JavaScript

funkcionalno programiranje

II izdanje

Federico Kereki

Napišite čist, robustan i održiv kod za veb i server pomoću funkcionalnog JavaScripta

kompjuter biblioteka

Packt>

Prevod II izdanja

JavaScript

funkcionalno programiranje

Napišite čist, robustan i održiv kod za veb i server pomoću funkcionalnog JavaScripta

Federico Kereki

kompjuter biblioteka

Packt›

Izdavač:

kompjuter biblioteka

Obalskih radnika 4a, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Federico Kereki

Prevod: Biljana Tešić

Lektura: Miloš Jevtović

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2020.

Broj knjige: 527

Izdanje: Prvo

ISBN: 978-86-7310-550-5

Mastering JavaScript

Functional Programming

Second Edition

Federico Kereki

ISBN 978-1-83921-306-9

Pisanje knjige uključuje mnogo ljudi. Ne mogu ih sve pomenuti i imenovati, ali postoje neki koji zaista zaslužuju da im se posebno zahvalim.

Zahvaljujem se Larissi Pinto, glavnoj urednici iz izdavačke kuće „Packt Publishing", koja je predložila temu ove knjige i pomogla mi da započnem pisanje. Takođe izražavam zahvalnost Mohammedu Yusufu Imaratwaleu, uredniku za razvoj sadržaja, i Ralphu Rosariou, tehničkom uredniku, za pomoć u oblikovanju i jasnijem i boljem strukturiranju ove knjige. Takođe da se zahvaljujem recenzentima Geronimou Garciji Sgrittaiju i Steveu Perkinsu, koji su pregledali početni nacrt knjige, poboljšavajući ga svojim komentarima.

Postoje još neki ljudi koji zaslužuju dodatnu pažnju. Ovu knjigu sam napisao u neobičnim okolnostima, oko 10.000 milja od kuće! Otišao sam iz Urugvaja, u kojem živim, da bih učestvovao u realizaciji jednog projekta u Indiji i tamo sam napisao svaku stranicu ove knjige. To ne bi bilo moguće da nisam imao potpunu podršku porodice, koja je ostala u Montevideu, ali sa kojom sam bio neprekidno u bliskoj vezi, zahvaljujući Internetu i sredstvima moderne komunikacije. Konkretno, moram izdvojiti moju suprugu Silviju Tosar, ne samo zbog podrške i pomoći u vezi sa projektom i knjigom, već i zato što mi je pomogla u svemu, a zahvaljujem se i ostatku njene porodice u Urugvaju. Ovu knjigu ne bi bilo moguće napisati bez podrške moje supruge koja je najzaslužnija što sam je i napisao.

Za drugo izdanje

Revidiranje i proširenje knjige za drugo izdanje su teški i zanimljivi zadaci. Imao sam veliku podršku izdavačke kuće „Packt" i moram sada da se zahvalim Aamiru Ahmedu, uredniku za razvoj sadržaja, Janei D'souzuji, tehničkoj urednici, i Crystianu Biettiu i (ponovo, za duple zasluge!) Steveu Perkinsu, recenzentima, koji su omogućili da napišem mnogo bolji tekst.

Federico Kereki

O AUTORU

Federico Kereki je urugvajski sistemski inženjer, koji ima diplomu magistra obrazovanja i više od 30 godina iskustva kao konsultant, programer sistema, univerzitetski profesor i pisac.

Trenutno je stručni saradnik u kompaniji „Globant“, u kojoj mora da koristi dobru kombinaciju radnih okvira, programskih alatki i operativnih sistema, kao što su JavaScript, Node.js, React Redux, SOA, Containers i PHP u Windowsu i Linuxu.

Održao je nekoliko kurseva informatike na univerzitetima Universidad de la Republica, Universidad ORT Uruguay i Universidad de la Empresa. Napisao je i tekstove za te kurseve.

Napisao je nekoliko članaka o JavaScriptu, razvoju veb stranica i otvorenom kodu za časopise, kao što su „Linux Journal“ i „LinuxPro Magazine“ u Sjedinjenim Američkim Državama i „Linux+“ i „Mundo Linux“ u Evropi, i za veb sajtove, kao što su Linux.com i IBM Developer Works. Napisao je i brošure o računarskoj bezbednosti („Linux in the Time ofMalwareand SSH: A Modern Lock for Your Server“), knjigu o GWT programiranju („Essential GWT: Buildingforthe Web with Google Web Toolkit“) i još jednu knjigu o JavaScript razvoju („Modern JavaScript Web Development Cookbook“).

Federico je održao govore o funkcionalnom programiranju pomoću JavaScripta na javnim konferencijama, kao što su „JSCONF 2016“ i „Week Week Santiago 2019“, i koristio je te tehnike za razvoj internet sistema za preduzeća u Urugvaju i inostranstvu.

Njegova trenutna interesovanja su usmerena na kvalitet softvera i softverski inženjering najviše pomoću agilnih metodologija, dok u praksi on koristi različite programske jezike, alatke i radne okvire, ali i softver otvorenog koda (FLOSS) gde god je to moguće.

Obično boravi, radi i predaje u Urugvaju, ali ovu knjigu je napisao dok je kreirao jedan projekat u Indiji, a revizije za drugo izdanje završene su tokom boravka u Meksiku.

O RECENZENTIMA

Steve Perkins je autor knjige „Hibernate Search by Example“. Koristi Javu i JavaScript od kraja devedesetih godina prošlog veka, a ušao je u „svet“ programskih jezika Scala, Groovy i Go. Živi u Atlanti, u Džordžiji, sa suprugom i dvoje dece. Trenutno je softverski arhitekta u kompaniji „Banian Hills Technologies“, u kojoj radi na platformi za upravljanje IoT uređajima i za analitiku.

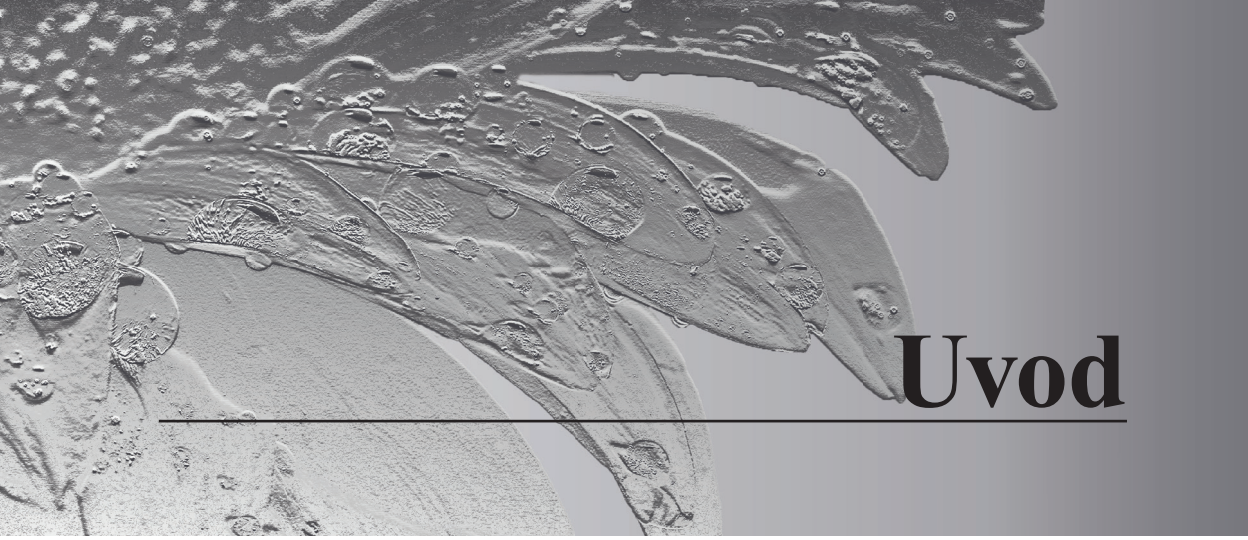
Kada ne piše kodove ili ne provodi vreme sa porodicom, Steve svira gitaru i gubi u igrama Bridge and Backgammon. Možete da posetite njegov tehnički blog na stranici steveperkins.com i da ga pratite na Twitteru na adresi @stevedperkins.

Cristian „Pusher“ Bietti je preduzetnik koji je proaktivan i ima kreativan stav pri suočavanju sa izazovima u novim tehnologijama, uz veliku želju da uvek sazna što više. On je viši programer, sa više od 18 godina iskustva u razvoju i dizajniranju softvera, a obučen je za širok spektar tehnologija i učestvovao je u realizaciji velikih bankarskih projekata i kreiranju malih aplikacija za mobilne telefone i društvene mreže, uključujući video-igre.

Fokusirao se na frontend i korisničko iskustvo (UI/UX). On je stručnjak za određene oblasti u kompaniji „Globant“, a radi u „Fintech“ industriji kao tehnički vođa i programer, jer voli da piše kodove.

„PACKT“ TRAŽI AUTORE KAO ŠTO STE VI

Ako ste zainteresovani da postanete autor za „Packt“, prijavite se na stranicu authors.packtpub.com. Sarađujemo sa hiljadama programera i tehničkih profesionalaca da bismo im pomogli da podele svoje mišljenje sa globalnom tehničkom zajednicom. Možete da podnesete osnovnu prijavu, da se prijavite za specifičnu temu za koju tražimo autore ili da pošaljete neke svoje ideje.



Uvod

U računarskom programiranju paradigmi ima u izobilju. Neki primeri uključuju imperativno programiranje, strukturirano programiranje (*bez goto instrukcija*), **objektno-orijentisano programiranje (OOP)**, aspektno-orijentisano programiranje i deklarativno programiranje. U poslednje vreme se pojavilo interesovanje za određenu paradigmu za koju se može pretpostaviti da je starija od većine (ako ne i svih) pomenutih paradigmi – za **funkcionalno programiranje (FP)**. U FP-u naglasak je na funkcije pisanja i njihovo povezivanje na jednostavne načine da bi bio proizveden što razumljiviji kod i kod koji se može lakše testirati.

