EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 17300/B/2017/36086129770333188

ECMASCRIPT

Bakalárska práca

2017 **Pavol Molnár**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

ECMASCRIPT

Bakalárska práca

Študijný program: Hospodárska informatika

Študijný odbor: Hospodárska informatika

Školiace pracovisko: Katedra aplikovanej informatiky

Vedúci záverečnej práce : Ing. Miroslav Kršjak, PhD.

Bratislava 2017 **Pavol Molnár**

Čestné vyhlásenie

Čestne vyhlasujem, že záverečnú prácu som vypracoval samostatne a že som uviedol všetku použitú literatúru.

Dátum: **.....**

(podpis študenta)

Poďakovanie

Touto cestou vyslovujem úprimné poďakovanie vedúcemu záverečnej bakalárskej práce Ing. Miroslavovi Kršjakovi, PhD., za odborné vedenie, usmernenie, cenné rady a podnetné pripomienky.

ABSTRAKT

MOLNÁR, Pavol: *Ecmascript.* – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – Vedúci záverečnej práce: Ing. Miroslav Kršjak, PhD. – Bratislava: FHI EU, 2017, 69 strán.

Cieľom záverečnej práce je popis histórie a vývoja skriptovacieho jazyka ECMAScript, ktorý predstavuje štandard kódovania pre ostatné skriptovacie jazyky, spolu s porovnaním rozdielov medzi jeho jednotlivými verziami a náhľadom do jeho očakávanej budúcnosti. Práca je rozdelená do troch kapitol. Obsahuje šesť ilustrácií a desať tabuliek. Prvá kapitola je venovaná vymedzeniu dôležitých pojmov, ktoré sa týkajú danej problematiky, popisu a z historického hľadiska aj rozboru dôvodov jeho vzniku a následného vývoja. V rámci popisu evolúcie jazyka je súčasťou prvej kapitoly tiež vymenovanie spomenutých verzií a ich stručný popis. Druhá kapitola obsahuje presné definície hlavných a vedľajších cieľov práce, vrátane stanovenia metód a postupov ich dosiahnutia. Záverečná kapitola analyzuje každú verziu jazyka samostatne, porovnáva ich navzájom z hľadiska prínosov, obsahuje komentáre a jednoduché úryvky kódov pre demonštráciu využitia uvedených noviniek. Výsledkom práce je detailná mapa vývoja štandardu, vysvetlenie syntaxe a rozbor výhod a nevýhod jeho jednotlivých špecifikácií.

Kľúčové slová:

ECMAScript, ECMA-262, štandard, JavaScript, analýza

ABSTRACT

MOLNÁR, Pavol: *Ecmascript.* – University of Economics in Bratislava. Faculty of Business Informatics; Department of Applied Informatics. – Lead of diploma thesis: Ing. Miroslav Kršjak, PhD. – Bratislava: FHI EU, 2017, 69 pages.

The aim of the thesis is to describe history and development of the ECMAScript scripting language, which represents an encoding standard for other scripting languages, along with comparing the differences between its individual versions and insight into its expected future. The work is divided into three chapters. It contains six illustrations and ten tables. The first chapter is devoted to the definition of important concepts relating to the given issue, it's description, and from the historical point of view, the analysis of the reasons for its origin and subsequent development. Within the description of the evolution of language, part of the first chapter is also the listing of the mentioned versions and their brief description. The second chapter contains precise definitions of the main and secondary objectives of the work, including the determination of methods and procedures for their achievement. The final chapter analyzes each language version separately, compares it to each other in terms of benefits, contains comments and simple code extracts to demonstrate the use of the stated innovations. The result of the thesis is a detailed map of the development of the standard, explanation of the syntax and the analysis of the advantages and disadvantages of its individual specifications.

Keywords:

ECMAScript, ECMA-262, standard, JavaScript, analysis

O B S A H

Zoznam ilustrácií a zoznam tabuliek

Zoznamy použitých skratiek a značiek

IT	Informačné technológie
CMS	Content Management System - systém pre správu obsahu webových stránok
Flash	Program na vytváranie animácií ako súčasť WWW stránok
CSS	Cascading Style Sheets – tabuľky kaskádových štýlov
API	Application Programming Interface – rozhranie pre programovanie aplikácií
DOM	Document Object Model – objektovo orientovaná reprezentácia dokumentu XML alebo HTML
ISO	International Organization for Standardization – medzinárodná organizácia pre štandardizáciu
IEC	International Electrotechnical Comission – medzinárodná elektrotechnická komisia
OO	Objektovo orientované

Úvod

Takmer každý človek, Slovák či Európan, sa v dnešnej dobe dennodenne dostáva do styku s informačnými technológiami. Pojmy ako softvér, programovanie, internetová stránka či aplikácia sú dnes známe aj tým najmenším z nás. Doba v ktorej žijeme, vyznačujúca sa veľkou informatizáciou spoločnosti, ide rýchlym krokom vpred a pre nikoho z nás dnes nie je tajomstvom, že spolu s týmto napredovaním úmerne rastie aj dopyt a ponuka v oblasti už spomínaných technológii. Ich vývoj a potenciál sa môžu javiť ako neobmedzené, najmä v dôsledku neustálych inovácií a predbiehania sa svetových firiem v tom, kto prinesie na trh niečo nové. Známe ale je, že každá spoločnosť, spočiatku skupina nadšencov, ktorá v danej oblasti niečo dosiahla, musela začať od jednoduchých a malých krokov, kým postupne nevybudovala a nezdokonalila konkrétny produkt. Každý softvér, aplikácia či programovací jazyk, pomocou ktorého sú vyvíjané, museli v priebehu rokov prechádzať rôznymi modifikáciami a úpravami pre rýchlejšie a jednoduchšie dosahovanie potrebných výsledkov.

V tejto práci sa budeme venovať programovaciemu jazyku, resp. štandardu, ktorý sa volá ECMAScript, ktorý stojí za fungovaním rôznych iných jazykov a systémov a ktorého princípy využíva nie jedna dnešná webová stránka. Spracovanie tejto témy bolo zapríčinené faktom, že jeho názov všeobecnej verejnosti nič nehovorí, no stojí za funkčnosťou fenoménu, akým sú webové stránky. V súčasnosti sa medzi IT vývojármi teší väčšej a väčšej obľube a stretáva sa s čoraz väčším portfóliom príležitostí na jeho využitie.

Prvá kapitola bude akýmsi teoretickým úvodom k danej problematike. V jej podkapitolách si detailnejšie rozoberieme čo je vlastne ECMAScript, kde a ako vznikol, kde a na čo sa používa a čo všetko z historického hľadiska predchádzalo jeho terajšej podobe, teda aký bol jeho vývoj. V nasledujúcej časti si presne stanovíme cieľ, ktorým bude analýza ECMAScriptu a vymenujeme konkrétne postupy (metódy), ktoré budú v dokumente použité. V poslednej časti vykonáme analýzu konkrétnych stupňov vývoja, prezentáciu využitia konkrétnych inovácií, ktoré dané verzie priniesli a potenciálnu prognózu tohto vývoja do budúcnosti.

1 Súčasný stav riešenej problematiky doma a v zahraničí

V rámci vývoja informačných technológií a informačnej doby ako takej sme boli najmä za posledné roky svedkami zvýšenej obľúbenosti používania webových stránok. Samotný počet stránok hovorí za všetko. Kým v roku 2000 bolo podľa [8] aktívnych stránok niečo vyše 17 miliónov, v roku 2009 ich bolo niečo vyše 238 miliónov a dnes ich je dokonca viac ako 1,1 miliardy. Spomenutia hodný je pri týchto informáciách fakt, že uvedené čísla znázorňujú iba tie stránky, ktoré vykazovali/vykazujú aktivitu. Súčet aktívnych a neaktívnych/už zrušených webových stránok by teda bol ďaleko vyšší.

Záujem spočiatku viditeľne rástol iba na strane používateľov, no za posledné roky taktiež aj na strane podnikateľov/vlastníkov stránok. Internetové prehliadače zaplavili s ich obsahom naše stolové počítače, laptopy a po novom aj smartfóny, tablety a rôzne iné zariadenia. Či už sa jedná o stránky spravodajské, bankové, informačné alebo o sociálne siete, každá jedna z týchto stránok zaznamenala a stále zaznamenáva technologický posun vpred. Je to zapríčinené zvýšeným portfóliom funkcionality, ktorú vývoj webových technológií umožňuje.

Aby bolo jasnejšie, akú konkrétnu úlohu v tomto vývoji zohrával a zohráva **ECMAScript**, priblížime si najprv konkrétny programovací jazyk s názvom **JavaScript**, ktorý je s celou problematikou úzko spätý a na základe ktorého bol ECMAScript vytvorený.

1.1 Čo je JavaScript ?

JavaScript je špeciálny **skriptovací** programovací jazyk, ktorý od svojho skromného počiatku v polovici 90. rokov 20. storočia, kedy bol akýmsi doplnkom od spoločnosti Netscape k jazyku Visual Basic od spoločnosti Microsoft, vyrástol do jedného z najobľúbenejších a najpoužívanejších jazykov na svete[2].

Medzi jazykmi JavaScript a Java neexistuje žiadny vzťah. Pod podobnosť názvov týchto jazykov sa podpísal marketing a jediné čo majú okrem podobných názvov spoločné je syntax, ktorá je v oboch prípadoch založená na syntaxi predka – jazyka C. JavaScript má medzi ostatnými jazykmi úplne jedinečné postavenie, nakoľko ho v dnešnej dobe

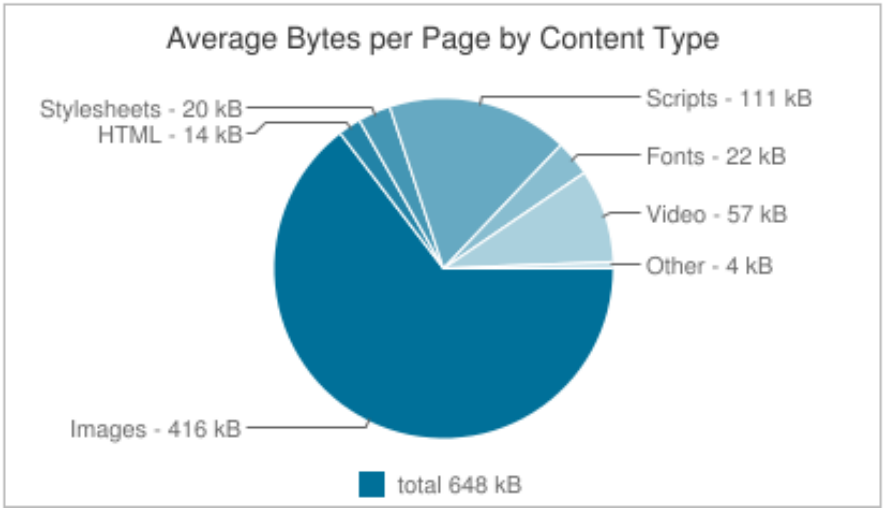
implementujú všetky moderné webové prehliadače. Využíva sa napr. pri tvorbe interaktívnych webových stránok (jeho najčastejšie využitie) a tvrdiť o ňom okrem iného môžeme, že je:

- multiplatformový (je mu úplne jedno na ako operačnom systéme je používaný),
- interpretovaný (nemusí sa kompilovať),
- objektovo orientovaný (kódy sa môžu skladať z tried, ku ktorým sa môžu definovať isté metódy a vlastnosti a z nich sa potom v konečnom dôsledku dajú vytvárať špeciálne objekty s danými vlastnosťami).

Napriek tomu, že začínal ako jednoduchý jazyk, ktorý nachádzal uplatnenie vo validácii formulárov a drobnej manipulácii s obsahom stránky, za niekoľko rokov sa pomerne veľkým spôsobom vyvinul a dnes je pomocou neho dokonca možné vytvárať rôzne bohaté klientské aplikácie. Okrem toho, v priebehu tohto obdobia začal jazyk JavaScript do určitej miery nahrádzať zásuvný modul Flash, teda prebral napr. funkciu premietania animácií, videí a iných médií v prehliadačoch.

Ak teda máme stránku vytvorenú pomocou HTML kódu, poprípade upravenú pomocou CSS, JavaScript je niečím, čo zabezpečuje, okrem iného, konkrétnu funkcionalitu daných prvkov tejto stránky. Inými slovami, tzv. „skripty“ vytvorené týmto jazykom dávajú danej webovej stránke život (sofistikovane určia správanie voči používateľovi) a umožňujú jej pomocou API rozhrania modelu DOM prepájať napr. rôzne texty, polia či premenné s inými prvkami stránky.

Ako môžeme vidieť na obrázku nižšie, priemerná webová stránka má podľa [36] z hľadiska objemu veľkosť 648kb, pričom najväčšiu časť tvoria o kapacite 416kb pochopiteľne obrázky. Spomenuté skripty však zaberajú podľa uvedeného zdroja tiež podstatnú časť objemu, v priemere 111kb na stránku, čo je takmer jedna šestina.



Obr. 1 - Priemerná veľkosť typov obsahu na webovej stránke

1.2 Od JavaScriptu k ECMAScriptu pomocou štandardizácie

1.2.1 Čo je štandardizácia ?

Štandardizácia ako pojem môže znamenať viacero vecí. Môže to byť:

- súhrn vzájomne podmienených činností a opatrení, ktoré vedú k účelnému zjednocovaniu opakujúcich sa riešení,
- súhrnné pomenovanie pre normalizáciu,
- úprava podľa jednotného vzoru[25].

Podľa spomenutého zdroja by pojem štandard mohol napr. v ekonomike alebo v technike znamenať názov pre normu (predpis určujúci isté hodnoty, vlastnosti, zloženie alebo spôsob výroby pre určitý predmet či výrobok) v danej organizácii, kontexte alebo štáte. Vo všeobecnosti je to teda akýsi ustálený vzor, resp. jednotná forma niečoho širokospektrálneho.

Pre lepšie pochopenie súvislosti medzi ECMAScriptom a JavaScriptom a úlohy, ktorú medzi nimi zohrávala štandardizácia, je potrebné pozrieť sa na históriu vývoja jazyka JavaScript.

1.2.2 Vývoj JavaScriptu a vznik ECMAScriptu

Pôvodným tvorcom jazyka JavaScript bol Brendan Eich, resp. firma Netscape Communications, ktorá si tohto programátora najala v roku 1995 za účelom zdokonalenia ich nápadu s vytvorením jazyka na spôsob jazyka Scheme pre ich prehliadač Netscape Navigator do konkrétneho produktu – konkrétnej podoby nového jazyka. Eichov produkt bol vyvíjaný pod názvom Mocha a jeho prvý oficiálny názov bol LiveScript, keď bol tento jazyk po prvý krát dodaný v beta verziách prehliadača Netscape Navigator 2.0 v septembri roku 1995. „Jazyk bol však premenovaný na JavaScript, keď bol nasadený do ďalšej beta verzie Netscape Navigator 2.0 v decembri.“[1]

Táto verzia jazyka spoločnosti Netscape predstavovala vysoký potenciál aj pre ostatné korporácie. Napríklad to bola spoločnosť Microsoft, ktorá v roku 1996 vyvinula skriptovacie technológie s názvami VBScript a JScript. JavaScriptu bol veľmi podobný práve JScript, jeho akási reverzne vyvinutá implementácia, ktorá bola súčasťou prehliadača Internet Explorer 3, patriaceho, prirodzene, spoločnosti Microsoft. Internet Explorer 3 síce ako prvý predstavoval podporu pre CSS a rôzne rozšírenia do HTML, ale v každom prípade bola táto jazyková implementácia značne odlišná tej v Netscape Navigator (neskôr Mozilla Firefox).

