POGLAVLJE 1

Uvod u JavaScript

JavaScript je programski jezik veba. Velika većina veb lokacija koristi JavaScript, a svi savremeni veb čitači – na radnim površinama, tabletima i telefonima – uključuju JavaScript interpreter, čime je JavaScript najrasprostranjeniji programski jezik u istoriji. Tokom poslednje decenije, Node.js je omogućio JavaScript programiranje izvan veb čitača, a dramatičan uspeh Noda znači da je JavaScript sada i najkorišćeniji programski jezik među programerima softvera. Bez obzira da li počinjete ispočetka ili već profesionalno koristite JavaScript, ova knjiga će vam pomoći da savladate jezik.

Ako ste već upoznati sa drugim programskim jezicima, možda će vam pomoći da znate da je JavaScript visoko dinamični, interpretirani programski jezik koji je dobro prilagođen objektno orijentisanim i funkcionalnim stilovima programiranja. JavaScript promenljive nisu tipizirane. Njegova sintaksa je bazirana na Javi, ali jezici inače nisu povezani. JavaScript svoje prvoklasne funkcije dobija iz Sheme i nasleđivanja prototipa iz malo poznatog jezika Self. Ali ne treba da znate nijedan od tih jezika ili da budete upoznati sa tim terminima da biste koristili ovu knjigu i naučili JavaScript.

Naziv „JavaScript“ je prilično pogrešan. Osim površne sintaktičke sličnosti, JavaScript se u potpunosti razlikuje od Java programskog jezika. A JavaScript je odavno prerastao svoje poreklo skriptnog jezika i postao je robustan i efikasan jezik opšte namene pogodan za ozbiljan softverski inženjering i projekte sa ogromnim bazama podataka.

JavaScript: Imena, verzije i načini rada

JavaScript je nastao u Netscapu u ranim danima veba, a tehnički je „JavaScript“ zaštitni znak licenciran od strane Sun Microsystems (sada Oracle) koji se koristi da opiše implementaciju Netscapovog jezika (sada Mozille). Netscape je predao jezik na standardizaciju Evropskom udruženju proizvođača računara (ECMA) – i zbog problema sa zaštitnim znakom, standardizovana verzija jezika bila je zaglavljena s rogobatnim nazivom „ECMAScript“. U praksi, svi jezik zovu JavaScript. Ova knjiga koristi naziv „ECMAScript“ i skraćenicu „ES“ da bi se odnosila na jezički standard i na verzije tog standarda.

Za veći deo 2010-ih, verziju 5 ECMAScript standarda podržavaju svi veb čitači. Ova knjiga tretira ES5 kao osnovu kompatibilnosti i više ne raspravlja o starijim verzijama jezika. ES6 je objavljen 2015. godine i dodao je nove funkcije – uključujući sintaksu klase i modula – koje su JavaScript promenile iz skriptnog jezika u ozbiljan opšti namenski jezik pogodan za softverski inženjering velikih razmera. Otkad je ES6, ECMAScript specifikacija prešla na godišnja izdanja, verzije jezika – ES2016, ES2017, ES2018, ES2019 i ES2020 – sada su identifikovane prema godini izdavanja.

Kako se JavaScript razvijao, dizajneri jezika pokušali su da isprave nedostatke u ranim (pre-ES5) verzijama. Da bi se održala kompatibilnost unazad, nije moguće ukloniti nasleđene funkcije, bez obzira koliko bile pogrešne. Ali u ES5 i novijim programima, programi mogu da se uključe u strogi JavaScript režim (*strict mode*) u kome su ispravljene brojne ranije greške jezika. Mehanizam za prijavu je direktiva „stroga upotreba" opisana u §5.6.3. U ovom odeljku sumirane su i razlike između starijeg i strogog JavaScripta. U ES6 i novijim, upotreba novih jezičnih funkcija često implicitno poziva na strogi režim rada. Na primer, ako koristite ključnu reč `class` u ES6 ili stvarate ES6 modul, tada je sav kod unutar klase ili modul je automatski strog, a stare, pogrešne funkcije nisu dostupne u tim kontekstima. Ova knjiga će pokriti nasleđene funkcije JavaScripta, ali pažljivo će naglasiti da one nisu dostupne u strogom režimu.

