

**priručnik za polaznike 2019 Srce**

**Osnove *JavaScripta***

C502

Ovu inačicu priručnika izradio je autorski tim Srca:

Autor: Denis Stančer (dorada: Domagoj Horvatović)

Recenzent: Edin Mujadžević (dorada: Krešimir Tkalec)

Urednik: Sabina Rako (dorada: Dominik Kenđel)

Lektorica: Jasna Novak Milić (dorada: Mia Kožul)



Sveučilište u Zagrebu

Sveučilišni računski centar

Josipa Marohnića 5, 10000 Zagreb

edu@srce.hr

ISBN 978-953-8172-75-5 (meki uvez)

ISBN 978-953-8172-76-2 (PDF)

Verzija priručnika: C502-20191212



|  |  |
| --- | --- |
|  | Ovo djelo dano je na korištenje pod licencom *Creative Commons Imenovanje-Nekomercijalno-Dijeli pod istim uvjetima 4.0 međunarodna*. Licenca je dostupna na stranici  http://creativecommons.org/licenses/by-nc-sa/4.0/. |

**Sadržaj**

[Uvod 1](#_Toc26957287)

[1. Što je *JavaScript*? 3](#_Toc26957288)

[1.1. Naziv *JavaScript* 3](#_Toc26957289)

[1.2. Povijest *JavaScript*a 3](#_Toc26957290)

[1.3. Skriptni jezici 6](#_Toc26957291)

[1.4. Preglednici (engl. *browsers*) 7](#_Toc26957292)

[1.5. Aplikacije za uređivanje teksta (engl. *editors*) 8](#_Toc26957293)

[1.6. *Document Object Model* u *JavaScriptu* 9](#_Toc26957294)

[1.7. Sigurnost 13](#_Toc26957295)

[1.8. Vježba: Početak rada s *JavaScriptom* 14](#_Toc26957296)

[2. Upoznavanje s jezikom *JavaScript* 15](#_Toc26957297)

[2.1. Način pisanja 15](#_Toc26957298)

[2.2. Uključivanje *JavaScripta* u HTML-dokument 18](#_Toc26957299)

[2.3. Pogreške 24](#_Toc26957300)

[2.4. Vježba: Upoznavanje s jezikom *JavaScript* 25](#_Toc26957301)

[3. Varijable i objekti 27](#_Toc26957302)

[3.1. Vrste podataka 27](#_Toc26957303)

[3.2. Varijable 30](#_Toc26957304)

[3.3. Objekti 34](#_Toc26957305)

[3.4. Vježba: Varijable i objekti 36](#_Toc26957306)

[4. Operatori 37](#_Toc26957307)

[4.1. Aritmetički operatori 37](#_Toc26957308)

[4.2. Operator pridruživanja 37](#_Toc26957309)

[4.3. Operatori uspoređivanja 38](#_Toc26957310)

[4.4. Logički operatori 40](#_Toc26957311)

[4.5. Operator spajanja 40](#_Toc26957312)

[4.6. Vježba: Operatori 41](#_Toc26957313)

[5. Funkcije 43](#_Toc26957314)

[5.1. Definiranje funkcije 43](#_Toc26957315)

[5.2. Poziv funkcije 44](#_Toc26957316)

[5.3. Doseg varijabli 45](#_Toc26957317)

[5.4. Vježba: Funkcije 46](#_Toc26957318)

[6. Naredbe za kontrolu tijeka 47](#_Toc26957319)

[6.1. Uvjetno izvođenje naredbi 47](#_Toc26957320)

[6.2. Višestruka usporedba 48](#_Toc26957321)

[6.3. Uvjetni operator 49](#_Toc26957322)

[6.4. Petlja s uvjetom na početku 49](#_Toc26957323)

[6.5. Petlja s uvjetom na kraju 50](#_Toc26957324)

[6.6. Petlja s poznatim brojem ponavljanja 50](#_Toc26957325)

[6.7. Vježba: Naredbe za kontrolu tijeka 51](#_Toc26957326)

[7. Obrasci 53](#_Toc26957327)

[7.1. Prvi obrazac 53](#_Toc26957328)

[7.2. Unos kraćih nizova znakova 54](#_Toc26957329)

[7.3. Izrada HTML-forme 58](#_Toc26957330)

[7.4. Izrada funkcija u *JavaScriptu* 63](#_Toc26957331)

[7.5. Vježba: Obrasci 71](#_Toc26957332)

[8. *JavaScript* biblioteka – *jQuery* 73](#_Toc26957333)

[8.1. Općenito o *JavaScript* bibliotekama 73](#_Toc26957334)

[8.2. *jQuery* 73](#_Toc26957335)

[8.3. Prerada obrazaca uz pomoć biblioteke *jQuery* 75](#_Toc26957336)

[8.4. Napredni primjeri (*jQuery*) 81](#_Toc26957347)

[9. Korisne skripte 85](#_Toc26957348)

[9.1. *Rollover* 85](#_Toc26957349)

[9.2. Preusmjeravanje 86](#_Toc26957350)

[9.3. Provjera pomoću regularnih izraza 87](#_Toc26957351)

[9.4. Upravljanje preglednikom 92](#_Toc26957352)

[9.5. Vježba: Korisne skripte 96](#_Toc26957353)

[10. Dodaci 97](#_Toc26957354)

[10.1. Zadaci 97](#_Toc26957357)

[10.2. Kompletan kôd obrasca iz cjeline 7. 98](#_Toc26957358)

[10.3. Rješenja vježbi 102](#_Toc26957359)

[10.4. Dodatni materijali 113](#_Toc26957360)

# Uvod

U okviru ovog tečaja upoznat ćete se s osnovama programiranja u *JavaScript*u. *JavaScript* je skriptni jezik (danas *de facto* standard) kojim se u statičke HTML-stranice mogu uvesti interaktivni elementi.

Za uspješno praćenje tečaja o *JavaScript*u potrebno je znati osnovno o radu u operacijskom sustavu, poznavati sintaksu HTML-a, a poželjna su i osnovna znanja iz programiranja.

Ovaj se priručnik sastoji od jedanaest poglavlja koja će biti detaljno obrađena u 12 školskih sati.

U ovom se priručniku za označavanje važnijih pojmova rabe podebljana slova. Nazivi datoteka oblikovani su podebljanim slovima i kurzivom.

Prečaci na tipkovnici označeni su ovako: [Ctrl]+[Alt]+[Del], [F1], a programski se kôd prikazuje posebnim oblikovanjem:



Napomene se nalaze u okvirima sa strane.

1. Što je *JavaScript*?

*Po završetku ovog poglavlja polaznik će moći:*

opisati razliku između programskih i skriptnih jezika

objasniti hijerarhiju za prikaz i interakciju s objektima u HTML dokumentu (DOM)

primijeniti JavaScript naredbe za dohvaćanje elemenata iz DOM-a.

*JavaScript* je jednostavan, interpretiran (interpreterski) programski jezik namijenjen ponajprije razvoju interaktivnih HTML-stranica. Jezgra *JavaScript*a uključena je u većinu današnjih preglednika (*Internet Explorer, Google Chrome, Mozilla Firefox, Opera, Safari* i drugi).

* 1. Naziv *JavaScript*

*JavaScript* omogućuje izvršavanje određenih radnji u inače statičnim HTML-dokumentima, npr. interakciju s korisnikom, promjenu svojstava preglednikova prozora ili dinamičko stvaranje HTML-sadržaja.

*JavaScript* nije pojednostavljena inačica programskog jezika *Java*. Povezuje ih jedino slična sintaksa i to što se koriste za izvršavanje određenih radnji unutar preglednika. Izvorno se *JavaScript* trebao zvati *LiveScript*, ali da bi se potakla uporaba novog skriptnog jezika, nazvan je slično jeziku *Java*, od kojeg se u tadašnje vrijeme dosta očekivalo.

* 1. Povijest *JavaScript*a

*JavaScript* se razvija od 1995. godine kada je *Netscape* objavio nekoliko prvih inačica jezika. Nedugo nakon toga *Microsoft* je objavio jezik sličan *JavaScript*u pod nazivom *JScript*. Danas je za standardizaciju skriptnih jezika, pa tako i *JavaScript*a, zadužena organizacija *ECMA* (<http://www.ecma.ch/>). Standardi se objavljuju pod nazivom *ECMAScript* i do sada je objavljeno pet inačica standarda *ECMA-262.*

U ovom priručniku pojam *JavaScript* označava bilo koju implementaciju jezika, uključujući i *Microsoft*ov *JScript*.

| **Inačica** | **Opis** |
| --- | --- |
| *JavaScript 1.0* | Izvorna inačica jezika, bila je puna pogrešaka. Implementirana u preglednik *Netscape 2*. |
| *JavaScript 1.1* | Dodan novi objekt Array (za rad s poljima); popravljene ozbiljne pogreške. Implementiran u preglednik *Netscape 3*. |
| *JavaScript 1.2* | Dodana nova naredba switch, regularni izrazi i drugo. Djelomično poštuje *ECMA v1* uz neke nekompatibilnosti. Implementiran u preglednik *Netscape 4*. |
| *JavaScript 1.3* | Ispravljene nekompatibilnosti *JavaScripta 1.2*. Usklađenost sa standardom *ECMA v1*. Implementiran u preglednik *Netscape 4.5.* |
| *JavaScript 1.4* | Implementiran samo u preglednik *Netscape* za poslužiteljske proizvode. |
| *JavaScript 1.5* | Uvedeno upravljanje iznimkama (engl. *exception handling*). Poštuje standard *ECMA v3*. Implementiran u preglednike *Mozilla Firefox* i *Netscape 6*. |
| *JScript 1.0* | Okvirno ekvivalentan *JavaScriptu 1.0*. Implementiran u prve inačice *preglednika Internet Explorer 3*. |
| *JScript 2.0* | Okvirno ekvivalentan *JavaScriptu 1.1*. Implementiran u kasnije inačice preglednika *Internet Explorer 3*. |
| *JScript 3.0* | Okvirno ekvivalentan *JavaScriptu 1.3*. Usklađen sa standardom *ECMA v1*. Implementiran u preglednik *Internet Explorer 4*. |
| *JScript 4.0* | Nije implementiran ni u jedan preglednik. |
| *JScript 5.0* | Podržano upravljanje iznimkama (engl. *exception handling*). Djelomično poštuje standard *ECMA v3*. Implementiran u preglednik *Internet Explorer 5*. |
| *JScript 5.5* | Okvirno ekvivalentan *JavaScriptu 1.5*. Potpuno usklađen sa standardom *ECMA v3*. Implementiran u preglednike *Internet Explorer 5.5* i *6*. (*IE 6* zapravo ima *JScript 5.6*, ali *5.6* se značajno ne razlikuje od *5.5* za *JavaScript* programere koji pišu za preglednike). |
| *ECMA v1* | Prva standardna inačica. Standardizirane su osnove *JavaScripta 1.1* i dodano je nekoliko novih mogućnosti. Nisu standardizirani naredba switch i regularni izrazi. Implementacije koje poštuju standard *ECMA v1* su *JavaScript 1.3* i *JScript 3.0*. |
| *ECMA v2* | Razvojna inačica koja nije donijela nove mogućnosti, ali je razjasnila dvosmislenosti. |
| *ECMA v3* | Standardizirani naredba switch, regularni izrazi i upravljanje iznimkama. Implementacije usklađene sa standardom *ECMA v3* su *JavaScript 1.5* i *JScript 5.5*. |
| *ECMA v4* | Ova inačica nikada nije zaživjela zbog nesuglasica u radnoj skupini u kojem bi se smjeru jezik trebao razvijati. |
| *ECMA v5* | Dodan način rada strict, razjašnjene su mnoge dvosmislenosti iz standarada *ECMA v3* i dodana je podrška za *JSON*. |
| *ECMA v5.1* | Dorađena inačica 5 koja ne sadržava nikakve novitete vezane uz sâm jezik. |
| *ECMA v6* | Očekuje se sredinom 2015. godine, a treba dodati novu sintaksu za objektno programiranje i još neke dodatke. Ta je inačica poznata i kao *ES6 Harmony*.  Dodane dvije nove ključne riječi *let* i *const* koje imaju doseg unutar jednog bloka (npr. for, if, …), dodane zadane vrijednosti parametara, te dvije nove naredbe *Array.find()* i *Array.findIndex()* |
| *ECMA v7* | Dodan eksponencijalni operator \*\* i naredba *Array.prototype.includes* |
| *ECMA v8* | Dodano punjenje (povećanje) nizova znakova, nova svojstva objekta, *Async* funkcije i dijeljenje memorije |
| *ECMA v9* | Dodana *rest/spread* svojstva, asinkrona iteracija, naredba *Promise.finally()*, te nova svojstva unutar *RegExp* (regularnih izraza) |

* 1. Skriptni jezici

Program koji obrađuje i izvršava skripte zove se *interpreter*. *Interpreter* čita kôd i prevodi ga u strojni jezik svakog puta kada se pokrene skripta. Svaki jezik koji se *interpret*ira, tj. koji izvršava *interpreter*, naziva se **skriptni jezik**. *Interpreter* za *JavaScript* ugrađen je u većinu današnjih preglednika (nema ga u nekim preglednicima baziranim na tekstu, npr. *Lynx*, *Line Mode Browser*, *Charlotte Web Browser*, zato jer preglednici bazirani na tekstu renderiraju samo tekst *web-*stranica).

Skriptni se jezici koriste jer je razvoj programa znatno jednostavniji. Za razliku od programa pisanih u pravim programskim jezicima, kôd skriptnog jezika ne treba prevoditi skripte u strojni jezik:

|  |  |
| --- | --- |
| **Koraci kod programskih jezika** | **Koraci kod skriptnih jezika** |
| 1. Napisati ili popraviti program. 2. Prevesti program u strojni jezik. 3. Pokrenuti prevedeni program. 4. Za popravke ponoviti od 1. koraka. | 1. Napisati ili popraviti skriptu. 2. Pokrenuti interpreter*.* 3. Za popravke ponoviti od 1. koraka. |

HTML je jezik koji se koristi za opis dokumenata i nema dinamičnih elemenata. Davno se ukazala potreba za uvođenjem dinamičnog načina stvaranja HTML-elemenata i stvaranje interaktivnog sadržaja u HTML-u.

Danas postoji nekoliko tehnologija za stvaranje interaktivnog sadržaja u HTML-dokumentima:

1. U prvoj skupini su tehnologije za dinamično stvaranje HTML-a, tj. stranice su zapisane pomoću nekog (obično skriptnog) jezika, koji interpretira poslužitelj i šalje korisniku HTML-kôd. Tipični su predstavnici ove skupine ASP.NET, PHP (PHP: *Hypertext Preprocessor*), a mogu se koristiti i *Java*, *Ruby (Ruby on Rails)*, *Python* i *Perl*.
2. U drugoj skupini su *Java* (u obliku *applet*a), *Flash* i *Shockwave* za čiji je prikaz potreban vanjski program (*plug-in*) koji zna interpretirati ili izvoditi navedene programe. Kad je sadržaj prikazan u pregledniku, on je dinamičan i neovisan o „okolnom” HTML-u.

