

Web-technológia

JavaScript

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

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

2021. szeptember 21.

Jellemzők:

- Egyetlen típus létezik csak: 64 bites lebegőpontos ábrázolás
- Pl. 42, 12.34, -34.56, 1e3, -1e3, 1e-3, -1e-3, -1.23e-4, -1.23E+4, ...
- Különleges értékek: Infinity, -Infinity, NaN
- Pl. 0/0 \rightarrow NaN, 1/0 \rightarrow Infinity

Operátorok

+ 5+3 \rightarrow 8

− 5-3 \rightarrow 2

× 5*3 \rightarrow 15

/ 5/3 \rightarrow 1.6666666666666667

% 5%3 \rightarrow 2, -5%3 \rightarrow -2, 5%-3 \rightarrow 2

** 5**3 \rightarrow 125

Precedencia táblázat

Jellemzők

- Unicode, 16 bit karakterenként
- Nincs specifikus típus egyetlen karakter tárolására
- Jelölés: ' -ok vagy " -ek között
- Pl. 'JavaScript', "JavaScript", "Guns 'n' Roses", "Egy\nKettő\nHárom", 'Guns \'n\' Roses', "Új sor \n megadásával kérhető."
- *Template literal*: ` -ek között, kifejezések kiértékelése
- Pl. `5 * 3 = \${5*3}` → "5 * 3 = 15"

Operátor

+ "Java" + 'Script' → "JavaScript"

Jellemzők

- Értékek: `true`, `false`
- Pl. `5 < 3` \rightarrow `false`

Logikai operátorok

`és` `true && false` \rightarrow `false`

`vagy` `true || false` \rightarrow `true`

`nem` `!true` \rightarrow `false`

Short circuit evaluation (pl. alapérték megadására):

```
undefined || "Gizi"  $\rightarrow$  "Gizi", null || "Gizi"  $\rightarrow$  "Gizi",  
"" || "Gizi"  $\rightarrow$  "Gizi", "Gizi" || "Mari"  $\rightarrow$  "Gizi"
```

Relációs operátorok

- `==`, `!=`, `<`, `<=`, `>`, `>=`
- Pl. `"Bill" != "Gates" → true`, `Infinity == Infinity → true`,
`de NaN == NaN → false`
- Karakterláncok összehasonlítása: karakterkódok alapján

Üres értékek: valaminek a hiányát jelzik

- undefined
- null

Egyoperandusú operátorok

típus `typeof(5) → "number", typeof("Gizi") → "string"`

— `-(5) → -5`

Háromoperandusú operátor

?: `1<2?"kisebb":"nagyobb" → "kisebb"`

Néhány példa:

- `5 * null → 0`
- `"5" - 3 → 2`
- `"5" + 3 → "53"`
- `"öt" * 3 → NaN, 5 * undefined → NaN`
- `false == 0 → true, true == 1 → true, true == 2 → false,`
- `"" == false → true`
- Definiált az érték? `null == undefined → true, null == 0 → false`

Típusok egyezését megkövetelő operátorok: `===`, `!==`

Változók (*variable, binding*)

- Deklaráció: `let` (blokk hatáskör), `var` (függvény hatáskör)
- Inkább tekinthető értékre mutató referenciának, mint valódi tárolónak

Példa

```
a → ReferenceError: a is not defined
let a
a → undefined
a = 5
a → 5
let b = 3, c
a * b → 15
```


Konstansok

- `const`

Példa

```
const c = 3.14
```

```
c = 2 → TypeError: invalid assignment to const 'c'
```

Névadási szabályok

- betűket, számokat, \$ és _ karaktereket tartalmazhat
- számjeggyel nem kezdődhet
- nem lehet foglalt szó (pl. let)
- kis- és nagybetűket megkülönbözteti
- javasolt stílus: *camel case* (hosszuValtozoNeve)

Változókkal használható (összetett és unáris) operátorok

- +=, -=, *=, /=, %=, &&=, ||=, **=, ...
- ++, -

Környezet (*environment*)

- adott pillanatban létező változók és értékeik
- gyakorlatilag soha nincs üres környezet

Megjegyzések

- // egysoros
- /* több
soros */

Szelekció

- `if(feltétel) utasítás;`
- `if(feltétel) {
 // utasítások
}`
- `if(feltétel) {
 // igaz ág utasításai
} else {
 // hamis ág utasításai
}`

- Mikor **nem** teljesül a *feltétel*?

