UVOD

Ovo je knjiga o korišćenju računara. Računari su danas postali uobičajeni kao i šrafcigeri, ali su poprilično složeniji i nije uvek lako naterati ih da rade ono što vi hoćete.

Ukoliko je zadatak koji imate za računar uobičajen, dobro shvaćen, kao što je prikazivanje e-poruke ili obavljanje posla kalkulatora, možete otvoriti odgovarajuću aplikaciju i baciti se na posao. Ali za jedinstvene ili neodređene zadatke, verovatno ne postoji aplikacija.

Tu u igru ulazi programiranje. *Programiranje* je čin konstruisanja *programa* – skupa preciznih instrukcija koje govore računaru šta da radi. Pošto su računari glupi ali i pedantni, programiranje je u svojoj osnovi naporno i frustrirajuće.

Srećom, ako možete da prevaziđete tu činjenicu, a možda i uživate u krutom razmišljanju u pojmovima s kojima glupe mašine mogu da se izbore, programiranje vam može doneti zadovoljstvo. Ono vam omogućava da u sekundama obavite stvari koje biste ručno radili *beskrajno*. Ono je način da od svog računara napravite alat koji će raditi stvari koje ranije nije mogao da uradi. I odlična je vežba za apstraktno razmišljanje.

Većina programiranja obavlja se pomoću programskih jezika. *Programski jezik* je veštački konstruisan jezik koji se koristi za zadavanje instrukcija računarima. Zanimljivo je da najefikasniji način koji smo pronašli za komuniciranje s računarom uveliko preuzima elemente iz načina na koje komuniciramo

međusobno. Kao i govorni jezici, računarski jezici omogućavaju da se reči i fraze kombinuju na nove načine, zahvaljujući čemu je moguće izraziti nove koncepte.

U jednom periodu, interfejsi zasnovani na jeziku, kao što su komandni odzivnici u BASIC-u i DOS-u iz osamdesetih i devedesetih godina prošlog veka, bili su glavni način interakcije s računarima. Oni su uveliko zamenjeni vizuelnim interfejsima kojima se lakše ovladava, ali nude manje slobode. Računarski jezici su i dalje tu, ako znate gde da ih pronađete. Jedan takav jezik, JavaScript, ugrađen je u svaki moderan čitač veba i stoga je dostupan na gotovo svakom uređaju.

Ova knjiga će pokušati da vas upozna s tim jezikom do mere da sa njim možete da radite korisne i zabavne stvari.

O programiranju

Pored objašnjavanja JavaScripta, predstaviću vam i osnovne principe programiranja. Kako se ispostavilo, programiranje je teško. Osnovna pravila su jednostavna i jasna, ali programi izgrađeni na tim pravilima obično postaju dovoljno kompleksni da uvode sopstvena pravila i složenost. Moglo bi se reći da gradite svoj lavirint, i mogli biste se izgubiti u njemu.

Dešavaće se da vam čitanje ove knjige deluje užasno frustrirajuće. Ukoliko ste početnik u programiranju, biće mnogo novog materijala koji treba da svarite. Mnogo tog materijala će se onda *kombinovati* na načine koji zahtevaju da pravite dodatne veze.

Na vama je da načinite potreban napor. Kada se budete mučili s praćenjem knjige, nemojte donositi preuranjen sud o sopstvenim sposobnostima. S vama je sve u redu – samo treba da budete uporni. Napravite pauzu, pročitajte ponovo materiju i pobrinite se da pročitate i razumete primere programa i vežbe. Učenje je naporan posao, ali sve što naučite biće vaše i učiniće kasnije učenje lakšim.

> Kada akcija postane neisplativa, prikupljajte informacije;
> Kada informacije postanu neisplative, spavajte.
> – Ursula K. Le Guin, *The Left Hand of Darkness*

Program je više od programa. To je deo teksta koji je napisao programer, to je upravljajuća sila koja tera računar da radi ono što radi, to su podaci u memoriji računara, ali upravljaju akcijama koje se izvode na toj istoj memoriji. Analogije koje pokušavaju da uporede programe sa objektima koji su nam poznati obično nisu adekvatne. Površno odgovarajuća je ona za mašinu – prisutno je mnogo odvojenih delova, a da bi cela stvar radila, moramo da uzmemo u obzir načine na koje su ti delovi povezani i na koje doprinose funkcionisanju celine.

