POGLAVLJE 1

Uvod u programiranje

Dobro došli u knjigu *Naučite JavaScript*.

Deo *Početni koraci* predstavlja uvod u više osnovnih koncepata programiranja – razume se, s posebnim osvrtom na JavaScript (koji ćemo često pisati skraćeno, JS) – i opisuje kako treba razumeti i pristupiti ostalim delovima knjige. Naročito ako tek počinjete da učite programiranje i/ili JavaScript, ovaj deo ukratko opisuje šta treba da znate kako biste napravili svoje *prve korake*.

Knjiga počinje objašnjenjima osnovnih principa programiranja na vrlo visokom nivou. Prvi deo knjige namenjen je onima koji započinju čitanje knjige *Naučite JavaScript* s malo ili nimalo prethodnog iskustva u programiranju, a knjigu su izabrali kao pomoć na putu ka razumevanju programiranja kroz objektiv JavaScripta.

Poglavlje 1 treba shvatiti kao kratak pregled stvari koje ćete morati više da učite i vežbate kako biste zaista *počeli da programirate*. Postoji i mnogo drugih odličnih izvora uvodnih informacija o programiranju koji vam omogućavaju da dalje obrađujete te teme. Preporučujem da učite i iz njih, a ne samo iz ovog poglavlja.

Kada procenite da ste dovoljno dobro savladali opšte osnove programiranja, poglavlje 2 će vam pomoći da savladate stil programiranja kakav je uobičajen u JavaScriptu. Poglavlje 2 vas upoznaje s JavaScriptom, ali ni ono nije sveobuhvatni vodič – čekaju vas i preostali delovi knjige *Naučite JavaScript!*

Ako već prilično dobro poznajete JavaScript, prvo pogledajte poglavlje 3 da biste stekli opštu sliku onoga što možete očekivati od knjige *Naučite JavaScript*, a zatim počnite da je čitate!

Programski kôd

Počnimo od samog početka.

Program, koji se često naziva *izvorni kôd* (engl. *source code*) ili samo *kôd* (engl. *code*) sastoji se od skupa specijalnih naredbi koje nalažu računaru koje poslove treba da obavi. Kôd se najčešće upisuje u tekstualnu datoteku, mada JavaScript omogućava i da kôd kucate direktno u prozor razvojne konzole čitača veba, što ćemo opisati uskoro.

Skup pravila za ispravne formate i kombinacije programskih naredbi zove se *računarski (programski) jezik*, a ponekad i *sintaksa jezika* – slično govornom jeziku čija pravila opisuju kako se reči pravilno pišu i kako se sastavljaju ispravne rečenice pomoću reči i znakova interpunkcije.

Programski iskazi

U programskom jeziku, svaka grupa reči, brojeva i operatora koja obavlja konkretan posao zove se *iskaz* (engl. *statement*). U JavaScriptu, iskaz može izgledati ovako:

```
a = b * 2;
```

Znaci `a` i `b` zovu se *promenljive* (engl. *variables*) (videti odeljak „Promenljive“ na strani 15), koje su nalik običnim kutijama za čuvanje raznih stvari. U programima, promenljive sadrže vrednosti (kao što je broj `42`) s kojima program radi. Zamislite ih kao simbolična mesta rezervisana za čuvanje samih vrednosti.

Nasuprot tome, `2` je samo po sebi vrednost, koja se zove *literal* ili *literalna vrednost* (engl. *literal value*) zato što se pojavljuje samostalno i nije smeštena u promenljivu.

Znaci `=` i `*` su *operatori* (videti odeljak „Operatori“ na strani 10) – oni izvršavaju akcije s vrednostima i promenljivama, kao što je dodeljivanje vrednosti i matematička operacija množenje.

Većina iskaza se u JavaScriptu završava znakom tačka i zarez (`;`).

Iskaz `a = b * 2;` grubo opisano, nalaže računaru da učita tekuću vrednost koja je smeštena u promenljivoj `b`, da zatim tu vrednost pomnoži sa `2`, a onda da dobijeni rezultat uskladišti u drugu promenljivu koju ćemo nazvati `a`.

Programi su samo kolekcije mnogih takvih iskaza, koji svi zajedno opisuju sve korake koje treba izvršiti da bi program obavio ono za šta je namenjen.

