Rad sa brojevima

U lekciji o tipovima podataka prvi put su prikazani osnovni postulati rada sa brojevima u JavaScript jeziku. U ovoj lekciji će problematici rada sa brojevima biti posvećena mnogo veća pažnja, pa će pored osnovnih biti ilustrovani i neki napredniji pristupi koji podrazumevaju korišćenje ugrađenih objekata i funkcija za rad sa brojevima.

Tipovi numeričkih podataka u JavaScriptu

Broj je jedan od prostih tipova u jeziku JavaScript. Iako većina jezika poseduje nekoliko različitih numeričkih tipova podataka (int, float, double…), JavaScript sve numeričke podatke predstavlja korišćenjem dva tipa:

- `number` – brojevi standardne preciznosti i
- `bigint` – celi brojevi povećane preciznosti.

number je osnovni JavaScript tip za predstavljanje brojeva. Definisanjem promenljive koja za vrednost ima neki numerički literal zapravo se dobija promenljiva tipa `number`.

bigint je tip podatka za predstavljanje celih brojeva povećane preciznosti.

Različiti oblici numeričkih literala

Prilikom formulisanja numeričkih literala moguće je koristiti jedan od nekoliko formata:

- celobrojni,
- decimalni,
- eksponencijalni,
- heksadecimalni,
- oktalni.

Celobrojni oblik – definiše broj bez decimala:

```
var x = 10;
var y = -338;
var z = 0;
```

Decimalni oblik – definiše broj sa decimalama:

```
var x = 56.00;
```

Eksponencijalni oblik – koristi se za predstavljanje suviše velikih ili suviše malih brojeva, odnosno za njihovo kompaktnije prikazivanje:

```
var y = 154e5;    // same as 15400000
```

```
var z = 154e-5;   // same as 0.00154
```

Heksadecimalni oblik – za razliku od dekadnog brojčanog sistema, omogućava prikaz brojeva sa bazom 16. Heksadecimalni zapis se predstavlja ciframa od 0 do 9 i slovima od A do F:

```
var x = 0xff;
var y = 0xccff;
```

Heksadecimalni literali uvek započinju karakterima **0x**, kako bi prevodilac znao da je namera da se definiše vrednost korišćenjem ove notacije. Stoga su u prikazanim primerima heksadecimalne vrednosti zapravo `ff` i `ccff`.

Prilikom formiranja heksadecimalnih literala karakteri nisu osetljivi na velika i mala slova, pa je tako karaktere od A do F moguće pisati i malim i velikim slovima.

Zanimljivost je da prilikom ispisa heksadecimalnih vrednosti, bilo u konzoli ili unutar HTML dokumenta, automatski dolazi do njihove konverzije u dekadni oblik:

```
var x = 0xCCFF;
console.log(x);
```

Prikazani kôd unutar konzole štampa vrednost 52479.

Oktalni oblik – omogućava definisanje numeričkih literala korišćenjem brojčanog sistema sa 8 cifara – od 0 do 7:

```
var x = 0312;
```

Oktalni literali uvek započinju karakterom **0**, kako bi se razlikovali od dekadnih i heksadecimalnih vrednosti. Nakon uvodnog karaktera 0, navodi se broj u oktalnom obliku. Oktalne vrednosti se, baš kao i heksadecimalne, prilikom ispisa, odnosno konvertovanja u tekst, podrazumevano predstavljaju u dekadnom obliku:

```
var x = 0312;
console.log(x);
```

Primer proizvodi rezultat 202.