Tieto rozdiely predstavovali v danej dobe veľký problém najmä pre dizajnérov a programátorov pri vytváraní samostatných webových stránok, ktoré by dokázali správne pracovať v oboch týchto prehliadačoch. Situácia spočiatku viedla k rozdeleniu stránok na dve skupiny: stránky najlepšie zobrazujúce sa v prehliadači Netscape Navigator a stránky najlepšie zobrazujúce sa v prehliadači Internet Explorer. Časy týchto problémov preto dostali príznačný názov „*the browser wars*“[2], teda vojny prehliadačov. Mnohí vývojári webových stránok sa vtedy snažili, aby ich stránky fungovali v oboch týchto hlavných prehliadačoch, ale márne. Bolo teda viac ako potrebné prísť s riešením, ktoré by

[1] JavaScript [online].[preložené a cit. 3.11.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/JavaScript>.

[2] PETER, I. 2004. *History of the Internet – the Browser wars.* [online]. The Internet History Project, 2004. [preložené a cit. 3.11.2016]. Dostupné na internete: <http://www.nethistory.info/History%20of%20the%20Internet/browserwars.html>.

zabezpečilo určitú konzistenciu medzi prehliadačmi Netscape-u a Microsoftu a tiež inými implementáciami skriptov.

Obe firmy sa v roku 1996 rozhodli, že predložia svoje jazyky spoločnosti ECMA (European Computer Manufacturers Association) International a tá mala štandardizáciou vytvoriť konkrétnu špecifikáciu týchto jazykov do jednotného tvaru, najmä pre jednoduchší a rýchlejší vývoj prehliadačov a prácu s nimi. Daná špecifikácia dostala názov **ECMA-262** a začalo sa na nej pracovať v novembri roku 1996. Špecifikácia nového štandardizovaného jazyka **ECMAScript** uzrela svetlo sveta v júni roku 1997, pri uverejnení prvého vydania špecifikácie ECMA-262. Názov bol akýmsi kompromisom medzi organizáciami zapojenými do štandardizácie tohto jazyka. Pán Eich neskôr pre *Mail.mozilla.org* vtipne poznamenal, že „ECMAScript bol vždy nežiaducim obchodným názvom, lebo znel ako kožné ochorenie“[3].

Napriek tomu, že sa medzi implementácie ECMAScriptu radí okrem iných aj už spomínaný JScript alebo ActionScript, pôvodne vytvorený spoločnosťou Macromedia, JavaScript, v auguste 1998 doplnkovo štandardizovaný organizáciou ISO, bol a dodnes je jeho najznámejšou a najpoužívanejšou implementáciou. Od júna 1997 bolo pod záštitou ECMA International samozrejme vydaných aj mnoho ďalších verzií špecifikácie ECMA-262.

1.3 ECMAScript

Ak by sme sa mali zamyslieť nad tým, čo to teda ECMAScript je, z vyššie uvedených informácií vieme pomocou určitého zhrnutia povedať, že je to akási štandardizovaná forma skriptovacieho jazyka, ktorá je „bázou“[4] pre viacero implementácií a jednou z nich je práve JavaScript. Laicky by sa dalo povedať, že od vzniku ECMAScriptu platí vzťah ECMAScript = JavaScript, avšak toto tvrdenie nie je úplne správne. „Funkcie JavaScriptu tvoriace jeho jadro (syntax, typy, objekty a pod.) sú síce založené na štandarde ECMAScript, no JavaScript má zároveň rôzne iné doplňujúce

[3] EICH, B. 2006. *Will there be a suggested file suffix for es4?* [online].[preložené a cit. 3.11.2016]. Dostupné na internete: <https://mail.mozilla.org/pipermail/es-discuss/2006-October/000133.html>.

[4] What is ECMAScript ? [online].[preložené a cit. 13.11.2016]. Dostupné na internete: <http://www.computerhope.com/jargon/e/ecmascript.htm>.

funkcie, ktoré nie sú súčasťou ECMA špecifikácie.“[5] Veľkou a podstatnou črtou, ktorou sa JavaScript zásadne odlišuje od iných programovacích jazykov, je, že nepotrebuje odovzdávať nič serveru. Komunikuje iba s klientom a práve na jeho strane (v prehliadači) sú interpretované dané aplikačné kódy. Je to tzv. „client-side“ skriptovacia špecifikácia[9].

Ak by sme mali vymenovať kde všade je dnes ECMAScript prítomný, bol by to veľmi dlhý zoznam, nakoľko môžeme tvrdiť, že je to všade tam, kde sa využívajú jeho implementácie a implementácie týchto implementácií. Príkladom takejto odvodenej implementácie je napr. veľmi známe a čoraz častejšie využívané run-time (behové) prostredie Node.js[6], ktorého programy sú písané v jazyku JavaScript. Je to niečo ako rozšírená konzola pre tento jazyk, resp. pre aplikácie písané v tomto jazyku.

Štandard ECMAScript sa nepoužíva len v prostrediach, ktoré sú založené na webe. S jeho využitím sa môžeme stretnúť aj v tzv. „offline“ zóne, napr. v PDF dokumentoch, v rôznych špecifických sieťových prehliadačoch alebo aj v rôznych aplikáciách, ako sú napr. desktopové widgety[7] v laptopoch alebo smartfónoch (aplikácie a miniaplikácie pre čo najjednoduchšie ovládanie iného programu alebo zariadenia). Výnimkou nie sú ani videohry, ktoré sú rozbiehané v prostredí Node.js.

1.3.1 História a vývoj ECMAScriptu

1.3.1.1 Prvá edícia ECMA-262

Ako sme si už povedali, prvá zmienka o ECMAScripte padla v júni v roku 1997, pri uverejnení prvého vydania špecifikácie ECMA-262. Bola to základná verzia vytvorená za všeobecným účelom zabezpečenia multiplatformového programovania a zoštandardizovania skriptovania webových stránok.

---

[5] Javascript: What is ECMAScript? [online].[preložené a cit. 2.12.2016]. Dostupné na internete: <http://www.programmerinterview.com/index.php/javascript/javascript-what-is-ecmascript/>.

[6] JUNA, J. 2013. *Základní vlastnosti a výhody Node.JS.* [online]. [preložené a cit. 3.11.2016]. Dostupné na internete: <http://www.nodejs.cz/vlastnosti-a-vyhody-nodejs/>.

[7] DRHLÍK, K. 2015. *Android pro začátečníky #7 – Co je to widget a jak jej používat?* [online]. [preložené a cit. 3.11.2016]. Dostupné na internete: <https://www.svetandroida.cz/android-pro-zacatecniky-widget-201508>.

1.3.1.2 Druhá edícia ECMA-262

Vydanie ďalšej verzie na seba nenechalo dlho čakať, prišlo už o rok, v júni 1998. Verzia s názvom ECMA-262 (2nd Edition) alebo ECMAScript 2 (ES2) so sebou priniesla iba menšie zmeny, najmä za účelom udržania tejto špecifikácie v medzinárodných normách ISO/IEC 16262.

1.3.1.3 Tretia edícia ECMA-262

Uvoľnenie ECMAScript jazykovej špecifikácie číslo 3 nasledovalo v decembri roku 1999 a táto špecifikácia predstavovala akúsi bázu pre moderné využitie skriptovacích jazykov. Prielomovými novinkami tejto verzie boli najmä:

- regulárne výrazy,
- zlepšená ovládateľnosť reťazcov,
- nové riadiace príkazy (napr. nový príkaz cyklu),
- tzv. „try/catch“ spracovanie výnimiek,
- prísnejšia definícia syntaktických chýb,
- formátovanie pre číselné výstupy a iné.

1.3.1.4 ECMA-357 (E4X)

Cieľom organizácie ECMA International ale nebol iba vývoj štandardov ECMA-262. V júni roku 2004 bol zverejnený špeciálny štandard s názvom ECMA-357, ktorý mal predstavovať akési definujúce rozšírenie k ECMAScriptu, známe ako ECMAScript pre XML (v skratke: E4X). Jeho cieľom bolo (ako jeho názov sčasti napovedá) poskytnúť alternatívu k DOM rozhraniu, ktorá používa jednoduchšiu syntax pre spracovanie a prístup k dokumentom typu XML[3][31]. ECMA tiež definovala tzv. ‚kompaktný profil‘ pre ECMAScript, známy pod anglickou skratkou ES-CP, alebo v niektorých zdrojoch pod názvom ECMA-327, ktorý bol navrhnutý pre zariadenia s obmedzenými zdrojmi[12].

V marci roku 2005 bol uverejnený v článku uvedeného slovenského zdroja tento text: „Vytváranie stránok pomocou webových štandardov umožňuje znižovať náklady na

ich tvorbu a zároveň zjednodušuje dostupnosť uvedených stránok.“[8] Z textu vyplýva, že už v danej dobe vydané verzie ECMA-262 plne podporovali žiadanú funkcionalitu stránok a s veľkou pravdepodobnosťou sa využívali vo väčšine z nich. Webovú stránku ako takú bolo už vtedy možné rozdeliť vďaka rôznym (nielen skriptovacím) webovým štandardom na tri komponenty, ktorými boli:

- Štruktúra (HTML, XHTML, XML),
- Prezentácia (CSS1, CSS2),
- Správanie (ECMAScript, DOM).

1.3.1.5 Štvrtá edícia ECMA-262 (nevydaná)

Aj napriek veľkej obľube, ktorej sa ECMAScript počas nasledujúcich rokov tešil, sa nepodarilo úspešne dotiahnuť do konca projekt ECMAScript 4, teda 4. edíciu špecifikácie ECMA-262. Dôvodmi, kvôli ktorým nebola táto špecifikácia vydaná, boli rôzne debaty medzi spoločnosťami Netscape a Microsoft týkajúce sa zložitosti jazyka, ktoré však nedosiahli konečné kompromisné stanovisko o tom, ako by finálna verzia mala vyzerať a práve preto sa v tomto dôsledku pozornosť dočasne presunula na už spomínaný štandard ECMAScript pre XML (E4X). Tieto otvorené debaty sa v danom momente stali najsledovanejšou verejnou diskusiou, resp. sledovanou témou v informatike.

Spočiatku navrhované štvrté vydanie ECMA-262 (ECMAScript 4 alebo ES4) malo byť po dlhšej dobe prvou hlavnou aktualizáciou pre ECMAScript, nakoľko posledné (tretie) vydanie bolo uverejnené až v roku 1999. Prehľad o jazyku bol vydaný pracovnou skupinou v októbri roku 2007 a dokončenie samotnej špecifikácie bolo stanovené na október roku 2008. Avšak, v auguste roku 2008 bolo vydanie tejto edície zamietnuté z vyššie uvedených dôvodov a spolu s týmto nariadením bolo kompletne zrušených mnoho funkcií, ktoré mali byť vydané spolu s danou verziou špecifikácie. Niektoré z funkcií pre ES4 boli využité v návrhu projektu s kódovým označením ECMAScript Harmony, ktorý v danom čase zahŕňal tieto novinky:

[8] MAKULOVÁ, S. 2005. *Prečo je potrebné dodržiavať štandardy World Wide Web konzorcia* [online].[cit. 3.12.2016]. Dostupné na internete: <http://itlib.cvtisr.sk/archiv/2005/3/preco-je-potrebne-dodrziavat-standardy-world-wide-web-konzorcia.html?page_id=1748>.

- triedy,
- modulárny systém,
- voliteľné anotácie písania a statické písanie, pravdepodobne využívajúce systém štrukturálneho písania,
- generátory,
- deštrukturalizačné priraďovanie,
- algebraické dátové typy.

Spočiatku tieto funkcie napriek skorému predstaveniu neboli zavedené do používania. Spomenutý návrh projektu bol implementovaný až v ECMAScripte 6.

Verzia ES3 obsahovala množstvo neošetrených pravidiel a zavedených prístupov. V nadväznosti na ich príchod boli uvedené niektoré chyby verzie ES3, najmä pre ich napravenie v očakávanej 4. verzii ECMA-262. Opravy týchto chýb a rôzne iné boli napokon zahrnuté vo verzií ECMAScript 5, teda až v piatej edícii štandardu ECMA-262[12].

1.3.1.6 Piata edícia ECMA-262

Popri spomínaných udalostiach o vývoji a následnom zamietnutí návrhu 4. verzie ECMAScriptu (v podaní tímu spoločnosti ECMA) sa iný tím, tvorený spoločnosťami Yahoo, Microsoft, Google a ďalšími, pokúšal v rovnakom čase o čosi podobné, teda o návrh novej verzie tohto skriptovacieho štandardu. Jednalo sa síce taktiež o v poradí štvrtú verziu, ale nakoľko našla táto skupina v pôvodnom ES3 viac nedostatkov ako tím ECMA, pracovný názov 4. verzie v tomto prípade nebol ECMAScript 4, ale ECMAScript 3.1, teda sa malo jednať o akúsi aktualizáciu verzie ES3. Toto vydanie sa zameriavalo primárne na bezpečnosť a knižničné aktualizácie s obzvlášť veľkým dôrazom na kompatibilitu. Neskôr sa ukáže, že práve vývoj so spomínaným dôrazom na kompatibilitu mal veľký vplyv na vývoj ECMAScriptu ako taký.

Krátky čas po už spomenutých rozbrojoch vo veci schválenia verzie ECMAScriptu 4 sa stretli oba tímy, ktoré pracovali na novej verzii tohto skriptovacieho štandardu – tím

ECMAScript 4, ktorý so svojim návrhom neuspel a tím ECMAScript 3.1, ktorý opravil chyby v tretej verzii. Ani jeden z týchto tímov nechcel opustiť svoj projekt, preto bol výsledkom tohto stretnutia istý kompromis. Na oboch verziách sa malo paralelne pracovať aj naďalej, pričom práca medzi týmito tímami mala byť navzájom skoordinovaná, aby zostal zabezpečený istý vzťah medzi danými verziami, a síce že verzia ECMAScript 3.1 mala zostať podmnožinou verzie ES4[12].

Rozdielne filozofie oboch tímov sa však po krátkom čase jasne prejavili v ďalších nezhodách a za následok to malo opätovné zrušenie daného podmnožinového pravidla. Navyše ostala vo vzduchu visieť otázka, či budú vôbec odporcovia ECMASriptu 4 niekedy v budúcnosti podporovať alebo implementovať daný štandard.

Podľa [12] sa po vyše roku od tejto nezhody vo veci vývoja a budúcnosti ECMAScriptu v rámci tzv. Ecma technického výboru 39 (Ecma Technical Committee 39, skratka: TC39) oba tímy opäť dohodli na novom kompromisnom riešení. Bol to júl roku 2008, kedy Brendan Eich verejne oznámil, že sa Ecma TC39 zameria na prácu na verzii ECMAScript 3.1 (neskôr premenovanej na ECMAScript 5) s plnou spoluprácou všetkých zúčastnených strán. V apríli roku 2009 publikoval výbor Ecma TC39 konečný návrh 5. edície a oznámil, že jej testovanie sa podľa predbežného plánu skompletizuje približne v polovici júla príslušného roka. 3. decembra bola po všetkých nezhodách a sedeniach konečne úspešne vydaná piata edícia špecifikácie ECMA-262.