Izrazi

Iskazi se sastoje od jednog ili više *izraza* (engl. *expressions*). Izraz čini svaka referenca na promenljivu ili vrednost, ili na skup jedne ili više promenljivih ili literala kombinovanih pomoću operatora.

Na primer, iskaz:

```
a = b * 2;
```

sastoji se od četiri izraza:

- `2` je *izraz s literalnom vrednošću*.
- `b` je *izraz s promenljivom*, što znači da učitava njenu tekuću vrednost.
- `b * 2` je primer *aritmetičkog izraza*, što znači da obavlja operaciju množenja.
- `a = b * 2` je *izraz za dodeljivanje vrednosti*, što znači da rezultat izraza `b * 2` dodeljuje promenljivoj `a` (više reči o izrazima biće u nastavku teksta).

Opšti izraz koji može da se navede samostalno zove se *iskaz izraza*, kao što je sledeći:

```
b * 2;
```

Ta vrsta iskaza nije mnogo uobičajena ni korisna, jer najčešće ne bi uopšte uticala na izvršavanje programa – izraz bi učitao vrednost `b` i pomnožio je sa `2`, ali zatim ne bi ništa učinio s dobijenim rezultatom.

Uobičajeniji iskaz izraza jeste iskaz *pozivnog izraza* (videti odeljak „Funkcije“ na strani 22) jer se ceo iskaz sastoji od izraza pozivanja funkcije:

```
alert( a );
```

Izvršavanje programa

Kako te kolekcije programskih iskaza nalažu programu šta treba da uradi? Programu je potrebno *izvršavanje*, što se zove i *pokretanje* programa.

Iskazi kao što je `a = b * 2` korisni su za programere dok ih čitaju i pišu, ali zapravo nisu u obliku koji računar može direktno da razume. Iz tog razloga, na računaru se koristi specijalna pomoćna alatka (to može biti *interpreter* ili *kompajler*), koja prevodi kôd koji napišete u komande koje računar razume.

Za neke računarske jezike, uobičajeno je da se komande prevode odozgo nadole, red po red, kad god se program izvršava, što se najčešće zove *interpretiranje* koda.

Za druge jezike, prevođenje se obavalja pre pokretanja programa, što se zove *kompajliranje* (ili *prevođenje*) koda; znači, kada se program zatim *pokrene*, izvršavaju se već prevedene i spremne naredbe.

Uobičajena tvrdnja je da je JavaScript *interpretiran* jezik zato što se izvorni JavaScript kôd obrađuje pri svakom izvršavanju. Ali, to nije sasvim tačno. Mašina (engl. *engine*) jezika JavaScript zapravo prvo *kompajlira* a odmah zatim izvršava kompajliran kôd.

Više informacija o kompajliranju JavaScript koda naći ćete u prva dva poglavlja drugog dela knjige – *Opseg vidljivosti i ograde*.

Pokušajte i sami

U ovom poglavlju, svaki programski koncept uvodi se pomoću jednostavnih primera koda, koji su svi napisani na JavaScriptu (razume se!).

Sledeće ne možemo istaći dovoljno jako: dok napredujete kroz ovo poglavlje – a možda će trebati da odvojite vreme kako biste ga prešli i više puta – trebalo bi da uvežbavate sva tri opisana koncepta tako što ćete svojeručno pisati kôd. To ćete najjednostavnije uraditi ako otvorite konzolu s razvojnim alatkama u čitaču veba koji najradije koristite (Firefox, Chrome, IE itd.).

Razvojnu konzolu obično otvarate pomoću odgovarajuće prečice s tastature ili stavke menija. Više informacija o otvaranju i upotrebi razvojne konzole u vašem čitaču veba naći ćete u članku „Upotreba razvojne konzole“ na adresi *http://blog.teamtreehouse.com/mastering-developer-tools-console*.

