Ara Pehlivanian, Don Nguyen

sitepoint

JavaScript

OKAMŽITĚ

computer press

OVLÁDNĚTE JAVASCRIPT ZA VÍKEND

Ara Pehlivanian, Don Nguyen

JavaScript Okamžitě

Computer Press
Brno
2014

JavaScript Okamžitě

Ara Pehlivanian, Don Nguyen

Překlad: Ondřej Baše
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-4163-2

Vydalo nakladatelství Computer Press v Brně roku 2014 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 18 313.

1. vydání

ALBATROS MEDIA a.s.

Obsah

Poděkování

Ara

Děkuji své milující manželce Kristě, bez jejíhož půvabu, trpělivosti a podpory bych tuto knihu nikdy nenapsal. Také děkuji našim dvěma úžasným dcerám.

Don

Lorraine, děkuji ti za to, že mi neustále podstrojuješ teplé jídlo, plníš mé srdce horoucí láskou a udržuješ zářivý úsměv na mém obličeji.

Úvod

JavaScript je mocný, všestranný a všudypřítomný programovací jazyk. Od svého skromného počátku[1] v polovině 90. let 20. století, kdy byl jakýmsi doplňkem od společnosti Netscape k jazyku Visual Basic společnosti Microsoft, vyrostl do jednoho z nejoblíbenějších[2] programovacích jazyků na světě.

JavaScript má jedinečné postavení mezi programovacími jazyky, jelikož jako jediný se nachází na téměř všech osobních počítačích po celém světě. Všechny moderní webové prohlížeče implementují jazyk JavaScript. Jedná se o skriptovací jazyk pro web. Přestože začínal jako jednoduchý jazyk, který nacházel uplatnění ve validaci formulářů a drobné manipulaci s obsahem stránky, vyvinul se, a proto v něm dnes můžete vytvářet bohaté klientské aplikace. Kromě toho – v průběhu tohoto období začal jazyk JavaScript nahrazovat do určité míry zásuvný modul Flash.

Protože web neustále roste a zdokonaluje se, touha po vývojářích ovládajících jazyk JavaScript také narůstá. Ať už jste zkušený programátor, který se chce tento jazyk naučit, nebo začátečník dychtící zaplnit volné pozice na pracovním trhu, tato kniha vám pomůže. Po přečtení této knihy budete schopní psát vlastní aplikace v jazyce JavaScript a budete mít dostatek znalostí, abyste se mohli vydat na cestu, na jejímž konci se z vás stane opravdový znalec tohoto jazyka.

Komu je tato kniha určena

Webovým návrhářům a vývojářům, kteří chtějí rychle proniknout do tajů jazyka JavaScript. Předpokladem k tomu je však základní znalost jazyků HTML a CSS.

Použité konvence

V této knize narazíte na řadu typografických konvencí a různých stylů rozvržení, které budou zvýrazňovat různé typy informací. Prohlédněte si proto následující konvence.

Ukázky zdrojového kódu

Zdrojový kód bude napsán neproporcionálním písmem; například takto:

```
<h1>Skvělý letní den</h1>
<p>Byl krásný letní den, který přímo vybízel k procházce parkem. Ptáčci
zpívali a všechny děti byly zpátky ve škole.</p>
```

1 *http://en.wikipedia.org/wiki/JavaScript#Birth_at_Netscape*

2 *http://javascript.crockford.com/javascript*

Pokud se daný zdrojový kód nachází v archivu zdrojových kódů k této knize, příslušné jméno souboru se objeví nad tímto výpisem:

příklad.css
```
.zapati {
  background-color: #CCC;
  border-top: 1px solid #333;
}
```

Pokud bude výpis zobrazovat jen část obsahu souboru, bude označen slovem úryvek:

```
priklad.css (úryvek)
border-top: 1px solid #333;
```

Když budeme k současnému příkladu přidávat další zdrojový kód, takový kód se bude zobrazovat tučně:

```
function animuj() {
  var nova_promenna = 'Ahoj';
}
```

Místo již napsaného zdrojového kódu bude výpis obvykle obsahovat tři tečky:

```
function animuj() {
  ...
  return nova_promenna;
}
```

Tipy, poznámky a upozornění

Tip: Haló, vy tam!
Tipy vám poskytují malé cenné rady.

Poznámka: Ehm, promiňte...
Poznámky jsou doplňující informace, které se týkají tématu, ale nejsou kriticky důležité.

Upozornění: Nezapomeňte vždy...
... věnovat pozornost těmto sdělením.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu přeložilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

Computer Press
Albatros Media a.s., pobočka Brno
IBC
Příkop 4
602 00 Brno

nebo

sefredaktor.pc@albatrosmedia.cz

Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Zdrojové kódy ke knize

Z adresy *http://knihy.cpress.cz/K2139* si po klepnutí na odkaz Soubory ke stažení můžete přímo stáhnout archiv s ukázkovými kódy.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nelze. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji oznámíte. Ostatní uživatele tak můžete ušetřit frustrace a nám můžete pomoci zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K2139* po klepnutí na odkaz Soubory ke stažení.

KAPITOLA 1

Začínáme

V této kapitole:
- Konzola
- Zápis kódu JavaScriptu do souborů HTML

Jako vývojáři pracující v jazyce JavaScript budete psát zdrojový kód, který poběží ve webovém prohlížeči. Mohli byste si hned vytvořit dokument HTML, do kterého byste vkládali ukázkové zdrojové kódy jazyka JavaScript z této knihy, ale než se pustíte do kompletního projektu, můžete si zatím vyzkoušet některé koncepce. Nejlepší způsob, jak můžete začít, je, že si otevřete svůj oblíbený webový prohlížeč a do jeho konzoly zapíšete zde uvedené příklady. Všechny moderní webové prohlížeče mají svou konzolu; níže pak najdete informace, jak v nich tuto konzolu otevřít. Po otevření konzoly uvidíte příkazový řádek, na jehož začátku se nachází text výzvy, za nějž můžete začít psát kód. Textem výzvy bývají obvykle jedna, dvě nebo tři pravé ostré závorky (>, >> nebo >>>).

