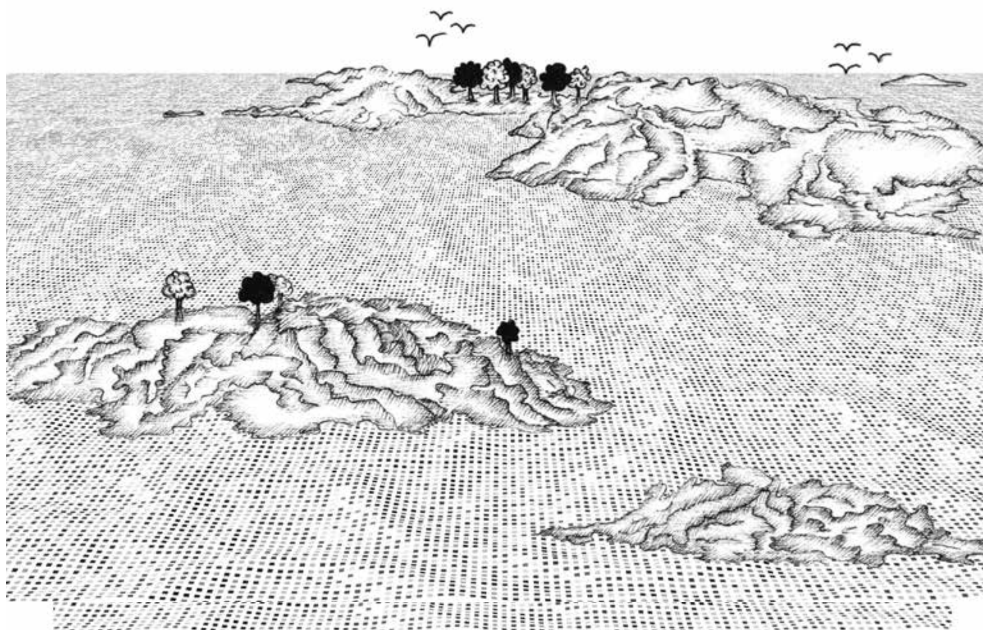
DEO I

JEZIK

„Ispod površine mašine, kreće se program. Bez truda, on se širi i skuplja. U velikoj harmoniji, elektroni se rasipaju i ponovo grupišu. Oblici na monitoru samo su talasi na vodi. Suština ostaje ispod, nevidljiva."

– Majstor Yuan-Ma, *The Book of Programming*



1

VREDNOSTI, TIPOVI I OPERATORI

Unutar računarskog sveta suštinski postoje samo podaci. Čitate podatke, menjate ih, pravite nove podatke. Ono što nije podatak - nije vredno spomena. Svi ti podaci su uskladišteni kao dugačke sekvence bitova i zbog toga su, u osnovi, slični.

Bitovi (engl. *bits*) su bilo šta što ima dve vrednosti, obično opisane kao nule i jedinice. U računaru oni poprimaju oblik kao što je visok ili nizak električni naboj, jak ili slab signal ili sjajna ili zamućena tačka na površini CD-a. Bilo koji komad informacije može se svesti na sekvencu nula i jedinica i time predstaviti bitovima.

Na primer, broj 13 možemo izraziti u bitovima. To funkcioniše isto kao decimalni broj, ali umesto deset različitih cifara, imate samo dve, a „težina“ svake se povećava dva puta, zdesna ulevo. Evo bitova koji čine broj 13, sa težinom cifara prikazanom ispod njih:

```
  0   0   0   0  1  1  0  1
128  64  32  16  8  4  2  1
```

Dakle, to je binarni broj 00001101. Vrednost cifara koje nisu nule su 8, 4 i 1, i u zbiru daju 13.

Vrednosti

Zamislite more bitova – ili čitav okean. Tipičan savremeni računar ima preko 30 milijardi bitova u svojoj radnoj verziji. Čvrsti disk obično ima nekoliko redova veličine više.

