POGLAVLJE 2

Leksička struktura

Leksička struktura programskog jezika je skup elementarnih pravila koja određuje kako pišete programe na tom jeziku. To je sintaksa jezika na najnižem nivou: određuje kako izgledaju imena promenljivih, znakovi razgraničenja za komentare i kako se jedna programska naredba odvaja od sledeće. Ovo kratko poglavlje dokumentuje leksičku strukturu JavaScripta. Pravilo obuhvata:

- Razlikovanje veličine slova, razmaka i kraja linija
- Komentare
- Literale
- Identifikatore i rezervisane reči
- Unicode
- Opciono tačku sa zapetom

2.1 Tekst JavaScript programa

JavaScript je jezik koji razlikuje velika i mala slova. To znači da rezervisane reči jezika, promenljive, imena funkcija i drugi *identifikatori* jezika uvek moraju biti napisani sa doslednom veličinom slova. Na primer, rezervisana reč (engl. *keyword*) `while` mora da se upiše „while„, a ne „While„ ili „WHILE„. Slično tome, `online`, `Online`, `OnLine` i `ONLINE` su četiri različita imena promenljivih.

JavaScript zanemaruje razmake koji se pojavljuju između tokena, tj. leksema (engl. *token*) u programima. JavaScript takođe uglavnom ignoriše krajeve redova (ali za izuzetak pogledajte §2.6). Budući da znakove za razmak i znakove za novi red možete slobodno koristiti u svojim programima, možete ih upotrebiti da formatirate i uvlačete redove na uredan i dosledan način što olakšava čitanje i razumevanje koda.

Pored znaka za razmak (`\u0020`), JavaScript prepoznaje i tabove, razne ASCII kontrolne znakove i razne znakove Unicoda kao znak za razmak. JavaScript prepoznaje nove redove, povratak na početak reda (engl. *carriage returns*) i par povratak na početak reda/novi red (engl. *carriage return/line feed*) kao završetak reda.

2.2 Komentari

JavaScript podržava dva stila komentara. Bilo koji tekst između // i kraja reda tretira se kao komentar i JavaScript ga ignoriše. Bilo koji tekst između znakova /* i */ takođe se tretira kao komentar; ovi komentari mogu da obuhvataju više linija, ali ne mogu da se ugnezde. Sledeći redovi koda su sve važeći JavaScript komentari:

```
// Ovo je komenatar u jednom redu

/* I ovo je komentar */

/*
 * Ovo je višeredni komentar. Dodatni znak * nije potreban na početku
 * svakog reda, ali izgleda lepše!
 */
```

2.3 Literali

Literal je doslovna vrednost koja se pojavljuje direktno u programu. Slede svi literali:

```
12              // Broj dvanaest
1.2             // Broj jedan tačka dva (jedan zapeta dva)
"Zdravo svima"  // Niz znakova
'Hej'           // Drugi niz znakova
True            // Bulova vrednost
False           // Druga Bulova vrednost
Null            // Prazno, bez objekta
```

Kompletni detalji o numeričkim i znakovnim literalima nalaze se u poglavlju 3.

2.4 Identifikatori i rezervisane reči

Identifikator (engl. *identifier*) je jednostavno ime. U JavaScriptu se identifikatori koriste za imenovanje konstanti, promenljivih, svojstava, funkcija i klasa i za obezbeđivanje oznaka za određene petlje u JavaScript kodu. JavaScript identifikator mora početi slovom, znakom podvlačenja (_) ili znakom dolara ($). Naredni znakovi mogu biti slova, cifre, znaci podvlačenja ili oznaka dolara. (Brojke nisu dozvoljene kao prvi znak, tako da JavaScript može lako razlikovati identifikatore od brojeva.) Sve su to legalni identifikatori:

```
i
ime_moje_variable
v13
_ostalo
$str
```

Kao i svaki jezik, JavaScript zadržava određene identifikatore za sopstvenu upotrebu. Ove „rezervisane reči“ ne mogu se koristiti kao važeći identifikatori. Oni su navedeni u sledećem odeljku.

2.4.1 Rezervisane reči

Sledeće reči su deo JavaScript jezika. Mnoge od ovih (kao što su `if`, `while` i `for`) su rezervisane reči koje se ne smeju koristiti kao imena konstanti, promenljivih, funkcija ili klasa (mada se mogu koristiti kao nazivi svojstava unutar objekta). Ostali (kao što su `from`, `of`, `get` i `set`) se koriste u ograničenom kontekstu bez sintaktičke nejasnoće i savršeno su važeći kao identifikatori. Ostale ključne reči (poput `let`) ne mogu se u potpunosti rezervisati da bi se zadržala kompatibilnost unazad sa starijim programima, pa postoje složena pravila koja regulišu kada se mogu koristiti kao identifikatori i kada ne mogu. (`let` se koristi kao naziv promenljive ako se, na primer, deklariše sa `var` izvan klase, ali ne i ako je deklarisano unutar klase ili sa `const`.) Najjednostavniji kurs je izbegavanje korišćenja bilo koje od ovih reči kao identifikatora, osim `from`, `set` i `target`, koji su bezbedni za upotrebu i koji su već u uobičajenoj upotrebi.

```
as      const     export    get         null    target   void
async   continue  extends   if          of      this     while
await   debugger  false     import      return  throw    with
break   default   finally   in          set     true     yield
case    delete    for       instanceof  static  try
catch   do        from      let         super   typeof
class   else      function  new         switch  var
```

JavaScript takođe zadržava ili ograničava upotrebu određenih ključnih reči koje jezik trenutno ne koristi, ali koje bi se mogle koristiti u budućim verzijama:

```
enum  implements  interface  package  private  protected  public
```

Iz istorijskih razloga, `arguments` i `eval` nisu dozvoljeni kao identifikatori u određenim okolnostima i najbolje ih je u potpunosti izbegavati.