Tip: Víceřádkový režim

Pokud chcete zadat do konzoly více řádků kódu současně a používáte webový prohlížeč Google Chrome nebo Safari, stiskněte kombinaci kláves *Shift* + *Enter*, čímž přejdete na nový řádek. Stiskem samotné klávesy *Enter* posléze svůj kód spustíte.

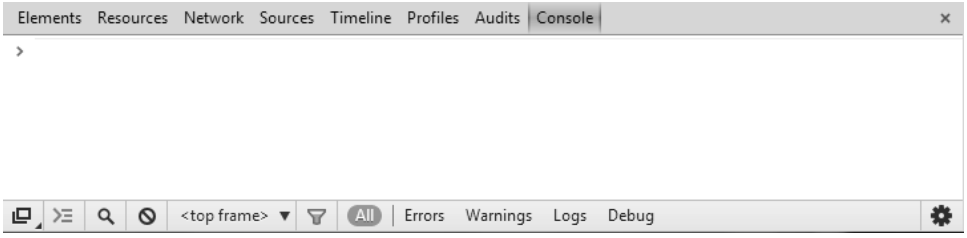
Jestliže používáte prohlížeč Firefox nebo Internet Explorer, víceřádkový režim aktivujete v konzole klepnutím na příslušné tlačítko v pravém dolním rohu. V tomto režimu stiskem klávesy *Enter* přecházíte na nový řádek, kdežto celý kód můžete spustit pomocí klávesové zkratky *Ctrl* + *Enter* nebo klepnutím na k tomu určené tlačítko.

Konzola

V této části kapitoly se dozvíte, jak otevřít vývojářskou konzolu v nejoblíbenějších webových prohlížečích současnosti. Pokud zde nenajdete svůj oblíbený prohlížeč, s použitím vyhledávače Google jistě najdete správný postup.

Google Chrome

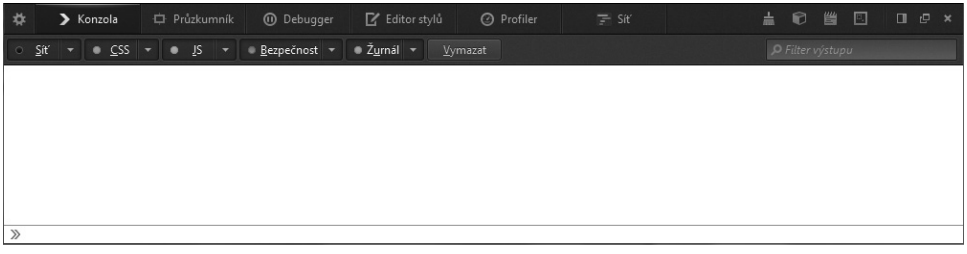
Vývojářskou konzolu prohlížeče Google Chrome aktivujete v operačních systémech Windows a Linux stiskem kláves *Shift* + *Ctrl* + *J*. Alternativně je možné používat také klávesu *F12* nebo *Fn* + *F12*, pokud máte místo funkčních kláves nastavené klávesy multimediální. V operačním systému Mac OS použijte kombinaci kláves *Option* + *Command* + *J*. Tímto způsobem otevřete panel **Console** v okně vývojářských nástrojů, jak je patrné na obrázku 1.1.



Obrázek 1.1. Konzola prohlížeče Google Chrome

Firefox

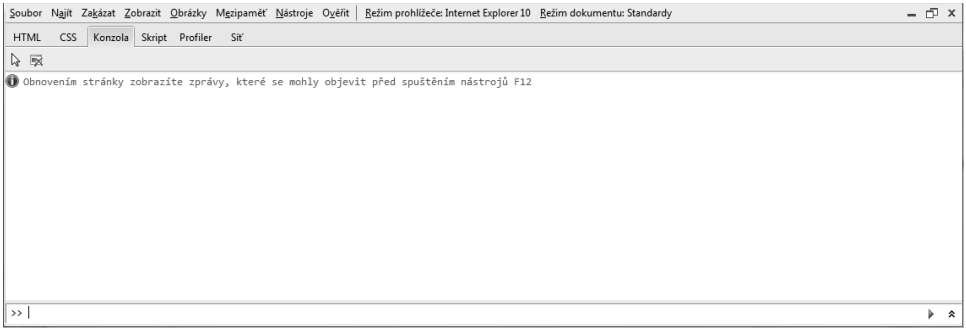
Konzolu otevřete v prohlížeči Firefox pod operačními systémy Windows a Linux stisknutím klávesové zkratky *Shift* + *Ctrl* + *K*. Alternativně je možné používat také klávesu *F12* nebo *Fn* + *F12*, pokud máte místo funkčních kláves nastavené klávesy multimediální. V operačním systému Mac OS stiskněte klávesy *Option* + *Command* + *K*. Výsledek si můžete prohlédnout na obrázku 1.2.



