Rad sa tekstom

Tekst je, pored brojeva, svakako najznačajniji tip podatka koji se može koristiti prilikom izvršavanja JavaScript koda. U lekciji o tipovima podataka izneti su neki osnovni postulati rada sa tekstualnim podacima, dok će lekcija pred vama u potpunosti biti posvećena takvoj problematici. Ponovo će biti prikazana osnovna pravila za definisanje tekstualnih literala, ali će ovoga puta pažnja biti posvećena i različitim specifičnim scenarijima. Takođe, u nastavku lekcije biće ilustrovane i različite ugrađene metode JavaScript jezika, koje mogu biti i više nego korisne prilikom rada sa tekstualnim podacima.

Definisanje tekstualnih literala

Osnovna razlika koja tekstualne literale odvaja od numeričkih ili logičkih jeste upotreba navodnika. Stoga sledeći primer ilustruje definisanje promenljive sa tekstualnom vrednošću:

```
var x = "this is some text";
```

JavaScript je jezik sa dinamičkim tipovima podataka. Stoga on samostalno pokušava da utvrdi tip definisane vrednosti. Kada se neka vrednost pojavi između navodnika, baš kao u prikazanom primeru, JavaScript zna da je reč o tekstualnom podatku. Stoga on tip promenljive `x` postavlja na `string`:

```
var x = "this is some text";
console.log(typeof x);   //string
```

Za definisanje tekstualnih literala moguće je koristiti i jednostruke navodnike:

```
var x = 'this is some text';
```

Kao što je moguće videti, JavaScript je fleksibilan po pitanju navodnika koji se koriste za definisanje tekstualnih literala. Ipak nije moguće kombinovati jednostruke i dvostruke navodnike prilikom navođenja jednog literala:

```
var x = 'this is some text";   //Uncaught SyntaxError: Invalid
or unexpected token
```

Escape karakter

U nekim situacijama može se javiti potreba za definisanjem jednostrukih ili dvostrukih navodnika unutar tekstualnog literala:

```
var x = 'It's alright';
```

Naravno, prikazana naredba proizvodi grešku:

```
Uncaught SyntaxError: Unexpected identifier
```

Razlog dobijanja greške vrlo je očigledan. Za definisanje tekstualnog literala korišćeni su jednostruki navodnici, dok je takav navodnik naveden i unutar samog literala. Zbog toga JavaScript prevodilac ne zna gde je početak a gde kraj tekstualne vrednosti, te dolazi do pojave greške. Najjednostavniji način za rešavanje upravo prikazanog problema jeste korišćenje različitih navodnika za definisanje vrednosti od onih koji će se pojaviti unutar samog literala:

```
var x = "He is called 'Johnny'";  // single quotes inside
double quotes
var x = 'He is called "Johnny"';   // double quotes inside
single quotes
```

Unutar prve naredbe, tekstualni literal je definisan korišćenjem dvostrukih navodnika. Zbog toga je unutar literala bez problema moguće koristiti jednostruke navodnike. Obrnuta je situacija unutar druge naredbe. Literal je omeđen jednostrukim navodnicima, te je stoga unutar njega bez problema moguće koristiti dvostruke navodnike.

Ipak, postavlja se pitanje – šta ukoliko unutar literala želimo postaviti baš one navodnike koji su upotrebljeni i za njegovo definisanje. Ili šta ukoliko je potrebno unutar jednog tekstualnog literala definisati oba tipa navodnika:

```
var x = 'This is single quote: '. This is double quote: ".';
```

Prikazana naredba ilustruje inicijalizaciju promenljive tekstualnim literalom. Tekstualni literal je oivičen jednostrukim navodnicima. Unutar literala pojavljuju se i jednostruki i dvostruki navodnici. Naravno, ovakva naredba proizvešće grešku:

```
Uncaught SyntaxError: Unexpected identifier
```

Rešenje upravo prikazanog problema ogleda se u upotrebi takozvanog **Escape karaktera**:

```
var x = 'This is single quote: \'. This is double quote: ".';
```

Sada je unutar tekstualnog literala, pre jednostrukog navodnika, postavljen karakter obrnute kose crte (engl. *backslash*). Reč je o specijalnom karakteru koji je potrebno postaviti pre bilo kog karaktera koji unutar tekstualnog literala može *zbuniti* JavaScript izvršno okruženje. Nakon ove male izmene, prikazana naredba više neće proizvoditi grešku, a unutar promenljive x biće smeštena tekstualna vrednost:

```
This is single quote: '. This is double quote: ".
```

Pored upravo prikazane situacije u kojoj se obrnuta kosa crta koristi kako bi se unutar tekstualnog literala ispisali navodnici, ovaj karakter je moguće koristiti u još nekim situacijama, za postizanje specijalnih značenja (tabela 11.1).