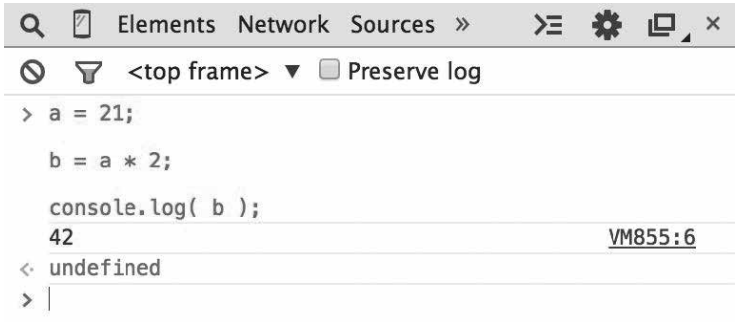
Da biste u konzolu upisali više redova istovremeno, pritisnite `<shift> + <enter>` kako biste prešli u sledeći prazan red. Čim pritisnete samo `<enter>`, konzola će izvršiti sve što ste upravo otkucali.

Razmotrimo sada postupak izvršavanja koda unutar konzole. Prvo, predlažem da otvorite novu praznu karticu u svom čitaču veba. To najradije radim tako što u polje za adresu upišem `about:blank`. Zatim, otvorite svoju razvojnu konzolu na način koji smo upravo opisali.

Upišite sledeći kôd i pogledajte kako radi:

```
a = 21;

b = a * 2;

console.log( b );
```

Ako prethodni kôd upišete u konzolu čitača Chrome, trebalo bi da dobijete nešto slično sledećem:



Slobodno pokušajte sami. Najbolji način da naučite programiranje jeste da počnete da pišete kôd!

Rezultati izvršavanja programa

U prethodnom primeru koda, upotrebili smo naredbu `console.log(..)`. Pogledajmo ukratko suštinu tog reda koda.

Možda ste već i sami pogodili, ali to je tačno način na koji ispisujemo tekst (što se zove i *rezultat izvršavanja programa* namenjen korisniku) na razvojnoj konzoli. Taj iskaz ima dve karakteristike koje bi trebalo objasniti.

Prvo, deo `log( b )` predstavlja pozivanje određene funkcije (videti odeljak „Funkcije“ na strani 22). Ono što se tu događa jeste da promenljivu `b` prosleđujemo navedenoj funkciji, čime od nje zahtevamo da preuzme vrednost `b` i ispiše je na konzoli.

Drugo, deo `console.` je referenca na određeni objekat u kojem se nalazi funkcija `log(..)`. Objekte i njihova svojstva detaljnije razmatramo u poglavlju 2.

Drugi način da dobijete rezultate koje možete videti na ekranu jeste da izvršite iskaz `alert(..)`. Na primer:

```
alert( b );
```

Ako ga izvršite, zapazićete da umesto da rezultate ispisuje na konzoli, taj iskaz otvara iskačući „OK“ prozor sa sadržajem promenljive `b`. Međutim, kodiranje i izvršavanje programa na konzoli lakše ćete savladati ako budete koristili iskaz `console.log(..)` a ne iskaz `alert(..)`, zato što tako možete da prikazujete više vrednosti istovremeno bez remećenja interfejsa čitača veba.

U ovoj knjizi prikazivaćemo rezultate programa pomoću iskaza `console.log(..)`.

Unošenje ulaznih podataka

Dok razmatramo rezultate programa, možda se pitate i šta je sa *ulaznim podacima* za programe (na primer, prihvatanjem podataka od korisnika).

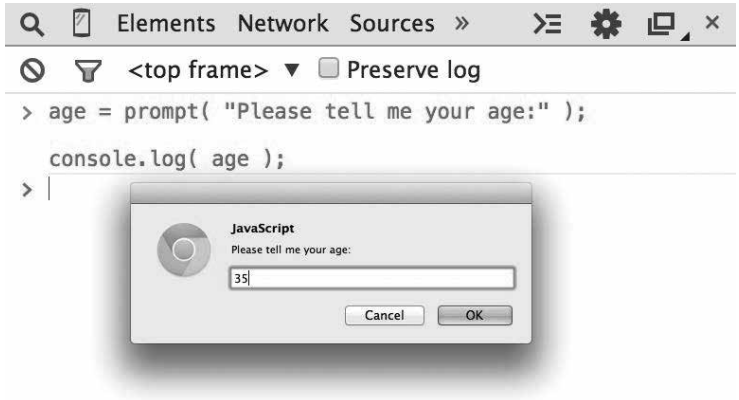
Najuobičajeniji način prijema ulaznih podataka jeste da HTML stranica prikaže određene elemente koji se koriste na obrascima (kao što su polja za unošenje teksta) u koje korisnik može nešto da upiše, a zatim se pomoću JS te vrednosti učitavaju u promenljive programa.