Obrázek 1.2. Konzola prohlížeče Firefox

Internet Explorer

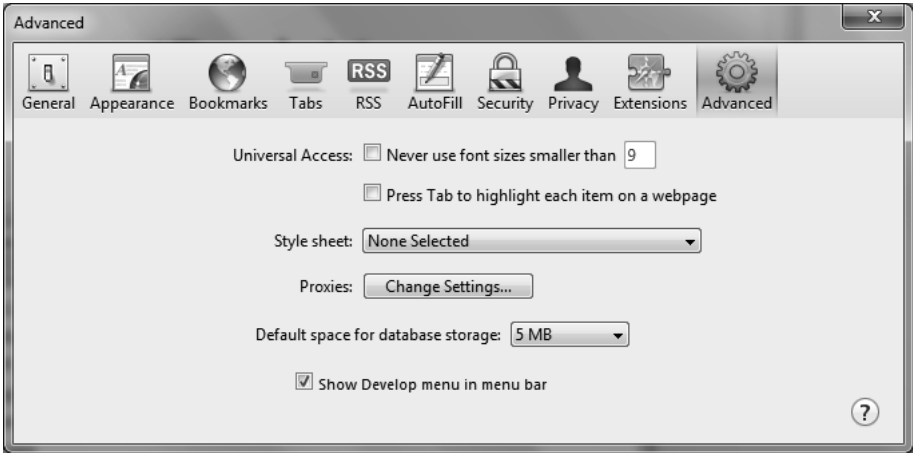
Konzolu prohlížeče Internet Explorer (viz obrázek 1.3) aktivujete stisknutím klávesy *F12* nebo kláves *Fn + F12*, jestliže máte místo funkčních kláves nastavené klávesy multimediální.



Obrázek 1.3. Vývojářské nástroje prohlížeče Internet Explorer

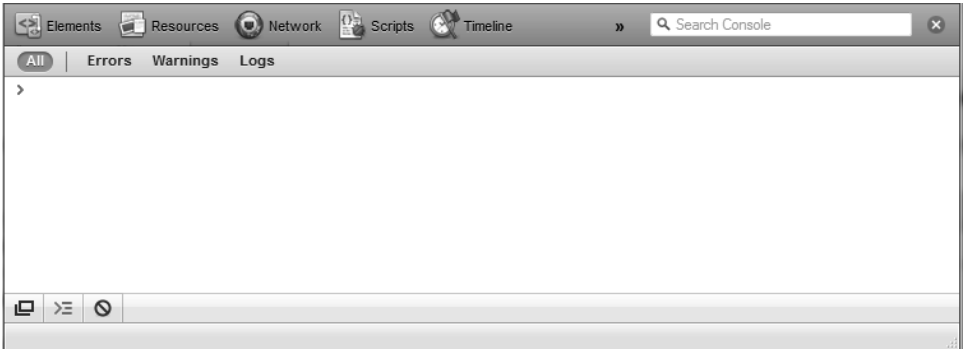
Safari

Aktivace konzoly v prohlížeči Safari je o něco náročnější než v jiných prohlížečích. Nejprve musíte povolit zobrazování nabídky **Develop**. Otevřete si okno s nastavením stiskem kláves *Command + čárka*, poté klepněte na kartu **Advanced**, na níž zatrhněte políčko **Show Develop menu in menu bar**, jak ukazuje obrázek 1.4.



Obrázek 1.4. Povolení nabídky Develop

Až dokončíte přechozí úkon, zavřete konfigurační okno a stisknutím kombinace kláves *Option* + *Command* + *C* otevřete konzolu, kterou si můžete prohlédnout také na obrázku 1.5.



Obrázek 1.5. Konzola prohlížeče Safari

Zápis kódu JavaScriptu do souborů HTML

Pokud chcete přeskočit práci s konzolou a pracovat rovnou v dokumentu HTML, můžete si vybrat ze dvou možností – psát kód jazyka JavaScript přímo do dokumentu HTML nebo do samostatného souboru a ten pak načítat v souboru HTML.

Soubor HTML

Jazyk HTML nabízí element `script`, do kterého můžeme vkládat spustitelný kód jazyka JavaScript. Zde je příklad velmi jednoduchého dokumentu HTML s jedním řádkem kódu JavaScriptu, jenž otevírá výstražné okno se zprávou „Ahoj světe!“:

js-v-html.html

```
<!DOCTYPE html>
<html lang="cs" dir="ltr">
<head>
  <meta charset="utf-8" />
  <title>Kód uvnitř dokumentu HTML</title>
  <script>
    alert("Ahoj světe!");
  </script>
</head>
<body>
</body>
</html>
```

Samostatný soubor

Kód jazyka JavaScript lze rovněž ukládat do samostatného souboru a na ten se poté odkázat z dokumentu HTML. Mohli bychom kupříkladu přesunout naše volání funkce `alert` do souboru s názvem *ahoj.js* (přípona *.js* označuje soubor obsahující zdrojový kód jazyka JavaScript) a následně ho načíst prostřednictvím atributu `src` elementu `script`.

externi-js.html

```
<!DOCTYPE html>
<html lang="cs" dir="ltr">
<head>
  <meta charset="utf-8" />
  <title>Kód v externím souboru</title>
  <script src="ahoj.js"></script>
</head>
<body>
</body>
</html>
```

Povšimněte si, že element `script` není samouzavírací. Jinými slovy – následující zápis by nefungoval:

```
<!-- Toto nebude fungovat. -->
<script src="ahoj.js" />
```

Vždy musíte uvést jak počáteční, tak koncovou značku, přestože mezi nimi není žádný obsah:

```
<script src="ahoj.js"></script>
```

Umístění elementu script

Ačkoliv se modelem DOM budeme zabývat podrobně později, měli bychom si alespoň říct, že když prohlížeč načítá dokument HTML, převádí jeho značky na svou interní reprezentaci s názvem **objektový model dokumentu**, který se však pro zjednodušení označuje jako **model DOM** (z anglického termínu **Document Object Model**). My pak můžeme v jazyce JavaScript napsat kód, v němž budeme pracovat s naší stránkou HTML prostřednictvím manipulace s modelem DOM. Musíme však být opatrní – snaha o přístup do určité části modelu DOM dříve, než ji webový prohlížeč stihne sestavit, vede k chybám.