Escape	Opis
\'	jednostruki navodnik
\"	dvostruki navodnik
\\	obrnuta kosa crta (engl. *backslash*)
\n	prelazak u novi red
\r	povratak na početak reda
\t	horizontalni tabulator

Tabela 11.1. Specijalni karakteri

Sve ovo praktično znači da se ispisivanje specijalnih karaktera unutar tekstualnih literala može obaviti baš kako je to navedeno unutar kolone Escape, tabele 11.1. Na primer, ukoliko je unutar tekstualnog literala potrebno ispisati obrnutu kosu crtu, dovoljno je uraditi:

```
var x = 'This is backslash character: \\';
```

Vrednost promenljive x nakon izvršavanja prikazane naredbe izgledaće ovako:

```
This is backslash character: \
```

Identična je situacija i sa ostalim specijalnim karakterima:

```
var x = 'Hello\nWorld!';
```

Prikazani tekstualni literal stvoriće sledeći efekat:

```
Hello
World!
```

Pitanje

Za prelazak u novi red unutar nekog stringa dovoljno je definisati:

- \'
- \"
- \\
- **\n**

Objašnjenje:

Specijalni skup karaktera \n koristi se za prelamanje reda unutar nekog stringa.

Nadovezivanje stringova

JavaScript omogućava nadovezivanje većeg broja tekstualnih literala i to korišćenjem operatora sabiranja. Drugim rečima, korišćenjem operatora sabiranja nad tekstualnih vrednostima obavlja se proces nadovezivanja koji se drugačije naziva konkatenacija:

```
var x = "Hello " + "World!";
```

U prikazanoj naredbi su definisana dva tekstualna literala. Drugi je nadovezan na prvi, korišćenjem operatora sabiranja. Ovakav pristup se veoma često koristi kada je potrebno definisati dugačku tekstualnu vrednost. Zbog svoje dužine, ona može biti nepregledna, pa se pribegava prelamanju teksta u više redova, pri čemu se redovi nadovezuju operatorom sabiranja:

```
var x = "Lorem ipsum dolor sit amet, consectetur " +
"adipiscing elit. Praesent fermentum, nibh ac lacinia " +
"malesuada, est neque, ullamcorper finibus ac ipsum.";
```

String – objektni omotač primitivnog tipa

U prethodnoj lekciji prvi put je spomenut pojam objektnih omotača primitivnih tipova. I `string` tip podatka poseduje svoj objektni omotač, koji se sasvim očekivano naziva `String`.

Eksplicitno kreiranje `String` objekta može se postići korišćenjem istoimene konstruktorske funkcije i ključne reči `new`:

```
var stringPrimitive = 'some text';
var stringObject = new String(stringPrimitive);
```

Tip promenljive `stringObject` sada je `object`, što se lako može proveriti:

```
console.log(typeof stringObject); //object
```

Primitivna vrednost iz `String` objekata lako se može dobiti korišćenjem `valueOf()` metode:

```
console.log(typeof stringObject.valueOf()); //string
```

Prethodni redovi su bili samo demonstrativne prirode. Naime, nema nikakvog razloga da `String` objekat kreirate na prikazani način, odnosno eksplicitno, korišćenjem ključne reči `new` i konstruktorske funkcije. Uvek kada se javi potreba za `String` objektom konverziju primitivne vrednosti će za nas uraditi sam JavaScript. Naime, osnovni razlog za korišćenje `String` objekta jeste veliki broj ugrađenih svojstava i metoda koje je moguće koristiti nad tekstualnim vrednostima. Ipak, kao što je to bio slučaj i kod numeričkih primitivnih vrednosti, metode `String` objekta je moguće pozivati i direktno nad primitivnim vrednostima zato što će se JavaScript pobrinuti da se takve metode pozovu nad objektnim omotačem.

