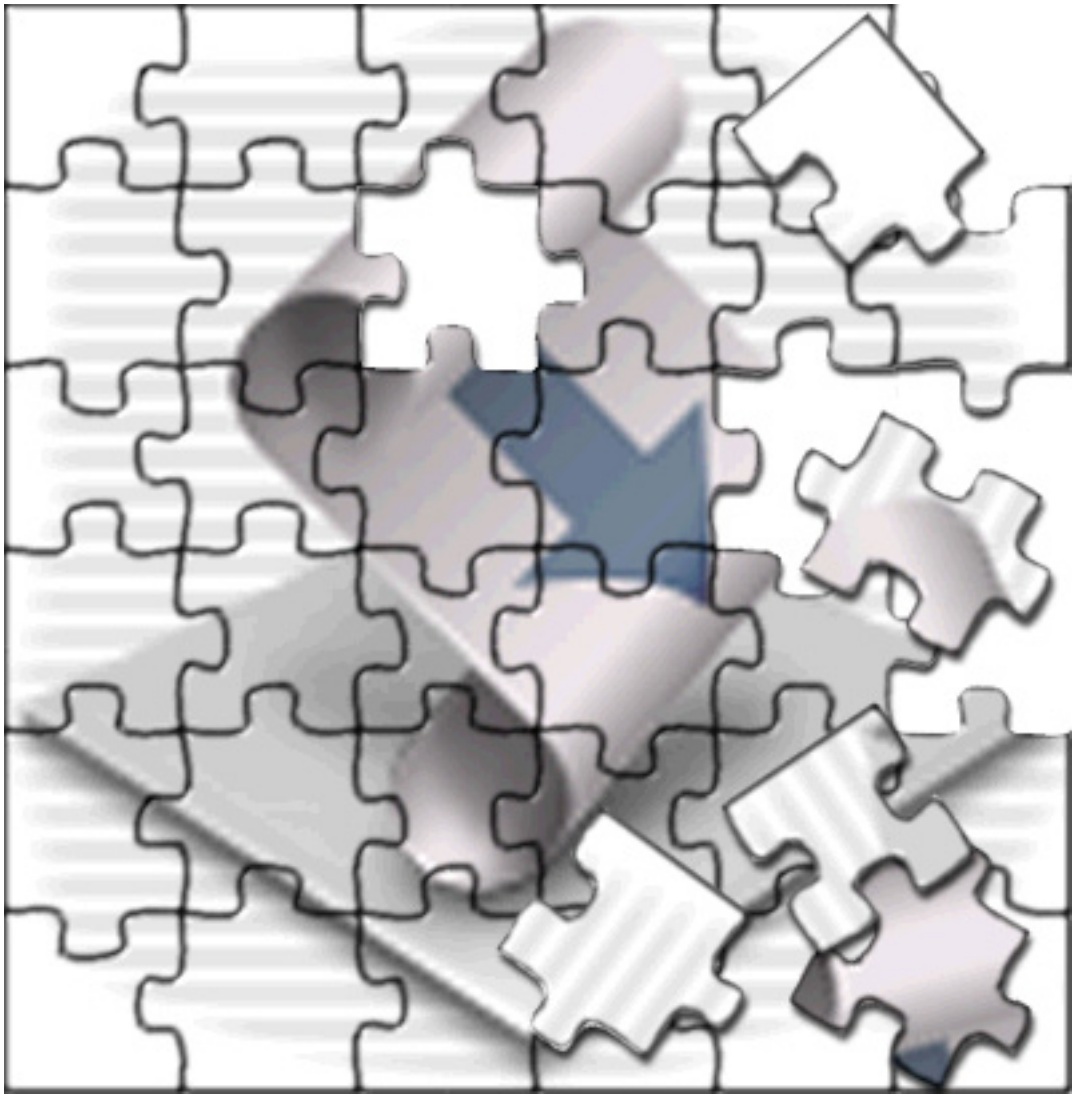
AppleScript für Absolute Starter



Von Bert Altenburg
Übersetzung und Cover-Grafik: Peter Fischer

Einleitung

AppleScript ist eine revolutionäre Technologie von Apple, die eine Kommunikation zwischen Computerprogrammen ermöglicht. Mit AppleScript können Sie z.B.

- eMail mit Mail empfangen und diese dann in einer Datenbank ablegen;
- ein Bildbearbeitungsprogramm anweisen, die Auflösung und Größe einer Serie von Bildern zu ändern und die resultierenden Bilder dann an einen anderen Computer senden oder im Web veröffentlichen
- und vieles, vieles mehr...

Ein "AppleScript" (oder kurz Skript) ist eine Serie von Befehlen. Die dazu verwendete Programmiersprache heißt "AppleScript". Diese Sprache ähnelt dem Englischen und sorgt dadurch dafür, dass AppleScript leicht lesbar, leicht zu schreiben und zu verstehen ist.

Trotz seiner Fähigkeit findet AppleScript bisher nur in einigen Bereichen regelmäßige und ausführliche Anwendung. Die Publishing-Industrie verlässt sich darauf für die Automation ihres Workflows (Photoshop, QuarkExpress, InDesign). "Filemaker Pro"-Entwickler benutzen es dafür, Mac-basierte Kiosksysteme zu erstellen, die Sie in Einkaufszentren und Museen (k-Builder) wiederfinden können. Abgesehen von den erwähnten Programmen sind viele andere bedeutende und bekannte Programme wie GraphicConverter, BBEdit und Word "AppleScriptbar". Dies trifft auch auf viele weniger bekannte Programme zu. Das bedeutet, Sie können AppleScript dazu verwenden, diese Programme herumzukommandieren. Dieses Buch konzentriert sich jedoch nicht auf die Programmsteuerung. Es gibt andere Bücher am Markt, die dies tun. Falls diese Bücher eine Einführung in AppleScript bieten, ist diese gewöhnlich oberflächlich und man geht schnell ans Eingemachte, was generell schon geringe oder gar gute Kenntnisse der AppleScript-Grundlagen voraussetzt. Das Ziel dieses Buches ist es, Ihnen diese Grundlagen zu vermitteln.

Es ist beabsichtigt, dieses Buch regelmäßig zu aktualisieren und zu erweitern. Sie sollten also gelegentlich nach neuen Versionen Ausschau halten (siehe Kapitel 15). Ein zweites Buch, das sich mit der Steuerung verschiedener Programme beschäftigt, ist geplant. Dieses Buch ist Freeware und sollte Sie ermutigen, auch andere Macintosh-Benutzer darauf aufmerksam zu machen. Beachten Sie bitte diesbezüglich im Kapitel 0 dieses Buches, wie Sie für den Mac werben können.

Wenn Sie erst mal in die Welt von AppleScript eingetaucht sind, werden Sie feststellen, dass der Ausdruck „AppleScript“ recht locker für drei verschiedene Begriffe verwendet wird:

- Die AppleScript-Sprache: Die englisch-ähnliche Sprache, die genutzt wird, Befehle an Ihren Mac zu geben;
- Ein AppleScript: Eine Serie von Anweisungen, alias ein Skript, geschrieben in der AppleScript-Sprache;
  und
- Ein Teil des Mac Betriebsystems (Mac OS X), welcher tatsächlich ein AppleScript liest und die enthaltenen Anweisungen ausführt.

Soweit in diesem Buch die Notwendigkeit besteht, sich ausdrücklich auf einen dieser drei Begriffe zu beziehen, werden jeweils folgende Begriffe verwendet:

- Die AppleScript-Sprache
- Ein AppleScript oder ein Skript
- Die AppleScript-Komponente von Mac OS X

Das Erlernen von AppleScript bildet einen idealen Einstieg in die Welt der Programmierung. Die meisten Vorarbeiten, die ein Programmierer in einer Programmiersprache wie z. B. Java zu tun hat, bevor er auch nur die kleinste Aufgabe ausführen kann, entfallen hier. AppleScript ist leicht genug, dass ein 10-jähriger es erlernen kann und dabei trotzdem so mächtig, dass selbst Profis daran Gefallen finden. Das lässt Ihnen einen großen Entwicklungsspielraum. Obwohl sich dieses Buch nicht damit befasst, können Sie mit AppleScript sogar Programme erstellen, die aussehen und funktionieren wie kommerzielle Programme, die Sie auf Ihrem Mac nutzen. Mit Knöpfen, Menüs, Rollbalken und all solchen Dingen. Dazu wird AppleScript Studio benötigt, das Ihnen Ihre Lieblings-Computerfirma kostenlos zur Verfügung stellt.

Was ist der Unterschied zwischen Scripting und Programmierung? Ich neige dazu, zu glauben, dass es sich um Scripting handelt, wenn es einfach ist und um Programmierung, wenn es schwierig ist. Javascripting sieht in meinem Buch jedoch schwer verständlich aus. Vielleicht steht diese Definition damit auf wackeligen Füßen.

Wie nutzt man dieses Buch?

Wie Sie sehen werden, sind einige Absätze in grüner Schrift dargestellt. Wir empfehlen Ihnen, jedes Kapitel (mindestens) zweimal zu lesen. Beim ersten Mal überspringen Sie den grünen Text. Wenn Sie das Kapitel das nächste Mal lesen, sehen Sie sich auch die grünen Absätze an. Sie werden das bisher Erlernte

vertiefen und weitere Besonderheiten, die beim ersten Mal abgelenkt hätten, zusätzlich lernen. Wenn Sie das Buch so nutzen, wird sich die Lernkurve auf einen erträglichen Steigungswinkel einpendeln.

Dieses Buch enthält dutzende von Skript-Beispielen. Um sicherzustellen, dass eine Erklärung auch mit dem richtigen Skript in Verbindung gebracht wird, ist jedes Skript durch eine Nummer zwischen zwei eckigen Klammern so gekennzeichnet: [4]. Die meisten Skripts bestehen aus zwei oder mehr Zeilen. Manchmal wird dann eine zweite Nummer dazu benutzt, eine spezielle Zeile zu bezeichnen. Zum Beispiel bezieht sich [4.3] auf die dritte Zeile von Skript [4].

Sie können das Reiten nicht lernen indem Sie ein Buch lesen. Ähnlich werden Sie auch nicht AppleScript lernen, wenn Sie es nicht schaffen, Ihre Flossen an den Mac zu bekommen. Dies ist ein elektronisches Buch. Sie haben keine Ausrede, nicht zum Skript Editor umzuschalten (siehe Kapitel 2).

Kapitel 0

Bevor wir beginnen

Ich schrieb dieses Buch für Sie. Da es kostenlos ist, erlauben Sie mir bitte, einige Worte über die Werbung für den Mac zu verlieren. Jeder Macintosh-Nutzer kann mit wenig Aufwand dazu beitragen, für seine bevorzugte Computer-Plattform zu werben. So geht‘s:

1) Je effizienter Sie mit Ihrem Mac arbeiten, desto leichter ist es, andere dazu zu bringen, einen Mac in Erwägung zu ziehen. Versuchen Sie also auf dem Laufenden zu bleiben, indem Sie Mac-orientierte Web-Seiten besuchen und Mac-Magazine lesen. Natürlich ist es auch großartig, AppleScript zu lernen und anzuwenden. In Firmen kann die Verwendung von AppleScript Unmengen an Geld und Zeit sparen.

2) Zeigen Sie der Welt, dass nicht jeder einen PC benutzt, indem Sie den Macintosh mehr ins Rampenlicht rücken. Ein hübsches Mac T-Shirt zu tragen ist eine Sache, aber sogar von zuhause aus können Sie Ihren Mac besser präsentieren. Wenn Sie das Programm CPU Monitor (im Utilities-Ordner innerhalb des Applications-Ordners auf Ihrem Mac) starten, werden Sie sehen, dass Ihr Mac die volle Prozessor-Power nur gelegentlich ausnutzt. Wissenschaftler haben einige "distributed computing" (DC) Projekte ins Leben gerufen, wie z. B. Folding@home, das diese ungenutzte Rechenleistung in Anspruch nimmt. Sie laden sich ein kleines, kostenloses Programm, "DC-Client" genannt, herunter und beginnen work units abzuarbeiten. Diese DC–Clients laufen auf der untersten Prioritätsstufe. Wenn Sie ein Programm auf Ihrem Mac benutzen, das die volle Rechenleistung beansprucht, hält sich der DC Client sofort bescheiden im Hintergrund. So werden Sie gar nicht bemerken, dass er läuft. Wie hilft dies dem Mac? Nun, die meisten DC Projekte führen Ranglisten der verarbeiteten work units auf ihren Webseiten. Wenn Sie einem Mac-Team beitreten (Sie werden deren Namen in den Ranglisten bemerken), können Sie dem Mac-Team Ihrer Wahl helfen, dessen Rang zu verbessern. So werden auch Nutzer anderer Plattformen sehen, was Macs auf die Beine stellen können. Es gibt DC Clients für viele Themenbereiche, wie etwa Mathematik, Krankheitsheilung und mehr. Um das für Sie passende DC Projekt zu finden, sehen Sie sich am besten mal auf folgender Webseite um:

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

Ein Problem gibt es mit diesem Vorschlag: Es kann süchtig machen.

3) Stellen Sie sicher, dass die Macintosh Plattform die beste Software hat. Nein, Sie brauchen nicht programmieren zu lernen. Machen Sie es sich zur Angewohnheit, den Entwicklern der Programme, die Sie nutzen, (höfliche) Rückmeldung zu geben. Tun Sie das auch, wenn Sie Software getestet haben und nicht mögen – teilen Sie den Entwicklern mit, warum dies so ist. Melden Sie Fehler, indem Sie die vorgenommenen Aktionen genau beschreiben, bei denen Sie den Fehler entdeckt haben. Eine großartige, kostenlose Anleitung, wie Sie dies tun sollten finden Sie auf:

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

4) Bezahlen Sie für die Software, die Sie nutzen. Solange der Macintosh Software Markt lebensfähig ist, werden die Entwickler auch Software liefern.

5) Bitte nehmen Sie mit mindestens 3 Macintosh Nutzern Kontakt auf, die nichts über dieses Buch wissen und sagen Sie ihnen, wo sie es finden können. Oder machen Sie sie auf die obigen 4 Punkte aufmerksam.

OK, während Sie einen DC–Client im Hintergrund herunter laden, wollen wir beginnen!

Kapitel 1

Ein Skript ist eine Serie von Anweisungen

AppleScript als Teil des Macintosh Betriebsystems kann nur eine sehr beschränkte Anzahl an Aufgaben erledigen. Zum Beispiel kann es einen Warnton erklingen lassen. Lassen Sie uns einen Blick auf das Skript [1] werfen, das benötigt wird, um Ihrem Mac einen Warnton zu entlocken.

[1]
```
beep
```

Dies dürfte das kürzeste Skript der Welt sein, bestehend aus einem einzigen Befehl oder Anweisung. Eine Zeile, die Befehle enthält, nennt man "Statement", auch wenn diese Zeile nur ein Wort lang ist. Wenn das obige Skript von Ihrem Mac ausgeführt wird, ertönt einmal der Warnton.

Um den Ton mehr als einmal erklingen zu lassen, können Sie den "beep"-Befehl mit einer Zahl versehen, die angibt, wie oft Sie den Ton hören wollen [2].

[2]
```
beep 2
```

Wie Sie aus dem Vergleich von Skript [1] und [2] sehen, ist dieses zusätzliche Stück Information nicht vorgeschrieben. Wenn Sie keine Zahl dazu schreiben, unterstellt AppleScript, dass Sie nur einen Ton hören wollen. So ist also 1 der Standard-Wert.

Wenn Sie denken, Beeps wären "PeeCee-isch", warum lassen wir dann AppleScript nicht auf die Macintosh-Art kommunizieren [3], indem wir folgendes Statement benutzen:

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

Sie können sogar eine andere Stimme wie "Fred", "Trinoids", "Cellos" oder "Zarvox" [4] auswählen, um die Standard-Stimme "Victoria" zu ersetzen.

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

#Hinweis: Im Allgemeinen beachtet AppleScript die Groß- und Kleinschreibung nicht. Das bedeutet, es macht nichts aus, ob Sie Groß- oder Kleinbuchstaben verwenden. Jedoch die Stimmen wie "Victoria" und "Zarvox" müssen richtig großgeschrieben werden. Grrr.#

Wie Sie sehen, ähneln die AppleScript-Befehle dem gesprochenen Englisch, was das Skript durchweg lesbar und verständlich macht – sogar wenn Sie noch nie Scripting-Erfahrungen gemacht haben. Obwohl die Skripts [1-4] vielleicht ganz spaßig sind, sehr nützlich sind sie nicht. Die AppleScript-Sprache verfügt über einige weitere Befehle, aber möglicherweise nicht genug, um Sie zu beeindrucken. Die Stärke von AppleScript kommt von der Tatsache, dass es erlaubt, mit anderen Programmen zu kommunizieren. Dies funktioniert, wenn diese Programme "AppleScriptbar" sind. Glücklicherweise sind dies viele Macintosh-Programme. Als Folge davon haben Sie nicht nur den garantierten, jedoch eingeschränkten Befehlssatz der AppleScript-Komponente von OS X zur Verfügung, sondern ebenfalls eine enorme Anzahl an Befehlen, die durch Ihre Programme bereitgestellt werden.

Einige Mac-Programme sind populärer als andere. Eines wird jedoch von jedem Macintosh-Nutzer verwendet: der Finder. Ja, der Finder ist ein Programm. Wenn Sie den Mac einschalten, startet es automatisch und läuft ständig. Es erlaubt Ihnen, Daten zu verschieben, Dateien auf Ihrer Festplatte zu finden, Ordner zu erstellen, sie zu kopieren und umzubenennen und vieles, vieles mehr. Wenn Sie z.B. den Papierkorb leeren, macht das der Finder für Sie. Obwohl Sie das Papierkorb-Entleeren mit der Maus oder der Tastatur durchführen können, mit AppleScript [5] ist es ebenfalls machbar.

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

Wie ein Vorgesetzter müssen Sie genau sagen
- wer eine Aufgabe zu erledigen hat und
- welche Aufgabe erledigt werden soll.

Es ist sinnlos, z.B. PhotoShop anzuweisen, den Papierkorb zu leeren. Photoshop weiß nicht, wie das geht. Die Anweisung zum Entleeren des Papierkorbs muss also an den Finder gerichtet werden.

Wie im richtigen Leben mag es sein, dass die Ihnen übertragene Aufgabe vielleicht nicht besonders anspruchsvoll ist, aber Ihr Mac ist ein äußerst

gewissenhafter Arbeiter und tut, was man ihm aufgetragen hat. Wenn sich im Papierkorb wichtige Dateien befanden – nach der Ausführung des obigen AppleScripts [5] sind sie für immer verloren.

Das erste Statement [5.1] ist das "tell"-Statement, wo wir die AppleScript-Komponente von Mac OS X beauftragen, ein oder mehrere Statements an ein anderes Programm weiterzuleiten – in diesem Fall den Finder. Die AppleScript-Komponente von Mac OS X tut dies so lange, bis sie auf das zwingend erforderliche "end tell"-Statement [5.3] trifft. Im obigen Skript [5] weisen wir AppleScript an, den Finder damit zu beauftragen, den Papierkorb zu leeren um danach die Anweisungen an den Finder wieder zu stoppen. Zusammengefasst werden die Zeilen

```
tell application "xyz"

end tell
```

ein "Tell"-Block genannt. Die Befehle, die vom Programm "xyz" ausgeführt werden sollen, befinden sich innerhalb des Tell-Blocks für Programm "xyz". Übrigens, obwohl AppleScript nicht besonders kleinlich ist wenn man es mit anderen Skript- und speziell Programmiersprachen vergleicht, einige Regeln gibt es auch hier. Eine dieser Regeln ist, dass der Programm-Name, wie im ersten Statement [5.1] zwischen doppelten Anführungszeichen stehen muss.

Es ist auch möglich, mehrere Befehle an den Finder zu senden. Im Beispiel [6] unten gibt es zwei Statements [6.2, 6.3], bestimmt für den Finder. Da beide vom Finder ausgeführt werden sollen, müssen sie innerhalb des Finder-Tell-Blocks stehen.

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

Nach dem Leeren des Papierkorbes öffnet der Finder ein Fenster, das den Inhalt Ihrer Festplatte anzeigt.

Wie Sie sehen, können wir den Finder das tun lassen, was wir wollen. Wir können ihn sogar anweisen, die Größe des Finder-Fensters zu ändern, es an eine andere Position auf dem Bildschirm zu verschieben und vieles, vieles mehr. Sie werden später erfahren, wie dies gemacht wird.

Wir können nun ein Skript erstellen, das beides enthält – Anweisungen an den Finder und an die AppleScript-Komponente selbst [7].

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

Erst ergehen die beiden Anweisungen an den Finder [7.2, 7.3]. Dann wird der "beep"-Befehl von AppleScript selbst ausgeführt. Eigentlich wird damit auch angezeigt, dass das Skript ausgeführt wurde.

Interessanterweise macht es nichts aus, ob der "beep"-Befehl (und andere Befehle, die von der AppleScript-Komponente des Mac OS X verstanden werden) innerhalb oder außerhalb des Tell-Blocks [8] stehen.

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

Wenngleich der Finder den "beep"-Befehl nicht kennt, die AppleScript-Komponente von Mac OS X kann damit umgehen. Das macht das Skript leichter lesbar und verständlicher. Andernfalls hätten Sie einen ersten tell block benötigt, der das erste Finder-Statement [8.2] enthält, dann ein Statement, bestehend aus dem beep-Befehl und schließlich einen zweiten tell block für das letzte Finder-Statement [8.4].

Wohlgemerkt, während Befehle, die die AppleScript-Komponente von Mac OS X versteht, überall in einem Skript stehen können, muss wirklich jede Anweisung an ein bestimmtes Programm, wie den Finder, innerhalb des tell blocks für dieses Programm stehen. Das folgende Skript [9] enthält einen fatalen Fehler (das letzte Statement [9.5]).