Da bi bio koristan, svaki jezik mora imati platformu ili standardnu biblioteku za obavljanje stvari kao što su osnovni unos i izlaz. Jezgro JavaScript jezike definiše minimalan API za rad sa brojevima, tekstom, nizovima, skupovima, mapama i slično, ali ne uključuje bilo koju ulaznu ili izlaznu funkcionalnost. Ulaz i izlaz (kao i sofisticiranije funkcije, kao što su rad sa mrežom, skladištenje i grafika) su odgovornost „domaćina" (engl. *host environment*) u okviru kojeg je ugrađen JavaScript.

Prvobitno okruženje domaćina za JavaScript bio je veb čitač, a to je i dalje najčešće okruženje u kome se izvršava JavaScript kod. Okruženje veb čitača omogućava JavaScript kod da dobije unos iz korisnikovog miša i tastature i pomoću HTTP zahteva. I omogućava JavaScript kodu da prikazuje izlaz korisniku korišćenjem HTML-a i CSS-a.

Od 2010. godine za JavaScript kod je dostupno još jedno okruženje domaćina. Umesto ograničavanja JavaScripta da radi sa API-jima koje pruža veb pretraživač, Node omogućava JavaScript pristup celom operativnom sistemu, omogućavajući JavaScript programima da čitaju i pišu datoteke, šalju i primaju podatke preko mreže i prave i opslužuju HTTP zahteve. Node je popularan izbor za implementaciju veb servera i takođe zgodan alat za pisanje jednostavnih uslužnih skripti kao alternativa skriptu školjke (engl. *shell*).

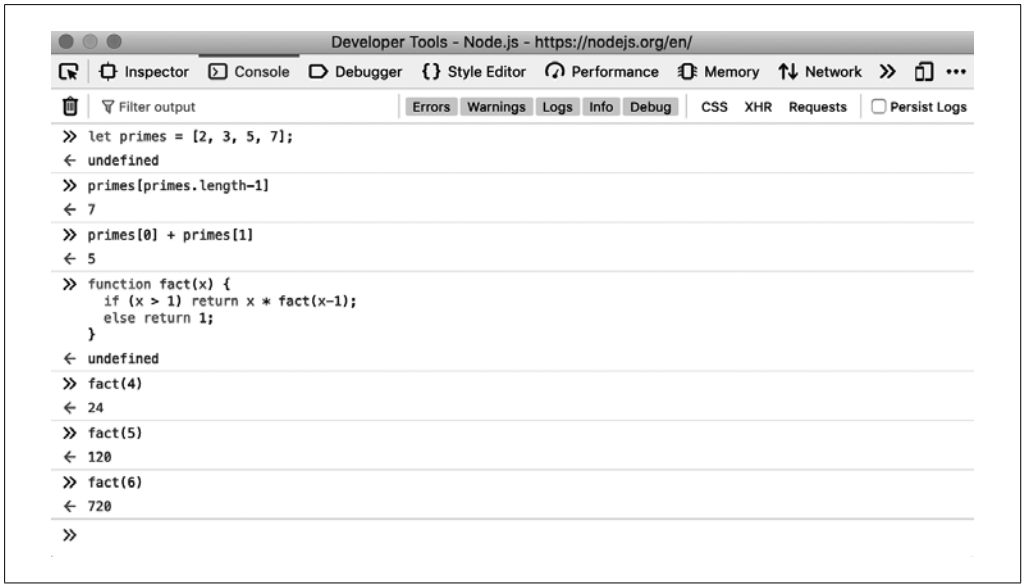
Većina ove knjige fokusirana je na sam jezik JavaScript. Poglavlje 11 dokumentuje JavaScript standardnu biblioteku, poglavlje 15 predstavlja okruženje domaćina veb čitača, a poglavlje 16 predstavlja okruženje kada je domaćin Node.

Ova knjiga prvo pokriva osnove niskog nivoa, a zatim nastavlja na one naprednije i apstrakcije višeg nivoa. Poglavlja su predviđena da se pročitaju više manje po redu. Ali učenje novog programskog jezika nikada nije linearni proces, a ni opis jezika nije linearan: svaka jezička karakteristika povezana je sa drugim osobinama, a ova knjiga je prepuna referenci – ponekad unazad, a ponekad unapred – na srodne materijale. Ovo uvodno poglavlje donosi brzi prvi prolazak kroz jezik, uvodeći ključne karakteristike koje će olakšati razumevanje dubljih tema u narednim poglavljima. Ako već imate praksu kao JavaScript programer, ovo poglavlje verovatno možete preskočiti. (Iako ćete možda uživati čitajući Primer 1-1 na kraju poglavlja pre nego što krenete dalje.)