Da bismo mogli da radimo sa tolikom količinom bitova, a da se ne izgubimo, moramo ih podeliti u komade koji predstavljaju delove informacija. U okruženju JavaScripta, ti komadi se nazivaju vrednosti (engl. *values*). Iako su sve vrednosti sačinjene od bitova, one igraju različite uloge. Svaka vrednsot ima tip koji određuje njenu ulogu. Neke vrednosti su brojevi, neke su tekst, neke su funkcije itd.

Da biste napravili vrednost, samo treba da pozovete njeno ime. To je zgodno. Ne morate da prikupljate gradivni materijal za vrednosti niti da plaćate za njih. Samo je pozovete, i vuuššš, imate je. Naravno, one ne nastaju ni iz čega. Svaka vrednost mora negde da bude uskladištena i ako želite da koristite ogromnu količinu vrednosti istovremeno, moglo bi vam ponestati memorije. Srećom, to je problem samo ako vam sve one trebaju istovremeno. Čim više ne budete koristili vrednost, ona će iščeznuti i ostaviti za sobom svoje bitove da budu reciklirani kao gradivni materijal za narednu generaciju vrednosti.

Ovo poglavlje predstavlja atomske elemente JavaScript programa, tj. jednostavne tipove vrednosti i operatore koji mogu delovati na takve vrednosti.

Brojevi

Vrednosti tipa *broj* (engl. *number*), nimalo iznenađujuće, jesu brojčane (numeričke) vrednosti. U JavaScript programu, one se zapisuju ovako:

```
13
```

Upotrebite to u programu i u memoriji računara će nastati obrazac bitova za broj 13.

JavaScript koristi fiksan broj bitova, 64, za skladištenje jedne brojčane vrednosti. Postoji određeni broj obrazaca koje možete napraviti pomoću 64 bita, a to znači da je broj različitih brojeva koje možete predstaviti ograničen. Sa *N* decimalnih cifara, možete predstaviti 10^N brojeva. Slično tome, kada imate 64 binarne cifre, možete predstaviti 2^{64} različitih brojeva, što je otprilike 18 triliona (US engl. *quintillion*) (18 sa 18 nula). To je mnogo.

Računarska memorija je nekada bila mnogo manja i ljudi su obično koristili grupe od 8 ili 16 bitova da bi predstavljali brojeve. Bilo je lako slučajno *premašiti* tako male brojeve – završiti s brojem koji se nije uklapao u dati broj bitova. Danas čak i računari koji staju u vaš džep imaju obilje memorije, pa slobodno možete koristiti 64-bitne komade, a o prekoračivanju treba da brinete samo ako radite sa zaista astronomskim brojevima.

Ipak, ne staju svi celi brojevi manji od 18 triliona u JavaScriptov broj. Ti bitovi skladište i negativne brojeve, pa jedan bit označava znak broja. Još je veća stvar to što neceli brojevi takođe moraju biti predstavljeni. Da bi se

to uradilo, neki od bitova se koriste za skladištenje položaja decimalne tačke. Stvarni maksimalan ceo broj koji se može uskladištiti kreće se pre u rangu 9 bilijardi (15 nula) – što je i dalje ugodno ogromno.

Decimalni brojevi (u programskim jezicima) se pišu pomoću tačke.

```
9.81
```

Za veoma velike ili veoma male brojeve, možete koristiti naučnu notaciju tako što ćete dodati *e* (za *eksponent*), i nakon toga napisati eksponent broja.

```
2.998e8
```

To je 2.998 × 108 = 299.800.000.

Izračunavanja sa celim brojevima (engl. *integers*) manjim od gorepomenutih 9 bilijardi, garantovano će uvek biti precizna. Nažalost, izračunavanja sa decimalnim brojevima obično nisu. Kao što π (pi) ne može biti precizno izražen konačnim brojem decimalnih mesta, mnogi brojevi gube preciznost kada je samo 64 bita dostupno za njihovo skladištenje. To je šteta, ali izaziva praktične probleme samo u određenim situacijama. Bitna stvar je da budete svesni tog ograničenja i da decimalne brojeve tretirate kao približne vrednosti, a ne precizne.