Jestliže vložíme element `script` do záhlaví dokumentu HTML (do elementu `head`), náš zdrojový kód se spustí okamžitě, a to právě v okamžiku, kdy ještě není sestavený model DOM. Jakékoliv odkazy na elementy modelu DOM tudíž povedou k chybám, protože prohlížeč ještě nenačetl žádné elementy z elementu `body`. S tímto problémem se lze vypořádat dvěma způsoby. Můžeme buď zabalit náš kód do volání funkce `window.onload()`, kterou prohlížeč spouští, až sestaví model DOM, nebo můžeme element/y `script` vložit na konec elementu `body`, těsně před koncovou značku `</body>`.

Druhá z uvedených metod je lepší, protože když prohlížeč sestavuje model DOM, posílá serveru požadavek pro každý obrázek (element `img`). Funkci `window.onload()` nezavolá, dokud nenačte všechny obrázky. To znamená, že kód jazyka JavaScript musí čekat na kompletní načtení všech obrázků a dalších prostředků. Na malé stránce s trochou textu a jedním nebo dvěma obrázky to nemusíme ani postřehnout, ale na stránkách se spoustou obrázků budeme pozorovat výraznou prodlevu před spuštěním našich skriptů. Z toho důvodu je vhodnější vkládat elementy `script` na konec těla dokumentu.

Níže uvedený dokument HTML obsahuje jak vložený kód jazyka JavaScript, tak odkazy na externí soubory JavaScriptu. Třebaže v kódu v záhlaví dokumentu nepracujeme s modelem DOM (pouze voláme funkci `alert()`), spustí se až po načtení všech skriptů na konci dokumentu:

```
<!DOCTYPE html>
<html lang="cs" dir="ltr">
<head>
  <meta charset="utf-8" />
  <title>Kód uvnitř dokumentu HTML </title>
  <script>
    window.onload = function() {
      alert("Ahoj světe!");
    }
  </script>
</head>
<body>
  <h1>Příklad</h1>
  <p>Toto je ukázkový dokument se spoustou obrázků.</p>
  <ul>
    <li><img src="obr1.jpg" alt="Obrázek 1" /></li>
    <li><img src="obr2.jpg" alt="Obrázek 2" /></li>
    <li><img src="obr3.jpg" alt="Obrázek 3" /></li>
    <!-- mnohem více obrázků -->
  </ul>
  <script src="ahoj.js"></script>
  <script src="dalsi.js"></script>
  <script src="a-dalsi.js"></script>
</body>
</html>
```

Shrnutí

Po přečtení této kapitoly byste měli být připravení na skok do světa vývoje v jazyce JavaScript. Tato kapitola vám ukázala, jak pracovat s jazykem JavaScript ve webovém prohlížeči dle vašeho výběru. Naučili jste se také různé metody vkládání kódu JavaScriptu do svých webových stránek. Teď už vám nic nebrání učit se základům jazyka JavaScript napříč touto knihou.

KAPITOLA 2

Proměnné

V této kapitole:
- Komentáře
- Deklarace
- Typy
- Operace
- Nástrahy slabě typovaného jazyka
- Převod typů
- Operátory porovnávání
- Řízení toku

Hlavní cíl programování spočívá v manipulaci s daty. Abychom však s daty mohli pracovat, musíme je nejdříve někam uložit. V jazyce JavaScript představují proměnné základní typ datového úložiště. Podobně jako v algebře – proměnné reprezentují informace, s nimiž chceme pracovat. Na rozdíl od algebry však nemusíme pojmenovávat proměnné pouze jednotlivými písmeny. Můžeme používat celá slova, abychom lépe popsali, s čím pracujeme – například `jmeno`, `poplatek`, `delka` nebo `sirka`. Lze rovněž kombinovat více slov dohromady – kupříkladu `krestniJmeno`, `seznamUkolu` nebo `casovyInterval`. V případě víceslovného názvu proměnné píšeme první slovo malými písmeny a všechna následující slova začínají velkým písmenem. Tomuto způsobu zápisu jmen se říká **„velbloudí" zápis** (**camelCase**), přičemž se jedná o jmennou konvenci rozšířenou napříč komunitou okolo jazyka JavaScript.

Komentáře

V ukázkových zdrojových kódech z této knihy se budeme často setkávat s komentáři, proto si nyní vysvětlíme, co jsou zač. **Komentář** je nespustitelný text, který vkládáme do kódu, abychom vysvětlili, co se v něm děje. V jazyce JavaScript můžou mít komentáře dvojí podobu – jednořádkové komentáře nebo víceřádkové komentáře. Jednořádkové komentáře začínají dvěma lomítky (`//`). Vše, co následuje za dvěma lomítky až do konce řádku, je nespustitelný text. Na druhou stranu – víceřádkový (nebo také blokový) komentář můžeme zapisovat na několik řádků. Víceřádkové komentáře začínají lomítkem s hvězdičkou (`/*`) a končí opačnou kombinací těchto znaků (`*/`).

