KAPITOLA 4

Proměnné, třídy, operátory a změna datového typu

V této kapitole najdete:

- Klíčové vědomosti a koncepce
- Práce s proměnnými
- Trvalé ukládání dat v atributech skriptu
- Provádění operací prostřednictvím operátorů
- Třídy
- Typová konverze

Klíčové vědomosti a koncepce

- Práce s proměnnými
- Datové typy
- Používání operátorů v operacích a srovnáních
- Třídy jazyka AppleScript
- Změna datového typu (typová konverze)

Často budete potřebovat ve vašich skriptech dočasně ukládat nějaká data, abyste je mohli použít později. Za tímto účelem se v jazyce AppleScript používají proměnné. Místo toho, abyste například uživatele žádali o zadání jeho jména pokaždé, když jej ve skriptu potřebujete použít, můžete jej o jeho zadání do políčka pro zadávání vstupu požádat pouze jednou, uložit jméno do proměnné a tu pak dále ve skriptu používat dle potřeby.

V této kapitole se naučíte proměnné deklarovat, přiřazovat jim data a používat je ve zdrojovém kódu. Dále se naučíte používat operátory jazyka AppleScript při provádění operací (například sčítání nebo dělení) nebo při srovnávání (například když potřebujete zjistit, zda je jedna hodnota větší než jiná hodnota nebo je jí rovná). Na konci kapitoly pak najdete výklad různých tříd a objektů, které vám AppleScript nabízí, a naučíte se provádět typovou konverzi (změnu datového typu).

Práce s proměnnými

Proměnná je pojmenovaná oblast v paměti počítače, do které můžete ukládat data – například název vaší společnosti, dva měsíce staré datum nebo kolik jste od té doby prodělali peněz.

Když potřebujete uložit ve skriptu nějaká data, použijete právě proměnnou. Obsah proměnné pak můžete získat, kdykoliv tyto informace potřebujete nějak použít, nebo jej přepsat informacemi novými.

Sedm datových typů

Když vytvoříte proměnnou, můžete jí přiřadit kterýkoliv ze sedmi typů dat. Jejich popis a příklady jejich využití najdete v tabulce 4.1.

Tabulka 4.1: Datové typy proměnných jazyka AppleScript

Datový typ	Data uložená v proměnné	Příklad nebo definice
Boolean	Pouze hodnota `true` nebo `false`	true
Integer	Celé číslo (bez desetinných míst)	10
Real	Číslo s dvojitou přesností (s desetinnými místy)	39282,87270
Date	Číslo s plovoucí desetinnou čárkou. Část čísla před čárkou reprezentuje datum, část čísla za čárkou čas.	AppleScript vám umožňuje získávat různé části data – například rok, den nebo čas.
List	Jakákoliv data, která specifikujete mezi dvěma složenými závorkami a oddělíte čárkami	**{„Praha“, „Brno“, „Ostrava“, „Hradec Králové“}**
Record	Seznam párů klíč-hodnota	**set client to {name:“Ostrava“, city:“-Průmyslové město“}**
String	Text uzavřený v dvojitých uvozovkách („“)	**set prompt to „Uložit dokument?“**

Při psaní skriptů v jazyce AppleScript za normálních okolností nemusíte specifikovat datové typy proměnných explicitně. Místo toho je AppleScript automaticky odvodí z dat, která proměnným přiřazujete, a sám proměnné odpovídajícím způsobem otypuje.

Představte si například, že vytvoříte proměnnou následujícím způsobem:

```
set IsUserSane to true
```

Protože proměnné přiřazujete hodnotu `true`, AppleScript odvodí, že by tato proměnná měla být typu Boolean – mít hodnotu `true`, nebo `false` – a přiřadí jí tento typ.

Poznámka

Chcete-li přiřadit nějaké proměnné řetězec literálů „True" nebo „False", vložte jej do dvojitých uvozovek. Applescript tak pozná, že má být proměnná typu String.

A podobně, když přiřadíte proměnné textový řetězec, AppleScript z ní automaticky udělá proměnnou řetězcovou:

```
set myUsername to "Petr"
```

Vytvoření proměnné

Chcete-li vytvořit proměnnou, stačí když prostřednictvím příkazu `set` specifikujete její název a přiřadíte jí nějaká data. Následující příkaz například vytvoří proměnnou s názvem `myGreeting` a přiřadí jí řetězec `"Dobré ráno!"`.

```
set myGreeting to "Dobré ráno!"
```

Vytvořená proměnná uchovává svůj obsah – přiřazená data –, dokud jej nějak nezměníte tak, že proměnné přiřadíte jiná data. To můžete udělat několika různými způsoby, jejichž popis najdete níže v této kapitole.

Rozdíl mezi příkazy set a copy

Ve všech až doposud uvedených příkazech se vždy používal při vytvoření proměnné a přiřazení jejích dat příkaz `set`. Je zde však ještě jeden příkaz, jehož prostřednictvím můžete vytvořit proměnnou a uložit do ní nějaká data – příkaz `copy`.

Příkaz `copy` funguje ve většině případů stejně jako příkaz `set`, ale má jinou syntaxi – jeho syntaxe je zrcadlově obrácená. Místo příkazu `set myGreeting to "Dobré ráno!"` byste například mohli použít následujícím způsobem příkaz `copy`:

```
copy "Dobré ráno" to myGreeting
```

Výsledkem provedení tohoto příkazu je vytvoření proměnné s názvem `myGreeting`, která obsahuje řetězec `"Dobré ráno!"`. V běžných případech, jako je tento, jsou příkazy `set` a `copy` víceméně zaměnitelné.