ECMAScript 5 (ES5) priniesol na svetlo tzv. **striktný mód**, vymoženosť zameranú na poskytnutie dôkladnejšej kontroly chýb a tiež na rôzne iné veci, o ktorých si povieme v praktickej časti práce.

Okrem už spomenutých noviniek ES5 boli prínosom tejto verzie aj:

- nové funkcie s názvami „getters“ a „setters“ (programovacie postupy, ktoré majú mnoho spoločného s prístupom OO programovania),

- špeciálna knižničná podpora pre JSON („tzv. ‚open-standard' formát, ktorý využíva ľudsky čitateľný text na prenos dátových objektov, ktoré pozostávajú z párov typu atribút/hodnota"[9]),

- reflexia na vlastnosti objektov.

Okrem vyššie uvedených funkcií a vymožeností sa dal postrehnúť rozdiel tiež v rozsahu implementácií ECMAScriptu, teda v rozsahu jeho využitia (viď.

Tab. 1 – Aplikácie a implementácie ECMAScriptu 5

Aplikácia	Implementácia (posledná verzia)	ECMAScript edícia
Mozilla Firefox, (Gecko layout engine), SpiderMonkey, Rhino	JavaScript 1.8.5	ECMA-262, edícia 5 a funkcie z nasledujúcej edície
Google Chrome (V8 engine)	JavaScript	ECMA-262, edícia 5
Safari (Nitro engine)	JavaScript	ECMA-262, edícia 5.1
Internet Explorer (Trident layout engine)	JScript 9.0	ECMA-262, edícia 5
Java	Nashorn 1.8.0	ECMA-262, edícia 5.1
Opera	ECMAScript	ECMA-262, edícia 5
RemObjects Script pre .NET	ECMAScript	ECMA-262, edícia 5
Appweb Web Server, Samba 4	Ejscript 0.9.9	ECMA-262, edícia 5.1
Microsoft .NET Framework	JScript .NET 8.0	ECMA-262, edícia 3
Adobe Flash a Adobe Flex	ActionScript 3	ECMA-262, edícia 3
Adobe Acrobat	JavaScript 1.7	ECMA-262, edícia 3
Adobe Creative Suite: InDesign, Illustrator, Photoshop, Bridge, After Effects, Premiere Pro	ExtendScript	ECMA-262, edícia 3
OpenLazslo	JavaScript	ECMA-262, edícia 3
CriScript, JScript pre herné platformy	CriScript 0.91.0	ECMA-262, edícia 3
iCab	InScript 3.22	ECMA-262, edícia 3
Max/MSP	JavaScript 1.5	ECMA-262, edícia 3
ANT Galio 3	JavaScript 1.5	ECMA-262, edícia 3
KDE	QtScript	ECMA-262, edícia 3
iné JavaScript aplikácie	TypeScript	ECMA-262, edícia 3, 5 a funkcie z nasledujúcej edície

[9] JSON [online].[preložené a cit. 3.12.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/JSON>.

Verzia ES5 bola veľmi kvalitným skriptovacím štandardom, nakoľko sa nenašli počas viacerých nasledujúcich rokov žiadne radikálne nedostatky. Počas dlhšieho obdobia nebolo nutné vyvinúť ďalšiu verziu, iba ak v dôsledku implementácie nových prvkov a inovácií v oblasti skriptov. Aktualizovaná verzia ECMAScriptu síce uzrela svetlo sveta už v júni roku 2011, no jej uverejnenie bolo zapríčinené najmä vydaním novej medzinárodnej normy ISO/IEC 16262:2011, s ktorou mala byť v súlade, podobne ako napr. druhá edícia ECMA-262, ktorá vznikla za rovnakým účelom. Nakoľko sa jednalo iba o aktualizáciu pre súlad so spomenutou normou, táto aktualizovaná verzia 5. edície dostala názov ECMAScript 5.1.

1.3.1.7 Šiesta edícia ECMA-262

Ďalšia verzia ECMAScriptu bola dokončená až v júni roku 2015. Jej oficiálny názov bol ECMAScript 2015, ale na základe zaužívaného zvyku v oblasti pomenovaní jednotlivých špecifikácií jej prischol názov ECMAScript 6, resp. ES6. Bol to opäť veľký krok vpred, ktorý už ale nedokázal zabrániť niektorým komplikáciám vo veci podpory prehliadačov. Tím, ktorý vyvíjal túto verziu preto prišiel s nápadom použiť na vyriešenie tohto problému tzv. „transpiler“, ktorý by sme mohli v slovenskom jazyku jednoslovne nazvať transkompilátorom. Jednalo sa o akýsi kompilátor, ktorého úlohou bolo uvedený ES6 kód „transpilovať“ do ES5 kódu, ktorý ma konzistentnejšiu podporu naprieč webovými prehliadačmi. Toto riešenie malo byť dočasné, kým sa webové prehliadače tiež nevyvinú a nezaktualizujú do foriem, ktoré budú viac podporovať ECMAScript 6. Funguje to asi tak, že transpiler vezme zdrojový kód programu napísaného v jednom programovacom jazyku ako vstup a vyprodukuje kód v inom programovacom jazyku ekvivalentný tomu prvému ako výstup. Napr. môže vykonať ekvivalentný preklad programu z jazyka Pascal na C[34].

A prečo vlastne vznikli problémy s podporou medzi prehliadačmi? Bolo to zapríčinené novými funkciami, ktoré nová verzia priniesla. „Jednou z hlavných noviniek ECMAScriptu 6 bola špeciálna syntax, ktorá umožnila písanie zložitejších aplikácií,

vrátane rôznych tried a modulov, definujúc ich sémanticky, za rovnakých podmienok ako už spomenutý striktný mód verzie ECMAScript 5.“[10]

Zároveň so zavedením transkompilátora bolo pridaných množstvo nových vymožeností, ktoré mali programátorom skriptov uľahčiť prácu. Jednalo sa o novinky v nasledujúcich oblastiach programovania:

- premenné,
- polia,
- narábanie s reťazcami,
- parametre.

Novinkami, o ktorých si povieme viac v analytickej časti práce, boli tiež nové funkcie polí, triedy a moduly. Do skriptovania boli zavedené aj dva úplne nové pojmy: operátor šírenia a deštrukturalizácia.

Ďalšími novými funkciami tejto štandardizovanej verzie boli napr. aj:

- iterátory („objekty, ktoré umožňujú programátorom prechádzať úložiská dát, najmä zoznamy“[11]),
- binárne dáta,
- typovanie polí,
- kolekcie (mapy, sady a slabé mapy),
- sľuby,
- číselné a matematické vylepšenia a iné.

[10] KAPPERT, L. 2015. *ECMAScript 6 (ES6): What's New In The Next Version Of JavaScript.* [online].[preložené a cit. 3.12.2016]. Dostupné na internete: <https://www.smashingmagazine.com/2015/10/es6-whats-new-next-version-javascript/>.

[11] Iterator [online].[preložené a cit. 3.12.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/Iterator>.

Zoznam noviniek bol teda naozaj rozsiahly a opäť priniesol funkcie veľmi podobné, ak nie rovnaké, ako prinášajú moderné OO jazyky (Java, C++ a pod.).

1.3.1.8 Siedma edícia ECMA-262

Doteraz posledná verzia ECMAScriptu bola dokončená v júni roku 2016. Jej oficiálny názov je ECMAScript 2016, ale nakoľko je to siedma verzia špecifikácie ECMA-262, je opäť možné ju pomenovať poradovým číslom 7, teda ECMAScript 7 (ES7).

Tvorcovia tejto verzie mali pri nej jasný cieľ – pokračovať v témach ako sú napr. jazyková reforma, izolácia kódu, ovládanie efektov či sprístupnenie knižničných nástrojov z verzie ES6. Siedma verzia zahŕňa dve nové funkcie:

- obsluhu umocňovania pomocou operátora umocňovania (**),
- funkciu *Array.prototype.includes*, ktorá dokáže určiť, či vybrané pole obsahuje vybraný prvok.

Spoločnosť ECMA International a jej tím vývojárov to za posledné roky nemal jednoduché. Ľudia, ktorí pracovali na vývoji špecifikácie ECMA-262 (a rôznych iných) sa museli vedieť vysporiadať s mnohými problémami, nedostatkami a obmedzeniami, s ktorými prišli do styku a zároveň sa snažiť vytvoriť pomocou štandardizácie lepšie a lepšie záchytné body pre rôzne skriptovacie jazyky a pre ich samostatný následný vývoj. Dalo by sa povedať, že výstup ich práce je výstupom takmer 15 ročného úsilia o vývoj tohto štandardu.

1.3.1.9 Ôsma edícia ECMA-262 (zatiaľ oficiálne nevydaná)

V januári a vo februári roku 2017, teda už v čase písania tejto práce, sa začali na internete objavovať rôzne informácie o príchode ďalšieho dielu ECMAScriptu. Podľa neoficiálneho, no pomerne dôveryhodného zdroja by mala daná verzia obsahovať tieto vymoženosti:

- nové typy hodnôt,

- prevod tzv. „zero-copy“[12] binárnych dát,
- rôzne číselné a matematické vylepšenia,
- syntaktickú integráciu so sľubmi (nové prvky *async* a *await*),
- pozorovateľné toky,
- nové *SIMD* typy,
- záznamy a n-tice,
- vlastnosti tried a inštancií,
- lepšie metaprogramovanie s triedami, a
- črty[12].

Väčšina týchto noviniek nám zatiaľ z praktického hľadiska nič konkrétne nehovorí, ale určite nám napovedá niečo o smerovaní vývoja skriptovania. Táto edícia totiž, napriek rôznym informáciám o jej uvedení v apríli, zatiaľ nevyšla.

1.3.2 Implementácia ECMAScriptu

ECMAScript je podporovaný v mnohých aplikáciách, najmä vo webových prehliadačoch, v ktorých je v prevažnej väčšine implementovaný JavaScriptom. Každý program, ktorý spustí zdrojový kód napísaný pod ECMAScript jazykovou normou (napr. v spomenutom JavaScripte) sa nazýva „ECMAScript engine“, v preklade motor ECMAScriptu[33]. Tieto implementácie niekedy môžu zahŕňať isté rozšírenia do jazyka, alebo do knižnice daného štandardu a súvisiacich API rozhraní, tak ako DOM, špecifikovaný World Wide Web Konzorciom (W3C).

Engine-ov štandardu ECMAScript existuje v dnešnej dobe pomerne veľa. Niektoré sú „open-source“, teda voľne dostupné a stiahnuteľné na internete, iné sú zasa chránené autorským zákonom a voľne neprístupné. Ako sme si už spomenuli, väčšina z nich sa

[12] ECMAScript [online].[preložené a cit. 30.4.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/ECMAScript>.

využíva vo webových prehliadačoch. Počítať by sa dali na desiatky a keďže sa niektoré využívajú len v špecifických prostrediach, bližšie si povieme iba o momentálne najpoužívanejších a najznámejších z nich, pričom si ich rozdelíme do dvoch skupín.

1.3.2.1 Prvá skupina

V tejto skupine si povieme o niekoľkých ECMAScript engine-och, patriacich do tzv. novej generácie engine-ov pre webové prehliadače. Spoločnou vlastnosťou tejto skupiny je implementovanie už spomínanej *just-in-time* kompilácie a variácie tohto nápadu. Jej výhodou je vhodnejšia práca s webovými aplikáciami napísanými v JavaScripte[33].

Ako už vieme, JavaScript je interpretovaný jazyk a potrebuje teda nejaký interpret, určité behové prostredie, ktoré sa postará o prevedenie užívateľského skriptu. K najznámejším kedysi patrili iba **SpiderMonkey**, ktorý sa nachádza v Mozilla Gecko aplikáciách (vrátane Firefox-u) a **JavaScriptCore**, ktorý bol tiež poznaný pod názvami Nitro, SquirrelFish či SquirrelFish Extreme a dodnes sa používa vo WebKit projektoch a taktiež v aplikácii Safari. Takmer raketový vzostup popularity ale zažil aj engine s názvom **V8**, vytvorený spoločnosťou Google pre prehliadač Chrome. Ten totiž totálne obrátil do tej doby pomerne vyrovnané rebríčky rýchlosti jednotlivých JavaScript interpreterov a svojou rýchlosťou nechal celú konkurenciu ďaleko za sebou. Navyše je jeho kód otvorený (open-source) a je teda pomerne ľahké ho zabudovať do iných aplikácií. Nie je divu, že práve nad týmto engine-om vzniklo množstvo aplikácií, do ktorých sa radia aj rôzne serverové engine-y. Najznámejšie z nich sú prehliadače Opera a spomínaný Google Chrome a tiež behové prostredie Node.js, ktoré pre nás už tiež nie je neznámym pojmom[12][14][33]. Spomenúť je taktiež treba aj engine jazyka JavaScript použitý v prehliadači Microsoft Edge, ktorý sa volá **Chakra**[33].

1.3.2.2 Druhá skupina

Nasledujúce engine-y používajú behové interpretery, ktoré nekompilujú do natívneho strojového kódu a vo všeobecnosti fungujú pomalšie:

- **Rhino** (jeden z niekoľkých JavaScript engine-ov od spoločnosti Mozilla, ktoré používajú Java platformu),
- **Espruino** (rozmerovo veľmi malý interpreter špecifický pre mikroprocesory),
- **MuJS** (knižnica navrhnutá pre zabudovanie do iného softvéru za účelom rozšírenia jeho skriptovacích schopností).

2 Cieľ práce, metodika práce a metódy skúmania

Hlavným cieľom záverečnej práce je detailný popis jazyka ECMAScript, teda v dnešnej dobe takmer všade prítomného a zároveň podľa názvu nie moc známeho štandardu pre skriptovacie jazyky. Súčasťou práce ako celku bude detailná mapa vývoja ECMAScriptu od jeho počiatku až po súčasnosť, vrátane ukážok a praktických príkladov jeho využitia za pomoci prezentácie skriptovacích kódov. Pod detailnou mapou rozumieme nielen popis jednotlivých verzií, ale aj ich vzájomné prechody a porovnanie. Po prečítaní práce by malo byť každému jasné, čo je ECMAScript, čo konkrétne podnietilo jeho vznik, aký bol jeho vývoj, resp. aký prínos mala každá novšia verzia, ako funguje a aký význam má pre nás ECMAScript v dnešnej dobe. Popis týchto udalostí zároveň umožní čitateľom sčasti nahliadnuť do istej fázy vývoja webových technológií ako takých. V závere práce sa tiež pozrieme na budúcnosť tohto štandardu, teda na novinky, ktoré môžeme očakávať v blízkej dobe.

Cieľom tejto práce je tiež to, aby ona sama predstavovala akýsi teoretický základ a ucelený úvod do danej problematiky pre ďalšie spracovanie, ako sú napr. bakalárske a diplomové práce, články, publikácie a pod.

Čo sa týka metodiky práce, objektom skúmania bude teda samotný štandard, ktorý môžeme z pohľadu programovania tiež chápať ako programovací jazyk (má svoje pravidlá syntax a pod.). Postup zvolíme veľmi podobný ako pri prvej časti práce, teda pôjdeme od prvej edície štandardu až po súčasnú s tým, že pri každej edícií pomocou analýzy nájdeme najdôležitejšie alebo najzaujímavejšie informácie a tie následne odprezentujeme. Okrem analýzy použijeme aj vedeckú metódu s názvom komparatívna analýza, ktorou vykonáme spomenuté porovnanie jednotlivých prechodov medzi verziami. Komparatívnou analýzou budeme tiež získavať užitočné informácie, ale budeme sa ňou sústrediť primárne na diferenciáciu konkrétnych dvoch verzií a zvýrazňovať ich odlišnosti. Tým môžeme chápať napr. popis noviniek v jazyku, príklady využitia týchto noviniek v programovaní, vysvetlenie syntaxe, pokrytie komerčnými implementáciami a pod.