Následují příklady jednořádkových komentářů:

```
// Chystáme se deklarovat proměnnou a přiřadit jí hodnotu.
var mojePromenna = "Ahoj světe."; // Už jsme to udělali.
```

A takto vypadá víceřádkový komentář:

```
/* Právě jsme deklarovali proměnnou mojePromenna a přidělili jí hodnotu
   "Ahoj světe.". Na začátek jsme zapsali komentář, kde jsme oznámili, že
   se to chystáme provést. Na konec jsme přidali další komentář, v němž
   jsme pouze sdělili, že už jsme to udělali. Nyní píšeme opravdu dlouhý
   komentář, který by se nevešel na jeden řádek, a proto jsme z něj
   udělali víceřádkový komentář.
*/
```

Deklarace

Deklarace proměnné je velmi jednoduchá:

```
var ukol = "Napsat první kapitolu.";
```

Výše uvedeným řádkem kódu jsme deklarovali proměnnou s názvem `ukol`, a to pomocí příkladu pro deklaraci proměnné. Prostřednictvím klíčového slova `var` můžeme deklarovat jednu nebo více proměnných a případně je rovnou inicializovat s hodnotami. Například následovně:

```
var ukol = "Napsat první kapitolu.",
    dokoncen = true;
```

Právě jsme deklarovali dvě proměnné (oddělené čárkou), přičemž první z nich jsme přidělili textový řetězec a druhé z nich pravdivostní hodnotu (za malý okamžik si podrobněji popíšeme typy proměnných, jakými jsou právě textové řetězce a pravdivostní hodnoty). K deklaraci obou proměnných jsme si vystačili s jediným deklaračním příkazem. Mohli bychom také použít více deklaračních příkazů – pro každou proměnnou jeden.

Poznámka: Velikost písmen

Jazyk JavaScript rozlišuje velikost písmen v názvech, takže ukol a Ukol jsou zcela rozdílné proměnné. Až si později budeme popisovat názvy funkcí, velikost písmen bude rovněž důležitá. Když se odkazujeme na proměnnou nebo funkci, musíme tudíž pokaždé používat správnou velikost písmen.

Upozornění: Ukončujte každý řádek

Středníkem na konci řádku říkáte prohlížeči, že řádek kódu je kompletní. Pokud jej vynecháte, prohlížeč se pokusí sám vložit řádek, až ho bude číst. Tato funkce se označuje jako automatické vkládání středníku. Praxí osvědčeným postupem však je, že byste neměli dávat prohlížeči jakoukoliv záminku k tomu, aby si musel něco domýšlet. Pro jistotu tedy ukončujte každý řádek kódu středníkem.

Ještě jedna připomínka k deklaraci a inicializaci proměnných – snažte se oddělovat od sebe tyto dva úkony. Snadno se totiž můžete dostat do problémů, až budete ladit svůj kód pomocí ladicího nástroje. Pokud deklarujete a inicializujete všechny proměnné jediným příkazem, některé ladicí nástroje mohou přeskočit celý deklarační příkaz v jediném kroku. Pokud se chyba nachází v jedné z inicializací, nebudete schopni prozkoumat jednotlivá přiřazení hodnot samostatně. Lepším řešením je deklarovat všechny proměnné dohromady, ale inicializovat je samostatně, takže například takto:

```
// deklarace
var ukol, dokoncen;

// inicializace
task = "Napsat první kapitolu.";
dokoncen = true;
```

Později si ukážeme, jak lze provádět výpočty a jiné operace v průběhu přiřazování hodnoty proměnné. Rovněž zjistíme, jak to může znepříjemnit ladění zdrojového kódu.

Poznámka: Úspora místa

Některé příklady v této knize můžou jít proti výše popsanému pravidlu – budou deklarovat a inicializovat proměnné jediným příkazem. Je tomu tak pouze z důvodu úspory místa na stránce.

Typy

Pokud jste se někdy setkali s programovacími jazyky C nebo Java, u předchozího příkladu vám určitě připadalo divné, že nikde nespecifikujete, že do proměnné `ukol` chcete ukládat textový řetězec (a ne pravdivostní hodnotu nebo číslo). Je tomu tak z toho důvodu, že jazyk JavaScript je **slabě typovaný jazyk**; na rozdíl od jazyků C nebo Java, které jsou silně typované. To znamená, že ačkoliv jazyk JavaScript interně rozlišuje typy, nemusíte explicitně deklarovat typ proměnné. Typ proměnné můžete navíc dynamicky měnit za běhu. Jazyk JavaScript umí sám určit, co chcete se svými proměnnými dělat, místo toho, aby se spoléhal na to, co mu řeknete. Ve skutečnosti ani nemůžete jazyku JavaScript sdělit, že proměnná je určitého typu – jedinou možností je přidělit proměnné hodnotu daného typu.

Existuje šest datových typů, s nimiž můžete pracovat – číslo (`Number`), textový řetězec (`String`), pravdivostní hodnota (`Boolean`), `Null`, `Undefined` a objekt (`Object`).

Number

Na rozdíl od jiných programovacích jazyků má JavaScript jediný číselný typ, a to typ `Number`. Podle standardu ECMAScript[1] odpovídá typ `Number` 64bitové hodnotě v binárním formátu s dvojitou přesností definovanou standardem IEEE 754. Do typu `Number` spadají všechny číselné hodnoty a speciální hodnota `NaN` (`Not-a-Number`), kladné nekonečno a záporné nekonečno.

Poznámka: Kdy číslo není číslem

Hodnota `NaN` (`Not-a-Number`) je speciální hodnota, kterou jazyk JavaScript vrací, když selže matematická funkce (například `Math.abs("foo")`) nebo když se nepodaří převést obecnou hodnotu na číslo (například `Number("foo")`). Hodnota `NaN` je také zvláštní v tom, že jako jediná hodnota jazyka JavaScript se nerovná sobě samé. Jiným slovy – výsledkem porovnání `NaN == NaN` a `NaN === NaN` je hodnota `false`, přestože logicky by to měla být hodnota `true`, jelikož `NaN` je `NaN`. Je tedy patrné, že `NaN` dostálo svému významu „není číslo", a proto si není rovné.

