JavaScript beépített objektumai

A JavaScript nyelvbe számos előre beépített objektum van, amelyekkel gyakori feladatok elvégzése válhat könnyűvé.

Number

A Number objektum (igazából függvény) a számok csomagolóobjektuma. Általa a számokon meghívható néhány speciális formátumba alakító metódus:

- `toExponential([tizedesek])`: tudományos formátumban adja vissza szövegként a számot. Az opcionális paraméter a tizedesjegyek számát határozza meg.
- `toFixed([tizedesek])`: nem tudományos formátumban adja vissza szövegként a számot. Az opcionális paraméter a megjelenítendő tizedesjegyek számát határozza meg. Ha kevesebb tizedesjegy van, akkor a maradék helyet `0`-val tölti fel. Paraméter alapértelmezett értéke `0`.
- `toPrecision([számjegyek])`: a paraméterül megadott számú számjegyre kerekíti a számot, és adja vissza szövegként.
- `toString([számrendszer])`: a számot adott számrendszerbeli szövegként adja vissza.

```
//Számok mint objektumok metódusai
var a = 12.345;
a.toExponential();      //"1.2345e+1"
a.toExponential(2);     //"1.23e+1"
a.toExponential(5);     //"1.23450e+1"
a.toFixed();            //"12"
a.toFixed(2);           //"12.34"
a.toFixed(5);           //"12.34500"
a.toPrecision();        //"12.345"
a.toPrecision(2);       //"12"
a.toPrecision(5);       //"12.345"
a = 42;
a.toString();           //"42"
a.toString(2);          //"101010"
a.toString(5);          //"132"

//A Number objektum tulajdonságai
Number.MAX_VALUE;             //A legnagyobb ábrázolható pozitív szám
Number.MIN_VALUE;             //A legkisebb ábrázolható negatív szám
Number.NaN;                   //NaN
Number.NEGATIVE_INFINITY;     //-Infinity
Number.POSITIVE_INFINITY;     //Infinity
```

String

A String objektum (ez is függvény) a szövegek csomagolóobjektuma, és jó pár hasznos metódust tesz elérhetővé a szövegpéldányokon. A teljesség igénye nélkül nézzünk néhány hasznos függvényt:

`charAt(i), charCodeAt(i)`: egy karakter vagy kódja.

`indexOf(mit, honnan), lastIndexOf(mit, honnan)`: keresés elölről, hátulról, mindkettő indexet ad vissza.

`localeCompare(mivel)`: összehasonlítás, eredménye `-1`, `0`, `1`.

`search(regexp), replace(mit, mivel), match(regexp)`: keresés, csere, reguláris kifejezésekkel is.

`substr(honnan, hossz), substring(mettől, meddig), slice(mettől, meddig)`: részszövegek meghatározása.

`split(szeparátor)`: szöveg felbontása szeparátor mentén, a felbontott szövegrészek tömbbe kerülnek.

```
'piros alma'.charAt(2);                    //"r"
'piros alma'.charCodeAt(2);                //114
'piros alma'.indexOf('alma');              //6
'piros alma'.localeCompare('piros körte');  //-1
'piros alma'.replace('piros', 'sárga');    //"sárga alma"
'piros alma'.substr(2, 3);                 //"ros"
'piros alma'.split(' ');                   //["piros","alma"]
```

Date

A Date objektumon keresztül különböző dátummal kapcsolatos funkcionalitás érhető el. Egy új időpontot létrehozni a **new** Date() utasítással lehet. Paraméter nélkül az aktuális időpontot adja vissza. Paraméterein keresztül adott időpont állítható be (**new** Date(év, hónap, nap[, óra, perc, másodperc, ezredmásodperc])). Műveletei három nagy csoportba tartoznak:

getterek: a dátum egyes részeinek lekérdezésére szolgáló függvények, pl. getFullYear() az évet, getMonth() a hónapot adja vissza, stb.

setterek: a dátum egyes részeinek beállítása végezhető el e függvények által, pl. setFullYear(), setMonth(), stb.

szöveggé alakító metódusok: pl. toLocaleString() a nyelvi beállításoknak megfelelő formában adja vissza a dátumot, toGMTString() GMT formában adja vissza, stb.

