

Web technológia

JavaScript, 2. rész

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM049.git

2022. október 5.

Főbb jellemzők:

- Immutable object (mint Java-ban)
- Létrehozás literálként: `'`-ok vagy `"`-ek között

Nyilvános tulajdonság:

`length`

A karakterlánc hossza

Metódusok

`charAt()`, `[]`

Adott indexű karakter lekérdezése

`indexOf(keresett[, tol])`, `lastIndexOf(keresett[, tol])`

Rész-karakterlánc (`keresett`) első/utolsó előfordulásának keresése `tol` indexű helytől kezdve. `indexOf`-nál negatív index is támogatott. Ha nincs találat, a visszatérési érték `-1`.

Adott indexű karakter lekérése, rész-karakterlánc keresése

```
1 console.log("JavaScript".charAt(0)); // J
2 console.log("JavaScript"[1]); // a
3 let js = "ABC"; js[0]="X"; console.log(js); // ABC -> immutable
4 console.log("JavaScript".indexOf("a")); // 1
5 console.log("JavaScript".indexOf("Script")); // 4
6 console.log("JavaScript".indexOf("a", 2)); // 3
7 console.log("JavaScript".indexOf("C++")); // -1
8 console.log("JavaScript".lastIndexOf("a")); // 3
9 console.log("JavaScript".lastIndexOf("a", 2)); // 1
```

`slice(tol[, ig])`

A $[tol, ig)$ index intervallumba eső karaktersorozat visszaadása. Negatív indexek támogatottak.

Rész-karakterlánc visszaadása

```
11 console.log("JavaScript".slice(4)); // "Script"
12 console.log("JavaScript".slice(0, 4)); // "Java"
13 console.log("JavaScript".slice(0, -6)); // "Java"
14 console.log("JavaScript".slice(-6)); // "Script"
```

`concat(s1[, s2[, ...[, sM]]]), +, +=`

Karakterláncok összefűzése. Az operátorok gyorsabban működnek.

`toLowerCase()`

Kisbetűs alak előállítása.

`toUpperCase()`

Nagybetűs alak előállítása.

Összefűzés, kis- és nagybetűs alakra alakítás

```
16 console.log("I_".concat("love_", "concat_", "so_", "much"));
17 // I love concat so much
18 console.log("but_" + "+/+=" + "operators_" + "are_" + "quicker!");
19 // but +/+= operators are quicker!
20 console.log("JavaScript".toLowerCase()); // javascript
21 console.log("JavaScript".toUpperCase()); // JAVASCRIPT
```

`trimStart()`, `trimEnd()`, `trim()`

Fehér karakterek eltávolítása egy karakterlánc elejéről, végéről, vagy mindkét végéről.

`split([elvalaszto [, max]])`

Karakterlánc szétDarabolása, *elvalaszto* jelek mentén (vagy reguláris kifejezéssel) és a darabok visszaszolgáltatása tömbben. *max* korlátozhatja a tömb méretét.

`join([elvalaszto])`

A **tömb** metódusa, mellyel elemei egyetlen karakterlánccá összefűzhetőek.

Fehér karakterek levágása, darabolás és összefűzés

```
23 console.log("[", "    JS    ".trimStart(), "]); // [ JS    ]
24 console.log("[", "    JS    ".trim(), "]); // [ JS ]
25 console.log("[", "    JS    ".trimEnd(), "]); // [    JS ]
26 let tomb = "a-b-c".split("-");
27 console.log(tomb); // [ "a", "b", "c" ]
28 console.log("<ul><li>" + tomb.join("</li><li>") + "</li></ul>");
29 // <ul><li>a</li><li>b</li><li>c</li></ul>
30 console.log(tomb.join("")); // abc
31 console.log(tomb.join()); // a,b,c
```

`padStart(hossz[, kitolto]), padEnd(hossz[, kitolto])`

Karakterlánc meghosszabbítása *kitolto* karakterrel balról vagy jobbról *hossz* hosszúságúra.

`repeat(db)`

Egymás után fűzés *db* alkalommal.

Kitöltés, ismétlés

```
33 console.log("[", "3".padStart(3), "]"); // [ 3 ]
34 console.log("[", "3".padStart(3, "0"), "]"); // [ 003 ]
35 console.log("[", "3".padEnd(3), "]"); // [ 3 ]
36 console.log("bla".repeat(3)); // blablabla
```

Statikus metódusok (\approx a Math objektum csak egy névtér):

`abs(n)`

n abszolút értékét adja vissza

`floor(n), ceil(n)`

n-nél kisebb/nagyobb egészek közül a legnagyobbat/legkisebbet adja

`round(n)`

n-et a legközelebbi egészre kerekíti

`min(v1, v2, ..., vn), max(v1, v2, ..., vn)`

a paraméterek közül a legkisebbet/legnagyobbat adja vissza

`pow(alap, kitevo)`

alap-ot *kitevo*-re emeli

`sqrt(n)`

n négyzetgyökét adja

`random()`

álvéletlen szám a $[0; 1)$ intervallumból

`sin()`, `cos()`, `tan()`

trigonometrikus függvények (paraméterek radiánban!)

`asin()`, `acos()`, `atan()`

trigonometrikus függvények inverz függvényei

`exp()`, `log()`

exponenciális fv., természetes alapú logaritmus

Konstansok

`PI` 3.1415...