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

Die AppleScript-Komponente von OS X weiß nicht, wie die Start-Festplatte zu öffnen ist und ist auch nicht bereit, nach einem Programm zu suchen, das dazu in der Lage ist. Der erste Teil des Skripts (Statements [9.2-3] innerhalb des Tell–Blocks) werden brav ausgeführt, aber das letzte Statement [9.5] kann nicht in die Tat umgesetzt werden.

Tauchen in einem laufenden Skript Probleme auf, werden weitere Befehle nicht ausgeführt [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"
```

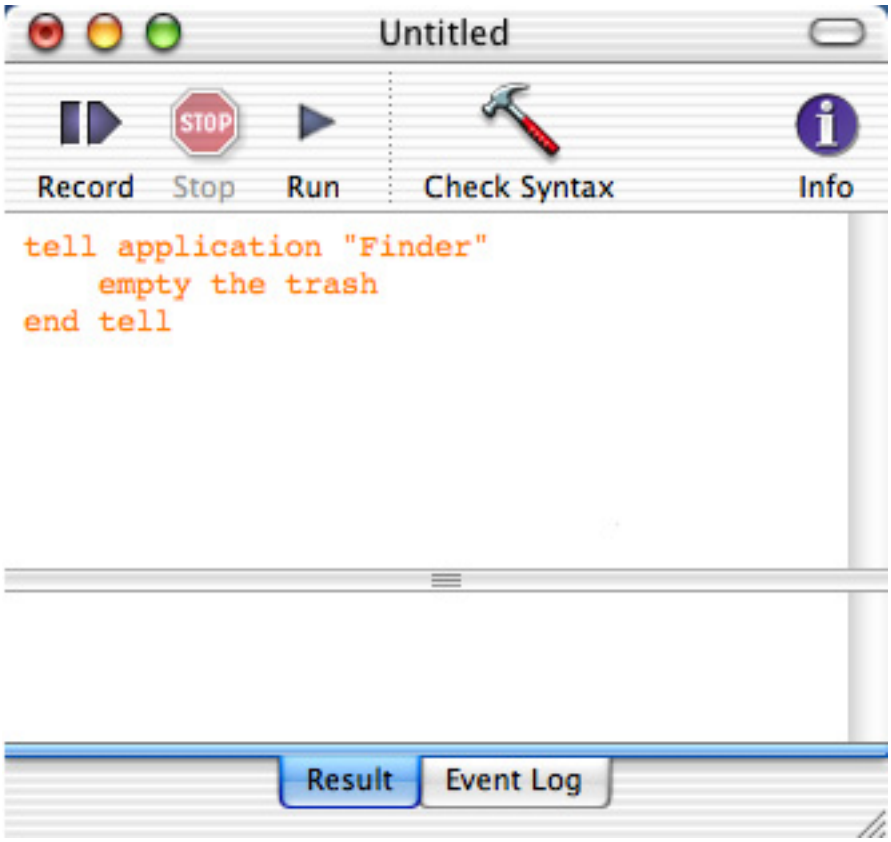
Nach dem Leeren des Papierkorbes wird die AppleScript-Komponente von Mac OS X beim Statement [10.4], welches an die Adresse des Finders gerichtet sein sollte, abbrechen. Sie werden den gesprochenen Satz von Statement [10.5] nicht zu hören bekommen, obwohl in diesem Statement selbst nichts falsch ist.

Kapitel 2

Ausführen und Sichern eines Skripts

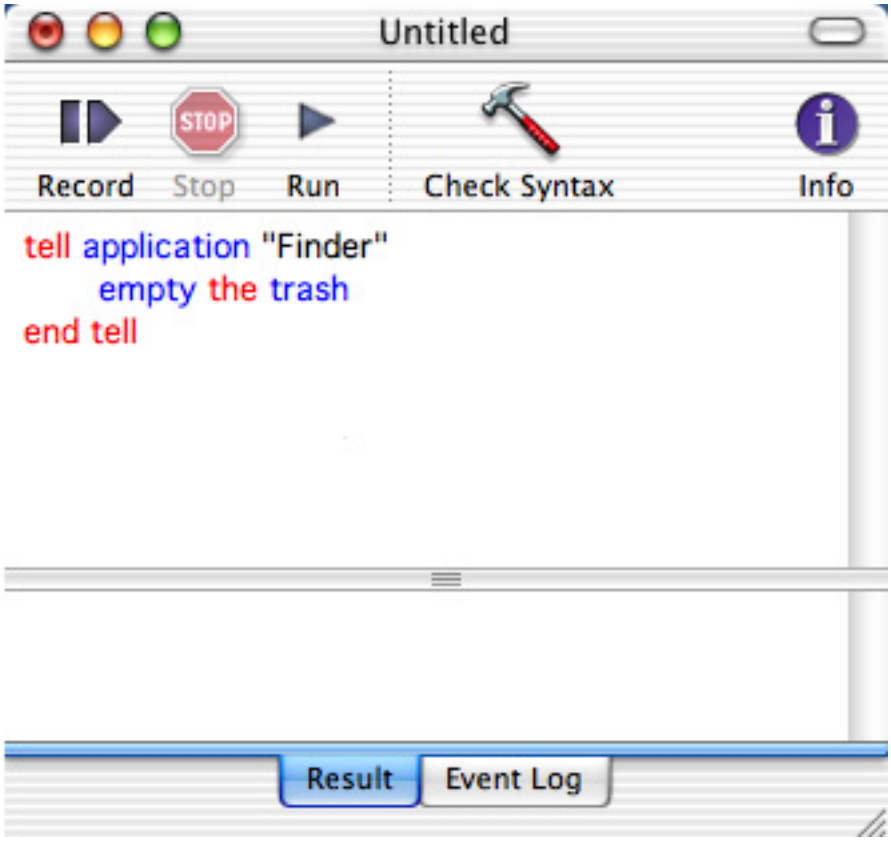
Sie haben bereits eine Anzahl an Skripten gesehen, und es besteht kein Zweifel, dass sie dem gesprochenen Englisch sehr ähnlich sind, was die Skripts sehr leicht lesbar und verständlich macht. Sie hätten Befehle des Skripts – wie "empty the trash" – auch selbst mit der Maus oder der Tastatur ausführen können. Nun wollen wir aber sehen, wie der Mac das für uns erledigen kann.

Der Skripteditor ist ein Programm, in dem Sie ein Skript eingeben, und es ausführen können. Sie finden den Skripteditor im AppleScript-Ordner, der sich wiederum im Applications-Ordner befindet. Nach dem Start werden Sie zwei Bereiche sehen. Im oberen sollten Sie Ihren Skripttext eingeben.



Nahe der Mitte werden Sie einen Knopf mit der Beschriftung "Check Syntax" sehen. Obwohl AppleScript wie Englisch aussieht, ist die AppleScript-Sprache weit davon entfernt, Englisch vollständig zu verstehen. "Yo Finder! Dump my garbage" oder "Hey Finder, clean out the bin" ist nicht gerade das, was der Finder

erwartet. Durch das Prüfen der Syntax Ihres Skripts, auch Kompilierung genannt, führt die AppleScript-Komponente von OS X eine Fehlerüberprüfung durch um festzustellen, ob das Skript verstanden wird. Kommt sie zu dem Schluss, dass dies der Fall ist, wird Ihr Skript schön bunt formatiert. Unkompilierter Text wird in orange angezeigt während nach der Kompilierung alle reservierten Codeworte in rot oder blau dargestellt werden.



Wird das Skript nicht kompiliert, weil Sie einen Fehler gemacht haben, werden Sie eine rätselhafte Mitteilung zu sehen bekommen, die Ihnen anzeigt, dass etwas schief gelaufen ist. Versuchen Sie einmal, eines der Anführungszeichen in Skript [2] wegzulassen und Sie werden selbst sehen, dass die AppleScript-Komponente von OS X nicht länger in der Lage ist, zu verstehen was Sie wollen.

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

Wenn alles gut geht, können Sie den Run-Knopf drücken und Ihr Skript wird ausgeführt werden. Nun, heizen Sie den Skripteditor an, packen eines der Skripte, die Sie hier gesehen haben hinein und versuchen Sie es!

#Als Kurzbefehl für die Syntax-Prüfung können Sie die Enter-Taste drücken. Die Enter-Taste ist die Taste gleich rechts neben der Leertaste (bei Laptops) oder ganz rechts unten auf dem Ziffernblock (Desktop Macs). Die Return-Taste (über der rechten Shift-Taste) arbeitet wie zu erwarten und erzeugt eine neue Zeile nach der aktuellen Zeile. Sie können die Return-Taste nicht für die Syntax-Prüfung benutzen.

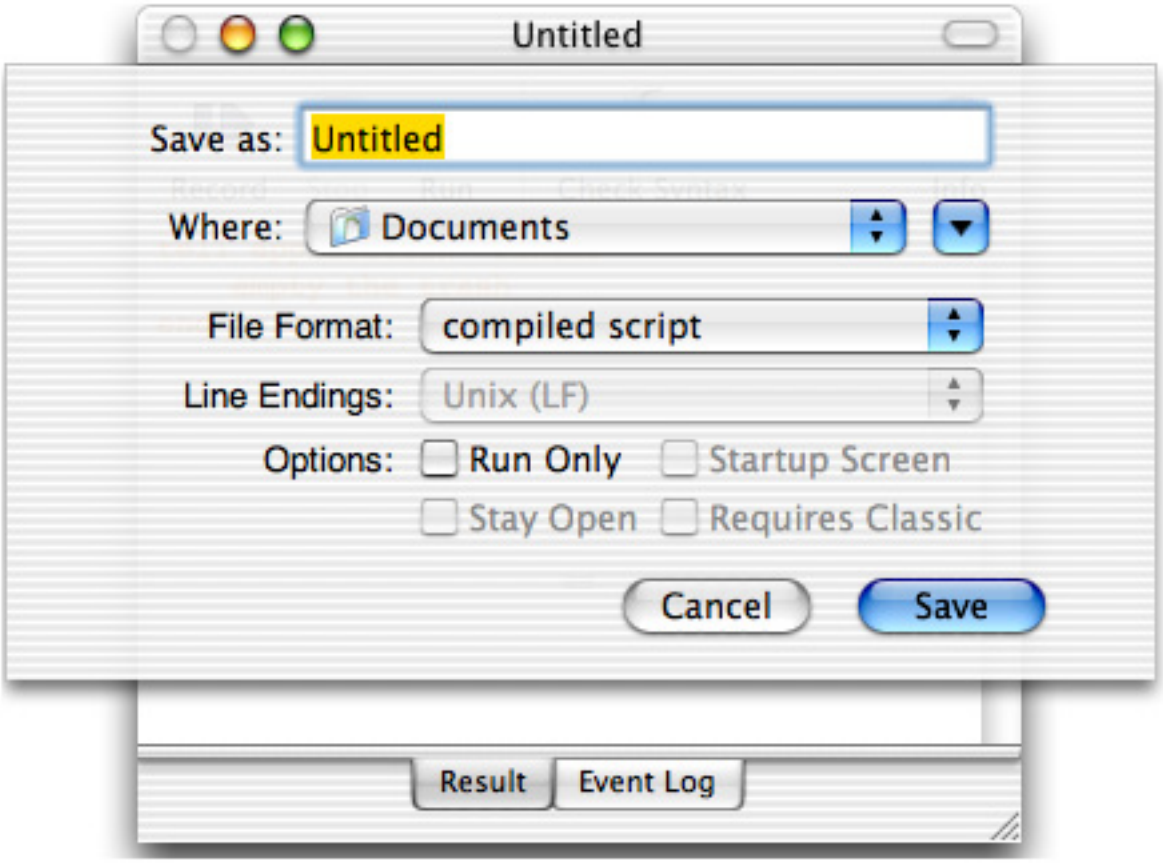
Es ist nicht wirklich nötig, den "Check Syntax"-Knopf vor der Ausführung des Skripts zu drücken. Wenn Sie den Run-Knopf betätigen, wird die Syntax geprüft und soweit alles in Ordnung ist, wird das Skript sofort ausgeführt.

Statt auf den Run-Knopf zu klicken, können Sie auch die Tastenkombination Befehlstaste-R benutzen.#

Sichern Ihres Skripts

Es gibt verschiedene Möglichkeiten, Ihr Skript zu sichern. Wenn das Skript die Syntax-Prüfung nicht erfolgreich besteht, ist Ihre einzige Alternative, es als reinen Text zu sichern.

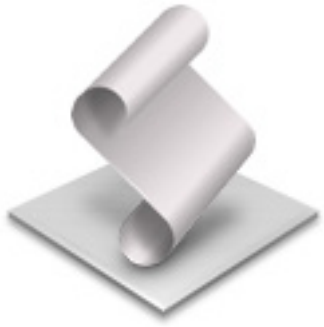
Falls die Syntax-Prüfung problemlos vonstatten geht, erscheint untenstehendes Dialog-Fenster, und Sie können Ihren Text als kompiliertes Skript oder als Programm sichern.



KOMPILIERTES SKRIPT: Wenn Sie auf das Icon Ihres kompilierten, gesicherten Skripts doppelklicken,

öffnet sich der Skripteditor und Sie können das Skript durch Drücken des Run-Knopfes ausführen.

PROGRAMM: Doppelklicken Sie auf das Icon Ihres gesicherten AppleScript-Programms



und das Skript wird sofort ausgeführt. Der Skripteditor wird also nicht extra geöffnet. Gesichert als Programm, können Sie das Skript als Startobjekt (in Ihren Systemeinstellungen) benutzen. Nach dem Anmelden wird Ihr Mac die Aufgaben erledigen, die in Ihrem Skript festgelegt sind. Wenn Sie ein Skript, das als Programm gesichert ist, bearbeiten müssen, starten Sie den Skripteditor und öffnen das Skript, indem Sie "Öffnen" aus dem "Ablage"-Menü wählen.

#WARNUNG: Durch Anklicken des entsprechenden Kästchens im Sichern-Dialog können Sie Ihr Skript als "nur ausführbar" (run-only) sichern. Stellen Sie sicher, dass Sie über eine Kopie Ihres Skripts verfügen, denn ein run-only Skript kann nicht im Skripteditor geöffnet und wieder bearbeitet werden.#

Kapitel 3

Leichteres Scripting (I)

In Kapitel 1 fanden Sie folgendes Skript vor:

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

Lassen Sie uns nun sehen, wie AppleScript und der Skripteditor versuchen, Ihnen das Scripten zu erleichtern.

Anstatt das ganze Wort "application" in der ersten Zeile eines Tell-Blocks einzutippen, können Sie schreiben

