POGLAVLJE 4

JavaScript

4.1 Kratak uvod u JavaScript

HTML5 možemo posmatrati kao građevinu koju podupiru tri stuba: HTML, CSS i JavaScript. Analizirali smo elemente ugrađene u HTML i nova svojstva koja CSS čine idealnom alatkom za dizajnere, a sada ćemo upoznati i jedan od najjačih „stubova": JavaScript.

JavaScript je interpretirani jezik koji ima više namena, a do sada se posmatrao samo kao dopuna HTML-a i CSS-a. Jedna od inovacija koja je pomogla da se pogled na JavaScript promeni bio je novi mehanizam čitača veba namenjen ubrzavanju obrade skriptova. On je pretvarao skriptove u mašinski kôd da bi brzina izvršavanja bila poput onih u samostalnim aplikacijama. Ove poboljšane mogućnosti znatno poboljšavaju performanse JavaScripta i pretvaraju ga u najbolju opciju kodiranja za veb.

Za izvršavanje JavaScript koda neophodno je hosting okruženje, ali je zahvaljujući novim interpreterima i kompajlerima poput Googleovog V8, on postao nezavisan i moćan jezik koji može da radi u gotovo svim okruženjima. Zaslužan je za revoluciju na internetu i procvat tržišta mobilnih uređaja.

Da bi se potpuno iskoristila napredna arhitektura JavaScripta, poboljšane su mogućnosti prenosivosti i integracije. Osim toga, u sve čitače veba podrazumevano su ugrađeni svi interfejsi za programiranje aplikacija (engl. *application programming interfaces*, *APIs*) koji dopunjavaju JavaScript osnovnim mogućnostima. Ovi novi API-ji (kao što su Web Storage, Canvas i drugi) predstavljaju interfejse za biblioteke ugrađene u čitače veba. Osnovna zamisao je da te moćne osobine budu svuda dostupne preko jednostavnih standardnih metoda za programiranje, proširujući opseg delovanja jezika i olakšavajući izradu atraktivnog i korisnog softvera za veb.

JavaScript je danas najpopularniji i najperspektivniji jezik, a HTML5 ga pretvara u osnovnu alatku programera za veb i mobilne uređaje. U ovom poglavlju upoznaćete osnovne koncepte JavaScripta, naučiti kako se ugrađuju skriptovi u HTML dokumente i primenjuju najnovije izmene u jeziku, što će vas pripremiti za ostale sadržaje u knjizi.

VAŽNO U ovoj knjizi predstavljen je uvod u JavaScript. Radićemo s jednostavnim i složenim osobinama, ali ćemo primenjivati samo osnovne koncepte neophodne za savladavanje novina u jeziku HTML5. Da biste proširili znanja u ovoj oblasti, posetite našu veb lokaciju i pratite veze za ovo poglavlje. Ako su vam ove informacije poznate, slobodno preskočite delove koje ste već savladali.

Jezik

JavaScript je programski jezik koji se potpuno razlikuje od HTML-a i CSS-a. HTML je jezik za označavanje, poput koda koji čitač veba interpretira kako bi organizovao informacije, a CSS se može posmatrati kao lista stilova (premda ga je nova specifikacija pretvorila u alatku koja je u većoj meri dinamička). S druge strane, JavaScript je jezik za pisanje skriptova, ali se može uporediti s drugim, „pravim" programskim jezicima, kao što su C++ ili Java. Jedina važna razlika između drugih programskih jezika i JavaScripta jeste njegova priroda. Za razliku od savremenih jezika koji su objektno orijentisani, JavaScript je jezik za pisanje skriptova zasnovan na prototipovima što ga, paradoksalno, mnogo više usmerava na objekte nego druge jezike, u šta ćete se i sami uveriti.

JavaScript može da obavlja brojne zadatke – od davanja instrukcija do izračunavanja složenih algoritama – ali njegova najvažnija osobina, kao i drugih programskih jezika, jeste sposobnost memorisanja i obrade podataka, što se postiže pomoću promenljivih.

Promenljive

Memorija računara ili mobilnog uređaja liči na ogromno saće, s milionima ćelija u koje se smeštaju podaci. Te ćelije imaju adrese, tj. brojeve koji ih identifikuju. Ćelije imaju ograničen prostor, a za smeštanje veće količine podataka najčešće je potrebna grupa od nekoliko ćelija. Rad sa takvim prostorom za čuvanje podataka je složen, i zato programski jezik sadrži promenljive (engl. *variables*) koje olakšavaju identifikaciju vrednosti u memoriji.