Nakoľko sa nám nepodarilo zohnať zdroje v tlačenej forme, ktoré by boli zamerané výlučne na samotný ECMAScript, pri analýze aj komparatívnej analýze budeme prevažne čerpať z online zdrojov.

3 Výsledky práce a diskusia

V tejto časti práce si jednotlivé verzie, o ktorých sme hovorili, priblížime praktickým spôsobom, teda nahliadneme do noviniek, ktoré jednotlivé verzie priniesli, najmä z hľadiska kódov. Všetky príklady využitia konkrétnych noviniek v programovaní nebudú prezentované ako kompletné programy, ale ako akési úryvky reálnych kódov.

3.1 ECMAScript 1 a ECMAScript 2

Prvé dve verzie štandardu ECMA-262 boli z funkčného hľadiska identické, preto sme sa rozhodli ich v tomto prípade spojiť. Jediným rozdielom bol fakt, že druhá verzia bola vydaná z dôvodu súladu s medzinárodnou normou ISO/IEC 16262.

Oba skriptovacie štandardy (ES1, ES2) boli založené na už existujúcich skriptovacích jazykoch, teda mali zabezpečiť vytvorenie akéhosi základu pre používanie skriptov vo webových stránkach. Všetky vtedajšie a zároveň v budúcnosti vydané verzie skriptovacích jazykov spĺňajúce štandard ECMA-262 mali poskytovať a podporovať všetky v danom štandarde popísané **pravidlá, typy, hodnoty, objekty, vlastnosti, funkcie a pod**. Výnimkou nebola ani **syntax**, ktorá mala byť jednotná. Nakoľko boli tieto verzie úplným začiatkom štandardizácie skriptovania, ich novinkami bolo vlastne všetko, čo ECMAScript na počiatku obsahoval, resp. definoval. Nakoľko nie je možné v tejto práci obsiahnuť a popísať všetky komponenty ECMAScript špecifikácie, v nasledujúcich riadkoch sa pokúsim čo najstručnejšie zahrnúť všetky podstatné časti týchto verzií a vytvoriť o tom určitý prehľad.

3.1.1 Definície základných pojmov

Pred charakteristikou jednotlivých komponentov štandardu je treba vysvetliť si pomocou neformálnych definícií niektoré často používané a ECMAScriptom štandardizované pojmy, ku ktorým sa podrobnejšie dostaneme neskôr.

Tab. 2 - Definície základných pojmov v ECMAScripte

Názov	Definícia (poznámka)
type	- dátový typ
primitive value	- primitívna hodnota je údaj reprezentovaný na najnižšej úrovni implementácie jazyka (**Undefined**, **Null**, **Boolean**, **Number** alebo **String**)
object	- neusporiadaná kolekcia vlastností, z ktorých každá môže a zároveň nemusí obsahovať atribúty s ňou spojené
constructor	- inštancia funkcie, ktorá vytvára a inicializuje objekty (každý konštruktor má pridružený tzv. *prototype object*, ktorý umožňuje dedičnosť objektov a zdieľanie ich vlastností)
prototype	- keď konštruktor vytvorí objekt, ten objekt implicitne odkazuje na konštruktoru pridružený prototyp, na ktorý sa môže odkazovať program pomocou výrazu `constructor.prototype` a tak môžu byť všetky prototypu pridelené vlastnosti zdieľané skrze dedičnosť
native object	- natívny objekt je objekt nezávislý od hostiteľského prostredia, priamo definovaný ECMAScript špecifikáciou
built-in object	- vstavaný objekt je natívny objekt, ktorý je prítomný na začiatku vykonávania ECMAScript programu - štandardné vstavané objekty sú definované ECMAScriptom, no aj konkrétna ECMAScript implementácia (napr. JavaScript) si môže pre svoje potreby špecifikovať a definovať ostatné
host object	- objekt dodávaný hosťovským prostredím (všetky objekty ktoré nie sú natívne)
undefined value	- nedefinovaná hodnota je primitívna hodnota použitá v premenných bez priradenej hodnoty
null value	- primitívna hodnota, ktorá v programe reprezentuje 0, prázdnu alebo neexistujúcu referenciu
boolean value	- hodnota true/false
Boolean	- dátový typ reprezentujúci logickú entitu s **boolean** hodnotou
boolean object	- objekt vytvoreným vstavaným Boolean konštruktorom obsahujúci boolean hodnotu ako argument
string value	- ľubovoľné konečné za sebou idúce poradie Unicode znakov
string object	- objekt vytvoreným vstavaným String konštruktorom obsahujúci reťazec znakov ako argument
number value	- číselná hodnota
number object	- objekt vytvoreným vstavaným Number konštruktorom obsahujúci číslo ako argument
Infinity	- primitívna hodnota nekonečno
NaN	- hodnota, ktorá „nie je číslom“ (definovaná IEEE Štandardom) - tieto hodnoty sú z hľadiska ECMAScriptu primárne nerozoznateľné (rozoznať ich je možné až dnes a iba v konkrétnych implementáciách)

3.1.2 Dátové typy

Prvými dátovými typmi podľa ECMAScriptu boli tieto druhy: **Undefined**, **Null**, **Boolean**, **String**, **Number**, **Object**, **Reference**, **List** a **Completion**. Všetky sa používajú dodnes, pričom posledné tri sa používajú iba ako medzivýsledky vyhodnotenia výrazov a nemôžu byť uložené do vlastností objektov.

Tab. 3 - Dátové typy

Názov	Definícia (poznámka)
Undefined	- dátový typ, ktorý má iba jednu hodnotu (nie viac), hodnotu **undefined** - každá premenná bez priradenej hodnoty je tohto typu
Null	- dátový typ obsahujúci jednu hodnotu **null** (tento typ sa používa ak chceme deklarovať premennú a zároveň zdôrazniť, že nemá žiadnu hodnotu, na rozdiel od hodnoty **undefined**, kde proste hodnota chýba)
Boolean	- dátový typ reprezentujúci logickú entitu s **boolean** hodnotou, teda hodnotu true/false
String	- dátový typ obsahujúci **string** hodnotu - je to množina všetkých konečných usporiadaných sekvencií nula alebo viac Unicode znakov
Number	- súbor číselných hodnôt vrátane špeciálnych **NaN** hodnôt, +∞ a -∞
Object	- neusporiadaná kolekcia vlastností - každá vlastnosť sa skladá z mena, hodnoty a množiny atribútov

Špecifikáciou ECMA-262 boli definované aj tzv. konverzie dátových typov (napr. zmena hodnoty číslo na hodnotu reťazec pomocou funkcie **ToString()**, pričom v zátvorke je očakávané zadať pôvodný prvok, teda prvok konverzie).

3.1.3 Operátory

Okrem toho boli definované aj **vstavané operátory**:

Tab. 4 - Operátory

Operátory	Znaky	Poznámka
Unárne	delete, void, typeof, ++, --, +, -, ~, !	- unárne + a – vyjadrujú kladnosť/zápornosť - ~ je bitový NOT operátor - ! je logický NOT operátor
Multiplikačné	*, /, %	
Aditívne	+, -	- buď vykonajú zreťazenie (pri typoch string) alebo numerické sčítanie (pri typoch number), pričom pri konfliktnej situácií (operátor pri rozličných typoch dát) je primárne zreťazenie
Bitového posunu	<<, >>, >>>	
Relačné	<, >, <=, =>	
Rovnosti	==, !=	
Binárne bitové	&, ^, |	
Binárne logické	&&, ||	
Podmienený	? :	- použitie: `(podmienka) ? urob1 : urob2`
Priradenia	=, *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=	- vykoná konkrétnu operáciu medzi ľavým aj pravým výrazom a výsledkom prepíše pôvodný výraz vľavo
Čiarka	,	- napr. pri napĺňaní dát do polí (viď nižšie)

3.1.3.1 Operátor new a polia

Všetky dátové typy, ktoré sme si ukázali sa dajú vkladať do premenných. Tak ako aj v iných programovacích jazykoch, aj v skriptovacích sa pracovalo a dodnes pracuje s prvkami, ktoré sa nazývajú **polia**. Tie umožňujú prácu s viacerými prvkami, teda s celou skupinou dát.

Polia sa spočiatku dali vytvárať len zavolaním konštruktora **Array()** a vytvorením jeho novej inštancie pomocou operátora **new**, ktorú program vložil do istej premennej:

```
var pole = new Array();
```

Takýmto spôsobom sa pomocou kľúčového slova *new* dali vytvárať a dodnes sa vytvárajú akékoľvek inštancie ľubovoľného konštruktora.

Tento spôsob vytvárania poľa mal však isté nevýhody. Konštruktor *Array()* bolo totiž možné prepísať, prípadne nahradiť nebezpečným kódom (napr. funkciou), ktorý by predstieral, že vytvára pole, ale v skutočnosti by napr. odosielal dáta cudziemu serveru[2]. Spoločnosť ECMA International tento problém vyriešila v nasledujúcej verzii, definíciou prvku s názvom **literál poľa**. Ten mal pre programátorov skriptov predstavovať zjednodušený a omnoho účinnejší a bezpečnejší spôsob tvorby polí jednoducho, pomocou hranatých zátvoriek:

```
var pole = [];
```

Tento spôsob sa používa dodnes. Pri vytváraní sa jednotlivé prvky oddeľujú čiarkou a dá sa k nim pristupovať pomocou ich indexov. Polia dokonca môžu nadobúdať aj prvky rôznych dátových typov (vrátane vnorených polí):

```
var pole = [1, "auto", undefined, null, []]
```

3.1.4 Cykly

„Ako podmienku môžeme vyhodnotiť akýkoľvek výraz, ktorý sa dá vyhodnotiť ako pravdivostná hodnota *true* alebo *false*“[13]. Je samozrejmé, že aj pred vydaním ECMAScriptu používali jednotlivé skriptovacie jazyky skripty, nakoľko sú základom podmieneného programovania ako takého. Po vydaní ECMAScriptu však musel každý skriptovací jazyk obsahovať minimálne tieto 2 a používať ich nasledujúcimi spôsobmi.

Tab. 5 - Cykly

Cyklus	Použitie	Príklad
while	`while (podmienka) { príkazy }`	`while (x > 0) { // x = 3 console.log(x); //výpis do konzoly x--; }`
for	`for (inicializácia; podmienka; konečný výraz) { príkazy }`	`for (var i = 0; i < 3; i++) { console.log('Máme ' + x + '.'); //x zmení na reťazec a zreťazí do vety }`

[13] BAŠE, O. 2014. *JavaScript Okamžite.* Brno : Computer Press, 2014. s. 82. ISBN 978-80-251-4163-2

3.1.5 Objekty

Základom ECMAScriptu bolo a dodnes je OO programovanie. To znamená, že základný jazyk a hostiteľské zariadenia sú poskytované objektami a samotný program je zoskupením komunikujúcich objektov.

Každý ECMAScript objekt je neusporiadanou kolekciou vlastností, pričom každá má 0 alebo viac atribútov, ktoré definujú, ako sa môže použiť každá vlastnosť. Napr. ak je atribút **ReadOnly** nastavený na hodnotu *true*, akákoľvek zmena hodnoty pomocou kódu nie je možná. Vlastnosti sú akési kontajnery, ktoré držia buď:

- iné objekty (vychádzajúce zo vstavaného typu **Object**), alebo
- v Tab. 2 spomínané primitívne hodnoty (**Undefined**, **Null**, **Boolean**, **Number** alebo **String**), alebo
- **metódy** (funkcie spojené s objektom na základe nejakej vlastnosti)[4].

ECMAScript jeho vydaním tiež definoval a dodnes definuje akúsi kolekciu vstavaných objektov: globálny (**Global**), už spomenutý jednoduchý objekt (**Object**), objekt funkcie (**Function**), tiež už spomenutý objekt poľa (**Array**) a zároveň typy ako objekt reťazca (**String**), true/false objekt (**Boolean**), číselný (**Number**), matematický (**Math**) a objekt dátumu (**Date**).

Mnoho vstavaných objektov sú v podstate funkciami: môžu byť vyvolané s argumentami. Niektoré z nich sú navyše konštruktormi: sú to funkcie určené pre použitie pomocou operátora *new*. Špecifikácia ECMA-262 popísala pre každú jednu vstavanú funkciu konkrétne argumenty, ktoré funkcia potrebuje a tiež vlastnosti pre objekty danej funkcie. Pre každý vstavaný konštruktor táto špecifikácia navyše popisuje vlastnosti jeho prototypu. Pomocou použitia výrazu **.prototype** sa teda vieme dostať k vlastnostiam vstavaných objektov (v tomto prípade funkcie)[5].

3.1.5.1 Global

Globálny objekt nemá vlastnosť *construct*, teda ho nie je možné použiť ako konštruktor s operátorom *new*. Tento objekt tiež nemá vlastnosť *call*, preto ho nie je možné

vyvolať ako funkciu. Hodnota jeho vlastnosti *prototype* závisí od konkrétnej implementácie. Môžeme o ňom povedať že je akýmsi základným prvkom, na ktorom sú postavené iné objekty s konkrétnymi vlastnosťami.

3.1.5.2 Object(...)

Jedná sa o prvok, ktorým sa dajú vytvárať iné objekty. Zavolaná funkcia `Object()` (bez argumentov) vytvorí a vráti nový objekt bez vlastností (okrem interných vlastností), presne ako keby bol zavolaný konštruktor bez argumentov.

Ak by sme chceli vytvoriť nový objekt pomocou konštruktora, použili by sme pri tom už známy výraz *new* (`new Object()`).

Príklad vytvorenia novej inštancie a jej následného vloženia do premennej:

```
var mojObjekt = new Object();
```

Podobne ako pri poliach máme už v dnešnej dobe z podobného dôvodu aj druhý spôsob vytvárania objektov (pomocou **literálu objektu)**:

```
var mojObjekt = {};
```

Podobne ako pri poliach je možné vkladať objekt do iného objektu, teda vytvárať tzv. vnorené objekty. Aj v tomto prípade sa prikláňame k názoru, že použitie literálu objektu je oproti prvej možnosti jednoduchší, bezpečnejší a aj preto vývojári používajú častejšie.

3.1.5.3 Function(...)

Ak je výraz `Function()` volaný ako funkcia, vytvorí a inicializuje to nový objekt funkcie. Dôsledok tohto volania je však ekvivalentný ako volanie funkcie s použitím výrazu `new Function()`, v oboch prípadoch sa totiž vytvorí objekt funkcie.

3.1.5.4 Array(...)

O vytváraní objektov typu pole sme si už vraveli v sekcii 3.1.3.1. Tento druh je tiež dodnes veľmi používaný, najmä pre ukladanie dát do jedného prvku (napr. premennej) za rovnakým účelom (napr. určitú číselnú sekvenciu).

Podobne ako pri vytváraní objektov funkcií je vytváranie objektov polí pomocou zavolania výrazu **Array()** a pomocou použitia operátora *new* ekvivalentné.

3.1.6 Dodatočné informácie a zhrnutie

S funkciami sa pracovalo a dodnes v jazyku ECMAScript pracuje rovnako ako v iných OO jazykoch, teda po deklarácii sa volajú v kóde ich samotným názvom, doplneným o zátvorky (prázdne, alebo s potrebnými argumentami, resp. parametrami).