Postoji i jednostavniji način da učitate ulazne podatke kada im je jedina svrha učenje i objašnjavanje nečega, kao u ovoj knjizi. Iskoristite funkciju `prompt(..)`:

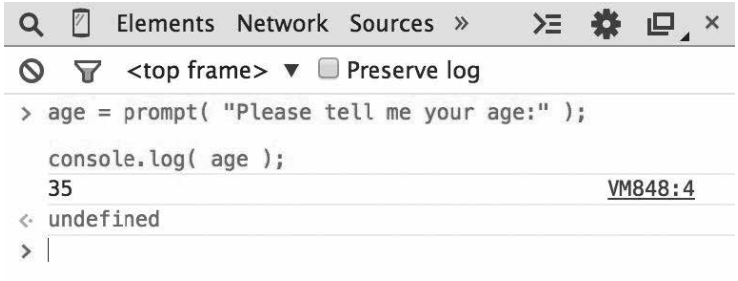
```
age = prompt( "Please tell me your age:" );

console.log( age );
```

Kao što ste možda i sami pretpostavili, sadržaj poruke koju prosledite funkciji `prompt(..)` – u ovom slučaju, `"Please tell me your age:"` (molimo vas da navedete svoje godine) – ispisuje se u iskačućem prozoru.

Trebalo bi da to bude nešto slično ovome:



Pošto pritiskanjem dugmeta „OK“ prosledite programu ulazni tekst, videćete da se vrednost koju ste otkucali smešta u promenljivu `age`, čiji sadržaj zatim *ispisujemo* pomoću funkcije `console.log(..)`:



Da bismo se bavili samo jednostavnim stvarima dok učimo osnovne koncepte programiranja, primeri u ovom odeljku neće zahtevati ulazne podatke. Ali pošto ste sada videli kako se koristi funkcija `prompt(..)`, ako želite da sebi postavite izazov, možete pokušati da koristite i ulazne podatke dok proučavate primere.

Operatori

Operatori određuju akcije koje izvršavamo s promenljivama i vrednostima. Već smo upoznali dva JavaScript operatora, `=` i `*`.

Operator `*` izvršava matematičko množenje. Sasvim jednostavno, zar ne?

Operator jednakosti `=` koristi se za *dodeljivanje vrednosti* – prvo se izračunava vrednost na *desnoj strani* (izvorna vrednost) operatora `=` a zatim se ona smešta u promenljivu koju zadamo na *levoj strani* (ciljna promenljiva).

Ovo možda liči na čudan obrnut redosled dodeljivanja vrednosti. Umesto `a = 42`, neko bi možda radije obrnuo redosled tako da izvorna vrednost bude na levoj strani a ciljna promenljiva na desnoj, kao u izrazu `42 -> a` (ovo nije ispravno u JavaScriptu!). Nažalost, redosled `a = 42`, kao i slične varijacije, preovlađuje u savremenim programskim jezicima. Ako vam izgleda neprirodan, odvojte malo vremena i vežbajte s njime u mislima da biste se navikli.

Razmotrite sledeće:

```
a = 2;
b = a + 1;
```

U ovom primeru dodeljujemo vrednost `2` promenljivoj `a`. Zatim, učitavamo vrednost promenljive `a` (koja je i dalje `2`), dodajemo joj `1` da bismo dobili rezultujuću vrednost `3`, koju zatim smeštamo u promenljivu `b`.

Mada tehnički gledano to nije operator, rezervisana reč `var` će vam trebati u svakom programu jer je to primarni način na koji *deklarišete* (kaže se i *definišete*) promenljive (videti odeljak „Promenljive“ na strani 15).

Trebalo bi da uvek definišete promenljivu po imenu pre nego što je upotrebite. Ali određenu promenljivu treba da definišete samo jedanput u svakom *opsegu vidljivosti* (videti odeljak „Opseg vidljivosti promenljivih“ na strani 23); posle toga je možete koristiti više puta, prema potrebi. Na primer:

```
var a = 20;

a = a + 1;
a = a * 2;
console.log( a )   // 42
```

Sledi nekoliko najuobičajenijih operatora u JavaScriptu:

Dodeljivanje vrednosti
: `=`, kao u `a = 2`.