Specijalni numerički literali

JavaScript poznaje tri specijalna numerička literala (vrednosti):

- `NaN`
- `Infinity`
- `- Infinity`

NaN je skraćenica za pojam *Not-a-Number*, zato što predstavlja vrednost koja se koristi da ukaže da vrednost nije validan broj. Sledeći primer proizvodi `NaN` vrednost:

```
var x = 0 / 0; // x is NaN
```

Vrednosti **Infinity** i **-Infinity** ukazuju da broj teži pozitivnoj ili negativnoj beskonačnosti. Tako će deljenje pozitivnog broja nulom proizvesti sledeću vrednost:

```
var x = 5 / 0; // x is Infinity
```

Deljenje negativnog broja nulom proizvodi negativnu beskonačnu vrednost:

```
var x = -5 / 0; // x is –Infinity
```

Preciznost brojeva u JavaScriptu

Za predstavljanje brojeva u kompjuterskoj memoriji JavaScript jezik koristi 64-bitni format koji je definisan IEEE 754 standardom. Ipak, ono što nas kao programere mnogo više treba da zanima jeste maksimalni broj cifara koji se može koristiti prilikom formulisanja celobrojnih i decimalnih vrednosti.

Kada su u pitanju celobrojne vrednosti, korišćenjem `number` tipa moguće je predstavljati brojeve od **-9007199254740991** do **9007199254740991.** Može se reći da `number` tip garantuje precizno rukovanje celobrojnim vrednostima sa maksimalno 15 cifara. Preko 15 cifara može doći do čudnog ponašanja i gubitka preciznosti. Ukoliko je potrebno predstaviti ceo broj izvan ovog opsega, neophodno je koristiti tip `bigint`. On se dobija dodavanjem karaktera **n** na kraj numeričkog literala:

```
var x = 9999999999999999n;
```

Nešto je komplikovanija situacija kada se govori o decimalnim brojevima u JavaScriptu. Maksimalni broj decimala koje je moguće navesti prilikom definisanja nekog broja je 17. Bitno je znati da se svi brojevi u decimalnom obliku predstavljaju korišćenjem `number` tipa. **Tip `bigint` isključivo je moguće koristiti za reprezentaciju celobrojnih vrednosti.** Takođe, veoma je bitno znati da JavaScript, kao i većina drugih programskih jezika koji za predstavljanje brojeva koriste IEEE 754 standard, poseduje određene poteškoće kada je u pitanju predstavljanje nekih decimalnih vrednosti:

```
var x = 0.1;
var y = 0.2;
var z = x + y;
console.log(z);
```

U prikazanom primeru obavlja se aritmetička operacija sabiranja nad brojevima `0.1` i `0.2`. Rezultat se smešta unutar promenljive `z`, a zatim se obavlja štampanje takve vrednosti unutar konzole:

```
0.30000000000000004
```

Iako je potpuno očekivano da se kao rezultat sabiranja dobije 0,3, unutar konzole se može videti da je vrednost nešto drugačija. Kao što je rečeno, razlog je način na koji se decimalni brojevi predstavljaju u binarnom obliku. Reč je o nedostatku koji poseduje gotovo svaki današnji programski jezik. Ipak, unutar većine jezika ovaj problem je rešen uvođenjem specijalnih numeričkih tipova podataka, koji su isključivo namenjeni preciznom radu sa

decimalnim vrednostima, što u nekim slučajevima može biti imperativ (bankarske transakcije, na primer).
Takvog nečeg za sada još nema u JavaScriptu, pa je veoma bitno poznavati ovakvu jezičku osobinu. Problem se u nekim situacijama može prevazići pretvaranjem decimalnih brojeva u cele brojeve, izvršavanjem operacije i ponovnim pretvaranjem u decimalni oblik:

```
var x = 0.1;
var y = 0.2;
var z = (x * 10 + y * 10) / 10;
console.log(z);
```

Decimalne vrednosti su sada prvo pretvorene u celobrojne (množenjem brojem 10), pa je tek onda obavljeno sabiranje. Kako bi se dobila tačna vrednost, zbir je na kraju podeljen brojem 10, čime je na kraju dobijen tačan rezultat:

```
0.3
```

Math objekat

JavaScript poseduje ugrađeni `Math` objekat, koji je moguće koristiti za obavljanje širokog spektra operacija nad numeričkim vrednostima. Ipak, bitno je naglasiti da je `Math` objekat moguće koristiti samo nad vrednostima `number` tipa, ali ne i nad `bigint` vrednostima.

`Math` objekat poseduje veliki broj različitih svojstava i metoda koje je moguće koristiti prilikom pisanja JavaScript koda. Neki od najznačajnijih `Math` objektnih članova biće predstavljeni u nastavku lekcije. Najznačajnija svojstva `Math` objekta ilustrovana su tabelom 10.1.