Vypracovanie a vydanie štandardu ECMAScript (prvých dvoch verzií) malo pre programátorov webových stránok veľký význam. Tak, ako bolo čiastočne spomenuté v úvode práce, v danom období boli dva bijúce sa webové prehliadače s dvoma samostatnými skriptovacími jazykmi a bolo ťažké navrhovať stránky, ktoré by bez problémov a rozdielov fungovali v oboch prípadoch na 100%. ECMAScript, ako základný koncept skriptovania, tento problém z veľkej časti vyriešil.

3.2 ECMAScript 3

Novinky, ktoré so sebou priniesla tretia verzia ECMAScriptu, sme si pomenovali v prvej časti práce. V nasledujúcej sekcii si priblížime tieto funkcie a vymoženosti, ktoré oproti predchádzajúcim verziám výrazne otvorili dvere pre tzv. moderné skriptovanie, najmä z hľadiska využitia v programovaní skriptov.

3.2.1 Regulárne výrazy

„Regulárny výraz bol a dodnes je špeciálny reťazec znakov, ktorý predstavuje určitý vzor pre textové reťazce.“[14] Najčastejšie sa pre to používa ku kontrole dát zadávaných napr. vo formulároch (rodné číslo, vek a pod.)[37].

Regulárne výrazy sú výrazy, ktoré používajú špeciálne znaky na vyjadrenie istých logických zákonitostí a pravidiel očakávaných reťazcov, rôznych sekvencií znakov a pod. Tieto špeciálne znaky používané v regulárnych výrazoch sa nazývajú **metaznaky**. Aj

[14] PECKA, M. *Regulární výrazy: Regexp není zaklínadlo* [online].[preložené a cit. 22.4.2017]. Dostupné na internete: <http://www.regularnivyrazy.info/>.

v tomto prípade si neuvedieme prvotné typy metaznakov definovaných ECMAScriptom, ale akýsi ucelený systém metaznakov ktoré sa dodnes používajú, resp. ktoré boli dodatočne dopĺňané a definované ďalšími verziami. Nakoľko však boli regulárne výrazy ako celok definované v skriptovaní touto verziou štandardu, pripisujeme celý pokrok práve jej.

Veľmi dôležitým a akýmsi základným metaznakom môžeme chápať **bodku** (.), ktorá predstavuje práve jeden ľubovoľný znak. Medzi metaznaky ale patria aj významné **kvantifikátory** (viď tabuľku Tab. 6), ktoré určujú povolený počet opakovaní znaku, ktorý kvantifikátoru predchádza. Ak by sme chceli zadať na konkrétnom mieste iba jeden znak, môžeme použiť **hranaté zátvorky** ([]) a v nich definovať rozmedzie, napr. konkrétne znaky (**[abc123]**) alebo interval (**[1-5]**). Ak by sme chceli túto podmienku uviesť záporne, stačí ak na začiatok do zátvoriek zavedieme znak **strieška** (^). Uvedenie zápisu **[^01]** by tak malo znamenať povolenie akéhokoľvek znaku okrem nuly a jednotky. Nakoľko sa v programovaní často používajú isté konkrétne rozmedzia znakov (napr. čísla), ECMAScript definoval aj isté skratky týchto skupín (viď Tab. 7). Ak by sme potrebovali opakovanie určitej sekvencie znakov (nie iba jedného), danú sekvenciu by sme dali do **oblých zátvoriek**. Za tie by sme potom mohli zaviesť napr. kvantifikátor, ktorý by určoval počet opakovaní pre celú sekvenciu. Ak by sme tiež programu chceli zadať na výber isté varianty textov, resp. znakových sekvencií, ako oddeľovač variantov by sme použili **vertikálnu čiaru** (|). Poslednými z elementárnych pravidiel a možností regulárnych výrazov, o ktorých si povieme, sú **hranice** (viď Tab. 8), alebo **ukotvenie**, ktoré sa dajú použiť v prípade, ak chceme, aby sa konkrétny reťazec nachádzal na začiatku, resp. na konci prehľadávaného textu[38].

Tab. 6 - Kvantifikátory regulárnych výrazov

Kvantifikátor	Poznámka (počet opakovaní)
?	- minimálne 0x, maximálne 1x
*	- minimálne 0x (maximálne neobmedzené)
+	- minimálne 1x (maximálne neobmedzené)
{n}	- práve 1x
{m,n}	- minimálne mkrát, maximálne nkrát
{m, }	- minimálne mkrát (maximálne neobmedzené)

Tab. 7 - Preddefinované skupiny znakov

Kód	Poznámka (skupina znakov)
`\d`	- číslice 0-9
`\D`	- všetko okrem číslic 0-9
`\w`	- slová (ekvivalentné zápisu **[a-zA-Z0-9_]**)
`\W`	- akýkoľvek znak okrem slov (ekvivalentné zápisu **[^a-zA-Z0-9_]**)
`\s`	- biele znaky (medzera, tabulátor, znaky pre zalomenie riadku)
`\S`	- akýkoľvek znak okrem bielych znakov

Tab. 8 - Hranice

Znak	Poznámka
`^`	- začiatok reťazca (vo vyhľadávanom texte)
`$`	- koniec reťazca (vo vyhľadávanom texte)

Uviesť tiež treba situáciu, kedy by sme chceli konkrétne metaznaky zapísať a chápať ich ako obyčajný znak. V takomto prípade sa pred daný znak píše **spätné lomítko** (\). Dohromady to teda platí pre tieto znaky: \, ^, **$**, **.**, **[**, **]**, |, **(**, **)**, **?**, ***, +, **{** a **}**.

3.2.1.1 Príklady

V nasledujúcej tabuľke si uvedieme niekoľko príkladov použitia regulárnych výrazov aj s vysvetlením, čomu z hľadiska využitia v praxi zodpovedajú.

Tab. 9 - Príklady použitia regulárnych výrazov

Regulárny výraz	Poznámka (čomu výraz zodpovedá)
`x+`	- sekvencia písmen **x** (jeden a viac)
`robot(a)?`	- **robot** alebo **robota**
`ma(ma|lina)`	- **mama** alebo **malina**
`[0-9]|[1-9][0-9]`	- čísla od **0** až do **99**
`(18|19)\d{2}`	- roky **1800-1999**
`\d{3,5}`	- sekvencie dvoch až piatich číslic
`^A.*`	- reťazec začínajúci písmenom **A**, za ktorým ide ľubovoľný (aj prázdny) počet ľubovoľných znakov
`\s+0$`	- reťazec, ktorý končím číslicou **5**, ktorej predchádza minimálne jeden **biely znak**
`y+z`	- **yz**, **yyz**, **yyyz** atď.
`c\+(d|e)`	- **c+d** alebo **c+e**

3.2.2 Lepšia ovládateľnosť reťazcov

Uvedením ECMAScriptu 3 sa do programovania skriptov zaviedli nové funkcie, ktoré mali pôvodne umožniť lepšie ovládanie reťazcov, no niektoré z nich zároveň umožnili aj efektívnejšie riadenie polí a prácu s nimi.

Týmito známymi a v budúcnosti využívanými funkciami boli napr. funkcia porovnávania reťazcov **match()**, nahrádzania **replace()**, ktorá očakáva dva parametre (hľadanáHodnota, náhradnáHodnota). Ďalšou takou funkciou bola vyhľadávacia funkcia **search()** a tiež funkcia spájania **concat()** a skracovania **slice()**, ktorých použitie si aj ukážeme.

3.2.2.1 Metóda concat()

Funkciu *concat()* môžeme, ako sme si už spomínali pri objektoch, chápať aj ako tzv. metódu. S touto metódou bolo a je možné spojovať dve alebo viac reťazcov, resp. polí do jedného. Pôvodné prvky zostanú nedotknuté. Ako príklad fungovania tejto funkcie si uvedieme prácu s poľami:

```
var pole1, pole2, pole3;
pole1 = ["Ahoj."];
pole2 = ["Ako sa máš?"];
pole3 = ["Ja pracujem."];
pole4 = pole1.concat(pole2, pole3);
//pole 4 obsahuje: ["Ahoj.", "Ako sa máš?", "Ja pracujem."]
```

Ako môžete vidieť, premenná **pole4** pred tým nebola deklarovaná, no napriek tomu bola vytvorená a naplnená obsahom prvých troch polí. Metoda *concat()* totiž vracia nové zlúčené pole so všetkými hodnotami. Pôvodné polia zmenené neboli, stále by obsahovali jeden prvok[2].

3.2.2.2 Metóda slice()

Podobne ako predošlú funkciu, aj túto môžeme chápať ako tzv. metódu. Aj v tomto prípade si ukážeme jej využitie na príklade s poľami.

Metóda *slice()* kopíruje vybranú časť poľa a následne ju vracia, teda vytvára akýsi výrez, resp. podmnožinu poľa. „Ani ona pôvodné pole neupravuje, ale vytvára jeho tzv. **plytkú kópiu**. Tejto funkcii hovoríme kde by mala začať a prípadne tiež kde by mala skončiť.“[15]

Tab. 10 - Príklady využiia metódy slice()

Príkaz	Poznámka (funkčnosť)
`pole.slice(2)`	- vracia kópiu poľa `pole` od indexu č. 2 až po koniec poľa
`pole.slice(-2)`	- začína naopak od konca poľa a vráti posledné dva prvky
`pole.slice(2, 4)`	- kopíruje pole `pole` od indexového poradia č. 2 po index č. 4

Vytvorenie plytkej kópie znamená, že ak by napr. naše pole obsahovalo ďalšie pole ako jeden zo svojich prvkov, táto metóda by skopírovala iba odkaz na toto vnorené pole. Inými slovami, všetky zmeny uskutočnené dodatočne na pôvodnom vnorenom poli by sa prejavili aj v jeho kópii. Ako príklad si uvedieme evidenciu akého zoznamu úloh:

```
var ulohy, vsetko, upratat, ostatne;
ulohy = [  "Ist si zabehat.",
           "Oholit sa.",
           [     "Upratat izbu.",
                 "Umyt dlazku.",
                 "Vycistit krb." ] ];
vsetko = ulohy.slice(0); // kopiruje cele pole ulohy
upratat = ulohy.slice(-1); // kopiruje iba vnorene pole
ostatne = ulohy.slice(0, 2); // kopiruje iba prve dve ulohy
```

3.2.3 Nové riadiace príkazy

ECMAScript 3 svojim uvedením priniesol okrem rôznych iných vylepšení aj niekoľko nových príkazov na riadenie programu. Novými samozrejme boli iba

[15] BAŠE, O. 2014. *JavaScript Okamžite.* Brno : Computer Press, 2014. s. 45. ISBN 978-80-251-4163-2

v štandardizovaní skriptovania, nie vo všeobecnom OO programovaní. Najznámejší a dodnes najpoužívanejší z nich je skokový príkaz `do-while`, teda ďalší do série príkazov cyklov.

3.2.3.1 Príkaz do-while

Tento príkaz je variantom príkazu *while,* ale s tým rozdielom, že v tomto prípade sa cyklus uskutoční aspoň raz. Pre ukážku využitia tohto riadiaceho príkazu si uvedieme podobný príklad, ako pri vyššie uvedených metódach, teda príklad s úlohami:

```
var pocetUloh;
do {
      console.log('Zostava uloh: ' + pocetUloh + '.');
      pocetUloh --;
} while (pocetUloh > 0);
```

3.2.3.2 Try/catch spracovanie výnimiek

Jazyk ECMAScript 3 poskytol vývojárom naprieč všetkými skriptovacími jazykmi a prehliadačmi oproti svojim predchádzajúcim verziám mechanizmus, ako ošetriť výnimočné situácie preskočením v toku programu na špeciálne časti kódu, určené pre ošetrenie chýb v programe. Tento mechanizmus a mnohé iné sa používajú dodnes naprieč všetkými jeho implementáciami. Aby sa zistili výnimky, je potrebné umiestniť potenciálne problematický kód do bloku s názvom **try**. Keď v ňom dôjde k chybe, vznikne výnimka a riadenie programu sa presunie ku kódu vo vnútri príslušného bloku s názvom **catch**. V prípade, že by nenastala žiadna výnimka sa kód pre jej spracovanie ignoruje a pokračuje ako obvykle[2]. Túto koncepciu si ako celok detailnejšie priblížime.

Príkaz *try* obsahuje blok kódu, v ktorom môže dôjsť k výnimočnej situácii, ako je napr chyba spustenia, alebo údaj *throw*. Klauzula *catch* poskytuje kód na spracovanie výnimiek. Keď zachytí výnimku, jej identifikátor je viazaný na túto výnimku. Kľúčovým slovom `throw` sa dáva najavo, že nastal chybný stav. Používa sa nasledujúcim spôsobom:

```
throw výraz;
```

Výnimku môžeme vyvolať napr. týmto spôsobom:

```
throw "Toto je chybová správa v tvare textového reťazca.";
```

Napriek tomu je však na to praktickejšie použiť objekt typu **Error**:

```
function vydel(delenec, delitel) {
	if (delitel===0) { // striktná rovnosť
		throw new Error("Deliteľom nemôže byť nula!");
	}
	return delenec / delitel;
}
```

Objekt, ktorý vytvoríme konštruktorom *Error()*, so sebou nesie viac informácií než textový reťazec. Napr. môžeme priamo v kóde testovať, či objekt je chybovým objektom, keď za jeho názov zapíšeme výraz `instanceof Error`.

A čo robiť s vyvolanou výnimkou? Už spomenutý blok `try` má tento formát:

```
try {
// kód, v ktorom môže dôjsť k chybe
} catch (identifikátor) {
// ošetrenie chýb
} finally {
// upratanie
}
```

Klauzula `try` bude pred blokom kódu, v ktorom môže nastať výnimka, zatiaľ čo v bloku `catch` ošetrujeme túto výnimku. Blok `finally` nie je povinný a spúšťa sa, či už k chybe dôjde alebo nie, pokiaľ ho používame, preto, aby sme „upratali“ po predošlom kóde. Teraz sa vrátime k funkcii *vydel()* z predchádzajúceho príkladu:

```
try {
	var vysledok = vydel(7, 0); // chyba – deliteľ je nula
	console.log(vysledok); // vypísať výsledok na konzolu
prehľadávača
} catch (vynimka) {
```

```
    console.log(vynimka.message);
} finally {
    console.log('Toto sa uskutoční vždy.');
}
```

Ako môžeme vidieť, v tomto prípade by nám vznikla chyba (deliteľ je nula). Volanie funkcie `console.log(vysledok)` by sa teda nedostalo na rad a tok programu by skočil do bloku `catch`. V ňom by vypisoval chybovú správu do konzoly a následne by sa presunul do bloku `finally`, do ktorého by sa však presunul aj tak.

3.3 ECMAScript 5 a ECMAScript 5.1

Rozdiely medzi ECMAScriptom 3 (ďalej ES3) a ECMAScriptom 5 (ďalej ES5) sa prejavili najmä v oblastiach ako sú **atribúty a vlastnosti**, **práca s funkciami** (nové vlastnosti), **práca s poľami** (nové vstavané metódy), nová **knižničná podpora dát typu JSON** a tiež bola evidovaná nová vymoženosť s názvom **striktný mód**. Rozdiely medzi ECMAScriptom 5 a ECMAScriptom 5.1 boli zasa naopak minimálne, nakoľko, ako už sme už spomenuli v tejto práci, bolo vydanie verzie 5.1 uskutočnené iba z dôvodu súladu s novou medzinárodnou normou ISO/IEC 16262:2011.