`E` 2.71...

Cinkelt kocka: a paraméterekkel megadható, hogy

- hány oldala van a kockának, és
- ezek dobási valószínűségét súlyokkal lehet befolyásolni

Függvény definíció és hívás egy lépésben

Cinkelt kocka

```
1 console.log(function cinkeltKocka(...sulyok) {  
2   let osszeg = 0;  
3   for(let s of sulyok) {  
4     osszeg += s;  
5   }  
6   let veletlen = Math.random()*osszeg;  
7   let oldal = 0;  
8   for(let s=0; s<=veletlen; s+=sulyok[oldal], oldal++);  
9   return oldal;  
10 }(1, 1, 1, 1, 1, 5));
```

Átváltás fokról radiánra

fok → radián

```
1  const deg2rad = a => Math.PI*a/180;  
2  console.log(deg2rad(0));    // 0  
3  console.log(deg2rad(60));   // 1.0471975511965976  
4  console.log(deg2rad(120));  // 2.0943951023931953  
5  console.log(deg2rad(180));  // 3.141592653589793  
6  console.log(deg2rad(360));  // 6.283185307179586
```

Számok és karakterláncok közötti átalakítások

```
1 console.log(12.3 + 4); // 16.3
2 console.log("12.3" + 4); // "12.34"
3 console.log(parseFloat("12.3") + 4); // 16.3
4 console.log(parseInt("12.3") + 4); // 16
5 console.log(+ "12.3" + 4); // 16.3
6 console.log(parseInt("0xFF", 16)); // 255
7 console.log(parseInt("FF", 16)); // 255
8 console.log(parseInt("12számár")); // 12
9 console.log(parseInt("számár12")); // NaN
10 console.log("12.3".toString()); // "12.3"
```

JSON: JavaScript Object Notation

- Adatcsere formátum: pl. XML elemzéshez külön parser kell, a JS értelmező viszont eleve adott
- Leggyakoribb alkalmazásai: [Asynchronous JavaScript and XML \(AJAX\)](#), [JSON Web Token \(JWT\)](#)
- Eltérések a JS objektumoktól, pl.
 - tulajdonságnevek idézőjelek között
 - csak egyszerű adat kifejezések szerepelhetnek (pl. függvényhívás, kötések nem)
- Annotálás, validálás → [JSON Schema](#)

Legfontosabb metódusok:

[stringify\(\)](#) JS objektum → JSON string

[parse\(\)](#) JSON string → JS objektum

JSON.js

```
1  let hallgato = {  
2      nev: {  
3          titulus: "ifj.",  
4          vezetekNev: "Nagy",  
5          keresztnév: "Istvan"  
6      },  
7      neptun: "a1b2c3",  
8      született: new Date(2000, 5, 23),  
9      aktiv: true,  
10     lezartFelevek: ["2021/22/1", "2021/22/2"]  
11 }  
12  
13 let szoveg = JSON.stringify(hallgato);  
14 console.log(szoveg)  
15 console.log(JSON.parse(szoveg))
```

Kimenet

```
{"nev":{"titulus":"ifj.","vezetekNev":"Nagy","keresztNev":"Istvan"},  
  "neptun":"a1b2c3","szuletett":"2000-06-22T22:00:00.000Z",  
  "aktiv":true,"lezartFelevek":["2021/22/1","2021/22/2"]}
```

```
Object { nev: {...}, neptun: "a1b2c3",  
szuletett: "2000-06-22T22:00:00.000Z",  
aktiv: true, lezartFelevek: (2) [...] }
```

Példány létrehozása:

`new Date()`

A kliens óraállását veszi fel

`new Date(időbélyeg)`

A Unix-időszámítás kezdete óta eltelt ennyi ezredmp.-es állást veszi fel

`new Date(dátumStr)`

Karakterláncként adott időpontot veszi fel

`new Date(év, hó, nap, [óra, perc, mp, ezredmp])`

Adott óraállást veszi fel; hónapok számozása 0-tól kezdődik, az időformátum 24 órás

Érdekesebb metódusok:

`getTime()`, `setTime()`, `Date.now()` Időbélyeg

`getFullYear()`, `setFullYear()` Évszám

`getMonth()`, `setMonth()` hónap, január = 0

`getDate()`, `setDate()` hónap napja

`getDay()` hét napja, vasárnap = 0

`getHours()`, `setHours()` óra 24 órás formátumban

`getMinutes()`, `setMinutes()` perc

`getSeconds()`, `setSeconds()` másodperc

`getMilliseconds()`, `setMilliseconds()` ezredmásodperc

Néhány exportálási lehetőség karakterláncokba

`toString()` Pl. Mon Oct 03 2022

`toISOString()` Pl. 12:40:51 GMT+0200 (közép-európai nyári idő)

`toDateString()`

Pl. Mon Oct 03 2022 12:40:51 GMT+0200 (közép-európai nyári idő)

`toUTCString()` Pl. Mon, 03 Oct 2022 10:40:51 GMT

`toISOString()` Pl. 2022-10-03T10:40:51.039Z

`toJSON()` Pl. 2022-10-03T10:40:51.039Z

`toLocaleDateString()` Pl. 2022. 10. 03.

`toLocaleTimeString()` Pl. 12:40:51

`toLocaleString()` Pl. 2022. 10. 03. 12:40:51

Ld. még: `getTimezoneOffset()`