Računar je fizička mašina koja služi kao domaćin za te nematerijalne mašine. Računari sami mogu da obavljaju samo glupo jednostavne stvari. Razlog zbog kog su tako korisni je što te stvari obavljaju neverovatno velikom brzinom. Program može genijalno da kombinuje ogromne brojeve prostih akcija da bi uradio veoma složene stvari.

Program je zgrada misli. Njena izgradnja ne košta ništa, ona nema težinu i lako raste pod našim prstima koji kuckaju.

Ali ako ne vodimo računa, veličina i složenost programa izmaći će kontroli, zbuniće čak i osobu koja ga je napravila. Držanje programa pod kontrolom glavni je problem u programiranju. Kada program radi, to je prelepo. Umetnost programiranja je veština upravljanja složenošću. Odličan program je potčinjen – pojednostavljen je u svojoj složenosti.

Neki programeri veruju da je tu složenost najbolje kontrolisati korišćenjem samo malog skupa dobro shvaćenih tehnika u programima koje pišu. Oni su sastavili stroga pravila („najbolje prakse“) koja propisuju formu koju program treba da ima i pažljivo ostaju unutar svoje komforne zone.

To ne samo da je dosadno, već je i neefikasno. Novi problemi često zahtevaju nova rešenja. Polje programiranja je mlado i još uvek se brzo razvija, i dovoljno je raznovrsno da ima prostora za sasvim različite pristupe. Postoji mnogo užasnih grešaka koje se mogu napraviti u dizajnu programa i ne treba da se ustežete da ih pravite, jer ćete ih tako razumeti. Osećaj za izgled dobrog programa razvija se praksom, ne učenjem liste pravila.

Zašto je jezik bitan

U vreme nastanka računara nisu postojali programski jezici. Programi su izgledali otprilike ovako:

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

To je program za sabiranje brojeva od 1 do 10 i prikazivanje rezultata: 1 + 2 + ... + 10 = 55. Mogao je da se izvede na jednostavnoj, hipotetičkoj mašini. Da bi se programirali rani računari, bilo je neophodno postaviti velike nizove prekidača u odgovarajući položaj ili bušiti rupe u kartonskim trakama i ubacivati ih u računar. Možete zamisliti koliko je ta procedura bila naporna i podložna greškama. Čak je i pisanje jednostavnih programa zahtevalo mnogo sposobnosti i discipline. Oni složeni su bili skoro nezamislivi.

Naravno, ručno upisivanje tih nedokučivih obrazaca bitova (jedinica i nula) davali su programeru duboki osećaj da je moćni čarobnjak. I to sigurno znači nešto kad je u pitanju zadovoljstvo poslom.

Svaki red prethodnog programa sadrži jednu instrukciju. Na srpskom bi se to moglo napisati ovako:

1. Uskladišti broj 0 na memorijsko mesto 0.
2. Uskladišti broj 1 na memorijsko mesto 1.
3. Uskladišti vrednost memorijskog mesta 1 na memorijsko mesto 2.

4. Oduzmi broj 11 od vrednosti na memorijskom mestu 2.
5. Ako je vrednost na memorijskom mestu 2 broj 0, nastavi sa instrukcijom 9.
6. Dodaj vrednost memorijskog mesta 1 memorijskom mestu 0.
7. Dodaj broj 1 vrednosti memorijskog mesta 1.
8. Nastavi sa instrukcijom 3.
9. Prikaži vrednost na memorijskom mestu 0.