Matematički
: `+` (sabiranje), `-` (oduzimanje), `*` (množenje) i `/` (deljenje), kao u `a * 3`.

Kombinovani s dodeljivanjem vrednosti
: `+=`, `-=`, `*=` i `/=` složeni su operatori koji kombinuju određenu matematičku operaciju s dodeljivanjem vrednosti, kao u `a += 2` (isto što i `a = a + 2`).

Inkrementiranje/dekrementiranje
: `++` (inkrementiranje), `--` (dekrementiranje), kao u `a++` (isto što i `a = a + 1`).

Pristupanje svojstvu objekta
: `.` kao u `console.log()`.

  Objekti su vrednosti koje sadrže druge vrednosti na određenim imenovanim mestima koja se zovu svojstva (engl. *properties*). `obj.a` predstavlja vrednost objektnog tipa koja se zove `obj` i ima svojstvo čije je ime `a`. Drugi oblik pristupanja svojstvima je `obj["a"]`. Videti poglavlje 2.

Jednakost
: `==` (labava jednakost), `===` (striktna jednakost), `!=` (labava različitost), `!==` (striktna različitost), kao u `a == b`.

  Videti odeljak „Vrednosti i tipovi“ na strani 12 i poglavlje 2.

Poređenje
: `<` (manje od), `>` (veće od), `<=` (manje od ili jednako), `>=` (veće od ili jednako), kao u `a <= b`.

  Videti odeljak „Vrednosti i tipovi“ na strani 12 i poglavlje 2.

Logički
: `&&` (i, konjukcija), `||` (ili, disjunkcija), kao u `a || b` koji bira `a` *ili* `b`.

  Ovi operatori se koriste za izražavanje kombinovanih uslova (videti odeljak „Uslovni iskazi“ na strani 18), kao `a` *ili* `b` je istinito.

Znatno detaljnija objašnjenja i opise operatora koji ovde nisu pomenuti naći ćete na veb stranici Mozilla Developer Network (MDN), u odeljku „Expressions and Operators“.

Vrednosti i tipovi

Ako pitate nekog zaposlenog u prodavnici mobilnih telefona koliko košta određeni uređaj, a on vam kaže „devedeset i devet, devedeset i devet“ (na primer, 99,99 dolara), on vam zapravo saopštava novčani iznos koji treba da platite (plus dažbine) da biste ga kupili. Ukoliko želite da kupute dva takva telefona, lako možete primeniti mentalnu matematiku da biste udvostručili tu vrednost i dobili 199,98 kao osnovnu cenu.

Ukoliko taj isti prodavac odabere neki sličan telefon i kaže da je „besplatan“ (i možda pri tome oponaša prstima znakove navoda), on vam ne saopštava nikakav iznos, nego samo drugačiji oblik predstavljanja očekivane cene ($0,00) – a to je reč „besplatno“.

Kada zatim pitate da li uz telefon dobijate i punjač, odgovor može biti samo „da“ ili „ne“.

Na vrlo slične načine, kada u programu izražavate određene vrednosti, birate različite oblike predstavljanja tih vrednosti u zavisnosti od toga šta nameravate da radite s njima.

Ti različiti oblici predstavljanja vrednosti zovu se *tipovi* (engl. *types*) u terminologiji programiranja. JavaScript ima ugrađene tipove za svaku od tzv. *primitivnih* (osnovnih) vrednosti:

- Kada treba da obavite neku matematičku operaciju, koristite tip `number`.
- Kada treba da ispišete neku vrednost na ekranu, koristite tip `string` (jedan ili više znakova, reči ili rečenica).
- Kada u programu treba da donesete određenu odluku, koristite tip `boolean` (`true` ili `false`).

Vrednosti koje se umeću direktno u izvorni kôd zovu se *literali*. Literali tipa `string` pišu se između navodnika (`"..."`) ili polunavodnika (`'...'`) – jedina razlika je da li više volite jedan ili drugi stil. Literali tipa `number` i `boolean` samo se navode takvi kakvi su (na primer, `42`, `true` itd.).

Razmotrite sledeće:

```
"I am a string";
'I am also a string';

42;

true;
false;
```

Osim vrednosti tipa `string`/`number`/`boolean`, uobičajeno je da programski jezici stavljaju na raspolaganje i *nizove*, *objekte*, *funkcije* i drugo. O vrednostima i tipovima biće znatno više reči dalje u ovom poglavlju, kao i u sledećem.