```
tell app "xyz"
```

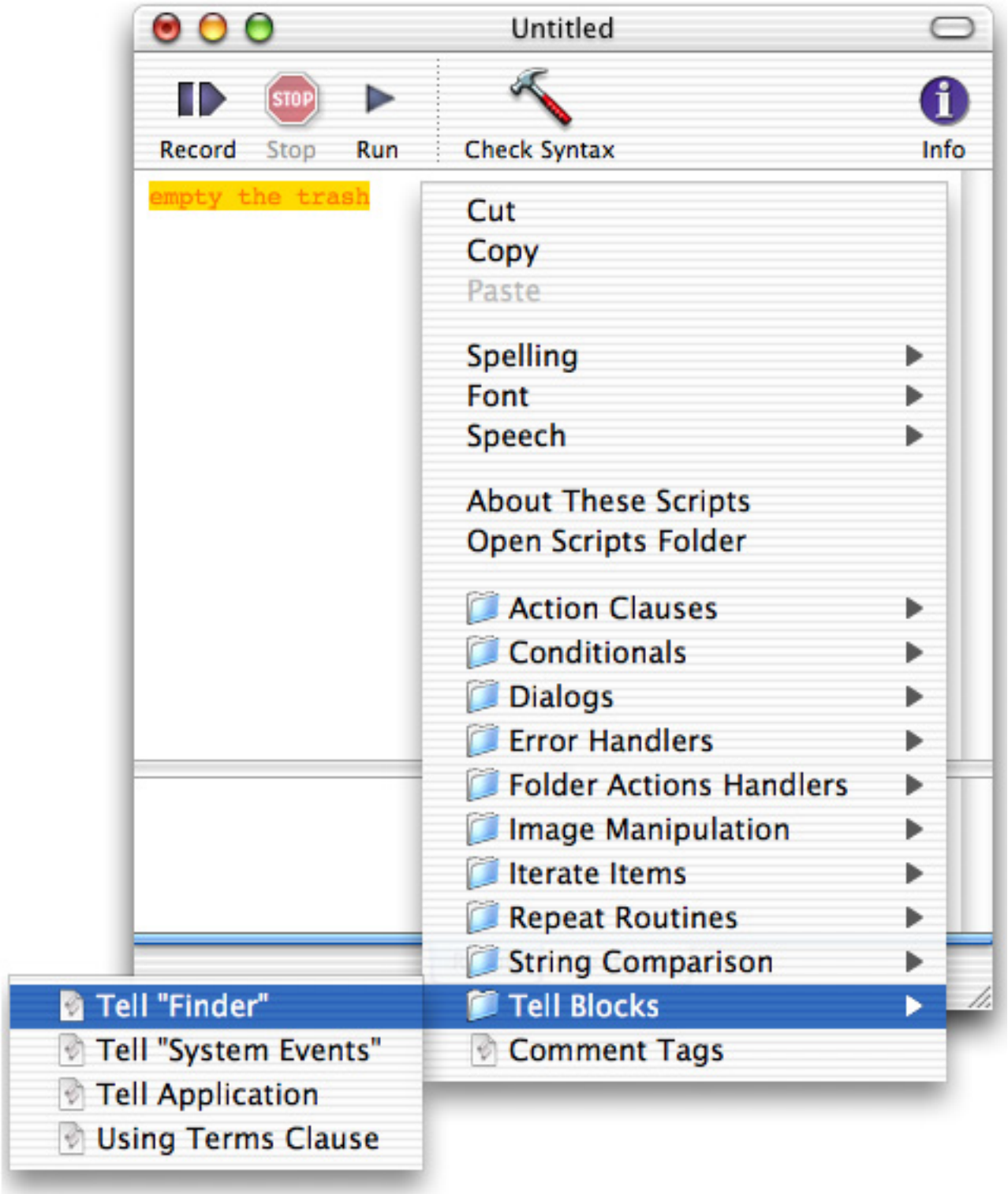
Nach der Kompilierung ergänzt der Skripteditor "app" zu "application". Noch besser, Sie müssen nicht einmal die Schreibweise des Namens von Programm "xyz" genau wissen oder gar eingeben. Tippen Sie einfach irgend etwas ein (vorausgesetzt es ist nicht der Name eines anderen Programms), wie etwa "pqr". Wenn Sie das Skript kompilieren, wird Ihnen AppleScript eine Liste mit allen skriptbaren Programmen auf Ihrem Mac anbieten. Sie wählen nur das zutreffende Programm aus und AppleScript wird "pqr" durch den korrekten Programm-Namen ersetzen und tatsächlich das Schreiben des tell Statements für Sie vollenden.

Durch die Benutzung eines Kontextmenüs, ermöglicht es Ihnen der Skripteditor sogar, einen Tell-Block zu erstellen, ohne dass Sie etwas tippen. Das ist die Art Menü, die erscheint, wenn Sie die Control-Taste gedrückt halten, während Sie etwas anklicken. Dieser Trick kann auf zwei Arten ausgeführt werden:

1) Control-Klick in das obere Feld des Skripteditors. Ein Menü erscheint (siehe Bild auf der nächsten Seite) und am unteren Ende dieses Menüs werden Sie einen Menüpunkt "Tell Blocks" sehen.

2) Wenn Ihr Skript bereits ein oder mehrere Statements für den Finder enthält, wie etwa "empty the trash", welche noch nicht im benötigten Tell–Block eingeschlossen sind, wählen Sie das/die Statement(s) aus und dann geht es

weiter wie unter 1). Sie können das auf dem untenstehenden Bild sehen. Ihre Statements werden automatisch von einem Tell-Block umrahmt.



Kapitel 4

Umgang mit Zahlen

In der Grundschule mussten Sie Rechenaufgaben lösen, bei denen die gepunkteten Stellen auszufüllen waren:

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

In der Hauptschule waren die Punkte dann hinfällig und Variablen mit den Namen "x" und "y" waren angesagt. Rückblickend werden Sie sich vielleicht wundern, warum man durch diese kleine Änderung der Schreibweise derart abgeschreckt wurde.

2 + 6 = x
y = 3 * 4

AppleScript benutzt ebenfalls Variablen. Variablen sind nichts weiter als sinnvolle Namen, um sich auf einen bestimmten Datenteil, wie Zahlen, zu beziehen. Variablennamen werden auch oft "Identifier" genannt, da sie einen Datenteil bezeichnen. Hier sind zwei Beispiele [1] mit AppleScript Statements, wo Variablen mittels des "set"-Befehls ein bestimmter Wert zugewiesen wird.

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

Während die Variablennamen für AppleScript keine besondere Bedeutung haben, können aussagekräftige Variablennamen ein Skript wesentlich leichter lesbar und daher auch verständlicher machen. Das ist ein großer Bonus, wenn Sie einen Fehler in Ihrem Skript suchen müssen (Fehler in Skripten und Programmen werden traditionell "Bugs" genannt). Vermeiden Sie daher nichtssagende Variablennamen wie "x". Der Variablenname für die Breite eines Bildes sollte z.B. "pictureWidth" [2] sein.

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

Beachten Sie bitte, dass Variablennamen aus einem einzelnen Wort (oder notfalls einem einzelnen Buchstaben) bestehen. Nach der Syntax-Prüfung wird der

Variablenname in grün dargestellt, damit Sie sofort erkennen können, dass es sich nicht um ein reserviertes Codewort von AppleScript handelt, welche rot oder blau angezeigt werden. Beachten Sie bitte auch, dass Daten (wie die Zahl "8" in Skript [2]) an ihrer schwarzen Farbe zu erkennen sind.

#
Obwohl Sie viel Freiraum bei der Gestaltung der Variablennamen haben, müssen die Variablennamen doch gewissen Regeln entsprechen. Würde ich sie hier alle aufzählen, wäre das bestimmt schnell langweilig. Die erste Regel, die Sie beachten müssen ist, keine AppleScript-Befehle oder andere reservierte Codewörter für den Variablennamen zu verwenden. Zum Beispiel sind "set", "say", "to" und "beep" Worte, die für AppleScript eine spezielle Bedeutung haben. Mit dem Verfassen von Variablennamen aus zusammengesetzten Wörtern, wie "pictureWidth" sind Sie immer auf der sicheren Seite. Um die Variablennamen lesbar zu halten, ist es empfehlenswert, innerhalb des Variablennamens Großbuchstaben zu verwenden.

Wenn Sie darauf bestehen, einige weitere Regeln zu lernen, lesen Sie den Rest dieses Absatzes. Abgesehen von Buchstaben sind auch Zahlen erlaubt – allerdings nicht am Anfang des Variablennamens. Ebenso ist auch der Unterstrich _ zulässig. #

Da wir nun wissen, wie wir Variablen einen Wert zuweisen, können wir Berechnungen damit anstellen. AppleScript ist fähig, mit den Grundrechenarten umzugehen. Es besteht also kein Anlass, ein anderes, spezielles Programm anzuweisen, die Flächenberechnung eines Bildes vorzunehmen. Hier ist das Skript [3], das genau dies tut.

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

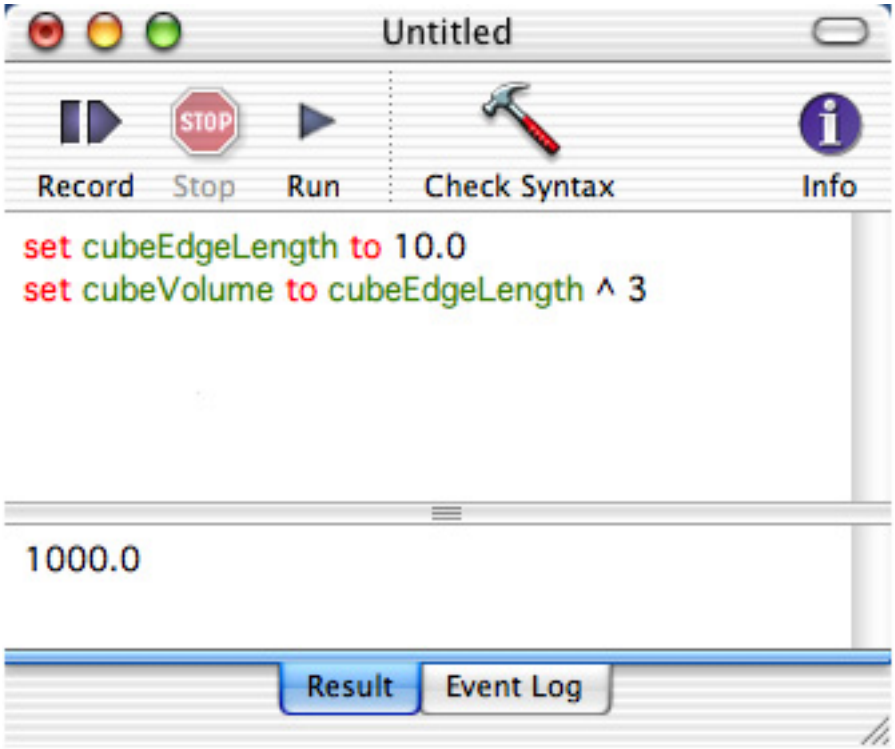
Benutzen Sie folgende Symbole, offiziell als Operatoren bekannt, um mathematische Berechnungen auszuführen.

+ für Addition
- für Subtraktion
/ für Division
* für Multiplikation

Eine Hochzahl (Exponent) schreibt man mit Hilfe des ^ Symbols. Hier ist ein Skript [4], welches das Volumen eines Würfels berechnet.

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

Wenn Sie dieses Skript [4] im Skripteditor ausführen, wird das Ergebnis im unteren Bereich angezeigt. Sollten Sie das Ergebnis nicht sehen können, bewegen Sie den waagrechten Balken nahe der beiden Tabulatoren nach oben. Der Ergebnis-Bereich zeigt das Ergebnis des zuletzt ausgeführten Statements an. Falls Ihr Skript nur aus dem Statement [4.1] besteht, zeigt der Ergebnis-Bereich 10.0 an. Für das komplette Skript [4] wird das Ergebnis 1000.0 dargestellt. Das ist der berechnete Wert des Ausdrucks "cubeEdgeLength^3".



Zahlen können in zwei Typen unterschieden werden: Ganzzahlen (Integer) und Bruchzahlen. Sie können jeweils ein Beispiel in den Statements [1.1] und [1.2] sehen. Ganzzahlen werden zum Zählen benutzt, was wir später noch tun werden, wenn wir eine Befehls-Serie eine bestimmte Anzahl mal wiederholen müssen (siehe Kapitel 13). Bruchzahlen oder reelle Zahlen (kurz: reals) kennen Sie z.B. von den durchschnittlich verwandelten Torchancen beim Fußball. Übrigens beide, Ganzzahlen und Bruchzahlen können negativ sein – das ist Ihnen sicher von Ihrem Bankkonto her bekannt.

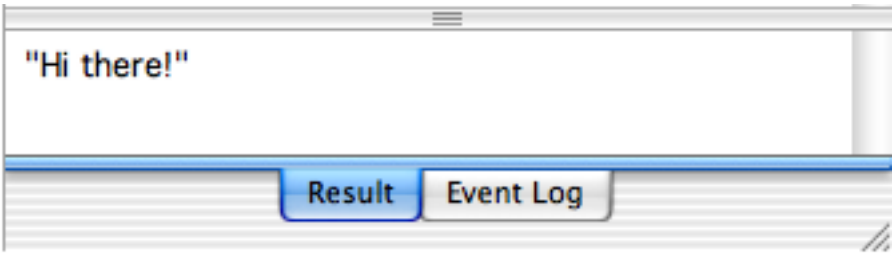
Kapitel 5

Umgang mit Text

Variablen können nicht nur benutzt werden, um Zahlen zu speichern. Das speichern von Text ist ebenfalls möglich. Ein Stückchen Text, sogar wenn es leer ist oder nur ein Zeichen lang, wird "string" genannt. Strings müssen zwischen doppelten Anführungszeichen platziert werden. Hier sind drei Beispiele [1] von Variablen mit erläuternden Namen und einem entsprechenden Textwert.

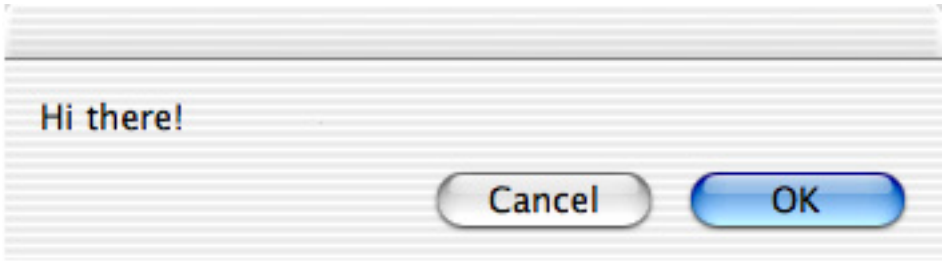
[1]
```
set emptyString to ""
set notEmptyContainsASpace to " "
set greeting to "Hi there!"
```

Nach der Ausführung des Skripts [1] zeigt der Ergebnis-Bereich den String zwischen doppelten Anführungszeichen an. Der Ergebnis-Bereich teilt uns also nicht nur das Ergebnis, sondern auch den Datentyp (Zahlen ohne Anführungszeichen; Textwerte mit Anführungszeichen) mit.



Da der Ergebnis-Bereich nur jeweils das Ergebnis des zuletzt ausgeführten Statements anzeigt, wird nur der entsprechende Gruß aus Statement [1.3] dargestellt.

Zusätzlich zum Ergebnis-Bereich bietet AppleScript eine nützliche Alternative, um mit der Welt in Kontakt zu treten: ein Dialog-Fenster. Es sieht so aus:



Sie können es mit folgendem Befehl aufrufen: "display dialog" gefolgt von den Daten (Zahl oder Text), die Sie anzeigen lassen möchten. Der obige Dialog wurde durch folgendes Skript [2] erstellt:

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

Warum haben Textwerte doppelte Anführungszeichen? AppleScript mag ja nur ein ziemlich eingeschränktes Vokabular enthalten, aber Ihr Skript zu lesen und dabei zu entziffern, was ein Befehl ist und was nicht, ist für einen Computer immer noch ziemlich schwierig. So verlässt sich AppleScript auf Hinweise, die ihm helfen, die Bedeutung jedes Elements eines Skript-Statements zu verstehen. Aus diesem Grund müssen wir Textwerte zwischen doppelte Anführungszeichen schreiben. Andernfalls könnte AppleScript einen Textwert mit einem Variablennamen verwechseln. Probieren Sie mal folgendes Skript [6] aus:

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

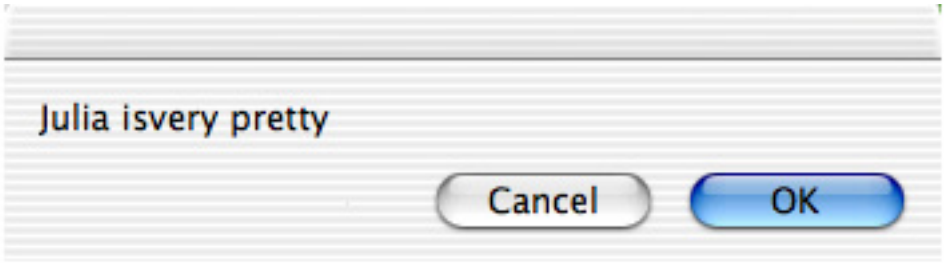
Führen Sie das Skript selbst aus, um zu sehen was es tut. Das Statement [6.2] zeigt den Text 'stringToBeDisplayed' an, wogegen das Statement [6.3] den Dialogtext 'Hi there!' hervor bringt. Da der Skripteditor ein kompiliertes Skript in unterschiedlichen Farben anzeigt, ist es für Sie leicht zu erkennen, dass 'stringToBeDisplayed' im Statement [6.3] ein Variablenname ist, weshalb es auch in grün dargestellt wird. 'stringToBeDisplayed' im Statement [6.2], in schwarzer Farbe, ist hingegen ein Textwert (String). Gelegentlich wird Ihnen die farbige Formatierung helfen, Fehler schneller zu finden.

Wie bereits gesagt benötigt AppleScript Hilfestellungen um das Englisch-ähnliche Skript in etwas zu übersetzen, was der Mac verstehen kann. Hier ein weiteres Beispiel dafür, warum diese Hinweise wichtig sind: Wenn wir 'dreißig' als eine Zahl zwischen doppelten Anführungszeichen schreiben, z.B. als "30", handelt es sich nicht mehr länger um eine Zahl, sondern um einen Textwert. Den richtigen Datentyp zu erkennen ist sehr wichtig, da einige Vorgänge nur an einem bestimmten Datentyp ausgeführt werden können. Während Sie z.B. zwei Zahlen dividieren können, funktioniert das nicht mit zwei Textwerten. Werfen wir einen Blick auf ein paar Arbeitsabläufe, die auf Textwerte angewendet werden können.

Wie Zahlen (und Wähler) können Strings manipuliert werden. Sie können Strings zusammenfügen, was "Concatenation" genannt wird, indem Sie das kaufmännische "Und"-Zeichen (&) verwenden [7].

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

Im dritten Statement [7.3] verknüpfen wir drei Textwerte. Zwei davon gehen auf Variablen zurück.



Beachten Sie bitte, dass die Anzahl der Leerzeichen zwischen den Strings und dem "Kaufmanns-Und" keine Auswirkung auf den resultierenden Textwert (Variable: resultingString) hat. Wenn Sie mehr als ein Leerzeichen eingegeben haben, reduziert dies der Skripteditor nach der Kompilierung auf 1. Brauchen Sie ein oder mehrere Leerzeichen, um die Wörter eines anzuzeigenden Satzes zu trennen, müssen Sie diese zwischen die doppelten Anführungszeichen eines Textwertes setzen. Im Statement [7.3] sollte, abgesehen vom Leerzeichen links des Wortes 'is', auch ein weiteres Leerzeichen nach dem Buchstaben 's' stehen.
Es gibt noch weitere Befehle, die auf Textwerte einwirken. Einige davon erfordern Stoff, den wir erst in späteren Kapiteln behandeln, weshalb wir es an dieser Stelle vorerst dabei bewenden lassen.
Aber wir können Ihnen ein weiteres Beispiel eines Befehls geben, der sich auf Textwerte bezieht. Sie können nach der Länge des Textes fragen [8].

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

Wenn Sie dieses Skript ausführen, ist das im Ergebnis-Bereich angezeigte Ergebnis 4. Denken Sie also bitte daran, dass bei der Berechnung der Länge eines Textes die Leerzeichen ebenfalls mitzählen.

Da Anführungszeichen benutzt werden, um Beginn und Ende eines Textwertes zu markieren, könnten Sie nun denken, dass es unmöglich ist, einen Text zu benutzen, der auch Anführungszeichen enthält. Natürlich bietet die AppleScript-Sprache auch hier einen Ausweg, genannt 'Maskierung'. Setzen Sie einen Backslash (linksseitiger Schrägstrich) vor das Anführungszeichen und AppleScript versucht nicht länger, dieses als das Ende eines Textwertes zu interpretieren [9].

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

#Wenn Sie genau darüber nachdenken, ist das Problem noch nicht gelöst. Denken Sie an den seltenen Fall, dass Ihr Text ein Anführungszeichen enthält, dem ein Backslash voraus geht. Angenommen Sie wollen den folgenden Text in einem Dialog darstellen:

blah blah \" blah.

Zunächst setzen wir einen Backslash vor den Backslash. Das bedeutet, dass AppleScript jede spezielle Bedeutung des nächsten Zeichens ignorieren wird, im Beispiel den zweiten Backslash. Natürlich müssen wir immer noch das Anführungszeichen maskieren, sonst würde AppleScript annehmen, dass unser Text hier endet. Darum muss dem Anführungszeichen ebenfalls ein Backslash voran gestellt werden, wie wir es bereits gesehen haben. Alles zusammen ergibt sich nun folgendes Statement [10]:

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

Um es etwas zu veranschaulichen, habe ich die maskierenden Backslashes im Fettdruck dargestellt, etwas was der Skripteditor nicht macht. Mit Backslashes sollten Sie aufpassen, denn sie können vor einigen anderen Zeichen eine spezielle Bedeutung haben. Zum Beispiel bezeichnet \n eine neue Zeile (Return) und \t einen Tabulatorsprung.#

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

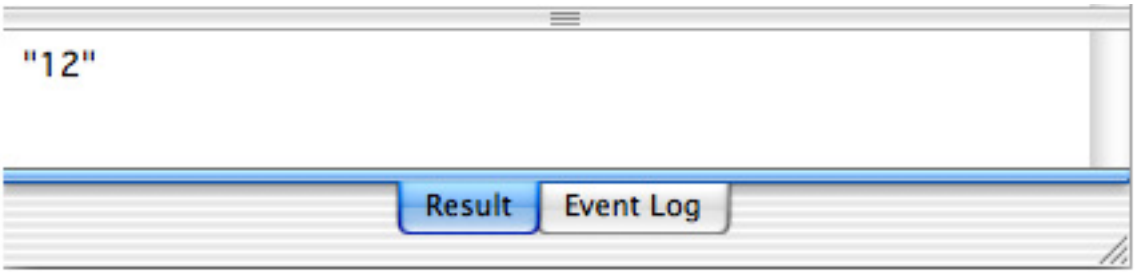
Wenn Sie versuchen, das Skript [11] auszuführen, können Sie sich ein Bild davon machen, wie sympathisch die Skriptsprache AppleScript ist. Sie versucht sogar, den Textwert in eine Zahl umzuwandeln. Hätte der String "15" statt "fifteen"

gelautet, wäre diese Umwandlung geglückt. Die Umwandlung von einem Datentyp in einen anderen wird Coercion genannt. Sie können dies, wie die nächsten beiden Beispiele zeigen, auch erzwingen [12].

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

Der Ergebnis-Bereich des Skripteditors zeigt Ihnen das Ergebnis des letzten Statements [12.2].

Die in 'coercedToString' enthaltenen Daten sind ein String, wie die Anführungszeichen zweifellos erkennen lassen (siehe Bild unten).



Bei einem Skript, das mit dem ersten Statement [12.1] geendet hätte, wäre im Ergebnis-Bereich 15 ohne Anführungszeichen zu sehen gewesen, was bedeutet, dass das Ergebnis eine Zahl und kein Text ist.

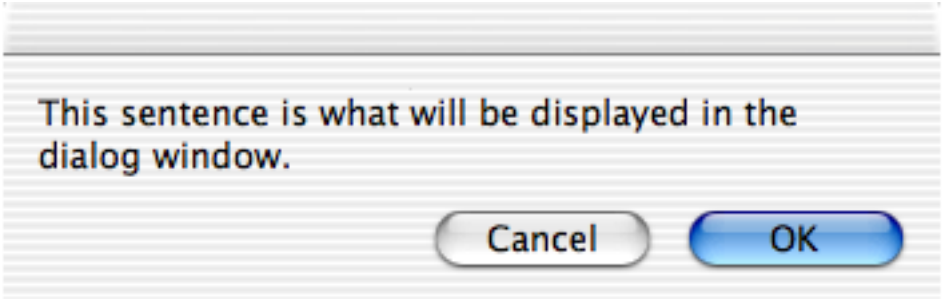
Kapitel 6

Listen

In den vorangegangenen Kapiteln haben Sie gesehen, wie man einfache Skripte erstellt, um grundlegende Berechnungen anzustellen und mit Text umzugehen. Dem Benutzer des Skripts könnte mit dem 'display dialog'-Befehl [1] das Ergebnis präsentiert werden. In diesem und dem nächsten Kapitel werden wir den 'display dialog'-Befehl verwenden, um mehr über weitere Aspekte der AppleScript-Sprache zu lernen.

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

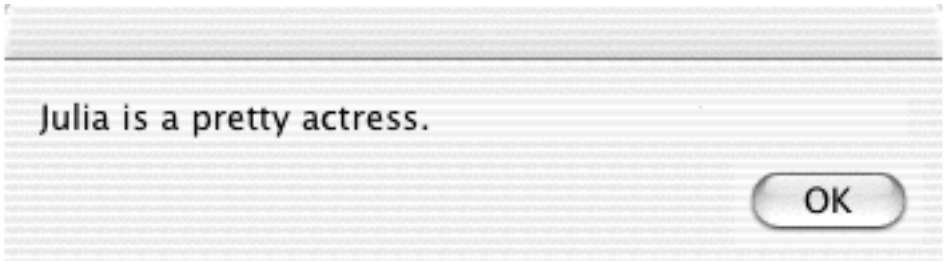
Wenn Sie Skript [1] ausführen, werden Sie ein Dialog-Fenster sehen, das standardmäßig über zwei Buttons verfügt: Ein Cancel–Knopf und ein OK–Button.



Der Cancel-Button bricht die Ausführung des weiteren Skripts ab. Da das obige Skript über keine weiteren Statements verfügt, ist der Cancel-Button reichlich überflüssig. Wir können ihn loswerden, indem wir die Buttons unseres Dialog–Fensters selbst festlegen. Der 'display dialog'–Befehl erlaubt Ihnen, eine Liste der Buttons zu definieren. In unserem Fall brauchen wir nur einen, nämlich einen Button mit der Aufschrift 'OK' [2.2].

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

Wenn Sie nun das Skript ablaufen lassen, werden Sie Sehen, dass der Cancel-Button verschwunden ist.



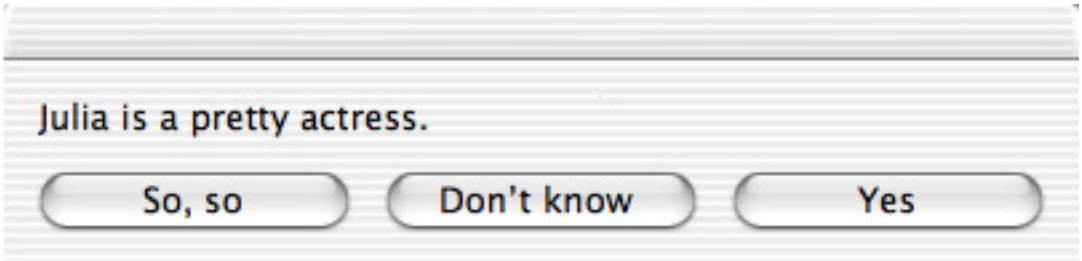
Wie das zweite Statement [2.2] zeigt, besteht die Liste aus dem Textwert "OK", der zwischen geschweifte Klammern gesetzt wird. Wozu dienen diese geschweiften Klammern? Wie wir bereits gesehen haben, benötigt AppleScript kleine Hilfestellungen, um jedes Element eines Skript-Statements zu verstehen. Der nötige Tipp, um AppleScript eine Liste erkennen zu lassen, besteht aus den geschweiften Klammern.

Die Liste in Statement [2.2] enthält nur einen Eintrag, den String "OK". Wenn eine Liste mehrere Positionen beinhaltet, werden diese durch Kommas getrennt [3].

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

Die Liste im obigen Statement [3] enthält 4 Einträge: Einen Textwert (das Komma innerhalb dieses Wertes wird nicht als Trennzeichen interpretiert) und drei Zahlen. Nun wollen wir uns wieder dem 'display dialog'-Befehl zuwenden und einen Dialog mit mehreren Buttons erstellen. Der AppleScript–Befehl 'display dialog' erlaubt einen, zwei oder drei Buttons, mit jeweils einem (recht kurzen) Text Ihrer Wahl. Um also ein Dialog-Fenster mit drei Buttons zu erstellen, müssen wir eine Liste mit drei Einträgen definieren. [4.2].

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



#Sie werden bemerken, dass kein Button farblich hervorgehoben ist, wenn Sie Buttons selbst definieren [2.2, 4.2]. Das bedeutet, dass die Person, die das Skript ausführt, nicht die Enter-Taste benutzen kann, um den Dialog zu beenden. Da diese Person ein Macintosh-Benutzer ist, der auf Benutzerfreundlichkeit Wert legt, wollen wir uns nun darum kümmern, dass ein Button hervorgehoben wird [5].

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

Statt des Namens des Buttons, der hervorgehoben werden soll, können Sie sich auch auf die Nummer des Buttons [6] beziehen, im Beispiel das dritte Objekt in der erstellten Liste.

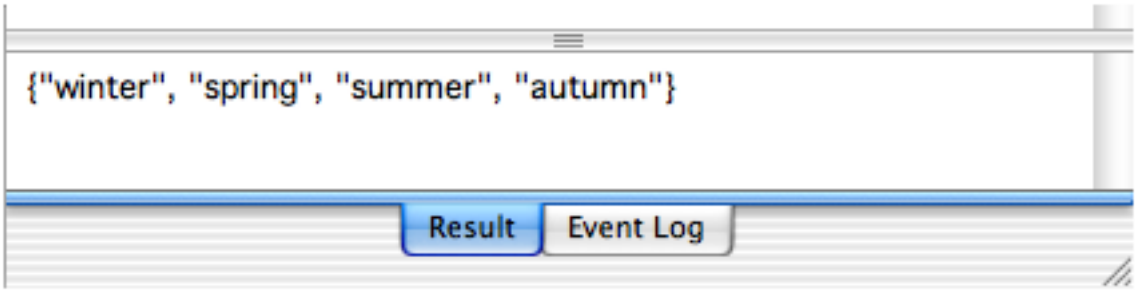
```
[6]
set stringToBeDisplayed to "Julia is a pretty actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don't know", "Yes"} ¬
      default button 3
#
```

Im nächsten Kapitel werden Sie lernen, wie man heraus bekommt, welcher Button gedrückt wurde. Jetzt wollen wir aber erst noch mehr über Listen lernen.

Listen können benutzt werden, um eine Serie von Daten zu speichern. Dazu müssen Sie wissen, wie man Listen bearbeitet und Daten daraus abfragt. Es ist einfach, Daten an den Anfang oder das Ende einer Liste anzufügen. Um Einträge zu einer Liste hinzuzufügen, benutzen wir das "Kaufmanns-Und", wie wir es bereits bei Textwerten gesehen haben.

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

Im Statement [7.4] erzeugen wir eine Liste, die aus 4 Einträgen besteht. Der Ergebnis-Bereich zeigt die geschweiften Klammern, charakteristisch für den Datentyp 'Liste'.



Bitte beachten Sie, dass 'addToBegin' and 'addToEnd' nur Variablennamen sind [7.1-2], die gewählt wurden, um Ihnen zu helfen, das Skript besser zu verstehen. Sie haben keine spezielle Funktion und tun nichts anderes, als sich auf die Originaldaten zu beziehen. Die grüne Farbe ist ein sicheres Zeichen dafür, dass es sich um Variablennamen handelt.

Sie können sich durch eine Zahl auf jeden Eintrag (item) in einer Liste beziehen. Die Position ganz links ist item 1, die nächste ist item 2 etc. Dies erlaubt Ihnen, einen bestimmten Wert einer Liste abzufragen, oder den Wert eines Listeneintrags (wie Text oder Zahl) abzuändern. Hier ist ein Beispiel [8].

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

Der 'get'-Befehl im letzten Statement [8.3], ermöglicht es uns, den Wert der Variablen myList im Ergebnis-Bereich anzeigen zu lassen. Dieser wird den Wert der Variablen myList so darstellen: {"winter", "spring"}

Konzentrieren wir uns auf das zweite Statement [8.2], könnten wir das selbe Ergebnis erreichen, wenn wir eines der beiden Statements aus Skript [9] statt des zweiten Statements in Skript [8] verwenden.

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

Das erste Statement [9.1] zeigt wieder einmal auf elegante Weise die Englisch-ähnliche Natur von AppleScript. Diese verbale Nummerierung funktioniert bis zum 10ten Objekt. Danach müssen Sie auf 'item 11', etc. zurückgreifen. Alternativ dazu können Sie auch '11th item', usw. entsprechend [9.2], schreiben. Abgesehen von den verbal nummerierten Listeneinträgen, gibt es noch das 'last' Objekt [10].

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

Sie brauchen also nicht zu wissen, wie lang eine Liste ist, um den Wert des letzten Listeneintrags auszulesen (oder diesen Wert zu verändern).

AppleScript ermöglicht Ihnen auch den Bezug auf Listenpositionen, indem in umgekehrter Richtung gezählt wird, z.B. von rechts nach links. Für diesen Zweck nutzen Sie negative Zahlen, wobei item -1 das letzte Objekt, item -2 das vorletze usw., ist. Skript [11] liefert exakt das gleiche Resultat wie Skript [10].

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

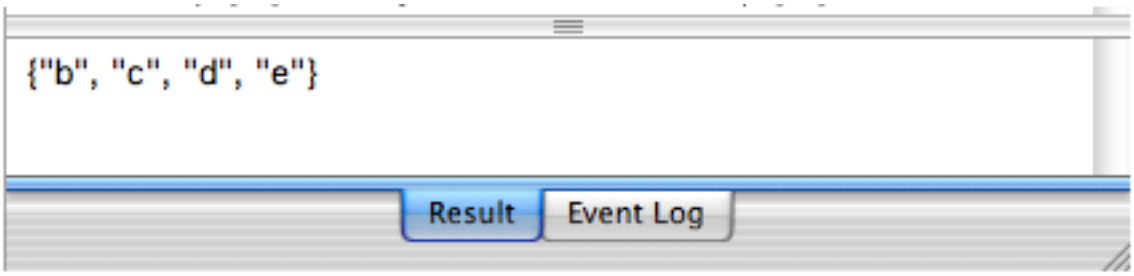
Sie wissen nun, wie Sie eine Liste erstellen können, wie man Einträge in diese einfügt und wie man die Werte von Listeneinträgen ändern kann. Ihnen ist ebenfalls bekannt, wie man einzelne Werte einer Liste abrufen kann. Möglicherweise interessiert es Sie jetzt auch noch, wie man eine Teilliste erstellt [12].

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

Im Statement [12.2] können Sie auch 'thru' statt 'through' verwenden, falls Sie dies bevorzugen. Wenn Sie die item-Zahlen [13.2] des bestimmten Bereichs umkehren, wird die resultierende Liste jedoch nicht umgekehrt [13].

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

Das Ergebnis ist genau das gleiche wie bei Skript [12].

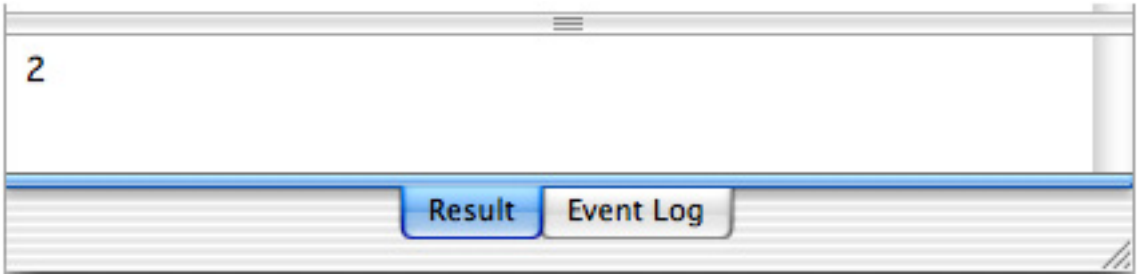


Um die Reihenfolge der Listeneinträge umzukehren, steht Ihnen der 'reverse'-Befehl zur Verfügung [14].

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

Gelegentlich müssen Sie wissen, wie lang Ihre Liste ist. Die Antwort bekommen Sie ganz einfach, indem Sie eines der beiden folgenden Statements [15] benutzen:

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



Letztendlich können Sie auch auf ein zufälliges Listenobjekt verweisen [16].

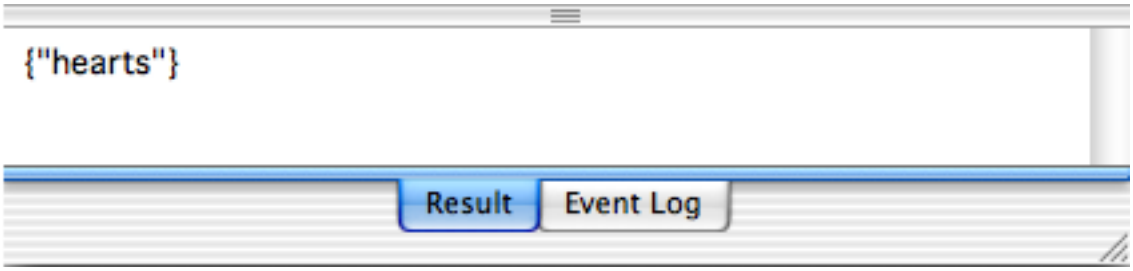
[16]
```
set x to some item of {"hearts", "clubs", "spades", "diamonds"}
```

Ich weiß, dies ist ein langes Kapitel, aber wir müssen wirklich ein paar Sachverhalte behandeln, die mit beidem, Listen und Textwerten, zu tun haben. Wir konnten das nicht in Kapitel 5 besprechen, das sich mit Textwerten beschäftigt, da wir Listen noch nicht ausführlich behandelt hatten. Halten Sie durch oder machen Sie eine kleine Pause.

In den vorangegangenen Kapiteln haben Sie gelernt, dass es möglich ist, einen Datentyp in einen anderen umzuwandeln (Coercion). Nun werden wir Ihnen zeigen, wie man einen Textwert (oder eine Zahl) in eine Liste verwandelt [17].

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

Nach der Umwandlung eines Textwertes in eine Liste ist das Ergebnis eine Liste, die nur einen einzigen Eintrag enthält (einen String):



Wenn man mit Listen und Textwerten umgeht, stellt die Coercion eine wichtige Sicherheitsmaßnahme dar, wie gleich gezeigt wird.

Sie erinnern sich, ein kaufmännische Und wird dazu benutzt, Textwerte zu verbinden. Was geschieht nun, wenn Sie das "Kaufmanns-Und" benutzen, um einen Textwert einer Liste hinzuzufügen [18]?

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

Der Datentyp der Variable 'theResult' hängt von dem Datentyp ab, der zuerst im zu untersuchenden Ausdruck [18.3] auftaucht. Da der Ausdruck mit der Variablen myList, welche eine Liste ist, beginnt, ist das Ergebnis eine Liste. Probieren Sie es selbst aus und überprüfen Sie den Ergebnis-Bereich. Er offenbart den Datentyp, indem er die geschweiften Klammern anzeigt. Hätten wir die Reihenfolge der Variablennamen myList und myString umgekehrt und geschrieben [19.3]:

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

wäre das Ergebnis ein Textwert gewesen. Es ist selbstverständlich, dass dieses Verhalten, wenn Sie nicht aufpassen schnell zu einem Bug in Ihrem Skript führen kann. Führen Sie eine Coercion [20] durch, um irgendetwas Unerwartetes zu vermeiden.

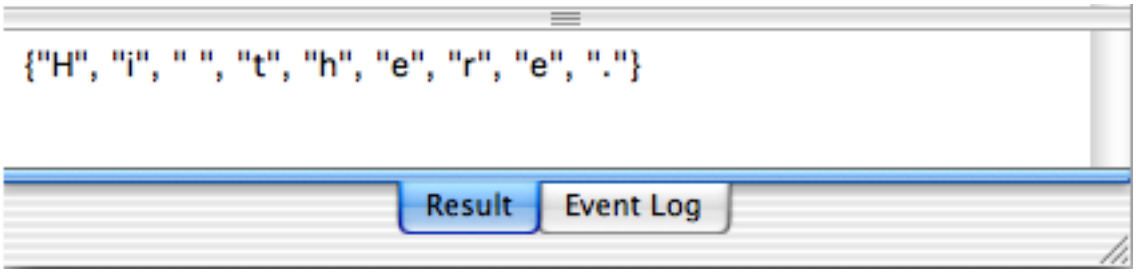
[20]
```
set myList to {"a"}
set myString to "b"
set theResult to (myString as list) & myList
```

Im Statement [20.3] wird der Text "b", auf den sich die Variable 'myString' bezieht, in eine Liste {"b"} umgewandelt, ungeachtet des Datentyps, auf den sich 'myString' bezieht. Da die Variable 'myString' nun auf eine Liste verweist, wird auch das Ergebnis eine Liste sein [20.3].
#

Abgesehen von der Coercion, um einen Textwert in eine Liste zu verwandeln, ist es auch möglich, eine Liste zu erstellen, die jeden Buchstaben eines Strings enthält [21]. (Hinweis: Obwohl auch 'item' anstatt 'character' funktioniert, wird unter OS X davon abgeraten.)

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

Die sich ergebende Liste, zu sehen im Ergebnis-Bereich, sieht so aus:



Statt der Aufspaltung in einzelne Buchstaben, wollen Sie vielleicht eher einen Satz in einzelne Wörter zerlegen. Dies geschieht unter Anwendung von 'AppleScript's text item delimiters' (Anm. d. Übersetzers: hier keine Übersetzung, da fester Begriff der AppleScript-Sprache). Sie legen ein Zeichen fest, das als Trennungszeichen dienen soll, um die Elemente abzustecken, aus denen eventuell die Liste bestehen soll. Um einen Satz in Worte aufzuteilen, wird als Trennzeichen das Leerzeichen benötigt. Testen Sie Skript [22]. Ein guter Skriptstil verlangt, dass Sie eine Änderung von Trennzeichen [22.3] auch wieder rückgängig machen [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
```