3.3.1 Vlastnosti a atribúty

Kód napísaný v ES3 mohol vytvárať iba vlastnosti, ktoré bolo možné zapísať, vyčísliť a vymazať. V ES3 boli k dispozícii len tri atribúty vlastností: *ReadOnly*, *DontEnum* a *DontDelete*. ES3 tiež nemohol vytvárať vlastnosti, ktoré sa vypočítavali pri priradení alebo čítaní, kdežto ES5 už áno. V ES5 boli spomínané atribúty nahradené atribútmi *Writable, Enumerable* a *Configurable*, ktoré boli ovládateľné. Bolo teda možné pre konkrétne vlastnosti pomocou ich charakteristík určiť, či sa dá meniť hodnotu vlastnosti, či má byť vlastnosť viditeľná pri prechádzaní príkazom *for...in* a tiež, či je možné samotnú vlastnosť zmazať. „Hodnoty atribútov boli programovo nastaviteľné a testovateľné.“[16] Okrem toho boli do ES5 pridané tzv. doplnkové vlastnosti **getter**

[16] HUANG, J. 2015. *EcmaScript 3 VS EcmaScript 5.* [online].[preložené a cit. 25.4.2017]. Dostupné na internete: <http://www.jiehuang.net/wordPress/uncategorized/ecmascript/>.

a **setter**. Od tohto momentu môžeme manipulovať ako s vlastnosťami, tak aj s atribútmi použitím ES5 kódu[39]. Najväčšiu zásluhu na tom mali najmä tieto funkcie:

- *Object.getOwnPropertyDescriptor* (vracia **deskriptor** danej vlastnosti objektu),
- *Object.defineProperty* (definuje novú vlastnosť objektu s atribútmi špecifikovanými deskriptorom),
- *Object.defineProperties* (hromadne definuje nové vlastnosti objektu s atribútmi špecifikovanými polom deskriptorov).

Deskriptor nejakej vlastnosti je v podstate objekt popisujúci jej atribúty a je cezeň tiež možné dostať alebo nastaviť spomenuté *getter* a *setter* vlastnosti. V kóde vyzerá takto:

```
{ writable: true, // je možné meniť hodnotu vlastnosti
configurable: true, // vlastnosť sa dá zmazať (operátorom delete)
enumerable: false, // vlastnosť nie je vidieť príkazom for...in }
```

Uvedieme si tiež použitie metódy *Object.defineProperty(obj, nazovVlastnosti, popisVlastnosti)* na definovanie vlastností a zároveň aj na zmenu atribútov vlastností.

Definovanie vlastnosti:

```
Object.defineProperty(o, "dlzka",{
     getter: function() {return this.computeLength()},
     setter: function(hodnota) {this.changeLength(hodnota)}
});
Object.defineProperty(o, "1", {hodnota: 1,
     enumerable: true, configurable: true}
);
```

Modifikácia atribútov vlastnosti:

```
Object.defineProperty(Array.prototype, "forEach", {
     enumerable: false, writable: false, configurable: false}
);
```

3.3.1.1 Reflexia objektov

Tento pojem znamená akúsi schopnosť objektov oznámiť svetu rôzne informácie o sebe. V ES3 táto možnosť takmer neexistovala. Ak áno, tak iba vo veľmi slabej forme. Podľa [18] bolo napr. taký typ objektu možné zistiť iba veľmi zhruba pomocou operátora *typeof* a napr. prototyp nešiel zistiť vôbec. „Vlastnosti obsiahnuté v objekte sa v ES3 dali zistiť pomocou príkazu *for...in*, ten ale neprechádzal úplne všetky vlastnosti a zahŕňal aj vlastnosti v prototypoch.“[17] Existovala iba funkcia *Object.prototype.hasOwnProperty*, vďaka ktorej bolo možné zistiť, či je daná vlastnosť definovaná priamo v objekte samotnom. Ak bola nejaká funkcia definovaná na prototype konštruktora istého objektu, znamenalo to že ju bolo možné volať priamo na inštanciách daného objektu (napr. volanie `{a: 42}.hasOwnProperty("a")` vracalo *true*). Ak však bola nejaká funkcia definovaná priamo na konštruktore objektu, volať ju na inštanciách nebolo možné[18].

ES5 možnosti reflexie podstatne rozšíril. Zapríčinené to bolo najmä rôznymi novými funkciami, medzi ktoré patrili aj:

- *Object.getPrototypeOf* (vracia prototyp objektu v parametri),
- *Object.keys* (vracia pole s názvami vlastností objektu v parametri s vynechaním tých, ktoré by vynechal príkaz *for...in*),
- *Object.getOwnPropertyNames* (vracia pole s názvami vlastností objektu v parametri vrátane tých, ktoré by vynechal príkaz *for...in*) a iné.

V rámci reflexie v ES3 tiež nebolo možné zistiť, či je nejaký objekt pole. Riešilo sa to zložitým použitím metódy *toString*. Od vydania ES5 sa ale detekcia dá uskutočniť ľahko, najmä vďaka vstavanej metóde *Array.isArray*, ktorá „pole spoľahlivo pozná“.[18]

3.3.2 Funkcie

Funkcia v ES5 ako prvok dostala oproti tej z ES3 tieto zmeny:

[17] MAJDA, D. 2009. *Co přináší nový ECMAScript 5?* [online].[preložené a cit. 26.4.2017]. Dostupné na internete: <https://www.zdrojak.cz/clanky/co-prinasi-novy-ecmascript-5/>. ISSN 1803-5620.

[18] MAJDA, D. 2009. *Co přináší nový ECMAScript 5?* [online].[preložené a cit. 26.4.2017]. Dostupné na internete: <https://www.zdrojak.cz/clanky/co-prinasi-novy-ecmascript-5/>. ISSN 1803-5620.

1. Hodnota kľúčového slova *this* nebude zmenená keď bude volaná metóda *.call()* alebo metóda *.apply()*

2. *Null* a *undefined* nebudú vracané do globálu

3. Prvok, ktorý nie je objektom, nebude preklopený na objekt

4. Nová metóda **Bind** (v preklade: metóda väzby). Táto väzbová metóda povoľuje používateľovi nastaviť hodnotu prvku *this* a aplikovať túto hodnotu do funkcie:

```
var obj = {meno : "pali"};
function funk() {console.log(this.meno);}
var funk1 = funk.bind(obj);
funk1(); //vypíše na konzolu: „pali“
```

Podobných objektových metód bolo týmto štandardom vytvorených viac.

3.3.3 Polia

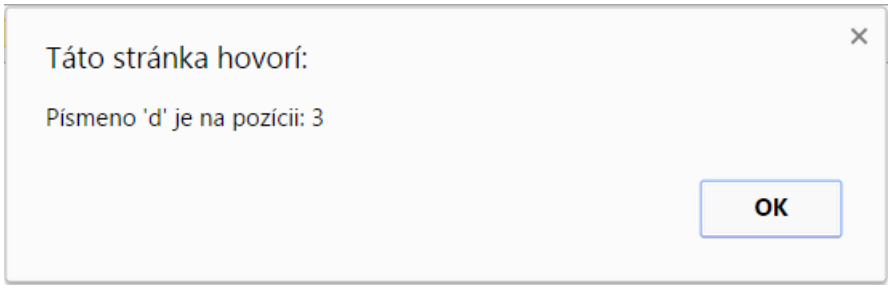
Manipulovať s poľami sa v ES3 moc nedalo. ES5 to umožnil niekoľkými novými metódami. Tieto metódy zožali medzi vývojármi veľký úspech, a v implementáciách ECMAScriptu, napr. v JavaScripte, sa používajú dodnes. Sú to funkcie *Array.prototype.indexOf* (striktne (pomocou operátora ===) hľadá prvý výskyt daného prvku v poli a vracia jeho indexové poradie), *Array.prototype.lastIndexOf* (podobne, ale začína od konca, teda hľadá posledný výskyt prvku), *Array.prototype.forEach* (prechádza celé pole a pre každý prvok vykonáva konkrétny úkon, resp. zadanú funkciu), *Array.prototype.map* (podobne, ale vracia pole, ktoré obsahuje hodnoty, ktoré vrátime funkciou spätného volania), *Array.prototype.every* (pre každý prvok v poli spúšťa danú overovaciu funkciu, resp. kontroluje, či všetky prvky spĺňajú isté kritérium), *Array.prototype.some* (podobne, ale na celkové vyhodnotenie *true* stačí, aby aspoň jeden prvok splnil dané kritérium), *Array.prototype.filter* (funguje podobne ako metódy *every()* a *some()*, ale s tou výnimkou, že kopíruje prvky, ktoré spĺňajú spomenuté kritérium do nového poľa), *Array.prototype.reduce* (prechádza dané pole v cykle a predáva funkcii

spätného volania ako argumenty predošlú a aktuálnu hodnotu prvku) a funkcia *Array.prototype.reduceRight* (funguje podobne ako metóda *reduce()*, ale v opačnom smere, teda začína od konca a postupuje smerom k začiatku). Zápis a prípadné zobrazenie fungovania niektorých z týchto funkcií si stručne predvedieme.

3.3.3.1 Ukážka funkcie index()

Nasledujúci kód a ostatné neobsahujú kompletný obsah použiteľného skriptovacieho kódu, ale iba úryvok určený na demonštrovanie fungovania konkrétnej funkcie. Na konci kódu sa nachádza volanie funkcie *alert()* na vytvorenie výstražného okna:

```
var abeceda = ["a", "b", "c", "d", "e"];
alert("Písmeno 'd' je na pozícii: " + abeceda.indexOf("d"));
```



Obr. 2 - Výstup kódu funkcie *indexOf()*

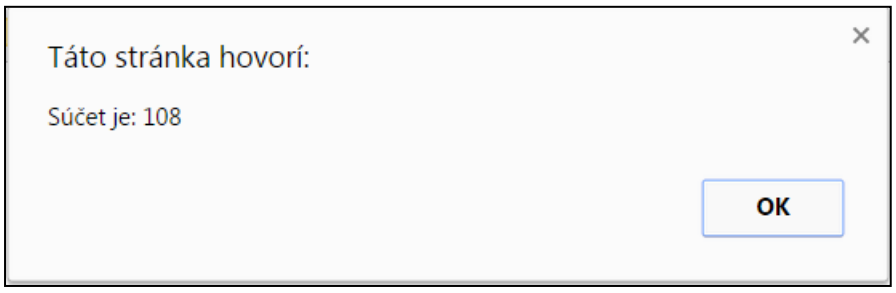
Ako môžeme vidieť, index zadaného písmena je číslo tri napriek tomu, že je v poli na štvrtom mieste. Je to zapríčinené indexovaním od čísla nula.

3.3.3.2 Ukážka funkcie forEach()

Pred uvedením ES5 sa rôzne matematické úkony nad poľami robili pomocou cyklu *for* skladajúcim sa zo štyroch častí: inicializácia, podmienky, konečný výraz a telo cyklu. Metóda *forEach()* to značne uľahčila. V našom príklade jej predávame anonymnú funkciu, ktorá prijíma ako argument hodnotu aktuálneho prvku poľa a tú ukladá do premennej `cislo`. Telo uvedenej vnútornej funkcie obsahuje iba jednoduchý matematický výpočet.

```
var pole, sucet;
pole = [4, 8, 15, 16, 23, 42];
sucet = 0;
```

```
pole.forEach(function(cislo) {
  sucet = sucet + cislo;
});
alert("Súčet je: " + sucet);
```



Obr. 3 - Výstup kódu funkcie *forEach()*

3.3.3.3 Ukážka funkcie map():

V nižšie uvedenom úryvku počítame pomocou metódy *map()* druhé mocniny všetkých prvkov poľa `pole`. Výsledné pole ukladáme pod názvom `umocnene`:

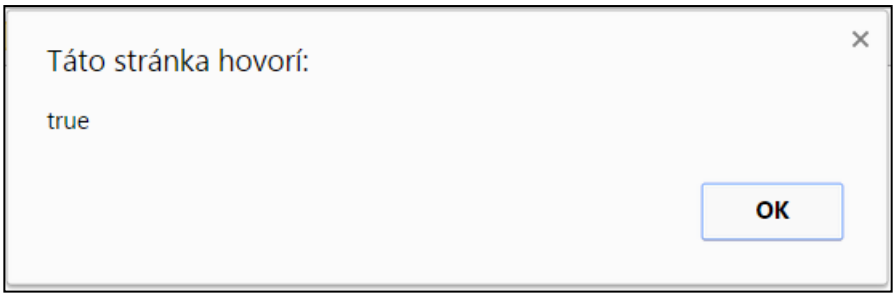
```
var pole = [1, 2, 3, 4, 5];
var umocnene;
umocnene = pole.map(function(cislo) {
  return (cislo * cislo);
}); // pole umocnene obsahuje [1, 4, 9, 16, 25]
alert(umocnene);
```

Obr. 4 - Výstup kódu funkcie *map()*

3.3.3.4 Ukážka funkcie every():

Niekedy sa stáva že musíme overovať, či dáta v poli vyhovujú istým pravidlám, resp. podmienkam. Vďaka tejto funkcii v ES5 je to veľmi ľahko uskutočniteľné. Táto metóda spúšťa jej predanú funkciu spätného volania pre každý prvok poľa. V závere vracia hodnotu *true*, ak sú všetky hodnoty v poli platné, inak *false*.

```
var pole, jePlatne;
pole = [1, 2, 3, 4, 5];
jePlatne = pole.every(function(cislo) {
 return (cislo < 10);
}); // jePlatne obsahuje true
alert(jePlatne);
```

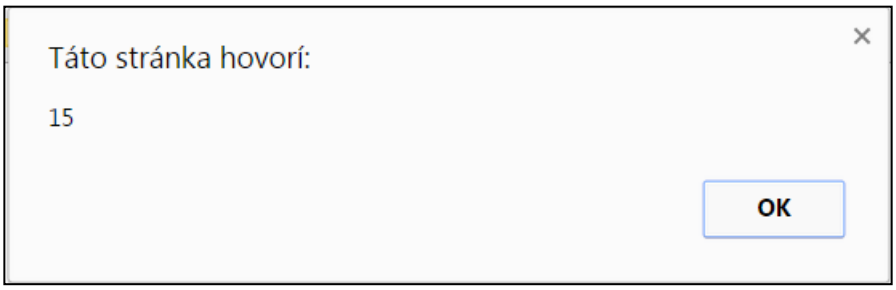


Obr. 5 - Výstup kódu funkcie *every()*

3.3.3.5 Ukážka funkcie reduce():

Občas je tiež potrebné urobiť matematický výpočet nad prvkami poľa a obdržať tak jedinú výslednú hodnotu. ES5 nám umožnil v takejto situácii použiť metódu *reduce()*, prípadne metódu *reduceRight()*. Prvá menovaná používa vnútornú funkciu podobne ako vyššie uvedené metódy, ale v argumentoch očakáva dva prvky. Prvým je buď prvý prvok poľa, alebo predchádzajúca hodnota funkcie vykonanej nad predošlým prvkom a druhým argumentom je hodnota aktuálneho prvku. Táto metóda navyše predáva funkcii spätného volania indexové poradie a odkaz na samotné pole, keby sme tieto údaje náhodou potrebovali pre náš výpočet. V nasledujúcom príklade si ale vystačíme so spomenutými argumentami.

```
var pole, sucet;
pole = [1, 2, 3, 4, 5];
sucet = pole.reduce(function(predosla, aktualna) {
  return predosla + aktualna;
}); // sucet obsahuje 15
alert(sucet);
```



Obr. 6 - Výstup kódu funkcie *reduce()*

3.3.4 JSON formát

V ES3 taktiež nebola žiadna normatívna technológia pre preklad JSON dát. Do ES5 boli pridané dve hlavné knižnice pre podporu tohto formátu. Z hľadiska kódu boli hlavnými funkciami *JSON.stringify()*, ktorá mala vrátiť hodnotu do JSON reťazca a *JSON.parse()*, ktorá mala zabezpečiť opačný proces[39].