2.5 Unicode

JavaScript programi su napisani pomoću Unicode skupa znakova, a možete koristiti bilo koje Unicode znakove u znakovnim nizovima i komentarima. Za prenosivost i lakoću uređivanja, uobičajeno je da se u identifikatorima koriste samo ASCII slova i cifre. Ali ovo je samo programska konvencija i jezik omogućava Unicode slova, cifre i ideografije (ali ne i emoje) u identifikatorima. To znači da programeri mogu koristiti matematičke simbole i reči iz neengleskih jezika kao konstante i promenljive:

```
const π = 3.14;
const sí = true;
```

2.5.1 Escape Unicode nizovi

Neki računarski hardver i softver ne može prikazati, uneti ili pravilno obraditi kompletan skup Unicode znakova. Da bi podržao programere i sisteme koji koriste stariju tehnologiju, JavaScript definiše Esc sekvence koje nam omogućavaju da pišemo Unicode znakove koristeći samo ASCII znakove. Ovi Unicode nizovi započinju znakom `\u` i prate ga tačno četiri heksadecimalne cifre (koristeći velika ili mala slova A – F) ili jedna do šest heksadecimalnih

cifara unutar vitičastih zagrada. Ovi Unicode nizovi mogu se pojaviti u JavaScript literalima, regularnim izrazima i identifikatorima (ali ne u rezervisanim rečima jezika). Unicode niz za znak „é“, na primer, je `\u00E9`; evo tri različita načina za pisanje imena promenljive koja uključuju ovaj znak:

```
let café = 1; // Definisanje promenljive pomoću Unicode znaka
caf\u00e9     // => 1; pristup promenljivoj pomoću escape niza
caf\u{E9}     // => 1; drugi oblik istog escape niza
```

Rane verzije JavaScripta podržale su samo četvorocifreni escape nizove. Verzija s vitičastima zagradama uvedena je u ES6 kako bi bolje podržao Unicode za koji je potrebno više od 16 bita, poput emojija:

```
console.log("\u{1F600}");  // Daje emoji nasmejano lice
```

Escape Unicode nizovi mogu se pojaviti i u komentarima, ali pošto se komentari zanemaruju, u tom kontekstu se jednostavno tretiraju kao ASCII znakovi i ne tumače se kao Unicode.

2.5.2 Normalizacija Unicoda

Ako koristite ne-ASCII znakove u svojim JavaScript programima, morate biti svesni da Unicode dopušta više od jednog načina kodiranja istog znaka. Znak „é“, na primer, može se kodirati kao jedinstveni Unicode znak `\u00E9` ili kao običan ASCII „e“, praćen znakom akutnog akcenta koji kombinuje znak `\u0301`. Ova dva kodiranja obično izgledaju potpuno isto kada ih prikaže editor teksta, ali imaju različite binarne kodove, što znači da ih JavaScript smatra različitim, što može dovesti do vrlo zbunjujućih programa:

```
const café = 1;  // Ova konstanta se zove "caf\u{e9}"
const café = 2;  // Ovo je drugačija konstanta: "cafe\u{301}"
café             // => 1: ova konstanta ima jednu vrednost
café             // => 2: ova, iako se ne razlikuje, ima drugu vrednost
```

Unicode standard definiše preferisano kodiranje svih znakova i određuje postupak normalizacije za pretvaranje teksta u kanonski oblik pogodan za upoređivanje. JavaScript podrazumeva da je kod koji tumači već normiran i *ne vrši* nikakvu svoju normalizaciju. Ako planirate da koristite Unicode znakove u svojim JavaScript programima, trebalo bi da obezbedite da vaš editor ili neki drugi alat izvodi Unicode normalizaciju vašeg izvornog koda kako biste sprečili da završite sa različitim, ali vizuelno istim identifikatorima.

2.6 Neobavezni znaci tačka sa zapetom

Kao i mnogi programski jezici, JavaScript koristi znak tačka sa zapetom (engl. *semicolon*) (`;`) za odvajanje naredbi (vidi poglavlje 5) jedne od druge. Ovo je važno za postizanje jasnog značenja vašeg koda: bez znaka razdvajanja, kraj jedne naredbe može se činiti početkom sledeće, ili obratno. U JavaScriptu obično možete izostaviti tačku sa zapetom između dve naredbe ako su te naredbe napisane u zasebnim redovima. (Takođe možete izostaviti tačku sa zapetom na kraju programa ili ako je sledeći token u programu zatvarajuća vitičasta zagrada: `}`.) Mnogi JavaScript programeri (i kod u ovoj knjizi) koriste tačku sa zapetom da izričito obeleže krajeve naredbi, čak i tamo gde nije potrebno. Drugi stil je izostavljanje tačke sa

zapetom kad god je to moguće, koristeći ih samo u retkim situacijama. Bez obzira koji stil izaberete, treba da razumete nekoliko detalja o neobaveznim znacima tačka sa zapetom u JavaScriptu.