Promenljive su imena dodeljena „ćeliji" ili grupi „ćelija" u kojima će se podaci čuvati. Na primer, da bismo u memoriju smestili vrednost 5, treba da znamo gde se taj broj čuva kako bi se kasnije mogao preuzeti. Promenljive omogućavaju da se taj prostor identifikuje pomoću odgovarajućeg imena i da se, upotrebom tog imena, preuzme sadržaj „ćelije".

Da bi deklarisao promenljivu, JavaScript upotrebljava rezervisanu reč **`var`**.

Listing 4-1: *Deklarisanje promenljive u JavaScriptu.*

```
<script>
  var mynumber = 2;
</script>
```

> **Podsetite se osnova** Oznake **<script>** javljaju čitaču veba da je izvorni kôd koji je umetnut između njih, napisan na JavaScriptu. To je jedan od načina da JavaScript kôd ugradite u HTML dokument, a uskoro ćemo analizirati i alternativna rešenja.

U listingu 4-1 pravimo promenljivu **mynumber** i u nju smeštamo vrednost **2**. Kažemo da je vrednost **dodeljena** (engl. *assigned*) promenljivoj. Prema tome, „dodelili smo vrednost **2** promenljivoj **mynumber"**.

JavaScript rezerviše prostor u memoriji, upisuje broj 2, pravi referencu na taj prostor i dodeljuje je imenu **mynumber**. Kad god upotrebimo tu referencu (ime **mynumber**), dobijamo broj **2**.

Listing 4-2: *Upotreba sadržaja promenljive.*

```
<script>
  var mynumber = 2;
  alert(mynumber);
</script>
```

U listingu 4-2 pravimo promenljivu i dodeljujemo joj vrednost 2 (sadržaj promenljive **mynumber** ima vrednost 2). Pomoću metode **alert()** sadržaj ove promenljive prikazuje se na ekranu.

> **Podsetite se osnova** Metoda **alert()** je unapred definisana metoda JavaScripta, koja generiše iskačuće prozore i prikazuje određene informacije na ekranu. U prozoru se prikazuju vrednosti deklarisane u zagradama (npr., **alert("Hello World");**). Kasnije ćemo proučiti ovu metodu i druge koje su joj slične.

> **Uradite sami** Za testiranje koda JavaScripta ne moramo praviti celu HTML strukturu. Izvorni kôd iz listinga 4-2 dovoljan je da čitači veba interpretiraju JavaScript. Kopirajte taj izvorni kôd u praznu tekstualnu datoteku, snimite je sa odgovarajućim imenom i nastavkom **.html** (npr., **jscode.html**), i otvorite je u svom čitaču veba. Videćete iskačući prozor koji prikazuje vrednost 2.

Promenljive se tako nazivaju zato što nisu stalne. Vrednost koja im je dodeljena možemo menjati kad god poželimo, što i jeste njihova najvažnija osobina.

Listing 4-3: *Dodeljivanje nove vrednosti promenljivoj.*

```
<script>
  var mynumber = 2;
  mynumber = 3;
  alert(mynumber);
</script>
```

Nakon što smo u listingu 4-3 napravili promenljivu **mynumber**, dodelićemo joj novu vrednost. Sada metoda **alert()** prikazuje broj 3. Rezervisana reč **var** ne mora da se koristi, jer se ovde nova vrednost dodeljuje samo pomoću imena promenljive i znaka =.

U stvarnim situacijama, vrednost promenljive obično koristimo za izvršavanje neke operacije i dodeljivanje rezultata toj istoj promenljivoj:

Listing 4-4: *Upotreba vrednosti promenljive.*

```
<script>
  var mynumber = 2;
  mynumber = mynumber + 1;
  alert(mynumber);
</script>
```

U ovom primeru, vrednost **1** dodaje se važećoj vrednosti promenljive **mynumber**, a rezultat se dodeljuje istoj promenljivoj. To je isto kao da ste sabrali 2 + 1, s tom razlikom što upotreba promenljive umesto broja omogućava da vrednost kasnije, tokom izvršavanja koda, promenite. Ako primenimo prethodni postupak i umesto broja **2** u promenljivu upišemo broj **5**, rezultat sabiranja biće **6**, a ne **3**.

Ovo ilustruje kolika je moć promenljivih. Ako izvršimo bilo koju matematičku operaciju ili spajanje teksta, promenljiva uvek zadržava poslednju vrednost koja joj je dodeljena.