Rozdíly mezi oběma příkazy se však projeví, když vytváříte proměnnou, která obsahuje datum, seznam, záznam (Record) nebo skriptový objekt. Zde jsou:

- Když použijete příkaz `set`, AppleScript přiřadí proměnné odkaz na objekt. Odkaz je ukazatel ukazující na objekt, který obsahuje data reprezentovaná proměnnou.

- Když použijete příkaz `copy`, AppleScript přiřadí proměnné oddělenou kopii objektu, která je na originálním objektu nezávislá – takže když po použití příkazu `copy` originální objekt změníte, bude originální objekt obsahovat jiná data než objekt reprezentovaný nově vytvořenou proměnnou.

To může vést ke zmatkům, když nastavíte hodnoty dvou proměnných tak, aby odkazovaly na stejný objekt. V následujícím úryvku zdrojového kódu se například vytvoří proměnná `CompanyOffices` a přiřadí se jí seznam tří měst: Praha, Brno a Ostrava. Dále se ve skriptu vytvoří proměnná `Destinations` a prostřednictvím příkazu `set` se jí přiřadí objekt `CompanyOffices`. Nakonec se změní první položka proměnné `Destinations` a zobrazí se dialog zobrazující první položku proměnné `CompanyOfficces`. O dialogových oknech se dočtete více v kapitole 8.

```
set CompanyOffices to {"Praha", "Brno", "Ostrava"}
set Destinations to CompanyOffices
tell Destinations to set {item 1} to {"Hradec Králové"}
display dialog item 1 of CompanyOffices
```

Když tento kód provedete, zobrazí se v dialogu Hradec Králové, a ne Praha. Výsledkem změny hodnoty proměnné `Destinations` je současná změna hodnoty `CompanyOffices`, protože obě proměnné vytvořené prostřednictvím příkazu `set` ukazují na stejný objekt.

Chcete-li zabránit tomu, aby se při změně hodnoty proměnné `Destinations` současně změnila i hodnota proměnné `CompanyOffices`, použijte místo příkazu `set` příkaz `copy` a vytvořte oddělenou kopii proměnné. Zdrojový kód (se změnou zvýrazněnou tučným písmem) pak zobrazí v dialogu podle očekávání hodnotu Praha.

```
set CompanyOffices to {"Praha", "Brno", "Ostrava"}
copy CompanyOffices to Destinations
tell Destinations to set {item 1} to {"Hradec Králové"}
display dialog item 1 of CompanyOffices
```

Pravidla pro názvy proměnných

Při vytváření názvů proměnných musíte v jazyce AppleScript dodržovat několik pravidel. Tato pravidla jsou poměrně volná, takže můžete vytvářet proměnné s nejrůznějšími názvy, aniž byste je porušili. Zde jsou:

- **Název proměnné musí začínat písmenem** Název každé proměnné musí začínat písmenem.
- **Název proměnné mohou tvořit pouze písmena, čísla a podtržítka** Po prvním písmenu názvu může následovat jakákoliv kombinace písmen, čísel a podtržítek. Mnoho lidí používá podtržítka v názvech proměnných k oddělení jednotlivých slov, protože v nich nelze používat mezery ani jiné interpunkční znaky. Název proměnné `first_name` je například čitelnější než název `firstname`. Jednotlivá slova v názvu proměnné můžete také oddělovat velkými písmeny (například `First Name`), nebo používat oba způsoby (například `First_Name`) – záleží na vás.
- **O velikost písmen se nemusíte starat** V názvech proměnných se nerozlišují velká písmena od malých, ale jazyk AppleScript automaticky upravuje písmena v názvu proměnné způsobem, kterým je poprvé zapíšete. Takže když vytvoříte proměnnou `myCompany`, můžete později v kódu zadat její název jako `mycompany` (nebo použít velká či malá písmena jakýmkoliv jiným způsobem – například `MYCOmpaNY`) a AppleScript při kompilaci skriptu opraví velikost písmen podle jejího původního názvu.

Tip

Automatická oprava velikosti písmen v názvech proměnných je většinou užitečná, ale může být také nepraktická, když se rozhodnete, že chcete změnit velikost písmen v původním názvu proměnné poté, co skript zkompilujete. V těchto případech musíte AppleScript Editor zavřít a opět spustit předtím, než jej budete moci přesvědčit, aby novou velikost písmen v názvu proměnné přijal.

- **Vyhněte se používání vyhrazených slov** Nepoužívejte vyhrazená slova jazyka AppleScript – žádné ze slov definovaných jako klíčová. Nevytvářejte například proměnné s názvy `result` nebo `error`, protože AppleScript tato slova používá. Toto pravidlo je sice zcela přirozené, ale v praxi je velmi snadné jej porušit, protože většina lidí si nedokáže zapamatovat všechna klíčová slova jazyka AppleScript. Pokud AppleScript ohlásí neočekávanou chybu syntaxe, podívejte se, jestli jste omylem nepoužili jako název nějaké klíčové slovo.

Poznámka

Pokud opravdu musíte, můžete použít jako název proměnné i vyhrazené slovo. Chcete-li to udělat, vložte jej mezi dvě svislé čáry (znaky |). Pokud jste například nuceni použít jako název proměnné slovo `error`, označte jej v kódu takto: `|error|`. Obvykle k tomu však není žádný důvod. Tuto syntaxi také můžete použít, když chcete vytvořit název proměnné obsahující znaky, které byste jinak v názvu použít nemohli, například mezery nebo symboly. K tomu však také obvykle není žádný důvod, pokud vám to nečiní nějaké zvláštní potěšení.