Ovaj interes za FP pojačava se uporedo sa razvojem JavaScripta. Uprkos pomalo užurbanom kreiranju (koje je 1995. godine za samo 10 dana uspeo da izvede Brendan Eich iz kompanije „Netscape"), JavaScripta je danas standardizovani programski jezik, koji brzo raste, sa više funkcija koje su naprednije od većine funkcija drugih sličnih popularnih jezika. Sveprisutnost ovog programskog jezika u pregledačima, serverima, mobilnim telefonima i tako dalje takođe je podstakla interesovanje za bolje razvojne strategije. Iako sam JavaScript nije zamišljen kao funkcionalan jezik, činjenica je da obezbeđuje sve funkcije koje su potrebne za rad na funkcionalan način, što je još jedna prednost.

Takođe se mora reći da se FP generalno ne primenjuje u računarstvu, možda zato što ima određene poteškoće u sebi, a smatra se da je *teorijski*, a ne *praktični*, pa, čak, i *matematički*, i da se u njemu verovatno koriste vokabular i pojmovi koji su strani programerima - na primer, funktori, monade, preklapanje, teorija kategorija, itd. Iako će vam učenje ove teorije sigurno biti korisno, može se reći da, čak i ako nemate znanje o prethodnim terminima, možete da razumete principe FP-a i kako da ga primenite u sopstvenom programiranju.

Funkcionalno programiranje nije nešto što morate da radite sami, bez ikakve pomoći. Postoji mnogo biblioteka i radnih okvira koji u većoj ili manjoj meri obuhvataju koncepte FP-a. Lista funkcionalnih pomoćnih alatki za kodiranje neprestano raste, od jQueryja (koji uključuje neke FP koncepte), Underscorea i njegovog „bliskog rođaka" Lodasha ili drugih biblioteka, kao što je Ramda, pa

sve do složenijih alatki za razvoj veb stranica, kao što su React Redux i Angular ili Elm (100 odsto funkcionalni jezik koji se kompajlira u JavaScriptu).

Učenje upotrebe FP-a može biti korisna investicija, pa iako možda nećete koristiti sve njegove metode i tehnike, sam početak primene nekih od njih će se isplatiti tako što ćete napisati bolji kod. Ne morate pokušavati da primenite celokupno funkcionalno programiranje od samog početka, već samo delimično, a takođe ne morate pokušavati da odbacite svaku nefunkcionalnu karakteristiku u JavaScript jeziku. Ovaj jezik sigurno ima i neke loše karakteristike, ali ima i nekoliko vrlo dobrih i moćnih karakteristika. Ideja nije da odbacite sve što ste naučili i da koristite i usvojite 100 odsto funkcionalan način, već je vodeća ideja *evolucija, a ne revolucija*. U tom kontekstu može se reći da ono što ćemo raditi nije FP, već **Sorta Functional Pogramming (SFP)**, **pomoću kojeg možemo da postignemo** fuziju paradigmi.

Završni komentar o stilu koda u ovoj knjizi je da postoji nekoliko vrlo dobrih biblioteka koje obezbeđuju FP alatke, kao što su biblioteke Underscore, Lodash, Ramda i još mnogo njih. Međutim, ja sam odbacio upotrebu ovih biblioteka, jer sam želeo da pokažem kako FP zaista funkcioniše. Lako je primeniti određenu funkciju iz nekog paketa, ali ako kodirate sve (tj. ako želite *osnovni* FP), verujem da ćete bolje razumeti neke aspekte. Osim toga, kao što sam komentarisao na nekim mestima, zbog moći i jasnoće streličastih i drugih funkcija, *čiste JavaScript verzije* mogu biti još jednostavnije za razumevanje.

KOME JE NAMENJENA OVA KNJIGA

Ova knjiga je namenjena programerima koji dobro poznaju JavaScript, koji koriste klijente (pregledače) ili servere (Node.js) i koji su zainteresovani da primene tehnike kako bi mogli da napišu bolji, testni, razumljivi i održivi kod. Neke pozadine u računarskoj nauci (uključujući, na primer, strukture podataka) i dobre programske tehnike takođe će biti korisne.

ŠTA OBUHVATA OVA KNJIGA

U ovoj knjizi ćemo na praktičan način prikazati FP, ali ćemo ponekad pomenuti i neke teorijske aspekte:

U Poglavlju 1, „*Postati funkcionalan - nekoliko pitanja*“, razmatramo FP i navedeni su razlozi za njegovo korišćenje i alatke koje će vam biti potrebne da biste iskoristili ostatak knjige.

U Poglavlju 2, „*Razmišljati funkcionalno - prvi primer*“, dat je prvi primer FP-a razmatranjem uobičajenog problema koji se odnosi na Veb i razmatramo nekoliko rešenja da biste se konačno usredsredili na funkcionalno rešenje.

U Poglavlju 3, „*Početak upotrebe funkcija - osnovni koncept*“, razmatramo koncept FP-a, tj. funkcije i različite opcije koje su dostupne u JavaScriptu.

U Poglavlju 4, „*Pravilno ponašanje - čiste funkcije*“, razmatramo koncept čistoće i čistih funkcija i biće prikazano kako taj koncept vodi ka jednostavnijem kodiranju i lakšem testiranju.

U Poglavlju 5, „*Deklarativno programiranje - bolji stil*“, koriste se jednostavne strukture podataka da bi bilo prikazano kako se postižu rezultati koji ne funkcionišu na imperativan, već na deklarativan način.

U Poglavlju 6, „*Izrada funkcija višeg reda*“, razmatramo funkcije višeg reda koje primaju ostale funkcije kao parametre i proizvode nove funkcije kao rezultate.

U Poglavlju 7, „*Funkcije transformacije - currying i parcijalna primena*“, istraženi su neki metodi za proizvodnju novih i specijalizovanih funkcija iz ranijih funkcija.

U Poglavlju 8, „*Funkcije spajanja - protočnost i kompozicija*“, prikazani su ključni koncepti koji se odnose na izradu novih funkcija spajanjem prethodno definisanih funkcija.

U Poglavlju 10, „*Obezbeđivanje čistoće – nepromenljivost*“, predstavljene su neke alatke koje vam mogu pomoći da radite na čist način obezbeđivanjem nepromenljivih objekata i struktura podataka.

U Poglavlju 11, „*Implementiranje projektnih obrazaca na funkcionalan način*“, prikazano je kako se nekoliko popularnih OOP projektnih obrazaca implementira (ili kako se ne implementira) kada programirate na FP načine.

U Poglavlju 12, „*Izrada boljih kontejnera - funkcionalni tipovi podataka*“, istraženi su neki funkcionalni obrasci visokog nivoa predstavljanjem tipova, kontejnera, funktora, monada i nekoliko drugih naprednijih koncepata FP-a.

Pokušao sam da primere u ovoj knjizi napišem tako da budu jednostavni i praktični, jer želim da se fokusiram na funkcionalne aspekte, a ne na sitne detalje nekih problema. U nekim odeljcima o programiranju se fokusiramo na učenje, recimo, određenog radnog okvira, a zatim razmatramo zadati problem, pokazujući kako da ga u potpunosti rešimo pomoću izabranih alatki. U stvari, na samom početku planiranja ove knjige razmišljao sam da razvijem aplikaciju koja će koristiti sve FP aspekte koje sam imao na umu, ali nije bilo načina da sve to uklopim u jedan projekat. Pomalo preterujući, osećao sam se kao lekar koji pokušava da pronađe pacijenta na kome će primeniti celokupno svoje medicinsko znanje i tretmane! Dakle, odlučio sam da prikažem mnogo individualnih tehnika koje se mogu koristiti u više situacija. Umesto da „gradim kuću“, pokazujem kako da složite „cigle“, kako da spojite elemente i tako dalje da biste mogli da primenite sve što vam treba za ono što želite.

DA BISTE DOBILI MAKSIMUM IZ OVE KNJIGE

Da biste razumeli koncepte i kod u ovoj knjizi, potrebni su vam samo JavaScript okruženje i uređivač teksta. Da budem iskren, čak sam razvio neke primere koji u potpunosti funkcionišu na Internetu samo pomoću alatke JSFiddle (na stranici https://jsfiddle.net/) i sličnih.

U ovoj knjizi koristićemo ES2019 i Node 13, a kod će se izvršavati na bilo kojem operativnom sistemu, kao što su Linux, Mac OSX ili Windows. Pročitajte odeljak „Tehnički zahtevi", koji se odnosi na neke druge alatke koje ćemo takođe koristiti.

Na kraju, potrebno vam je neko iskustvo u korišćenju najnovije verzije JavaScripta, jer ona uključuje nekoliko funkcija koje vam mogu pomoći u pisanju sažetijeg i kompaktnijeg koda. Često ćemo ukazivati na dokumentaciju na Internetu, poput dokumentacije koja je dostupna na **Mozilla Development Networku (MDM-u)**, na adresi https://programer.mozilla.org/, da biste stekli bolje znanje.

PREUZIMANJE DATOTEKA SA PRIMERIMA KODA

Datoteke sa kodom možete da preuzmete tako što ćete pratiti sa našeg sajta:

https://bit.ly/2yx6XUB

Kada je datoteka preuzeta, raspakujte ili ekstrahujte direktorijum, koristeći najnoviju verziju:

- **WinRAR/7-Zip** za Windows
- **Zipeg/iZip/UnRarX** za Mac
- **7-Zip/PeaZip** za Linux

UPOTREBLJENE KONVENCIJE

Postoji veliki broj konvencija teksta koje su upotrebljene u ovoj knjizi.