> **Podsetite se osnova** Osim operatora **+**, JavaScript sadrži brojne druge operatore. Najčešće se koriste operatori **+** (sabiranje), **–** (oduzimanje), ***** (množenje) i **/** (deljenje), ali postoje i logički operatori, kao što su **&&** (i) i **||** (ili), operatori poređenja, kao što su **==** (jednako), **!=** (nije jednako), **<** (manje od), **>** (veće od), i mnogi drugi. Neke od njih upotrebićete u nastavku ove knjige, a ceo spisak potražite na našoj veb lokaciji (pratite veze za ovo poglavlje).

Promenljive mogu da sadrže brojeve, tekst i unapred definisane vrednosti (kao što su logičke vrednosti **true** ili **false**), ili osnovne vrednosti (kao što su **null** ili **undefined**). Brojevi se izražavaju kao i u prethodnim primerima, a tekst se mora nalaziti u polunavodnicima ili navodnicima.

Listing 4-5: *Dodeljivanje znakovnog niza promenljivoj.*

```
<script>
  var mytext = "Hello World!";
  alert(mytext);
</script>
```

Promenljive mogu da sadrže i nekoliko vrednosti istovremeno. Za to se koriste strukture koje se zovu **nizovi** (engl. *arrays*) i koji su višedimenzionalne promenljive. Nizovi se definišu pomoću jednostavne notacije sa uglastim zagradama, unutar kojih se vrednosti razdvajaju zarezima. Te vrednosti se kasnije identifikuju pomoću indeksa, počevši od vrednosti 0.

Listing 4-6: *Pravljenje višedimenzionalne promenljive.*

```
<script>
  var myarray = ["red", "green", "blue"];
  alert(myarray[0]);
</script>
```

U listingu 4-6 pravimo niz **`myarray`** koji sadrži tri vrednosti: *red*, *green* i *blue*. JavaScript automatski dodeljuje indeks **`0`** za prvu vrednost, **`1`** za drugu i **`2`** za treću. Da bismo preuzeli ove informacije, treba da navedemo indeks kad god koristimo niz. To se radi tako što se broj indeksa stavi u uglaste zagrade iza imena promenljive. Na primer, da bismo dobili prvu vrednost niza **`myarray`**, pišemo **`myarray[0]`**, kao u našem primeru.

Nizovi, kao i promenljive, mogu da sadrže bilo koji tip vrednosti. Možemo da napravimo niz poput onog iz listinga 4-6, koristeći brojeve, znakovne nizove, logičke vrednosti, a vrednost možemo da deklarišemo i kao **`praznu`** (**`null`**) ili **`nedefinisanu`** (**`undefined`**).

Listing 4-7: *Upotreba različitih tipova vrednosti.*

```
<script>
  var myarray = ["red", , 32, null, "HTML5 is awesome!"];
  alert(myarray[1]);
</script>
```

> **Podsetite se osnova** Vrednosti **`null`** i **`undefined`** ugrađene su u JavaScript. Obe predstavljaju odsustvo vrednosti, ali **`undefined`** može da označava i stanje kada neko svojstvo objekta ili neki element niza nisu definisani. U primeru iz listinga 4-7, drugi element niza nije definisan. Ako pokušamo da učitamo ovaj element, dobićemo vrednost **`undefined`**.
>
> **Uradite sami** Kopirajte kôd iz listinga 4-7 u praznu tekstualnu datoteku, snimite je sa odgovarajućim imenom i nastavkom **`.html`** (npr., **`jscode.html`**), i otvorite datoteku u svom čitaču veba. Menjajte indeks tako da prikažete svaku vrednost niza.

Nad nizovima možemo da izvršavamo operacije i memorišemo rezultat, kao što smo radili i s pojedinačnim promenljivama.

Listing 4-8: *Rad sa nizovima.*

```
<script>
  var myarray = ["red", 32];
  alert(myarray[0]);
  myarray[0] = myarray[0] + " color";
  myarray[1] = myarray[1] + 10;
  alert(myarray[0] + " " + myarray[1]);
</script>
```

Pomoću skripta iz listinga 4-8 naučili smo kako se menja vrednost niza, ali i kako se spaja tekst. Nizovi se ponašaju isto kao i pojedinačne promenljive, s tom razlikom što navodimo indeks kad god ih koristimo. Ako upotrebimo izraz **`myarray[1] = myarray[1] + 10`**, naložićemo interpreteru JavaScripta da preuzme važeću vrednost niza **`myarray`** sa indeksom **`1`** (32), da toj vrednosti doda 10 i sačuva rezultat (42) u istom nizu i sa istim indeksom; vrednost **`myarray[1]`** biće 42.