1.1 Istraživanje JavaScripta

Kada učite novi programski jezik, važno je da isprobate primere iz knjige, a zatim ih modifikujete i izvršite ponovo da biste proverili svoje razumevanje jezika. Da biste to učinili, potreban vam je JavaScript interpreter.

Najlakši način da isprobate nekoliko linija JavaScripta je da otvorite razvojni alat (eng. *developer tools*) u svom veb čitaču (sa F12, Ctrl-Shift-I ili Command-Option-I) i izaberete karticu Console. Zatim, kada se pojavi odzivnik (engl. *promt*), možete otkucati kod i videti rezultate dok kucate. Razvojni alat čitača se pojavljuju kao okna na dnu ili desno od prozora čitača, ali obično ih možete izdvojiti kao odvojene prozore (kao što je prikazano na slici 1-1), što je često prilično zgodno.



Slika 1-1. JavaScript konzola u Firefoxovoj alatki za razvoj

Drugi način da se isproba JavaScript kod je preuzimanje i instaliranje Noda sa *https://nodejs.org*. Jednom kada je Nod instaliran na vašem sistemu, možete jednostavno otvoriti prozor terminala i otkucati **node** da biste započeli interaktivnu JavaScript sesiju poput ove:

```
$ node
Welcome to Node.js v12.13.0.
Type ".help" for more information.
> .help
.break Sometimes you get stuck, this gets you out
.clear Alias for .break
.editor Enter editor mode
.exit Exit the repl
.help Print this help message
.load Load JS from a file into the REPL session
.save Save all evaluated commands in this REPL session to a file

Press ^C to abort current expression, ^D to exit the repl
> let x = 2, y = 3;
undefined
> x + y
5
> (x === 2) && (y === 3)
true
> (x > 3) || (y < 3)
false
```

1.2 Hello World

Kada ste spremni da počnete eksperimentisati sa dužim komadima koda, ova linijska interaktivna okruženja možda više nisu prikladna i verovatno ćete radije da upišete kod u editor teksta. Odatle možete kopirati i prebaciti u JavaScript konzolu ili u sesiju Noda. Ili možete da sačuvate svoj kod u datoteci (tradicionalno tip datoteke za JavaScript kod je *.js*), a zatim pokrenite tu datoteku JavaScript koda pomoću Noda:

```
$ node snippet.js
```

Ako koristite Node na neinteraktivni način kao ovaj, on neće automatski ispisati vrednost celog koda koji pokrenete, tako da ćete to morati sami da uradite. Možete koristiti funkciju `console.log()` da prikažete tekst i ostale JavaScript vrednosti u prozoru vašeg terminala ili u konzoli alata za razvoj. Tako, na primer, ako napravite datoteku *hello.js* koja sadrži ovu liniju koda:

```
console.log ("Hello World!");
```

i izvršite datoteku pomoću `node hello.js`, videćete ispisanu poruku „Hello World!“ (Pozdrav svetu).

Ako želite da se poruka prikazana u JavaScript konzoli, prikaže u veb čitaču, napravite novu datoteku pod nazivom *hello.html* i stavite ovaj tekst u nju:

```
<script src="hello.js"> </script>
```

Zatim učitajte *hello.html* u svoj veb čitač koristeći `file://` URL poput ovog:

```
file:///Users/username/javascript/hello.html
```

Otvorite prozor alata za razvoj da biste videli pozdrav u konzoli.

1.3 Tura kroz JavaScript

Ovaj odeljak predstavlja brzi uvod, kroz primere koda, u JavaScript jezik. Nakon ovog uvodnog poglavlja, zaronimo u JavaScript na najnižem nivou: u drugom poglavlju su objašnjene stvari poput JavaScript komentara, tačaka sa zapetom i skup znakova Unicode. Sa poglavljem 3 počinje biti zanimljivije: objašnjava JavaScript promenljive i vrednosti koje možete dodeliti tim promenljivim.