`Code InText` - Ukazuje na reči koda u tekstu, nazive tabela baze podataka, nazive direktorijuma, nazive datoteka, ekstenzije datoteka, nazive putanje, skraćene URL-ove, korisnički unos i Twitter postove. Evo i primera: „Sada ćemo pregledati funkciju `once()`."

Blok koda je prikazan na sledeći način:

```
function newCounter() {
    let count = 0; return function() {
      count++;
      return count;
    };
  }

  const nc = newCounter();
  console.log(nc()); // 1
  console.log(nc()); // 2
  console.log(nc()); // 3
```

Kada želimo da skrenemo pažnju na određeni deo bloka koda, relevantne linije ili stavke pišemo zadebljanim slovima:

```
function fact(n) {
  if (n === 0) {
    return 1;

  } else {
    return n * fact(n - 1);
  }
}

console.log(fact(5)); // 120
```

Zadebljana slova - Ukazuju na novi termin, važnu reč ili reči koje vidite na ekranu. Na primer, reči u menijima ili okvirima za dijalog prikazane su u tekstu zadebljanim slovima. Evo i primera: „Izaberite opciju **EXPERIMENTAL** da biste u potpunosti omogućili ES10 podršku."

Napomene ili važna obaveštenja prikazani su ovako.

Saveti i trikovi su prikazani ovako.

POVRATNE INFORMACIJE

Povratne informacije od naših čitalaca su uvek dobrodošle.

Štamparske greške - Iako smo preduzeli sve mere da bismo obezbedili tačnost sadržaja, greške su moguće. Ako pronađete neku grešku u ovoj knjizi, bili bismo zahvalni ako biste nam to prijavili - posetite stranicu knjige:

https://bit.ly/3c56urc

i napišite komentar.

Piraterija - Ako pronađete ilegalnu kopiju naših knjiga u bilo kojoj formi na Internetu, molimo vas da nas o tome obavestite i da nam pošaljete adresu lokacije ili naziv veb sajta. Kontaktirajte sa nama na adresi copyright@packt.com i pošaljite nam link ka sumnjivom materijalu.

Ako ste zainteresovani da postanete autor - Ako postoji tema za koju ste specijalizovani, a zainteresovani ste da pišete ili sarađujete na nekoj od knjiga, pogledajte vodič za autore na adresi authors.packtpub.com.

Recenzija

Kada pročitate i upotrebite ovu knjigu, zašto ne biste napisali vaše mišljenje na sajtu sa kojeg ste je poručili? Potencijalni čitaoci mogu da upotrebe vaše mišljenje da bi se odlučili na kupovinu, mi u „Kompjuter biblioteci“ saznaćemo šta mislite o našem konkretnom proizvodu, a naš autor može da vidi povratne informacije o svojoj knjizi.

TEHNIČKI ZAHTEVI

Da bih razvio i testirao kod u ovoj knjizi, koristio sam nekoliko verzija široko dostupnog softvera, uključujući pregledače i Node.js, ali i neke druge pakete.

Za ovo drugo izdanje knjige na mom glavnom uređaju pokrenuto je *Tumbleweed* „klizno“ izdanje OpenSUSE Linuxa sa sajta https://www.opensuse.org/#Tumbleweed, koje trenutno uključuje kernel 5.3.5 (termin „klizno“ podrazumeva da se softver ažurira kontinuirano da biste mogli da dobijete najnovije verzije svih paketa). Takođe sam testirao delove koda ove knjige na različitim Windows 7 i Windows 10 uređajima.

Što se tiče pregledača, obično koristim Chrome sa sajta https://www.google.com/chrome/browser/, a trenutno imam verziju 78. Koristim i Firefox sa sajta https://www.mozilla.org/en-US/firefox/, a na mom uređaju imam verziju 72. Takođe sam pokrenuo kod, koristeći JSFiddle okruženje na Internetu sa sajta https://jsfiddle.net/.

Na strani servera koristim Node.js verziju 13.6 sa sajta https://nodejs.org/.

Za prevođenje koristim Babel sa sajta https://babeljs.io/; trenutna verzija paketa babel-cli je 7.7.7.

Za testiranje koristim Jasmine sa sajta https://jasmine.github.io/, a najnovija verzija na mom uređaju je 3.5.0.

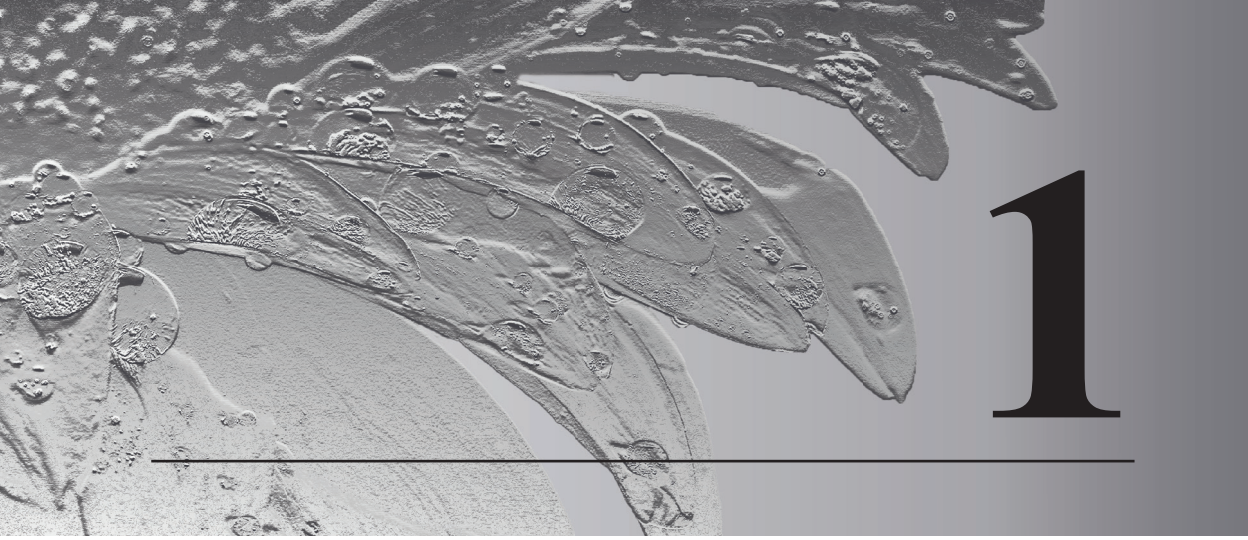
Na kraju, za formatiranje koda koristim Prettier sa sajta https://prettier.io/. Možete ga instalirati lokalno ili pokrenuti na Internetu, na adresi https://prettier.io/playground/; verzija koju imam je 1.19.1.

„Svet“ Jave je prilično dinamičan i sigurno je da će kada pročitate ovu knjigu svi prethodno navedeni softveri biti ažurirani više puta. Svaki pojedinačni softver koji sam koristio kada sam napisao prvo izdanje ove knjige tokom vremena je ažuriran nekoliko puta. Međutim, imajući u vidu standardizaciju JavaScripta i veliki značaj kompatibilnosti sa starijim verzijama, ne bi trebalo da imate problema kada koristite druge verzije.

Postanite član Kompjuter biblioteke

Kupovinom jedne naše knjige stekli ste pravo da postanete član Kompjuter biblioteke. Kao član možete da kupujete knjige u pretplati sa 40% popusta i učestvujete u akcijama kada ostvarujete popuste na sva naša izdanja. Potrebno je samo da se prijavite preko formulara na našem sajtu.
Link za prijavu: http://bit.ly/2TxekSa

Skenirajte QR kod
registrujte knjigu
i osvojite nagradu



1

Postati funkcionalan - nekoliko pitanja

Funkcionalno programiranje (ili **FP**) postoji od najranijih dana računarstva i doživljava neku vrstu preporoda, zbog sve veće upotrebe u nekoliko radnih okvira i biblioteka, posebno u **JavaScriptu (JS-u)**. U ovom poglavlju ćemo uraditi sledeće:

- Predstavićemo neke koncepte FP-a da biste stekli uvid u to šta je FP.
- Prikazaćemo prednosti (i probleme) koji podrazumevaju korišćenje FP-a i zašto bi trebalo da ga koristimo.
- Razmotrićemo zašto se JavaScript može smatrati prikladnim jezikom za FP.
- Pregledaćemo funkcije JavaScript jezika i alatke koje bi trebalo da znate da biste u potpunosti iskoristili sve iz ove knjige.

Kada pročitate ovo poglavlje, poznavaćete osnovne alatke koje ćemo koristiti u ostatku knjige.

ŠTA JE FUNKCIONALNO PROGRAMIRANJE?

Ako se vratite na istoriju računara, ustanovićete da se drugi najstariji programski jezik Lisp, koji je i dalje u upotrebi, zasniva na FP-u. U međuvremenu se pojavilo mnogo funkcionalnijih programskih jezika i FP se primenjuje mnogo više. Međutim, čak i ako pitate ljude šta je FP, verovatno ćete dobiti dva vrlo različita odgovora.

Kada je reč o još nekim zanimljivostima ili istoriji programskih jezika, napomenućemo da je najstariji programski jezik koji se još uvek koristi Fortran, a pojavio se 1957. godine, tj. godinu dana pre jezika Lisp. Ubrzo nakon jezika Lisp, pojavio se još jedan dugovečni jezik COBOL, koji služi za poslovno-orijentisano programiranje.

U zavisnosti od toga koga pitate, saznaćete da je FP koncept moderan, napredan i „prosvetljen" pristup programiranju, koji ostavlja svaku drugu paradigmu iza sebe ili da je uglavnom teorijska ideja, sa više komplikacija nego koristi, koju je praktički nemoguće implementirati u realnom svetu . A, kao i obično, pravi odgovor nije u krajnostima, već negde između. Prvo ćemo pregledati teoriju nasuprot prakse da biste videli kako možemo da planiramo da koristimo FP.