Svojstvo	Opis
`Math.LN2`	prirodni logaritam broja 2; iznosi približno 0.693
`Math.LN10`	prirodni logaritam broja 10; iznosi približno 2.303
`Math.PI`	odnos obima kruga i njegovog prečnika; približno 3.14159
`Math.SQRT2`	koren iz 2; približno iznosi 1.414
`Math.E`	Ojlerov broj, poznatiji kao broj e, osnova prirodnog logaritma

Tabela 10.1. Svojstva Math objekta

Iz tabele 10.1. se može videti da su svojstva `Math` objekta zapravo konstante koje je moguće koristiti za lako dobijanje nekih predefinisanih matematičkih vrednosti. Svakako nama najbliža je vrednost PI.

```
console.log(Math.PI);
```

Prikazana linija koda unutar konzole proizvodi sledeći rezultat:

```
3.141592653589793
```

Konstanta `Math.PI` se, na primer, može upotrebiti prilikom računanja površine kruga:

```
var r = 13;
```

```
var area = r*r*Math.PI;
console.log(area);
```

Promenljiva `r` predstavlja radijus, odnosno poluprečnik kruga. Definisano je da je poluprečnik kruga `13`. Unutar promenljive `area` smešta se rezultat izraza kojim se dobija površina kruga: `r` se množi sa `r`, pa sve to sa konstantom `PI`, koja je dobijena korišćenjem objekta `Math`.

Objekat `Math` poseduje i veliki broj različitih metoda koje je moguće koristiti u kontekstu rada sa numeričkim vrednostima (tabela 10.2).

Metoda	Opis
`Math.abs(x)`	vraća apsolutnu vrednost prosleđenog broja
`Math.round(x)`	metoda za zaokruživanje decimalnih vrednosti; broj zaokružuje na najbliži ceo broj
`Math.ceil(x)`	metoda za zaokruživanje decimalnih vrednosti; broj zaokružuje na prvu veću celobrojnu vrednost
`Math.floor(x)`	metoda za zaokruživanje decimalnih vrednosti; broj zaokružuje na prvu manju celobrojnu vrednost
`Math.pow(x, y)`	obavlja stepenovanje; prvi parametar je osnova, a drugi stepen na koji treba da se podigne prvi parametar
`Math.random()`	metoda za generisanje nasumičnog broja
`Math.sign(x)`	metoda koja utvrđuje da li je prosleđeni broj pozitivan, negativan ili jednak nuli
`Math.sqrt(x)`	metoda za računanje korena

Tabela 20.2. Metode Math objekta

U nastavku lekcije biće ilustrovano korišćenje prikazanih metoda klase `Math`.

Zaokruživanje decimalnih vrednosti

`Math` objekat poseduje tri metode koje je moguće koristiti za zaokruživanje decimalnih vrednosti:

- `round()`
- `ceil()`
- `floor()`

Metoda **round()** obavlja matematičko zaokruživanje na bliži ceo broj:

```
var x = Math.round(5.4);   //x=5
var y = Math.round(5.6);   //y=6
var z = Math.round(5.5);   //z=6
```

Promenljiva `x` nakon zaokruživanja će imati vrednost 5, zato što se tako zaokružuje vrednost 5,4, koja je bliža vrednosti 5 nego 6. Vrednost 5,6 zaokružuje se na 6. Kada se vrednost nalazi između dva cela broja (5,5), ona se uvek zaokružuje na prvi veći ceo broj.

Metoda **ceil()** decimalnu vrednost uvek zaokružuje na prvi veći ceo broj:

```
var x = Math.ceil(5.4);   //x=6
var y = Math.ceil(5.6);   //y=6
var z = Math.ceil(5.5);   //z=6
```

Metoda **floor()** zaokružuje decimalnu vrednost na prvi manji ceo broj:

```
var x = Math.floor(5.4);   //x=5
var y = Math.floor(5.6);   //y=5
var z = Math.floor(5.5);   //z=5
```

Generisanje nasumičnih brojeva

Objekat `Math` omogućava generisanje nasumične numeričke vrednosti, i to korišćenjem metode **random()**. Metoda `random()` može da vrati vrednosti između 0 i 1. Pri tome je vrednost 0 uključena u opseg mogućih vrednosti, a vrednost 1 nije.