Vytvoření proměnné odkazující na jiný objekt

Místo toho, abyste proměnné přiřadili obsah nějakého objektu, jí také můžete přiřadit odkaz na objekt. Díky tomu můžete pracovat kdykoliv, když proměnnou použijete, s aktuálním obsahem objektu, a ne s obsahem, který jste do proměnné vložili při jejím vytvoření. To se hodí, když se může hodnota odkazovaného objektu změnit za běhu skriptu.

Chcete-li vytvořit odkaz, použijte při vytváření proměnné operátor `reference to`. Následující blok příkazu tell například vytvoří proměnnou `myWin` odkazující na okno aplikace Finder, které je v popředí. Poté nastaví atribut `position` proměnné `MyWin`.

```
tell the application "Finder"
    set myWin to a reference to the front window
    set the position of myWin to {800, 44}
end tell
```

Na tom není nic složitého. Použití odkazu však začne být zajímavé ve chvíli, kdy se změní samotný objekt. Následující rozšířený blok příkazu `tell` (s úpravami zvýrazněnými tučným písmem) otevírá další okno aplikace Finder, které tentokrát zobrazuje obsah systémového disku. Protože je toto nové okno nyní v popředí, proměnná `myWin` bude odkazovat na něj a druhý příkaz `set the position of myWin` změní polohu nového okna, a ne okna, které skript otevřel jako první.

```
tell the application "Finder"
    set myWin to a reference to the front window
    set the position of myWin to {800, 44}
    open startup disk
    set the position of myWin to {0, 44}
end tell
```

Rozsahy platnosti a životnost proměnných

AppleScript vám umožňuje deklarovat dva různé druhy proměnných: lokální proměnné a globální proměnné.

Za normálních okolností proměnné vytvářítе, když je ve skriptu potřebujete použít, jako jste to dělali ve výše uvedených příkladech. Avšak vytvoříte-li proměnnou tímto způsobem, jedná se o proměnnou *lokální* – proměnnou, která je dostupná pouze v části skriptu, ve které se vytváří, a která uchovává svoji hodnotu pouze po dobu provádění skriptu.

Píšete-li skript, který má pouze jednu část, vystačíte si při ukládání dat s lokálními proměnnými. Ale když píšete skript, který obsahuje více různých podprogramů (*podprogram* je oddělená sekce zdrojového kódu se specifickou funkcí), může se stát, že budete muset použít také proměnné globální. *Globální* proměnná je dostupná ve všech podprogramech ve skriptu i v jeho hlavním těle. Lokální proměnná vytvořená v podprogramu je oproti tomu dostupná pouze v tomto podprogramu, a nelze k ní přistupovat v žádném z ostatních podprogramů ani v hlavním těle skriptu.

Oblasti, ve které je proměnná dostupná, se říká *rozsah platnosti proměnné*, takže globální proměnné mají globální rozsah platnosti a lokální proměnné lokální rozsah platnosti.

Chcete-li vytvořit globální proměnnou, musíte to udělat předem, aby o ní skript věděl. Globální proměnnou deklarujete prostřednictvím klíčového slova `global` a jejího názvu. V následujícím příkazu se například deklaruje globální proměnná `myCity`:

```
global myCity
```

Za normálních okolností se všechny globální proměnné deklarují v hlavním těle skriptu, a ne v jeho podprogramech – viz následující příklad, ve kterém je deklarace globální proměnné `myUserName` zvýrazněná tučným písmem. Globální proměnná je tak dostupná v hlavním těle skriptu a ve všech jeho podprogramech, což obvykle chcete. Ve skriptu se nejprve volá podprogram `get_user_name`, který zobrazí dialog vyžadující zadání jména uživatele, uloží jej do proměnné `myUserName` a poté zavolá podprogram `show_user_name`, který obsah proměnné `myUserName` zobrazí v dalším dialogu.

```
global myUserName

get_user_name()
show_user_name()

on get_user_name()
    display dialog "Zadejte prosím vaše jméno:" default answer ""
    set myUserName to text returned of the result
end get_user_name

on show_user_name()
    display dialog myUserName
end show_user_name
```

Někdy však také musíte deklarovat globální proměnnou pouze v podprogramech, které ji potřebují. V následujícím skriptu se deklaruje globální proměnná `myUserName` v podprogramu `get_user_name` (opět zvýrazněno tučně), takže je dostupná v tomto podprogramu a v hlavním těle skriptu, ale ne v podprogramu `show_user_name`:

```
get_user_name()
show_user_name()
```

```
on get_user_name()
    global myUserName
    display dialog "Zadejte prosím vaše jméno:" default answer ""
    set myUserName to text returned of the result
end get_user_name

on show_user_name()
    display dialog myUserName
end show_user_name
```

V tomto případě není přesunutí deklarace do podprogramu `get_user_name` vhodné, protože podprogram `show_user_name` selže a ohlásí chybu. Stane se to proto, že podprogram `show_user_name` neví o proměnné `myUserName`, jejíž obsah má zobrazit v dialogu prostřednictvím příkazu `display dialog`.

Abyste tuto chybu napravili, musíte deklarovat globální proměnnou `myUserName` také v podprogramu `show_user_name` – viz kód vytištěný tučně:

```
get_user_name()
show_user_name()

on get_user_name()
    global myUserName
    display dialog "Zadejte prosím vaše jméno:" default answer ""
    set myUserName to text returned of the result
end get_user_name

on show_user_name()
    global myUserName
    display dialog myUserName
end show_user_name
```

Poznámka

Název každé globální proměnné ve skriptu musí být jedinečný. Název každé lokální proměnné pak musí být jedinečný v jejím rozsahu platnosti, ale pokud chcete, můžete používat stejné názvy proměnných v různých rozsazích platnosti. Obecně vzato však není opětovné používání názvů lokálních proměnných ve stejném skriptu vhodné, protože to může být matoucí.