Teorija nasuprot prakse

U ovoj knjizi nećemo teoretski govoriti o FP-u. Umesto toga, naša poenta je da vam pokažemo kako se neke od FP tehnika i načela mogu uspešno primeniti za uobičajeno, svakodnevno JavaScript programiranje. Ali - i ovo je važno - nećemo o tome govoriti na dogmatski način, već na vrlo praktičan način. Nećemo odbaciti korisne JavaScript konstrukcije samo zato što se ne ispunjavaju akademska očekivanja FP-a. Slično tome, nećemo izbeći praktične JavaScript funkcije samo da bi se uklopili u FP paradigmu. Zapravo, skoro bismo mogli reći da ćemo koristiti Sorta Functional Programming (SFP), jer će naš kod biti kombinacija klasičnih imperativnih FP funkcija i objektno-orijentisanog programiranja (OOP-a).

Međutim, budite oprezni: ovo što smo upravo istakli ne znači da ćemo svu teoriju ostaviti po strani. Bićemo izbirljivi i osvrnućemo se samo na osnovne teorijske tačke, naučićete neke termine i definicije i objasnićemo osnovne FP koncepte, ali uvek ćemo pratiti ideju izrade stvarnog i korisnog JavaScript koda, umesto da pokušavamo da ispunimo neke mistične i dogmatske FP kriterijume.

OOP je bio način da se reši inherentna složenost pisanja velikih programa i sistema i razvije čista, proširiva i skalabilna arhitektura aplikacija. Međutim, zbog obima današnjih veb aplikacija, složenost svih baza kodova se kontinuirano povećava. Takođe novije funkcije JavaScripta omogućavaju razvoj aplikacija koje nije bilo moguće razviti pre samo nekoliko godina. Razmislite o mobilnim (hibridnim) aplikacijama koje se izrađuju pomoću platformi Ionic, Apache Cordova ili React Native ili o desktop aplikacijama koje su izrađene pomoću platformi Electron ili NW.js. JavaScript jezik je takođe prešao na pozadinsko programiranje u radnom okruženju Node.js, tako da je danas upotreba ovog jezika ozbiljno povećana, pa se on bavi svim dodatnim složenostima modernog dizajna.

Drugačiji način razmišljanja

FP je drugačiji način pisanja programa i ponekad ga je teško naučiti. U većini jezika programiranje se vrši na imperativan način: program je niz iskaza izvedenih na propisani način, a željeni rezultat se postiže kreiranjem objekata i manipulisanjem njima, što, obično, znači modifikaciju samih objekata. FP se zasniva na kreiranju željenog rezultata ocenjivanjem izraza izrađenih iz funkcija koje su sastavljene. U FP-u je uobičajeno da se prosleđuju funkcije (kao što je prosleđivanje parametara drugim funkcijama ili vraćanje funkcija kao rezultat izračunavanja), da se ne koriste petlje (već rekurzija) i da se „preskaču" sporedni efekti (kao što su modifikacija objekata ili modifikacija globalnih promenljivih).

Drugim rečima, FP se fokusira na ono *šta* treba da uradite, a ne *kako* treba da uradite. Umesto da brinete o petlji ili nizovima, radite na višem nivou, tako što ćete se fokusirati na ono šta treba da uradite. Nakon što se naviknete na ovaj stil, ustanovićete da vaš kod postaje jednostavniji, kraći i elegantniji i da se može lako testirati i debagovati. Međutim, nemojte FP smatrati ciljem, već samo sredstvom za postizanje cilja, baš kao i sve softverske alatke. Funkcionalni kod nije dobar samo zato što je funkcionalan, a pisanje lošeg koda je podjednako moguće pomoću FP-a, baš kao i pomoću neke druge tehnike!

Šta FP nije?

Pošto smo razmatrali šta je FP, sada ćemo razjasniti i neke uobičajene zablude i pogledati šta FP *nije*:

- **FP nije „akademska kula od slonovače" -** Istina je da je **lambda račun** na kojem je FP zasnovan razvio Alonso Church 1936. godine kao alatku za dokazivanje važnih rezultata u teorijskoj računarskoj nauci (koja je prethodila modernim računarskim jezicima tokom više od 20 godina!). Međutim, FP jezici se danas koriste za sve vrste sistema.
- **FP nije suprotnost objektno-orijentisanom programiranju (OOP-u) –** U FP nije od suštinskog značaja izabrati deklarativni ili imperativni načina programiranja. Možete da kombinujete i da spajate načine programiranja onako kako vama odgovara, a mi ćemo u ovoj knjizi predstaviti najbolje načine.
- **FP nije mnogo komplikovan za učenje -** Neki od FP jezika prilično se razlikuju od JavaScripta, ali su razlike, uglavnom, sintaksičke. Kada naučite osnovne koncepte, videćete da u JavaScriptu možete dobiti iste rezultate kao i u FP jezicima.

Takođe je relevantno spomenuti da nekoliko modernih radnih okvira, poput kombinacije Reacta i Reduxa, obuhvataju FP ideje.

Na primer, u Reactu se kaže da je **prikaz** (tj. ono što korisnik u određenom trenutku vidi) funkcija trenutnog **stanja**. Koristite tu funkciju da biste izračunali šta HTML i CSS moraju proizvesti u svakom trenutku, razmišljajući po modelu **crne kutije**.

Slično tome, u Reduxu imate koncept **akcija** koje obrađuju **reduktori**. Akcija obezbeđuje neke podatke, a reduktor je funkcija koja generiše novo stanje za aplikaciju na funkcionalan način izvan trenutnog stanja i konkretnih podataka. Dakle, i zbog teorijskih prednosti (koje ćemo razmatrati u sledećem odeljku) i zbog praktičnih prednosti (kao što je upotreba najnovijih radnih okvira i biblioteka), ima smisla razmotriti FP kodiranje, koje ćemo sada nastaviti.

ZAŠTO SE KORISTI FP?

Tokom godina je razvijen veliki broj stilova programiranja, koji imaju i mnogo nedostataka. Međutim, FP se pokazao kao prilično fleksibilan i danas je od velikog značaja. *Zašto želite da koristite FP?* Prvo bi trebalo da postavite pitanje šta želite da dobijete pomoću njega, a tek onda sledi pitanje *da li ćete ono što želite dobiti pomoću FP-a*. Odgovorićemo na ova važna pitanja u narednim odeljcima.

Šta vam je potrebno?

Sigurno se možemo složiti da je sledeća lista karakteristika univerzalna. Vaš kod treba da ima sledeće kvalitete:

- **modularnost** - Funkcionalnost vašeg programa treba podeliti na nezavisne module, od kojih svaki sadrži ono što je potrebno za izvršavanje jednog aspekta funkcionalnosti programa. Izmene modula ili funkcije ne bi trebalo da utiču na ostatak koda.
- **razumljivost** - Osoba koja čita vaš program trebalo bi da prepozna komponente, funkcije i veze koda, bez nepotrebnog napora. Razumljivost je usko povezana sa **održavanjem** koda; vaš kod mora biti održavan neko vreme u budućnosti, bez obzira da li se menja ili da li mu se dodaje nova funkcija.
- **mogućnost testiranja** - **Jediničnim testiranjem** se isprobavaju mali delovi vašeg programa, pri čemu se proverava njihovo ponašanje, nezavisno od ostatka koda. Vaš stil programiranja treba da pogoduje načinu pisanja koda koji pojednostavljuje pisanje jediničnih testova. Ovi testovi su takođe poput dokumentacije, jer mogu pomoći čitaocima da shvate šta kod treba da radi.

- **proširivost** - Činjenica je da će vašem programu biti potrebno održavanje da biste mu možda dodali novu funkciju. Te izmene bi trebalo da utiču na strukturu i tok podataka originalnog koda samo minimalno (ili možda čak ni malo). Male izmene ne bi trebalo da podrazumevaju veliko ozbiljno refaktorisanje koda.
- **ponovna upotrebljivost** - Cilj **ponovne upotrebe koda** je ušteda resursa, vremena i novca i smanjenje redutantnosti korišćenjem prethodno napisanog koda. Postoje neke karakteristike koje pomažu u postizanju ovog cilja, kao što su **modularnost** (koju smo već pomenuli), **visoka kohezija** (svi delovi u modulu su spojeni), **labavo spajanje** (moduli su međusobno nezavisni), **razdvajanje delova programa** (delovi programa bi trebalo što manje da se preklapaju u funkcijama) i **skrivanje informacija** (unutrašnje izmene modula ne bi trebalo da utiču na ostatak sistema).

Šta ćete dobiti?

Dakle, da li vam FP obezbeđuje pet karakteristika koje su upravo navedene u prethodnom odeljku?

- U FP-u je cilj napisati odvojene nezavisne funkcije koje se zatim spajaju da bi bili „proizvedeni“ konačni rezultati.
- Programi koji su napisani u funkcionalnom stilu obično su čistiji, kraći i lakši za razumevanje.
- Funkcije se mogu same po sebi testirati, a FP kod ima prednosti u postizanju tog testiranja.
- Funkcije možete upotrebljavati u drugim programima, jer su samostalne, tj. ne zavise od ostatka sistema. Većina funkcionalnih programa deli zajedničke funkcije, koje ćemo razmotriti u ovoj knjizi.
- Funkcionalni kod ne uključuje sporedne efekte, što znači da možete razumeti namenu funkcije, tako što ćete je proučavati, a da pri tom ne morate da uzmete u obzir ostatak programa.

Konačno, kada se naviknete na FP stil programiranja, kod postaje razumljiviji i lakši za proširenje. Izgleda da se svih pet karakteristika može postići pomoću FP-a!

Da biste bolje shvatili razloge zbog kojih se koristi FP, preporučujem da pročitate knjigu „WhyFunctional Programming Matters“ autora Johna Hughesa, koja je dostupna na Internetu, na adresi www.cs.kent.ac.uk/people/staff/dat/miranda/whyfp90.pdf. Ta knjiga nije usmerena na JavaScript, ali argumenti su, u svakom slučaju, lako razumljivi.