JSON.stringify() → reťazec:

```
var jsonStr = '{"a":1, "b":2, "c":3}';
JSON.parse(jsonStr);
JSON.parse(jsonStr, function(kluc, hodnota) {
    return typeof hodnota === 'number' ? hodnota * 2 : hodnota;
}); // {a:2, b:4, c:6}
```

3.3.5 Striktný mód

Striktný mód vznikol ako nová koncepcia zameraná na poskytnutie dôkladnejšej kontroly chýb a tiež na vyvarovanie sa chybám a hrozbám ešte pred prekladom kódu. Dalo a dodnes sa vďaka nej dá vyvarovať rôznym nepríjemnostiam, ktoré majú tendenciu vznikať po preklade kódu. Zápisom `"use strict";` na začiatku oblasti platnosti sa dá povedať interpreteru konkrétnej implementácie ECMAScriptu, že by mal zamedziť potenciálne nebezpečným spôsobom zaobchádzania s kódom.

3.4 ECMAScript 6

V tejto sekcií sa pozrieme na veľkou mierou využiteľné možnosti, ktoré oproti svojmu predchodcovi priniesol ECMAScript 6 (ES6). Nakoľko bolo týchto prírastkov obrovské množstvo, bližšie sa pozrieme iba na akýsi výber daného portfólia, teda na

najdôležitejšie a dodnes najpopulárnejšie zmeny oproti ES5 vo vybraných oblastiach[20]. Pripomenúť si však vopred musíme fakt, že väčšinu z nich nedokázali v riadnej miere implementovať webové prehliadače a preto bolo nutné použiť niektorý z transkompilátorov, o ktorých sme si už čo to vraveli, na prekompilovanie kódu ES6 na ES5 (napr. BabelJs na kompiláciu v jazyku JavaScript). V nasledujúcej časti práce si priblížime spomenuté oblasti zmien a tiež niektoré samostatné prvky ako novinky ES6.

3.4.1 Premenné

Do vydania ES6 boli programátori zvyknutí používať na deklarovanie premenných výraz `var`. Po jeho vydaní bolo možné na podobnú vec používať aj výraz `let`. Jediný rozdiel spočíval v rozsahu. Výraz `var` vytváral premenné, ktoré vnímali svoj rozsah po okolitú premennú a rozsahom premennej vytvorenej výrazom `let` mal byť pre zmenu len ten blok, v ktorom sa nachádzala. To malo viesť a aj viedlo k zjednodušeniu kódu zmenšením počtu premenných.

```
if(true) {
   let a = 25;
}
console.log(a); // na konzole by bol vypísaný výraz undefined
```

Vytvorený však bol aj iný spôsob, ako deklarovať premenné v rozsahu konkrétneho bloku. Výraz `const` umožnil deklarovať *read-only* referenciu k hodnote. Nutné bolo priamo priradiť premennú. Ak by sa niekto pokúšal zmeniť premennú alebo by hodnotu nenastavil okamžite, nasledovala chyba:

```
const nasa_konstanta = 9;
nasa_konstanta = 10 // chyba (Error)
const nejaka_konstanta; // chyba (Error)
```

Meniť vlastnosti objektu alebo prvky poľa však zostalo možné:

```
const nas_objekt = {nejakaVlastnost: 15};
nas_objekt.nejakaVlastnost = 'nejakyNapis'; // v poriadku
```

3.4.2 Funkcie polí

Ako sme spomínali v prechádzajúcej časti práce, funkcie polí boli skvelými doplnkami pre skriptovanie. Vývojárom umožnili robiť kódy stručnejšími a prehľadnejšími. ES6 však priniesol ešte efektívnejšie písanie týchto funkcií:

```
// ES5:
var knihy = [{nazov: 'a', cena: 6}, {nazov: 'b', cena: 7}];

var nazvy = knihy.map(function (kniha) {
   return kniha.nazov;
});

// ES6:
let knihy = [{nazov: 'a', cena: 6}, {nazov: 'b', cena: 7}];

let nazvy = knihy.map( kniha => kniha.nazov );
```

Ako môžeme vidieť, v syntaxi ES6 bolo vyňaté kľúčové slovo `function`. Čo zostalo je nula a viac argumentov, šípka `=>` a telo funkcie. Príkaz `return` bol implicitne pridaný. Pri žiadnom alebo viac ako jednom argumente bolo nutné pridať zátvorky.

3.4.3 Reťazce

V oblasti pracovania z reťazcami stojí za spomenutie novinka s názvom **šablónové literály.** Tie totiž poskytujú jednoduchý spôsob vytvárania reťazcov a vykonávania ich vzájomnej interpolácie. Ich syntax je založená na použitý znaku **$** a **zložených zátvoriek**. Tu je krátka demonštrácia ich fungovania v porovnaní s podobnou prácou v ES5:

```
let meno = 'Peto', pomarance = 3, mrkvy = 4,
polievky = function() { return 5; };

console.log(`Tento clovek sa vola ${meno}.`);
```

```
console.log(`On drzi ${pomarance} pomarance, ${mrkvy} mrkvy a
${polievky()} polievok. `);

// ES5:
console.log('On drzi ' + pomarance + ' pomarance, ' + mkrvy + '
mrkvy a ' + polievky() +' polievok.');
```

Tento nástroj má samozrejme mnoho ďalších možností využitia. Napriek tomu sa dá už na tomto krátkom a jednoduchom príklade vidieť, ako veľmi uľahčuje prácu s reťazcami.

3.4.4 Polia

Uvedením ES6 získali aj objekty typu *Array* (polí) niečo, čo uľahčilo vývojárom prácu s nimi. Dostali totiž nové metódy. Spomedzi väčšieho množstva si povieme o funkciách *of(), find(), findIndex()* a *fill()*.

3.4.4.1 Metóda of()

Použitie výrazu `Array.of` sa správa veľmi podobne ako konštruktor poľa. Táto metóda ošetruje jeden konkrétny prípad, keď mu odovzdávame jeden číselný argument. Zavedenie tohto prvku do skriptovania viedlo častokrát medzi vývojármi k názoru, že je užitočnejšie ako použitie výrazu `new Array()`. Sú však aj prípady, kedy bude lepšie použiť literál poľa.

```
let a = new Array(2); // do premennej pole: [undefined, undefined]
let b = Array.of(12); // pole: [12]
let c = [1, 2, 3]; // použitie literálu poľa
```

3.4.4.2 Metóda find()

Táto metóda prehľadáva pole a vracia prvý prvok, pri ktorom je podmienka vyhodnotená ako *true*:

```
[8, 2, 4, 12].find(x => x === 4) // vráti prvok 4
```

3.4.4.3 Metóda findIndex()

Táto metóda funguje veľmi podobne, ale vracia index daného nájdeného prvku:

```
[8, 2, 4, 12].findIndex(x => x === 2) // vráti 1 (index prvku 2)
```

3.4.4.4 Metóda fill()

„Zavedením tejto funkcie sa umožnilo prepisovať konkrétne elementy poľa podľa daných argumentov“[19]. Pri jednom zadanom argumente prepíše táto metóda všetky prvky poľa danou hodnotou. Pri dvoch argumentoch znamená prvý hodnotu, ktorá bude použitá na prepísanie a druhý indexové poradie začiatku prepisovania. Pri troch znamená tretí argument indexové poradie konca prepisovania (pokiaľ sa bude prepisovať).

```
[2, 3, 4, 5].fill(9) // [9, 9, 9, 9]
[6, 7, 8, 9, 10, 11, 12].fill(4, 2, 4) // [6, 7, 4, 4, 4, 11, 12]
```

3.4.5 Operátor šírenia

Operátor šírenia, resp. rozšírenia `...` bol vytvorený za účelom rozvinutia syntaxe jazyka a jej obohatenie o možnosť napr. šírenia prvkov poľa na konkrétnych miestach. Na to je totiž spomínaný operátor viac než vhodný. Príkladom jeho využitia môžu byť napr. volania funkcií. Ukážeme si niekoľko príkladov, na ktorých sa pokúsime čo najefektívnejšie demonštrovať jeho využiteľnosť.

Na úvod sa pozrieme na možnosť šírenia prvkov poľa v inom poli:

```
let hodnoty = [3, 6, 7];
let nejake = [...hodnoty, 9]; // do premennej vloží [3, 6, 7, 9]
let ine = [...hodnoty, 9, ...hodnoty]; // [3, 6, 7, 9, 3, 6, 7]
```

Veľké využitie vidíme aj pri volaní funkcií s argumentami:

```
let hodnoty = [15, 16, 17];
```

[19] KAPPERT, L. 2015. *ECMAScript 6 (ES6): What’s New In The Next Version Of JavaScript.* [online].[preložené a cit. 28.4.2016]. Dostupné na internete: <https://www.smashingmagazine.com/2015/10/es6-whats-new-next-version-javascript/>.

```
function nejakaFunkcia (x, y, z) {
  console.log(x); }
//ES6:
nejakaFunkcia(...hodnoty); // na konzolu vypíše 15
// ES5:
nejakaFunkcia.apply(null, hodnoty);
```

Ako môžeme vidieť, vďaka tejto vymoženosti sa v ES6 nemusí zakaždým používať v príklade uvedená funkciá *apply()* a mnohé iné. Syntax je veľmi flexibilná, nakoľko je možné operátor šírenia použiť kdekoľvek v zozname argumentov[20].

3.4.6 Deštrukturalizácia

Tento prvok priniesol oproti ES5 účinný spôsob extrakcie rôznych dát z objektov alebo polí. Pre jednoduché znázornenie si ukážeme príklad s využitím poľa. S touto syntaxou sa môžu v jednom kroku priradiť k premennej viaceré hodnoty:

```
let [a, b] = [1, 2]; // a = 1, b = 2
// ES5:
var pole = [1, 2];
var a = pole[0];
var b = pole[1];
```

Možné je tiež jednoduchým spôsobom hodnoty v poli vymeniť:

```
let a = 3, b = 5;
[a, b] = [b, a]; // a = 5, b = 3
```

Táto syntax je aplikovateľná aj pri objektoch:

```
let objekt = {c: 5, d: 6};
let {c, d} = objekt; // c = 5, d = 6
// zmena názvov premenných:
```

```
let objekt = {e: 7, f: 8};
let {e: a, f: b} = objekt; // a = 7, b = 8
```

3.4.7 Parametre a moduly

3.4.7.1 Parametre

Oproti ES5 je v ES6 možné definovať predvolené hodnoty parametrov funkcií:

```
function nejakaFunkcia(i, j = 1) { return i * j; }
nejakaFunkcia (3); // 6
nejakaFunkcia (3, undefined); // 6
nejakaFunkcia (3, 4); // 12
```

V ES5 sme predtým museli najskôr naplniť niektoré argumenty:

```
function nejakaFunkcia (i, j) {
   j = j === undefined ? 2 : j;
   return i * j;
}
```

Predvolená hodnota sa použije buď nezadaním argumentu, alebo zadaním *undefined* argumentu.

Používať sa to dá aj v kombinácii so spomenutým operátorom rozšírenia:

```
function nejakaFunkcia(k, ...zostavajuce) {
   return k * zostavajuce.length;
}
nejakaFunkcia(5, 0, 0, 0); // vráti 15 (5 * 3)
```

3.4.7.2 Moduly

Každý seriózny ECMAScript program v ES5 používal nejaký modulárny systém. Mohli to byť napr. *AMD* alebo *CommonJS*. Webové prehliadače však vtedy v sebe nemali zabudované žiadne modulárne systémy. Špecifikácia ES6 teda prišla okrem novej syntaxe

s vylepšením, ktoré zahŕňalo mechanizmus načítania pre moduly. Fungovanie modulov bolo postavené na princípe dvoch nových kľúčových slov, a síce `export` a `import`.

Ako môžeme vidieť na nasledujúcom príklade, v programe je možné uviesť viacero `export` výrazov. Každý musí explicitne uviesť typ exportovanej hodnoty (v tomto prípade `function` a `var`):

```
// lib/math.js
export function spocitaj(m, n) {
   return m + n; }
export var pi = 3.141593;

// aplikacia.js
import { spocitaj, pi } from "lib/math";
console.log('2π = ' + spocitaj(pi, pi));
```

Ak by bolo potrebné importovať modul ako celok, môže sa použiť zástupný znak *, v kombinácii s kľúčovým slovom `as` za účelom lokálneho pomenovania modulu:

```
// aplikacia.js
import * as math from "lib/math";
console.log('2π = ' + math.spocitaj(math.pi, math.pi));
```

3.4.8 Triedy

Mnohým je určite z programovania v iných OO jazykoch známe používanie tried. Aj ES6 sa pridal k jazykom, ktoré ho podporujú a bola to veľmi diskutovaná téma, najmä medzi programátormi. Niektorí zastávali názor, že ich zavedenie bolo veľkým opozitom proti prototypovému charakteru jazyka ECMAScript (a jeho implementácii), iní si zasa mysleli, že to bol dobrý krok, nakoľko mal znížiť prekážku vstupu pre začiatočníkov a ostatných ľudí, ktorí pred tým programovali v iných jazykoch. Triedy boli založené na kľúčových slovách `class` a `constructor`. Na ukázanie syntaxe si uvedieme krátky príklad:

```
class Lod {
```

```
    constructor(nazov) {
        this.name = nazov;
        this.kind = 'lod';
    }
    vratNazov() {
        return this.name;
    }
}
// Vytvorenie inštancie
let mojaLod = new Lod('titanic');
```

Na dedenie zo základnej triedy sa používa kľúčové slovo `extends` a zápis vyzerá asi takto:

```
class Jachta extends Lod { ...
```

3.4.9 Zhrnutie

ES6 priniesol obrovské zlepšenia, najmä v ohľade zvýšenia možností ľahšie vyvíjať samotný kód. Zavedenie transkompilátora umožnilo všetkým vývojárom efektívne previesť a uzatvoriť celý kód do ES5, kým mali prehliadače pridávať a podporovať viac a viac funkcií v súlade ES6.

3.5 ECMAScript 7

Okrem opravy chýb a menších vylepšení bol prínos ECMAScriptu 7 (ES7) pomerne malý. Ako sme už v tejto práci spomínali, doplnenie tohto štandardu sa nieslo v znamení pridania dvoch nových vymožeností:

- *operátor umocňovania* (**),
- funkciu *Array.prototype.includes,* ktorá dokáže určiť, či vybrané pole obsahuje vybraný prvok.