Aritmetika

Glavna stvar koju ćete raditi s brojevima jeste artimetika. Aritmetičke operacije kao što je sabiranje ili množenje uzimaju dve brojčane vrednosti i od njih proizvode nov broj. Evo kako to izgleda u JavaScriptu:

```
100 + 4 * 11
```

Znaci + i * su *operatori*. Prvi predstavlja sabiranje, a drugi množenje. Postavljanje operatora između dve vrednosti primeniće tu operaciju na vrednosti i proizvesti novu vrednost.

Ali da li ovaj primer znači „saberi 4 i 100, i rezultat pomnoži brojem 11“ ili se množenje obavlja pre sabiranja? Kao što ste i pretpostavljali, množenje se dešava prvo. Ipak, kao i u matematici, to možete promeniti obavijajući sabiranje zagradama.

```
(100 + 4) * 11
```

Za oduzimanje se koristi operator -, a deljenje se može obaviti operatorom /.

Kada se operatori javljaju zajedno bez zagrada, redosled kojim se primenjuju određen je *prioritetom* operatora. Prethodni primer je pokazao da množenje ide pre sabiranja. Operator / ima isti prioritet kao *. Isto važi i za + i -. Kada je više operatora sa istim prioritetom jedan uz drugi, na primer, `1 - 2 + 1`, oni se primenjuju sleva udesno: `(1 - 2) + 1`.

Ta pravila prioriteta nisu nešto o čemu treba da brinete. Kada ste u nedoumici, samo dodajte zagrade.

Postoji još jedan aritmetički operator koji možda nećete odmah prepoznati. Znak `%` se koristi za predstavljanje *ostatka*. `X % y` jeste ostatak deljena broja `X` brojem `Y`. Na primer, `314 % 100` daje rezultat `14`, a `144 % 12` daje 0. Prioritet operatora za ostatak isti je kao za množenje i deljenje. Ovaj operator često se naziva i *modulo*.

Posebni brojevi

U JavaScriptu postoje tri posebne vrednosti koje se smatraju brojevima, ali se ne ponašaju kao obični brojevi.

Prve dve su su `Infinity` i `-Infinity` i one predstavljaju pozitivnu i negativnu beskonačnost. `Infinity - 1` je i dalje `Infinity`, itd. Ipak, nemojte biti previše poverljivi prema izračunavanju zasnovanom na beskonačnosti. Ono nije matematički ispravno i brzo će dovesti do narednog posebnog broja: `NaN`.

`NaN` predstavlja „nije broj", iako tip te vrednosti *jeste* broj. Taj rezultat ćete dobiti, na primer, kada pokušate da izračunate `0 / 0` (deljenje nule nulom), `Infinity - Infinity`, ili bilo koju od više numeričkih operacija koje ne daju smislen rezultat.

Znakovni nizovi

Naredni osnovni tip podataka jeste *znakovni niz* (engl. *string*). Znakovni nizovi se koriste za predstavljanje teksta. Oni se pišu postavljanjem njihovog sadržaja između navodnika.

```
`Down on the sea`
"Lie on the ocean"
'Float on the ocean'
```

Možete koristiti polunavodnike, navodnike ili obrnute polunavodnike da biste označili znakovne nizove, samo je bitno da znak za navođenje na početku i na kraju znakovnog bude isti.

Gotovo sve se može postaviti između navodnika, a JavaScript će od toga napraviti vrednost znakovnog niza. Međutim, neki znaci su teži. Možete zamisliti koliko bi teško bilo postaviti navodnike između navodnika. Znak za *novi red* (engl. *newline*) koji dobijete kada pritisnete E N T E R, može se dodati bez korišćenja izlazne sekvence za znakove samo ako znakovni niz obuhvatite obrnutim polunavodnicima (`).