Evo nekoliko primera koda koji ilustruju ta dva poglavlja:

```
// Bilo šta iza dve kose crte je komentar
// Čitajte komentare pažljivo: objašanjavaju JavaScript kod.

// Promenljiva je simboličeno ime za vrednost.
// Promenljive se deklarišu sa rezervisanom reči let.
let x;                  // Deklarisanje promenjive imena x.
// Vrednosti se dodeljuju promenljivoj sa znakom =
X = 0;                  // Sada promenljiva x ima vrednost 0
x                       // => 0: Promenljiva rezultuje svojoj vrednosti.

// JavaScript podržava nekoliko tipova vrdnosti
x = 1;                  // Brojeve.
x = 0.01;               // Brojevi mogu biti celobrojni ili decimalni.
x = "hello world";      // Niz znakova unutar navodnika.
x = ‘JavaScript’;       // Jednostruki navodnici takođe označavaju niz znakova.
x = true;               // Bulova vrednost.
x = false;              // Druga Bulova vrednost.
x = null;               // Null is a specijalna vrednost koja znači "bez vrednosti."
x = undefined;          // Undefined is druga specijalna vrednost kao i null.
```

Dva druga veoma važna *tipa* kojima JavaScript programi mogu da manipulišu su objekti i nizovi. To su teme iz poglavlja 6 i 7, ali su toliko važne da ćete ih videti mnogo puta pre nego što dođete do tih poglavlja:

```
// Objekat je najvažniji JavaScript tip podataka.
// Objekat je kolekcija parova imena/vrednosti ili niz preslikanih vrijednosti.
let knjiga = {             // Objekti su zatvoreni u vitičaste zagrade.
  tema: "JavaScript",      // Svojstvo "tema" ima vrednost "JavaScript"
  izdanje: 7               // Svojstvo "izdanje" ima vrednost 7
};                         // Vitičasta zagrada označava kraj objekta.

// Pristupite svojstvima objekta pomoću . ili []::
knjiga.tema                // => "JavaScript"
knjiga["izdanje"]          // => 7: another way to access property values.
knjiga.autor = "Flanagan"; // stvaranje novog svojstva dodeljivanjem.
knjiga.sadrzaj = {};       // {} prazan objekat bez svojstava.
```

```
// Uslovno pristupite svojstvima sa ?. (ES2020):
knjiga.sadrzaj?.pog1?.sek1 // => undefined: knjiga.sadrzaj nema svojstvo pog1.

// JavaScript takođe podržava nizove (numerički indeksirane liste) vrednosti:
let prosti = [2, 3, 5, 7]; // Niz od 4 vrednosti, ograničen sa [ i ].
prosti[0]                  // => 2: prvi element (indeks 0) niza.
prosti.length              // => 4: koliko ima elemenata u nizu.
prosti[prosti.length-1]    // => 7: poslednji element nizu.
prosti[4] = 9;             // Dodavanje novog elementa dodelom.
primes[4] = 11;            // Ili mijenjati postojećeg element dodelom.
let empty = [];            // [] je prazan niz bez elemenata.
empty.length               // => 0

// Nizovi i objetki mogu sadržati druge nizove i objekte:
let koordinate = [         // Niz sa 2 elementa.
  {x: 0, y: 0},            // Svaki elenment je jedan objekat.
  {x: 1, y: 1}
];
let data = {               // Objekat sa dva svojstva
  trial1: [[1,2], [3,4]],  // Vrednost svakog svojstva je jedan niz.
  trial2: [[2,3], [4,5]]   // Elementi niza su nizovi.
};
```

Sintaksa komentara u primerima koda

Možda ste u prethodnom kodu primetili da neki od komentara počinju strelicom (`=>`). Oni pokazuju vrednost proizvedenu kodom i moj su pokušaj da u štampanoj knjizi oponašam interaktivno JavaScript okruženja kao što je konzola veb čitača.

Ti `// =>` komentari takođe služe kao *tvrdnja* (engl. *assertion*), i napisao sam alat koji testira kod i proverava da li proizvodi vrednost koja je navedena u komentaru. Ovo bi trebalo da pomogne da se smanji broj grešaka u knjizi.

Postoje dva srodna stila komentara/tvrdnja. Ako vidite komentar oblika `// a == 42`, to znači da će, kada se kod ispred komentara pokrene, varijabla `a` imati vrednost 42. Ako vidite komentar oblika `// !`, To znači da kod ispred komentara daje izuzetak (a ostatak komentara iza uskličnika obično objašnjava kakvu vrstu izuzeća dobijamo).

Videćete ove komentare kroz celu knjigu.

Sintaksa za prikazivanje elemenata niza u uglastim zagradama ili za mapiranje imena svojstava objekata sa vrednostima svojstava unutar vitičastih zagrada poznata je kao izraz inicijalizatora (engl. *initializer expression*), a to je samo jedna od tema poglavlja 4. *Izraz* (engl. *expression*) je fraza JavaScripta koja može biti *izračunata* (engl. *evaluated*) da da vrednost. Na primer, upotreba `.` i `[]` da uputi na vrednost svojstva objekta ili elementa niza.