Jazyk JavaScript poskytuje funkci `isNaN()`, s níž můžeme ověřovat, jestli se vrácená hodnota rovná `NaN`, ale ve skutečnosti může být její výsledek spíše matoucí. Důvodem je, že funkce `isNaN()` nejprve převádí jí předaný výraz na číslo procesem, jejž nazýváme **převod typů** (popíšeme si ho podrobně později), a proto některé hodnoty převádí na čísla, zatímco jiné nikoliv. Tato funkce vrací například správně hodnotu `true` pro hodnoty `NaN`, `undefined` a `{}` (prázdný objekt zapsaný pomocí literálu objektu), což nejsou čísla. Nevrací však už hodnotu `true` pro hodnoty `true` a `null`, což také nejsou čísla.

Mnohem záludněji se ale tato funkce chová v případě textových řetězců. Vrací hodnotu `false` pro čísla v textových řetězcích; například volání `isNaN("42")` vrací hodnotu `false`, protože funkce `isNaN()` převádí textový řetězec `"42"` na hodnotu `42`, a to samozřejmě je číslem. Rovněž vrací hodnotu `false` pro prázdné textové řetězce (`""`) a řetězce složené pouze z mezer (`"  "`), jelikož tyto textové řetězce převádí na hodnotu `0`, což je číslo. Pro textové řetězce s textem, který není možné převést na číslo, vrací hodnotu `true` (například `isNaN("foo")`). Přestože textový řetězec `"foo"` se nerovná hodnotě `NaN`, tato funkce pro něj vrací hodnotu `true`.

Jediným spolehlivým řešením, jak zjistit, jestli se testovaná hodnota skutečně rovná hodnotě `NaN`, je porovnat ji se sebou samou. Protože hodnota `NaN` je jedinou hodnotou JavaScriptu, která se sobě nerovná, porovnání stejných hodnot vrátí hodnotu `false`, a v takovém případě víme, že před sebou máme hodnotu `NaN`.

String

Do proměnných typu `String` lze ukládat všechny možné textové řetězce. Příklady:

```
"Ahoj světe!"
"1, 2, 3, 4, 5"
"!@#$%^&*()_+"
```

1 *http://www.ecma-international.org/publications/standards/Ecma-262.htm*

Boolean

Proměnné typu `Boolean` mohou obsahovat pouze pravdivostní hodnotu `true` (pravda) nebo `false` (nepravda).

Undefined

Datový typ `Undefined` (respektive jeho jediná hodnota `undefined`) je poněkud zvláštní, jelikož odpovídá stavu, v němž se proměnná nachází po své deklaraci, ale před svou inicializací (přiřazením hodnoty). Deklarováním proměnné ale sdělujeme, že vytváříme novou proměnnou (definujeme ji), a proto je tento datový typ neobvyklý.

Null

Datový typ `Null` (respektive jeho jediná hodnota `null`) používáme, pokud chceme deklarovat proměnnou a záměrně zdůraznit, že nemá žádnou hodnotu (na rozdíl od hodnoty `undefined`, jež vyjadřuje, že hodnota prostě chybí).

Object

Objekt (hodnota datového typu Object) je kolekce vlastností. Vlastnosti můžou být libovolného z předchozích datových typů, jiné objekty a funkce (více informací si uvedeme později).

Operace

Mít data je skvělé, ale dělat s nimi něco je ještě lepší. Ať už máme několik textových řetězců, které chceme spojit dohromady, nebo několik čísel, nad nimiž chceme provést nějaký výpočet. S pomocí jazyka JavaScript můžeme provádět tyto operace opravdu jednoduše. Řetězení textových řetězců dohromady spočívá pouze v zápisu operátoru `+`:

```
var jmeno, prijmeni, celeJmeno;

jmeno = "Jan";
prijmeni = "Novák";
celeJmeno = jmeno + " " + prijmeni; // celeJmeno obsahuje Jan Novák
```

Matematické operace fungují dle očekávání – sčítání (`+`), odečítání (`-`), násobení (`*`), dělení (`/`) a modulo (`%`):

```
var hrebiky, srouby, material;

hrebiky = 155;
srouby = 89;
material = hrebiky + srouby; // material obsahuje 244
```

Obdobně:

```
var mistni, spolecne, mezisoucet, celkem;

mistni = 0.095;
spolecne = 0.05;
mezisoucet = 10;
celkem = mezisoucet + (mezisoucet * mistni) +
  (mezisoucet * spolecne); // celkem obsahuje 11.45
```

S operátorem `%` (modulo) můžeme počítat zbytek po celočíselném dělení. Výpočet `10 % 3` tudíž vrací číslo `1`.

Nástrahy slabě typovaného jazyka

Ačkoliv interpreter JavaScriptu se snaží, jak nejlépe umí, aby určil, co děláme s proměnnými, a nevyžaduje po nás, abychom uváděli datové typy, my bychom si měli řádné rozmyslet, jaká data budeme do proměnných ukládat; jinak se můžeme dostat do problémů.

Například zde:

```
var pocetUkoluJana = 11,
    pocetUkoluJany = "42",
    celkovyPocetUkolu = pocetUkoluJana + pocetUkoluJany;
```

Na první pohled je patrné, že Jan má 11 úkolů a Jana má 42 úkolů, což by mělo být 53 úkolů celkem. Protože ale hodnotou proměnné `pocetUkoluJany` je textový řetězec, interpreter JavaScriptu vyhodnotí operátor `+` jako pokus o řetězení, a ne sčítání. Výsledek tedy je `"1142"` místo očekávaného čísla `53`.