Předem můžete deklarovat i proměnnou lokální – v deklaraci použijete s jejím názvem klíčové slovo `local`. V následujícím příkazu se například deklaruje lokální proměnná `Boss`:

```
local Boss
```

Každá deklarace lokální proměnné musí být ve stejné části skriptu, ve které chcete proměnnou používat – v jeho hlavním těle (pokud ji nechcete používat v podprogramu), nebo v podprogramu, který ji používá.

Zeptejte se odborníka

Otázka: Musím deklarovat lokální proměnné předem prostřednictvím klíčového slova `local`?

Odpověď: Ne, nemusíte.

Dokonce i když začnete deklarovat globální proměnné, nemusíte předem deklarovat lokální proměnné prostřednictvím klíčového slova `local` – můžete je i nadále vytvářet prostřednictvím příkazu `set` kdekoliv ve vašem zdrojovém kódu.

Ale – cítili jste, že nějaké to „ale" přijde, že? – když používáte globální proměnné, může se hodit používat i deklarace proměnných lokálních, aby bylo na první pohled jasné, jaké jsou rozsahy platnosti jednotlivých proměnných.

Další výhodou deklarování lokálních proměnných předem je možnost umístit všechny jejich deklarace v podprogramu na jedno místo. Budete tak mít dokonalý přehled nad všemi proměnnými, které podprogram používá. To se hodí, když se ke svému kódu vrátíte po nějaké době nebo když se jej pokouší pochopit někdo jiný.

Vyzkoušejte si

Použití globální proměnné

V tomto příkladu prostřednictvím globální proměnné zpřístupníte data různým podprogramům ve skriptu. V tomto příkladu se pracuje se skriptem, který jste viděli již výše v této kapitole. Ve skriptu je několik věcí, jejichž podrobný popis jsme zatím neuvedli, včetně vytváření a volání podprogramů a zobrazování dialogů, ale uvidíte, že na tom není nic těžkého. Při psaní skriptu postupujte následujícím způsobem:

1. Stiskem kláves ⌘+**N**, nebo výběrem položky **New** z nabídky **File** vytvořte nový skript.
2. Na začátku skriptu vytvořte globální proměnnou `myUserName` – uděláte to tak, že do skriptu vložíte klíčové slovo `global` a za ním uvedete název proměnné:
```
global myUserName
```
3. Zavolejte podprogram `get_user_name` – uděláte to tak, že vložíte do skriptu jeho název následovaný párem kulatých závorek (zvýrazněno tučným písmem). Tímto způsobem vyjádříte, že chcete podprogram `get_user_name` provést.
```
global myUserName
get_user_name()
```
4. Na dalším řádku zavolejte stejným způsobem podprogram `show_user_name` (zvýrazněno tučným písmem):
```
global myUserName
get_user_name()
show_user_name()
```
5. Vytvořte podprogram `get_user_name` – uděláte to tak, že do skriptu vložíte klíčové slovo `on` a za ním uvedete název podprogramu následovaný párem kulatých závorek. Na další řádek poté vložte klíčové slovo `end`, kterým podprogram ukončíte (zvýrazněno tučným písmem):
```
global myUserName
get_user_name()
show_user_name()

on get_user_name()

end
```
6. Stiskem kláves ⌘+**K**, nebo klepnutím na tlačítko **Compile** skript zkompilujte. AppleScript automaticky přidá do příkazu `end` název podprogramu `get_user_name` (zvýrazněno tučným písmem):

```
global myUserName
get_user_name()
show_user_name()

on get_user_name()

end get_user_name
```

7. Do podprogramu `get_user_name` přidejte příkaz `display dialog`, který zobrazí dialog vyžadující zadání jména uživatele do textového pole, které je na začátku prázdné (parametr `default answer ""` – výchozí odpověď). Nastavte hodnotu globální proměnné na text vrácený textovým polem. Příslušné úpravy jsou označené tučně:

```
global myUserName
get_user_name()
show_user_name()

on get_user_name()
    display dialog "Zadejte prosím vaše jméno:" default answer ""
    set myUserName to text returned of the result
end get_user_name
```

8. Vytvořte podprogram `show_user_name` a vložte do něj příkaz `display dialog`, který zobrazí obsah proměnné `myUserName`. Příslušné úpravy jsou označené tučně:

```
global myUserName
get_user_name()
show_user_name()

on get_user_name()
    display dialog "Zadejte prosím vaše jméno:" default answer ""
    set myUserName to text returned of the result
end get_user_name

on show_user_name()
    display dialog myUserName
end show_user_name
```

9. Stiskem kláves ⌘+**R**, nebo klepnutím na tlačítko **Run** skript spusťte. Až se objeví první dialog, zadejte do textového pole nějaké jméno a klepněte na tlačítko **OK**. Zkontrolujte, že se zadaný text zobrazí v druhém dialogu.
10. Uložte skript pod libovolným názvem.

Trvalé ukládání dat v atributech skriptu

Když potřebujete uchovat data od jednoho spuštění skriptu k druhému, použijte místo proměnné atribut skriptu. *Atribut skriptu* jsou nějaká data uložená ve skriptu, která můžete používat a ukládat dle potřeby.

Pro používání atributů skriptu platí téměř stejná pravidla jako pro používání proměnných (viz výše):

- Každý název atributu skriptu musí být v rámci skriptu jedinečný. (Atributy skriptů jsou ve skriptu globální – jejich platnost nemůžete omezovat pouze na některé jeho oblasti.)