U nastavku lekcije biće ilustrovano korišćenje nekoliko svojstava i metoda koje su sastavni deo `String` objekta.

Transformisanje stringa

Pod pojmom transformisanja `stringa` misli se na pretvaranje svih slova u velika ili mala. Za obavljanje takvog posla `String` objekat poseduje dve metode:

- `toLowerCase()`
- `toUpperCase()`

Metoda `toLowerCase()` koristi se za pretvaranje svih slova nekog teksta u mala:

```
var x = "SOME TEXT";
console.log(x.toLowerCase());
```

Prikazane naredbe proizvode sledeći rezultat:

```
some text
```

Jasno je da su sva velika slova tekstualnog literala pretvorena u mala.

Metoda `toUpperCase()` koristi se za pretvaranje svih malih slova u velika:

```
var x = "some TEXT";
console.log(x.toUpperCase());
```

Ispis unutar konzole je sledeći:

```
SOME TEXT
```

Upravo prikazane metode, ali i sve ostale metode objekta `String` moguće je pozivati i direktno nad tekstualnim literalima:

```
console.log("SOME TEXT".toLowerCase()); //some text
```

Utvrđivanje dužine stringa

Objekat `String` poseduje i jedno veoma korisno svojstvo, koje je moguće koristiti za utvrđivanje dužine `stringa`:

```
var x = "some text".length;
```

Nakon tekstualnog literala napisan je kôd za pristup svojstvu `length`, čija vrednost odslikava dužinu teksta. Tako će, nakon izvršavanja prikazane naredbe, promenljiva `x` imati vrednost 9 (imajte na umu da se u dužinu jednog tekstualnog literala ubrajaju i razmaci).

Pristup pojedinačnim karakterima

Stringovi su u svojoj osnovi sačinjeni iz karaktera. Jedan `string` može imati jedan karakter ili više karaktera. Karakteri mogu biti slova, brojevi, znakovi interpunkcije, prazna mesta...

Objekat `String` omogućava pristup pojedinačnim karakterima jednog `stringa`.

Za obavljanje takvog posla koristi se metoda `charAt()`:

```
var x = "Hello World!";
console.log(x.charAt(6));
```

Rezultat ovakvog koda biće sledeći ispis unutar konzole:

```
W
```

Jasno je da je metodom `charAt()`, kojoj je prosleđena vrednost 6, dobijen karakter `W`. Vrednost koja se prosleđuje metodi `charAt()` predstavlja indeks karaktera unutar `stringa`.

Bitno je znati da indeksiranje karaktera započinje od 0.

Dobijanje isečka stringa

Nekada može biti veoma korisno izolovati samo jedan deo nekog `stringa`. To se može postići metodom `substring()`.

Njena sintaksa izgleda ovako:

```
substring(indexStart, indexEnd)
```

Metoda `substring()` može da prihvati dva parametra, pri čemu je samo prvi parametar obavezan:

- `indexStart` – indeks početnog karaktera, uključujući i karakter na definisanom indeksu,
- `indexEnd` – indeks poslednjeg karaktera, isključujući karakter na definisanom indeksu.

Sledeći primer ilustruje korišćenje metode `substring()` sa jednim ulaznim parametrom:

```
var x = "Hello World!";
console.log(x.substring(6));
```

U konzoli će nakon izvršavanja prikazanog koda da bude ispisano:

```
World!
```

S obzirom na to da je naveden samo početni indeks, završetak isečka je ujedno i završetak kompletnog `stringa`, pa upravo zbog toga ova metoda kao svoju povratnu vrednost emituje `World!`.

Ipak, ukoliko je potrebno izolovati reč `Hello`, neophodno je metodi `substring()` proslediti i drugi parametar:

```
var x = "Hello World!";
console.log(x.substring(0, 5));
```

Ovoga puta će unutar konzole biti ispisano:

```
Hello
```

Početak dobijenog isečka je prvi karakter originalnog `stringa` sa indeksom 0. Poslednji karakter dobijenog isečka ima indeks 4, zato što drugi parametar metode `substring()` predstavlja isključujući indeks. Poslednji karakter ima indeks 4, ali je on zapravo peti po redu, zato što indeksiranje karaktera unutar `stringova` započinje od 0.