Jedan od najčešćih načina za formiranje izraza u JavaScriptu je upotreba *operatora*:

```
// Operatori deluju na vrednosti (operande) da bi proizveli novu vrednost.
// Aritmetički operatori su neki od najjednostavnijih:

3 + 2                           // => 5: sabiranje
3 - 2                           // => 1: oduzimanje
3 * 2                           // => 6: množenje
3 / 2                           // => 1.5: deljenje
koordinata[1].x - koordinata[0].x // => 1: nešto složeniji operadni
"3" + "2"                       // => "32": + sabira brojeve, a spaja znakove

// JavaScript definiše neke skrećenice aritmetičkih operacija
let broj = 0;                   // Defisanje promenljive
broj++;                         // Povećava za 1 vrednost promenljive
broj--;                         // Smanjuje za 1 vrednost promenljive
broj += 2;                      // Dodaje 2: isto kao count = broj + 2;
broj *= 3;                      // Množi sa 3: isto kao count = broj * 3;
broj                            // => 6: imena promenljivih su i izrazi

// Operatori za jednakost i relaciju testiraju da li su dve vrednosti jednake,
// nejednake, manje od, veće od, itd. Oni rezultuju sa true (tačno) ili false (netačno).
let x = 2, y = 3;               // Ovi znaci = su dodele, ne test jednakosti
x === y                         // => false: jednakost
x !== y                         // => true: nejednakost
x < y                           // => true: manje od
x <= y                          // => true: manje od ili jednako
x > y                           // => false: veće od
x >= y                          // => false: veće od ili jednako
"dva" === "tri"                 // => false: dva niza znakova se razlikuju
"dva" > "devet"                 // => true: "dv" je alfabetsko veće od "de"
false === (x > y)               // => true: false je jednako sa false

// Logički operatori kombinuju ili menjaju Bulove vredsnosti
(x === 2) && (y === 3)          // => true: oba poređenja su true.
                                // && je AND (logičko I)
(x > 3) || (y < 3)              // => false: ni jedno poređenje nije tačno.
                                // || je OR (logičko ILI)
!(x === y) // => true: ! menja Bulovu vrednost
```

Ako su JavaScript izrazi poput fraza, onda su JavaScript *naredbe* (engl. *statements*) poput rečenica. Naredbe su tema poglavlja 5. Grubo, izraz je nešto što izračunava vrednost, ali ne čini ništa: to ni na koji način ne menja stanje programa. Naredbe, s druge strane, nemaju vrednost, ali menjaju stanje. Gore ste videli deklaracije promenljivih i naredbe za dodeljivanje. Druga široka kategorija naredbi su *kontrolne strukture* (engl. *control structures*), kao što su uslovi i petlje. Videćete primere u nastavku, nakon što obradimo funkcije.

Funkcija (engl. *function*) je imenovani i parametrizovan blok JavaScript koda koji ste definisali jednom, a posle toga se može ponovo i ponovo pozivati. Funkcije nisu obuhvaćene formalno do poglavlja 8, ali poput objekata i nizova, videćete ih mnogo puta pre nego što dođete do tog poglavlja. Evo nekoliko jednostavnih primera:

```
// Funkcije su parametrizovani blokovi JavaScript koda koje možemo pozvati.
function plus1(x) {         // Definišite funkciju nazvanu "plus1" parametrom "x"
    return x + 1;           // Vraća vrednost koja je za 1 veća od primljene vrednosti
}                           // Funkcije se nalaze unutar vitičastih zagrada

plus1(y)                    // => 4: y je 3, tako da ovaj poziv vraća 3+1

let kvadrat = function(x) { // Funkcije su vrednosti i mogu se dodeliti promenljivama
    return x * x;           // Izračunava vrednost funkcije
};                          // tačka sa zapetom označava kraj dodele.

kvadrat(plus1(y))           // => 16: poziva dve funkcije u jednom izrazu
```