Nije zlato sve što sija

Međutim, pokušajmo da malo balansiramo. Korišćenje FP-a nije „čudotvorni lek" koji će automatski poboljšati vaš kod. Neka FP rešenja su, zapravo, komplikovana i postoje programeri koji veoma mnogo uživaju u pisanju koda i pitaju se *šta neki kod radi.* Ako niste pažljivi, vaš kod može biti *samo za pisanje* i praktično ga ne možete održavati. Potreban vam je razumljiv, proširiv i ponovo upotrebljiv kod!

Još jedan nedostatak je što je možda teže pronaći programere koji su vešti u FP-u. Velika većina današnjih veb kodova napisana je na imperativne i nefunkcionalne načine, a većina programera navikla je na takav način rada. Za neke programere promena „opreme" i započinjanje pisanja programa na drugačiji način može se pokazati kao nepremostiva prepreka.

Konačno, ako pokušate da postanete potpuno funkcionalni, možda ćete se sukobiti sa JavaScriptom, a jednostavni zadaci mogu biti teški za obavljanje. Kao što smo rekli na početku, odlučićemo se za *sorta FP*, tako da nećemo drastično odbacivati ni jednu jezičku karakteristiku koja nije 100 odsto funkcionalna. Na kraju krajeva, želimo da koristimo FP da bismo pojednostavili kodiranje, a ne da bismo ga učinili kompleksnijim.

Iako ću se truditi da vam prikažem, baš kao i prilikom svake promene, prednosti funkcionalnog koda, uvek će biti poteškoća. Međutim, potpuno sam uveren da ćete moći nadvladati promene i da će vaša organizacija razviti bolji kod primenom FP-a. Usudite se da menjate! Dakle, pod pretpostavkom da se slažete da se FP može primeniti na vaše probleme, razmotrimo sada i drugo pitanje - da li možemo JavaScript koristiti na funkcionalan način i da li je to prikladno.

DA LI JE JAVASCRIPT FUNKCIONALAN?

U ovom trenutku nameće se još jedno važno pitanje - *da li je JavaScript funkcionalan jezik*. Kada obično razmišljate o FP-u, lista pomenutih programskih jezika ne uključuje JavaScript, ali uključuje manje uobičajene opcije, kao što su Clojure, Erlang, Haskell i Scala. Međutim, ne postoji precizna definicija za FP jezike ili precizan skup funkcija koje takvi jezici treba da sadrže. Glavna poenta je da možete smatrati jezik funkcionalnim ako podržava uobičajeni stil programiranja povezan sa FP-om. Prvo ćete razjasniti zašto biste uopšte želeli da koristite JavaScript i kako se taj jezik razvijao do njegove trenutne verzije, a zatim ćete videti neke ključne funkcije koje ćete koristiti da biste izvršavali zadatke na funkcionalan način.

JavaScript kao alatka

Šta je JavaScript? Ako uzmete u obzir **indekse popularnosti**, poput onih na sajtu www.tiobe.com/tiobe-index/ ili na sajtu http://pipl.github.io/PIPL.html, videćete da je JavaScript konstantno u prvih 10 najpopularnijih jezika. Sa akademske tačke gledišta, jezik je vrsta kombinacije koja „pozajmljuje" funkcije iz više različitih jezika. Nekoliko biblioteka pomoglo je razvoju JavaScript jezika obezbeđivanjem funkcija koje nisu bile lako dostupne, kao što su klase i nasleđivanje (današnja verzija jezika podržava klase, ali do skoro ih nije podržavala), pa su morale biti postignute pomoću nekih **prototipskih** trikova.

Naziv JavaScript je izabran da bi bila iskorišćena popularnost Jave - baš kao marketinški trik! Prvi naziv bio je Mocha, zatim LiveScript, a tek onda JavaScript.

JavaScript je postao neverovatno moćan. Kao i sve moćne alatke, on obezbeđuje način da se „proizvode" ne samo odlična rešenja, ali i da nastane velika šteta. FP se može smatrati načinom za smanjenje ili izostavljanje nekih najlošijih delova jezika i fokusiranje na rad na bezbedniji i bolji način. Međutim, zbog ogromne količine postojećeg JavaScript koda, ne možete očekivati da će on olakšati velike obrade jezika koje mogu prouzrokovati neuspeh većine sajtova. Morate naučiti da živite sa dobrim i zlim i da jednostavno izbegavate loše delove.

Osim toga, taj jezik ima širok spektar dostupnih biblioteka koje ga upotpunjuju ili proširuju na više načina. U ovoj knjizi ćemo se fokusirati na samostalno korišćenje JavaScripta, ali ukazaćemo na postojeći dostupan kod.

JavaScript se može smatrati funkcionalnim, zbog nekoliko funkcija, kao što su prvoklasne funkcije, anonimne funkcije, rekurzije i zatvaranja, na koje ćemo se vratiti kasnije. Sa druge strane, JavaScript takođe ima veliki broj aspekata *koji nisu FP*, kao što su sporedni efekti (**nečistoća**), promenljivi objekti i praktična ograničenja rekurzije. Dakle, prilikom programiranja na funkcionalan način iskoristićemo sve relevantne i odgovarajuće jezičke funkcije i pokušaćemo da ublažimo probleme koji su nastali zbog konvencionalnijih delova jezika. U tom kontekstu, JavaScript će biti funkcionalan ili neće biti funkcionalan, u zavisnosti od *vašeg* stila programiranja.

Ako želite da koristite FP, trebalo bi da odlučite koji jezik ćete koristiti. Međutim, odlučiti se za potpuno funkcionalne jezike možda i nije toliko mudro. Razvijanje koda nije tako jednostavno kao samo korišćenje jezika; sigurno će vam biti potrebni radni okviri, biblioteke i druge alatke. Ako možemo da iskoristimo sve ponuđene alatke, ali da istovremeno uvedemo FP načine rada u kodu, dobićemo najbolje iz oba „sveta“, bez obzira da li je JavaScript funkcionalan.

Funkcionalnost pomoću JavaScripta

JavaScript se razvijao tokom godina, a verzija koju ćemo koristiti se (nezvanično) naziva JS10 i (zvanično) ECMAScript 2019, koja se skraćeno, obično, zove ES2019 ili ES10; ova verzija je završena u junu 2019. godine. Prethodne verzije su bile sledeće:

- ECMAScript 1, jun 1997. godine
- ECMAScript 2, jun 1998. godine, ta verzija je, u osnovi, bila ista kao i prethodna
- ECMAScript 3, decembar 1999. godine, sa nekoliko novih funkcija
- ECMAScript 5, decembar 2009. godine (nikada nije postojala verzija ECMAScript 4, jer se od nje odustalo)
- ECMAScript 5.1, jun 2011. godine
- ECMAScript 6 (ili ES6; kasnije preimenovana u verziju ES2015), jun 2015. godine
- ECMAScript 7 (takođe ES7 ili ES2016), jun 2016. godine
- ECMAScript 8 (ES8 ili ES2017), jun 2017. godine
- ECMAScript 9 (ES9 or ES2018), jun 2018. godine

ECMA je prvobitno bila skraćenica za European Computer ManufacturersAssociation, ali danas se taj naziv više ne smatra akronimom. Organizacija je odgovorna za više standarda koji nisu JavaScript, uključujući JSON, C# i Dart. Više detalja potražite na ECMA stranici, na adresi www.ecma-international.org/.

Standardnu specifikaciju jezika možete pročitati na stranici ww.ecma-international.org/ecma-262/7.0/. Kad god u tekstu ukazujemo na JavaScript bez dodatnih specifikacija, ES10 (ES2019) je ono na šta ukazujemo. Međutim, kada je reč o jezičkim karakteristikama koje se koriste u ovoj knjizi, uglavnom ne biste nailazili na probleme ukoliko biste koristili samo ES2015.

Ni jedan pregledač u potpunosti ne implementira ES10. Većina pregledača ima stariju verziju JavaScript 5 (od 2009. godine), sa (uvek rastućim) funkcijama od ES6 do ES10. Ovo je problem, ali, srećom, rešiv. Ubrzo ćemo razmatrati taj problem. Koristićemo ES10 u celoj knjizi.

U stvari, postoji samo nekoliko razlika između verzije ES2016 i verzije ES2015, kao što su metod Array.prototype.includes i eksponencijalni operator **. Postoji više razlika između verzije ES2017 i verzije ES2016, kao što su funkcije async i await, neke funkcije za dopunjavanje znakovnog niza i mnoge druge, ali one neće uticati na vaš kod. U kasnijim poglavljima ćemo takođe pregledati alternative za još modernije dodatke, kao što je flatMap().

Pošto ćemo koristiti JavaScript, prvo ćemo razmotriti njegove najvažnije funkcije koje se odnose na ciljeve koje želimo postići pomoću FP-a.

Ključne funkcije JavaScripta

JavaScript nije čisto funkcionalan jezik, ali ima sve funkcije koje su potrebne da bi delovao kao da jeste funkcionalan. Glavne funkcije jezika koje ćemo koristiti su:

- funkcije kao objekti prve klase
- rekurzija
- streličaste funkcije
- zatvaranja
- proširivanje

Sada ćemo pogledati nekoliko primera svake od ovih funkcija i saznaćete zašto će nam biti potrebne. Međutim, imajte u vidu da postoji više funkcija JavaScripta koje ćemo koristiti; u narednim odeljcima su istaknute samo najvažnije koje ćemo koristiti za FP.

Funkcije kao objekti prve klase

Reći da su funkcije **objekti prve klase** (koji se zovu i „**građani" prve klase**) znači da možete koristiti funkcije na isti način kao i druge objekte. Na primer, možete da uskladištite funkciju u promenljivu, da je prosledite drugoj funkciji, da je odštampate i tako dalje. Ovo je zaista „ključ" za izvršavanje FP-a; često ćemo prosleđivati funkcije kao parametre (drugim funkcijama) ili vraćati funkciju kao rezultat poziva funkcije.