- Každý název atributu skriptu musí začínat písmenem.
- Po prvním písmenu názvu může následovat jakákoliv kombinace písmen, čísel a podtržítek, ale nelze používat mezery a symboly.
- Pro názvy atributů nepoužívejte vyhrazená slova jazyka AppleScript. Pokud opravdu chcete, můžete použít jako název atributu i vyhrazené slovo. Uděláte to tak, že jej vložíte mezi dvě svislé čáry (chcete-li například použít jako název atributu slovo „dialog“, použijte `|dialog|`). Tuto syntaxi také můžete použít, když chcete vytvořit název atributu obsahující znaky, které byste jinak v názvu použít nemohli, například mezery nebo symboly.

Atribut skriptu deklarujete prostřednictvím klíčového slova `property`, za které vložíte název atributu, mezeru, dvojtečku, další mezeru a jeho výchozí hodnotu. Atribut na rozdíl od proměnné nelze deklarovat, aniž byste mu přiřadili výchozí hodnotu, ale jeho výchozí hodnota může být prázdná (například prázdný řetězec `""` nebo prázdný seznam `{}`).

Poznámka

Atribut můžete inicializovat s jakýmkoliv datovým typem nebo objektem – například se seznamem, oknem nebo dokumentem.

V následujícím příkazu se například deklaruje atribut `committee_name` a přiřazuje se mu řetězec `"Řídící výbor managementu"`:

```
property committee_name : "Řídící výbor managementu"
```

Atributy je nejlepší deklarovat hned na začátku skriptu, kde si každý, kdo bude váš kód číst, jejich deklarací hned všimne, ale není to povinné – deklarace atributů můžete umístit kamkoliv do hlavního těla skriptu; nemůžete je deklarovat v bloku příkazu `tell` nebo v podprogramu.

Chcete-li, můžete atributy používat k ukládání dat, která se nikdy nezmění, ale ve většině případů je lepší taková data „natvrdo“ zapsat do skriptu. Atributy vám pak zřejmě budou připadat užitečnější k ukládání dat, která se *mění*. Do atributu můžete například uložit název složky, ve které uživatel skript naposledy spouštěl a poté ji použít při následujícím spuštění skriptu jako složku výchozí – viz příklad. V tomto příkladu se používá příkaz `choose folder`, o kterém se dozvíte více v kapitole 8.

```
property starting_folder : "/"
set starting_folder to choose folder default location starting_folder
```

Upozornění

Při používání atributů skriptů si musíte dávat pozor na jednu věc – jakmile odpovídajícím způsobem nastavíte hodnotu atributu a skript zkompilujete, nespouštějte jej, aby se hodnota atributu před distribucí skriptu nezměnila. Jinak se skript příště spustí s hodnotou atributu, kterou jste v něm do atributu naposledy uložili. Místo toho skript zkompilujte a otestujte. Pokud bude fungovat tak, jak má, udělejte v něm nějakou triviální změnu (například přidejte komentář) a zkompilujte jej znovu, aby si zachoval vlastnosti, které jste specifikovali při jeho psaní.

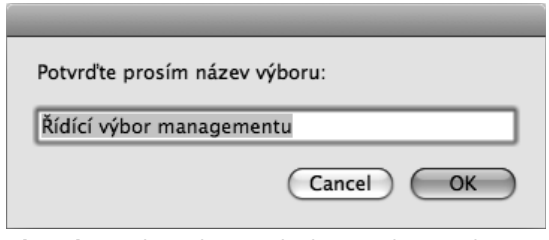
Vyzkoušejte si

Použití atributu skriptu

V tomto příkladu deklarujete atribut skriptu, který bude obsahovat název výboru, zobrazíte dialog vyžadující potvrzení názvu výboru a poté nastavíte hodnotu atributu na základě hodnoty zadané v tomto dialogu. O dialozích se toho dozvíte více v kapitole 8, ale na tomto náhledu do způsobu jejich využití není nic těžkého.

Při psaní skriptu postupujte následujícím způsobem:

1. Stiskem kláves ⌘+**N**, nebo výběrem položky **New** z nabídky **File** vytvořte nový skript.
2. Deklarujte atribut s názvem `committee_name` a nastavte jeho výchozí hodnotu na `"Řídící výbor managementu"`:
```
property committee_name : " Řídící výbor managementu"
```
3. Přidejte příkaz `display dialog`, který je v následujícím výpisu označený tučně. Tento dialog zobrazí hodnotu atributu `committee_name` a požádá uživatele o její potvrzení nebo změnu:
```
property committee_name : "Řídící výbor managementu"
display dialog "Potvrďte prosím název výboru:" default answer committee_name
```
4. Přidejte příkaz `set`, který nastaví hodnotu atributu `committee_name` na text vrácený z textového pole dialogového okna (zvýrazněno tučným písmem):
```
property committee_name : "Řídící výbor managementu"
display dialog "Potvrďte prosím název výboru:" default answer committee_name
set committee_name to text returned of the result
```
5. Stiskem kláves ⌘+**R**, nebo klepnutím na tlačítko **Run** skript spusťte. Zobrazí se dialog, který je na obrázku 4.1.



Obrázek 4.1: Chcete-li změnit hodnotu atributu ve skriptu, zadejte jiný název výboru

6. Zadejte do dialogu jiný název výboru a klepnutím na tlačítko **OK** dialog zavřete. AppleScript uloží zadaný název do skriptu.
7. Spusťte skript znovu. Tentokrát bude dialog zobrazovat vámi zadaný název.
8. Uložte skript pod libovolným názvem.