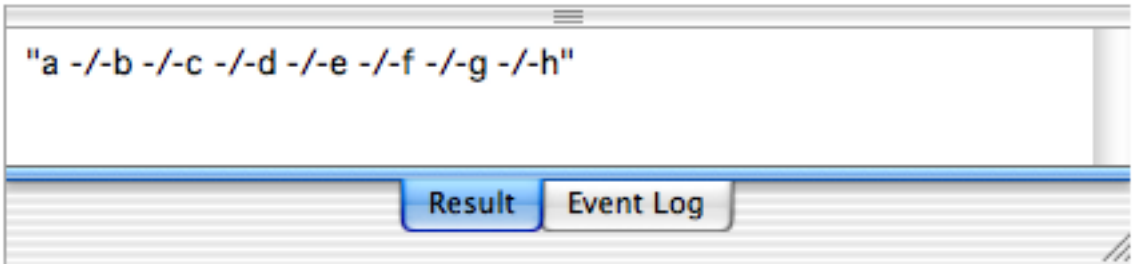
Sehen Sie sich bitte genau das Statement [22.4] an, wo es 'text item' heißt, nicht nur 'item'. Dies wäre ein Fehler, der oft gemacht wird. text item text item text item. Haben Sie es?

Es ist einfach, eine Liste in einen String umzuwandeln [23].

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

Wenn Sie ein bestimmtes Zeichen oder eine Reihe von Zeichen im String benötigen, welche die originalen Listeneinträge voneinander trennen, sollten Sie AppleScript's text item delimiters entsprechend setzen [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
```



Kombiniert man das Gelernte aus Skript [22] und Skript [24], haben Sie nun die Möglichkeit einen Suchen- und Ersetzen-Vorgang in einem Text durchzuführen.

#Der Grund, warum Sie die vorherigen AppleScript text item delimiters wieder herstellen sollten, liegt in der möglichen Unachtsamkeit anderer Scripter. Deren Skript könnte von einem anderen bestimmten Wert der AppleScript text item delimiters ausgehen. Eine Änderung durch Ihr Skript, wird von der AppleScript-Komponente von OS X jedoch auch beibehalten, nachdem das Skript beendet ist. #

Lassen Sie uns zusammenfassen: Es gibt verschiedene Datentypen wie Zahlen, Textwerte und Listen. Jeder Datentyp hat seine eigenen Operatoren, die Sie anwenden können, um Ihre Daten zu bearbeiten. Für die Handhabung von Listen gibt es eine ganze Menge Operatoren. In diesem Kapitel über Listen haben wir auch noch das Eine oder Andere über Textwerte gelernt, was wir vorher nicht behandeln konnten.

Kapitel 7

Datensätze (records)

Im vorigen Kapitel haben wir den 'display dialog'-Befehl als Einleitung zum Listen-Datentyp benutzt. Jetzt wollen wir ihn verwenden, um einen weiteren Datentyp vorzustellen.

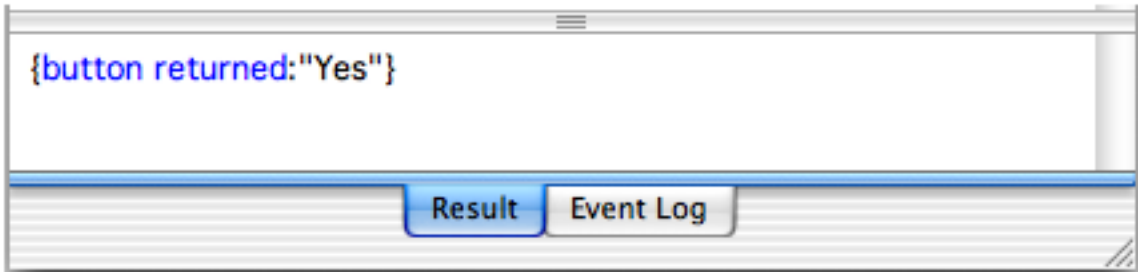
Der 'display dialog'-Befehl erlaubt es, eine Reihe von Buttons zu bestimmen, indem wir eine Liste mit bis zu drei Textwerten festlegen [1.2].

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

Aber wie wissen wir jetzt, welcher Button angeklickt wurde? Untersuchen Sie nun einmal das Skript [2].

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

Wenn Sie dieses Skript [2] ausführen, wird das Ergebnis in etwa wie nachfolgend abgebildet aussehen — natürlich abhängig davon, welchen Button Sie gedrückt haben:



Dieses Ergebnis stellt bereits einen weiteren Datentyp dar, genannt 'record' (Datensatz). Wie bei Listen gibt es auch hier geschweifte Klammern, aber im Gegensatz zu Listen besteht nun jedes Element aus zwei Teilen, die durch einen Doppelpunkt getrennt sind. Auf ein Element eines Datensatzes wird zugegriffen wie auf eine Property. Der erste Teil der Property ist ein Label (Etikett oder auch Name; hier 'button returned'), der zweite Teil enthält den aktuellen Wert dieser

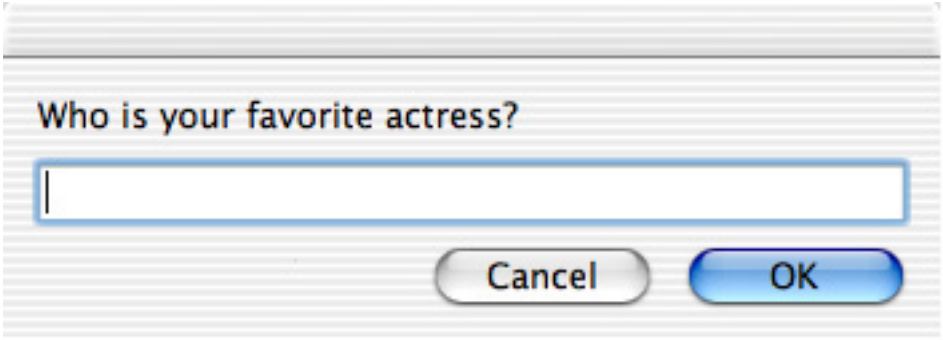
Property (hier den Textwert "Yes"). Da der zweite Teil ein Daten–Wert ist, wird er vom Skripteditor schwarz angezeigt.

Um nun heraus zu finden, welcher Button gedrückt wurde, brauchen wir nur den Wert der Property mit dem Namen 'button returned' abzufragen [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
```

Im Statement [3.3] legen wir fest, dass in der Variablen "theButtonPressed" der Wert der Property 'button returned' des Datensatzes 'tempVar' abgelegt werden soll. Das war's dann auch schon! Wir wissen nun, welcher Button des Dialoges gedrückt wurde.