Ako ste obavljali asinhrone Ajax pozive, onda ste već koristili **povratni poziv**, tj. funkciju koja će biti pozvana nakon završetka Ajax poziva, a zatim prosleđena kao parametar. Koristeći jQuery, mogli biste napisati nešto poput sledećeg:

```
$.get("some/url", someData, function(result, status) {
    // check status, and do something
    // with the result
});
```

Funkcija `$.get()` prima funkciju povratnog poziva kao parametar i poziva je nakon što je dobijen rezultat.

To se bolje rešava na savremeniji način upotrebom promisa ili async/wait funkcije, ali za primere iz ove knjige stariji način je dovoljan. Vratićemo se na promise u odeljku „Izrada boljih kontejnera“ u Poglavljua 12, *„Izrada boljih kontejnera - funkcionalni tipovi podataka“*, kada budemo razmatrali monade. Posebno pogledajte odeljak „Neočekivane monade - promisi“.

Pošto se funkcije mogu sačuvati u promenljivim, takođe možete napisati kod koji je sličan kodu u nastavku. Obratite pažnju na to kako koristimo promenljivu `doSomething` u pozivu `$.get(...)`:

```
var doSomething = function(result, status) {
    // check status, and do something
    // with the result
};

$.get("some/url", someData, doSomething);
```

Videćete više ovakvih primera u Poglavlju 6, *„Izrada funkcija višeg reda“*.

Rekurzija

Rekurzija je najmoćnija „alatka“ za razvoj algoritama i odlična pomoć za rešavanje velikih klasa problema. Ideja je da funkcija u određenom trenutku može da pozove samu sebe, a kada je *taj* poziv završen, da nastavi da se izvršava, bez obzira na rezultat koji je primila. Rekurzija je, obično, korisna za određene klase problema ili definicije. Najčešći primer je funkcija faktorijela (faktorijel *n* se piše kao *n!*), kao što je definisano za negativne celobrojne vrednosti:

Ako je *n* jednako 0, onda je *n!=1.*

Ako je *n* veći od 0, onda je *n! = n * (n-1)*!.

Vrednost n! predstavlja broj načina na koje možete rasporediti n različitih elemenata u nizu. Na primer, ako želite da stavite pet knjiga u nizu, možete odabrati bilo koju od njih za prvo mesto, a zatim na sve moguće načine rasporediti ostale četiri, dakle 5! = 5*4!. Ako nastavite da radite na ovom primeru, dobićete 5! = 5*4*3*2*1 = 120, pa je n! proizvod svih brojeva do n.

Ovo se može odmah pretvoriti u kod:

```
function fact(n) {
 if (n === 0) {
 return 1;

 } else {
 return n * fact(n - 1);
 }
}


console.log(fact(5)); // 120
```

Rekurzija će biti od velike pomoći za projektovanje algoritama. Iako nam nije namera da pomoću rekurzije bez petlji `while` ili `for` projektujemo algoritme, zanimljivo je da možemo to da uradimo! Celo Poglavlje 9, „*Funkcije projektovanja - rekurzija*“, posvetićemo projektovanju algoritama i rekurzivnom pisanju funkcija.

Zatvaranja

Zatvaranje je način na koji se implementira skrivanje podataka (pomoću privatnih promenljivih), pri čemu dobijamo module i druge dobre funkcije. Ključni koncept zatvaranja je da se ono može odnositi ne samo na lokalne promenljive kada definišete funkciju, već i na sve izvan konteksta funkcije. Možemo napisati funkciju brojanja koja će uskladištiti broj pomoću zatvaranja:

```
function newCounter() {
 let count = 0; return function() {
 count++;
 return count;
 };
}


const nc = newCounter();
console.log(nc()); // 1
console.log(nc()); // 2
console.log(nc()); // 3
```

Čak i nakon prekida funkcije `newCounter()`, unutrašnja funkcija i dalje ima pristup promenljivoj `count`, koja nije dostupna ni jednom drugom delu vašeg koda.

Ovo nije baš dobar primer FP-a - ne očekuje se da funkcija (nc() u ovom slučaju) vrati drugačije rezultate kada se pozove zajedno sa istim parametrima.

Pronaći ćemo nekoliko načina za zatvaranja, kao što su **memoizacija** (pogledajte Poglavlje 4, *„Pravilno ponašanje - čiste funkcije“*, i Poglavlje 6, *„Izrada funkcija višeg reda“*) i obrazac **modula** (pogledajte Poglavlje 3, *„Početak upotrebe funkcija - osnovni koncept“*, i Poglavlje 11, *„Implementiranje projektnih obrazaca na funkcionalan način“*).

Streličaste funkcije

Streličaste funkcije su samo kraći i sažetiji način za kreiranje (neimenovane) funkcije. One mogu da se koriste skoro na svim mestima na kojima se može koristiti klasična funkcija, ali ne mogu da se koriste kao konstruktori. Sintaksa je `(parameter,anotherparameter, ...etc) => { statements }` ili `(parameter, anotherparameter, ...etc) =>`*`expression`*. Prva sintaksa omogućava da napišete onoliko koda koliko želite, a druga je skraćenica za `{return` *`expression`*`}`. Mogli bismo ponovo da napišemo prethodni primer Ajaxa:

```
$.get("some/url", data, (result, status) => {
  // check status, and do something
  // with the result
});
```

Nova verzija koda faktorijela mogla bi biti poput sledećeg koda:

```
const fact2 = n => {
  if (n === 0) { return 1;

  } else {
    return n * fact2(n - 1);
  }
};
console.log(fact2(5)); // also 120
```

Streličaste funkcije se, obično, zovu anonimne funkcije, jer nemaju nazive. Ako treba da referencirate streličastu funkciju, moraćete da je dodelite promenljivoj ili atributu promenljive da biste mogli da je koristite, kao što je to urađeno u ovoj knjizi. O streličastim funkcijama ćete saznati više detalja u odeljku „Streličaste funkcije“ u Poglavlju 3, *„Početak upotrebe funkcija - osnovni koncept“*.

Verovatno biste poslednju funkciju iz koda napisali u jednoj liniji - možete li da uočite ekvivalent našem prethodnom kodu? Upotreba ternarnog operatora umesto iskaza `if` je sasvim uobičajena:

```
const fact3 = n => (n === 0 ? 1 : n * fact3(n - 1));

console.log(fact3(5)); // again 120
```

Zahvaljujući ovom kraćem obliku koda, ne morate da pišete povratnu vrednost (`return`), jer se ona podrazumeva.

U lambda računu funkcija kao što je x => 2*x bila bi predstavljena kao λx.2*x. Iako postoje sintaksičke razlike, definicije su analogne. Funkcije sa više parametara su malo složenije; funkcija (x,y)=>x+y bi bila izražena kao λk.λy.x+y. O ovim funkcijama ćete saznati više detalja u odeljku *„Lambda izrazi i funkcije“* u Poglavlju 3, *„Početak upotrebe funkcija - osnovni koncept“*, i u odeljku „Currying“ u Poglavlju 7, *„Funkcije transformacije - currying i parcijalna primena“*.

Ima još jedan detalj koji treba da imate na umu: kada streličasta funkcija ima jedan parametar, možete izostaviti zagrade oko nje. Ja radije volim da ostavim zagrade, ali na kod sam primenio JS alatku za „ulepšavanje“ *Prettier*, koja ih uklanja. Vi ćete odlučiti da li ćete dodati zagrade ili ne. Više detalja o ovoj alatki potražite na stranici https://github.com/prettier/prettier.) Uzgred budi rečeno, koristio sam opcije formatiranja `--print-width 75 --tab-width 2 --no-bracket-spacing`.

Proširivanje

Operator proširivanja (pogledajte stranicu https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Spread_operator) omogućava da proširite izraz na mestima na kojima je potrebno više argumenata, elemenata ili promenljivih. Na primer, možete da zamenite argumente u pozivu funkcije, kao što je prikazano u sledećem kodu:

```
const x = [1, 2, 3];

function sum3(a, b, c) {
  return a + b + c;
}

const y = sum3(...x); // equivalent to sum3(1,2,3)
console.log(y); // 6
```

Takođe možete da kreirate nizove ili da ih pridružite, kao što je prikazano u sledećem kodu:

```
const f = [1, 2, 3];

const g = [4, ...f, 5]; // [4,1,2,3,5]

const h = [...f, ...g]; // [1,2,3,4,1,2,3,5]
```

Operator proširivanja možete da koristite i u objektima:

```
const p = { some: 3, data: 5 };
const q = { more: 8, ...p }; // { more:8, some:3, data:5 }
```

Možete ga koristiti i u funkcijama koje očekuju zasebne parametre, umesto niza. Najčešći primeri su funkcije `Math.min()` i `Math.max()`:

```
const numbers = [2, 2, 9, 6, 0, 1, 2, 4, 5, 6];
const minA = Math.min(...numbers); // 0

const maxArray = arr => Math.max(...arr);
const maxA = maxArray(numbers); // 9
```

Možete da napišete i sledeću jednakost, s obzirom da metod `.apply()` zahteva niz argumenata, ali metod `.call()` očekuje pojedinačne argumente:

```
someFn.apply(thisArg, someArray) === someFn.call(thisArg, ...someArray);
```

Ako ne možete da zapamtite argumente koji su potrebni za metode .apply() i .call(), ovaj mnemonik vam može pomoći: A je za niz, a C je za zarez. Više informacija potražite na stranicama https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply i https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_ Objects/Function/call.