Iako je ovo već čitljivije od bućkuriša bitova, još uvek je krajnje nejasno. Korišćenje imena umesto brojeva za instrukcije i memorijska mesta pomaže.

```
 Podesi "total" na 0.
 Podesi "count" na 1.
[petlja]
 Podesi "compare" na "count".
 Oduzmi 11 od "compare".
 Ako je "compare" nula, nastavi od [kraj].
 Dodaj "count" na "total".
 Dodaj 1 na "count".
 Nastavi od [petlja].
[kraj]
 Prikaži "total".
```

Vidite li kako program sada radi? Prva dva reda daju dvama memorijskim mestima početne vrednosti: total će se koristiti da bi se izgradio rezultat izračunavanja, a `count` će pratiti broj koji trenutno geldamo. Redovi koji koriste `compare` verovatno su najčudniji. Program želi da vidi da li je `count` jednako 11 da bi odlučio da li može prestati s radom. Pošto je naša hipotetička mašina krajnje primitivna, ona može da testira samo da li je neki broj nula i da donosi odluku na osnovu toga. Zbog toga koristi memorijsko mesto označeno sa `compare` da bi izračunala vrednost `count` – 11 i donela odluku na osnovu te vrednosti. Naredna dva reda dodaju vrednost `count` rezultatu i povećavaju `count` za 1 svaki put kada program odluči da `count` još uvek nije 11.

Evo istog programa u JavaScriptu:

```
let total = 0, count = 1;
while (count <= 10) {
  total += count;
  count += 1;
}
console.log(total);
// ! 55
```

Ova verzija daje više poboljšanja. Najbitnije od svega, više nema potrebe da zadajemo način na koji želimo da program skače napred nazad. Konstrukcija `while` se brine za to. Ona nastavlja da izvršava blok (obuhvaćen vitičastim zagradama) ispod nje sve dok je uslov koji joj je dat tačan. Taj uslov je `count` <= 10, što znači „*count* je manje ili jednako 10“. Više ne moramo da pravimo privremenu vrednost i poredimo je sa nulom, što je bio samo nezanimljiv detalj. Deo moći programskih jezika jeste u tome što mogu da se pobrinu za nezanimljive detalje.

Na kraju pograma, kada se konstrukcija `while` završi, operacija `console.log` se koristi za ispisivanje rezultata.

Najzad, evo kako bi program izgledao ako bi nam bile dostupne zgodne operacije `range` i `sum`, od kojih prva pravi kolekciju brojeva u nekom opsegu, a druga izračunava zbir kolekcije brojeva:

```
console.log(sum(range(1, 10)));
// → 55
```

Pouka ove priče je da se isti program može prikazati i naširoko i ukratko, na nejasan i na jasan način. Prva verzija programa je bila ekstremno nejasna, dok je poslednja gotovo ista kao engleski jezik: `log` (evidentiraj) `sum` (zbir) `range` (opsega) brojeva od 1 do 10. (U kasnijim poglavljima ćemo videti kako se definišu operacije kao što su `sum` i `range`.)

Dobar programski jezik pomaže programeru omogućavajući mu da na višem nivou govori o akcijama koje računar treba da izvrši. To pomaže da se izostave detalji, obezbeđuje zgodne gradivne blokove (kao što su `while` i `console.log`), dozvoljava vam da definišete sopstvene gradivne blokove (kao što su `sum` i `range`), i čini lakim uklapanje tih blokova.

Šta je JavaScript?

JavaScript je predstavljen 1995. kao način da se programi dodaju veb stranama u čitaču Netscape Navigator. Taj jezik je u međuvremenu prihvaćen u svim glavnim grafičkim čitačima veba. On je učinio mogućim moderne veb aplikacije – aplikacije s kojima možete imati direktne interakcije, ne morajući ponovo da učitavate stranu za svaku akciju. JavaScript se koristi i na tradicionalnijim veb prezentacijama kako bi se omogućili razni vidovi interaktivnosti i sposobnosti.

Bitno je napomenuti da JavaScript nema skoro ništa sa programskim jezikom Java. Slično ime je bilo inspirisano marketinškim razlozima, pre nego dobrim rasuđivanjem. Kada je JavaScript uveden, jezik Java je bio obilno reklamiran i sticao je popularnost. Neko je pomislio da je dobra ideja ukačiti se na taj uspeh. I sada smo zaglavili sa ovim imenom.