**Napomena**

*Java* *applet*i su manji programi napisani u *Javi* koji se izvršavaju u pregledniku.

1. U trećoj su skupini tzv. klijentski jezici, jer se njihov kôd interpretira na klijentskoj strani, tj. u pregledniku (klijentu). Glavni predstavnik klijentskih jezika je *JavaScript*. Nekada je bio značajan i skriptni jezik *VBScript* koji se koristio u starijim inačicama preglednika *Internet Explorer*a.

Primjer toka podataka od *web*-poslužitelja do *web*-preglednika:

**PHP,   
ASP,**

**…**

**POSLUŽITELJ**

**INTERNET**

**HTML, *JavaScript***

**JVM**

**JAVA**

**PREGLEDNIK**

**HTML, *JavaScript, Java*, ...**

***Java Virtual Machine***

Za PHP to izgleda ovako:

**PHP**

**POSLUŽITELJ**

**INTERNET**

**HTML i *JavaScript***

**PREGLEDNIK**

**HTML i *JavaScript***

* 1. Preglednici (engl. *browsers*)

Danas je podrška za *JavaScript* izvrsna u svim preglednicima, tako da se autori preglednika više ne natječu u podršci nego u brzini izvođenja određenih algoritama.

Popis preglednika koji podržavaju standard *ECMA-262* inačica 5:

* *Internet Explorer 10+*
* *Firefox 21+*
* *Safari 6+*
* *Chrome 23+*
* *Opera 15+.*

Popis preglednika koji podržavaju standard *ECMA-262* inačica 6:

* *Internet Explorer 11*
* *Firefox 66+*
* *Safari 12.1+*
* *Chrome 74+*
* *Opera 58+*

Osim utrke u brzini autori preglednika natječu se u jednostavnosti razvoja, odnosno tko će omogućiti što više dodatnih značajki za razvoj. Što je bolja podrška za razvoj, to će se više razvijatelja koristiti tim preglednikom i preporučivati ga svojim klijentima. Tako raste i broj korisnika tog preglednika.

Četiri najčešće korištena preglednika (*Internet Explorer i Edge, Google Chrome*, *Safari i Mozilla Firefox*) danas imaju alate za razvijatelje (d*eveloper tools*) do kojih se u svim preglednicima dolazi jednako – odabirom tipke F12 (osim Safari preglednika u kojem se dolazi odabirom Ctrl + Alt + I).

Od razvijateljskih značajki za ovaj su tečaj najzanimljivije *Console* i DOM (Document Object Model).

* 1. Aplikacije za uređivanje teksta (engl. *editors*)

*JavaScript* se može pisati u bilo kojem uređivaču teksta (*editor*) koji podržava standard *ASCII*. Blok za pisanje (*Notepad*) prisutan je u svim inačicama operacijskog sustava *Windows* pa se nameće kao pogodan i za programiranje u *JavaScript*u, međutim to se ne preporuča jer je kôd nepregledan i može vrlo lako doći do greške koju je onda teško otkriti.

Da bi se programiranje ubrzalo i učinilo učinkovitijim, poželjno je rabiti uređivač teksta koji ima ova svojstva (popis preporučenih uređivača nalazi se u zadnjem poglavlju):

* mogućnost otvaranja više dokumenata u jednom prozoru
* razlikuje dijelove kôda i prikazuje ih u različitim bojama (bojanje sintakse)
* podrška za UTF-8
* automatsko poravnavanje (*tidy*)
* provjera stila kodiranja u skladu s preporučenim postupcima (*linter*).
  1. *Document Object Model* u *JavaScriptu*

*Document* *Object Model* (DOM) je model za prikaz i interakciju s objektima u HTML-dokumentu. Omogućava jednoznačan i jednostavan pristup dijelovima (HTML-) dokumenta te rukovanje njegovim dijelovima (npr. elementi u HTML-dokumentu).

*JavaScript* definira svoj DOM u obliku ovakve hijerarhijske strukture:



Svakom objektu ili svojstvu pristupa se kroz taj model, tj. *document* je osnovni objekt preko kojeg se pristupa svim drugim objektima dokumenta.

Na primjer, u dokumentu koji sadrži ovaj programski kôd:

Vrijednosti polja u koje se upisuje ime pristupa se ovako:

document.forms[0].elements[0].value

Vidljivo je da je tekstni objekt prvi u polju elemenata koji se nalaze u prvom obrascu. Da bi se programiranje pojednostavilo, omogućeno je imenovanje pojedinih objekata. Na primjer, dodavanjem naziva obrascu (name=**"**frm\_a**"**) i dodavanjem naziva tekstnom polju za unos imena (name=**"**ime**"**) dobije se:



Vrijednosti polja u tom slučaju pristupamo jednostavnije:

document.**frm\_a.ime**.value

Najjednostavniji način dohvata vrijednosti polja je pomoću jedinstvenog identifikatora (ID). Naime, preporuka je da vrijednost atributa ID mora biti jedinstvena na razini dokumenta. Prvobitni se dokument može dodatno urediti dodavanjem atributa ID (id=“ime“) za polje za unos imena: 

Vrijednosti polja u tom slučaju pristupamo još jednostavnije:

document.getElementById(**'ime'**).value

Potrebno je naglasiti da atribut name i dalje treba biti naveden zbog načina na koji se obrađuju obrasci.

Vrijednost polja također se može dohvatiti upotrebom atributa klasa (class). Za razliku od jedinstvenog identifikatora više elemenata može imati istu klasu.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>Popis</title>

</head>

<body>

<form action="">

Unesite ime:

<input type="text" name="ime" class="ime" value="" />

<br/>

<input type="submit" value="Pošalji" onclick="return provjeri();" />

</form>

</body>

</html>

U ovom slučaju vrijednosti polja pristupamo na slijedeći način:

document.getElementsByClassName(**'ime'**)[0].value

U slučaju da dohvaćamo više elemenata sa istom klasom kroz njih možemo proći pomoću for ili for-of petlje:

var klase = document.getElementsByClassName(**'ime'**);

for(var i = 0; i < klase.length; i++){

document.alert(klase[i].value);

}

var klase = document.getElementsByClassName(**'ime'**);

for(klasa of klase){

document.alert(klasa.value);

}

Objekt window također je važan objekt pomoću kojeg se upravlja prozorom preglednika. U jednom od primjera detaljnije će se obraditi uporaba objekta window.

Dvije često korištene funkcije vezane su uz objekte document i window:

* document.write – funkcija koja upisuje niz znakova koji se proslijedi kao prvi argument u HTML-dokument na mjestu gdje se fukncija poziva
* window.alert – funkcija ispisuje niz znakova koji se proslijedi kao prvi argument u zasebnom prozoru i ne dopušta nastavak izvršavanja programa dok se ne zatvori.
  1. Sigurnost

Kad god se programi (kao što su skripte *JavaScript*a, *Java*-programi ili makronaredbe programa *Microsoft Word*) nalaze u odijeljenim dokumentima, osobito u dokumentima koji se šalju Internetom, prisutna je opasnost od virusa i drugih zloćudnih programa. Tvorci *JavaScript*a bili su svjesni tih sigurnosnih problema i onemogućili su programima *JavaScript*a postupke koji bi za posljedicu imali brisanje ili izmjenu podataka na korisnikovu računalu. Kao što je već naglašeno, programi *JavaScript*a ne mogu pristupati lokalnim datotekama, tj. ne mogu zaraziti druge datoteke ili brisati postojeće.

Isto tako, programi *JavaScript*a ne mogu obavljati mrežne radnje, tj. *JavaScript* može učitavati adrese *web*-sadržaja (URL) i slati podatke iz HTML-obrazaca poslužiteljskim skriptama, ali ne može ostvarivati neposredne veze s drugim računalima i tako pokušati pogoditi lozinku na nekom lokalnom poslužitelju.

Međutim, budući da su preglednici složeni programi, u početku implementacija interpretera *JavaScript*a nije uvijek poštivala propisane standarde. Na primjer, preglednik *Netscape 2* (objavljen 1995. godine) omogućavao je pisanje programa *JavaScript* koji je automatski dohvatio adresu elektroničke pošte bilo kojeg posjetitelja određene stranice i u njegovo ime, bez njegove potvrde, poslao e-mail. Taj je propust, uz niz drugih, manje opasnih, popravljen. Međutim, ne postoji jamstvo da se neće otkriti novi propusti u implementaciji *JavaScript*a i tako omogućiti zlonamjernicima iskorištavanje tih propusta.

* 1. Vježba: Početak rada s *JavaScriptom*

1. Izradite HTML-datoteku naziva Vjezba1.html ovog sadržaja:



1. Elementu input, tipa text dodajte atribut name i dajte mu vrijednost po svom izboru.
2. *JavaScript* program pišite unutar elementa <script>.
3. Upišite svoje ime unutar input polja, unutar funkcije poruka dohvatite tu vrijednost pomoću atributa name, te ju sa window.alert() funkcijom ispišite na ekranu.
4. Elementu input dodajte atribut id i dajte mu vrijednost po svom izboru.
5. Unutar funkcije poruka ponovno dohvatite vrijednost input polja, sada pomoću atributa id, te ju sa window.alert() funkcijom ispišite na ekranu.
6. Zamijenite atribut id sa atributom class, te pomoću njega dohvatite vrijednost input polja i ispišite njegovu vrijednost sa window.alert() funkcijom.

1. Upoznavanje s jezikom *JavaScript*

*Po završetku ovog poglavlja polaznik će moći:*

* opisati način pisanja naredbi, varijabli i komentara u jeziku JavaScript
* povezati JavaScript s HTML-dokumentom
* razlikovati načine pozivanja JavaScript kôda
* prepoznati grešku u kôdu korištenjem web-preglednika.

2.1. Način pisanja

*JavaScript* se može pisati prema standardu *Unicode*, no preporuča se pisanje prema standardu *ASCII*, osim u komentarima i nizovima znakova.

**2.1.1. Velika i mala slova**

**Napomena**

Velika i mala slova su najčešći uzrok početničkih pogrešaka u *JavaScript*u.

Prilikom pisanja sintakse *JavaScript*a važno je zapamtiti:

***JavaScript* razlikuje velika i mala slova!**

To znači da se ključne riječi, varijable, funkcije i drugi nazivi moraju pisati dosljedno s obzirom na velika i mala slova. Tako se, na primjer, ključna riječ mora pisati while, a ne While ili WHILE. Također, varijable stranaa, stranaA, StranaA i STRANAA četiri su različite varijable.

Radi izbjegavanja pogrešaka, preporuča se dosljednost prilikom korištenja malih i velikih slova u nazivima. Dobra praksa je slijediti način imenovanja koji se već koristi za nazive postojećih funkcija i svojstava u *JavaScript*u, a to je tzv. *camelCasing*. Nazivi se pišu malim slovima, a ako naziv sadrži dvije ili više riječi, druga i svaka sljedeća riječ počinju velikim slovom (npr. getElementById).

**Napomena**

Iznimka od ovog načina pisanja su nazivi svojstava objekata DOM-a koji se svi pišu malim slovima, npr. onclick.

2.1.2. Znak za završetak naredbe

Znakom kojim se u *JavaScript*u označava kraj naredbe je točka-zarez (;). Točka-zarez nije obavezan znak, ali se njegova uporaba preporuča. Naime, *JavaScript* ubacuje točku-zarez na kraju retka ako pojedini kôd izgleda kao naredba, što u nekim situacijama nije to što je programer htio. Na primjer kôd:



*JavaScript* će interpretirati kao:



(dakle ubačena je točka-zarez na kraju retka), iako je programer htio:



**2.1.2. Komentari**

U *JavaScript*u se komentari označavaju jednako kao i u programskim jezicima C i C++. Bilo koji tekst između // i kraja retka smatra se komentarom. Bilo koji tekst između znakova /\* i \*/ je komentar i zanemaruje se. Druga vrsta komentara može se protezati u više redaka, ali se ne smiju ugnježđivati. Evo nekoliko ispravnih komentara:



2.1.3. Varijable

Nazivi varijabli i funkcija trebaju zadovoljavati ovo pravilo: prvi znak u nazivu treba biti slovo, podcrta ( \_ ) ili dolarski znak ($). Znakovi koji slijede mogu biti slova, brojevi, podcrta ili dolarski znak. U praksi je najbolje pisati varijable tako da započinju slovom i imaju imena koja jasno objašnjavaju njihovu namjenu. Ne treba se bojati varijablama dati dugačka imena ukoliko će ona time biti laka za raspoznati unutar kôda. Primjeri ispravnih naziva:



Nazivi ne smiju biti isti kao ključne riječi. U tablici je naveden popis ključnih riječi u *JavaScript*u koje treba izbjegavati kod imenovanja varijabli:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| break | do | if | switch | typeof |
| case | else | in | this | var |
| catch | false | instanceof | throw | void |
| continue | finally | new | true | while |
| default | for | null | try | with |
| delete | function | return | let | await |
| debugger | yield |  |  |  |

Ova se imena trenutačno ne rabe u *JavaScript*u, ali ih standard *ECMAScript v3* navodi kao moguće ključne riječi u budućnosti, pa bi i njih trebalo izbjegavati:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Abstract | double | goto | native | static |
| boolean | enum | implements | package | super |
| byte | export | import | private | synchronized |
| char | extends | int | protected | throws |
| class | final | interface | public | transient |
| const | float | long | short | volatile |

Također treba izbjegavati nazive globalnih varijabli i funkcija predefiniranih u *JavaScript*u:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| arguments | encodeURI | Infinity | Object | String |
| Array | Error | isFinite | parseFloat | SyntaxError |
| Boolean | escape | isNaN | parseInt | TypeError |
| Date | eval | Math | RangeError | undefined |
| decodeURI | EvalError | NaN | ReferenceError | unescape |
| decodeURIComponent | Function | Number | RegExp | URIError |
| hasOwnProperty | isPrototypeOf | length | name | prototype |
| toString | valueOf |  |  |  |

Također treba izbjegavati upotrebu imena HTML i *Window* objekata i svojstva:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| alert | all | anchor | anchors | area |
| assign | blur | button | checkbox | clearInterval |
| clearTimeout | clientInformation | close | closed | confirm |
| constructor | crypto | defaultStatus | document | element |
| elements | embeded | embeds | encodeURI | endcodeURIComponent |
| escape | event | fileUpload | focus | form |
| forms | frame | innerHeight | innerWidth | layer |
| layers | link | location | mimeTypes | navigate |
| navigator | frames | frameRate | hidden | history |
| image | images | offscreenBuffering | open | opener |
| option | outerHeight | outerWidth | packages | pageXOffset |
| pageYOffset | parent | parseFloat | parseInt | password |
| pkcs11 | plugin | prompt | propertyIsEnum | radio |
| reset | screenX | screenY | scroll | secure |
| select | self | setInterval | setTimeout | status |
| submit | taint | text | textarea | top |
| unescape | untaint | window |  |  |

2.2. Uključivanje *JavaScripta* u HTML-dokument

*JavaScript* se ovako može uključiti u HTML-dokument:

* pisanjem kôda između oznaka <script> i </script>
* iz vanjske datoteke uporabom atributa src u oznaci <script>
* preko obrade događaja navedenog kao HTML-atribut (npr. onclick ili onmouseover)
* u URL-u koji se koristi posebnim protokolom javascript:

U praksi, najbolji način je pisanje HTML, CSS i *JavaScript* kôda u zasebnim datotekama. Događaji kao što su *onclick* i *onmouseover* također se mogu pozivati i definirati unutar *JavaScript* datoteke, te se na taj način u potpunosti odvaja *JavaScript* kôd od HTML-a.