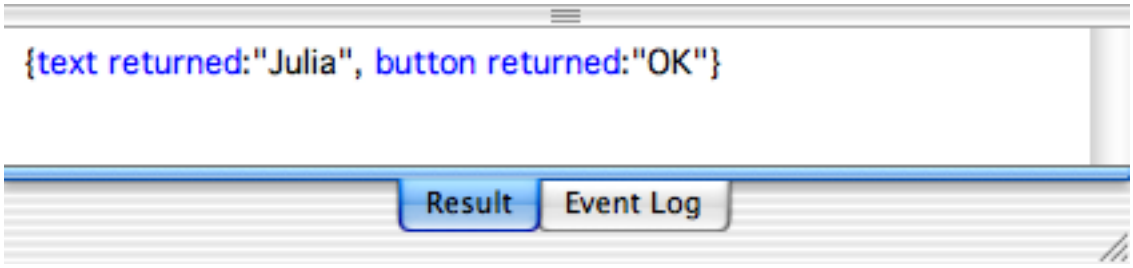
Die Möglichkeiten des Dialog-Fensters beschränken sich auf Zahlen und (kurze) Textwerte. Listen und Datensätze können damit nicht dargestellt werden. Dagegen kann der Ergebnis-Bereich alle Datentypen, die wir bis jetzt kennen gelernt haben, darstellen. Trotzdem bietet das Dialogfenster noch etwas sehr nützliches: Es ermöglicht dem Benutzer, Zahlen oder Text einzugeben, die dann vom Skript verarbeitet werden können.



Um solch ein Eingabefeld zu erhalten, müssen Sie eine Antwort-Vorgabe als Text, z.B. einen leeren Text "" [4], in den Dialog integrieren.

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

Wenn Sie Skript [4] laufen lassen, ist das Ergebnis ein Datensatz mit zwei Properties, d.h. zwei Name/Wert-Paare. Der entsprechende Datensatz sieht im Ergebnis-Bereich so aus:



Beachten Sie, dass die Properties (Eigenschaften) wie Elemente einer Liste mit Komma voneinander getrennt werden. AppleScript kann Datensätze aufgrund des Doppelpunktes nicht mit Listen verwechseln. Im Gegensatz zu einer Liste, bei der Sie sich merken müssen, an welcher Position welche Information steht, macht Ihnen die Tatsache, dass auf Daten eines Datensatzes zugegriffen werden kann, indem man den entsprechenden Namen referenziert, das Leben wesentlich einfacher. Um den Namen der Schauspielerin zu erhalten, müssen wir nur den Wert der Property 'text returned' abfragen [5.2].

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

# Beachten Sie bitte, dass der Wert von 'text returned' ein Textwert ist. Es ist stets ein Text, auch wenn der Benutzer eine Zahl eingibt. Gibt der Benutzer z.B. 30 ein, ist der Wert der Variablen textEntered nicht 30, sondern "30". Wenn Sie mit den eingegebenen Daten eine Berechnung anstellen wollen, haben Sie Glück. AppleScript wird versuchen, den Textwert automatisch in eine Zahl umzuwandeln [6.3]. Sie müssen also kein Statement zur Datentypumwandlung [7.1] nach [6.2] einfügen.
```

[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
```

Die Datentypumwandlung funktioniert, wenn der Benutzer eine Zahl, wie etwa 30, eingibt. Wird jedoch 'dreißig' oder '30 Jahre' eingegeben, ist AppleScript nicht in der Lage, die Umwandlung vorzunehmen und das Skript wird durch einen Fehler beendet. Da Sie vom Benutzer nicht exakt das erwarten können, was Sie beabsichtigt haben, werden Sie lernen müssen, Skripte zu schreiben, die verschiedene Verhaltensweisen des Benutzers in Betracht ziehen. In Kapitel 10 werden Sie lernen, wie man mit dieser Problematik umgeht.#

Sie können eigene Datensätze erstellen, indem Sie eine Variable mit einem Name/Wert-Paar versehen [8].

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

Beachten Sie, dass die Eigenschaft 'age' in grün angezeigt wird, d.h. sie wurde von Ihnen definiert. Wie üblich wird der Wert selbst in schwarz dargestellt.

#Bezüglich der Skripte [3] und [5] werden Sie sehen, dass die Etiketten der Eigenschaften 'button returned' und 'text returned' Besonderheiten von AppleScript sind. Sie erkennen das an ihrer blauen Farbe und ganz bestimmt auch daran, dass sie aus zwei Worten bestehen. Wenn Sie ihre eigenen Datensätze erstellen, ist es Ihnen nicht erlaubt, zwei oder mehrere Worte zu benutzen, um eine Eigenschaft zu bezeichnen [9.1]. Das Etikett (oder Name) einer Eigenschaft muss ein einzelnes Wort sein [9.2].

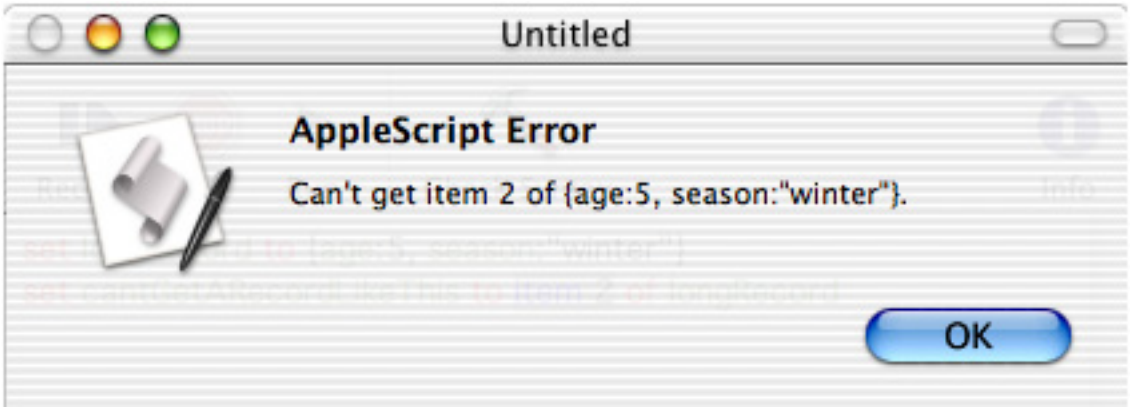
[9]
```
set improperlyNamedProperty to {my property: "This is not correct"}
set properlyNamedProperty to {myproperty:"This is correct"}
```

#

Sie können Datensätze wie Listen miteinander verknüpfen. Aber aufgepasst: Wenn die Datensätze gleichlautende Etiketten von Eigenschaften enthalten, kann das Ergebnis durchaus anders ausfallen als Sie erwarten.

Bitte versuchen Sie nicht, auf Eigenschaften eines Datensatzes mit 'item' zuzugreifen. Obwohl das folgende Skript [10] die Syntax-Prüfung übersteht, erlaubt es AppleScript nicht, sich per 'item' auf ein Name/Wert-Paar eines Datensatzes zu beziehen [10.2]

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



Sie können jedoch die Anzahl der Eigenschaften in einem Datensatz abfragen [11]:

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

Um einen neuen Datensatz zu erstellen, der eine Eigenschaft eines anderen Datensatzes enthält, müssen Sie wie folgt vorgehen [12].

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

Der Ergebnisbereich zeigt den neuen Datensatz als {age:5} an. Es ist möglich, das Skript [12] kürzer zu schreiben, aber Sie werden es vielleicht schwerer lesbar finden [13].

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

Unglücklicherweise ist es nicht möglich, die Name/Wert-Paare, die in einem Datensatz enthalten sind, heraus zu finden. D.h. Sie können keine Liste der Property-Etiketten erstellen. Entsprechend ist es auch nicht möglich, die Etiketten eines Datensatzes zu ändern. Wenn Sie ein anderes Etikett verwenden wollen, sollten Sie den Datensatz, wie in Skript [13] gezeigt, aufs Neue erstellen.

#Lassen Sie uns dieses Kapitel mit einer üblen Falle beenden. Hier ist ein Skript, das mit keinerlei Überraschungen aufwartet [14].

[14]
```
set firstValue to 30
set rememberFirstValue to firstValue -- A copy is made and stored by 'refToFirstValue'.
set firstValue to 73 --Change the value of original variable.
get rememberFirstValue -- We ask for the value of 'rememberFirstValue'.
```

Das Ergebnis ist 30. Bei Datensätzen (und Listen!) verhält es sich vollkommen unterschiedlich, was zu schwer auffindbaren Fehlern führen kann. Untersuchen Sie Skript [15].

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

Das Ergebnis lautet {age:73}!!! Der set-Befehl macht keine Kopie, wenn die Variable einen Datensatz oder eine Liste beinhaltet. Um sicherzustellen, dass die Daten kopiert werden, müssen sie den copy-Befehl verwenden [16].

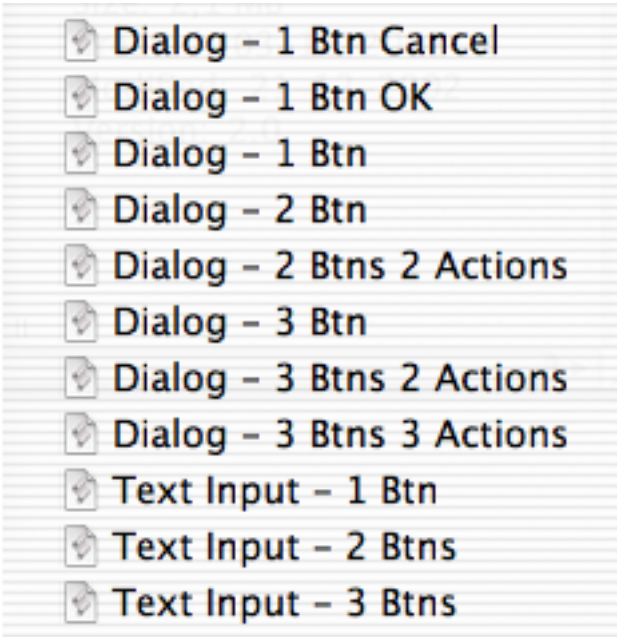
[16]
```
set personalData to {age:30}
copy personalData to rememberPersonalData
set age of personalData to 73
get rememberPersonalData
```

#

Kapitel 8

Leichteres Scripting (II)

Im vorigen Kapitel haben wir verschiedene Arten gesehen, Dialog-Fenster anzuzeigen. Wenn Sie das Gefühl haben, sich alle Varianten merken zu können – umso besser. Falls Sie eine einfache Methode bevorzugen, erinnern Sie sich, dass ein Control-Klick im oberen Bereich des Skript Editors ein Kontextmenü öffnet. Einer der Menüpunkte lautet "Dialogs". Klicken Sie darauf und Sie werden folgendes Untermenü zu sehen bekommen.



'Btn' steht hier für Button. Die jeweils voran gestellte Zahl repräsentiert die Anzahl der Buttons. Vergessen Sie vorerst mal die Menüeinträge, die das Wort 'Actions' enthalten. Sie werden diese verstehen, sobald wir Kapitel 10 erreicht haben.

Die letzten drei Menüeinträge im obigen Untermenü erlauben dem Benutzer, Text einzugeben. Schreiben Sie folgendes im Skript Editor:

```
set temp to
```

Stellen Sie sicher, dass nach dem 'to' ein Leerzeichen ist. Machen Sie nun nach dem Leerzeichen einen Control-Klick und wählen Sie 'Text Input – 2 Btns'. Hiermit wird das benötigte Statement erzeugt, das dem User ein Eingabefeld präsentiert, mittels dessen Daten an ein Skript übergeben werden können. Ähnliches haben wir in Kapitel 7 besprochen, als es um die Eigenschaft 'text returned' ging.

Wenn Sie wollen, kann display dialog eine vorgegebene Antwort anbieten, die der Benutzer, wenn nötig, durch etwas anderes ersetzen kann. Wenn eine bestimmte Antwort wahrscheinlich ist, wird Ihr Skript benutzerfreundlicher, wenn diese Antwort bereits als Vorgabe angeboten wird [1.1]. Sie wissen doch, Macintosh-User lieben Benutzerfreundlichkeit. Selbst wenn die Antwort nicht wahrscheinlich ist, gibt sie dem Benutzer doch einen Anhaltspunkt, welche Art von Antwort von ihm erwartet wird.

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

Wenn Sie Skript [1] ausführen, wird Ihnen der folgende Dialog angezeigt. Wenn Sie mit der Antwort einverstanden sind, brauchen Sie nur die Enter-Taste zu drücken. Andernfalls können Sie einen anderen Namen eingeben.

Kurz gesagt, es ist nicht nötig, dass Sie sich alle exakten Befehle für die diversen Erscheinungsformen von display dialog merken. Der Skripteditor macht das für Sie und Sie können sie in Ihr Skript einfügen, ohne etwas zu tippen.

Kapitel 9

Kein Kommentar? Inakzeptabel!

Verschiedene Faktoren sorgen dafür, dass AppleScripts leicht zu lesen, zu schreiben und zu pflegen sind. Einige liegen außerhalb Ihrer Kontrolle, wie z. B. die englisch-ähnliche Natur der Skriptsprache. Für andere jedoch sind Sie zuständig, wie etwa aussagekräftige Variablennamen. In diesem Kapitel werden wir einen weiteren wichtigen Faktor behandeln.

Während wir uns im Moment noch mit Beispielen beschäftigen, die nur ein paar Zeilen lang sind, werden Ihre künftigen AppleScripte immer länger und länger werden. Von größter Bedeutung beim Schreiben von Skripten ist nicht nur, dafür zu sorgen, dass Ihr Skript tatsächlich das tut, was Sie von ihm erwarten, sondern auch, dass es gut dokumentiert ist. Später, wenn Sie das Skript mal für eine Weile nicht gesehen haben und es bearbeiten wollen, werden Sie die Kommentare wirklich brauchen, um schnell zu verstehen, was ein bestimmter Skriptteil macht und warum der Teil gerade an dieser Stelle steht. Ich rate Ihnen dringendst, sich die Zeit zu nehmen und Ihr Skript zu kommentieren. Wir können Ihnen garantieren, dass Sie die darauf verwendete Zeit in der Zukunft vielfach wieder einsparen. Auch wenn Sie später das Skript an jemanden weitergeben, wird derjenige in der Lage sein, es schnell anzupassen, wenn Sie es gut kommentiert haben.

Um einen Kommentar zu erstellen, beginnt man mit zwei Bindestrichen.

```
-- This is a comment
```

Nach der Kompilierung (Syntax-Check), wird der Kommentar in Grau dargestellt.

```
-- This is a comment
-- This is a comment spanning more than
      one line
```

In früheren Tagen wurden mehrzeilige Kommentare zwischen (* *) gesetzt.

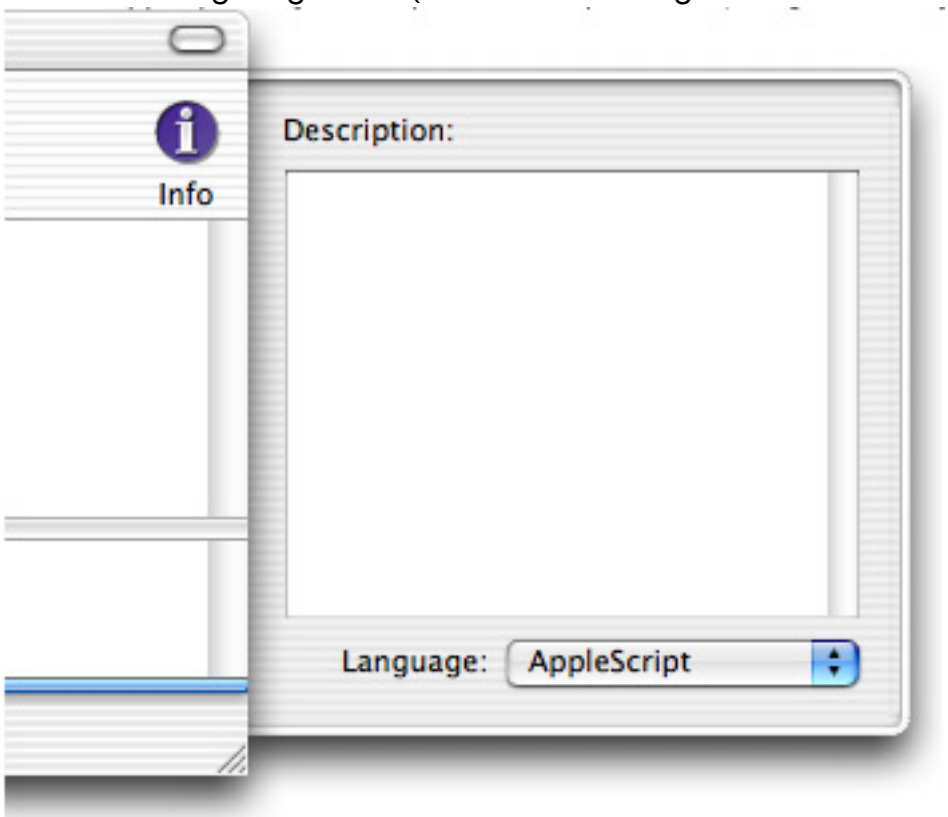
```
(* This is a comment
extending over two lines.*)
```

Seitdem das Kontextmenü verfügbar ist, empfiehlt sich dies jedoch nicht länger. Sie sollten bei den zwei Bindestrichen bleiben. Der Grund dafür ist eine Praxis, die man Auskommentierung nennt.

Indem Sie einen Teil Ihres Skripts zwischen (* *) platzieren, können Sie vorübergehend Teile des Skripts deaktivieren ('auskommentieren'), um zu sehen, ob der Rest wie erwartet funktioniert. Dadurch können Sie Fehler ausmerzen. Falls der auskommentierte Teil z.B. einen Wert für eine bestimmte Variable liefern sollte, können Sie vorübergehend eine zusätzliche Zeile einfügen, wo Sie der Variablen einen passenden Wert zuweisen, um den Rest des Skripts zu testen.

Das im Skripteditor verfügbare Kontextmenü erlaubt Ihnen ein leichtes Auskommentieren. Wenn Sie Skript-Teil auskommentieren wollen, wählen Sie den entsprechenden Teil z.B. durch Ziehen mit der Maus aus. Dann machen Sie einen Control-Klick in den ausgewählten Teil und wählen 'Comment Tags'. Es wird Ihnen nun angeboten, Kommentarzeichen zum ausgewählten Text hinzuzufügen oder zu entfernen. Klicken Sie auf 'Add' und die Sache ist erledigt. Um die Kommentarzeichen zu entfernen, müssen sich diese im ausgewählten Skriptbereich befinden. Die Auswahl muss dabei nicht ganz exakt sein. Wählen Sie dann 'Remove'. Wenn Ihre Kommentare und Erklärungen nach zwei Bindestrichen platziert waren, werden sie diese Prozedur überstehen.

Klicken Sie im Skripteditor auf den Info-Button, um eine Slidebar erscheinen zu lassen (Bild unten), wo Sie den Verwendungszweck des Skripts allgemein beschreiben können. Zusätzlich oder alternativ dazu sollten Sie den oberen Teil des Skripts mit einer Erklärung beginnen (unter Benutzung der beiden Bindestriche).



Meine Darstellung der Wichtigkeit von Kommentaren ist wirklich nicht übertrieben. Die Skripte dieses Buchs enthalten nicht so viele Kommentare, wie wir gewöhnlich geschrieben hätten, weil sie ja bereits von Erklärungen umgeben sind.

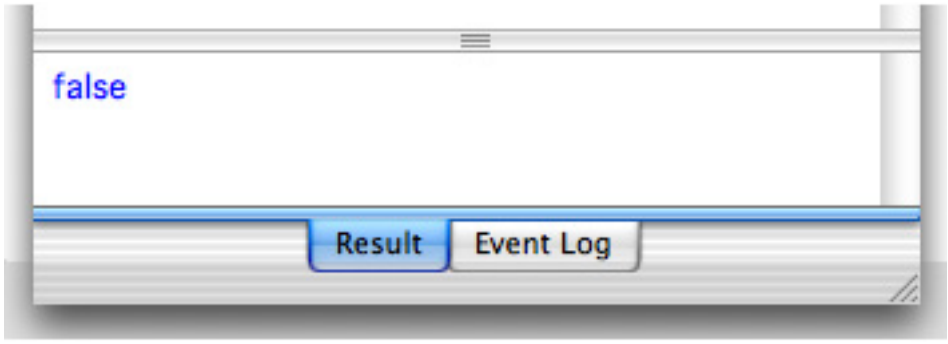
Kapitel 10

Bedingte Anweisungen

Hin und wieder werden Sie wollen, dass Ihr Skript eine Serie von Anweisungen nur ausführt, wenn eine bestimmte Bedingung erfüllt ist. Tippen Sie die folgende Zeile [1] in den Skripteditor und klicken Sie auf "Run".

[1]
73 = 30

Sie werden dies im Ergebnisbereich zu sehen bekommen:



AppleScript wertet den Vergleich in Skript [1] aus und kommt zu dem Schluss, dass es falsch ist, d.h. nicht wahr. Wenn Sie etwas wie '30 = 30' eingeben, werden Sie das Ergebnis 'true' sehen.

Es ist diese Fähigkeit von AppleScript, zwei Werte zu vergleichen, die im 'if...then' Statement benutzt wird, um Anweisungen nur auszuführen, wenn eine Bedingung zutrifft. Das 'if...then' Statement wird eine bedingte Anweisung genannt. Beachten Sie bitte, dass dieses Statement eine 'end if' Anweisung benötigt. Mehr dazu später.

[2]
```
if true then
	-- actions performed
end if
if false then
	-- these actions are not performed
end if
```

Nun ersetzen wir das Wort 'true' im Statement [2.1] durch einen Vergleich. Falls der Vergleich wahr ('true') ergibt, werden die Anweisungen in Zeile [2.2] ausgeführt.

[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
```

Wäre der Vergleich 'ageEntered is myAge' [3.3] zutreffend gewesen, wäre das Dialog-Fenster erschienen. Da '73' aber nicht gleich '30' ist, werden Sie das Dialog–Fenstr [3.4] nicht sehen.
Falls mehrere Befehle ausgeführt werden sollen, wenn die Bedingung zutrifft, müssen diese alle im 'if...then...end if' Block enthalten sein [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."
```

Sie werden die Ähnlichkeit zwischen einem Tell-Block und dem 'if...then' Statement bereits bemerkt haben. In beiden Fällen wird eine Anweisung benötigt, die das Wort 'end' enthält. Durch 'end if' kann AppleScript erkennen, welches die auszuführenden Anweisungen sind, falls der Vergleich 'true' ergibt. In Skript [4] werden Sie das letzte Statement [4.7] hören, ungeachtet dessen, ob die Bedingung [4.3] zutrifft oder nicht.