Operator **`+`** koristi se sa brojevima i znakovnim nizovima (tekstom); u slučaju znakovnih nizova, on spaja tekst i generiše nov znakovni niz koji sadrži obe vrednosti. U našem primeru, vrednost **`myarray[0]`** bila je „red“, ali je kasnije dodat znakovni niz „color“ i sada je vrednost niza **`myarray`** sa indeksom **`0`** „red color“.

> **Podsetite se osnova** Na kraju poslednjeg skripta, metoda **`alert()`** prikazuje obe vrednosti niza razdvojene razmacima. Vrednosti i razmak povezuje operator **`+`**. Ovaj operator se koristi ne samo za izvršavanje operacija, nego i za spajanje (nadovezivanje) vrednosti i znakovnih nizova radi stilizovanja (brojevi se automatski prevode u znakovne nizove).

JavaScript pruža jednostavan način za rad sa nizovima. Vrednosti možemo izdvajati ili dodavati pomoću narednih metoda:

`push(`*`vrednost`*`)` – Pomoću ove metode se dodaje nova vrednost na kraj niza. Atribut ***`vrednost`*** je vrednost koja se dodaje.

`shift()` – Pomoću ove metode uklanja se i vraća prva vrednost niza.

`pop()` – Pomoću ove metode uklanja se i vraća poslednja vrednost niza.

Listing 4-9: *Dodavanje vrednosti u niz i njeno izdvajanje iz niza.*

```
<script>
  var myarray = ["red", 32];
  myarray.push('car');
  alert(myarray[2]);
  alert(myarray.shift());
</script>
```

Vrednost dodata metodom **push()** dodeljuje se sledećem dostupnom indeksu u nizu. U listingu 4-9, vrednost koja se umeće metodom **push()** dobija indeks **2**. Nakon toga, taj indeks omogućava da se ista vrednost prikaže na ekranu.

Na kraju skripta izdvajamo i prikazujemo prvu vrednost niza na ekranu. Vrednosti izdvojene metodama **shift()** i **pop()** nisu više deo niza. Nakon izvršavanja skripta u ovom primeru, u nizu preostaju samo dve vrednosti: **32** i **car**.

Uslovi i petlje

U računarskim programima često se koriste uslovi i složeni ciklusi obrade. JavaScript sadrži ukupno četiri naredbe za obradu koda zavisno od uslova koje je odredio programer: **if**, **switch**, **for** i **while**.

Listing 4-10: *Provera uslova naredbom* `if`.

```
<script>
  var myvariable = 9;
  if(myvariable < 10) {
    alert("the number is smaller than 10");
  }
</script>
```

> **Podsetite se osnova** Preporuka je da se svaki red u JavaScriptu zatvori tačkom i zarezom, što nije neophodno nakon bloka naredaba (bloka koda koji se nalazi unutar vitičastih zagrada).

Naredbom **if** proverava se izraz u zagradama i obrađuju instrukcije u vitičastim zagradama ako je uslov tačan. U skriptu iz listinga 4-10, vrednost 9 dodeljuje se promenljivoj **myvariable,** a zatim se naredbom **if** promenljiva poredi sa brojem **10**. Ako je vrednost promenljive manja od **10**, metoda **alert()** prikazuje poruku na ekranu.

Ova naredba ima važan dodatak koji omogućava da izvršimo neku akciju bez obzira na to da li je uslov ispunjen ili nije. Naredbom **if else** proverava se uslov u zagradama, i – bilo da je uslov ispunjen ili nije – izvršava se određena (različita) akcija.

Listing 4-11: *Provera dva uslova pomoću naredbe* **if else**.

```
<script>
  var myvariable = 9;
  if(myvariable < 10) {
    alert("the number is smaller than 10");
  }else{
    alert("the number is 10 or bigger");
  }
</script>
```

U sledećem primeru, skript proverava dva uslova: kada je broj manji od **10,** i kada je broj jednak ili veći od **10**.

Za proveru više uslova, u JavaScriptu se koristi naredba **switch**.