2.2.1. Unutar HTML-datoteke

***HTML***

**JavaScript**

Unutar HTML-dokumenta *JavaScript*-skripte pišu se između oznaka <script> i </script>. Proizvoljan broj *JavaScript*-naredbi obrađuje se prilikom učitavanja HTML-dokumenata i to redom kojim su napisne. Element <script> može se pojaviti u elementu <head> ili u elementu <body>.

U elementu <head> uobičajeno je pisati programe koji očekuju neku korisnikovu radnju, tj. učitaju se u preglednik i čekaju događaj koji bi ih pokrenuo.

U elementu <body> uobičajeno je pisati programe koji se izvršavaju odmah pri učitavanju. Najčešće je to generiranje nekog dijela HTML‑dokumenta.

Najbolja praksa je pisanje *JavaScript* kôda na dnu <body> elementa, tj. točno prije nego se <body> element zatvori. Na taj način korisniku će se prvo učitati vizualni dio stranice, a potom funkcionalni koji će dodati interaktivnost stranici. Ovisno o načinu na koji se elementima HTML-a pridružuje neki *JavaScript* kôd, tj. poziv funkcije, *JavaScript* skripta se može ili mora nalaziti/pozivati nakon učitavanja DOM-a stranice kako bi *JavaScrip*t kôd funkcionirao.

Jedan HTML-dokument može sadržavati nekoliko elemenata <script>. Te odvojene skripte obrađuju se redom kojim su napisane u dokumentu. Odvojene skripte čine jedan *JavaScript*-program u jednom HTML‑dokumentu, tj. funkcije i varijable definirane u jednoj skripti dostupne su u svim sljedećim skriptama u istom HTML-dokumentu.

Iako je *JavaScript* najčešće rabljen skriptni jezik u HTML-dokumentima, on nije jedini. Vrsta skriptnog jezika navodi se u atributu type elementa <script>. Preglednici koji imaju *interpreter* za navedeni jezik obrade skriptu, a ostali ju zanemare.

Primjena atributa type:



Danas više nisu u uporabi preglednici koji ne prepoznaju element <script> pa se sve rjeđe koristi „skrivanje” kôda *JavaScript*:



Ovako gornji kôd interpretira preglednik koji ne prepoznaje element <script>:

* Nepoznati element (<script>).
* W3C preporuka daje uputu da se sadržaj nepoznatog elementa prikaže u pregledniku.
* Preglednik prikazuje sadržaj elementa <script>, a to je komentar, odnosno ne prikazuje ništa.

S druge strane, preglednik koji prepoznaje element <script> kôd interpretira ovako:

* Preglednik zna kako obraditi element <script>.
* Je li prvi redak iz elementa <script> početak HTML-komentara (odnosno 4 znaka <!--).
* Ako da, tada se ignorira taj prvi redak i ostatak se sadržaja elementa <script> interpretira kao *JavaScript.*
* Zato posljednji redak u elementu <script> ima *JavaScript*‑komentar (//) ispred završetka HTML-komentara (-->).

2.2.2. *JavaScript* u zasebnoj datoteci

Od inačice 1.1 *JavaScript* u elementu <script> podržava atribut src. Vrijednost tog atributa je URL datoteke koja sadrži *JavaScript*-program.



Datoteka *JavaScript* obično ima nastavak .js i sadrži čisti *JavaScript*, bez elementa <script>.

Bilo koji kôd unutar oznaka zanemaruje se. Nužno je primijetiti da je završna oznaka </script> obavezna iako je atribut src naveden i iako nema *JavaScript*- kôda između oznaka <script> i </script>.

Nekoliko je razloga za uporabu atributa src:

* Pojednostavljuje HTML-datoteku i smanjuje njezinu veličinu, jer nema velikih blokova *JavaScript*a u dokumentu. Smanjenjem veličine datoteka će se brže preuzeti s poslužitelja.
* Ako se neka funkcija *JavaScript*a rabi u više HTML-dokumenata, poziva se jedna datoteka, što omogućuje jednostavnije održavanje.
* Kada više HTML-dokumenata rabi jednu datoteku *JavaScript*a, ona se učitava u preglednik samo prilikom prve uporabe. Nakon toga se datoteka lokalno sprema na disk i svakog sljedećeg puta interpretira se iz lokalne datoteke.
* Budući da atribut src sadrži URL, program *JavaScript* s jednog poslužitelja može se rabiti na nekoliko drugih (npr. src="http://www.server.hr/js/obrasci.js").

Upotrebom novih *boolean* atributa *async* i *defer* može se odrediti kada će se koja skripta učitati i izvršiti bez obzira na to gdje se u HTML*-*dokumentu skripta poziva. *Async* atribut učitava *JavaScript* datoteku asinkrono sa ostatkom *web*-stranice, međutim izvršava ju čim se datoteka učita, te time produljuje učitavanje ostatka stranice, ukoliko se stranica nije do kraja učitala. Ovaj način učitavanja pogodan je za skripte koje ne ovise o drugim skriptama kako bi pravilno funkcionirale.

*Defer* atribut, jednako kao i *async* atribut, učitava *JavaScript* datoteku asinkrono sa ostatkom *web*-stranice, međutim ne izvršava ju sve dok se cijela stranica ne učita. Ako se više skripti poziva upotrebom *defer* atributa, tada se gleda redoslijed kojim se skripte pozivaju u kôdu, kako bi se osigurao pravilan rad cijelog *JavaScript* kôda.

U svakom slučaju, upotreba *async* i *defer* atributa ubrzava cijelokupno učitavanje *web*-stranice, što je posebno važno kod većih *web*-stranica.

Najbolji način pozivanja *JavaScript* skripti upotrebom *async* i *defer* atributa je tako da pozive smjestimo u <head> atribut HTML-dokumenta.

<script async type=“text/javascript“ src=“javascript.js“></script>

<script defer type=“text/javascript“ src=“javascript.js“></script>

2.2.3. Obrada događaja

Program *JavaScript* izvršava se samo jednom i to kada se HTML‑dokument učita u preglednik. Takav način uporabe *JavaScript*a ne omogućuje interakciju s korisnikom. Zbog toga je za većinu HTML‑elemenata definirano više događaja.

HTML-elementi imaju posebne atribute pomoću kojih se može povezati kôd *JavaScript*a s određenim događajem. Ti atributi počinju slovima on... , a kao vrijednost atributa navode se naredbe *JavaScript*a koje će se izvršiti kad se dogodi taj događaj.



U gornjem primjeru pritiskom korisnika na dugme „Poruka“ dogodi se događaj click, izvršava se naredba navedena u atributu onclick. U primjeru se poziva funkcija window.alert koja korisniku prikazuje obavijest.

Kao vrijednost atributa može se navesti nekoliko naredbi *JavaScript*a, no ako je naredbi previše, uobičajeno je da se napiše funkcija (u elementu <script> ili u zasebnoj datoteci) te se zatim poziva ta funkcija. Tako se smanjuje miješanje programskog kôda *JavaScript*a i sadržaj HTML‑dokumenta.

2.2.4. *JavaScript* u URL-u

Kôd *JavaScript*a može se uključiti u HTML i pomoću pseudoprotokolne oznake javascript u URL-u. Taj posebni protokol označava da je tijelo stranice iza navedenog URL-a proizvoljan *JavaScript*-program koji će obraditi interpreter. Ako *JavaScript*-program u javascript: URL-u ima više naredbi, one moraju biti odvojene točka-zarezom. Na primjer:



Kad preglednik učita takav *JavaScript* URL, izvede se *JavaScript-*program, izračuna se vrijednost posljednje naredbe i ta se vrijednost pretvori u niz znakova. Konačno, dobiveni niz znakova prikaže se u pregledniku.

Kad *JavaScript* URL sadrži program koji ne vraća nikakvu vrijednost, preglednik izvrši program, ali ne mijenja sadržaj dokumenta.



Vrlo često se javascript: URL koristi za izvršavanje programa *JavaScript*a bez promjene sadržaja dokumenta. To se postiže tako da se osigura da posljednja naredba u URL-u ne vraća nikakvu vrijednost. Jedan od načina je uporaba operatora void.

Sljedeći primjer pokazuje javascript: URL koji otvara novi prazan prozor, bez promjene sadržaja trenutačnog dokumenta:



Bez operatora void u prozoru preglednika bila bi prikazana vrijednost posljednje naredbe pretvorena u niz znakova. U našem slučaju to je window.open(). Niz znakova bio bi: [object] (ili [object Window] u pregledniku *Mozilla* *Firefox*, a u pregledniku *Google Chrome* ta se vrijednost ne ispisuje).

Takav javascript: URL može se rabiti na svim mjestima gdje se koristi uobičajen URL. Na primjer, za brzo testiranje može se u pregledniku u polje *Adresa* (*Location*) upisati bilo koji *JavaScript*-program (naravno kraći).

Česta uporaba takvog oblika je kao vrijednost atributa action u elementu <form>.

2.3. Pogreške

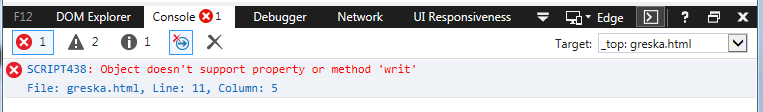
Prilikom pogreške u izvođenju programa *JavaScript*a, pojedini preglednici to prikazuju različito. Ako se program *JavaScript*a ne ponaša kako se očekuje, najbolje je pokrenuti F12, konzolu *developer tools,* jer se tada u njoj mogu pratiti sve pogreške koje se javljaju tijekom izvođenja programa.

U ovoj skripti pogreška je u 11. retku (*Line 13*) i 5. stupcu (*Col 5*).

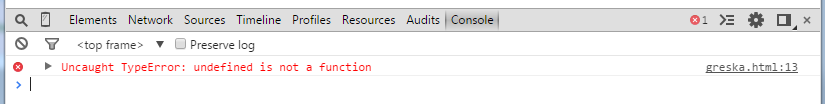


Pogreška je opisana u polju *Pogreška* (*Error*). Naime, opis pogreške navodi da objekt ne podržava metodu (tj. objekt document ne podržava metodu writ). Drugim riječima, metoda write pogrešno je napisana.

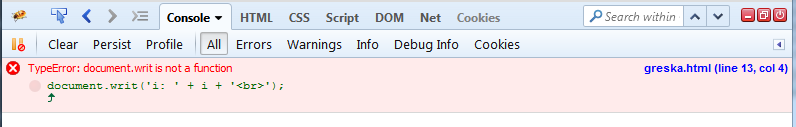
Izgled pogreške u pregledniku *Internet Explorer*:



Izgled pogreške u pregledniku *Google Chrome*:



Izgled pogreške u pregledniku *Mozilla Firefox*:



2.4. **Vježba: Upoznavanje s jezikom *JavaScript***

1. Izradite HTML-datoteku naziva Vjezba2.html ovog sadržaja:



1. Unutar elementa <script> dodajte komentar po izboru.
2. Stvorite dvije nove varijable, te u njih spremite proizvoljan niz znakova (npr. Danas je ponedjeljak.). Vrijednosti varijabli ispišite na ekranu pomoću document.write() funkcije.
3. Stvorite novu *JavaScript* datoteku imena ***javascript.js***. Sav kôd unutar <script> elementa kopirajte u novu datoteku, pozovite ju upotrebom <script> elementa, te atributa type i src u zaglavlju datoteke ***Vjezba2.html***.
4. Unutar elementa input, tipa submit dodajte onclick atribut koji će pozvati funkciju imena pozdrav.
5. Unutar datoteke ***javascript.js*** stvorite funkciju imena pozdrav. Unutar nje deklarirajte varijablu ime, te u nju spremite vrijednost input polja sa id atributom name. Pomoću funkcije window.alert() ispišite poruku koja će ispisati tekst „Bok, “ i vaše ime koje ste upisali u input polju.
6. Varijable i objekti

*Po završetku ovog poglavlja polaznik će moći:*

razlikovati vrste podataka

objasniti inicijalizaciju i korištenje varijabli

izraditi objekt u JavaScriptu

objasniti pristupanje svojstvima i metodama objekta.

3.1. Vrste podataka

3.1.1. Brojevi

Svi brojevi u *JavaScript*u prikazuju se kao brojevi s pomičnim zarezom (*floating-point*) u 64-bitnom formatu, tj. mogu se prikazati brojevi koji nisu veći (po apsolutnoj vrijednosti) od 1,7976931348623157 x 10308 i nisu manji od 5 x 10 -324.

Cijeli brojevi mogu se prikazati od -9007199254740992 (-253) do 9007199254740992 (253).

Osim brojeva s bazom 10, *JavaScript* prepoznaje i brojeve s bazom 16 (heksadecimalne). Heksadecimalan broj mora započeti s „0x” ili „0X” i nastaviti se nizom heksadecimalnih znamenki (tj. 0-9 i A-F).



Brojevi s pomičnim zarezom prikazuju se na dva načina: klasični način s točkom ili tzv. „inženjerski” način sa slovom E.



Kad rezultat operacije na brojevima postane prevelik za prikaz u *JavaScript*u, broj se prikazuje kao posebna vrijednost Infinity. Slično tome, kad je negativan broj manji od najmanjeg broja koji se može prikazati u *JavaScript*u, prikazuje se kao posebna vrijednost -Infinity.

Ako se prilikom neke matematičke operacije dobije nedefinirani rezultat (npr. dijeljenje s nulom), broj se prikazuje pomoću posebne vrijednosti NaN (*Not-a-Number* = nije broj). Ta vrijednost nije jednaka ni jednoj drugoj vrijednosti, pa čak ni samoj sebi. Zbog toga postoji posebna funkcija isNaN(), koja provjerava je li neka vrijednost broj ili ne. Funkcija isFinite() provjerava je li vrijednost broj i može li se prikazati kao konačan broj u *JavaScript*u.

U tablici se navode još neke posebne vrijednosti:

|  |  |
| --- | --- |
| **Konstanta** | **Značenje** |
| Infinity | Posebna vrijednost koja predstavlja beskonačnost. |
| NaN | Posebna vrijednost *nije-broj*. |
| Number.MAX\_VALUE | Najveći broj koji se može prikazati. |
| Number.MIN\_VALUE | Najmanji (najbliži nuli) broj koji se može prikazati. |
| Number.NaN | Posebna vrijednost *nije-broj*. |
| Number.POSITIVE\_INFINITY | Posebna vrijednost koja predstavlja beskonačnost. |
| Number.NEGATIVE\_INFINITY | Posebna vrijednost koja predstavlja negativnu beskonačnost. |

Vrijednosti Infinity i NaN definirane su u standardu *ECMAScript v1* tako da nisu implementirane prije *JavaScripta 1.3*.