Provádění operací prostřednictvím operátorů

Pojem o*perátor* v jazyce AppleScript označuje znak, který provádí nějakou operaci s určenými daty. Některé operátory jsou charakteristické pro jazyk AppleScript, ale jiné, které mají obecnější funk-

ci, jistě budete znát. Ve výrazu 100 - 50 je například znak - operátorem pro odčítání, který vám říká (nebo jazyku AppleScript), abyste od první hodnoty (100) odečetli druhou hodnotu (50).

Podobně jako operátor odčítání i většina ostatních operátorů pracuje se dvěma hodnotami, neboli *operandy*. Tomuto druhu operátorů se pak říká *operátory binární*. Dalším druhem operátorů jsou *operátory unární*, které pracují s jediným operandem.

V tabulce 4.2 jsou operátory rozdělené do kategorií, jejich popis a příklad využití.

Tabulka 4.2: Operátory jazyka AppleScript

Operátor	Popis	Příklad
Aritmetické operátory		
+	Sčítání	5 + 3 = 8
-	Odčítání	5 – 3 = 2
-	Unární negace (označení záporného čísla)	–3
*	Násobení	5 * 3 = 15
/	Dělení	6 / 3 = 2
Div	Celočíselné dělení (vrací celočíselnou část výsledku dělení a ignoruje zbytek)	`27 div 7` vrací `3` `28 div 7` vrací 4
Mod	Modulo (zbytek po celočíselném dělení)	`9 mod 2` vrací 1 `10 mod 2` vrací 0 `24 mod 7` vrací 3
^	Umocnění	2^3 vrací 8 2^4 vrací 16 2^5 vrací 32
Logické operátory		
And	Konjunkce (tenhle ***a*** tenhle)	name begins with „T“ and size is greater than 10000000
Or	Disjunkce (tenhle ***nebo*** tenhle)	name begins with „T“ or name begins with „W“
Not	Negace (tenhle ***ale ne*** tenhle)	name begins with „T“ not name begins with „W“
Operátor zřetězení		
&	Zřetězení (spojení dvou řetězců dat do jednoho)	„Dobré ráno,„ &„vesmíre!“ vytvoří řetězec Dobré ráno, vesmíre!
Omezující operátory		
begin[s] with/start[s] with (začíná na)	Najde uvedenou položku na začátku cíle.	name begins with „A“
end[s] with (končí na)	Najde uvedenou položku na konci cíle.	name ends with „tion“

Operátor	Popis	Příklad
contains (obsahuje)	Najde v cíli uvedenou položku.	name contains „test“
does not contain/ doesn't contain (neobsahuje)	Najde cíl, který neobsahuje uvedenou položku.	name does not contain „Projekt“
is in (je v)	Najde cíl, který odpovídá jedné z uvedených položek.	name extension is in {„doc“,„docx“}
is not in (není v)	Najde cíl, který neodpovídá ani jedné z uvedených položek.	name extension is not in {„doc“,„docx“}
is contained by (je obsažen/a/o v)	Zjistí, zda danou entitu obsahuje nějaká jiná entita.	{„Tokyo“,„Paříž“} contains {„Paříž“} vrací `true`
is not contained by/ isn't contained by (není obsažen/a/o v)/	Zjistí, zda danou entitu nějaká jiná entita neobsahuje.	{„Tokyo“,„Paříž“} does not contain {„Soul“} vrací `true`
Srovnávací operátory pro rovnost		
is equal to (je rovno)	Můžete také použít =, `equal`, `equals`, nebo `equal to`. Když skript zkompilujete, AppleScript automaticky nahradí jakékoliv variace výrazem `is equal to`.	1 is equal to 2 vrací `false`
is not equal to (není rovno)	Můžete také použít /=, `does not equal`, `doesn't equal` nebo `is not equal` (bez „`to`“). Když skript zkompilujete, AppleScript automaticky nahradí jakékoliv variace výrazem `is not equal to` a znaky /= znakem ≠ (není rovno).	„kočka“ is not equal to „pes“ vrací `true`
is (je)	Pokud je první operand totožný s druhým operandem, nebo je mu roven, vrátí `true`; jinak vrátí `false`.	object1 is object2
is not (není)	Pokud první operand není totožný s druhým operandem, nebo mu není roven, vrátí `true`; jinak vrátí `false`.	object1 is not object2
Srovnávací operátory priority		
is less than (je menší než)	Můžete také použít < nebo `less than`. Když skript zkompilujete, AppleScript automaticky nahradí všechny výrazy `less then` výrazem `is less then`. Pokud je první operand menší než druhý operand, vrátí `true`.	1 is less than 2

Operátor	Popis	Příklad
is greater than (je větší než)	Můžete také použít > nebo `greater than`. Když skript zkompilujete, AppleScript automaticky nahradí všechny výrazy `greater then` výrazem `is greater then`.	„moucha“ is greater than „slon“ vrací `false`
is greater than or equal to (je větší nebo rovno)	Můžete také použít ≥ nebo `is greater than or equal`. Když skript zkompilujete, AppleScript automaticky nahradí všechny textové verze výrazem `is greater than or equal to`.	4 is greater than or equal to 5 vrací `false`
is not greater than or equal to (není větší nebo rovno)	Můžete také použít `is not greater than or equal`, `isn't greater than or equal` nebo `isn't greater than or equal to`. Když skript zkompilujete, AppleScript automaticky nahradí všechny textové verze výrazem `is not greater than or equal to`.	4 is not greater than or equal to 5 vrací `true`
is less than or equal to (je menší nebo rovno)	Můžete také použít ≤ nebo `is less than or equal`. Když skript zkompilujete, AppleScript automaticky nahradí všechny textové verze výrazem `is less than or equal to`.	10 is less than or equal to 10 vrací `true`
is not less than or equal to (není menší nebo rovno)	Můžete také použít `is not less than or equal`, `isn't less than or equal` nebo `isn't less than or equal to`. Když skript zkompilujete, AppleScript automaticky nahradí všechny textové verze výrazem `is not less than or equal to`.	10 is not less than or equal to 10 vrací `false`
comes before (je před)	Otestuje, zda je nějaké číslo nebo řetězec v pořadí před jiným číslem nebo řetězcem.	1 comes before 2 vrací `true`