Wir können auch einen alternativen Befehlssatz bereit stellen für den Fall, dass die Bedingung nicht zutrifft. Dies geschieht durch die Benutzung eines 'if...then...else' Statements [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
```

Hier wird der Dialog aus Statement [5.6] angezeigt, da der Vergleich in Statement [5.3] 'false' ergibt. Sie können noch viele weitere Vergleiche anstellen. Hier sind die Operatoren, die zum Vergleichen der jeweiligen Datentypen zur Verfügung stehen.

Abgesehen vom Gleichheitszeichen in Statement [5.3], stehen Ihnen für Zahlen die folgenden Operatoren zur Verfügung:

Für Zahlen:

=	is (oder 'is equal to')
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to

Die zweite Spalte zeigt nicht bloß die Erklärungen der Symbole der ersten Spalte an, sondern kann ebenso in Skripten verwendet werden [6].

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

Wenn Sie >= eintippen und die Syntax-Prüfung durchführen, konvertiert AppleScript diesen Vergleichsoperator zum offiziellen Symbol ≥. Ebenso wird das Kleiner-Gleich-Zeichen <= durch ≤ ersetzt. Sie müssen die Zeichen in der richtigen Reihenfolge schreiben, sonst wird sich AppleScript beklagen. Es ist halt doch nicht immer so benutzerfreundlich, wie es eigentlich sein könnte.

In Kapitel 7 stießen wir auf das folgende Skript [7], das uns erlaubte herauszufinden, welcher Button gedrückt wurde.

[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
```

Durch das 'if...then' Statement können wir nun eine gezielte Aktion ausführen [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
```

Wenn der Benutzer den Button 'Yes' drückt, wird der Satz aus Statement [8.5] gesprochen. In jedem Fall fährt das Skript mit dem Warnton im letzten Statement fort.

Im vorherigen Skript mussten die Textwerte identisch sein. Jedoch kann AppleScript wesentlich mehr als das. Hier sind vier Beispiele [9]. Sie werden bemerken, dass 'end if' nicht benötigt wird, wenn die auszuführenden Befehle in der selben Zeile wie das 'if...then' Statement stehen.

[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
```

Unten sehen Sie eine Übersicht der Vergleichsoperatoren für Textwerte.

Für Textwerte:
begins with (oder 'starts with')
ends with
is equal to
comes before
comes after
is in
contains

Sie können die Sache auch negativ abgrenzen:
does not start with
does not contain
is not in
etc.

Wenn Sie 'doesn't' schreiben, wird dies automatisch zu 'does not' neu formatiert. 'does not begin with' wird automatisch zu 'does not start with'.

Die 'comes before' und 'comes after' Vergleichsoperatoren funktionieren alphabetisch. So wird das folgende Statement zum Ertönen des Warntones führen [10].

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

```
# Beim Vergleichen von Textwerten mag die Großschreibung für Sie eine Rolle spielen. Sie brauchen dies AppleScript nur zu signalisieren. Natürlich müssen Sie dies hinterher auch wieder abschalten [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
```

Defaultmäßig berücksichtigt AppleScript auch unsichtbare Zeichen (Leerzeichen, Zeilenschaltung, Tabulator). Wenn Sie das nicht wollen, tippen Sie 'ignoring white space', wie in Skript [12] demonstriert. Denken Sie bitte daran, dass Sie ein 'end ignoring' oder 'end considering' Statement einfügen müssen.

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

Sie können AppleScript auch anweisen, Satzzeichen oder diakritische Zeichen zu ignorieren. #

Für den Datentyp Liste stehen uns weniger Vergleichsoperatoren zur Verfügung als für Textwerte. Andererseits sind dies die gleichen. Sie müssen also nicht mehr Operatoren lernen als unbedingt nötig.

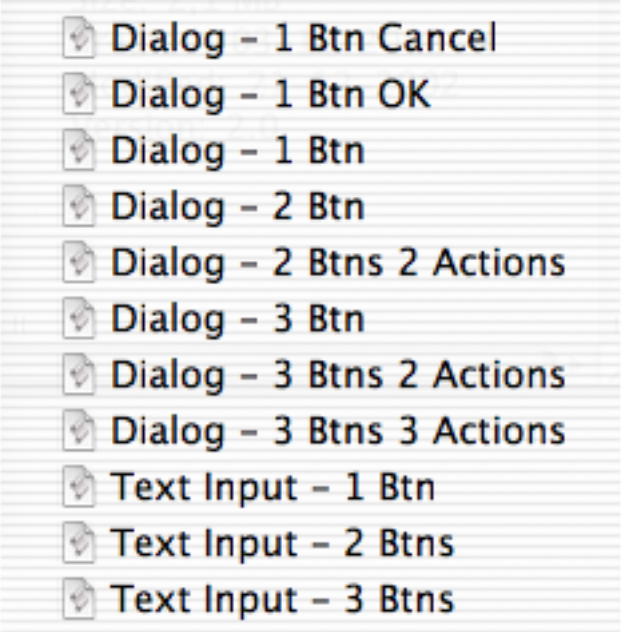
Für Listen:
begins with
ends with
contains
is equal to
is in

Sie werden oft einzelne Elemente einer Liste (oder verschiedener Listen) vergleichen. Lassen Sie uns ein praktisches Beispiel betrachten [8]. Was ist, wenn wir drei Buttons in unserem Dialog haben? Durch Verschachtelung der 'if...then' Statements [13] werden alle Optionen abgedeckt.

[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
```

Wie Sie sehen, erleichtern die Einrückungen die Lesbarkeit des Skripts. Dennoch ist es etwas heikel zu lesen. Und es ist viel zu tippen und Sie können dabei leicht ein 'end if' Statement falsch setzen. Als Folge davon wird das Skript nicht kompiliert werden. Die Rettung naht in Form des Kontextmenüs von Skripteditor. Erinnern Sie sich an die Menüeinträge mit der Bezeichnung Actions?



Klicken Sie auf denjenigen für 3 Buttons, 3 Actions und es wird Ihnen der volle, benötigte Befehlssatz präsentiert. Sie müssen nur noch die Werte der Listen [14.1] ausfüllen und die gewünschten Aktionen eintippen.

[14]
```
display dialog "" buttons {"", "", ""} default button 3
set the button_pressed to the button returned of the result
if the button_pressed is "" then
	-- action for 1st button goes here
else if the button_pressed is "" then
	-- action for 2nd button goes here
else
	-- action for 3rd button goes here
end if
```

Die ersten beiden Zeilen bedürfen einer Erklärung. Wie Sie sehen können, unterscheidet sich Statement [15.1] von Statement [8.2] darin, dass wir keine Variable mit dem Ergebnis des display dialog Befehls füllen. Genau wie der Ergebnisbereich automatisch das Ergebnis des jeweils zuletzt ausgeführten Statements anzeigt, können Sie sich auf das Ergebnis des vorangegangenen Statements beziehen, indem Sie die reservierte Variable 'result' benutzen. Dadurch, dass wir das if-Statement in die selbe Zeile [14.5] wie 'else' schreiben, ist ein gesondertes 'end if' nicht nötig. Alles in allem wird das Skript leichter lesbar. #

Für Datensätze ist die Anzahl der Vergleichsoperatoren noch geringer als für Listen.

<u>Für Datensätze</u>
contains
is equal to -- or =

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

Ist Ihnen aufgefallen, dass im Statment [15.1] ein Etikett in Blau und das andere in Grün dargestellt wird? Sie werden dadurch gewarnt, dass eines der Etiketten ein reserviertes Wort ist. Obwohl Sie AppleScript nicht davon abhält, als Eigenschafts-Etiketten reservierte Schlüsselwörter, wie 'to', 'set', 'string' etc. zu benutzen, könnte Sie das gelegentlich in Schwierigkeiten bringen. Vermeiden Sie also reservierte Wörter, indem Sie die Tipps für Variablennamen in Kapitel 4 beherzigen.

Die beschränkte Anzahl an Vergleichsoperatoren für Datensätze stellt in der Regel kein Problem dar, da gewöhnlich nicht komplette Datensätze, sondern die einzelnen Werte eines Datensatzes verglichen werden.

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

Wenn Sie zwei Werte (egal welchen Datentyps) vergleichen, wie wir es im ersten Teil dieses Kapitels gesehen haben, versuchen Sie in Wirklichkeit, heraus zu finden, ob ein Vergleich true oder false ergibt. Tatsächlich repräsentieren diese jedoch einen speziellen Datentyp, genannt 'Boolean'. Variablen dieses Typs können nur einen der beiden Werte true und false beinhalten. Für Zahlen haben wir Operatoren wie '+' und '-'. Wenn Sie einen solchen Operator benutzen, ist das Ergebnis eine Zahl. Für Booleans haben wir die Operatoren 'and', 'or' und 'not'. Das Ergebnis einer solchen Operation ist wiederum ein Boolean.

'not' ist von den dreien am einfachsten. 'not true' ist false und 'not false' ist true.

[17]
```
set x to not true -- So, x is false
```

Wie im folgenden Skript [18] für den 'and' Operator gezeigt, müssen beide Variablen x und y den Wert true haben, damit z ebenfalls true ist.

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

Dagegen genügt beim 'or' Operator, dass ein Wert, x oder y true ergibt [19].
[19]
```
set x to true
set y to false
set z to (x or y) -- z is true
```

Warum erzähle ich Ihnen all dies? Nun, unter Umständen möchten Sie eine Reihe von Anweisungen nur ausführen lassen, wenn mehrere Bedingungen erfüllt sind. Im folgenden Skript [20] wird nur der Dialog des letzten Statements angezeigt werden.

[20]
```
set x to 5
set y to 7
set z to "Julia"
if x = 5 and y = 6 then display dialog "Beide Bedingungen treffen zu."
if x = 5 or z = "actress" then display dialog "Zumindest eine Bedingung trifft zu."
```

Im Statement [20.4] ergibt der erste Vergleich true, der zweite jedoch nicht. Da 'and' erfordert, dass beide Vergleiche zutreffen, wird der Dialog nicht angezeigt. In Statement [20.5] gibt sich 'or' bereits zufrieden, wenn eine Bedingung erfüllt ist.

In Skript-Statements wie [20.4, 20.5] sollten Sie runde Klammern benutzen, da dies die Lesbarkeit erleichtert (und möglicherweise die Zuverlässigkeit erhöht).

Kapitel 11

Versuch ohne Fehlschlag

In allen, bisher besprochenen Skripten wird das jeweilige Skript abgebrochen, sobald AppleScript bei der Ausführung auf ein Problem stößt.

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

Wenn Sie die Syntax-Prüfung für obiges Skript [1] durchführen, wird AppleScript keine Probleme melden. Falls Sie das Skript jedoch ausführen, werden Sie den gesprochenen Satz [1.3] nicht hören, da die weitere Ausführung des Skripts bereits in Statement [1.2] gestoppt wird, obwohl danach noch das gültige Statement [1.3] steht.

Natürlich ist der Abbruch eines Skripts nicht unbedingt das, was Sie wollen. Wenn z. B. Ihr Skript das Vorhandensein eines bestimmten Ordners mit Dateien, die verarbeitet werden sollen, voraussetzt, dieser Ordner jedoch gelöscht wurde, könnten Sie dem Benutzer die Alternative anbieten, einen anderen geeigneten Ordner auszuwählen.

Beim Schreiben eines Skripts müssen Sie die Statements erkennen, die anfällig für Probleme während der Ausführung sind. Schließen Sie diese Statements in eine try ... end try Block-Anweisung ein, wie hier [2] demonstriert.

[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"
```

Wenn Sie nun das Skript ausführen, werden Sie den zweiten Satz [2.6] hören, da AppleScript nach der 'end try' Anweisung [2.4] fortfährt.

In Kapitel 7, das sich mit Datensätzen befasst, stießen wir auf das folgende Skript [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
```

Das Problem an diesem Skript ist, dass es nicht mehr funktioniert, sobald jemand etwas anderes als eine Zahl eingibt. Wir können den Abbruch des Skripts verhindern und dem Benutzer eine nützliche Rückmeldung geben.

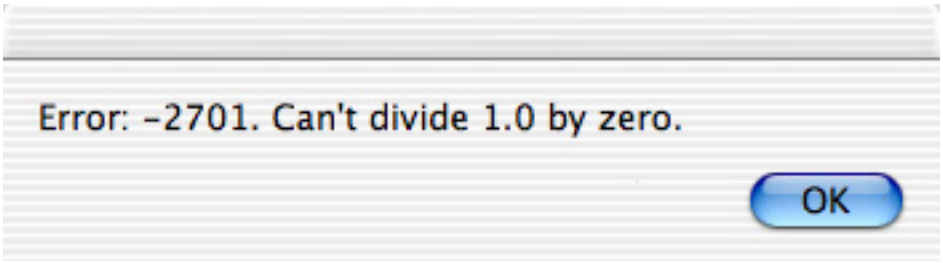
[4]
```
set temp to display dialog "What is your age in years?" default answer ""
set ageEntered to text returned of temp
try
	-- First we check if the user entered a number
	set ageEntered to ageEntered as number
	set ageInMonths to ageEntered * 12
	display dialog "Your age in months is " & ageInMonths
on error
	-- If it is not a number, the entry must have been text."
	display dialog "Instead of a number, like 30 , you entered text."
end try
```

Wenn jetzt der User keine Zahl eingibt, bekommt er eine entsprechende Rückmeldung. Lästig ist jedoch, dass der Benutzer das Skript erneut starten muss. In Kapitel 13 werden wir dieses Problem beheben.

Der Script Editor erleichtert es Ihnen, try-Blocks einzufügen. Wählen Sie einfach das oder die Statements, welche innerhalb des try-Blocks erscheinen sollen und benutzen Sie das Kontextmenü, um den try-Block Ihrer Wahl zu selektieren. Wie derjenige, der in Skript [17] benutzt wird.

[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
```

Wenn Sie nach 'on error' einen Variablennamen verwenden, wird die Fehlerbeschreibung in dieser Variablen gespeichert. Stellen Sie einer Variablen 'number' voraus, wird die entsprechende Fehlernummer in dieser Variablen abgelegt. In Statement [17.3] haben wir beides. Der entsprechende Dialog sieht so aus:

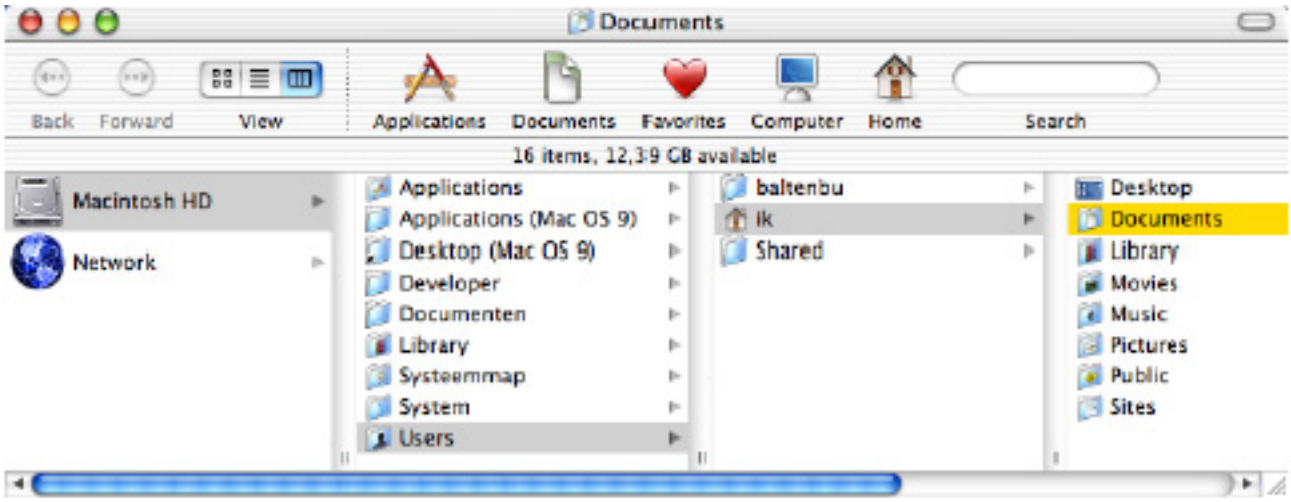


#

Kapitel 12

Pfade zu Dateien, Ordnern und Programmen

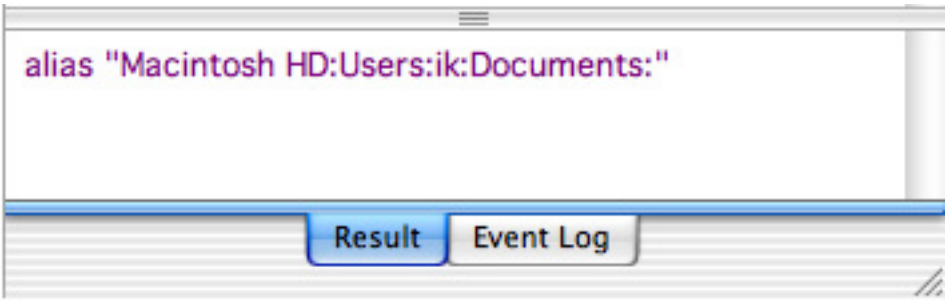
Lassen Sie uns einmal betrachten, wie Ordner und Dateien auf Ihrer Festplatte organisiert sind. Wenn Sie ein Finder-Fenster in der Spaltendarstellung öffnen, sieht es etwa so aus:



Hier haben wir eine Festplatte, die Ordner, Programme und Dateien (das obige Bild zeigt keine Dateien und Programme) enthält. All diese Elemente sind hierarchisch organisiert. Dies erlaubt es uns, den Ablageort einer Datei (oder eines Ordners oder Programms) mit Hilfe einer Pfadangabe zu definieren. Wollen wir einmal sehen, wie so ein Pfad aussieht [1].

[1]
```
choose folder
```

Hier sehen Sie die Anzeige des Ergebnisfelds, wenn ich meinen Dokumente-Ordner auswähle:



Sie sehen eine Pfadangabe, die grundsätzlich so aussieht:

Festplatte:Ordnername:Unterordnername:Unterordnername:

Verfolgen Sie den Pfad im obigen Bild (Screenshot des Finder-Fensters) bis zu 'Documents'. Wenden Sie dann Ihre Aufmerksamkeit dem Bild des Ergebnisbereichs zu. Beachten Sie bitte, dass Doppelpunkte in Pfadangaben als Trennzeichen fungieren. Deshalb sind sie auch nicht in Datei- oder Ordnernamen erlaubt. Sie können dies ausprobieren, indem Sie einen neuen Ordner auf dem Desktop anlegen und versuchen, im Ordnernamen einen Doppelpunkt zu verwenden. Letztendlich können Sie noch sehen, dass die Pfadangabe mit einem Doppelpunkt endet, was uns anzeigt, dass sich der Pfad auf einen Ordner bezieht.

Wir können den Pfad dazu benutzen, dem Finder die Anweisung zu geben, den Ordner zu öffnen [2].

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

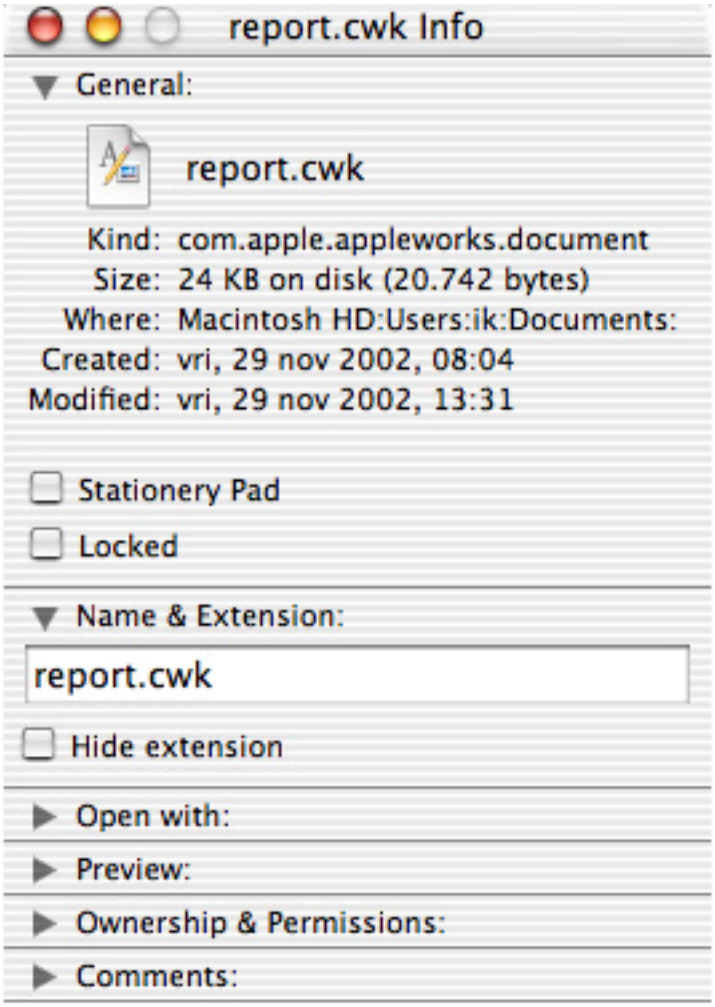
Wie Sie bei der Ausführung von Skript [2] (vergessen Sie nicht, den Benutzernamen und evtl. den Festplattennamen vorher anzupassen) sehen können, ist AppleScript recht nachsichtig und beschwert sich nicht, wenn Sie den Doppelpunkt am Ende der Pfadangabe vergessen. Beachten Sie jedoch, dass die Groß-/Kleinschreibung beim Festplattennamen exakt stimmen muss. Ts, ts, Apple!