Najjednostavniji primer korišćenja metode `random()` može da izgleda ovako:

```
var x = Math.random();   //x=0.1745352041420174
```

Naravno, sa svakim novim izvršavanjem prikazane naredbe vrednost promenljive `x` biće drugačija.

Stepenovanje i korenovanje

Stepenovanje i korenovanje je veoma lako moguće obaviti korišćenjem ugrađenih metoda objekta `Math`. Za korenovanje se koristi metoda **sqrt()**:

```
var x = Math.sqrt(16);    //x=4
```

Stepenovanje se postiže korišćenjem metode **pow()**:

```
var x = Math.pow(2, 2);   //x=4
var y = Math.pow(4, 2);   //y=16
var z = Math.pow(2, 4);   //z=16
```

Metoda `pow()` prihvata dva parametra. Prvi predstavlja osnovu koja sa stepenuje drugim parametrom. Tako se parametri metoda `pow()` unutar prikazanih naredbi mogu pročitati: *dva na kvadrat*, *četiri na kvadrat*, *dva na četvrti*. Naravno, sve ovo je bilo moguće postići i bez metode `pow()`, ali na znatno komplikovaniji način:

```
var x = 2 * 2;   //x=4
var y = 4 * 4;   //y=16
var z = 2 * 2 * 2 * 2;   //z=16
```

Znak i apsolutna vrednost

Apsolutna vrednost nekog broja može se dobiti korišćenjem metode **abs()**. Ova metoda negativne brojeve pretvara u pozitivne, dok pozitivni brojevi ostaju istog znaka:

```
var x = Math.abs(-200);   //x=200
var y = Math.abs(0);   //y=0
var z = Math.abs(198);   //z=198
```

Nekada se može javiti potreba za utvrđivanjem znaka neke numeričke vrednosti. Kada se kaže utvrđivanje znaka, misli se na proveru koja govori da li je broj pozitivan ili negativan. Takav posao lako se može obaviti korišćenje metode **sign()**:

```
var x = Math.sign(-200);   //x=-1
var y = Math.sign(0);   //y=0
var z = Math.sign(198);   //z=1
```

Metoda `sign()` može da vrati tri različite povratne vrednosti:

- -1 kada je broj negativan,
- 0 kada je broj jednak nuli,
- 1 kada je broj pozitivan.

Pitanje

Koja metoda se može koristiti za dobijanje nasumične numeričke vrednosti između 0 do 1?

- **random()**
- shuffle()
- mix
- rand()

Objašnjenje:

Objekat `Math` omogućava generisanje nasumične numeričke vrednosti i to korišćenjem metode `random()`. Metoda `random()` može da vrati vrednosti između 0 i 1.

Objektni omotači primitivnih numeričkih tipova

JavaScript poseduje dva primitivna numerička tipa: `number` i `bigint`. Kada se kaže da su oni primitivni, prevashodno se misli na to da oni nisu objekti, te da ne mogu da poseduju svojstva i metode, kao što je to slučaj sa objektima. Ipak, svaki od primitivnih tipova podataka u JavaScriptu je propraćen sa po jednim objektom. Svi takvi objekti se objedinjeno nazivaju objektni omotači primitivnih tipova.

Kada govorimo o numeričkim vrednostima i numeričkim primitivnim podacima, JavaScript poseduje dva objektna omotača primitivnih tipova:

- `Number` i
- `BigInt`.

Odmah možete primetiti da dva navedena objekta imaju identične nazive kao i primitivni tipovi na koje se odnose. Razlika je početno veliko slovo, kako bi se lakše razlikovao primitivni tip i odgovarajući omotač primitivnog tipa.