Da biste takve znakove uključili u znakovni niz, koristićete narednu notaciju: kad god se u tekstu nalazi obrnuta kosa crta (\), ona označava da znak nakon nje ima posebno značenje. To je *izlazna sekvenca* za znak. Ako pre navodnika postavite obrnutu kosu crtu, nećete završiti znakovni niz, već će navodnik biti njegov deo. Kada se nakon obrnute kose crte pojavi znak `n`, on se tumači kao nov red. Slično tome, `t` nakon obrnute kose crte predstavlja znak za tabulator. Pogledajte naredni znakovni niz:

```
"Ovo je prvi red\nA ovo je drugi"
```

Tekst koji on sadrži je ovaj:

```
Ovo je prvi red
A ovo je drugi
```

Naravno, postoje situacije kada ćete želeti da obrnuta kosa crta u znakovnom nizu bude samo obrnuta kosa crta, a ne poseban kod. Ukoliko dve obrnute kose crte slede jedna za drugom, one će se sažeti i ostaće samo jedna u rezultujućoj vrednosti znakovnog niza. Evo kako možete prikazati znakovni niz `"Znak za nov red zapisuje se kao "\n"."`:

```
"Znak za nov red zapisuje se kao \"\\n\"."
```

I znakovni nizovi moraju da budu oblikovani kao nizovi bitova da bi mogli da postoje u računaru. Način na koji JavaScript to radi zasnovan je na standardu *Unicode*. Taj standard dodeljuje broj svakom znaku koji bi vam ikada mogao zatrebati, uključujući i znakove iz grčkog jezika, arapskog, japanskog, jermenskog itd. Ukoliko imamo broj za svaki znak, znakovni niz može biti opisan kao sekvenca brojeva.

I to je ono što JavaScript radi. Ali tu postoji komplikacija: JavaScript za predstavljanje koristi 16 bitova po znakovnom elementu, pa može da opiše 2^{16} različitih znakova. Međutim, Unicode definiše više znakova od toga – trenutno je to oko dvaput više. Znači, neki znakovi, kao što su mnogi emotikioni, zauzimaju do dve „znakovne pozicije“ u JavaScriptovim znakovnim nizovima. Vratićemo se tome u odeljku „Znakovni nizovi i kodovi znakova“, na strani 87.

Znakovni nizovi se ne mogu deliti, množiti ili oduzimati, ali operator `+` *može* se koristiti na njima. On ne samo da ih sabira, već ih *nadovezuje* (engl. *concatenate*) – lepi dva znakovna niza jedan za drugi. Naredni red će proizvesti znakovni niz `"concatenate"`:

```
"con" + "cat" + "e" + "nate"
```

Postoji više funkcija (*metoda*) koje se mogu koristiti da bi se na vrednostima tipa znakovni niz izvele druge operacije. O njima ću više reći u odeljku „Metode“ na strani 60.

Znakovni nizovi napisani sa polunavodnicima ili navodnicima ponašaju se isto – razlika je samo u tome koji tip navodnika želite da upotrebite unutar samog znakovnog niza. Znakovni nizovi postavljeni u obrnute polunavodnike obično se nazivaju *šablonski literali* (engl. *template literals*) i mogu da urade još neke trikove. Osim što mogu da se prostiru u više redova, u njih možete ugraditi druge tipove vrednosti.

```
`pola od 100 je ${100 / 2}`
```

Kada u šablonskom literalu nešto napišete unutar `${}`, rezultat toga će biti izračunat, pretvoren u znakovni niz i dodat na to mesto. Gornji primer daje `pola od 100 je 50`.

Unarni operatori

Nisu svi operatori simboli. Neki se pišu kao reči. Jedan primer je operator `typeof`, koji vraća tip vrednosti koju ste mu prosledili.

```
console.log(typeof 4.5)
// → number console.log(typeof "x")
// → string
```

U ovom primeru koda koristimo `console.log` da bismo naznačili da želimo da vidimo rezultat neke procene. Više o tome u narednom poglavlju.