Mein Dokumente-Ordner enthält eine AppleWorks-Datei mit dem Namen "report". Wollen wir einmal diese Datei öffnen [3].

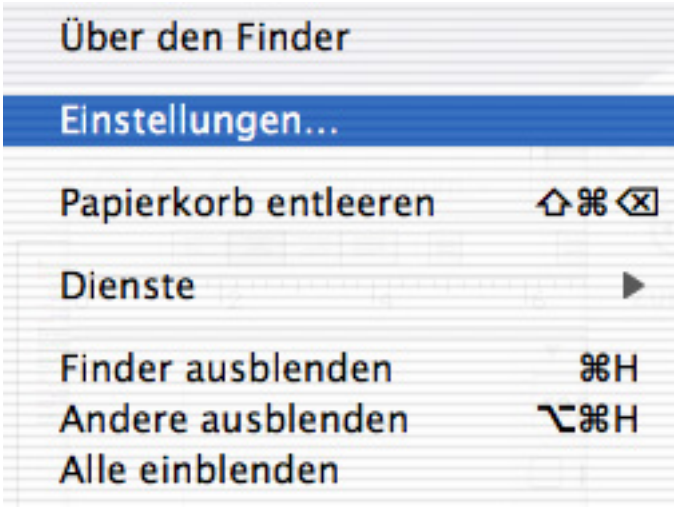
[3]
```
tell application "Finder"
	open file "Macintosh HD:users:ik:Documents:report.cwk"
end tell
```

Als erstes gilt zu beachten, dass das Statement [3.2] einen 'file' und nicht 'folder' angibt, da wir uns auf eine Datei beziehen. Als zweites muss beachtet werden, dass am Ende der Pfadangabe die Dateiendung angegeben werden muss. Hier die Endung 'cwk', was sich vom ursprünglichen Namen von AppleWorks ableitet, der Clarisworks lautete.

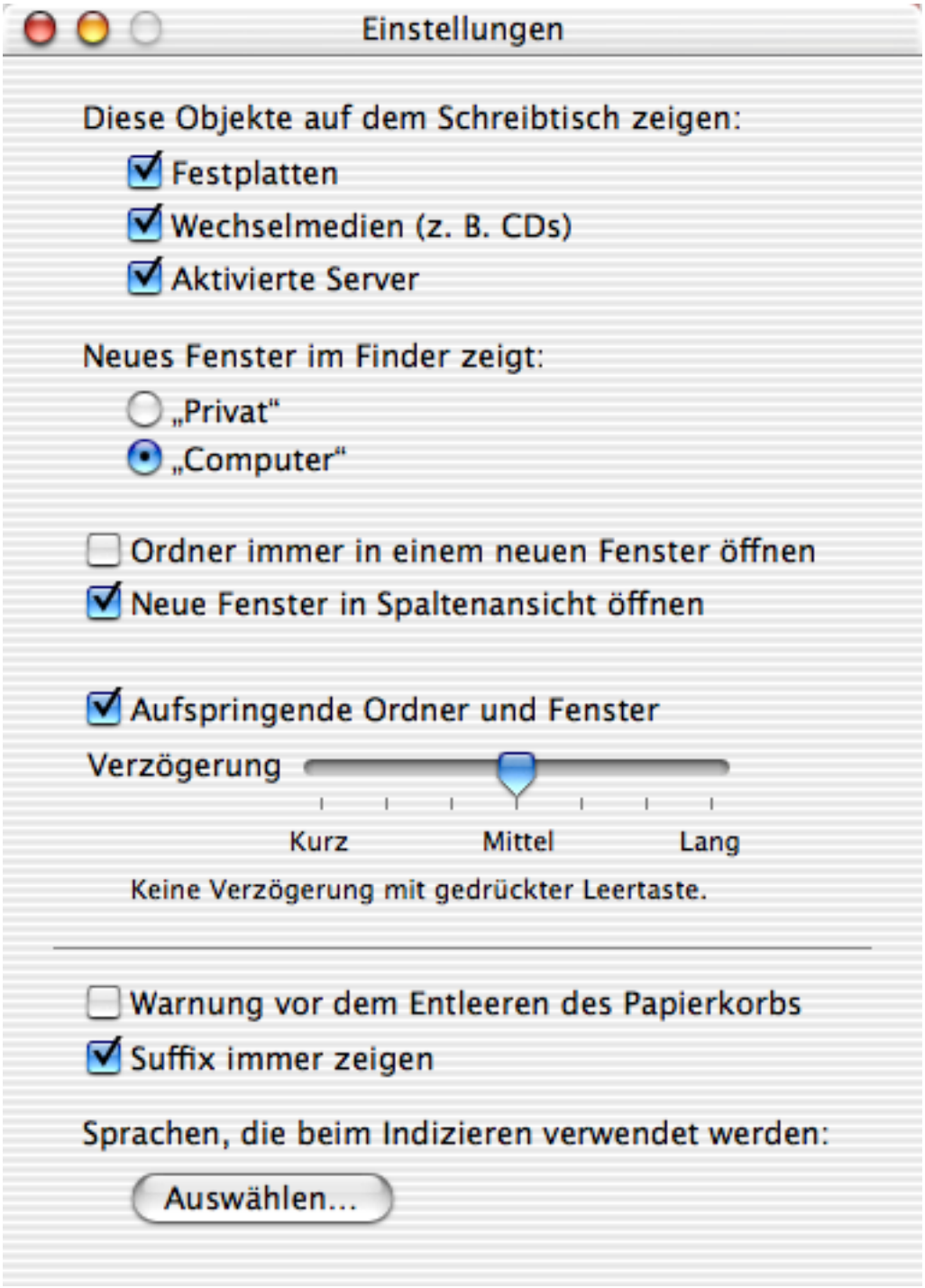
In Mac OS X ist die Dateiendung defaultmäßig ausgeblendet. Sie können die Dateiendung herausfinden, indem Sie die Datei auswählen und Befehlstaste-i drücken, um Informationen über diese Datei zu erhalten.



Stattdessen können Sie sich aber auch dafür entscheiden, die Dateiendungen stets sichtbar zu machen. Dies geschieht, indem Sie die Finder-Einstellungen entsprechend treffen. Klicken Sie auf den Menüpunkt 'Finder' in der Menüleiste des Finders. Es öffnet sich das Menü. Wählen Sie 'Einstellungen...'.



Im erscheinenden Fenster setzen Sie nun das Häkchen beim untersten Auswahlpunkt.



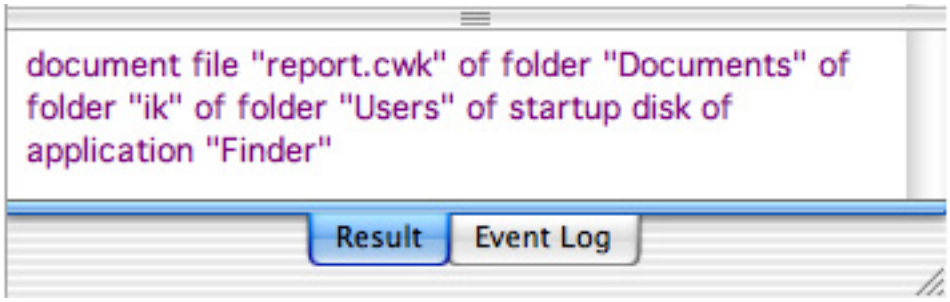
#

Wenn Sie die Pfadangabe zur Datei 'report.cwk' in einer Variablen speichern wollen, werden Sie geneigt sein, dies so [4] zu tun.

[4]

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

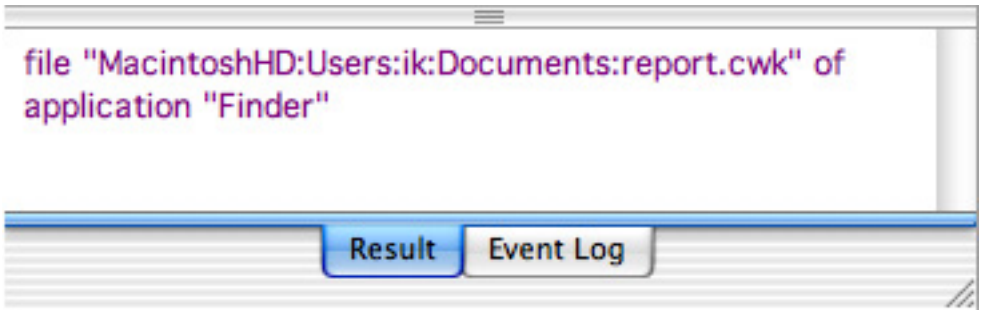
Dies ergibt jedoch nicht das Pfadformat, das wir oben gesehen haben. Es sieht nun aus wie:



Dieser Stil ist etwas unangenehm zu lesen, speziell bei längeren Pfaden und am schlimmsten ist, dass es ein Format ist, das so nur vom Finder erkannt wird. Sie werden wohl die abstraktere Schreibweise einer Pfadangabe mit Doppelpunkten bevorzugen oder üblicherweise auch benötigen. Sie können den Finder dazu zwingen, diese Schreibweise zu liefern, indem Sie den 'a reference to'-Befehl [5] verwenden.

[5]

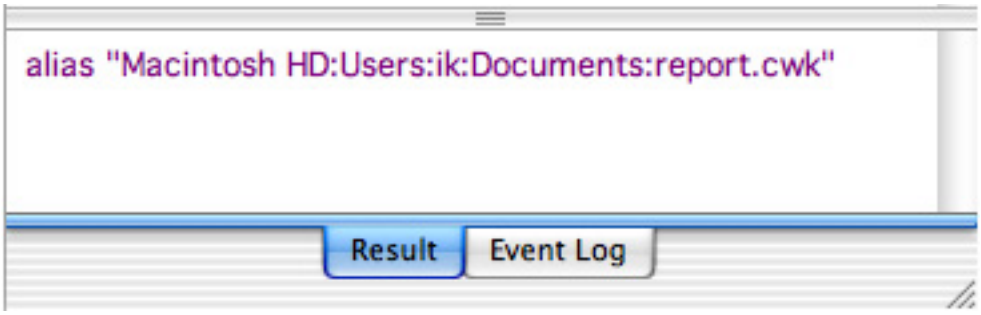
```
tell application "Finder"
	set thePath to a reference to file "Macintosh HD:Users:
                                        ik:Documents:report.cwk"
end tell
```



Beachten Sie, dass der Ergebnisbereich aussagt, dass es sich hier um eine Datei (file) handelt. Führen Sie nun bitte das unten stehende Skript [6] aus und wählen Sie die selbe Datei 'report.cwk'.

[6]
choose file

Der Ergebnisbereich sieht nun so aus:



Es heißt nun 'alias' statt 'file'! Um den Unterschied zu erklären, den das für AppleScript bedeutet, lassen Sie uns erst mal ein Alias besprechen, wie Sie es von Ihrer Arbeit im Finder gewohnt sind.

Angenommen ich habe auf meinem Schreibtisch ein Alias der Datei 'report.cwk' erstellt, die sich in meinem Dokumente-Ordner befindet. Nun, wenn ich die Datei 'report.cwk' aus dem Dokumente-Ordner heraus in einen anderen Ordner bewege, wird durch einen Doppelklich auf das Alias die Original-Datei immer noch geöffnet. Großartig! Ich kann sogar die Original-Datei 'report.cwk' in etwas wie 'funny_story.cwk' umbenennen. Das funktioniert, weil das Alias nicht den Ablageort (und den Namen) der Datei 'report.cwk' in der Form

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

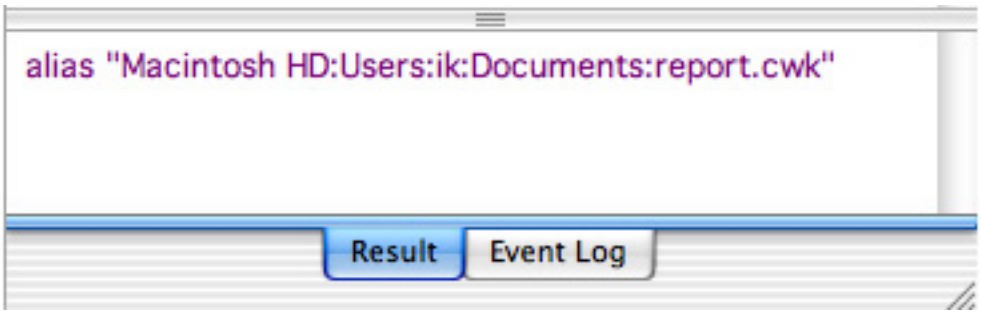
speichert, sondern eher eine Art ID-Nummer. Der Finder unterhält zu diesem Zweck eine Datenbank dieser IDs zusammen mit den aktuellen Ablageorten der entsprechenden Dateien (und Ordner und Programme). Wenn ich also die Datei 'report.cwk' bewege, bleibt deren einzigartige ID trotzdem gleich. Der Finder ändert nur die Einträge der internen Datenbank entsprechend dem neuen Ablageort. Wird nun ein Doppelklick auf das Alias ausgeführt, benutzt der Finder die dort enthaltene ID, um die zugehörige Datei zu finden und wird dann die richtige Datei öffnen.

Um ein Skript zu erhalten, das nicht abbricht, wenn eine Datei (oder ein Ordner) entfernt oder umbenannt wurde, sollte unser Skript anstatt einer festen Pfadangabe die ID der Datei (des Ordners) beinhalten. AppleScript ermöglicht auch dies [7].

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

Es ist sehr wichtig, darauf hinzuweisen, dass sich das Statement aus Skript [7] auf die originale Datei innerhalb des Dokumente-Ordners bezieht und nicht etwa auf eine vom Benutzer erstellte Alias-Datei, wie diejenige auf meinem Schreibtisch. In dem Statement von Skript [7] ist 'alias' ein Schlüsselwort, das uns nach der Kompilierung (d.h. Syntax-Prüfung) anzeigt, dass das Skript die ID der Datei benutzt. Der Finder sucht bei der Ausführung des Skripts deshalb nicht nach einer Datei, deren Ablageort ihm durch die Pfadangabe mitgeteilt wird, sondern vollzieht seinen Suchvorgang mit Hilfe der Datei–ID.

Wenn Sie Skript [7] ausführen und das Ergebnis im Ergebnisbereich überprüfen, sieht es so aus:



Sie werden also die ID selbst nicht sehen. Dennoch sagt uns das Wort 'alias' vor der Pfadangabe, dass das Skript intern mit der ID arbeitet und nicht mit einem festgelegten Pfad. Verschieben Sie die Datei 'report.cwk' nachdem das Skript kompiliert ist an einen anderen Ablageort und starten Sie das Skript erneut. Ich habe die Datei in einen Ordner mit dem Namen 'Miscellaneous' innerhalb meines Dokumente-Ordners gelegt. Obwohl das Skript und im Besonderen der Pfad im Skript [7] nicht geändert wurde, ist das Ergebnis nun

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

Probieren Sie es selbst (nochmal: Ihre Pfadangabe sieht unterschiedlich aus, da Ihr Login-Name wohl kaum 'ik' lautet und Sie eventuell auch andere Ordnernamen benutzen)! Wenn Sie das Skript als ein kompiliertes Skript oder ein AppleScript-Programm gesichert haben, wird die ID gespeichert, und beim nächsten Aufruf des Skripts wird es fehlerfrei laufen, auch wenn die Original-Datei verschoben oder umbenannt worden ist. Es sollte Sie allerdings nicht erstaunen, wenn ein Fehler auftritt, falls Sie die Datei gelöscht haben.

Zusammenfassend kann man sagen, dass es zwei Wege gibt, mit Pfaden in Skripten umzugehen. Sie können entweder den Ablageort einer Datei angeben (feste Pfadangabe per file "hier Pfad") oder Sie benutzen das Schlüsselwort 'alias', um Ihr Skript nach der Kompilierug, gegenüber Verschieben und Umbenennen der Datei/des Ordners/des Programms unempfindlich zu machen.

Obwohl die Verwendung von 'alias' oft den besten Weg darstellt, muss sich die Datei, auf die Sie sich beziehen, beim Kompilieren am angegebenen Ablageort sein. Auch können Sie das Skript nicht einfach als AppleScript-Programm an jemanden weitergeben, da selbst wenn eine Datei mit dem selben Namen am angegebenen Ort auf dem Computer dieser Person existiert, diese jedoch eine andere ID hat.

Nur ein Hinweis zu Skript [7]. Ich war persönlich überrascht, dass dies ohne einen Finder-Tell-Block funktioniert. Schließlich wird die interne Datenbank, die IDs und Ablageorte enthält, vom Finder verwaltet. Da der Finder das einzige Programm ist, das diese Informationen liefern kann und die AppleScript-Komponente von OS X dies weiß, fragt AppleScript den Finder anscheinend im Hintergrund nach der ID oder durchsucht die Datenbank direkt. Dieses Verhalten ist eine absolute Ausnahme. Zum Beispiel benötigt das Öffnen einer Datei mit Hilfe einer Pfadangabe einen Tell-Block (siehe Skript [3]). Der Grund dafür ist, dass der Finder nicht das einzige Programm ist, das eine bestimmte Datei öffnen kann, sondern das Erstellungsprogramm dieser Datei und möglicherweise auch ein anderes Programm dazu in der Lage sind. Ein Bild im jpeg-Format kann zum Beispiel von PhotoShop und von Ihrem Browser geöffnet werden. Der Tell-Block wird benötigt, um festzulegen, welches Programm das Öffnen übernehmen soll.#

Nun da wir wissen, wie wir Dateien und Ordner adressieren, können wir sie auch bewegen oder kopieren [8].

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

Kapitel 13

Wiederholungen (Schleifen)

In allen Skripten, die wir bisher besprochen haben, wurde jede Anweisung nur einmal ausgeführt. Von Zeit zu Zeit wird es aber nötig sein, dass ein oder mehrere Statements mehrere Male ausgeführt werden. AppleScript bietet verschiedene Wege, dies zu erreichen.

Wenn Sie wissen, wie oft das Statement (oder die Gruppe von Anweisungen) ausgeführt werden soll, können Sie dies mit der entsprechenden Anzahl im 'repeat'-Statement von Skript [1] angeben. Die Zahl muss eine Ganzzahl sein, da Sie eine Operation nicht, sagen wir mal 2,7 mal ausführen können.

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

Ähnlich dem, was wir bei den 'tell'-, 'try'- und 'if...then'-Statements gesehen haben, ist ein 'end repeat'-Statement zwingend erforderlich, um AppleScript anzuzeigen, welche Statements zu der Gruppe von Statements gehören, die wiederholt werden sollen.

Anstatt eine Zahl anzugeben, können Sie auch eine Variable benutzen [2].

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

Hier [3] ist ein realistischeres Beispiel von Skript [2]. Wenn Skript [3] ausgeführt wird, kann der Benutzer des Skript eine Zahl in ein Dialog-Fenster eingeben. Da alle eingegebenen Werte in 'text returned' enden, müssen wir den Eingabewert in eine Ganzzahl konvertieren. Das ist nicht möglich, wenn der Benutzer Text oder eine Fließkomma-Zahl (Bruchzahl) eingibt. Wir müssen also entsprechende Vorsichtsmaßnahmen ergreifen.

[3]

```
-- Der Benutzer kann bestimmen, wie oft der Text gesprochen werden soll
set textToDisplay to "How often has the sentence to be repeated?"
-- Die Zahl '2' wird angezeigt, um dem Benutzer einen Hinweis auf den erwarteten Wert zu geben
display dialog textToDisplay default answer "2"
set valueEntered to text returned of the result
try
	-- valueEntered ist ein Text (keine Ganzzahl) und sieht demnach so aus: "2"
	-- Hier versuchen wir, den eingegebenen Wert in eine Ganzzahl umzuwandeln. Falls dies nicht gelingt, verhütet das try-Statement, dass das Skript abgebrochen wird.
	set valueEntered to valueEntered as integer
end try
-- Wenn valueEntered der richtigen Klasse entspricht (d.h. Ganzzahl), kann die Schleife ausgeführt werden. Falls nicht, präsentieren wir einen Dialog.
if class of valueEntered is integer then
	-- Die Schleife wird ausgeführt, falls die Umwandlung nicht mißlungen ist
	repeat valueEntered times
		say "Julia is a beautiful actress"
	end repeat
else
	display dialog "You did not enter a (valid) number."
end if
```

Nach dem 'else'-Statement [3.21] wird dem Benutzer ein Vorwurf gemacht, ohne einen Hinweis darauf, wie es richtig gewesen wäre und der Benutzer muss das Skript erneut starten. Dies bringt Ihnen nicht viele Punkte in der Macintosh Community ein. Es gibt zwei alternative 'repeat'-Statements [4, 5], die Ihnen erlauben, eine Schleife so oft auszuführen, bis eine Bedingung erfüllt ist.

[4]

```
set conditionMet to false
repeat while conditionMet is false
	-- if (some test is passed) then execute the following statement
	-- set conditionMet to true
end repeat
```

[5]

```
set conditionMet to false
repeat until conditionMet is true
	-- if (some test is passed) then execute the following statement
	-- set conditionMet to true
end repeat
```

Lassen Sie uns das 'repeat'-Statement aus Skript [4] benutzen, um die Unbequemlichkeiten aus Skript [3] zu beseitigen. Wir werden die Anfrage so oft wiederholen, bis der eingegebene Wert in eine Ganzzahl umgewandelt werden kann. Im Falle einer erfolgreichen Umwandlung werden wir die Variable 'correctEntry' auf true setzen, woraufhin die Schleife verlassen und das restliche Skript [6] ausgeführt wird. Falls der eingegebene Wert nicht zu einer Ganzzahl gewandelt werden kann, werden wir dem Benutzer eine detaillierte Rückmeldung geben.