Baš kao i `Math` objekat, i `Number` i `BigInt` objekti poseduju određeni broj svojstava i metoda koje je moguće koristiti prilikom rada sa brojevima. Na primer, korišćenje jednog svojstva `Number` objekta može da izgleda ovako:

```
var x = Number.MAX_SAFE_INTEGER;
console.log(x);
```

Kôd će proizvesti maksimalnu celobrojnu numeričku vrednost koja se može predstaviti korišćenjem `number` tipa:

```
9007199254740991
```

Još jedna veoma zanimljiva osobina primitivnih tipova podataka može se razumeti poznavanjem objektnih omotača. JavaScript jezik omogućava da se uradi nešto ovako:

```
var x = 123.5512;
console.log(x.toFixed(2));
```

Unutar druge naredbe, nad promenljivom `number` tipa obavljeno je pozivanje metode `toFixed()`. Metoda `toFixed()` se inače koristi kako bi neki broj zaokružila na određeni broj decimala. U primeru je kao parametar prosleđen broj 2, što praktično znači da će metoda `toFixed()` numeričku vrednosti zaokružiti na dve decimala. Povratna vrednost ove metode je tipa `string`.

Sada se sa pravom možete zapitati kako je ovako nešto moguće, s obzirom na to da je već rečeno da primitivni tipovi ne mogu imati svojstva ni metode. Ipak, u prikazanom primeru se jasno vidi da se metoda `toFixed()` poziva nad promenljivom tipa `number`. U ovakvim situacijama, JavaScript u pozadini automatski obavlja pretvaranje prostog tipa u njegov objektni omotač. Na taj način je omogućeno pozivanje metode nad prostim tipom. Čim se logika metode koja se nalazi unutar objektnog omotača završi, takav objekat se uništava.

Konverzija teksta u broj

Ponekad se može javiti potreba za konvertovanjem tekstualnih vrednosti u numeričke, a za obavljanje takvog posla JavaScript poseduje dve ugrađene metode, prikazane tabelom 10.3.

Potpis metode	Opis
`parseInt(string, radix)`	parsira tekstualnu vrednost koja je prosleđena kao prvi parametar, korišćenjem osnove koja je prosleđena kao drugi parametar i isporučuje celobrojnu vrednost
`parseFloat(string)`	parsira tekstualnu vrednost koja je prosleđena kao parametar isporučuje vrednost sa decimalama

Tabela 10.3. Metode za konverziju tekstualnih podataka u brojeve

Metoda **parseInt()** parsira `string` vrednost koja je prosleđena kao prvi parametar korišćenjem osnove predstavljene drugim parametrom.

Drugi parametar definiše osnovu matematičkog brojčanog sistema koji će se koristiti prilikom konverzije (dekadni, oktalni, heksadecimalni).

```
parseInt("13", 10); //13
```

Prvi primer proizvodi numeričku vrednost 13. Situacija je jasna. Metodi `parseInt()` se prosleđuje tekstualna vrednost 13 i zahteva se njena konverzija u broj dekadnog sistema. Stoga se kao konačna vrednost dobija broj `13`.

```
parseInt("015", 10); //15
parseInt("015", 8); //13
```

Drugi primer ilustruje konvertovanje identične tekstualne vrednosti (015), ali sa različitim drugim parametrom koji definiše brojčani sistem. Kada se kao drugi parametar navede `10`, tekst biva pretvoren u broj `15`, iako započinje karakterom 0. Takav karakter se u ovoj situaciji ignoriše. Ali kada se kao drugi parametar metodi `parseInt()` prosledi vrednost 8, tekst biva konvertovan u broj oktalnog sistema (13).

Metodu `parseInt()` moguće je koristiti i za konvertovanje teksta u heksadecimalni oblik:

```
parseInt("0xFF", 16); //255
```

Prosleđena tekstualna vrednost je u karakterističnom heksadecimalnom obliku (započinje karakterima 0x). Kako bi se ovakav tekst pretvorio u heksadecimalnu numeričku vrednost `number` tipa, kao drugi parametar se prosleđuje vrednost 16.