Math

A Math objektumon keresztül számos matematikai függvény érhető el. A teljesség igénye nélkül néhány példa:

- **trigonometriai függvények:** sin(rad), asin(n), cos(rad), acos(n), tan(rad), atan(n);
- **hatványok, logaritmus:** pow(alap, kitevő), exp(n), log(n), stb.;
- **véletlenszámok:** random() (0 és 1 közötti számot generál);
- **kerekítések:** round(n), floor(n), ceil(n).

A Math objektum jó néhány konstanst is rendelkezésünkre bocsát (csupa nagy betűsek):

- PI
- E
- SQRT2
- LN2
- stb.

```
Math.PI; //3.141592653589793
Math.sin(90 * Math.PI / 180); //1; 90 fok szinusza
Math.random(); //pl. 0.47057095554085615
Math.random(); //pl. 0.5286946792885748
Math.round(1.6); //2
Math.floor(1.6); //1
```

Tömbműveletek

A tömbök is speciális objektumok, megannyi hasznos metódussal. Néhány fontosabb tömbfüggvény:

```
pop(), push(e), shift(e), unshift(): végéről, végére, elejére, elejéről.

reverse(): megfordítja a tömb elemeit.

splice(honnan, mennyit): kivág a tömbből elemeket. Ezzel lehet úgy törölni, hogy a
tömb hossza csökkenjen.

join(szeparátor): a tömb elemeit a szeparátor-ral elválasztva szöveggé fűzi össze.
```

```
var t = [1, 2, 3, 4, 5];
```

```
t.push(6);  //6; tömb hosszát adja vissza; tömb: [1, 2, 3, 4, 5, 6]
t.pop();          //6; tömb: [1, 2, 3, 4, 5]
t.unshift(0);     //6; tömb hosszát adja vissza; tömb: [0, 1, 2, 3, 4, 5]
t.shift();  //0; tömb: [1, 2, 3, 4, 5]
t.reverse();      //[5, 4, 3, 2, 1]; tömb: [5, 4, 3, 2, 1]
t.splice(2, 1); //[3]; kivágott elemeket adja vissza; tömb: [5, 4, 2, 1]
t.join('###');    //"5###4###2###1"
```

Reguláris kifejezések

A reguláris kifejezések olyan minták, amelyek bizonyos karakterkombinációk szövegre illeszkedését vizsgálják. A minta az ábécé betűin túl speciális karaktereket használ az illeszkedés leírására.

Mielőtt a részletekbe belemennénk, nézzük meg a következő példát, amely azt vizsgálja, hogy egy adott szövegben szerepel-e valamilyen szám:

```
/\d+/.test('101 kiskutya'); //true
/\d+/.test('A halál 50 órája'); //true
/\d+/.test('Tíz kicsi néger'); //false
```

A sorok elején / jelek között található a minta, aminek illeszkedését a szövegre a test() metódussal nézzük meg. Mivel bármilyen számot nézni szeretnénk, így olyan szabályt kell megfogalmaznunk, ami tetszőleges számú számkaraktert néz egymás után. Egy tetszőleges számot a \d helyettesít, azt pedig a + jel írja le, hogy a számból 1 vagy több lehet egymás után.

A mintát természetesen lementhetjük egy változóba is:

```
var regkif = /\d+/;
regkif.test('A halál 50 órája'); //true
```

- ^: szöveg elejére illeszkedik
- $: szöveg végére illeszkedik (soronként)
- *: legalább 0-szor ismétlődik az előtte lévő karakter vagy csoport
- +: legalább 1-szer ismétlődik az előtte lévő karakter vagy csoport
- ?: 0-szor vagy 1-szer ismétlődik az előtte lévő karakter vagy csoport
- {n,m}: legalább n-szer, legfeljebb m-szer ismétlődik az előtte lévő karakter vagy csoport
- .: bármilyen karakterre illeszkedik
- [karakterek]: a megadott karakterek bármelyikére illeszkedik
- (minta): csoportosítás. A zárójelekben megadott mintákra később hivatkozni lehet a \1, \2, stb. módon.
- \d: számok: [0-9]
- \D: nem számok