Operátor	Popis	Příklad
does not come before (není před)	Otestuje, zda nějaké číslo nebo řetězec není v pořadí před jiným číslem nebo řetězcem. Můžete také použít výraz `doesn't come before`. Když skript zkompilujete, AppleScript automaticky nahradí všechny textové verze výrazem `does not come before`.	`1 does not come before 2` vrací `false`
comes after (je za)	Otestuje, zda je nějaké číslo nebo řetězec v pořadí za jiným číslem nebo řetězcem.	„konec“ comes after „začátek“ vrací `true`
does not come after (není za)	Otestuje, zda nějaké číslo nebo řetězec není v pořadí za jiným číslem nebo řetězcem. Můžete také použít výraz `doesn't come after`. Když skript zkompilujete, AppleScript automaticky nahradí všechny textové verze výrazem `does not come after`.	„začátek“ does not come after „konec“ vrací `true`

Poznámka

Chcete-li vložit do skriptu znak √, použijte znaky >=; když skript zkompilujete, AppleScript je automaticky nahradí znakem odmocniny. A podobně, chcete-li vložit do skriptu znak Π, použijte znaky <=, a chcete-li vložit znak ≠, použijte znaky `/=`.

Třídy

Pojem *třída* označuje v jazyce AppleScript kategorii objektů, které mají podobné charakteristické vlastnosti. Třídu `file` například tvoří odkazy na objekty v souborovém systému – soubory, složky nebo oddíly disku. Jednotlivé objekty třídy `file` se od sebe obvykle vzájemně liší, protože každý z nich odkazuje na jiný soubor (složku, oddíl), ale jedná se o objekty stejného typu.

Popis tříd zabudovaných do jazyka AppleScript najdete v tabulce 4.3. Jak jste viděli již v předchozí kapitole, aplikace TextEdit má například třídu `document`, která reprezentuje objekty dokumentů, a třídu `paragraph`, která reprezentuje objekty odstavců.

Tabulka 4.3: Zabudované třídy jazyka AppleScript

Třída	Popis	Příklad nebo doplňující informace
alias	Odkaz na existující soubor, složku nebo diskový oddíl v souborovém systému počítače Macintosh. Alias neexistujícího objektu vytvořit nemůžete.	set myAlias to „Macintosh HD:Users:“ as alias

Třída	Popis	Příklad nebo doplňující informace
application	Aplikace v počítači Macintosh nebo na serveru, ke kterému je počítač připojený.	tell application „TextEdit" to quit
boolean	Booleovská hodnota, buď true, nebo false.	set docExists to true
class	Třída objektu nebo hodnoty.	`class of 123.45` vrací `real`
constant	Hodnota předdefinovaná jazykem AppleScript nebo nějakou aplikací; své vlastní konstanty ve skriptech vytvářet nemůžete.	AppleScript nabízí textové konstanty pro práci s textem – například `tab`, `space`, `return`, `linefeed` a `quote`.
date	Den v týdnu, datum (měsíc, den, rok) a čas (hodiny, minuty, sekundy).	Příkaz `current date` vrací aktuální datum – například: `"Thursday, April 1, 2010 9:48:16 AM"`
file	Odkaz na soubor, složku nebo diskový oddíl v souborovém systému počítače Macintosh. Můžete vytvořit i objekt typu `file`, který odkazuje na neexistující objekt.	V příkazu `set myFile to choose file name` se prostřednictvím příkazu `choose file name` umožňuje uživateli specifikovat název souboru (viz kapitola 8).
integer	Celé číslo (číslo bez desetinných míst).	set myInteger to 7
list	Seřazená kolekce položek.	set myList to {„vejce",„kuře",„slepice"}
number	Celé, nebo reálné číslo.	Jedná se o abstraktní třídu; skutečná třída jakéhokoliv čísla musí být vždy buď `integer`, nebo `real`.
POSIX file	Pseudotřída, která vrací objekt třídy `file`. Nejedná se o třídu; umožňuje vám vyhodnocovat POSIX specifikátor souboru.	`set fileName to POSIX file "/System"` vrací objekt třídy file, například `"Macintosh HD: System"`
real	Číslo, které může mít i desetinná místa.	set myReal to 1.43
record	Kolekce atributů označených popisky, ke kterým můžete přistupovat místo na základě jejich pozice na základě hodnot jejich popisků.	set myDog to {name:"Žeryk", animal:"Pes", breed:"Vořech"}
reference	Objekt odkazující na jiný objekt.	set docWin to a reference to the front window of the application „TextEdit"
RGB	Seznam celočíselných hodnot v intervalu 0-65535 o třech položkách reprezentujících červenou, zelenou a modrou složku barvy.	`{65535, 0, 0}` specifikuje plnohodnotnou červenou
script	Skript v jazyce AppleScript.	Tento skript zobrazuje aktuální datum: script DateScript display dialog (current date) as string end script
text	Řetězec Unicode znaků.	set headline to „Zvláštní nabídka"
unit types	Kolekce tříd měrných jednotek – obsahuje například třídy `feet` a `miles`.	set distance to 200 as miles

Typová konverze

Typová konverze je operace, při které konvertujete data jednoho typu na data jiného typu. Typovou konverzi například použijete, když potřebujete konvertovat celé nebo reálné číslo na řetězec.