3.1.2. Nizovi znakova

U *JavaScript*u se nizom znakova smatra bilo koji niz znakova *Unicode*, duljine 1 ili više, omeđen dvostrukim ili jednostrukim navodnicima (" ili '). Niz znakova započet s dvostrukim navodnikom mora završiti dvostrukim navodnikom. Također, niz znakova započet jednostrukim navodnikom mora završiti jednostrukim navodnikom. Niz znakova započet dvostrukim navodnikom može sadržavati proizvoljan broj jednostrukih navodnika i obratno.

Niz znakova treba biti napisan u jednom retku. Ako postoji potreba za unosom novog retka, potrebno je rabiti znak \n.



*JavaScript* se često kombinira s HTML-om, koji također rabi dvostruke ili jednostruke navodnike u vrijednostima argumenata. Stoga je poželjno imati stalan stil pisanja navodnika, na primjer:



Novi redak nije jedini slučaj kad treba napisati poseban znak. U *JavaScriptu* postoji nekoliko posebnih nizova znakova posebnog značenja:

**Napomena**

Neki znakovi ne rade u svim preglednicima.

|  |  |
| --- | --- |
| **Niz** | **Značenje** |
| \0 | NUL znak (\u0000). |
| \b | *Backspace* (\u0008). |
| \t | Vodoravni tabulator (\u0009). |
| \n | Novi red (\u000A). |
| \v | Okomiti tabulator (\u000B). |
| \f | *Form feed* (\u000C). |
| \r | *Carriage return* (\u000D). |
| \" | Dvostruki navodnici, čak i unutar niza znakova započetih dvostrukim navodnicima (\u0022). |
| \' | Jednostruki navodnici, čak i unutar niza znakova započetih jednostrukim navodnicima (\u0027). |
| \\ | Obrnuta kosa crta (\u005C). |
| \xXX | Znak *Latin-1* naveden s dvije heksadecimalne znamenke XX. |
| \uXXXX | Znak *Unicode* naveden s četiri heksadecimalne znamenke XXXX. |

Nizovi znakova spajaju se operatorom +:



3.1.3. Logičke vrijednosti

Logičke vrijednosti (tzv. Booleove vrijednosti) imaju samo dvije moguće vrijednosti koje se prikazuju pomoću konstanti true i false. Logičke vrijednosti obično su rezultat uspoređivanja:



3.1.4. Polja

Polje je skup vrijednosti u kojem svaki element ima jedinstveni redni broj koji se naziva indeks. Vrijednost određenog elementa dobije se tako da se iza naziva polja vrijednost indeksa stavi u uglate zagrade, tj. izraz a[2] označava treći element u polju a, jer su **u *JavaScript*u elementi polja indeksirani od nule**.

Polja mogu sadržavati bilo koju vrstu podataka *JavaScript*a, uključujući i reference na druga polja, objekte ili funkcije. Budući da se u *JavaScript*u ne navodi vrsta podatka pojedinih vrijednosti, nije nužno da su u jednom polju sve vrijednosti iste vrste (kao što je to u većini drugih jezika).

Polja se stvaraju pomoću objekta Array. Kad je polje stvoreno, može mu se dodati proizvoljan broj elemenata:



Polja se mogu stvoriti i tako da im se odmah postave početne vrijednosti. Prethodni primjer mogao je izgledati ovako:



Ako pri stvaranju objekta Array proslijedite samo jedan parametar, on označava duljinu polja. Sljedeći primjer stvara novo polje od 10 elemenata:



3.1.5. Nepostojeća i nedefinirana vrijednost

Ključna riječ null označava posebnu vrijednost koja nema nikakvu vrijednost. Kad varijabla ima vrijednost null, tada ne sadrži ni jednu ispravnu vrstu podatka: objekt, polje, broj, niz znakova ili logičku vrijednost.

Druga je posebna vrijednost undefined, koja se dobije kad se uporabi varijabla koja je bila deklarirana, ali nema početnu vrijednost ili svojstvo objekta koje ne postoji.

Vrijednost undefined nije isto što i null, iako će test jednakosti vratiti istinu. Kao što će se pokazati u nastavku, za razlikovanje se koriste operatori === ili typeof.

3.2. Varijable

Varijabla je ime povezano s vrijednošću, tj. varijabla sadržava određenu vrijednost. Varijable omogućavaju pohranu i rukovanje podacima u programu.

Kao što je prije rečeno, prvi znak naziva varijable mora biti slovo, podcrta ( \_) ili dolarski znak ($). Znakovi koji slijede mogu biti slova, brojevi, podcrta ili dolarski znak. Ako se *JavaScript-* i HTML-datoteke spremaju u kôdnoj stranici UTF-8, tada se mogu koristiti i znakovi s dijakriticima (č, ž, š, ć i đ). Međutim, preporuka je rabiti samo znakove *ASCII*, odnosno 26 slova engleske abacede.

*JavaScript* je jezik u kojem se prilikom deklariranja varijable ne navodi vrsta podatka. Štoviše, jedna varijabla može sadržavati vrijednost bilo koje vrste, npr:



Varijabla dobije vrstu podatka na osnovi podatka koji sadrži. Štoviše, *JavaScript*, ako je to potrebno, automatski mijenja vrstu varijable. Prije nego se rabi varijabla, poželjno ju je deklarirati pomoću ključne riječi var, na primjer:



Ako pri deklaraciji varijable nije navedena početna vrijednost, ona se automatski postavlja na vrijednost undefined.

ECMAScript 2015 donio je dvije nove *JavaScript* ključne riječi, let i const. Te ključne riječi stvaraju varijable koje mogu imati doseg unutar nekog bloka (Block Scope varijable). Za razliku od globalnih i lokalnih varijabli deklariranih sa ključnom riječi var, blok varijable deklarirane unuatar nekog bloka (npr. for(){} funkcije) sa ključnom riječi let ne utječu na varijable deklarirane izvan bloka.

var x = 10;

// x = 10

{

var x = 2;

// x = 2

}

// x = 2

var x = 10;

// x = 10

{

let x = 2;

// x = 2

}

// x = 10

Varijable deklarirane sa ključnom riječi let izvan bloka/funkcije (globalan doseg) ili unutar funkcije (lokalan doseg) ponašaju se jednako kao i varijable deklarirane sa ključnom riječi var.

Deklarianje iste varijable sa ključnim riječima let i var unutar istog dosega ili bloka nije dopušteno.

var x = 10;

let x = 5;

// greška

Varijable deklarirane sa ključnom riječi const ponašaju se isto kao varijable deklarirane sa ključnom riječi let, međutim ne može im se dodijeliti nova vrijednost. const varijablama mora se dodjeliti vrijednost u trenutku kada su deklarirane. Klučna riječ const ne označava konstantu/stalnu vrijednost, već definira stalnu referencu na zadanu vrijednost. Zato se vrijednost variable ne može promijeniti, međutim može se promijeniti svojstvo objekta definiranih sa ključnom riječi const.

const auto = {type: “Fiat“, model: “500“,

color:“blue“};

// može se mijenjati svojstvo

auto.color = “red“;

// može se dodati svojstvo

Auto.owner = “Marko“;

Varijabli deklariranoj sa ključnom riječi const ne može se dodijeliti novi objekt.

const auto = {type:“Fiat“, model:“500“,

color:“blue“};

//greška

auto = {type:“Volvo“, model:“EX60“, color:“red“};

Poljima deklariranim sa ključnom riječi *const* također se mogu mijenjati i dodavati elementi, međutim jednako kao i kod objekata, ne može se dodijeljivati novo polje.

const auti = ["BMW", "Audi", "Ford"];

// može se mijenjati element

auti[0] = "Toyota";

// može se dodati element

auti.push("Volvo");

// greška

auti = ["Mercedes", "Peugeot", "Citroen"];

3.3. Objekti

Objekt je složena vrsta podatka koji skuplja više vrijednosti u jednu cjelinu. Također se može reći da je objekt skup imenovanih vrijednosti. Te se vrijednosti nazivaju svojstva objekta. Želi li se pristupiti određenom svojstvu, navede se ime objekta, točka i na kraju ime svojstva.

Ako objekt osoba ima svojstva ime i zanimanje, tada se tim svojstvima pristupa ovako:



Objekti se stvaraju pozivom posebne funkcije (konstruktor). Npr. ovim se recima stvaraju objekti:



Postupak ili metoda je funkcija *JavaScript*a koja se poziva kroz objekt:



U gornjem primjeru ključna riječ this služi za pristup svojstvima objekta unutar njegove metode.

**O  
b  
j  
e  
k  
t**

**Svojstva**

**Metode**

**A  
r  
r  
a  
y**

**length  
...**

**sort  
join  
reverse  
...**

U praksi je najbolje stvarati objekte, polja, nizove, regularne izraze, itd. pomoću njihovih znakova, tj. skraćenica:

- { } umjesto new Object()

- " " umjesto new String()

- [ ] umjesto new Array()

- /()/ umjesto new RegExp()

3.4. **Vježba: Varijable i objekti**

1. Izradite HTML-datoteku naziva varijable.html ovog sadržaja:



1. *JavaScript* program pišite unutar elementa <script>.
2. Deklarirajte varijablu iBroj i dodijelite joj vrijednost 1.
3. Deklarirajte varijablu sNiz1 i dodijelite joj vrijednost 'Niz znakova'.
4. Deklarirajte varijablu sNiz2 i dodijelite joj vrijednost '3.14' .
5. Deklarirajte varijablu sNiz3 i dodijelite joj vrijednost 'U dva<br />retka'.
6. Deklarirajte varijablu bLogicka i dodijelite joj vrijednost true.
7. Vrijednosti ispišite pomoću funkcije document.write.
8. Promijenite vrijednost varijable bLogicka u false te ju ponovo ispišite.

1. Operatori

*Po završetku ovog poglavlja polaznik će moći:*

opisati i koristiti aritmetičke operatore

opisati i koristiti operator pridruživanja

opisati i koristiti operatore uspoređivanja

opisati i koristiti logičke operatore

opisati i koristiti operator spajanja.

Operatori se rabe pri obavljanju određenih operacija nad varijablama i konstantama. Pregled svih operatora i njihovih svojstava nalazi se u dodatku ovog priručnika.

4.1. Aritmetički operatori

Aritmetički operatori su operatori osnovnih računskih matematičkih operacija.

|  |  |
| --- | --- |
| **Operator** | **Značenje** |
| zbrajanje (+) | zbraja dva broja ili spaja dva niza znakova |
| oduzimanje (-) | oduzima drugi broj od prvoga ili kao unaran operator vraća negativnu vrijednost broja |
| množenje (\*) | množi dva broja |
| dijeljenje (/) | dijeli prvi broj drugim i rezultat je uvijek decimalan broj; dijeljenje s nulom daje pozitivnu ili negativnu beskonačnost (ovisi o prvom broju) dok 0/0 daje NaN |
| modulo (%) | vraća ostatak od dijeljenja prvog broja drugim |
| inkrement (++) | povećava vrijednost operanda za jedan |
| dekrement (--) | umanjuje vrijednost operanda za jedan |



4.2. Operator pridruživanja

Operator pridruživanja rabi se za pridruživanje vrijednosti varijabli.

Budući da je operator pridruživanja asocijativan (i to s desna), moguće je napisati:

Sve tri varijable imat će vrijednost 0.

Operator pridruživanja ima i poseban oblik: pridruživanje s operacijom, tj. oblik:



Operator pridruživanja s operacijom ne može se koristiti s operatorima inkrement (++) i dekrement (--).

4.3. Operatori uspoređivanja

Operatori uspoređivanja najčešće se koriste pri grananju (if) i uvjetnim petljama (while i do..while).

|  |  |
| --- | --- |
| **Operator** | **Značenje** |
| manji od (<) | rezultat je true ako je prvi operand manji od drugoga; inače false |
| veći od (>) | rezultat je true ako je prvi operand veći od drugoga; inače false |
| manji ili jednak (<=) | rezultat je true ako je prvi operand manji ili jednak drugom; inače false |
| veći ili jednak (>=) | rezultat je true ako je prvi operand veći ili jednak drugom; inače false |
| jednak (==) | rezultat je true ako je prvi operand jednak drugom; inače false |
| identičan (===) | rezultat je true ako je prvi operand jednak drugom i jednake su vrste podatka; inače false |
| različit (!=) | rezultat je true ako je prvi operand različit od drugoga; inače false |
| ne-identičan (!==) | rezultat je true ako je prvi operand različit od drugoga ili su različite vrste podatka; inače false |

Vrijednosti koje se uspoređuju mogu biti bilo koje vrste. No budući da se mogu uspoređivati samo brojevi i nizovi znakova, *JavaScript* prije uspoređivanja obavlja određena pretvaranja:

* Ako su oba operanda brojevi ili se mogu pretvoriti u brojeve, uspoređuju se kao brojevi.
* Ako su oba operanda nizovi znakova ili se mogu pretvoriti u niz znakova, uspoređuju se kao nizovi znakova.
* Ako je jedan operand niz znakova ili se može pretvoriti u niz znakova te ako je drugi operand broj ili se može pretvoriti u broj, niz znakova se pokušava pretvoriti u broj i uspoređuju se kao brojevi. Ako niz znakova nije broj, pretvara se u NaN, tj. uspoređivanje nije uspjelo.
* Ako se objekt može pretvoriti u broj ili u niz znakova, *JavaScript* pretvara objekt u broj. To znači da se objekti *Date* uspoređuju kao brojevi.
* Ako se jedan od operanada koji se uspoređuju ne može uspješno pretvoriti u broj ili u niz znakova, uspoređivanje je uvijek neuspješno, tj. dobiva se *false*.
* Ako je jedan od operanda NaN, uspoređivanje je uvijek neuspješno.

Važno je primijetiti da se nizovi znakova uspoređuju znak po znak, rabeći kôd svakog znaka iz tablice *Unicode*. To znači da uspoređivanje nizova znakova može biti neobično. Naime, „Zagreb” je manji od „auto”.

Nizovi znakova uspoređuju se znak po znak, ukoliko je prvi znak iz jednog niza jednak znaku iz drugog niza prelazi se na slijedeći znak, sve dok znak iz jednog niza nije različit od znaka drugog niza, ili dok se ne dođe do kraja nizova (ukoliko su jednaki). U unicode tablici slovo a ima veću vrijednost kôda od znaka Z, zato je niz „Zagreb“ manji od „auto“.

Uspoređivanje također razlikuje velika i mala slova (jer su im kôdovi različiti).

