSourcebook.com/>

и конечно

<http://macscripter.net/>

На сайте Macscripter есть форум, где можно получить ответы на свои вопросы. Опытные участники помогут Вам в любом вопросе создания сценариев. Да, мы создаем общину Макинтош на этом сайте. На сайте можно найти огромное количество ссылок на различные источники информации.

Я надеюсь Вы насладились этой книгой и языком сценариев AppleScript. Ох, и, пожалуйста, не пропускайте Главу 0.

Берт Алтенбург

ГЛАВА 16

ПРИМЕЧАНИЕ К РУССКОМУ ПЕРЕВОДУ

До сих пор я не встречал ничего на русском языке по AppleScript. Были краткие введения в различных книгах, например: Дэвид 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:

#Hi Vladimir, You're most welcome to translate the booklet into Russian. I'll have to dig through my files to send you the latest version, but I'll do it this weekend. I believe I wrote it in AppleWorks. Would that be ok?#

и дал разрешение перевести эту книгу на русский язык! Она самая первая!

Special Thanks, Bert Altenburg!

Yes, and I like the idea a lot. But don't underestimate the amount of time it takes to do this. You won't finish it in just a weekend. Take your time: if you do it steady, you'll make it. The German guy did it and (even though he lost part of the translation due to a mishap) finished it. A French guy didn't complete the translation.

As to the translation, of course you must put your name next to ours.

BTW, I'm currently working on a book on leaning Objective-C with Xcode. Perhaps you will want to translate that one too! :-)

Have a nice weekend. #

Да уж! Выходные будут на славу!

Thanks, a nice weekend!

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. Хочется порекомендовать Вам источник, которым сам пользовался при переводе книги, хотя честно сказать, ничего не понял. Наверно, это лишний раз доказало мне необходимость сделать перевод именно начальных знаний. Рекомендую для начала сходить на сайт:

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

там же есть форум, на котором можно задать вопросы и даже получить ответы:

<http://www.yezhe.ru/applescript/ASForum/>

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