Konverzije tipova

Ako imate vrednost tipa `number` (broj) ali treba da je prikažete na ekranu, morate je prvo pretvoriti u tip `string`. U JavaScriptu se to pretvaranje zove *konverzija tipa* (engl. *coercion*). Slično tome, ako neko upiše grupu numeričkih znakova na obrazac veb stranice za e-trgovinu, to je tip `string`; ukoliko vam zatim zatreba da tu vrednost upotrebite u matematičkim operacijama, morate je *konvertovati* u tip `number`.

JavaScript ima više mehanizama za bezuslovnu konverziju *tipova*. Na primer:

```
var a = "42";

var b = Number( a );

console.log( a );  // "42"
console.log( b );  // 42
```

Prikazana upotreba funkcije `Number(..)` (ugrađena funkcija) predstavlja *eksplicitnu* konverziju proizvoljnog izvornog tipa u tip `number`. To bi trebalo da je samo po sebi razumljivo.

Ali, šta se događa kada pokušate da poredite dve vrednosti koje izvorno nisu istog tipa, pa bi bila potrebna *implicitna* konverzija tipa?

Kada se poredi znakovni niz `"99.99"` s brojem `99.99`, većina ljudi bi se složila da su te vrednosti ekvivalentne. Ali one nisu sasvim iste, zar ne? To je ista vrednost predstavljena u dva različita oblika, tj. dva različita *tipa*. Moglo bi se reći da su one „labavo jednake“, zar ne?

Da bi vam olakšao rešavanje ovakvih uobičajenih situacija, JavaScript se u određenim slučajevima sam umeša i *implicitno* pretvara vrednosti u odgovarajuće ciljne tipove.

Tako da ako upotrebite operator labave jednakosti `==` da biste uporedili `"99.99" == 99.99`, JavaScript će konvertovati levu stranu `"99.99"` u njen ekvivalent tipa `number`, što je `99.99`. Poređenje se zatim svodi na `99.99 == 99.99`, čiji rezultat je, razume se, `true`.

Uprkos tome što je osmišljena da pomogne, implicitna konverzija tipova može da vas zbuni ako niste prethodno odvojili vreme i naučili pravila koja pri tome važe. Pošto većina JS programera to ne čini, uobičajeno osećanje je da konverzija tipova zbunjuje i uvodi u programe neočekivane greške, pa bi je zato trebalo izbegavati. Ponekad se čak naziva i greškom u dizajnu jezika.

Međutim, implicitna konverzija je mehanizam koji *se može naučiti*, štaviše, *treba da ga nauči* svako ko namerava da se ozbiljno bavi programiranjem na JavaScriptu. Ne samo da više ne zbunjuje kada savladate njegova pravila, nego može zapravo i poboljšati vaše programe! Uloženi trud je vredan rezultata.

Više informacija o konverziji tipova naći ćete u poglavlju 2 ovog dela knjige i u poglavlju 4 četvrtog dela – *Tipovi i gramatika*.

Komentari u kodu

Prodavac mobilnih telefona može da napiše nekoliko beležaka o odlikama novog modela telefona u ponudi ili novim planovima firme za buduće ponude. Te napomene su namenjene samo zaposlenima – njih ne treba da čitaju kupci. Uprkos tome, te beleške omogućavaju zaposlenima da lakše obavljaju svoje poslove jer je bolje dokumentovano šta i zbog čega treba da pričaju kupcima.

Jedna od najvažnijih lekcija koje možete naučiti u vezi s pisanjem koda jeste da on nije namenjen samo računaru. Programski kôd je u svakom svom deliću namenjen isto toliko, ako ne i više, programeru koliko i kompajleru.

Računaru je bitan samo mašinski kôd, što je niz binarnih jedinica i nula, koji se dobija nakon *kompajliranja* (*prevođenja*) izvornog koda. Postoji gotovo beskonačan broj programa koji mogu rezultovati istim nizom nula i jedinica. Odluke koje donosite u vezi s načinom na koji pišete program imaju svoju težinu – ne samo za vas, nego i za druge članove razvojnog tima, pa čak i za vas ubuduće.