Gore navedena pravila za pretvaranje tipova prilikom uspoređivanja ne primijenjuju se za operator identičnosti (===), koji zahtijeva da dvije varijable budu jednake i po vrijednosti i po vrsti podatka. U sljedećem primjeru dobiva se različit rezultat prilikom uspoređivanja s pojedinim operatorom:



4.4. Logički operatori

Logički operatori se najčešće koriste kod složenih uvjeta za grananja i petlje.

|  |  |
| --- | --- |
| **Operator** | **Značenje** |
| logičko i (&&): | rezultat je true ako i samo ako su oba operanda true; inače daje false |
| logičko ili (||): | rezultat je true ako je jedan od operanada true; inače daje false |
| logičko ne (!): | unarni operator kod kojeg je rezultat true samo ako je operand false; inače daje false |



4.5. Operator spajanja

Operator + spaja dva niza znakova u jedan novi. Npr:



Operator + daje viši prioritet nizovima znakova nego brojevima. Dakle, ako je jedan operand niz znakova, onda se i drugi operand pretvori u niz znakova i obavi se spajanje. Kod operatora uspoređivanja je obratno. Naime, ako je jedan operand broj, drugi se pretvara u broj i obavlja se uspoređivanje. Stoga je važno pripaziti:



4.6. Vježba: Operatori

1. Izradite HTML-datoteku, naziva operatori.html, istog sadržaja kao u vježbi 3.
2. Deklarirajte dvije varijable iA i iB i dodijelite im proizvoljne cjelobrojne vrijednosti.
3. Deklarirajte varijable iSuma, iRazlika i iModulo (koje će sadržavati redom: sumu, razliku, modulo varijabli iA i iB).
4. Postavite inicijalne vrijednosti varijabli iSuma, iRazlika i iModulo na nulu koristeći se operatorom pridruživanja.
5. Deklarirajte varijablu bLogicka.
6. Prikažite inicijalne vrijednosti varijabli iA i iB pomoću funkcije document.write.
7. Izračunajte tri aritmetičke operacije iz druge točke, dodijelite ih odgovarajućim varijablama i prikažite ih.
8. Inkrementirajte varijablu iA i dekrementirajte varijablu iB te prikažite nove vrijednosti varijabli iA i iB.
9. Već izračunatoj sumi dodajte novu vrijednost varijable iA koristeći se pridruživanjem s operacijom. Prikažite novu vrijednost varijable iSuma.
10. Varijabli bLogicka dodijelite rezultat usporedbe je li vrijednost varijable iA veća od 5. Prikažite vrijednost varijable bLogicka.
11. Prikažite iA + iB i uočite da se ne dobije zbroj već niz znakova.
12. Ispišite (iA + iB) i uočite da je zbroj točno ispisan.
13. Funkcije

*Po završetku ovog poglavlja polaznik će moći:*

definirati funkciju

izraditi i pozvati funkciju unutar JavaScripta

razlikovati lokalne i globalne varijable.

5.1. Definiranje funkcije

Funkcija je konstrukcija u *JavaScript*u pomoću koje grupiramo veći broj naredbi koje izvršavamo navodeći ime funkcije. Tako skraćujemo pisanje programa (ako više puta pozivamo istu funkciju) te pojednostavljujemo program (npr. cijeli program može se sastojati od poziva funkcija proizvoljnog imena te tako pokazuje logiku programa, što bi inače bilo skriveno u velikom broju naredbi). Najčešći je način definiranja funkcije pomoću ključne riječi function:

function <naziv\_funkcije>(<argument 1>, <argument 2>, ...){

// naredbe

}

Popis argumenata nije obavezan, ali okrugle zagrade jesu. Tijelo funkcije piše se u vitičastim zagradama {...} koje označavaju blok naredbi, tj. to je način da se nekoliko naredbi poveže u jednu cjelinu.

Funkcija može vratiti vrijednost naredbom return, ali i ne mora. Evo nekoliko primjera:



5.2. Poziv funkcije

Funkcija se poziva tako da se navede njezin naziv, a argumenti funkcije u okruglim zagradama. Ako funkcija nema argumenata, ne navodi se ništa, ali zagrade su obavezne. Ako se funkciji proslijedi manje argumenata nego ih sadrži definicija funkcije, drugi argumenti dobiju vrijednost undefined. Na primjer, gore definirane funkcije pozivaju se ovako:



5.3. Doseg varijabli

U *JavaScript*u vrijednost je varijable dostupna na dva načina – samo unutar određene funkcije ili u cijelom programu. Varijable koje su dostupne u cijelom programu nazivamo globalne varijable, a varijable koje su dostupne samo unutar funkcije nazivamo lokalne varijable. Prilikom uporabe varijabli poželjno je uvijek ih deklarirati pomoću ključne riječi var:

* + Ako je varijabla deklarirana pomoću ključne riječi var ili joj je samo dodijeljena vrijednost u glavnom programu (izvan svih funkcija), varijabla je globalna.
  + Ako je varijabla deklarirana unutar određene funkcije pomoću ključne riječi var, tada je varijabla lokalna.
  + Ako varijabla nije deklarirana, nego joj je samo dodijeljena neka vrijednost unutar određene funkcije, tada je varijabla globalna.

Treba biti oprezan kod deklariranja globalnih varijabli kako one ne bi promjenile vrijednosti lokalnih varijabli (ukoliko imaju isti naziv kao lokalne varijable). Globalne varijable trebalo bi koristiti što manje i samo kada su nužno potrebne.



5.4. Vježba: Funkcije

1. Izradite HTML-datoteku naziva funkcije.html istog sadržaja kao u vježbi 3.
2. Iskoristite definicije funkcija ispis i fUdaljenost iz točke 5.1.
3. Deklarirajte varijable sMjesto, iMjerilo i iUkupno.
4. Varijabli sMjesto dodijelite proizvoljan naziv grada.
5. Varijabli iMjerilo dodijelite vrijednost 25.
6. Izračunajte udaljenost mjesta od mora, tj. duljinu puta od točke (0,0) do točke (3,5) kroz točku (2,1) na karti mjerila pohranjenoj u varijabli iMjerilo.
7. Ispišite dobrodošlicu u grad.
8. Ispišite izračunatu udaljenost do mora (u kilometrima).
9. Naredbe za kontrolu tijeka

*Po završetku ovog poglavlja polaznik će moći:*

nabrojati i opisati naredbe za grananje unutar JavaScripta

nabrojati i opisati naredbe za izradu petlji unutar JavaScripta.

6.1. Uvjetno izvođenje naredbi

Osnovna naredba za grananje je naredba if. Njezin je najjednostavniji oblik:



gdje je uvjet logički izraz čiji je rezultat istina (true) ili neistina (false). Kad operandi u izrazu nisu logički, prevode se u logičke vrijednosti. Ove se vrijednosti uvijek prevode u false:

* null
* undefined
* prazan niz znakova ('' ili "")
* broj 0
* NaN.

Sve druge vrijednosti prevode se u true.

U ovom primjeru vrijednost varijable ime je prazan niz, što će se prevesti kao false.



Vitičaste zagrade, koje označavaju blok naredbi, nisu potrebne ako iza uvjeta slijedi samo jedna naredba, ali se preporuča uvijek ih pisati radi jednoznačnosti.

Drugi oblik je oblik if..else:



Taj oblik ima još jedan blok, koji se izvodi ako uvjet nije zadovoljen.

6.2. Višestruka usporedba

Ako jednu vrijednost treba usporediti više puta, rabi se nekoliko naredbi if..else:



Međutim, takav način pisanja višestruke usporedbe prilično je nepregledan. Stoga se rabi naredba switch:



Ta naredba provjerava je li izraz u okruglim zagradama (odmah iza ključne riječi switch) istovjetan (koristi se operator ===), dakle, i po vrsti jednak, jednoj od vrijednosti iza ključne riječi case. Ako takva vrijednost postoji, izvršava se blok naredbi iza te vrijednosti do kraja cijele naredbe switch ili do ključne riječi break, koja prekida blok koji se trenutačno izvršava, tj. završava naredbu switch. Dakle, moguće je tu naredbu napisati tako da se jedan blok izvrši za više vrijednosti:



**Treba pripaziti da se na kraju svakog bloka naredbi napiše naredba break.** Ako se ona nehotice izostavi, neće biti prijavljena pogreška, nego će se izvršiti i sljedeći blok naredbi.

Ako ne postoji vrijednost koja je istovjetna provjeravanom izrazu, izvršava se blok iza ključne riječi default. Iako je uobičajeno da se blok default piše posljednji (jer onda ne treba pisati naredbu break), to nije obavezno, ali ga je preporučljivo napisati, čak i kada za njega možda nema potrebe. Taj je blok ravnopravan drugim blokovima.

6.3. Uvjetni operator

Uvjetni operator jedini je operator s tri operanda u *JavaScript*u:



što je jednako ovom:



Taj operator najčešće se koristi pri inicijalizaciji varijabli:



6.4. Petlja s uvjetom na početku

Osnovna je petlja u *JavaScriptu* petlja while:



Tijelo petlje izvršava se sve dok je uvjet zadovoljen. Stoga bi se naredbom while(true){...} napravila beskonačna petlja. Ako uvjet nije zadovoljen prije početka petlje, njezino tijelo se neće izvesti niti jednom.

6.5. Petlja s uvjetom na kraju

Često je potrebno izvršiti neke postupke barem jednom i tek tada provjeriti određeni uvjet. U takvom slučaju koristi se do..while konstrukcija:



Konstrukcija do..while mora se završiti kao naredba točka-zarezom (zato jer završava uvjetom, a ne blokom kao osnovna petlja while).

6.6. Petlja s poznatim brojem ponavljanja

Kad je točno poznat broj ponavljanja nekog postupka ili su poznati početni i krajnji uvjet, koristi se petlja for:



Petlja for istovjetna je ovoj petlji while:

  
Primjer uporabe:



6.7. Vježba: Naredbe za kontrolu tijeka

1. Izradite HTML-datoteku naziva petlje.html istog sadržaja kao u vježbi 3.
2. Definirajte funkciju ispis kao u vježbi 5.
3. Deklarirajte varijable iA, iB, sOperacija i iRazlika.
4. Varijablama iA i iB dodijelite proizvoljne vrijednosti.
5. Varijabli sOperacija dodijelite niz znakova +.
6. Rabeći jednostavno grananje provjerite je li varijabla iA veća ili manja od iB.
7. Rabeći višestruko grananje ispišite koja je operacija zapisana u varijabli sOperacija.
8. Izračunajte i ispišite razliku brojeva iA i iB, ali tako da oduzmete veći broj od manjeg. Uporabite uvjetni operator.
9. Uporabom petlje for ispišite 10 redaka teksta: "*For: redak x*", gdje je x broj retka.
10. Napišite prethodnu petlju for pomoću petlje while.
11. Obrasci

*Po završetku ovog poglavlja polaznik će moći:*

primijeniti JavaScript kôd za dohvaćanje elemenata iz obrasca

izraditi HTML-formu

primijeniti funkcije na izrađenu HTML-formu.

7.1. Prvi obrazac

HTML-obrazac dio je dokumenta koji sadrži tekst i posebne elemente zvane kontrole (potvrdni okvir (*checkbox*), izborno dugme (*radio button*), izbornik itd.) Korisnici obično „popune” obrazac mijenjajući kontrole (unose tekst, označavaju mogućnosti, odabiru od ponuđenog itd.). Na kraju „popunjavanja” korisnik šalje podatke na obradu. Obrasci su najčešći način prijave ili ispunjavanja podataka na HTML-stranicama. Sastoje se od niza elemenata od kojih svaki ima svoje posebnosti. Ovo je najjednostavniji obrazac koja ima samo jedno dugme *Poruka*:



Budući da je pisanje *JavaScript*a u odvojenoj datoteci praktičnije, preporuča se zasebno pisanje HTML-a i *JavaScript*a:



7.2. Unos kraćih nizova znakova

Kroz ovo poglavlje proći će se kroz primjere pristupa vrijednostima polja za unos u koje korisnik unosi svoje podatke. Kraći nizovi znakova (nizovi koji imaju samo jedan red) unose se u tzv. tekstno polje. Primjer se nadogradi jednim poljem za unos imena:





Rabi se polje forms (indeks je 0, jer je taj obrazac prvi u polju forms), što je prilično nečitko, a može biti i problematično. Da bi se izbjegao takav način pisanja, svakom elementu obrasca dodijeli se atribut id. HTML preporuka definira id atribut kao atribut koji jedinstveno definira element na razini dokumenta, odnosno u jednoj HTML-datoteci ne smiju biti dva elementa s istom vrijednosti id atributa.



Element se tada dohvaća pomoću funkcije getElementById.



Prvi je korak u provjeri podataka koje upisuje korisnik: je li korisnik uopće upisao potrebni podatak:



U sljedećm koraku elementu form se dodaju atributi action i method:



Unutar atributa action postavlja se putanja do poslužiteljske skripte koja obrađuje podatke koje korisnik unese unutar forme i pošalje na server.

Da bi se mogla iskoristiti poslužiteljska skripta, potrebno je promijeniti vrstu dugmeta iz button u submit.

Svaki element u HTML-u (a to znači i u HTML-obrascu) ima svoje podrazumijevano ponašanje. Na primjer, element a je hiperlink na drugi dokument i njegovo je ponašanje da se klikom mišem na taj element prikaže dokument. Klikom mišem na dugme za slanje (element input tipa submit*)* podaci iz obrasca šalju se na URL naveden u atributu action.

*JavaScript*om se ta podrazumijevana ponašanja mogu promijeniti. Na primjer, ako je *Submit* dugme povezano s događajem click, dugme će se ponašati podrazumijevano samo ako funkcija u tom događaju vrati true. Ako funkcija vrati false, podrazumijevano ponašanje se prekida.

Stoga će funkcija provjeri vratiti true ako je provjera prošla, odnosno false ako nije.



Za ovu vježbu napravit će se obrazac u koji će korisnici upisivati svoje ime i prezime, odabrati kakvu vrstu računala koriste, imaju li pristup Internetu putem DSL tehnologije, te koji operacijski sustav i verziju Service Pack paketa koriste. Postojeći primjer će se nadograditi sa svim potrebnim elementima, te će se dodati sav potreban *JavaScript* kôd kako bi sve pravilno funkcioniralo.

7.3. Izrada HTML-forme

U HTML-dokumentu treba se nalaziti cjelokupna forma koju će korisnici ispunjavati, međutim prvo treba pripremiti HTML*-*dokument:



Unutar html elementa prvo treba dodati head element u kojem se nalaze meta, title i link (koji povezuje CSS-skriptu) elementi. Nakon zatvaranja head elementa dodaje se body element u kojem će se nalaziti cjelokupna forma.



Unutar form elementa dodaju se atributi action i method, kakvi su već opisani u prethodnom poglavlju, koji služe za predaju forme na server kako bi se podaci obradili. Nakon zatvaranja form elementa i prije zatvaranja body elementa dodaje se poziv na *JavaScript* datoteku u kojoj se nalazi sav *JavaScript* kôd zaslužan za funkcioniranje forme.