- \s: fehér szóköz
- \w: alfanumerikus karakter: [A-Za-z0-9_]

Az illeszkedés módját a minta után megadott módosítók befolyásolják:

- i: kis és nagybetűket nem különbözteti meg
- g: globális egyezés, ne álljon meg az elsőnél, hanem vizsgálja végig a szöveget
- m: sorvégjeleken átívelő egyezés

```
/^teremtő/.test('Teremtőnk kérünk tégedet'); //false
/^teremtő/i.test('Teremtőnk kérünk tégedet'); //true
```

A reguláris kifejezéseknek mint objektumoknak két saját metódusa van:

- test: megvizsgálja, hogy a minta illeszkedik-e a paraméterként megadott szövegre.
- exec: elvégzi a minta illesztését a paraméterül megadott szövegre, és a keresés eredményét tömbként adja vissza.

```
/m.g/g.test('Isten, áldd meg a magyart!'); //true
var t = /m.g/g.exec('Isten, áldd meg a magyart!');
t;                //["meg"]; az illeszkedő szövegrész
t.index;          //12; az illeszkedés helye
t.input;          //"Isten, áldd meg a magyart!"; az eredeti szöveg
```

A reguláris kifejezéseket intenzíven használják a különböző szövegmetódusok:

search(minta): megnézi, hogy a szövegben megtalálható-e a minta. Az illeszkedés helyét adja vissza.

match(minta): a keresés eredményét tömbben adja vissza.
replace(mit, mivel): az illeszkedő részeket kicseréli.

split(szeparátor): a feldarabolás alapjaként szolgáló szeparátor lehet reguláris kifejezés is.

Időzítők

JavaScriptben lehetőség van egy függvény végrehajtását egy megadott idő eltelte utánra időzíteni. Alapvetően két függvény segítségével érhető ez el. A setTimeout() függvény egy függvény egyszeri végrehajtását végzi el a megadott idő elteltével. A setInterval() függvény viszont egy függvény adott időközönkénti ismételt végrehajtására szolgál. Mindkét függvénynél első paraméter a végrehajtandó függvény referenciája (vagy egy függvényliterál), második paraméterként pedig a késleltetés idejét kell megadni ezredmásodpercekben. Mindkét függvény egy időzítőazonosítóval tér vissza, amelyen keresztül meg lehet állítani az időzített végrehajtást. A setTimeout()-tal indított időzítőt a clearTimeout() függvénnyel lehet leállítani, a setInterval() párja a clearInterval(). Mindkét megállító függvénynek paraméterül az időzítőazonosítót kell megadni. Általános alakjuk tehát a következő:

```
//setTimeout és clearTimeout
var időzítőAzonosító = setTimeout(függvény, ms);
clearTimeout(időzítőAzonosító);
//setInterval és clearInterval
var időzítőAzonosító = setInterval(függvény, ms);
clearInterval(időzítőAzonosító);
```

```
//Meghívandó függvény definiálása
function csorog() {
      console.log('Brrrrrrr');
}

//setTimeout példa: óracsörgés 2 másodperc múlva
setTimeout(csorog, 2000);

//vagy egyben
setTimeout(function () {
      console.log('Brrrrrrr');
}, 2000);

//Időzítő leállítása még a csörgés előtt
var idozito = setTimeout(csorog, 2000);
clearTimeout(idozito);

//óra csörgése 2 másodpercenként
setInterval(csorog, 2000);

//ismételt csörgés leállítása
var idozito2 = setInterval(csorog, 2000);
clearInterval(idozito2);
```

Az időzítőket főleg hosszan tartó folyamatoknál használjuk. Azért van rájuk szükség, mert a JavaScript kód a böngészőben ugyanazon a programszálon fut, mint a felhasználói felület kezelése. Ez azt jelenti, hogy ha van egy 10 másodpercig futó kódunk (pl. prímszámkeresés), akkor a felület (azaz maga a böngésző) 10 másodpercig nem használható, nem reagál. Időzítők segítségével azonban lehetőség van ütemezni ezeknek a végrehajtását, időt adva a böngészőnek az egyéb felhasználói események végrehajtására is. Legelterjedtebb használata az animációknál volt (a CSS3 animációk megjelenése előtt).