Ostali prikazani operatori radili su sa dve vrednosti, ali `typeof` uzima samo jednu. Operatori koji koriste dve vrednosti nazivaju se *binarni* operatori, dok se oni koji uzimaju jednu nazivaju *unarni* operatori. Operator minus može se koristiti i kao binaran i kao unaran operator.

```
console.log(- (10 – 2))
// → -8
```

Bulove vrednosti

Često je korisno imati tip vrednosti koji pravi razliku između dve mogućnosti, kao što su „da“ i „ne“ ili „uključeno“ i „isključeno“. Za tu namenu, JavaScript ima tip *Boolean* (Bulove, logičke vrednosti), koji ima samo dve vrednosti, tačno – *true* i netačno – *false.*

Poređenje

Evo jednog načina da se proizvedu Bulove vrednosti:

```
console.log(3 > 2)
// → true console.log(3 < 2)
// → false
```

Znakovi `>` i `<` su tradicionalni simboli za „veće od“ i „manje od“. Oni su binarni operatori. Njihova primena za rezultat ima Bulovu vrednost koja označava da li je izraz tačan u datom slučaju.

I znakovni nizovi se mogu porediti na isti način.

```
console.log("Aardvark" < "Zoroaster")
// → true
```

Način na koji su znakovni nizovi poređani otprilike je alfabetski, ali nije baš onakav kakav biste videli u rečniku: velika slova su uvek „manja“ nego mala slova, pa je `"Z" < "a"`, a nealfabetski znakovi (!, – itd) su takođe uključeni u redosled. Kada poredi znakovne nizove, JavaScript prolazi kroz znakove sleva udesno, poredeći Unicode kodove jedan po jedan.

Drugi slični znakovi su `>=` (veće ili jednako), `<=` (manje ili jednako), `==` (jednako) i `!=` (različito).

```
console.log("Itchy" != "Scratchy")
// → true
console.log("Apple" == "Orange")
// → false
```

Postoji samo jedna vrednost u JavaScriptu koja nije jednaka sama sebi, a to je `NaN` („nije broj" – engl. „*not a number*").

```
console.log(NaN == NaN)
// → false
```

`NaN` bi trebalo da označava da nema rezultata u računanju koje nema smisla i taj rezultat, nije jednak ni jednom *drugom* rezultatu bilo kog izračunavanja koje nema smisla.

Logički operatori

Postoje neke operacije koje se mogu primenjivati na same Bulove vrednosti. JavaScript podržava tri logička operatora: *i*, *ili* i *ne*. Oni se mogu koristiti za „rasuđivanje" o Bulovim vrednostima.

Operator `&&` predstavlja logičko *i*. To je binarni operator i njegov rezultat je *true* (tačno) samo ako su obe date vrednosti *true*.

```
console.log(true && false)
// → false
console.log(true && true)
// → true
```

Operator `||` označava logičko *ili*. On daje rezultat *true* ako je bilo je koja dodeljena vrednost *true*.

```
console.log(false || true)
// → true
console.log(false || false)
// → false
```

Ne se piše kao znak uzvika (`!`). To je unarni operator koji obrće vrednost koja mu je data — `!true` daje `false`, a `!false` daje `true`.

Pri mešanju Bulovih operatora sa aritmetičkim i drugim operatorima, nije uvek očigledno kada su vam potrebne zagrade. U praksi je obično dovoljno znati da, od operatora koje ste dosad videli, `||` ima najniži prioritet, sledi `&&`, pa operatori poređenja (`>`, `==` itd) i onda ostali. Taj redosled je izabran tako da vam je, u tipičnim izrazima kao što je naredni, potrebno što je moguće manje zagrada:

```
1 + 1 == 2 && 10 * 10 > 50
```