U ES6 i u novijim verzijama, postoji kratka sintaksa za definisanje funkcija. Ta sažeta sintaksa koristi => za odvajanje liste argumenata od tela funkcije, tako da su funkcije definisane na ovaj način poznate kao *funkcije strelice* (engl. *arrow functions*). Funkcije strelice najčešće se koriste kada želite da prosledite neimenovanu funkciju kao argument drugoj funkciji. Prethodni kod izgleda ovako kada je prerađen da bi koristio strelice:

```
const plus1 = x => x + 1;   // Ulaz x se mapira u izlaz x + 1
const kvadrat = x => x * x; // Ulaz x se mapira u izlaz x * x
plus1(y)                    // => 4: isti poziv funkcije
kvadrat(plus1(y))           // => 16
```

Kada koristimo funkcije sa objektima, dobijamo *metode*:

```
// Kad se funkcije dodijele svojstvima objekta, nazivamo
// ih "methodama."  Svi JavaScript objekti (uključujući i nizove) imaju methode:
let a = [];                // Pravimo prazan niz
a.push(1,2,3);             // metoda push() dodaje element u niz
a.reverse();               // Druga metoda: preokreće redosled elemenata

// Možemo da definišemo sopstvene metode. Ključna reč "this" odnosi se na objekat
// na kome je metoda definisana: u ovom slučaju ukazuje na niz od ranije.

points.dist = function() { // Definišite metodu za računanje deljne između dve tačke
    let p1 = this[0];      // Prvi element niza na koji se pozvamo
    let p2 = this[1];      // Drugi element objekta "this"
    let a = p2.x-p1.x;     // Razlika u x koordinatama
    let b = p2.y-p1.y;     // Razlika u y koordinatama
    return Math.sqrt(a*a + // Pitagorina teorema
                     b*b); // Math.sqrt() izračunava kvadratni koren
};
points.dist()              // => Math.sqrt(2): distanca između naše 2 tačke
```

Sada, kao što je obećano, evo nekoliko funkcija čija tela pokazuju uobičajene JavaScript naredbe kontrolne strukture:

```
// JavaScript naredbe uključuju uslove i petlje koristeći sintaksu
// od C, C ++, Java i drugih jezika.
function abs(x) {          // Funkcija računa apsolutnu vrednost.
    if (x >= 0) {          // if naredba...
```

```
        return x;          // izvršava ovaj kod ako je poređenje true.
    }                      // Ovo je kraj if uslova.
    else {                 // Opcioni else uslov se izvršava ako je
        return -x;         // poređenje false.
    }                      // Vitičasta zagrada je opciona za jednu naredbu u uslovu.
}                          // return nardba je unutar if/else.
abs(-10) === abs(10)       // => true

function sum(array) {      // Izračunava sumu vrednosti elemenata niza
    let sum = 0;           // Počinje od toga da vrednos sum 0.
    for(let x of array) {  // Ide kroz niz dodeljući svaki element u x.
        sum += x;          // Dodaje vrednost elementa sumi.
    }                      // Ovo je kraj petlje.
    return sum;            // Vraća sumu.
}
sum(primes)                // => 28: suma prvih 5 prostih brojeva 2+3+5+7+11

function factorial(n) {    // Funkcija izračunava faktorijel
    let product = 1;       // Počinje od toga da je product 1
    while(n > 1) {         // Ponavlja naredbu u {} dok je (while) izraz u () true
        product *= n;      // Skraćeni oblik za product = product * n;
        n--;               // Skraćeni oblik za n = n - 1
    }                      // Ovo je kraj petlje
    return product;        // Vraća product
}
factorial(4)               // => 24: 1*4*3*2

function factorial2(n) {   // Drgua verzija sa drugačijom petljom
    let i, product = 1;    // Počinje od 1
    for(i=2; i <= n; i++)  // Automatski povećava i do 2 do n
        product *= i;      // Radi ovo svaki put. {} nisu potrebne za jednoredu petlju
    return product;        // Vraća factorijel
}
factorial2(5)              // => 120: 1*2*3*4*5
```