Unutar forme prvo se dodaju elementi za unos imena i prezimena korisnika pomoću input elementa. Svakom elementu dodaje se atribut id kako bi se vrijednosti polja mogle lako dohvatiti unutar *JavaScript* kôda.

Nakon polja za unos dodaju se radio gumbi pomoću kojih korisnik odabire vrstu računala koju posjeduje.



Kako bi saznali koristi li korisnik DSL tehnologiju za pristup Internetu dodaje se checkbox, koji korisnik može, ali i ne mora označiti. U svakom slučaju forma će se na kraju moći predati.



Na kraju svega treba dodati mogućnost odabira operacijskog sustava i Service pack paketa kojeg korisnik ima na svojem računalu. Kako korisnik ne bi mogao odabrati nepostojeće kombinacije tih dviju komponenata pobrinut će se *JavaScript* funkcija kroz koju će se kasnije proći.

Odabir OS-a:



Odabir Service Pack paketa:





Na kraju forme dodaje se input polje tipa submit koje služi za predaju ispunjenog obrasca.



Sada kada je cijela forma pripremljena, sve što je potrebno je napisati *JavaScript* kôd koji će dodati neka pravila oko ispune same forme. Ukoliko korisnik ne ispuni neko obvezatno polje forma se neće moći predati. Isto tako potrebno je napisati funkciju koja će korisniku prikazati pravilne mogućnosti odabira Service pack paketa, ovisno o odabranom Operacijskom sustavu. Kako se iz kôda može vidjeti, nigdje u formi se ne nalazi onclick atribut koji bi pozvao neku *JavaScript* funckiju. To je napravljeno iz razloga što će na sve potrebne HTML*-*elemente unutar *JavaScript* kôda biti postavljeni event listeneri (pomoću addEventListener naredbe) koji će pokretati određene funkcije. Kako bi kôd ispravno, tj. uopće radio *JavaScript* poziv se u ovom slučaju mora nalaziti na kraju body. Ukoliko se poziv postavi prije stvaranja potrebnih elemenata na stranici, *JavaScript* neće moći na njih postaviti addEventListener metodu.

7.4. Izrada funkcija u *JavaScriptu*

7.4.1. Funkcija puniSelect

U slijedećemn poglavlju prolazi se kroz kôd koji se piše u zasebnu *JavaScript* datoteku (npr. <script type="text/javascript" src="javas.js"></script>) ili unutar script elementa. Taj kôd služi kako bi forma koja je stvorena kroz prijašnje poglavlje pravilno funkcionirala.

Na samom početku dohvatit će se select element sa listom Operacijskih sustava (operacijski\_sustav) na kojega se postavlja event listener koji se aktivira svaki puta kada korisnik promjeni vrijednost navedenog select elementa, tj. kada korisnik promjeni vrijednost operacijskog sustava. Taj event listener pokrenuti će funckiju nazvanu puniSelect, u koju će poslati trenutnu vrijednost select polja, kako bi se u polju sa listom Service Packova prikazivale pravilne vrijednosti. Sam element se pohranjuje u varijablu naziva operacijski\_sustav, a dohvaća se pomoću jedinstvenog id atributa.



Osim toga pri svakom učitanju stranice funckija puniSelect će se iznova pozvati pomoću window.onload naredbe. Na taj način će se prilikom svakog učitanja stranice vrijednost operacijskog sustava postaviti na *Windows 7* kako je definirano unutar HTML-datoteke, dok će vrijednost *Service Pack* paketa biti postavljena na *Service Pack 1*.

Funkcija puniSelect:



Sama funkcija puniSelect vrlo je jednostavna, a funkcionira na način da ovisno o poslanom atributu element dohvaća sve elemente unutar select polja koji nemaju atribut class jednak poslanom atributu, te ih sprema u varijablu list.

Svim tim elementima će CSS-atribut display biti postavljen na none. Na taj način korisnici neće moći odabrati, niti vidjeti Service Pack verzije koje ne pripadaju uz određeni operacijski sustav. Ista stvar mogla se postići postavljanjem CSS-atributa display na none za sve elemente select polja.

U varijablu display pohranjujemo listu svih elemenata koji imaju atribut class jednak poslanom atributu element. Time se dobiva popis svih elemenata unutar polja sa popisom Service Packova koji odgovaraju odabranom Operacijskom sustavu. Naravno, svim elementima unutar popisa se CSS-atribut display postavlja na vrijednost block, kako bi bili vidljivi korisniku. Kroz svaki od elemenata unutar varijable display prolazi se pomoću for naredbe.

Na samom kraju funkcije select elementu sa id atributom srv\_pack postavlja se vrijednost na zadnji element display varijable (koja u sebi sadrži listu svih ispravnih Service Packova za odabrani OS).

7.4.2. Funkcija provjeri

Sada kada je dobivena funkcionalnost svih elemenata forme potrebno je napraviti funkciju koja će provjeravati koja polja nisu ispunjena kako bi se korisnika moglo upozoriti. Funkcija će biti podijeljena u dva dijela koji čine jednu cjelinu. Prvo će se proći i objasniti prvi dio funkcije, a zatim drugi dio.



Naredbom querySelector odabire se input polje tipa submit, tj. “gumb“ submit na kojeg se postavlja event listener. Kada korisnik klikne na gumb submit poziva se funkcija provjeri u koju se šalje atribut e (element na kojega je korisnik kliknuo). Taj atribut je potreban kako bi se moglo spriječiti slanje forme na server ukoliko korisnik nije ispunio sve elemente forme na ispravan način.



U samoj funkciji na početku se nalazi popis svih potrebnih varijabli koje sve imaju definirane istu početnu vrijednost, a ta vrijednost je prazan niz znakova (prazan string).



Nakon svakog nepravilno ispunjenog elementa forme korisniku će se ispisati poruka koja ga na to upozorava. Kako se te poruke ne bi duplicirale ukoliko korisnik više puta nepravilno popuni neko polje, te kako bi se uklonile kada korisnik neko polje ispravno popuni brine se slijedeća varijabla i for funkcija. U varijablu remove pohranjuje se popis svih stvorenih elemenata sa klasom missing-value, koji služe za ispis poruka korisniku. Pomoću naredbe for prolazi se kroz sve elemente unutar varijable remove koji su unutar nje pohranjeni u obliku polja (array), te ih se briše upotrebom metode remove().



U ostatku kôda dohvaćaju se vrijednosti elemenata unutar obrasca kako bi se provjerilo je li forma ispravno ispunjena, te koje vrijednosti je korisnik unio u nju kako bi se mogla ispisati odgovarajuća poruka na ekran.



U varijable ime i prezime spremaju se vrijednosti polja u koje je korisnik upisao svoje ime i prezime. Obje se dohvaćaju pomoću jedinstvenog id atributa. Ako korisnik nije ispunio neko od polja, nakon neispunjenog polja stvara se novi paragraf sa klasom missing-value koje korisnika upozorava da polje nije ispunjeno. Paragraf se stvara pomoću metode *insertAdjacentHTML* koja ga stvara iza *div* elementa sa odgovarajućom klasom. Osim toga unutar varijable error dodaje se poruka koja će korisnika dodatno upozoriti da određeno polje nije ispunjeno putem window.alert metode.

Ako su polja ispravno ispunjena unutar varijable message dodati će se tekst koji ispisuje vrijednosti koje je korisnik upisao u ta polja.

Ovime je završen prvi dio funkcije, u drugom dijelu prolazi se kroz provjeru ostalih polja forme.



U varijablu racunalo pohranjuju se svi elementi kojima je atribut name jednak komp (radio gumbi). Tim elementima se zatim pomoću for i if funkcija pristupa i provjerava je li neki od njih označen od strane korisnika. Ako jest, kao i u prvom dijelu, varijabli message dodati će se odgovarajući tekst koji će ispisati odabranu vrijednost, a ako nije varijabli error dodati će se tekst upozorenja kako korisnik nije odabrao vrstu računala koju posjeduje, te će se ispisati paragraf upozorenja.



Pošto ne treba provjeravati je li korisnik označio checkbox element koji govori na koji način se korisnik spaja na Internet, može se direktno u varijablu message dodati odgovarajuća poruka. Ako je checkbox sa id atributom Internet označen, u varijablu će se upisati text koji se nalazi u roditeljskom (parent) elementu, tj. tekst koji korisnik vidi u formi pod navedenim checkbox elementom. U suprotnom ispisuje se poruka koja kaže kako korisnik pristupa Internetu putem modema. Metodom textContent dohvaća se text elementa, dok se trim() metodom miču nepotrebne bjeline sa početka i kraja teksta.



Varijable os i sp služe za pohranu vrijednosti koje je korisnik odabrao putem select elementa. Za njih ne treba provjeravati jesu li prazne ili ne pošto je vrijednost operacijskog sustava i Service Pack paketa uvijek odabrana. Zahvaljujući tome njihove vrijednosti se mogu direktno dodati u varijablu message bez dodatnih provjera i uvjeta.



**Napomena**

Kompletan kôd obrasca iz ove cjeline nalazi se u 10. cjelini “**Kompletan kôd obrasca iz cjeline 7.**“

Na kraju funkcije dolazi zadnja provjera. Ako korisnik nije nešto ispunio, tj. ako varijabla error nije prazna, ispisati će se poruka na ekranu koja će korisnika upozoriti koja sve polja forme nije ispunio, te će se spriječiti slanje forme na server na obradu pomoću preventDefault metode. Međutim, ako je korisnik popunio sva polja forme na ekranu će se ispisati sve što je dodano u varijablu message, tj. ispisati će se sve vrijednosti koje je korisnik unio i odabrao unutar forme, a forma će se predati na server kako bi se podaci obradili.



Time je gotova funkcija provjeri, primjer bi se još mogao nadopuniti sa mogućnošću provjere upisanih vrijednosti u polja za unos imena i prezimena kako korisnici ne bi mogli unutar njih upisivati brojeve i znakove (osim – i sl.).

7.5. Vježba: Obrasci

1. Izradite HTML-datoteku naziva forma.html i u njoj izradite formu u koju će korisnik unijeti svoje ime, odabrati koju vrstu računala posjeduje, te koji operativni sustav ima na svojem računalu.
2. Izradite datoteku *javascript.js* i u njoj napravite funkciju naziva provjeri koja će se pozvati svaki puta kada korisnik pritisne gumb za slanje forme.
3. Funkcija treba provjeravati polja forme koja je korisnik obavezan ispuniti, te ako korisnik neko polje nije ispunio treba se pojaviti poruka koja ga o tome obaviještava.
4. Kada korisnik u potpunosti ispuni formu, te ju preda, treba se na ekranu ispisati poruka koja mu prikaže sve podatke koje je unio.
5. *JavaScript* biblioteka – *jQuery*

*Po završetku ovog poglavlja polaznik će moći:*

napisati poziv jQuery biblioteke unutar HTML-kôda

razlikovati jQuery sintaksu od JavaScript sintakse

koristiti napredne funkcije jQuerya.

8.1. Općenito o *JavaScript* bibliotekama

*JavaScript* je dugo vremena bio samo dodatak HTML-u. *Google* je 2004. godine izradio *Gmail* u kojemu se koristila asinkrona komunikacija između preglednika i poslužitelja. Ista tehnologija upotrijebljena je godinu dana poslije za *Google Maps*. Ta je tehnologija izazvala malu revoluciju i pokrenula postupak koji je kasnije nazvan revolucija *Web 2.0*.

Navedena tehnologija dobila je naziv AJAX – *Asynchronous JavaScript and XML.*

Budući da je uporaba AJAX-a bila složena, počele su se pojavljivati gotove skripte koje su olakšavale programiranje. Takve su skripte bile jezgra za pojavu niza *JavaScript* biblioteka (*libraries*).

Među razlozima za pojavu *JavaScript* biblioteka bilo je i ujednačavanje načina rada s različitim preglednicima. Korištenjem biblioteke programer više ne mora brinuti o različitostima među pojedinim preglednicima.

Najpoznatije su:

* *jQuery* (http://jquery.com/)
* *Dojo* *Toolkit* (http://dojotoolkit.org/)
* *Prototype* (http://prototypejs.org/).

*jQuery* je danas gotovo standard, naročito jer je kompanija *Twitter* odlučila upotrijebiti upravo *jQuery* u svojoj besplatnoj distribuciji za standardizirani razvoj *web*-stranica pod nazivom *Twitter Bootstrap* (http://getbootstrap.com/).

8.2. *jQuery*

*jQuery* se sastoji od osnovne distribucije i od velikog broja dodataka (*plug-ins*).

Osnovna distribucija može se koristiti na dva načina:

* Klasično snimanje na lokalni disk *web*-sjedišta
* CDN (*Content Delivery Network*).

Uporaba preko CDN-a ima dvije prednosti:

* Za smanjenje količine podataka koja putuje od poslužitelja do preglednika koristi se tzv. minimizirana inačica datoteke *JavaScript*. To znači da su iz datoteke (programa) uklonjene sve nevažne praznine, a ponekad se još obavi i skraćivanje naziva varijabli tako da se redom dodjeljuju nazivi oblika a, b, c, …, z, odnosno nadalje aa, ab, ac, …, az, ba, bb, bc, …
* Uporabom CDN-a postoji velika vjerojatnost da je korisnik posjetio *web*-sjedište koje se koristi istom CDN-datotekom pa korisnikov preglednik ne dohvaća ponovo istu datoteku, nego se koristi postojećom u lokalnoj predmemoriji (*cache*) te se tako dodatno smanjuje količina podataka koja putuje od poslužitelja do preglednika.

Međutim, postoji i jedan nedostatak – minimizirani kôd teško je čitati pa ako postoji potreba za upoznavanjem s bibliotekom *jQuery,* preporuča se snimiti neminimiziranu inačicu.

*jQuery* se najčešće koristi tako da se za određene elemente dokumenta definiraju određene akcije.

U *jQuery*ju se elementi određuju CSS-selektorima, odnosno na isti način kao i u CSS-u. U HTML-dokumentu:



Vidi se da se *jQuery* rabi pomoću funkcije jQuery.

Međutim, postoji i kraći način uporabe (tim se načinom koristi većina korisnika): znak $ koji je alias za funkciju jQuery:

**Napomena**

Primjeri se nalaze u datotekama: ***jQuery/primjer2.html*** i ***jQuery/primjer3.html***.



U prepravljenom primjeru pokazano je kako se identificiraju elementi pomoću atributa id. Ako je potrebno identificirati elemente kojima je atribut class postavljen na određenu vrijednost, to se postiže ovako:



Da bi se dobio uvid u to kako izgleda rad s *jQuery*jem i njegovim dodacima, prepravit će se obrazac iz glavnog dijela tečaja.

8.3. Prerada obrazaca uz pomoć biblioteke *jQuery*

Za provjeru obrasca koristit će se dodatak *jQuery Validate*. Za ulančano povezivanje elemenata select korsitit će se dodatak *Chained*.