Razmotrite sledeći kod. Obzirom da se dve naredbe pojavljuju u zasebnim redovima, prva tačka sa zapetom se može izostaviti:

```
a = 3;
b = 4;
```

Napisano ovako, prva tačka sa zapetom je potrebna:

```
a = 3; b = 4;
```

Imajte na umu da JavaScript ne tretira svaki kraj reda kao tačku sa zapetom: obično tretira kraj reda kao tačku sa zapetom samo ako ne može da raščlani (engl. *parse*) kod bez dodavanja implicitnog znaka tačka sa zapetom. Formalnije (i uz tri izuzetaka opisana malo kasnije), JavaScript tretira kraj reda kao tačku sa zapetom ako se sledeći znak koji nije prazan ne može protumačiti kao nastavak trenutne naredbe. Razmotrite sledeći kod:

```
let a
a
=
3
console.log(a)
```

JavaScript tumači ovaj kod ovako:

```
let a; a = 3; console.log(a);
```

JavaScript tretira prelom prvog reda kao tačku sa zapetom jer ne može raščlaniti kod `let a` bez tačke sa zapetom. Drugi `a` može stajati samo kao naredba `a;`, ali JavaScript ne tretira drugi kraj red kao tačku sa zapetom jer može nastaviti sa raščlanjivanjem duže naredbe `a = 3;`.

Ova pravila o kraju naredbi dovodi do nekih iznenađujućih slučajeva. Ovaj kod izgleda kao dve odvojene naredbe odvojene novim redom:

```
let y = x + f
(a+b).toString()
```

Zagrade u drugom redu koda mogu se protumačiti kao poziv funkcije `f` iz prve linije, a JavaScript interpretira kod ovako:

```
let y = x + f(a+b).toString();
```

Verovatno ovo nije tumačenje koje je autor koda hteo. Da bi ovo radilo kao dve odvojene naredbe, u ovom slučaju je potrebna eksplicitna tačka sa zapetom.

Generalno, ako naredba započinje sa `(`, `[`, `/`, `+` i `-`, postoji šansa da se ona može protumačiti kao nastavak prethodne naredbe. Naredbe koje počinju sa `/`, `+` i `–` su u praksi prilično retke, ali naredbe koje počinju sa `(` i `[` uopšte nisu neuobičajene, bar u nekim stilovima JavaScript programiranja. Neki programeri vole da stave odbrambenu tačku sa zapetom na početak svake takve naredbe tako da će kod i dalje raditi ispravno čak i ako je prethodna naredba promenjena i prethodna tačka sa zapetom uklonjena:

```
let x = 0                          // tačka i zapeta izostavljeni ovde
;[x,x+1,x+2].forEach(console.log) // Odbrambena ; čini naredbu odvojenu
```

Postoje tri izuzetka od opšteg pravila da JavaScript interpretira krajeve redova kao tačku sa zapetom, kada drugi red ne može analizirati kao nastavak naredbe u prethodnom redu. Prvi izuzetak uključuje `return`, `throw`, `yield`, `break` i `continue` naredbe (vidi poglavlje 5). Ove naredbe često su samostalne, ali ih ponekad prati identifikator ili izraz. Ako se posle bilo koje od ovih reči (pre bilo kojeg drugog tokena) pojavi kraj reda, JavaScript će uvek kraj reda shvatiti kao tačku sa zapetom. Na primer, ako napišete:

```
return
true;
```

JavaScript smatra da mislite:

```
return; true;
```

A vi ste možda hteli:

```
return true;
```

To znači da ne smete umetati kraj reda između `return`, `break` ili `continue` i izraza koji prati ključnu reč. Ako umetnete kraj reda, verovatno će se kod izvršti na neočigledan način koji je teško shvatiti.

Drugi izuzetak uključuje operatore `++` i `--` (§4.8). Ovi operatori mogu biti operatori prefiksa koji se pojavljuju pre izraza ili postfiks operatori koji se pojavljuju posle izraza. Ako želite da koristite bilo koji od ovih operatora kao postfiks operatore, oni se moraju pojaviti u istoj liniji kao i izraz na koji se odnose. Treći izuzetak uključuje funkcije definisane pomoću sažete sintakse „strelica“: => strelica se mora pojaviti u istom redu kao i lista parametara.

2.7 Rezime

Ovo poglavlje pokazalo je kako se JavaScript programi pišu na najnižem nivou. Sledeće poglavlje nas vodi jedan korak više i uvodi osnovne tipove i vrednosti (brojevi, znakovne nizove i tako dalje) koji služe kao osnovne jedinice izračunavanja u JavaScript programima.