Nekon njegovog prihvatanja van Netscapea, napisan je standardan dokument koji opisuje način na koji bi jezik JavaScript trebalo da radi da bi razni delovi softvera koji su tvrdili da podržavaju JavaScript zaista govorili o istom jeziku. On je nazvan ECMAScript standard, po organizaciji Ecma International koja je obavila standardizaciju. U praksi, pojmovi ECMAScript i JavaScript mogu se koristiti kao sinonimi – to su dva imena za isti jezik.

Ima onih koji će raći *užasne* stvari o JavaScriptu. Mnoge od tih stvari su tačne. Kada sam prvi put morao da napišem nešto u JavaScriptu, brzo sam počeo da ga prezirem. Prihvatao je gotovo sve što bih upisao, ali je to tumačio na način koji je potpuno različit od onoga što sam ja mislio. Naravno, razlog za to uveliko leži u činjenici da nisam imao pojma šta radim, ali tu postoji i jedan pravi problem: JavaScript je suludo liberalan u onome što

dozvoljava. Zamisao iza takvog dizajna bila je da će on učiniti programiranje na JavaScriptu jednostavnijim za početnike. Zapravo, on gotovo da otežava pronalaženje problema u programima jer vam ih sistem neće pokazati.

Ova fleksibilnost, ipak, ima i svoje prednosti. Ona ostavlja prostor za mnogo tehnika koje su nemoguće u krućim jezicima, a kao što ćete videti (na primer, u poglavlju 10), ona se može upotrebiti i za prevazilaženje nekih nedostataka JavaScripta. Kada sam ispravno naučio jezik i neko vreme radio s njim, naučio sam da *volim* JavaScript.

Postoji nekoliko verzija JavaScripta. ECMAScriptova verzija 3 je bila široko podržana verzija u vreme kada je JavaScript počinjao da dominira, otprilike između 2000. i 2010. Tokom tog vremena, u toku je bio rad na ambicioznoj verziji 4, za koju je planirano više radikalnih poboljšanja i proširenja jezika. Menjanje živog, naširoko korišćenog jezika na tako radikalan način pokazalo se kao politički teško i rad na verziji 4 je obustavljen 2008, vodeći ka znatno manje ambicioznoj verziji 5. Objavljena 2009, verzija 5 je donela samo neka nekontroverzna poboljšanja. Potom je, 2015, izašla verzija 6, krupna izmena koja je uključivala neke od ideja planiranih za verziju 4. Nakon toga, imali smo nove, male dopune svake godine.

Činjenica da se jezik razvija znači da čitači moraju neprekidno da ostanu u koraku s njim, a ako koristite stariji čitač, on možda neće podržavati svaku mogućnost. Dizajneri jezika paze da ne naprave izmene koje bi zaustavile postojeće programe, pa novi čitači i dalje mogu da izvršavaju stare programe. U ovoj knjizi koristim verziju 2017 JavaScripta.

Veb čitači nisu jedine platforme na kojima se koristi JavaScript. Neke baze podataka, kao što su MongoDB i CouchDB, koriste JavaScript kao svoj jezik za izradu skriptova i upita. Nekoliko platformi za desktop i serversko programiranje, od kojih je najvažniji projekat Node.js (tema poglavlja 20), nude okruženje za programiranje na JavaScriptu van čitača.

Kôd, i šta ćete s njim

Kôd je tekst koji čini programe. Većina poglavlja u ovoj knjizi sadrži podosta koda. Verujem da su čitanje koda i pisanje koda nezamenljivi delovi učenja programiranja. Pokušajte da ne prelazite ovlaš preko primera – pažljivo ih pročitajte i shvatite. To je možda sporo i zbunjujuće na početku, ali obećavam da ćete ga brzo savladati. Isto važi i za vežbe. Nemojte pretpostaviti da ih razumete dok ne napišete rešenje koje radi.

Preporučujem da isprobate svoja rešenja za vežbe u pravom JavaScript interpreteru. Na taj način ćete dobiti trenutnu povratnu informaciju o tome da li ono što radite funkcioniše i, nadam se, doći ćete u iskušenje da eksperimentišete i odete i dalje od vežbi.