Poslednji logički operator o kojem ću govoriti nije unarni, ni binarni, već je *ternarni*, i radi sa tri vrednosti. Piše se sa znakom pitanja i dvotačkom, ovako:

```
console.log(true ? 1 : 2);
// → 1
console.log(false ? 1 : 2);
// → 2
```

Naziva se *uslovni* (engl. *conditional*) operator (ili prosto *ternarni* operator, jer je jedini takav operator u jeziku). Vrednost levo od upitnika „bira“ koja će od druge dve vrednosti biti rezultat. Kada je vrednost levo od upitnika *true*, ona za rezultat bira srednju vrednost, a kada je *false*, rezultat je desna vrednost.

Prazne vrednosti

Postoje dve posebne vrednosti, zapisuju se kao `null` i `undefined`, koje se koriste za označavanje nedostatka *smislene* vrednosti. One same za sebe jesu vrednosti, ali ne sadrže nikakve informacije.

Mnoge operacije u jeziku koje ne proizvode smislenu vrednost (kasnije ćete videti neke) za rezultat imaju `undefined` prosto zato što moraju da daju *neku* vrednost.

Razlika u značenju između `undefined` i `null` je slučajnost JavaScriptovog dizajna, i u većini slučajeva nije bitna. U situacijama kada treba da se bavite tim vrednostima, preporučujem da ih tretirate kao vrednosti koje uglavnom mogu zamenjivati jedna drugu.

Automatska konverzija tipa

U uvodu sam pomenuo da JavaScript daje sve od sebe da bi prihvatio gotovo svaki program koji mu date, čak i programe koji rade čudne stvari. To je fino prikazano u narednim izrazima:

```
console.log(8 * null)
// → 0
console.log("5" - 1)
// → 4
console.log("5" + 1)
// → 51
console.log("five" * 2)
// → NaN
console.log(false == 0)
// → true
```

Kada je operator primenjen na „pogrešan“ tip vrednosti, JavaScript će tiho pretvoriti tu vrednost u tip koji mu treba, koristeći skup pravila koja često nisu ono što biste očekivali. To se naziva *konverzija tipa* (engl. *type coercion*). Vrednost `null` u prvom izrazu postaje `0`, a `"5"` u drugom izrazu postaje `5` (umesto znakovnog niza postaje broj). U trećem izrazu, `+` pokušava da nadoveže znakovne nizove pre nego da sabere brojeve, pa je `1` pretvoreno `"1"` (umesto broja postaje znakovni niz).

Kada se u broj pretvara nešto što ne odgovara broju na očigledan način (kao što je `"five"` ili `undefined`), dobićete vrednost `NaN`. Dalje aritmetičke operacije nad vrednošću `NaN` i dalje za rezultat imaju `NaN`, pa ako se nađete u nekoj od tih neočekivanih situacija, potražite nenamerne konverzije tipova.

Kada poredite vrednosti istog tipa koristeći `==`, rezultat je lako predvideti: treba da dobijete *true* kada su obe vrednosti iste, osim u slučaju `NaN`. Međutim, kada se tipovi razlikuju, JavaScript koristi složen i zbunjujuć skup pravila da bi utvrdio šta da radi. U većini slučajeva, prosto pokušava da konvertuje jednu od vrednosti u tip druge vrednosti. Međutim, kada se `null` ili `undefined` javljaju na bilo kojoj strani operatora, rezultat će biti *true* samo ako je i na drugoj strani operatora vrednost `null` ili `undefined`.

```
console.log(null == undefined);
// → true
console.log(null == 0);
// → false
```

Takvo ponašanje je često korisno. Kada hoćete da testirate da li je neka vrednost prava vrednost, a ne `null` ili `undefined`, možete da je uporedite sa `null` koristeći operator `==` (ili `!=`).