Pronalazak teksta unutar stringa

Objekat `String` poseduje metodu koju je moguće koristiti za pronalazak nekog karaktera ili teksta unutar `string` literala.

Reč je o metodi `indexOf()`:

```
indexOf(searchValue)
```

Metoda `indexOf()` prihvata jedan parametar koji definiše frazu na osnovu koje će biti obavljena pretraga:

```
var someText = "The indexOf() method returns the position of
the first occurrence of a specified value in a string.";
console.log(someText.indexOf("method"));
```

Unutar upravo prikazanog primera prvo je deklarisana i inicijalizovana promenljiva sa nazivom `someText`. Za njenu vrednost je postavljen određeni tekst. Unutar druge naredbe, nad promenljivom `someText` poziva se metoda `indexOf()` i njoj se prosleđuje tekstualna vrednost *method*.

S obzirom na to da se tekst *method* nalazi unutar vrednosti promenljive `someText`, metoda `indexOf()` kao svoju povratnu vrednost emituje početni indeks na kome se nalazi tražena fraza:

```
14
```

Iz ovoga se može zaključiti da je prikazana pretraga uspešno obavljena i da prvi karakter tražene fraze unutar originalnog teksta ima indeks 14.

Kada metoda `indexOf()` ne pronađe traženi tekst, kao svoju povratnu vrednost ona emituje `-1`:

```
var someText = "The indexOf() method returns the position of
the first occurrence of a specified value in a string.";
console.log(someText.indexOf("function"));
```

Sada traženi tekst (*function*) ne postoji unutar teksta koji predstavlja vrednost promenljive `someText`. Zbog toga je povratna vrednost metode `indexOf()`:

```
-1
```

Na ovaj način možemo znati da se reč *function* ne pojavljuje unutar tekstualnog literala koji se pretražuje.

Trimovanje stringa

Trimovanje je pojam koji se odnosi na uklanjanje svih vrsta praznih karaktera (razmaci, uvlačenja, prelasci u novi red...) sa početka i kraja neke `string` vrednosti. `String` objekat poseduje nekoliko metoda kojima se može obaviti opisani posao:

- `trim()` – uklanja prazne karaktere sa početka i kraja `stringa`
- `trimStart()` – uklanja prazne karaktere sa početka `stringa`
- `trimEnd()` – uklanja prazne karaktere sa kraja `stringa`

Upotreba metode `trim()` može da izgleda ovako:

```
var someText = " Hello World! ";
console.log(someText.trim());
```

Tekst koji je definisan kao vrednost promenljive `someText` započinje i završava se praznim karakterima. Prilikom ispisivanja vrednosti ove promenljive unutar konzole, nad njom se poziva metoda `trim()` kako bi se takvi prazni karakteri uklonili:

```
Hello World!
```

Kao ispis unutar konzole dobija se tekst bez praznih karaktera.

Metode `trimStart()` i `trimEnd()` upotrebljavaju se na identičan način, ali svaka od njih ima svoj specifičan efekat. Metoda `trimStart()` uklanja prazne karaktere samo sa početka `stringa`:

```
var someText = "           Hello World!   ";
console.log(someText.trimStart());
```

Ispis je:

```
Hello World!
```

Iako se na prvi pogled ne primećuje, prazni karakteri na kraju `stringa` su i dalje tu, u šta je moguće uveriti se selektovanjem teksta.

Metoda `trimEnd()` uklanja prazne karaktere samo sa kraja `stringa`:

```
var someText = "           Hello World!   ";
console.log(someText.trimEnd());
```

Ispis u konzoli je:

```
Hello World!
```

Veoma je bitno znati da prikazane metode za trimovanje ni na koji način ne menjaju promenljivu nad kojom se pozivaju. Ona i dalje ostaje u svom originalnom obliku (sa praznim karakterima), dok se obrađeni `string` isporučuje kao povratna vrednost metoda za trimovanje.

Nadovezivanje stringova metodom concat()

Već više puta do sada je prikazan osnovni pristup za nadovezivanje tekstualnih vrednosti, koji podrazumeva korišćenje aritmetičkog operatora za sabiranje. Pored takvog pristupa, nadovezivanje `stringova` je moguće obaviti i korišćenjem ugrađene metode `String` objekta. Reč je o metodi `concat()`:

```
let txt1 = "Java";
let txt2 = "Script";
let txt3 = txt1.concat(txt2);
console.log(txt3);
```

U prikazanom primeru su prvo deklarisane i inicijalizovane dve promenljive tipa string.