Na kraju, kada se metodi `parseInt()` prosledi tekst koji ne može pretvoriti u broj, ona emituje vrednost `NaN`:

```
parseInt("hello", 10); //Nan
```

Metoda **parseFloat()** parsira tekst koji je prosleđen kao parametar i vraća decimalnu `number` vrednost:

```
parseFloat("3.14"); //3.14
```

Za razliku od metode `parseInt()`, metoda `parseFloat()` prihvata samo jedan parametar. U prikazanom primeru je to tekst 3.14. Stoga je povratna vrednost metode broj `3,14`.

Evo još jednog sličnog primera:

```
parseFloat("6.18"); //6.18
```

Metodu `parseFloat()` moguće je koristiti i za konverziju teksta sa brojem bez decimala:

```
parseFloat("5"); //5
```

Identično kao i `parseInt()`, metoda `parseFloat()` emituje `NaN` vrednost kada ne može da obavi konverziju:

```
parseFloat("some text"); //Nan
```

Primer – Program za računanje površine kruga

U nastavku će biti prikazan primer za računanje površine kruga. Formula za računanje površine kruga je:

$$P = r^2\Pi$$

Površina kruga se dobija množenjem poluprečnika sa poluprečnikom, a sve to sa konstantom Pi. Rešenje je sledeće:

```
let r = parseFloat(prompt("Please enter radius:", 0));
let p = Math.pow(r, 2) * Math.PI;
alert("The surface of the circle is " + p.toFixed(2));
```

Kompletan program poseduje tri linije. U prvoj liniji se od korisnika preuzima vrednost poluprečnika i takva vrednost se konvertuje u tip `number` korišćenjem metode `parseFloat()`. Obratite pažnju na to da je prilikom poziva metode `prompt()` naveden i drugi parametar koji predstavlja podrazumevanu vrednost koja će biti korišćena ukoliko korisnik ne unese nijednu vrednost.

U drugoj liniji se obavlja računanje površine. Za podizanje poluprečnika na kvadrat iskorišćena je metoda objekta `Math` – `pow()`. Za dobijanje vrednosti konstante Pi iskorišćeno je svojstvo `Math` objekta – `PI`.

U poslednjoj liniji rezultat je prikazan korisniku korišćenjem ugrađene metode `alert()`.

Poluprečnik je zaokružen na dve decimale korišćenjem metode `toFixed()`, kojoj se prosleđuje željeni broj decimala.

Rezime

- `number` je osnovni JavaScript tip za predstavljanje brojeva.
- `bigint` je tip podatka za predstavljanje celih brojeva povećane preciznosti.
- Prilikom formulisanja numeričkih literala moguće je koristiti celobrojni, decimalni, eksponencijalni, heksadecimalni i oktalni oblik.
- JavaScript poznaje tri specijalna numerička literala: `NaN`, `Infinity` i `–Infinity`.
- Tip `bigint` isključivo je moguće koristiti za reprezentaciju celobrojnih vrednosti.
- JavaScript poseduje ugrađeni `Math` objekat koji je moguće koristiti za obavljanje širokog spektra operacija nad numeričkim vrednostima.
- Metoda `round()` obavlja matematičko zaokruživanje na bliži ceo broj.
- Metoda `ceil()` decimalnu vrednost uvek zaokružuje na prvi veći ceo broj.
- Metoda `floor()` zaokružuje decimalnu vrednost na prvi manji ceo broj.
- Objekat `Math` omogućava generisanje nasumične numeričke vrednosti i to korišćenjem metode `random()`.
- Za korenovanje se koristi metoda `sqrt()`.
- Stepenovanje se postiže korišćenjem metode `pow()`.
- Apsolutna vrednost nekog broja može se dobiti korišćenjem metode `abs()`.
- Provera znaka neke numeričke vrednosti se može obaviti korišćenjem metode `sign()`.
- Metoda `toFixed()` se koristi kako bi neki broj zaokružila na određeni broj decimala.
- Metoda `parseInt()` parsira string vrednost koja je prosleđena kao prvi parametar, korišćenjem osnove predstavljene drugim parametrom.
- Metoda `parseFloat()` parsira tekst koji je prosleđen kao parametar i vraća decimalnu `number` vrednost.