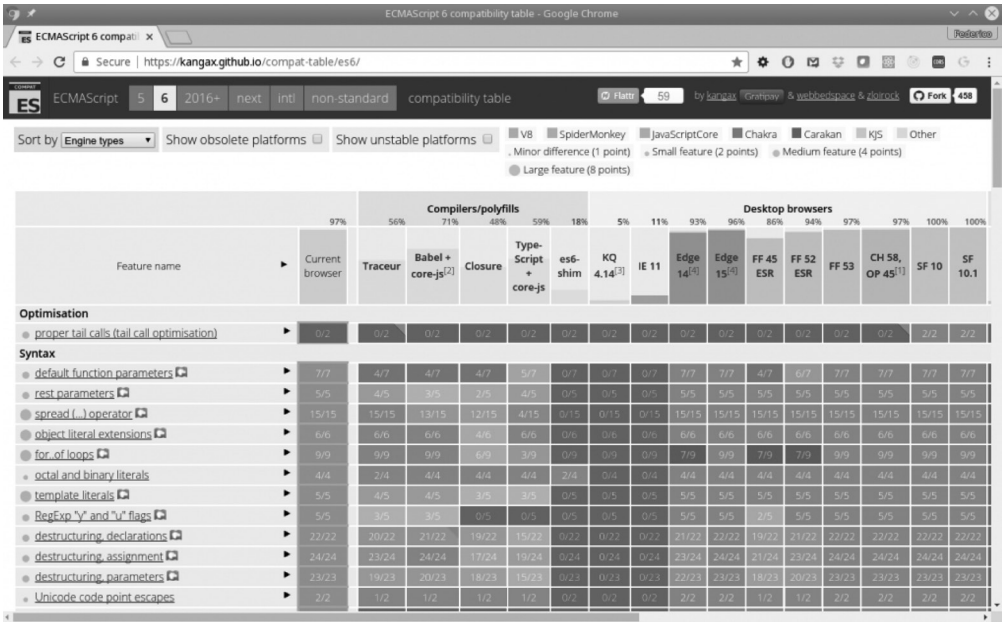
Upotreba operatora proširivanja pomaže da napišemo kraći i sažetiji kod, pa ćemo iskoristiti ovaj operator. Videli ste sve najvažnije JavaScript funkcije koje ćemo upotrebiti. Ovo poglavlje završavamo pregledom nekih alatki koje ćemo koristiti.

KAKO SE KORISTI JAVASCRIPT?

Sve je u najboljem redu, ali, kao što smo već napomenuli, dešava se da JavaScript verzija koja je dostupna nije ES10, već ranija verzija JS5. Izuzetak je radno okruženje Node.js, koje je zasnovano na Chromeovoj v8 JavaScript mašini visokih performansi, koja već ima nekoliko ES10 funkcija na raspolaganju. Bez obzira na to, u današnje vreme pokrivenost verzijom ES10 nije 100 odsto, a tu su i funkcije koje će vam nedostajati (više informacija o radnom okruženju Node.js i v8 JavaScript mašini potražite na stranici https://nodejs.Org/en/docs/es6/). To će se sigurno u budućnosti promeniti, jer će Internet Explorer postepeno zastarevati, a najnoviji „Microsoftov" pregledač će deliti Chromeovu mašinu, ali za sada moramo još uvek koristiti starije i manje moćne mašine.

Dakle, šta možete učiniti ako želite da kodirate pomoću najnovije verzije, ali je dostupna ranija lošija verzija? Ili šta se dešava ako većina vaših korisnika koristi starije pregledače, koji ne podržavaju fantastične funkcije koje želite da koristite? Pogledajte neka rešenja za to.

Ako želite da budete sigurni u svoj izbor, pre nego što upotrebite neke nove funkcije, pogledajte tabelu kompatibilnosti na stranici https://kangax.github.io/compattable/es6/ (slika slika 1.1). Konkretno za Node.js pogledajte stranicu http://node.green/.



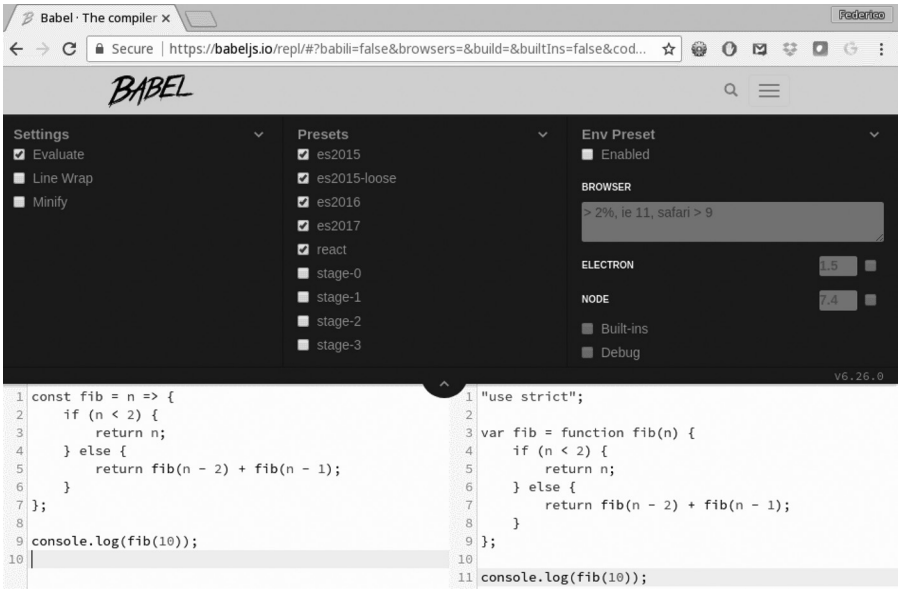
Slika 1.1 Najnovije JavaScript funkcije možda nisu široko i u potpunosti podržane, tako da ćete morati da proverite njihovu kompatibilnost pre upotrebe.

Korišćenje transpajlera

Da biste rešili problem koji se odnosi na dostupnost i kompatibilnost, na raspolaganju vam je nekoliko transpajlera koje možete koristiti. **Transpajleri** prihvataju vaš originalni ES10 kod, koji može da koristi najmodernije JavaScript funkcije i pretvara ga u ekvivalentni JS5 kod. To je transformacija od izvora do izvora, umesto koda od izvora do objekta koji bi se koristio u kompilaciji. Možete da kodirate pomoću naprednih ES10 funkcija, ali će pregledači korisnika primiti JS5 kod. Transpajler će vam takođe omogućiti da pratite predstojeće verzije jezika, uprkos vremenu koje je potrebno da pregledači usvoje nove standarde na desktop i mobilnim uređajima.

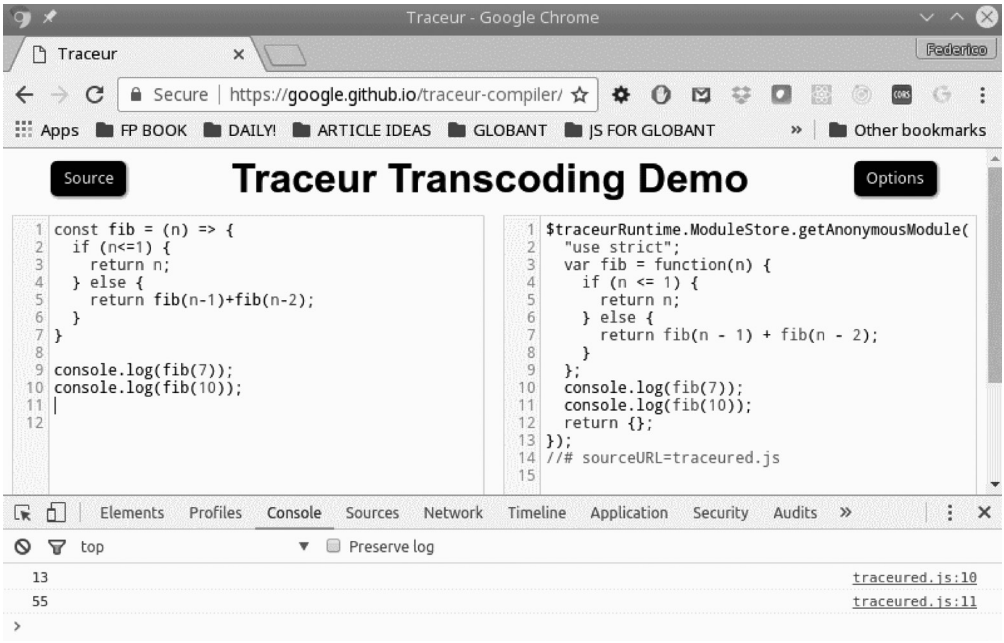
Reč transpajler (prevodilac) je složenica od prevoda i kompajlera. Postoji mnogo takvih kombinacija u tehnološkom govoru: e-pošta (elektronski i pošta), emotikon (emocija i ikona), zlonamerni softver (zlonamerni i softver) ili alfanumerički (abecedni i numerički) i mnoge druge.