Listing 4-12: *Upotreba naredbe* `switch`.

```
<script>
  var myvariable = 9;
  switch(myvariable) {
    case 5:
      alert("the number is five");
      break;
    case 8:
      alert("the number is eight");
      break;
    case 10:
      alert("the number is ten");
      break;
    default:
      alert("the number is " + myvariable);
  }
</script>
```

Naredba **switch** izračunava izraz (obično je to samo jedna promenljiva), uparuje rezultat s naredbama **case** u vitičastim zagradama i, u slučaju uspeha, izvršava instrukcije deklarisane za taj slučaj. U primeru iz listinga 4-12, naredba **switch** izračunava vrednost promenljive **myvariable**. Ta vrednost se poredi sa svakim pojedinačnim slučajem (engl. *case*). Na primer, ukoliko je vrednost **5**, kontrola se prenosi u prvi slučaj (**case**), a metoda **alert()** na ekranu prikazuje tekst „the number is five". Ako prvi slučaj (**case**) ne odgovara vrednosti promenljive, izračunava se sledeći slučaj, i tako dalje. Ukoliko nijedan uslov ne odgovara zadatoj vrednosti, izvršiće se podrazumevane instrukcije u slučaju označenom sa **default**.

Kôd u svakom uslovu **case** mora da se završi instrukcijom **break**. Ona obaveštava interpreter JavaScripta da ne mora da izvršava ostale naredbe.

Naredbe **switch** i **if** su korisne, ali obavljaju jednostavan zadatak: izračunavaju izraz, izvršavaju grupu instrukcija zavisno od rezultata i nakon toga vraćaju kontrolu skriptu. Postoje situacije u kojima to nije dovoljno. Instrukcije ponekad treba izvršiti po nekoliko puta za isti uslov, a ponekad se uslov mora ponovo izračunati nakon moguće izmene tokom procesa. U takvim slučajevima možemo koristiti dve naredbe: **for** i **while**.

Listing 4-13: *Upotreba naredbe* `for`.

```
<script>
  var myvariable = 9;
  for(var f = 0; f < myvariable; f++) {
    alert("The current number is " + f);
  }
</script>
```

Naredba **`for`** izvršava kôd u vitičastim zagradama sve dok je ispunjen uslov koji je deklarisan kao drugi parametar. Sintaksa ove naredbe je **`for(inicijalizacija; uslov; inkrement)`**. Prvi parametar definiše početne vrednosti za petlju; drugi parametar definiše uslov koji se proverava; poslednji parametar definiše kako se početne vrednosti menjaju u svakom ciklusu.

U skriptu iz listinga 4-13 deklarišemo promenljivu **`f`** i dodeljujemo joj početnu vrednost **`0`**. Zatim se ispituje ispunjenost uslova, tj. da li je vrednost promenljive **`f`** manja od vrednosti promenljive **`myvariable`**. Ako jeste, izračunava se kôd u vitičastim zagradama. Nakon toga, interpreter JavaScripta izvršava poslednji parametar naredbe **`for`** i ponovo proverava uslov. Ako je uslov ispunjen, instrukcije se izvršavaju još jednom. Petlja se nastavlja sve dok uslov ne postane neistinit.

Promenljiva **`f`** inicijalizuje se vrednošću **`0`**. Po završetku prvog ciklusa, instrukcija **`f++`** uvećava vrednost promenljive **`f`** za 1, a uslov se ponovo proverava. U svakom ciklusu vrednost promenljive **`f`** povećava se za 1. Proces se nastavlja sve dok promenljiva **`f`** ne postigne vrednost **`9`**, kada uslov postaje neistinit (9 nije manje od 9), i petlja se prekida.

> **Podsetite se osnova** Operator JavaScripta ++ služi i za sabiranje. On vrednosti promenljive dodaje vrednost 1 i rezultat vraća u promenljivu, što se može izraziti kao **`variable = variable + 1`**.

Naredba **`for`** je korisna kada možemo unapred da odredimo uslove, kao što je početna vrednost petlje, ili kako će se te vrednosti kretati. Kada uslovi nisu jasni, možemo koristiti naredbu **`while`**.

Listing 4-14: *Upotreba naredbe* `while`.

```
<script>
  var myvariable = 9;
  while(myvariable < 100) {
    myvariable++;
  }
  alert("The final value of myvariable is " + myvariable);
</script>
```

Naredba **while** zahteva samo deklaraciju uslova, navedenu u zagradama, i kôd koji se izvršava, naveden u vitičastim zagradama. Petlja se izvršava sve dok je uslov istinit. Ako je rezultat prvog izračunavanja uslova **false**, kôd se nikada ne izvršava. Da bi se instrukcije izvršile bar jednom, bez obzira na rezultat uslova, koristi se naredba **do while**.