JavaScript podržava objektno orijentisani stil programiranja, ali se značajno razlikuje od „klasičnih“ objektno orijentisanih programskih jezika. Poglavlje 9 detaljno pokriva objektno orijentisano programiranje u JavaScriptu, sa puno primera. Evo vrlo jednostavnog primera koji pokazuje kako definisati JavaScript klasu koja će predstavljati 2D geometrijske tačke. Objekti koji predstavljaju instance ove klase imaju jedinstvenu metodu, nazvanu `distance()`, koja izračunava udaljenost tačke od početka:

```
class Point {              // Po konvenciji, imena klasa počinju velikim slovom.
    constructor(x, y) {    // Funkcija constructor za inicijalizaciju novih instanci.
        this.x = x;        // Ova ključna reč je objekat koji se inicijalizuje.
        this.y = y;        // Smešta argmente funkcije kao svojstva objekta.
    }                      // Ne treba return u funkciji konstruktoru

    distance() {           // Method izračunava distancu od početka do tačke.
        return Math.sqrt(  // Vraća kvadrati koren od x² + y².
            this.x * this.x +  // this se odnosi na objekat Point object za koji
```

```
            this.y * this.y    // se poziva metoda distance.
        );
    }
}

// Upotrebite funkciju constructor Point() sa "new" da biste kreirali Point objekte
let p = new Point(1, 1);   // Geometrijska tačka (1,1).

// Sada upotrebite metodu p objekta Point
p.distance()               // => Math.SQRT2
```

Ovde se završava uvodna turneja po osnovnoj sintaksi i mogućnostima JavaScripta, ali knjiga se nastavlja sa samostalnim poglavljima koja pokrivaju dodatne funkcije jezika:

Poglavlje 10, Moduli

Prikazuje kako JavaScript kod u jednoj datoteci ili skripti može koristiti JavaScript funkcije i klase definisane u drugim datotekama ili skriptama.

Poglavlje 11, Standardna biblioteka JavaScripta

Obuhvata ugrađene funkcije i klase koje su dostupne svim JavaScript programima. Ovo uključuje važne strukture podataka kao što su mape i skupovi, klase regularnih izraza za upoređivanje tekstualnih uzoraka, funkcije za serializaciju JavaScript strukrutra podataka i mnogo više.

Poglavlje 12, Iteratori i generatori

Objašnjava kako radi `for/of` petlja i kako možete napraviti sopstvene iterabilne klase `for/of`. Takođe obuhvata funkcije generatora i naredbu o prinosu.

Poglavlje 13, Asinhroni JavaScript

Ovo poglavlje je detaljno istraživanje asinhronog programiranja u JavaScript-u, obuhvata povratne pozive i događaje, API-e zasnovane na obećanjima i ključne reči asinhronizacije i čekanja. Iako jezgro JavaScript jezika nije asinhrono, asinhroni API-ji su zadati i veb čitačima i u Nodu, a ovo poglavlje objašnjava tehnike rada sa tim API-jima.

Poglavlje 14, Metaprogramiranje

Predstavlja brojne napredne funkcije JavaScripta koje mogu biti zanimljive programerima koji pišu biblioteke koda za upotrebu od strane drugih JavaScript programera.

Poglavlje 15, JavaScript u veb čitačima

Uvodi okruženje veb čitača, objašnjava kako veb čitači izvršavaju JavaScript kod i pokriva najvažniji od mnogih API-ja definisanih od strane veb čitača. Ovo je daleko najduže poglavlje u knjizi.

Poglavlje 16, JavaScript na strani servera i Node

Uvodi okruženje Noda, a pokriva osnovni programski model i strukture podataka i API-je koji su najvažniji za razumevanje.

Poglavlje 17, JavaScript alati i proširenja

Pokriva alate i jezične nastavke o kojima vredi znati jer se široko koriste i mogu vas učiniti produktivnijim programerom.

1.4 Primer: Histogrami učestalosti znakova

Ovo poglavlje se završava kratkim, ali netrivijalnim JavaScript programom. Primer 1-1 je Node program koji čita tekst sa standardnog unosa, izračunava histogram frekvencije znakova iz unetog teksta, a zatim ispisuje histogram. Možete pozvati ovaj program da analizirate frekvenciju znakova svog izvornog koda:

```
$ node charfreq.js < charfreq.js
T: ########### 11.22%
E: ########## 10.15%
R: ####### 6.68%
S: ###### 6.44%
A: ###### 6.16%
N: ###### 5.81%
O: ##### 5.45%
I: ##### 4.54%
H: #### 4.07%
C: ### 3.36%
L: ### 3.20%
U: ### 3.08%
/: ### 2.88%
```

Ovaj primer koristi brojne napredne JavaScript funkcije i namenjen je da prikaže kako mogu izgledati JavaScript programi u stvarnom svetu. Ne bi trebalo očekivati da ćete razumeti sav kod, ali budite sigurni da će sve to biti objašnjeno u narednim poglavljima.