Ali šta ako hoćete da proverite da li se nešto odnosi baš na vrednost `false`? Izrazi kao što su `0 == false` i `"" == false` su *true*. Kada *ne* želite da dođe do automatske konverzije tipa, postoje dva dodatna operatora: `===` i `!==`. Prvi proverava da li je vrednost *precizno* jednaka drugoj, a drugi proverava da li nije precizno jednaka. Tako `"" === false` ima rezultat *false* kao što se i očekuje.

Preporučujem da troznačne operatore za poređenje koristite u odbrambene svrhe, da biste sprečili neočekivane konverzije tipa da vas sapletu. Međutim, kada ste sigurni da su tipovi sa obe strane isti, nema problema s korišćenjem kraćih operatora.

Skraćeno izračunavanje logičkih operatora

Logički operatori `&&` i `||` na čudan način izlaze na kraj s vrednostima različitog tipa. Pretvoriće vrednost sa njihove leve strane u Bulov tip da bi odlučili šta da rade, ali zavisno od operatora i rezultata te konverzije, oni će vratiti ili *originalnu* levu vrednost ili desnu vrednost.

Operator `||`, na primer, vraća vrednost koja mu stoji na levoj strani kada se ona može pretvoriti u *true*, a inače vraća vrednost na desnoj strani. To ima očekivano dejstvo kada su vrednosti Bulove, a radi nešto analogno za vrednosti drugih tipova.

```
console.log(null || "user")
// → user
console.log("Agnes" || "user")
// → Agnes
```

Tu funkcionalnost možemo koristiti kao način da se vratimo na unapred zadatu vrednost. Ako imate vrednost koja bi mogla biti prazna, možete nakon nje postaviti `||` sa zamenskom vrednošću. Ukoliko početna vrednost može biti konvertovana u *false*, umesto nje ćete dobiti zamensku vrednost. Pravila

za konvertovanje znakovnih nizova i brojeva u Bulove vrednosti kažu da se `0`, `NaN` i prazan znakovni niz (`""`) računaju kao `false`, a sve ostale vrednosti se računaju kao `true`. Znači `0 || -1` daje rezultat `-1`, a `"" || "!?"` daje rezultat `"!?"`.

Operator `&&` radi slično, ali obrnuto. Kada je vrednost na njegovoj levoj strani nešto što se pretvara u *false*, on vraća tu vrednost, a inače vraća vrednost koja mu stoji desno.

Drugo bitno svojstvo ova dva operatora jeste da se deo koji im stoji desno procenjuje samo kada je to neophodno. U slučaju `true || X`, ma šta da je `X` – čak i ako je to deo programa koji radi nešto *užasno* – rezultat će biti *true*, i `X` neće biti procenjivano. Isto važi i za `false && X`, što je *false* i zanemaruje `X`. To se naziva *skraćeno izračunavanje* (engl. *short-circuit evaluation*).

Uslovni operator funkcioniše na sličan način. Između druge i treće vrednosti, procenjuje se samo ona koja je izabrana.

Rezime

U ovom poglavlju smo razmotrili četiri tipa JavaScriptovih vrednosti: brojeve, znakovne nizove, Bulove vrednosti i nedefinisane vrednosti.

Takve vrednosti se prave upisivanjem njihovog imena (`true`, `null`) ili vrednosti (`13`, `"abc"`). Možete kombinovati i transformisati vrednosti pomoću operatora. Videli smo binarne operatore za aritmetiku (`+`, `-`, `*`, `/` i `%`), operatore za nadovezivanje znakovnih nizova (`+`), poređenje (`==`, `!=`, `===`, `!==`, `<`, `>`, `<=`, `>=`) i logičke operatore (`&&`, `||`), kao i nekoliko unarnih operatora (`-` za negaciju broja, `!` za logičku negaciju i `typeof` za pronalaženje tipa vrednosti) i jedan ternarni operator (`?:`) za biranje jedne od dve vrednosti na osnovu treće vrednosti. Time ste dobili dovoljno informacija da biste JavaScript koristili kao džepni kalkulator, ali ne i za nešto više od toga. Naredno poglavlje počinje da povezuje ove izraze u osnovne programe.