Abychom získali správný výsledek, musíme se zajistit, aby obě proměnné obsahovaly čísla, a ne textové řetězce:

```
var pocetUkoluJana = 11,
    pocetUkoluJany = 42,
    celkovyPocetUkolu = pocetUkoluJana + pocetUkoluJany;
```

Převod typů

Pokud interpeteru jazyka JavaScript předložíme operaci na konfliktních datových typech, snaží se je před jejím vykonáním normalizovat. Je tomu tak u aritmetických operací, operace řetězení (jak jsme si ukázali) a při porovnávání hodnot (které si popíšeme za chvíli). Protože nemůžeme sčítat slovo a číslo, převádí nejprve čísla na textový řetězec a posléze spojí tyto dva textové řetězce dohromady (jak jsme zjistili před okamžikem). Podobně – jestliže zkusíme sečíst číslo a pravdivostní hodnotu (`true` nebo `false`), převede v prvé řadě pravdivostní hod-

notu na její číselný ekvivalent (`1` pro `true` a `0` pro `false`) a ten přičte k danému číslu. Jedná se o velmi důležitou koncepci jazyka JavaScript, protože její neporozumění může vést k chybám v našich programech.

Operátory porovnávání

Pomocí operátorů porovnávání srovnáváme dvě hodnoty a získáme buď hodnotu `true`, nebo `false` (na základě výsledku tohoto srovnání). Kdybychom kupříkladu potřebovali zjistit, jestli je číslo `10` větší než číslo `5`, napsali bychom výraz `10 > 5`, přičemž výsledkem tohoto porovnání by byla hodnota `true`. Na druhou stranu – výraz `10 > 11` vrací hodnotu `false`.

Rovnost (==)

Operátor `==` vrací hodnotu `true`, pokud se obě hodnoty rovnají. Jestliže porovnávané hodnoty nejsou stejného typu, převádí je interpreter jazyka JavaScript na stejný typ a potom aplikuje striktní porovnávání. V případě, že budeme mít číslo a pravdivostní hodnotu, převede nejprve pravdivostní hodnotu na číslo. Pokud budeme porovnávat textový řetězec a číslo, převede textový řetězec na číslo, než začne porovnávat. Pokud se jedná o objekty, rovnají se, jestliže se odkazují na stejné místo paměti:

```
1 == 1     // vrací true
"1" == 1   // vrací true ("1" převádí na 1)
1 == true  // vrací true
0 == false // vrací true
"" == 0    // vrací true ("" převádí na 0)
"  " == 0  // vrací true ("  " převádí na 0)

0 == 1     // vrací false
1 == false // vrací false
0 == true  // vrací false

var x, y;  // deklarujeme proměnné x a y
x = {};    // vytváříme objekt, který ukládáme do proměnné x
y = x;     // odkazujeme proměnnou y na umístění objektu x
x == y;    // vrací true (odkazují se na stejný objekt v paměti)
x == {};   // vrací false (nejedná se o stejný objekt)
```

Nerovnost (!=)

Stejný princip jako u rovnosti, ale opačný výsledek – hodnota `true`, pokud se hodnoty nerovnají. V tomto případě dochází také k výše popsaným převodům typů:

```
1 != 1     // vrací false
"1" != 1   // vrací false ("1" převádí na 1)
1 != true  // vrací false
0 != false // vrací false
"" != 0    // vrací false ("" převádí na 0)
"  " != 0  // vrací false ("  " převádí na 0)

0 != 1     // vrací true
1 != false // vrací true
0 != true  // vrací true

var x, y;  // deklarujeme proměnné x a y
x = {};    // vytváříme objekt, který ukládáme do proměnné x
y = x;     // odkazujeme proměnnou y na umístění objektu x
x != y;    // vrací false (odkazují se na stejný objekt v paměti)
x != {};   // vrací true (nejedná se o stejný objekt)
```

Striktní rovnost (===)

Při striktním porovnávání nedochází k převodům typů. Zatímco operace `"" == 0` by vrátila hodnotu `true`, v případě striktní rovnosti `"" === 0` bychom získali hodnotu `false`, protože prázdný textový řetězec se nerovná nule:

```
1 === 1     // vrací true
"1" === 1   // vrací false ("1" nepřevádí)
1 === true  // vrací false
0 === false // vrací false
"" === 0    // vrací false ("" nepřevádí)
"  " === 0  // vrací false ("  " nepřevádí)

0 === 1     // vrací false
1 === false // vrací false
0 === true  // vrací false

var x, y;   // deklarujeme proměnné x a y
x = {};     // vytváříme objekt, který ukládáme do proměnné x
y = x;      // odkazujeme proměnnou y na umístění objektu x
x === y;    // vrací true (odkazují se na stejný objekt v paměti)
x === {};   // vrací false (nejedná se o stejný objekt)
```

Striktní nerovnost (!==)

Stejný princip jako u striktní rovnosti, ale s opačným výsledkem. Operátor `!==` vrací `true`, pokud se hodnoty neshodují. Oproti obyčejné nerovnosti opět interpreter JavaScriptu nepřevádí typy:

```
1 !== 1     // vrací false
"1" !== 1   // vrací true ("1" nepřevádí)
1 !== true  // vrací true
0 !== false // vrací true
"" !== 0    // vrací true ("" nepřevádí)
"  " !== 0  // vrací true ("  " nepřevádí)

0 !== 1     // vrací true
1 !== false // vrací true
0 !== true  // vrací true

var x, y;   // deklarujeme proměnné x a y
x = {};     // vytváříme objekt, který ukládáme do proměnné x
y = x;      // odkazujeme proměnnou y na umístění objektu x
x !== y;    // vrací false (odkazují se na stejný objekt v paměti)
x !== {};   // vrací true (nejedná se o stejný objekt)
```

Větší než (>)