Uobičajeni transpajleri za JavaScript su Babel (na https://babeljs.Io/) i Traceur (na https://github.Com/google/traceur-compiler). Pomoću alatki, kao što su npm ili webpack, prilično je jednostavno konfigurisati nešto tako da kod bude automatski preveden i obezbeđen krajnjim korisnicima. Takođe možete izvršiti prevod na Internetu. Na s*lici 1.2* pogledajte primer upotrebe Babelovog okruženja na Internetu:



Slika 1.2 Babel transpajler pretvara ES10 kod u kompatibilni JS5 kod

Ako više volite transpajler Traceur, možete da koristite njegovu alatku sa stranice https://google.github.io/traceurcompiler/demo/repl.html#, ali ćete morati da otvorite programersku konzolu da biste videli rezultate vašeg pokrenutog koda (na *slici 1.3* pogledajte primer prevedenog koda). Odaberite opciju **EXPERIMENTAL** da biste u potpunosti omogućili podršku ES10:



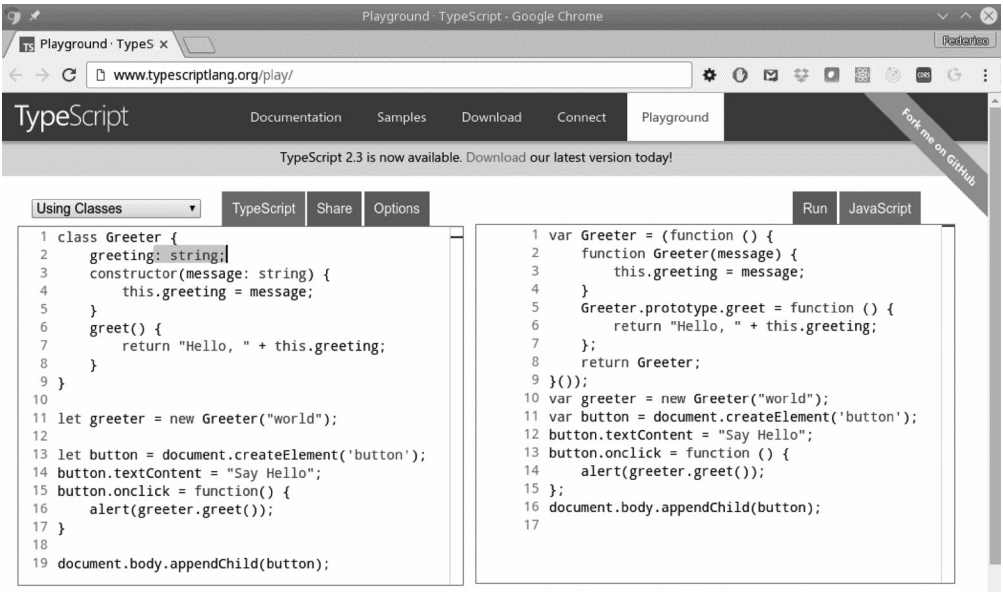
Slika 1.3 Transpajler Traceur podjednako je validna alternativa za prevod ES10 koda u JS5 kod.

Korišćenje transpajlera takođe je odličan način za učenje novih jezičkih funkcija. Samo ukucajte neki kod sa leve strane i videćete ekvivalentni kod sa desne strane. Alternativno, možete koristiti alatke interfejsa komandne linije (CLI) da biste preveli izvornu datoteku, a zatim pregledajte dobijeni izlaz.

Postoji krajnja mogućnost koju bi trebalo da razmotrite: umesto JavaScripta odlučite se za „Microsoftov" TypeScript (sa sajta http://www.typescriptlang.org/), skup jezika koji je sam kompajliran u JS5. Glavna prednost TypeScripta je mogućnost dodavanja (opcionih) provera statičkih tipova u JavaScriptu, što pomaže u otkrivanju određenih grešaka u programiranju tokom kompajliranja. Međutim, budite pažljivi: baš kao u Babelu ili u Traceuru, neće biti dostupan celokupan ES10.

Takođe možete izvršiti provere tipa bez TypeScripta, koristeći Facebook Flow (pogledajte sajt https://flow.org/).

Ako se odlučite za upotrebu TypeScripta, možete ga takođe testirati na Internetu na njegovom **„igralištu“** (pogledajte stranicu http://www.Typecriptlang.org/play/). Možete postaviti opcije za proveru tipa podataka da budu više ili manje stroge, a kod možete da pokrenete i na licu mesta (pogledajte *sliku 1.4* za više detalja):

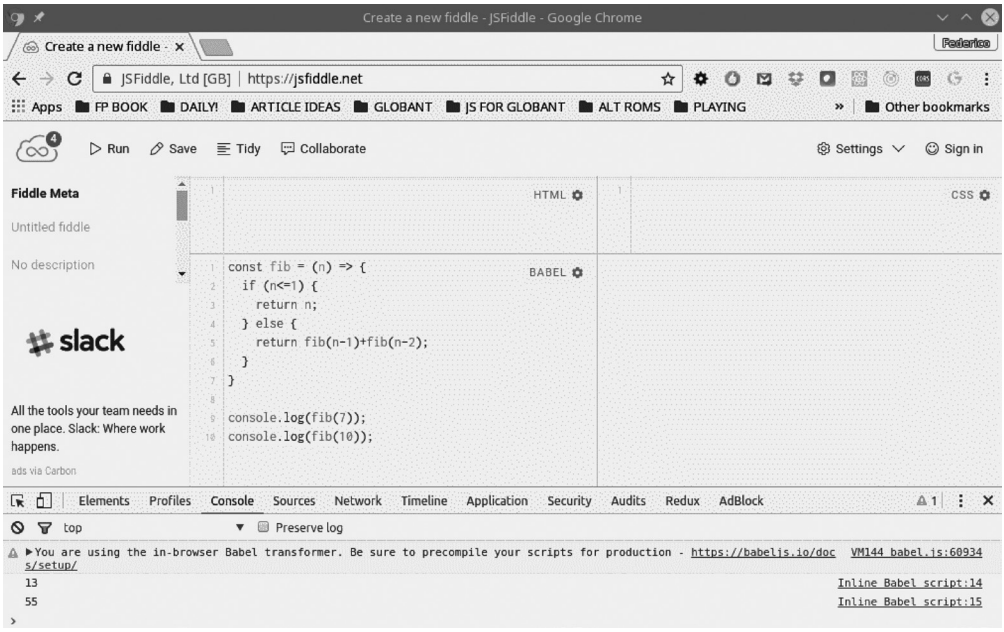


Slika 1.4 TypeScript dodaje funkcije provere tipa za bezbednije programiranje.

Korišćenjem TypeScripta možete da izbegnete uobičajene greške koje se odnose na tip podataka. Pozitivan trend je da većina alatki (radni okviri, biblioteke i tako dalje) počinje da izbegava uobičajene greške, tako da će rad biti lakši.

Rad na Internetu

Postoji još nekoliko alatki na Internetu pomoću kojih možete testirati svoj JavaScript kod. Pogledajte JSFiddle (na sajtu https://jsfiddle.net/), CodePen (na sajtu https://codepen.io/) i JSBin (na sajtu http://jsbin.com/). Možda ćete morati da navedete da li želite da koristite Babel ili Traceur da novije jezičke funkcije ne bi bile odbijene. Primer JSFiddlea možete videti na *slici 1.5*:



Slika 1.5 JSFiddle omogućava isprobavanje modernog JavaScript koda (ali i HTML-a i CSS-a), bez potrebe za nekim drugim alatkama.

Upotreba ovih alatki omogućava da veoma brzo isprobate kod ili da malo eksperimentišete, što mogu da vam garantujem, jer sam na taj način već isprobao veći deo koda u knjizi!

Testiranje

Takođe ćemo se dotaknuti i testiranja, što je, na kraju krajeva, i jedna od glavnih prednosti FP-a. Za testiranje ćemo koristiti alatku Jasmine (https: //jasmine.github.io/), mada bismo mogli da se odlučimo i za alatku Mocha (http://mochajs.org/).

Možete da pokrenete Jasmine pakete testova pomoću pokretača, kao što je Karme (https://karma-runner.github.io), ali ja sam se odlučio za samostalne testove; više detalja potražite na stranici https://github.com/jasmine/jasmine#installation.

REZIME

U ovom poglavlju ste videli osnove FP-a, deo njegove istorije, njegove prednosti (ali, da budemo iskreni, i neke moguće nedostatke), zašto ga možete primeniti u JavaScriptu (koji se, obično, ne smatra funkcionalnim jezikom) i koje alatke će vam biti potrebne da biste isprobali sve primere iz knjige.

U Poglavlju 2, „Razmišljati funkcionalno - prvi primer“, pregledaćemo primer jednostavnog problema, sagledaćemo problem na uobičajene načine i na kraju ćemo ga rešiti na funkcionalan način i analizirati prednosti načina koji smo primenili.

PITANJA

Klase kao objekti prve klase - Saznali smo da su funkcije objekti prve klase, ali da li ste znali da postoje i *klase kao objekti* (iako, naravno, govoriti o klasama kao o *objektima* zvuči čudno)? Pogledajte sledeći primer da biste videli šta ga pokreće! Napomena: u ovom primeru je namerno dodat neki neobičan kod:

```
const makeSaluteClass = term =>
 class {
    constructor(x) {
      this.x = x;
    }

    salute(y) {
      console.log(`${this.x} says "${term}" to ${y}`);
    }
  };

const Spanish = makeSaluteClass("HOLA");
new Spanish("ALFA").salute("BETA");
// ALFA says "HOLA" to BETA

new (makeSaluteClass("HELLO"))("GAMMA").salute("DELTA");
// GAMMA says "HELLO" to DELTA

const fullSalute = (c, x, y) => new c(x).salute(y);
const French = makeSaluteClass("BON JOUR");
fullSalute(French, "EPSILON", "ZETA");
// EPSILON says "BON JOUR" to ZETA
```

1.2 **Greške faktorijela** - Faktorijeli, kao što smo ih definisali, treba da se izračunaju samo za cele brojeve koji nisu negativni. Međutim, funkcija koju smo napisali u odeljku „Rekurzija“ ne potvrđuje da je njen argument validan. Da li možete da dodate potrebne provere (pokušajte da izbegnete ponovljene i redundantne testove)?

1.3.„**Penjanje“ faktorijela** - Primena faktorijela u primerima iz knjige počinje množenjem sa *n*, zatim sa *n-1*, pa sa *n-2* i tako dalje, na način koji bismo mogli nazvati *silazni*. Da li možete da napišete novu verziju funkcije faktorijela koja će praviti petlju *nagore*?

1.4. **Skraćivanje koda** - Ovo nije samo po sebi cilj, ali pomoću streličastih funkcija i nekih drugih funkcija JavaScripta možete skratiti `newCounter()` na polovinu njegove dužine. Da li znate kako?