Primer 1-1. Izračunavanje histograma učestalosti znakova pomoću JavaScripta

```
/**
* Ovaj Node program čita tekst sa standardnog ulaza, izračunava frekvenciju
* svakog slova u tom tekstu, i prikazuje histograma najviše
* korišćenih znakovi. Za pokretanje je potreban Node 12 ili noviji.
*
* U okruženju tipa Unix možete pozvati program ovako:
*   node charfreq.js < corpus.txt
* /

// Ova klasa proširuje Map tako da metoda get () vraća zadatu
// vrednost umesto null kada indeks (key) nije na mapi
class DefaultMap extends Map {
  constructor(defaultValue) {
    super();                           // poziva konstruktor superclass
    this.defaultValue = defaultValue; // pamti unapred zadatu vrednost
  }
```

```
  get(key) {
    if (this.has(key)) {              // ako je indeks key u mapi
    return super.get(key);            // vraća njegovu vrednost iz superklase.
  }
  else {
    return this.defaultValue;         // U protivnom vraća zadatu vrednost
  }
}

}
// Ova klasa izračunava i prikazuje histogram učestalosti znakova
class Histogram {
  constructor() {
    this.letterCounts = new DefaultMap(0); // mapira znakove u broj
    this.totalLetters = 0;                 // koliko ima ukupno slova
  }

  // Ova funkcija ažurira histogram slovima iz teksta
  add(text) {
    // Uklanjamo duple prazne znakove iz teksta, i pretvaramo u velika slova
    text = text.replace(/\s/g, "").toUpperCase();

    // Sada prolazimo kroz znakove teksta
    for(let character of text) {
      let count = this.letterCounts.get(character); // Stara vrednost koliko ih ima
      this.letterCounts.set(character, count+1);    // Povećavamo
      this.totalLetters++;
    }
  }

  // Pretvara histogram u niz znakova da prikaže ASCII grafiku
  toString() {
    // Pretvara Map u niz [key,value] nizova
    let entries = [...this.letterCounts];

    // Sortira niz po broju, a zatim abecedno
    entries.sort((a,b) => {           // Funkcija za sort redosled
      if (a[1] === b[1]) {            // Ako su brojevi isti
        return a[0] < b[0] ? -1 : 1; // sortiraj abecedno.
      } else {                        // Ako se brojevi razlikuju
        return b[1] - a[1];           // sortiraj po većem broju
      }
    });

    // Pretvara brojeve u procente
    for(let entry of entries) {
      entry[1] = entry[1] / this.totalLetters*100;
    }

    // Odbacuje vrednosti manje od 1%
    entries = entries.filter(entry => entry[1] >= 1);
```

```
    // Pretvara svaku stavku u liniju teksta
    let lines = entries.map(
      ([l,n]) => `${l}: ${"#".repeat(Math.round(n))} ${n.toFixed(2)}%`
    );

    // Vraća spojene linije, razdvojene znakom za novi red.
    return lines.join("\n");
  }
}


// Ova asihrona funkcija (vraća obećanje) stvara objekat Histogram,
// asinhrono čita delova teksta sa standardnog ulaza i dodaje te komade u
// histogram. Kada dođe do kraja podataka, vraća ovaj histogram
async function histogramFromStdin() {
  process.stdin.setEncoding("utf-8"); // Čita Unicode nizove znakove, ne bajtove
  let histogram = new Histogram();
  for await (let chunk of process.stdin) {
    histogram.add(chunk);
  }
  return histogram;
}

// Ova poslednja linija koda je glavno telo programa.
// Stvara objekat Histogram sa standardnog ulaza, a zatim prikazuje histogram.
histogramFromStdin().then(histogram => { console.log(histogram.toString()); });
```

1.5 Rezime

Ova knjiga objašnjava JavaScript odozdo nagore. To znači da počinjemo sa detaljima niskog nivoa poput komentara, identifikatora, promenljivih i tipova; zatim nadograđujemo izraze, naredbe, objekte i funkcije; a zatim pokrivamo jezičke apstrakcije visokog nivoa poput klasa i modula. Reč *sveobuhvatan* u naslovu ove knjige shvatam ozbiljno, a naredna poglavlja objašnjavaju jezik na nivou detalja koji u početku može biti neugodan. Međutim, istinsko savladavanje JavaScripta zahteva razumevanje detalja i nadam se da ćete imati vremena da pročitate ovu knjigu od korica do korica. Ali molim vas nemojte osećati da to morate da učinite pri prvom čitanju. Ako se nađete zaglavljeni u nekom delu, jednostavno pređite na sledeći. Možete se vratiti i savladati detalje nakon što steknete znanje jezika u celini.