Operátor `>` vrací hodnotu `true`, pokud je hodnota vlevo od něj větší než hodnota vpravo od něj. Před samotným porovnáním dochází k převodu typů:

```
0 > 1  // vrací false
1 > 1  // vrací false
2 > 1  // vrací true
2 > "" // vrací true ("" převádí na 0)
```

Větší nebo rovno (>=)

Operace obsahující operátor `>=` vrací hodnotu `true`, jestliže je hodnota vlevo od něho větší nebo rovna hodnotě vpravo od něho. Interpreter JavaScriptu převádí před tímto porovnáváním datové typy:

```
0 >= 1   // vrací false
1 >= 1   // vrací true
"1" >= 1 // vrací true ("1" převádí na 1)
```

Menší než (<)

S operátorem < obdržíme hodnotu `true` v případě, že hodnota vlevo od něj je menší než hodnota vpravo od něj. Opět dochází k převodu typů:

```
0 < 1  // vrací true
1 < 1  // vrací false
2 < 1  // vrací false
2 < "" // vrací false ("" převádí na 0)
```

Menší nebo rovno (<=)

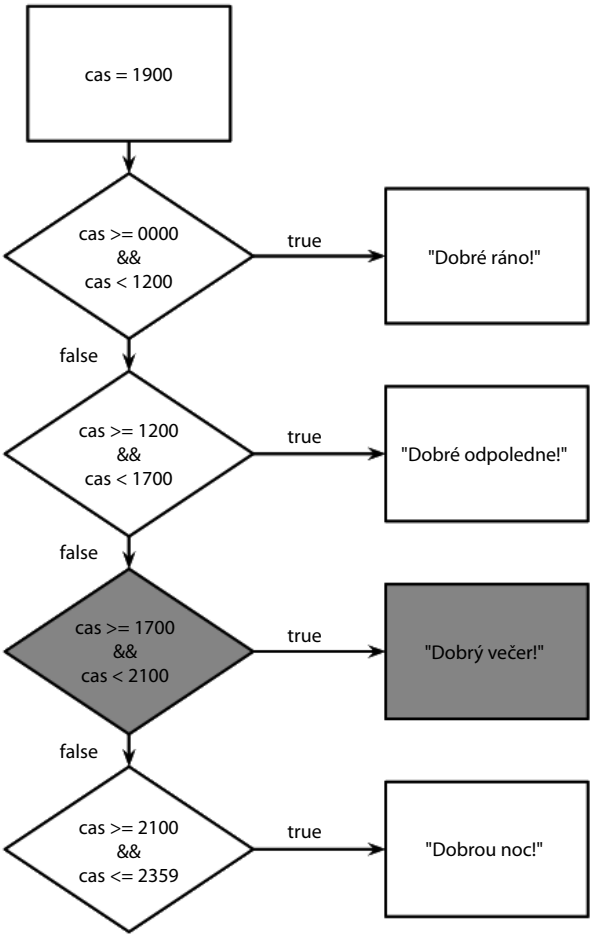
Operátor <= vrací hodnotu `true`, když je hodnota vlevo od něj menší nebo rovna hodnotě vpravo od něj. Před tímto porovnáním rovněž dochází k převodu typů:

```
0 <= 1   // vrací true
1 <= 1   // vrací true
"1" <= 1 // vrací true ("1" převádí na 1)
2 <= 1   // vrací false
"2" <= 1 // vrací false ("2" převádí na 2)
```

Řízení toku

Protože jsme se seznámili s operátory porovnávání, můžeme s nimi začít řídit tok našich programů. V programech často potřebujeme spouštět různé bloky kódu za různých podmínek. Například ráno bychom mohli zobrazovat uživatelům zprávu `"Dobré ráno!"`, odpoledne zprávu `"Dobré odpoledne!"` a večer pak zprávu `"Dobrý večer!"`. Udělali bychom to tak, že bychom vyhodnotili podmínky příkazy `if` a `else`, čímž bychom rozvětvili průchod programem.

Na obrázku 2.1 si předvedeme, jak lze řídit tok programu prostřednictvím porovnávacích operací. Pokud bude splněna podmínka, program odskočí daným směrem. V opačném případě dojde k vyhodnocení následující podmínky. V našem diagramu znázorňujícím tok programu budeme předpokládat, že čas `1900` označuje 7 hodin večer. Budeme ověřovat různé časové intervaly, abychom určili, jakou zprávu zobrazíme uživateli. Pokud bude časová hodnota větší nebo rovna `0000` (půlnoc) a současně menší než `1200` (poledne), zobrazíme zprávu `"Dobré ráno!"`. V případě, že bude větší nebo rovna `1200` a menší než `1700` (5 hodin odpoledne), zobrazíme zprávu `"Dobré odpoledne!"`. Stejným způsobem budeme postupovat, dokud neověříme všechny časové intervaly do půlnoci.



Obrázek 2.1. Tok programu

Poznámka: Symboly &&

Povšimněte si symbolů **&&** na obrázku 2.1. Jedná se o logický operátor **AND**. K dispozici je také logický operátor **||** (**OR**) a operátor **!** (**NOT**). Logické operátory převádějí jim předané hodnoty na pravdivostní hodnoty. Operátory **AND** (s významem „a současně") a **OR** (s významem „nebo") přijímají dvě hodnoty a vrací jedinou hodnotu. Operátor **NOT** (s významem „negace") obrací pravdivost hodnoty.

Níže si můžete prohlédnout několik příkladů použití operátoru **AND**. Všimněte si, že pokud interpreter JavaScriptu vyhodnotí první ze dvou hodnot jako hodnotu **false**, vrátí ji; v opačném případě vrátí druhou hodnotu:

```
true && true   // vrací true
true && false  // vrací false
false && true  // vrací false
```