[6]

```
set correctEntry to false
repeat while correctEntry is false
	-- Der Benutzer kann bestimmen, wie oft der Text gesprochen werden soll
	set textToDisplay to "How often has the sentence to be repeated?"
	-- Die Zahl '2' wird angezeigt, um dem Benutzer einen Hinweis auf den erwarteten Wert
		zu geben
	display dialog textToDisplay default answer "2"
	set valueEntered to text returned of the result
	try
		-- valueEntered ist immer ein Text.
		-- Hier versuchen wir, den eingegebenen Wert in eine Ganzzahl umzuwandeln.
	Falls dies nicht gelingt, springen wir zur 'on error section'
		set valueEntered to valueEntered as integer
		-- Das Setzen von correctEntry auf true beendet die Schleife
		set correctEntry to true
	on error
		-- Ausführliche Rückmeldung.
		try
			-- Zuerst prüfen wir, ob der Benutzer eine Bruchzahl eingegeben hat
			set valueEntered to valueEntered as number
			display dialog "You entered a fractional number instead of an
integer."
		on error
			-- Wenn es keine Zahl ist, muss es ein Text sein"
			display dialog "Instead of an integer, like 9 , you entered text."
```

```
            end try
            -- Da der Wert von correctEntry immer noch false ist, läuft die Schleife weiter.
        end try
end repeat
-- Das Skript schafft es nur bis hierher, falls correctEntry true ist. CorrectEntry ist nur true wenn valueEntered erfolgreich in eine Ganzzahl umgewandelt wurde.
repeat valueEntered times
        say "Julia is a beautiful actress"
end repeat
```

Beachten Sie bitte, dass die Platzierung des Statements 'set correctEntry to true' sehr wichtig ist. Es muss sich

- innerhalb des (ersten) try-Blocks und
- nach dem Statement, das einen Fehler verursachen kann (hier der Versuch der Datentypumwandlung)

befinden. Andernfalls würde correctEntry auf true gesetzt werden, ungeachtet dessen, ob die Bedingung (erfolgreiche Umwandlung zu einer Ganzzahl) erfüllt ist.

#
Obwohl Skript [3] die beabsichtigte Aktion (Ein gesprochener Satz wird eine bestimmte Anzahl oft wiederholt) genauso wie Skript [6] ausführen kann, ist es weder benutzerfreundlich, noch stabil. D.h. es kann fehlschlagen wenn der Benutzer falsche Daten eingibt und es liefert nicht die geeignete Rückmeldung im Falle dass der Benutzer einen Fehler macht. Dennoch kann auch Skript [6] derart ergänzt werden, dass der Wert von valueEntered begrenzt wird (z.B. mit dem Skriptfragment [7]), um zu verhindern, dass der Satz 10.000 mal gesprochen wird. Andererseits ist Skript [6] vielleicht schon des Guten zu viel für Ihre Zwecke.

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

Wenn Sie ein wirklich stabiles Skript brauchen, sollten Sie es ausgiebig testen. Versuchen Sie Text einzugeben, Bruchzahlen, extreme Werte usw., um sicherzustellen, dass sich das Skript richtig verhält. Unberücksichtigt bleibt in Skript [6] die Möglichkeit, dass der Benutzer eine negative Zahl eingibt. Sie können diese entweder in eine positive Zahl umwandeln oder dem Benutzer einfach anzeigen, dass eine positive Zahl erwartet wird. Interessanterweise scheitert Skript [6] nicht an der Eingabe einer negativen Zahl. Versuchen Sie es selbst, indem Sie Skript [1] modifizieren. #

Die 'repeat'-Statements der Skripte [4] und [5] können beinahe für alle Zwecke eingesetzt werden. Sie können eine Schleife ausführen um sicherzustellen, dass

- ein Benutzer eine Datei oder einen Ordner auswählt
- ein Wort in einer bestimmten Datei vorhanden ist

etc.

Im Gegensatz zu den allgemeinen Zwecken der 'repeat'-Statements der Skripte [4] und [5] sind die der Skripte [1] und [2] für zahlenbezogene Bedingungen gedacht. Es gibt noch ein paar weitere 'repeat'-Statements.

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

Wie Sie sehen können, ist es möglich, die Variable aus Statement [8.1], d. h. 'counter' innerhalb Ihres Skripts benutzen. Dennoch können Sie den Wert der Variablen nicht innerhalb der Schleife ändern.

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

Wenn Sie das Skript ausführen, wird kein gesprochener Satz ausgelassen und alle Flaschen von 1 bis 5 werden konsumiert.

Im Statement [9.1] wird eine Schrittweite von 1 defaultmäßig benutzt. Wollen Sie eine andere Schrittweite benutzen, dann können Sie das auch [10.1].

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

In Skript [10] beträgt die Schrittweite 2 und Sie werden 3 mal einen gesprochenen Satz hören {für die Werte 1, 3 und 5).

Wenn Sie eine Liste haben und jedes Element davon muss für oder von einer Operation genutzt werden, könnten Sie die Anzahl der Listenelemente zählen und eine Schleife wie in Skript [11] ausführen.

[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
```

Wie auch immer, AppleScript bietet eine elegante Alternative, die unten demonstriert wird. Das untenstehende Skript [12] erlaubt Ihnen, die Anzahl an Ordnern festzustellen, die sich in einem vom Benutzer ausgewählten Ordner befinden. Danach wird in einer Schleife eine Liste mit den Namen aller vorhandenen Ordner erstellt.

[12]

```
set folderSelected to choose folder "Select a folder"
-- Um herauszufinden, welche Ordner innerhalb des ausgewählten Ordners vorhanden sind, benötigen wir die Hilfe des Finders.
-- Hinweis: "every folder" schließt NICHT die Ordner innerhalb anderer Ordner ein. Es sind nur die Ordner, die Sie sehen würden, wenn Sie den Ordner im Finder öffnen.
tell application "Finder"
	set listOfFolders to every folder of folderSelected
end tell
-- Das Ergebnis ist eine Liste von Ordnerreferenzen (Pfade), die außerhalb des Finder-Tell-Blocks verarbeitet werden können
-- Kommentieren Sie alle folgenden Statements aus und benutzen Sie den Ergebnis-Bereich, um zu sehen, dass die Liste "listOfFolders" Elemente enthält, die so aussehen: folder "reports" of folder "Documents" of folder "ik" of folder "Users" of startup disk of application "Finder".
-- Nur der Finder und die AppleScript-Komponente von Mac OS X können mit solchen Referenzen umgehen.
-- Die Ordnernamen sollen in einer neuen Liste gespeichert werden, die hier erstellt wird
set theList to {}
repeat with aFolder in listOfFolders
	-- Wir haben Referenzen zu Ordnern und da eine Referenz den Ordnernamen enthält, kann die AppleScript-Komponente von OS X an den Ordnernamen gelangen, ohne dabei auf den Finder angewiesen zu sein.
	-- Wenn Sie weitere Eigenschaften eines Ordners herausfinden wollen,z.b.., die Ordnergröße (in Bytes), wird ein Ausflug zum Finder nötig (d.h. ein Tell-Block für das nächste Statement)
	set temp to the name of aFolder
	-- Hier fügen wir die Namen der Liste hinzu
	set theList to theList & temp
end repeat
```

Kapitel 14

Alles Routine

Durch die englisch-ähnliche Sprache von AppleScript ist es leicht, Skripte zu lesen und zu schreiben. Wir haben gesehen, dass dies aber auch in Ihre Verantwortung fällt. Z.B. ist es an Ihnen, aussagekräftige Variablennamen zu wählen und Ihr Skript mit sinnvollen Kommentaren zu versehen. AppleScript kann Ihnen dabei helfen, Ihre Skripts lesbarer zu halten, indem es 'Handler' (Routinen) zur Verfügung stellt. Stellen Sie sich vor, Sie haben die gleiche Gruppe von Anweisungen an mehreren Stellen Ihres Skripts. Falls dort ein Fehler ist, müssen Sie diesen an allen Stellen ausbessern. AppleScript bietet nun die Möglichkeit, diese Statements zu gruppieren und durch einen Namen zu benennen. Wenn Sie diesen Namen im Skript aufrufen, wird die so benannte Befehlsgruppe ausgeführt.

So wird ein Handler definiert [1].

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

Um ihn zu benutzen, muss Ihr Skript ihn etwa so aufrufen [2].

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

Es spielt keine Rolle, ob der Handler in Ihrem Skript vor oder nach diesem Aufruf definiert wird.

Der Handler aus Skript [1] ist reichlich unflexibel. Es wäre nett, wenn wir dem Handler mitteilen könnten, welchen Text er anzeigen soll. Nun raten Sie mal – Das ist genau das, wofür Handler bestimmt sind [3].

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

Im Statement [3.1] nimmt die Variable 'textToDisplay' den Wert entgegen, der ihr beim Handleraufruf (in den Statements [3.4] und [3.5], die jeweils einen Wert zwischen den Anführungszeichen enthalten, der dann an den Handler weitergereicht wird) übergeben wird.

Anstatt die Daten erst beim Handleraufruf festzulegen, können Sie auch eine Variable benutzen [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)
```

Beachten Sie, dass sich der Variablenname, der beim Aufruf des Handlers benutzt wird, von dem in der Handler-Definition unterscheidet [4.1]. Sie brauchen also nicht zu wissen (oder nachzusehen), welcher Variablenname vom Handler benutzt wird. Natürlich muss Ihnen aber klar sein, welchen Datentyp der Handler erwartet.

Sie können nicht nur Informationen an einen Handler übergeben, sondern auch Informationen aus diesem zurück liefern.

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

Der Handler 'circleArea()' führt die Berechnung pi*3^2 durch und liefert automatisch das Ergebnis. Jedoch funktioniert diese automatische Rückmeldung, genau wie beim 'get'-Befehl, nur für die letzte, durch den Handler ausgeführte Anweisung. Um sicherzustellen, dass Sie das Ergebnis erhalten, das Sie wollen, auch wenn es irgendwo innerhalb einer Gruppe von Anweisungen generiert wird [6.3], benutzen Sie das Schlüsselwort 'return' [6].

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

Wenn der Vergleich in Statement [6.2] true ergibt, liefert das Statement [6.3], welches nicht die letzte Anweisung im Handler ist, das Ergebnis.

Sie müssen sich nicht auf die Übergabe von einzelnen Werten [7.7] an einen Handler beschränken.

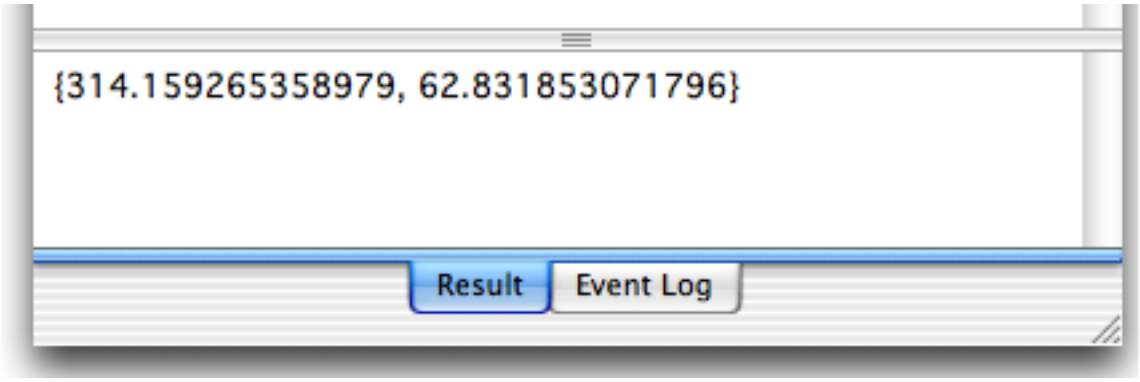
[7]
```
on largest(a, b)
	if a > b then
		return a
	end if
	return b
end largest
set theLargest to largest(5, 3)
```

Das obige Skript liefert den größeren von zwei Werten. Beachten Sie, dass in Statement [7.1] zwei Variablen zur Übernahme der Werte bereit stehen. Dementsprechend müssen auch beim Aufruf des Handlers zwei Werte [7.7] angeboten werden. In einem praxisbezogenen Skript würde zumindest einer dieser Werte in Form einer Variablen übergeben werden.

Es ist ebenso möglich, mehr als einen Wert vom Handler zurück zu bekommen. Stellen Sie diese Daten hierfür in Form einer Liste zur Verfügung [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)
```

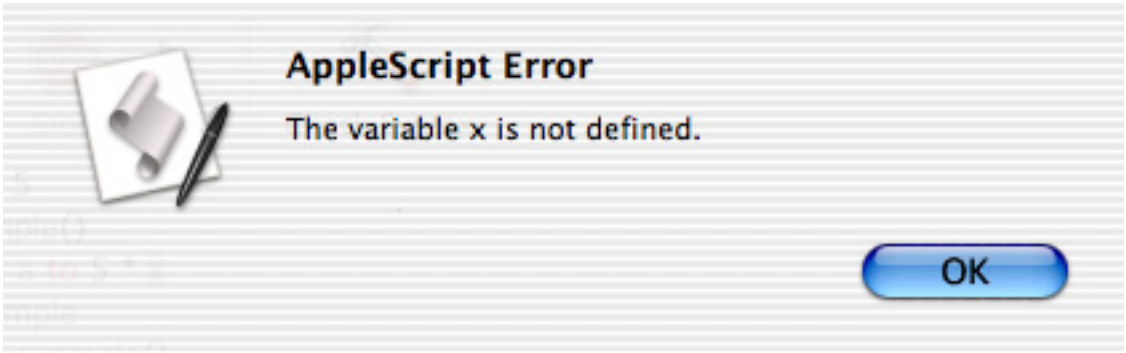
Das obige Skript liefert eine Liste zurück, die beide Werte enthält – Werte für die Kreisfläche und den Umfang eines Kreises mit dem angegebenen Radius.



Das Schreiben eines Skripts als eine Reihe von Handlern mit ein paar Statements, die das Grundgerüst des Skripts darstellen, kann Ihnen, dank einer wichtigen Eigenschaft von Handlern, eine Menge Ärger und Zeit ersparen. Hier eine Frage an Sie (die ich gleich selbst beantworten werde): Was passiert, wenn Ihr Skript einen Variablennamen benutzt, der bereits für andere Zwecke innerhalb eines Handlers vorhanden ist? Keine Angst, das stört nicht. Um diesen Punkt zu belegen, führen Sie das folgende Skript [9] aus.

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

Sie werden folgende Fehlermeldung erhalten:



Innerhalb des Handlers ist die Variable x nicht definiert. Wenn Sie wollen, dass der Wert von x [9.1] auch innerhalb des Handlers bekannt ist, müssen Sie ihn wie in Skripten, wie z.B. [4], übergeben.

Umgekehrt hat auch die Änderung einer Variablen innerhalb eines Handlers keinen Einfluß auf außerhalb. Hier ist der Beweis [10].

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

Der Ergebnisbereich zeigt an, dass x 5 ist. Es besteht also keine Wechselbeziehung zwischen den Handlern und dem Skript, in dem sie sind, mit Ausnahme dessen, was ausdrücklich an den Handler übergeben und von diesem zurück geliefert wird. Ich sollte noch erwähnen, dass es sehr wohl eine Möglichkeit gibt, die Variable x auch ohne Übergabe innerhalb des Handlers funktionieren zu lassen. Dadurch würde das Skript jedoch schwerer zu lesen und zu debuggen sein. Außerdem würde der große Vorteil von Handlern, der im nächsten Absatz besprochen wird, zunichte gemacht werden.

Abgesehen von den obigen, wichtigen Pluspunkten (Lesbarkeit etc.) ist es von großem Nutzen, dass Sie Handler in anderen Skripten wiederverwenden können. Da der Handler bereits in einem Skript genutzt und getestet wurde, können Sie ziemlich sicher sein, dass er auch in anderen Skripten funktioniert. Dadurch sparen Sie sich beim Erstellen des neuen Skripts Zeit. Sie brauchen nicht zu denken, dass Sie Statements, die sich nicht in einem Handler befinden, so einfach von einem Skript in das andere kopieren können. Zunächst müssten Sie das gesamte Skript durchgehen um herauszufinden, was kopiert werden muss, was zweifellos viel Zeit in Anspruch nimmt. Darüberhinaus wäre das Risiko zu groß, dass der kopierte Skriptteil nicht komplett ist und/oder in Ihrem neuen Skript nicht zuverlässig funktioniert. Vertrauen Sie mir – Handler sind der richtige Weg.

Ok, Sie können es nun genießen, Ihre Handler in anderen Skripten effektiv wieder zu verwenden. Aber was, wenn Sie einen Fehler in einem Handler finden oder an eine Verbesserung denken? Sie müssten all Ihre Skripte ändern. AppleScript bietet auch für dieses Problem eine Lösung – den 'load script'-Befehl. Alles was Sie tun müssen, ist

1) Sichern Sie einen oder mehrere Handler in einem kompilierten Skript
2) bauen Sie in das Skript, das die Handler benötigt, folgende Anweisung ein

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

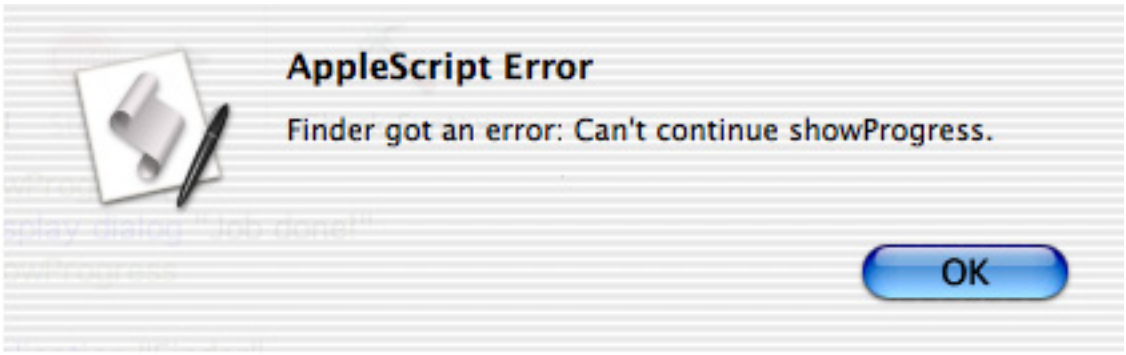
Um einen Handler des kompilierten Skripts zu benutzen, benötigt man einen Tell-Block.

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

Eine seltsame Eigenart von Handlern ist es, dass bei der Benutzung von Tell-Blöcken ohne eine zusätzliche Maßnahme der Handler innerhalb des Tell-Blocks anscheinend als nicht definiert gilt. Wenn Sie Skript [11] ausführen

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

erscheint folgende Fehlermeldung:



Die Abhilfe ist einfach. Geben Sie einfach an, dass es sich um einen Handler des Skripts selbst handelt [12.7].

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

Dies ist nur bei Handlern notwendig, nicht bei Variablen [13], wie Sie unten sehen können.

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

Das Ergebnis ist 5.

Kapitel 15

Informationsquellen

Das einzige Ziel diese Buchs ist, Ihnen die Grundlagen von AppleScript beizubringen. Wenn Sie mit dem Buch zweimal durch sind und die Beispiele auch in eigenen Variationen getestet haben, sind Sie bereit, zu lernen, wie man die Programme, die für Sie wichtig sind, per Skript steuert.

Ein wichtiger Rat für alle, die mit dem Schreiben ihrer eigenen Skripts beginnen wollen: Tun Sie es nicht! Irgendjemand hat womöglich schon erledigt, was Sie brauchen. Sie können sich also Zeit sparen, indem Sie nach solch einem Skript suchen und es an Ihre Bedürfnisse anpassen. Wo findet man diese Skripte? Natürlich im Internet. Ihr erster Besuch sollte der Seite von Apple unter

http://www.apple.com/applescript

gelten, die viele wertvolle Links bereit hält.

Ich empfehle Ihnen auch dringendst, die Seite

http://macscripter.net

zu bookmarken. Macscripter hat ein (englisch-sprachiges) Forum, wo Sie Ihre Fragen stellen können. Hilfsbereite Mitglieder und auch die Mitarbeiter von Macscripter werden ihr Möglichstes tun, um Ihnen zu helfen. Ja, wir sprechen hier über die Macintosh Community. Die Site bietet auch eine große Anzahl Links zu anderen Seiten und Informationsquellen.

Deutschsprachige Unterstützung erhalten Sie auf der Seite des Übersetzers

http://fischer-bayern.de/applescript

Dort finden Sie das erste und wohl größte deutschprachige AppleScript-Forum, nützliche Workshop-Artikel, die deutschen Ausgaben des MacScripter's Magazine, eine Seite über AppleScript-Bücher und eine Download-Seite mit freien Skripten.

Falls Sie einen professionellen AppleScript-Programmierer mit Ihrer Arbeit betrauen möchten, empfehle ich Ihnen Daniel Blanken, den Sie über seine Website

http://advancedscripting.de

kontaktieren können.

Ich hoffe, Sie haben das Buch genossen und werden AppleScript treu bleiben. Oh, und bitte vergessen Sie Kapitel 0 nicht.

Bert

Spezieller Dank ergeht an Claus C. Pilz,

http://www.pilz-online.de

der mir bei der Umsetzung des Materials als PDF mit Rat und Tat zur Seite gestanden hat. Seine Vorschläge haben meine Übersetzung an vielen Stellen in verständlichere Bahnen gelenkt.

Peter Fischer

Index

L

M

N

O

P

R

S

T

U

V

W