Zatim je vrednost promenljive `txt3` formirana konkatenacijom vrednosti promenljivih `txt1` i `txt2`. Konkatenacija je obavljena korišćenjem metode `concat()`. Na ovaj način se unutar konzole dobija:

```
JavaScript
```

Primer – Parsiranje i obrada teksta

U nastavku će biti prikazan primer obrade tekstualnih podataka. Naime, primer će podrazumevati kreiranje programa, koji će imaginarni broj bankovnog računa iz pisane forme, prevoditi u elektronsku.

U pisanoj formi, bankovni račun može da izgleda ovako: `20123456789`

U elektronskoj formi isti bankovni račun treba da izgleda ovako: `2000000000123456789`

Razlika između dve prikazane forme se ogleda u sedam cifara `0`, koje su u elektronskoj formi dodate nakon prve dve cifre. Program treba da od korisnika preuzme bankovni račun u pisanoj formi i da od njega kreira elektronsku formu. Za obavljanje ovakvog posla, uneti broj računa je potrebno podeliti na dva dela, ubaciti sedam nula i sve delove spojiti u jednu celinu.

Rešenje može da izgleda ovako:

```
const INSERT = "0000000";

let bankAccount = prompt("Please enter bank account number:");

let firstPart = bankAccount.substr(0, 2);
let secondPart = bankAccount.substr(2, 9);

var bankAccountFinal = firstPart + INSERT + secondPart;
alert("Electronic form is " + bankAccountFinal);
```

Prvom naredbom kreira se jedna konstanta sa nazivom `INSERT`, unutar koje se čuva vrednost koju je potrebno umetnuti unutar bankovnog računa. Ova vrednost se neće menjati, pa je zbog toga ona deklarisana kao konstanta.

Drugom naredbom se od korisnika preuzima vrednost bankovnog računa. Vrednost se ne konvertuje u broj, kao što je to bio slučaj u prethodnim primerima, s obzirom na to da nam je sada potrebna vrednost u `string` formatu.

Naredne dve naredbe obavljaju izvlačenje delova bankovnog računa. Prva dva karaktera se smeštaju unutar promenljive `firstPart`, a ostatak unutar promenljive `secondPart`. Za obavljanje ovakvog posla korišćena je metoda `substr()`, koja omogućava dobijanje isečka `stringa`.

Pretposlednjom naredbom, korišćenjem konkatenacije, obavlja se objedinjavanje tri `string` vrednosti u jednu.

Poslednjom naredbom se obavlja prikaz konačne vrednosti.

Rezime

- Osnovna razlika koja tekstualne literale odvaja od numeričkih ili logičkih literala jeste upotreba navodnika.
- Tekstualne vrednosti se mogu definisati korišćenjem jednostrukih ili dvostrukih navodnika.
- Korišćenjem karaktera obrnute kose crte (engl. *backslah*), unutar stringa je moguće uvrstiti brojne specijalne karaktere, čije pojavljivanje je inače zabranjeno.
- JavaScript omogućava nadovezivanje većeg broja tekstualnih literala korišćenjem operatora sabiranja ili metode `concat()`.
- Objektni omotač primitivnog string tipa je objekat `String`.
- Metoda `toLowerCase()` koristi se za pretvaranje svih slova nekog teksta u mala.
- Metoda `toUpperCase()`koristi se za pretvaranje svih malih slova u velika.
- Objekat `String` poseduje svojstvo `length`, koje je moguće koristiti za utvrđivanje dužine stringa.
- Za pristup pojedinačnim karakterima jednog stringa moguće je koristiti metodu `charAt()`.
- Izolovanje jednog dela nekog stringa može se obaviti korišćenjem metode `substring()`.
- Da bi se utvrdilo da li neki tekst poseduje određeni karakter ili podtekst, moguće je koristiti metodu `indexOf()`.
- Za uklanjanje praznih karaktera sa početka i kraja stringa moguće je koristiti metodu `trim()`; ukoliko je isto potrebno obaviti samo sa početka ili samo sa kraja stringa, moguće je koristiti metode `trimStart()` i `trimEnd()`, respektivno.