Prvo treba učitati *jQuery* i potrebne dodatke:



Tekstna polja, potvrdni okviri (*check button*), izborna dugmad (*radio button*) ne mijenjaju se. Elemente *select* treba prilagoditi (tj. upisati sve potrebne podatke) da bi dodatak *jQuery* mogao popunjavati drugi izbornik:



Drugi se izbornik s prvim povezuje preko atributa class:  


Budući da se *jQuery*-kôd ne bi izvršio prije nego što su učitani svi HTML‑elementi, program *JavaScript* piše se na kraju HTML-dokumenta (tj. učitava se iz datoteke):

. . .

<script type="text/javascript"

src="http://cdn.jsdelivr.net/jquery/1.11.2/jquery.min.js">  
 </script>

<script type="text/javascript"

src="http://cdn.jsdelivr.net/jquery.validation  
 /1.13.1/jquery.validate.js">

</script>

<script type="text/javascript"

src="http://cdn.jsdelivr.net/jquery.chained  
 /0.9.9/jquery.chained.js">

</script>

<script type="text/javascript" src="obrazac.js"></script>

</body>

Obrada je povezanih elemenata *select* jednostavna:

  
Provjera je li korisnik ispunio potrebna polja nešto je složenija, jer se trebaju navesti pravila, a da bi se zadržala funkcionalnost iz originalnog primjera, ispisuju se upisani podaci (ako su upisani svi potrebni podaci).

Općenita obrada podataka pomoću dodatka *jQuery Validate* ima oblik:

$("#obrazac1").validate({

rules: {

// pravila koja moraju zadovoljavati polja

},

messages: {

// poruke koje se ispisuju za pojedinu pogrešku

},

submitHandler: function (form) {

// popis naredbi koje se izvršavaju prije nego   
 // se pošalju podaci

form.submit();

}

});

Konkretno, za gornji su primjer pravila:



Poruke:



Budući da se dio obrasca koji u sebi ima izbornu dugmad (*radio button*) sastoji od većeg broja elemenata (odnosno od pojedine izborne dugmadi), nije dobro pogrešku ispisivati uz pojedino izborno dugme. Stoga je potrebno promijeniti mjesto gdje će se ispisivati pogreška (iza elementa label):



Konačno se ispišu podaci koje je korisnik upisao (prije nego se podaci pošalju na odgovarajući URL):



Kompletan *jQuery* kôd:



7. 4. Napredni primjeri (*jQuery*)

Primjer 1: Upis vrijednosti unutar polja za unos teksta kako bi se prikazali i filtrirali preporučeni pojmovi za odabir.

Korisnik unutar polja za unos teksta upisuje proizvoljno što želi, ako se tekst koji korisnik unosi podudara sa nekom vrijednosti iz liste pojmova taj pojam/ ti pojmovi će se filtrirati i prikazati korisniku koji ih onda može, ali ne mora odabrati klikom na njih.

Sam popis je sakriven od korisnika sve dok korisnik ne klikne na polje za unos. Nakon čega popis postaje vidljiv i korisnik ga može pretraživati ili filtrirati upisom željenih pojmova.

Cijela stvar funkcionira tako da se određeni pojmovi u listi prikazuju ili sakrivaju postavljanjem CSS *display atributa* na vrijednost block ili none.

HTML:



*JavaScript*:



Primjer 2: Upis vrijednosti unutar polja za unos teksta kako bi se prikazali i filtrirali pojmovi od kojih korisnik mora jedno odabrati.

Klikom na select polje korisniku se prikazuje polje za unos teksta pomoću kojega može filtrirati opcije koje mu se nude. Za razliku od prvog primjera, ovdje korisnik ne može upisati što god želi i nastaviti dalje sa ispunjavanjem forme, tj. ponuđeni pojmovi nisu opcionalni, već korisnik jednog od njih mora odabrati. Sam princip rada kôda ova dva primjera je veoma sličan, u oba slučaja elementi se prikazuju i sakrivaju mjenjanjem CSS *display atributa*.

HTML:



*JavaScript:*



Osim *jQuery*-em ovi primjeri također se mogu napraviti i upotrebom čistog *JavaScripta*.

1. Korisne skripte

*Po završetku ovog poglavlja polaznik će moći:*

izraditi Rollover efekt

objasniti i koristiti regularne izraze

nabrojati i primijeniti naredbe za upravljanje preglednikom.

9.1. *Rollover*

*Rollover* (okretanje) je u prošlosti jedan od najčešće korištenih vizualnih efekata kod kojih se rabio *JavaScript* (danas se češće rabi CSS). To je efekt kad se slika (obično je to slika s tekstom koja stoji u izborniku) promijeni u trenutku kad korisnik postavi pokazivač miša na tu sliku.

Primjer:



Napisana je funkcija koja postavlja argument src za određeni objekt (u našem slučaju je to this, tj. objekt na kojem se dogodio događaj).

**Napomena**

Događaji *mouseover* i *mouseout* objašnjeni su u **dodatku E** na kraju priručnika.



Uporabom iste funkcije može se upravljati i drugim objektima. Na primjer, napravi se mali album:



9.2. Preusmjeravanje

Ponekad postoji potreba da se na osnovi određenih kriterija (npr. jezik u kojemu su stranice pisane ili vrsta preglednika kojom se korisnik služi) korisnici upute na druge stranice. U tu svrhu rabi se preusmjeravanje pomoću objekta location i njegovog svojstva href:



9.3. Provjera pomoću regularnih izraza

Prilikom unosa podataka korisnici slučajno unose nepravilne podatke. Bilo bi poželjno da se može provjeriti što su korisnici unijeli te da ih se može upozoriti ako su podaci neispravni. *JavaScript* od inačice 1.2 ima u sebi podršku za regularne izraze (*regular expressions*), koji omogućuju pisanje određenih pravila (izraza) i provjeru pridržava li se određeni niz znakova tih pravila. Srž je regularnih izraza suradnja dvaju dijelova: što želimo i koliko toga želimo. Prvi se naziva kvalifikator, a drugi kvantifikator.

| **Kvalifikator** | **Značenje** |
| --- | --- |
| <znak> | Točno taj znak |
| [...] | Skup znakova unutar uglatih zagrada |
| [^...] | Komplement skupa znakova, tj. ne uključuje znakove unutar uglatih zagrada |
| . | Bilo koji znak osim kraja reda |
| \w | Bilo koje slovo *ASCII*, podcrta ili broj, isto kao [a-zA-Z0-9\_] |
| \W | Bilo koji znak koji nije slovo *ASCII*, podcrta ili broj, isto kao [^a-zA-Z0-9\_] |
| \s | Razmak |
| \S | Bilo koji znak koji nije razmak |
| \d | Broj, isto kao [0-9] |
| \D | Bilo koji znak koji nije broj, isto kao [^0-9] |
| {*n*,*m*} | Prethodni kvalifikator mora se pojaviti najmanje *n* puta, a najviše *m* puta. |
| {*n*,} | Prethodni kvalifikator mora se pojaviti najmanje *n* puta (ili više). |
| {*n*} | Prethodni kvalifikator mora se pojaviti točno *n* puta. |
| ? | Prethodni kvalifikator može se pojaviti jednom, a i ne mora; isto kao {0,1}. |
| + | Prethodni kvalifikator mora se pojaviti barem jednom; isto kao {1,}. |
| \* | Prethodni kvalifikator može se pojaviti bilo koliko puta, a i ne mora; isto kao {0,}. |

Ako je u proizvoljan izraz potrebno staviti nekoliko mogućnosti, tada se one grupiraju u okrugle zagrade i odvoje okomitom crtom. Na primjer, sljedeći izraz odgovara brojevima 099, 098, 091 i 092:

Sljedeći izraz odgovara bilo kojem načinu pisanja riječi *JavaScript*:

Analiza izraza:

^ Znak za početak (retka ili niza znakova).

[Jj] Na početku se može pojaviti bilo koji znak (samo jedan) iz skupa (ovdje J i j).

(ava)? Niz znakova „ava” (grupirani okruglim zagradama); može se, ali i ne mora pojaviti.

\s? Jedan razmak; može se pojaviti, ali i ne mora.

[Ss] Može se pojaviti bilo koji znak (samo jedan) iz skupa (ovdje S i s).

$ Znak za kraj (retka ili niza znakova).

Dakle, bilo koji od sljedećih nizova odgovara izrazu: „JavaScript”, „javascript”, „JScript”, „Java Script”, pa čak i „javaScript”.

Nedostaje funkcija koja će uneseni niz znakova usporediti s navedenim izrazom i vratiti informaciju odgovara li niz znakova izrazu. Najprije obrazac:



U ovom obrascu dopušteno je da se za ime i prezime upišu samo slova (*ASCII* i slova s dijakritičkim znakovima), razmak i crtica – barem jednom (uočite znak + pred kraj izraza). Za telefonski broj može se unijeti predbroj i znak za razdvajanje (uočite ? iza grupirajućih zagrada). Predbroj počinje znakom 0 nakon kojega slijedi 1 ili dvije znamenke. Znak za razdvajanje može biti razmak ili kosa crta (ovdje osigurana s obrnutom kosom crtom, jer kosa crta označava početak i kraj regularnog izraza). Nakon toga slijedi jedan broj od 1 do 9 (prva nula nema smisla) te nakon njega dva ili tri broja koje može, ali ne mora, slijediti crtica za razdvajanje. Na kraju slijede točno tri broja. Ovdje se nalazi program koji provjerava izraze (program uz provjeru izraza i vraća fokus na polje koje nije ispravno, jer se hvata događaj blur koji se okida u trenutku napuštanja polja):



Metoda test objekta RegExp provjerava zadovoljava li niz znakova, proslijeđen kao argument, zadani regularni izraz.

Program se doradi tako da se doda argument koji predstavlja poruku koja će se prikazati prilikom pogreške te se korisniku pomogne tako da se iz njegova neispravnog niza znakova uklone znakovi koji tu ne smiju biti (no, to još uvijek ne znači da je niz pravilnog oblika):



Odgovarajući obrazac:



9.4. Upravljanje preglednikom

Objekt window omogućuje rukovanje prozorom preglednika. Rukovanje trenutačnim prozorom preglednika često su zlorabili oglašivači pa je u današnjim preglednicima moguće isključiti odziv na mijenjanje izgleda prozora preglednika. Zato je u primjeru prvo potrebno otvoriti novi prozor da bi se mogla mijenjati njegova svojstva:



Pomoću ovih se funkcija *JavaScript*aupravlja stanjem preglednika:



Često je potrebno kakav sadržaj (najčešće hijerarhijski organiziran sadržaj) prikazati u novom prozoru. To se postiže uporabom funkcije window.open koja ima općeniti oblik:

Treći argument je niz znakova u kojem su zapisani parametri kao parovi ključa i vrijednosti (width=100, height=50). Od parametara su najzanimljiviji:

* top, left, height, width: y koordinata gornjeg ruba, x koordinata lijevog ruba, visina, širina
* menubar, toolbar, location, scrollbars, status, resizable: izbornik, alatna traka, adresno polje, trake za pomak, statusno područje; prozoru se može mijenjati veličina.

Sljedećim obrascem kontrolira se izrada novog prozora:



Ovim se programom *JavaScript*a izrađuje novi prozor:



9.5. Vježba: Korisne skripte

1. Izradite HTML-datoteku naziva skripte.html na kojoj se nalazi jedna slika sa okvirom debljine 5px, crne boje. Slika treba biti visine 300px. Ispod nje nalazi se gumb sa tekstom Slika 3. Također stvorite polje za unos teksta i tipku sa tekstom Provjeri ispod tog polja.
2. Svaki put kada korisnik prijeđe sa mišem preko slike slika se treba zamjeniti sa drugom slikom i visina joj se treba postaviti na 400px. Okvir slike treba promjeniti boju u nasumično generiranu boju. (Koristite naredbu Math.floor(Math.random() \* 256) i postavite rgb vrijednosti boja u CSS (rgb(crvena, zelena, plava)))
3. Kada korisnik makne miš sa slike slika se treba vratiti u prvotno postavljenu sliku, te joj se visina treba smanjiti na 300px.
4. Kada korisnik prijeđe preko gumba sa tekstom Slika 3, slika se treba zamjeniti sa trećom slikom, a pozadina gumba promijeniti u narančastu boju.
5. Korisnik u polje za unos teksta treba unijeti broj između 0 i 99. Ako se uneseni broj ne nalazi u tom rasponu, korisnik nakon pritiska na tipku Provjeri treba dobiti obavijest da je unio neispravnu vrijednost, te da mora unijeti broj između 0 i 99. Ako je broj ispravan korisnik treba dobiti obavijest da je unio ispravnu vrijednost.
6. Dodaci
8. 1. Zadaci

**Pogađanje brojeva**

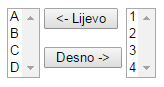
A1. Napišite program koji će naći slučajan broj (funkcijom Math.random() koja vraća slučajan broj od 0.0 do 1.0) do 100. Ispišite dobiveni broj.

B1. Napišite program koji će dopustiti korisniku unos broja od 0 do 100 u jedno tekstno polje. Program mora pamtiti koji je to unos po redu te ispisivati sve prije unesene brojeve.

C1. Združite programe pod A i B te napišite novi program koji korisniku dopušta da pogodi koji je slučajan broj program našao. Korisnik mora u pregledniku vidjeti koji mu je to pokušaj, koji broj je unio te sve svoje prijašnje pokušaje. Najveći broj pokušaja je 7, nakon toga se ispiše slučajan broj.

**Odabir većeg broja vrijednosti**

A2. Napišite program koji će na kraj elementa <select> dodavati vrijednosti iz tekstnog polja.

B2. Napišite program koji će na kraj elementa <select> dodavati vrijednosti iz tekstnog polja. Omogućite i brisanje odabrane stavke iz popisa.

C2. Napišite program (rabeći postupke iz A i B) koji će prebacivati vrijednosti iz jedne liste u drugu.

* 1. Kompletan kôd obrasca iz cjeline 7.

HTML kôd:







*JavaScript* kôd:





* 1. Rješenja vježbi

**Vježba 1.8: Početak rada s *JavaScriptom***

2.



4. 

5. 

6. 

7. 



**Vježba 2.4: Upoznavanje s jezikom *JavaScript***

2.



3. 

4. 

5. 

6. 