Najjednostavniji način da pokrenete probni kod iz knjige, i da eksperimentišete s njim, jeste da ga pogledate u internet verziji knjige na lokaciji *https://eloquentjavascript.net* . Tamo možete da pritisnete bilo koji primer koda da biste ga izmenili i pokrenuli, i videli rezultat koji daje. Da biste radili na vežbama, idite na lokaciju *https://eloquentjavascript.net/code*, koja sadrži početni kod za svaku vežbu u programiranju i omogućava da pogledate rešenja.

Ako želite da pokrenete programe definisane u knjizi izvan veb lokacije knjige, moraćete da obratite pažnju. Mnogi programi su samostalni i trebalo bi da rade u bilo kom JavaScript okruženju. Međutim, kod u kasnijim poglavljima je često napisan za određeno okruženje (čitač veba ili Node.js) i može se pokrenuti samo tamo. Pored toga, mnoga poglavlja definišu veće programe, pa delovi koda koji se pojavljuju u njima zavise jedni od drugih ili od spoljnih datoteka. Izolovano okruženje (engl. *sandbox)* na veb lokaciji nudi hiperveze do Zip datoteka koje sadrže sve skriptove i datoteke s podacima potrebne da bi se izvršio kod za dato poglavlje.

Pregled knjige

Ova knjiga ima tri dela. Prvih 12 poglavlja govore o jeziku JavaScript. Narednih sedam poglavlja govore o veb čitačima i načinu na koji se JavaScript koristi za njihovo programiranje. Najzad, dva poglavlja za temu imaju Node.js, još jedno okruženje za programiranje na JavaScriptu.

U celoj knjizi nalazi se pet *projektnih poglavlja*, koja opisuju veće primere programa da biste okusili pravo programiranje. Redom, radićemo na izgradnji robota za isporuku, programskom jeziku, platformeru, programu za bojenje piksela i dinamičkoj veb prezentaciji.

Jezički deo knjige počinje sa četiri poglavlja koja prikazuju osnovnu strukturu jezika JavaScript. Ona predstavljaju kontrolne strukture (kao što je reč `while` koju ste videli u ovom uvodu), funkcije (pisanje vaših gradivnih blokova) i strukture podataka. Nakon njih, moći ćete da pišete osnovne programe. Potom, u poglavljima 5 i 6, predstavljamo tehnike korišćenja funkcija i objekata za pisanje *apstraktnijeg* koda i za obuzdavanje složenosti.

Nakon prvog projektnog poglavlja, jezički deo knjige nastavlja se poglavljima o radu sa greškama i ispravljanju grešaka, regularnim izrazima (oni su bitan alat za rad sa tekstom), modularnosti (još jedna odbrana od složenosti) i asinhronom programiranju (radu sa događajima koji zahtevaju vreme). Drugo projektno poglavlje završava prvi deo knjige.

Drugi deo, poglavlja 13 do 19, opisuje alate kojima JavaScript u čitaču ima pristup. Naučićete kako da prikazujete stvari na ekranu (poglavlja 14 i 17), kako da reagujete na korisnički unos (poglavlje 15) i kako da komunicirate putem mreže (poglavlje 18). I u ovom delu postoje dva projektna poglavlja.

Nakon toga, poglavlje 20 opisuje Node.js, a poglavlje 21 pravi malu veb prezentaciju koristeći taj alat.

Na kraju, poglavlje 22 opisuje neke okolnosti koje se pojavljuju pri optimizovanju JavaScript programa da bi bili brži.

Tipografska pravila

U ovoj knjizi, tekst napisan neproporcionalnim fontom predstavljaće elemente programa – to su ponekad kompletni delovi, a ponekad samo ukazuju na deo obližnjeg programa. Programi (a već ste ih videli nekoliko) napisani su na sledeći način:

```
function factorial(n) {
if (n == 0) {
return 1;
} else {
return factorial(n - 1) * n;
}
}
```

Ponekad, da biste videli rezultat koji program daje, očekivani rezultat biće napisan nakon programa, nakon dve kose crte i strelice.

```
console.log(factorial(8));
// → 40320
```

Srećno!