Trebalo bi da pišete programe koji ne samo što rade ispravno, nego i izgledaju razumljivo kada ih neko proučava. Tome možete mnogo doprineti ako birate dobra imena za promenljive (videti odeljak „Promenljive“ na strani 15) i funkcije (videti odeljak „Funkcije“ na strani 22).

Drugi važan deo su komentari u kodu. To su kratki tekstovi koji se dodaju u program isključivo kao objašnjenja za ljude koji čitaju kôd (uključujući i vas, kao programera). Interpreter/kompajler uvek zanemaruje te komentare.

Postoje razna mišljenja o tome šta čini dobro dokumentovan kôd; nije moguće definisati apsolutna i univerzalna pravila. Ipak, pojedine napomene i smernice prilično su korisne:

- Programski kôd bez komentara nikad nije optimalan.
- Previše komentara (na primer, po jedan u svakom redu) verovatno je znak loše napisanog koda.
- Trebalo bi da komentari objašnjavaju *zašto*, a ne *šta*. Mogu da objašnjavaju i *kako* ako je ono što je napisano posebno zbunjujuće.

U JavaScriptu, moguće su dve vrste komentara: jednoredni i višeredni.
Razmotrite sledeće:

```
// Ovo je jednoredni komentar

/* a ovo je
     višeredni
          komentar.
                    */
```

Jednoredni komentar (označen znakovima //) pogodan je kada nameravate da dodate komentar neposredno iznad jednog iskaza ili čak na kraj reda. Sve u redu iza znakova //, do kraja reda, smatra se komentarom (i kompajler ga zato zanemaruje). Nema nikakvog ograničenja po pitanju toga šta možete napisati u sklopu jednorednog komentara.

Razmotrite sledeće:

```
var a = 42;    // 42 je smisao života
```

Višeredni komentar (označen znakovima `/* .. */`) pogodan je ako imate objašnjenje koje zauzima više redova.

Evo uobičajenog oblika upotrebe višerednih komentara:

```
/* Koristi se sledeća vrednost zato što
   je dokazano da ona odgovara na svako
   pitanje postavljeno u univerzumu. */
var a = 42;
```

Višeredni komentar se može umetnuti na proizvoljno mesto u redu, čak i usred reda, zato što se završava znakovima `*/`. Na primer:

```
var a = /* proizvoljna vrednost */ 42;

console.log( a ); // 42
```

Jedini element koji se ne može pojaviti u višerednom komentaru jeste grupa znakova `*/`, zato što bi se to tumačilo kao kraj komentara.

Trebalo bi svakako da svoje učenje programiranja počnete navikom da komentarišete programski kôd. Pošto ćete u celom preostalom delu ovog poglavlja videti da koristim komentare kako bih objasnio određene stvari, činite to isto u svom kodu. Verujte mi, biće vam zahvalan svako ko čita vaš kôd!

Promenljive

U većini programa koji rade nešto korisno, potrebno je pratiti određenu vrednost koja se menja tokom izvršavanja programa jer učestvuje u više operacija potrebnih za poslove koje program obavlja.

U svom programu, to ćete najlakše obezbediti ako tu vrednost dodelite određenom simboličkom kontejneru – koji se zove *promenljiva* (engl. *variable*) – zato što se vrednost u tom kontejneru može tokom vremena *menjati*, prema potrebama.

U nekim programskim jezicima, promenljivu (kontejner) deklarišete tako da uvek sadrži samo određen tip vrednosti – recimo, `number` ili `string`. *Statičko određivanje tipa*, poznato i kao *nametanje tipa*, uglavnom se navodi kao prednost kad je reč o ispravnosti programa, zato što sprečava nenamerne konverzije tipova.

U drugim programskim jezicima, tipovi vrednosti smatraju se važnijim od tipova promenljivih. *Labavo određivanje tipa*, što je poznato i kao *dinamičko određivanje tipa*, omogućava da jedna promenljiva može u svakom datom trenutku sadržati vrednost proizvoljnog tipa. To se uglavnom navodi kao prednost kad je reč o fleksibilnosti programa, jer omogućava da ista promenljiva predstavlja vrednost bez obzira na oblik u kojem se ta vrednost može naći u svakom datom trenutku logičkog toka programa.

JavaScript koristi ovaj drugi pristup, *dinamičko određivanje tipa*, što znači da promenljive mogu da sadrže vrednosti proizvoljnog *tipa*, bez ikakvog nametanja *tipa*.