**Vježba 3.4: Varijable i objekti**

**Rezultat**

**Broj:** 1  
**Niz 1:** Niz znakova  
**Niz 2:** 3.14  
**Niz 3:** U dva  
retka!  
**Logička:** true  
**Logička:** false



**Vježba 4.6: Operatori**

**Rezultat**

Početna vrijednost iA: 10  
Početna vrijednost iB: 3  
Suma: 13  
Razlika: 7  
Modulo: 1  
iA nakon ++: 11  
iB nakon --: 2  
Sumi dodamo iA: 24  
Je li iA > 5: true  
Paziti na konverzije iA + iB: 112  
Paziti na konverzije (iA + iB): 13

<script type="text/javascript">

**2., 3., 5.:**

var iA = 10,

iB = 3,

iSuma,

iRazlika,

iModulo,

bLogicka;

**4.**

iSuma = iRazlika = iModulo = 0;

**6.**

document.write('Početna vrijednost iA: ' + iA + '<br />');

document.write('Početna vrijednost iB: ' + iB + '<br />');

**7.**

iSuma = iA + iB;

iRazlika = iA - iB;

iModulo = iA % iB;

document.write('Suma: ' + iSuma + '<br />');

document.write('Razlika: ' + iRazlika + '<br />');

document.write('Modulo: ' + iModulo + '<br />');

**8.**

iA++;

iB--;

document.write('iA nakon ++: ' + iA + '<br />');

document.write('iB nakon --: ' + iB + '<br />');

**9.**

iSuma += iA;

document.write('Sumi dodamo iA: ' + iSuma + '<br />');

**10.**

bLogicka = iA > 5;

document.write('Je li iA > 5: ' + bLogicka + '<br />');

**11.**

document.write('Paziti na konverzije iA + iB: ' +

iA + iB + '<br />');

**12.**

document.write('Paziti na konverzije (iA + iB): ' +

(iA + iB) + '<br />');

</script>

**Vježba 5.4: Funkcije**

**Rezultat**

Dobrodošli u Velegrad!  
Velegrad je 150 kilometara udaljen od mora.



**Vježba 6.7: Naredbe za kontrolu tijeka**

<script type="text/javascript">

**2.**

function ispis(sPoruka) {

document.write(sPoruka, '<br />');

}

**3., 4., 5.:**

var iA = 10,

**Rezultat**

iA je manji  
Zbrajanje  
Prava razlika: 3  
For: redak 1  
For: redak 2  
For: redak 3  
For: redak 4  
For: redak 5  
For: redak 6  
For: redak 7  
For: redak 8 For: redak 9  
For: redak 10  
While: redak 1  
While: redak 2  
While: redak 3  
While: redak 4  
While: redak 5  
While: redak 6  
While: redak 7  
While: redak 8 While: redak 9  
While: redak 10

iB = 13,

sOperacija = '+',

iRazlika,

iBrojac,

jBrojac;

**6.**

if (iA > iB) {

ispis("iA je veci");

} else {

ispis("iA je manji");

}

**7.**

switch (sOperacija) {

case '+':

ispis('Zbrajanje');

break;

case '-':

ispis('Oduzimanje');

break;

case '\*':

ispis('Množenje');

break;

case '/':

ispis('Dijeljenje');

break;

default:

ispis('Nepoznato');

}



**Vježba 7.5: Obrasci**

1.



2., 3., 4.:



**Vježba 9.5: Korisne skripte**

1.



2.



3.



4.



5.



* 1. Dodatni materijali

**A. Operatori**

U ovoj tablici stupac označen s „P” je prioritet operatora, a stupac „A” je asocijativnost operatora (koja može biti lijeva ili desna).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P** | **A** | **Operator** | **Tip operanda** | **Opis** |
| 15 | L | . | objekt, varijabla | svojstvo ili metoda |
| 15 | L | [] | polje, cijeli broj | indeks polja |
| 15 | L | ( ) | funkcija, argumenti | poziv funkcije |
| 14 | D | ++ | broj ili vraijabla | pre- ili post-inkrement (unaran) |
| 14 | D | -- | broj ili varijabla | pre- ili post-dekrement (unaran) |
| 14 | D | - | broj | unarni minus (negacija) |
| 14 | D | + | broj | unarni plus (nema-operacije) |
| 14 | D | ~ | cijeli broj | bit komplement (unaran) |
| 14 | D | ! | boolean | logički komplement (unaran) |
| 14 | D | typeof | bilo koji | tip podatka (unaran) |
| 14 | D | void | bilo koji | nedefinirana vrijednost (unaran) |
| 13 | L | \*, /, % | broj | množenje, dijeljenje, ostatak |
| 12 | L | +, - | broj | zbrajanje, oduzimanje |
| 12 | L | + | niz znakova | spajanje niza znakova |
| 11 | L | << | cijeli broj | pomak u lijevo |
| 11 | L | >> | cijeli broj | pomak u desno (puni se 1) |
| 10 | L | >>> | cijeli broj | pomak u desno (puni se 0) |
| 10 | L | <, <= | broj ili niz znakova | manje i manje ili jednako |
| 10 | L | >, >= | broj ili niz znakova | veće i veće ili jednako |
| 10 | L | in | niz znakova, objekt | provjera da li postoji svojstvo |
| 9 | L | == | bilo koji | test jednakosti |
| 9 | L | != | bilo koji | test nejednakosti |
| 9 | L | === | bilo koji | test identičnosti |
| 9 | L | !== | bilo koji | test neidentičnosti |
| 8 | L | & | cijeli broj | bit AND |
| 7 | L | ^ | cijeli broj | bit XOR |
| 6 | L | | | cijeli broj | bit OR |
| 5 | L | && | boolean | logički AND |
| 4 | L | || | boolean | logički OR |
| 3 | D | ?: | boolean, bilo koji, bilo koji | uvjetni operator (3 operanda) |
| 2 | D | = | broj ili varijabla,  bilo koji | pridruživanje |
| 2 | D | \*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |= | broj ili varijabla, bilo koji | pridruživanje s operacijom |
| 1 | L | , | bilo koji | višestruki izračun izraza |

**B. Polja**

Pokazano je da se polja stvaraju preko objekta Array. Ovdje je navedeno koja svojstva ima polje i koje sve metode postoje za rukovanje poljima.

Najvažnije svojstvo polja je duljina polja. Duljina polja je broj za jedan veći od najvećeg indeksa u polju. Budući da su polja u *JavaScript*u dinamička, duljina se osvježi svakog puta kad se promijeni polje:

Tipična primjena bila bi ovakva:

Metode su za rukovanje poljima:

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Primjer** | **Značenje** |
| join | a.join("<separator>") | Metoda koja spaja sve elemente polja u jedan niz znakova odvojenih nizom znakova <separator>. |
| reverse | a.reverse() | Metoda koja razmješta elemente polja obratno, tj. prvi element postaje zadnji, zadnji element prvi. |
| sort | a.sort(<funkcija>) | Razvrstava polje po redoslijedu *ASCII*. Ako je potreban drugi redoslijed, kao argument se navodi ime funkcije koja će usporediti vrijednosti. |
| concat | a.concat(<polje>) | Spaja polje a i polje koje je navedeno kao argument. |
| slice | a.slice(<početak>, <kraj>) | Vraća elemente polja od indeksa <početak> do indeksa <kraj> kao novo polje. Kad je indeks negativan, indeks se računa od kraja polja. |
| push | a.push(<polje>) | Dodaje na kraj polja elemente u polju <polje>. |
| pop | a.pop() | Briše zadnji element iz polja i vraća njegovu vrijednost. |
| shift | a.shift() | Briše prvi element iz polja i vraća njegovu vrijednost. |
| unshift | a.unshift(<polje>) | Dodaje na početak polja elemente u polju <polje>. |

**C. Datumi**

Rukovanje datumima vrši se preko objekta Date. Novi se objekt radi ovako:



Operacije na datumima obavljaju se kao i na brojevima, jer su interno datum i vrijeme broj milisekundi proteklih od 1. siječnja 1970.



Neke metode za rukovanje datumima i vremenom:

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Primjer** | **Značenje** |
| getTime | d.getTime() | Vraća broj proteklih milisekundi od 1.1.1970. |
| getSeconds | d.getSeconds() | Vraća broj sekundi (0-59) od određenog datuma. |
| getMinutes | d.getMinutes() | Vraća broj minuta (0-59) od određenog datuma. |
| getHours | d.getHours() | Vraća broj sati (0-23) od određenog datuma. |
| getDate | d.getDate() | Vraća dan u mjesecu (1-31) od određenog datuma. |
| getDay | d.getDay() | Vraća dan u tjednu (0:nedjelja – 6:subota) od određenog datuma. |
| getMonth | d.getMonth() | Vraća mjesec u godini (0:siječanj-11:prosinac) od određenog datuma. |
| getFullYear | d.getFullYear() | Vraća godinu od određenog datuma. |
| setSeconds | d.setSeconds() | Postavlja broj sekundi za određeni datum. |
| setMinutes | d.setMinutes() | Postavlja broj minuta za određeni datum. |
| setHours | d.setHours() | Postavlja broj sati za određeni datum. |
| setDate | d.setDate() | Postavlja dan u mjesecu za određeni datum. |
| setDay | d.setDay() | Postavlja dan u tjednu za određeni datum. |
| setMonth | d.setMonth() | Postavlja mjesec u godini za određeni datum. |
| setFullYear | d.setFullYear() | Postavlja godinu za određeni datum. |
| toString | d.toString() | Vraća niz znakova koji predstavlja određeni datum. |

**D. Matematičke funkcije**

Objekt Math definira nekoliko konstanti i funkcija koje su potrebne za složenije matematičke operacije.

Konstante:

|  |  |
| --- | --- |
| **Konstanta** | **Značenje** |
| Math.E | Baza prirodnog logaritma (*e*). |
| Math.LN10 | Prirodni logaritam od 10. |
| Math.LN2 | Prirodni logaritam od 2. |
| Math.LOG10E | Logaritam s bazom 10 od e. |
| Math.LOG2E | Logaritam s bazom 2 od e. |
| Math.PI | Konstanta π |
| Math.SQRT1\_2 | Konstanta kvadratni korijen od 1/2. |
| Math.SQRT2 | Konstanta kvadratni korijen od 2. |

Funkcije:

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Primjer** | **Značenje** |
| abs | Math.abs(n) | Vraća apsolutnu vrijednost broja. |
| acos | Math.acos(n) | Vraća arkus kosinus broja. |
| asin | Math.asin(n) | Vraća arkus sinus broja. |
| atan | Math.atan(n) | Vraća arkus tangens broja. |
| atan2 | Math.atan2(x,y) | Vraća arkus tangens dva broja (kut koji tvore x i y). |
| ceil | Math.ceil(n) | Vraća najbliži cijeli broj koji je veći ili jednak n. |
| cos | Math.cos(n) | Vraća kosinus broja. |
| exp | Math.exp(n) | Vraća *en*. |
| floor | Math.floor(n) | Vraća najbliži cijeli broj koji je manji ili jednak n. |
| log | Math.log(n) | Vraća prirodni logaritam broja (ln *n*). |
| max | Math.max(a,b) | Vraća veći broj od dva. |
| min | Math.min(a,b) | Vraća manji broj od dva. |
| pow | Math.pow(x,y) | Vraća *xy*. |
| random | Math.random(n) | Vraća slučajan broj od 0 do 1. |
| round | Math.round(n) | Zaokružuje broj na najbliži cijeli broj. |
| sin | Math.sin(n) | Vraća sinus broja. |
| sqrt | Math.sqrt(n) | Vraća kvadratni korijen broja. |
| tan | Math.tan(n) | Vraća tangens broja. |

## 

**E. Događaji**

Važniji događaji definirani u DOM-u:

|  |  |  |
| --- | --- | --- |
| **Događaj** | **DOM svojstvo** | **Pokreće se** |
| abort | onabort | Prekinuto učitavanje slike. |
| blur | onblur | Element gubi fokus za unos. |
| change | onchange | Izbor u <select> elementu ili drugim elementima gubi fokus i vrijednost mu se promijeni od kada je dobio fokus. |
| click | onclick | Klik mišem na element. |
| dblclick | ondblclick | Dvostruki klik mišem na element. |
| error | onerror | Došlo je do pogreške prilikom učitavanja slike. |
| focus | onfocus | Element je dobio fokus za unos. |
| keydown | onkeydown | Pritisnuta tipka. |
| keypress | onkeypress | Pritisnuta i otpuštena tipka. |
| keyup | onkeyup | Tipka je otpuštena. |
| load | onload | Učitavanje dokumenta je završeno. |
| mousedown | onmousedown | Tipka miša je pritisnuta (nije još otpuštena). |
| mousemove | onmousemove | Miš je pomaknut. |
| mouseout | onmouseout | Miš je pomaknut izvan elementa. |
| mouseover | onmouseover | Miš je pomaknut na element. |
| mouseup | onmouseup | Tipka miša je otpuštena. |
| reset | onreset | Obrazac je postavljena na početne vrijednosti. |
| resize | onresize | Veličina prozora preglednika se promijenila. |
| select | onselect | Tekst je odabran (selektiran, zacrnjen). |
| submit | onsubmit | Obrazac je poslan na obradu. |
| unload | onunload | Na mjesto trenutačnog dokumenta učitava se novi. |

**F. Niz znakova**

Nizovi znakova usko su vezani uz objekt String, iako se rijetko stvaraju eksplicitnim korištenjem objekta. Za objekt String definiran je velik broj metoda. Tu su opisane najčešće korištene metode:

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Primjer** | **Značenje** |
| charAt | sIme.charAt(n) | Vraća znak koji se nalazi na poziciji n. |
| charCodeAt | sIme.charCodeAt(n) | Vraća *Unicode* kôd znaka koji se nalazi na poziciji n. |
| concat | sIme.concat(sPrezime, sAdresa,. . .) | Spaja niz znakova s nizovima znakova koji su proslijeđeni. |
| indexOf | sIme.indexOf(znak) | Vraća indeks na kojemu se nalazi prvo pojavljivanje znaka znak. |
| lastIndexOf | sIme.lastIndexOf(znak) | Vraća indeks na kojemu se nalazi zadnje pojavljivanje znaka znak. |
| split | sIme.split(delimiter) | Dijeli niz znakova na mjestima na kojima se nalazi delimiter i vraća polje kao rezultat. |
| substr | sIme.substr(indeks, duljina) | Vraća podniz od pozicije indeks u duljini duljina. |
| substring | sIme.substring(indeksA, indeksB) | Vraća podniz od pozicije indeksA do pozicije indeksB. |
| toLowerCase | sIme.toLowerCase() | Vraća niz znakova u kojemu su sva slova pretvorena u mala. |
| toUpperCase | sIme.toUpperCase() | Vraća niz znakova u kojemu su sva slova pretvorena u velika. |

Niz znakova ima i svojstvo length u koje je pohranjena trenutačna duljina niza znakova.

**G. Aplikacija za uređivanje teksta – *Brackets***

Preporučena aplikacija za uređivanje HTML, CSS i *JavaScripta* datoteka je *Brackets* - http://brackets.io/.

Preporučeni dodaci za *Brackets* koji ubrzavaju pisanje i razvoj:

* *JSHint*
* *QuickDocsJS*
* *Emmet*
* *Brackets File Icons*
* *jsbeautifier*
* *CDN Finder*
* *Minifier*
* *Various improvements*

Napomena: *Various improvements* instalirati zadnji.

Također postoje mnogi drugi izvrsni uređivači teksta kao što su:

* Sublime Text - https://www.sublimetext.com/
* Atom - https://atom.io/
* Visual Studio Code - https://code.visualstudio.com/
* Notepad++ (nema iOS verzije) –

<https://notepad-plus-plus.org/>

***Bilješke:***