Listing 4-15: *Upotreba naredbe* `do while`.

```
<script>
  var myvariable = 9;
  do{
    myvariable++;
  }while(myvariable > 100);
  alert("The final value of myvariable is " + myvariable);
</script>
```

U primeru iz listinga 4-15 menjamo uslov u „**myvariable** veće od 100“. Početna vrednost dodeljena promenljivoj bila je 9 pa ovaj uslov nije ispunjen. Ako upotrebimo naredbe **do while**, kôd unutar petlje izvršiće se bar jednom pre izračunavanja uslova. Iskačući prozor na kraju prikazuje vrednost **10** (9 + 1 = 10).

Funkcije i anonimne funkcije

Do sada smo radili u globalnom prostoru. To je glavni prostor u kome se kôd obrađuje u isto vreme kada ga interpreter očitava. Instrukcije se izvršavaju sekvencijalno, po redosledu kojim su navedene u kodu, ali to možemo promeniti upotrebom funkcija.

Funkcija je blok koda koji se identifikuje pomoću imena. Postoji nekoliko prednosti deklarisanja koda unutar funkcije nad deklarisanjem u globalnom prostoru. Najvažnija je to što se funkcije deklarišu ali se ne izvršavaju sve dok se ne pozovu pomoću imena.

Listing 4-16: *Deklarisanje funkcija.*

```
<script>
  function myfunction(){
    alert("I'am a function");
  }
  myfunction();
</script>
```

> **Podsetite se osnova** U JavaScriptu, znakovni niz se može deklarisati pomoću navodnika ili polunavodnika (npr., **"Hello"** ili **'Hello'**). Ako znakovni niz sadrži isti tip navodnika koji se koristi za njegovo deklarisanje, moramo ga napisati uz korišćenje izlazne sekvence, tj. pomoću obrnute kose crte (npr., **'I\'m Tom'**).

Funkcije se deklarišu pomoću rezervisane reči **function**, imena i koda u vitičastim zagradama. Da biste pozvali funkciju (to jest, izvršili je), morate upotrebiti njeno ime i par zagrada na kraju, kao što je prikazano u listingu 4-16.

Kao i u slučaju promenljivih, interpreter JavaScripta učitava funkciju, smešta njen sadržaj u memoriju i dodeljuje referencu imenu funkcije. Kada pozovemo funkciju, interpreter proverava referencu i učitava funkciju iz memorije. To omogućava da pozivamo funkciju više puta, kad god nam zatreba, čime funkcije pretvaramo u moćne jedinice za obradu podataka.

Listing 4-17: *Obrada podataka pomoću funkcija.*

```
<script>
  var myvariable = 5;
  function myfunction(){
    myvariable = myvariable * 2;
  }
  for(var f = 0; f < 10; f++){
    myfunction();
  }
  alert("The value of myvariable is now " + myvariable);
</script>
```

U primeru iz listinga 4-17 kombinuju se različite naredbe koje smo već proučavali. U njemu se deklariše promenljiva i dodeljuje joj se vrednost **5**. Zatim se funkcija, pod imenom **myfunction()**, deklariše, ali se ne izvršava. Naredba **for** definiše petlju koja će se izvršavati dok god je vrednost promenljive **f** manja od 10. Na kraju, poslednja vrednost smeštena u promenljivoj **myvariable** prikazaće se na ekranu.

Kôd unutar petlje poziva funkciju, pa će svaki ciklus petlje izvršavati funkciju **myfunction()**. Ona množi vrednost promenljive **myvariable** sa **2** i smešta rezultat u istu promenljivu. Kad god se funkcija izvrši, vrednost promenljive **myvariable** se uvećava. Iz toga se vidi kako rade funkcije kao jedinice obrade: deklarišemo ih jednom, damo im ime radi referenciranja i nakon toga možemo da ih koristimo kad god poželimo.

Funkciju možemo deklarisati i pomoću anonimne funkcije koja se dodeljuje promenljivoj. Ime promenljive je ono koje kasnije koristimo za pozivanje funkcije. Pomoću ove metode funkcije se mogu definisati unutar drugih funkcija, pretvarati u konstruktore objekata ili se mogu praviti složeni programski obrasci, što ćete naučiti u nastavku.

Listing 4-18: *Deklarisanje funkcija pomoću anonimnih funkcija.*

```
<script>
  var myvariable = 5;
  var myfunction = function(){
    myvariable = myvariable * 2;
```