AppleScript provádí v mnoha případech nezbytné typové konverze automaticky a vy se o nich dozvíte pouze v případě, že dojde k výskytu nějaké chyby – například když váš zdrojový kód potřebuje přetypovat nějaká data na datový typ, na který je přetypovat nelze.

Typovou konverzi na konkrétní datový typ můžete provádět také ručně, kdykoliv potřebujete. Konverzi provedete prostřednictvím operátoru `as` a specifikace cílového datového typu. Následující příkazy například vytvoří celočíselnou proměnnou `myInteger` o hodnotě `100`, a poté ji prostřednictvím výrazu `as string` konvertují na řetězec. výsledkem je hodnota `"100"` (včetně dvojitých uvozovek, které označují, že se jedná o řetězec).

```
set myInteger to 100
myInteger as string
```

V tabulce 4.4 je úplný výčet typových konverzí, které můžete v jazyce AppleScript provádět, spolu s jejich popisem a příklady jejich použití.

Tabulka 4.4: Typové konverze v jazyce AppleScript

Původní třída	Přetypování na třídu	Popis nebo poznámka
alias	List	Vrací seznam o jedné položce
alias	Text	Vrací textový řetězec
boolean	Integer	Vrací 1 pro `true` a 0 pro `false`
boolean	List	Vrací seznam o jedné položce, buď `{true}`, nebo `{false}`
boolean	Text	Vrací řetězec buď `"true"`, nebo `"false"`
class	List	Vrací seznam o jedné položce
class	Text	Vrací řetězec
constant	List	Vrací seznam o jedné položce
constant	Text	Vrací řetězec
date	List	Vrací seznam o jedné položce ve formátu zobrazovaném v předvolbách **International** – například `"Thursday, April 1, 2010 6:44:03 AM"`
date	Text	Vrací řetězec ve formátu zobrazovaném v předvolbách **International** – například `"Thursday, April 1, 2010 6:44:03 AM"`
file	List	Vrací seznam o jedné položce
file	Text	Vrací textový řetězec
integer	List	Vrací seznam o jedné položce – například `{150}`
integer	Real	Vrací reálné číslo – například `150.0`
integer	Text	Vrací řetězec obsahující celočíselnou hodnotu – například „150“

Původní třída	Přetypování na třídu	Popis nebo poznámka
item from list	[různé]	Položku na seznamu můžete přetypovat na jakoukoliv třídu, na kterou můžete přetypovat jedinou položku původní třídy. Máte-li například seznam položek třídy `alias`, můžete je přetypovat na položky třídy `text`.
list	Text	`{"Ráno", "Odpoledne", "Večer"} as string` vrací `RánoOdpoledneVečer`
number	Integer	Vrací celočíselnou část čísla.
number	List	Vrací seznam o jedné položce obsahující dané číslo – například `{178.24}`
number	Real	Vrací reálné číslo – například `178.24`
number	Text	Vrací řetězec obsahující dané číslo – například „178.24"
POSIX file	List	Vrací seznam o jedné položce
POSIX file	Text	Vrací řetězec
real	Integer	Vrací celočíselnou část čísla – `178.24 as integer` například vrací `179`
real	List	Vrací seznam o jedné položce – například `{178.24}`
record	List	Vrací seznam s odstraněnými popisky. `{name:"Žeryk", animal:"Pes", breed:"Vořech"} as list` například vrací `{"Žeryk", "Pes", "Vořech"}`
reference	[různé]	Odkaz můžete přetypovat na jakýkoliv typ, na který můžete přetypovat odkazovaný objekt.
script	List	Vrací seznam o jedné položce.
text	Integer	Vrací celé číslo z řetězce obsahujícího nějaké číslo. `"150.5" as integer` například vrací `150`.
text	List	Vrací seznam o jedné položce řetězce – `"Praha" as list` například vrací `{"Praha"}`
text	Real	Vrací reálné číslo z řetězce obsahujícího nějaké číslo. `"150.5" as integer` například vrací `150.5`.
unit types	Integer	Vrací celé číslo odpovídající celočíselné části hodnoty specifikované položky (například `liters`).
unit types	List	Vrací seznam o jedné položce obsahující hodnotu specifikované položky (například `miles`).
unit types	Real	Vrací reálné číslo odpovídající hodnotě specifikované položky (například `centimeters`).
unit types	Text	Vrací řetězec obsahující hodnotu specifikované položky (například `degrees Celsius`).

Vyzkoušejte si

Vytvoření proměnné a její přetypování

V tomto příkladu vytvoříte proměnnou jedné ze tříd jazyka AppleScript a přiřadíte jí nějaká data. Poté prostřednictvím typové konverze vrátíte data v jiném formátu.

1. Stiskem kláves ⌘+**N**, nebo výběrem položky **New** z nabídky **File** vytvořte v aplikaci AppleScript Editor nový skript.
2. Prostřednictvím příkazu `set` vytvořte proměnnou `ThisDay` a přiřaďte jí aktuální datum:
   ```
   set ThisDay to current date
   ```
3. Přidejte příkaz `display dialog`, který zobrazí v dialogu obsah proměnné `ThisDay` přetypovaný na řetězec, aby jej mohl dialog zobrazit:
   ```
   display dialog ThisDay as string
   ```
4. Stiskem kláves ⌘+**R**, nebo klepnutím na tlačítko **Run** na liště nástrojů skript spusťte. AppleScript zobrazí dialog s aktuálním datem.
5. Uložte skript pod libovolným názvem.