3.5.1 *Operátor umocňovania*

Od júna minulého roka (2016) môžu vývojári na umocňovanie používať namiesto funkcie *pow()* nový operátor, ktorého použitie si demonštrujeme na stručnom príklade:

```
let mocnina = 2 ** 4; // mocnina = 16
// alebo:
let mocnina = 2;
mocnina **= 3; // mocnina = 8

//ekvivalentný príklad funkcie umocňovania v ES6:
let mocnina = Math.pow(2, 5); // mocnina = 32
```

3.5.2 *Metóda includes()*

Táto nová funkcia je „podobná funkcii *indexOf()*“[20]. Rozdiel je len v tom, že táto metóda nezisťuje index daného prvku, ale zisťuje, či je nejaký prvok členom daného poľa. Okrem toho tiež táto funkcia prehľadáva aj dáta typu *NaN*. Hodný spomenutia je tiež fakt, že nerozlišuje hodnoty *+0* a *-0*[41]. V nasledujúcom kóde si na úryvkoch kódov ukážeme, ako táto funkcia vyhodnocuje prípadné situácie:

```
['a', 'b', 'c'].includes('a'); vracia true
[NaN].includes(NaN); vracia true
[NaN].indexOf(NaN); vracia -1
[-0].includes(+0); vracia true
```

3.6 ECMAScript 8

Ako už bolo v práci spomenuté, tento štandard zatiaľ nebol oficiálne uverejnený. Na internete sú síce dostupné isté syntaktické vysvetlenia a príklady k niektorým z uvedených noviniek, ale nakoľko zatiaľ nie je dokončená oficiálna dokumentácia tohto projektu, uverejnená na webovej stránke spoločnosti ECMA International a slúžiaca na overenie a porovnanie údajov, informácie o týchto vymoženostiach môžu byť irelevantné,

[20] RAUSCHMAYER, A. 2016. *ES2016 feature: Array.prototype.includes*. [online].[preložené a cit. 30.4.2017]. Dostupné na internete: <http://2ality.com/2016/02/array-prototype-includes.html>.

skreslené a tiež nepravdivé. Analýza tohto obsahu a komparatívne porovnanie s predošlou verziou jazyka by za týchto podmienok mohli byť mätúce a mohli by mať negatívny dopad na cieľ samotnej práce. Vykonávať ich teda v tomto prípade nebudeme.

3.7 ES.Next

Tento pojem predstavuje akýsi názov, ktorý odkazuje na čokoľvek, čo je spojené s ďalšou verziou ECMAScriptu, ktorá je v danom čase v príprave, resp. ešte nie je dokončená. Aj napriek faktu, že ani nasledujúca a pre krátky časový horizont očakávaná verzia ECMAScriptu doteraz nebola oficiálne uverejnená, sa na internete objavili informácie o edícii špecifikácie ECMA-262 plánovanej na rok 2018.

Súčasťou vydania plánovaného na rok 2018 (a tiež práve očakávaného – na rok 2017) by podľa [42] okrem spomenutých funkcionalít a metód mohli byť aj tieto návrhy:

- revízia metódy *Function.prototype.toString*,
- revízia literálov šablón (priniesť by mala viac „syntaktickej slobody“[21] pre vnútorné časti označených literálov šablón),
- globálna premenná *global* (mala by poskytnúť nový štandardný spôsob prístupu ku globálnym objektom),
- nové vlastnosti operátorov šírenia (*spread*) a zvyšku (*rest*),
- asynchrónna iterácia (tento návrh by mal priniesť „podporu pre asynchrónne doručované dáta, ako napr. riadky textu čítané asynchrónne z určitého priečinku alebo http pripojenia“[22]),
- nový operátor *import()* (umožniť by mal dynamické načítavanie ECMAScript modulov, ktoré sú momentálne statické (musí byť

[21] RAUSCHMAYER, A. 2017. *ES proposal: Template Literal Revision.* [online].[preložené a cit. 30.4.2017]. Dostupné na internete: <http://2ality.com/2016/09/template-literal-revision.html>.

[22] RAUSCHMAYER, A. 2016. *ES proposal: asynchronous iteration.* [online].[preložené a cit. 30.4.2017]. Dostupné na internete: <http://2ality.com/2016/10/asynchronous-iteration.html>.

špecifikované čo importujeme a exportujeme v čase kompilácie a nemôžeme do toho zasahovať počas behu programu))[45],

- nové *RegExp* tvrdenia spätne sledujúce kód,
- možnosť natívneho získania prístupu k vlastnostiam znakov Unicode v *RegExp* výrazoch[46],
- nové *RegExp* pomenované skupiny a iné možnosti práce s regulárnymi výrazmi.

Záver

Aj napriek pomerne veľkému problému, ktorým bolo získavanie metodicky potrebných a pre prácu dôležitých údajov z neucelených zdrojov, sa nám podarilo naplniť cieľ práce, teda zostaviť detailný popis jazyka ECMAScript a vykonať ako analýzu jednotlivých verzii, tak aj ich vzájomnú komparatívnu analýzu, ktorá bola vykonaná a zdokumentovaná krok po kroku, od prvej až po poslednú verziu ECMAScriptu, vždy medzi dvomi z hľadiska vývoja najbližšími verziami. V rámci týchto metodických postupov boli na stručných ukážkach kódovania demonštrované rôzne využitia a zápisy noviniek, ktoré nové verzie so sebou priniesli.

Počas vypracovávania práce sme v rámci problematiky skriptovania mali možnosť hlbšie nahliadnuť do vývoja webových technológií a tiež zistiť, že to so samotným vývojom štandardu ECMAScript nebolo vždy jednoduché. Spočiatku predstavoval akúsi elementárnu kombináciu syntaxí a pravidiel skriptovania dvoch vtedy najpoužívanejších, no medzi sebou súperiacich jazykov, neskôr však začali byť popri rôznych zásadných udalostiach vyvíjané jeho novšie verzie a postupom času začal ECMAScript nadobúdať čím ďalej, tým väčšiu podobu OO jazyka, najmä vďaka rôznym inovačným prvkom a metódam, pri ktorých sa jeho tvorcovia inšpirovali inými OO jazykmi. Vývoj sa za posledné roky oproti začiatkom podstatne zrýchlil a uberá sa dobrým smerom. Tvrdiť to môžeme z dôvodu, že sa momentálne čaká na vydanie ECMAScriptu s číslom osem, ktorý je za posledné roky už v poradí tretím rozšírením a zároveň z dôvodu, že jeho implementácia JavaScript sa za ten čas stala jedným z najobľúbenejších a najpoužívanejších programovacích jazykov vôbec. Pri prehliadaní moderných interaktívnych webových stránok je takmer nemožné nájsť takú, ktorá by neobsahovala elementy JavaScriptu. Vytvárať a vylepšovať sa v ňom začali aj rôzne CMS systémy pre tvorbu webových stránok, ako napr. systém WordPress.

Veríme, že táto práca bude do budúcnosti slúžiť ako ucelený podklad a vhodný zdroj pre ďalšie spracovania problematiky.

Zoznam použitej literatúry

Knihy/monografie:

[1] BRAITWAITE, R. 2016. *JavaScript Allongé, the "Six" Edition : Programming from Functions to Classes in ECMAScript 2015.* Reg „raganwald" Braitwaite, 2016. 513s.

[2] BAŠE, O. 2014. *JavaScript Okamžite.* Brno : Computer Press, 2014. 160 s. ISBN 978-80-251-4163-2

[3] TRILEC, P. 2008. *Porovnání volně dostupných VoiceXML interpreterů* [online]. Brno : Masarykova univerzita, 2008. [cit. 3.12.2016]. Dostupné na internete: <https://is.muni.cz/th/143307/fi_b/bakalarka-print.txt>.

[4] ECMA INTERNATIONAL. 1997. *Standard ECMA-262.* [online]. Ecma International, 1997. Dostupné na internete: <https://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%201st%20edition,%20June%201997.pdf>.

[5] ECMA INTERNATIONAL. 1998. *Standard ECMA-262: 2nd Edition.* [online]. Ecma International, 1998. Dostupné na internete: <https://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%202nd%20edition,%20August%201998.pdf>.

[6] ECMA INTERNATIONAL. 1999. *Standard ECMA-262: 3rd Edition.* [online]. Ecma International, 1998. Dostupné na internete: <https://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%203rd%20edition,%20December%201999.pdf>.

[7] ECMA INTERNATIONAL. 2016. *ECMAScript 2016 Language Specification : 7th Edition.* [online]. Ecma International, 2016. Dostupné na internete: <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.

Internetové zdroje:

[8] Total number of Websites [online]. [preložené a cit. 1.11.2016]. Dostupné na internete: <http://www.internetlivestats.com/total-number-of-websites/#trend>.

[9] What is ECMAScript ? [online].[preložené a cit. 13.11.2016]. Dostupné na internete: <http://www.computerhope.com/jargon/e/ecmascript.htm>.

[10] Javascript: What is ECMAScript? [online].[preložené a cit. 2.12.2016]. Dostupné na internete: <http://www.programmerinterview.com/index.php/javascript/javascript-what-is-ecmascript/>.

[11] MAKULOVÁ, S. 2005. *Prečo je potrebné dodržiavať štandardy World Wide Web konzorcia* [online].[cit. 3.12.2016]. Dostupné na internete: <http://itlib.cvtisr.sk/archiv/2005/3/preco-je-potrebne-dodrziavat-standardy-world-wide-web-konzorcia.html?page_id=1748>.

[12] ECMAScript [online].[preložené a cit. 29.11.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/ECMAScript>.

[13] ECMAScript [online]. Dostupné na internete: <https://cs.wikipedia.org/wiki/ECMAScript>.

[14] MALÝ, M. 2010. *Node.js – s JavaScriptem na server.* [online].[preložené a cit. 4.12.2016]. Dostupné na internete: <https://www.zdrojak.cz/clanky/node-js-s-javascriptem-na-server/>. ISSN 1803-5620.

[15] Standard ECMA-262 [online]. Dostupné na internete: <http://www.ecma-international.org/publications/standards/Ecma-262-arch.htm>.

[16] Úvod – ECMAScript pro XML [online]. Dostupné na internete: <https://developer.mozilla.org/cs/docs/ECMAScript_pro_XML/%C3%9Avod>.

[17] Standard ECMA-357 [online]. Dostupné na internete: <http://www.ecma-international.org/publications/standards/Ecma-357.htm>.

[18] MAJDA, D. 2009. *Co přináší nový ECMAScript 5?* [online].[preložené a cit. 26.4.2017]. Dostupné na internete: <https://www.zdrojak.cz/clanky/co-prinasi-novy-ecmascript-5/>. ISSN 1803-5620.

[19] AN, N. 2015. *Introduction to ECMASCRIPT 2015 – ES6.* [online]. Dostupné na internete: <http://www.codeproject.com/Tips/1046032/Introduction-to-ECMASCRIPT-ES>.

[20] KAPPERT, L. 2015. *ECMAScript 6 (ES6): What's New In The Next Version Of JavaScript.* [online].[preložené a cit. 3.12.2016]. Dostupné na internete: <https://www.smashingmagazine.com/2015/10/es6-whats-new-next-version-javascript/>.

[21] JANOVSKÝ, D. *Úvod do JavaScriptu.* [online].[preložené a cit. 2.11.2016]. Dostupné na internete: <https://www.jakpsatweb.cz/javascript/javascript-uvod.html>.

[22] HREBENAR, J. *Úvod do JavaScriptu.* [online].[preložené a cit. 2.11.2016]. Dostupné na internete: <http://www.pestujemeweb.cz/obsah/javascript/javascript-uvod.php>.

[23] JavaScript [online]. Dostupné na internete: <https://sk.wikipedia.org/wiki/JavaScript>.

[24] JavaScript [online]. Dostupné na internete: <https://cs.wikipedia.org/wiki/JavaScript>.

[25] „Štandardizácia“ – význam cudzieho slova [online].[cit. 2.11.2016]. Dostupné na internete: <http://slovnik.azet.sk/slovnik-cudzich-slov/?q=%C5%A1tandardiz%C3%A1cia>.

[26] JavaScript [online].[preložené a cit. 3.11.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/JavaScript>.

[27] PETER, I. 2004. *History of the Internet – the Browser wars.* [online]. The Internet History Project, 2004. [preložené a cit. 3.11.2016]. Dostupné na internete: <http://www.nethistory.info/History%20of%20the%20Internet/browserwars.html>.

[28] DRHLÍK, K. 2015. *Android pro začátečníky #7 – Co je to widget a jak jej používat?* [online].[preložené a cit. 3.11.2016]. Dostupné na internete: <https://www.svetandroida.cz/android-pro-zacatecniky-widget-201508>.

[29] JUNA, J. 2013. *Základní vlastnosti a výhody Node.JS.* [online]. [preložené a cit. 3.11.2016]. Dostupné na internete: <http://www.nodejs.cz/vlastnosti-a-vyhody-nodejs/>.

[30] EICH, B. 2006. *Will there be a suggested file suffix for es4?* [online].[preložené a cit. 3.11.2016]. Dostupné na internete: <https://mail.mozilla.org/pipermail/es-discuss/2006-October/000133.html>.

[31] ECMAScript for XML [online].[preložené a cit. 29.11.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/ECMAScript_for_XML>.

[32] JSON [online].[preložené a cit. 3.12.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/JSON>.

[33] List of ECMAScript engines [online].[preložené a cit. 3.12.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/List_of_ECMAScript_engines>.

[34] Source-to-source compiler [online].[preložené a cit. 3.12.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/Source-to-source_compiler>.

[35] Iterator [online].[preložené a cit. 3.12.2016]. Dostupné na internete: <https://en.wikipedia.org/wiki/Iterator>.

[36] Interesting stats [online].[preložené a cit. 29.12.2016]. Dostupné na internete: <http://httparchive.org/interesting.php>.

[37] PECKA, M. *Regulární výrazy: Regexp není zaklínadlo*. [online].[preložené a cit. 22.4.2017]. Dostupné na internete: <http://www.regularnivyrazy.info/>.

[38] PECKA, M. *Základy regulárních výrazů.* [online].[preložené a cit. 22.4.2017]. Dostupné na internete: <http://www.regularnivyrazy.info/regularni-vyrazy-zaklady.html>.

[39] HUANG, J. 2015. *EcmaScript 3 VS EcmaScript 5.* [online].[preložené a cit. 25.4.2017]. Dostupné na internete: <http://www.jiehuang.net/wordPress/uncategorized/ecmascript/>.

[40] RAUSCHMAYER, A. 2016. *ES2016 feature: exponentiation operator (**).* [online]. Dostupné na internete: <http://2ality.com/2016/02/exponentiation-operator.html>.

[41] RAUSCHMAYER, A. 2016. *ES2016 feature: Array.prototype.includes*. [online].[preložené a cit. 30.4.2017]. Dostupné na internete: <http://2ality.com/2016/02/array-prototype-includes.html>.

[42] RAUSCHMAYER, A. 2017. *Feature watch: ECMAScript 2018.* [online].[preložené a cit. 30.4.2017]. Dostupné na internete: <http://2ality.com/2017/02/ecmascript-2018.html>.

[43] RAUSCHMAYER, A. 2016. *ES proposal: Template Literal Revision.* [online].[preložené a cit. 30.4.2017]. Dostupné na internete: <http://2ality.com/2016/09/template-literal-revision.html>.

[44] RAUSCHMAYER, A. 2016. *ES proposal: asynchronous iteration.* [online].[preložené a cit. 30.4.2017]. Dostupné na internete: <http://2ality.com/2016/10/asynchronous-iteration.html>.

[45] RAUSCHMAYER, A. 2017. *ES proposal: import() – dynamically importing ES modules.* [online]. Dostupné na internete: <http://2ality.com/2017/01/import-operator.html>.

[46] ECMAScript proposal: Unicode property escapes in regular expressions [online].[preložené a cit. 30.4.2017]. Dostupné na internete: <https://github.com/tc39/proposal-regexp-unicode-property-escapes>.