- `false`
- `0`
- `""`
- `NaN`
- `null`
- `undefined`

Több irányú elágazás

```
switch(kifejezés) {  
    case érték1:  
        // utasítások  
        break;  
    case érték2:  
    case érték3:  
        // utasítások  
        break;  
    default:  
        // utasítások  
        break;  
}
```

Az értéknek és a típusnak is egyeznie kell!
A *default* ág elhagyható.

Ciklusok

```
for(előkészítés; ismétlési_feltétel; frissítés) {  
    // Ciklusmag utasításai  
}
```

```
while(ismétlési_feltétel) {  
    // Ciklusmag utasításai  
}
```

```
do {  
    // Ciklusmag utasításai  
} while (ismétlési_feltétel);
```

break, continue

Háromszög rajzolás (megoldás)

A böngésző JavaScript konzolján egy sornyi szöveget a `console.log()` hívással tud megjeleníteni. Használja ezt a következő háromszög megrajzolására:

```
*  
**  
***  
****  
*****
```

X rajzolás (megoldás)

Most rajzoljon 5x5-ös méretű X-et csillagokból:

```
*   *  
 *  *  
  *  
 *  *  
*   *
```

Sakktábla (megoldás)

Rajzoljon meg egy 8x8-as méretű sakktáblát, szintén csillagokból!

```
* * * *  
* * * *  
  * * * *  
* * * *  
  * * * *  
* * * *  
  * * * *  
* * * *
```


FizzBuzz (megoldás)

Vizsgálja meg az egész számokat 1-től 100-ig, majd a vizsgálat eredményét jelenítse meg egymás alatti sorokban! Ha a szám osztható 3-mal, írja ki, hogy *Fizz*, ha 5-tel osztható, akkor azt, hogy *Buzz*, ha pedig 3-mal és 5-tel is osztható, akkor azt, hogy *FizzBuzz*! Ha egyik számmal sem osztható, akkor írja ki a vizsgált számot!

```
1
2
Fizz
4
Buzz
Fizz
...
```

Definíció: a függvény, mint *érték* jelenik meg (**hatvanyDef.js**)

```
1  const hatvany = function(alap , kitevo) {  
2      let h = 1;  
3      for(let k=1; k<=kitevo; k++) {  
4          h *= alap;  
5      }  
6      return h;  
7  }  
8  
9  console.log(hatvany(2, 0)); // 1  
10 console.log(hatvany(2, 1)); // 2  
11 console.log(hatvany(2, 2)); // 4  
12 console.log(hatvany(3, 2)); // 9
```

Deklaráció: helye a hatókörön belül bárhol lehet (**hatvanyDek.js**)

```
1 console.log(hatvany(2, 0)); // 1
2 console.log(hatvany(2, 1)); // 2
3 console.log(hatvany(2, 2)); // 4
4 console.log(hatvany(3, 2)); // 9
5
6 function hatvany(alap, kitevo) {
7     let h = 1;
8     for(let k=1; k<=kitevo; k++) {
9         h *= alap;
10    }
11    return h;
12 }
```

Nyíl (*arrow*) függvény: tömörebb megadás (*hatvanyNyil.js*)

```
1  const hatvany = (alap , kitevo) => {  
2    let h = 1;  
3    for(let k=1; k<=kitevo; k++) {  
4      h *= alap;  
5    }  
6    return h;  
7  }  
8  
9  console.log(hatvany(2, 0)); // 1  
10 console.log(hatvany(2, 1)); // 2  
11 console.log(hatvany(2, 2)); // 4  
12 console.log(hatvany(3, 2)); // 9
```

Nyíl függvények

- Ha pontosan egy paramétert fogad, a paraméterlista körüli zárójelek elhagyhatóak
- Ha egyetlen paramétert sem fogad, üres zárójelpár jelzi a paraméterlistát
- Ha a függvény teste egyetlen kifejezés értékét szolgáltatja, a `return` és a blokk elhagyható

nyilValtozatok.js

```
1 const negyzet = alap => alap*alap;  
2 console.log(negyzet(3)); // 9  
3  
4 const udvozol = () => console.log("Szia!");  
5 udvozol(); // Szia!
```

hatokor.js

```
1  let a = 1; // globális
2  {
3    let a = 2; // elfedés, lokális
4    let b = 3; // lokális
5    var c = 4; // globális
6    console.log('a=${a}, b=${b}, c=${c}'); // a=2, b=3, c=4
7  }
8  //console.log('a=${a}, b=${b}, c=${c}'); // ReferenceError: b is not defined
9  console.log('a=${a}, c=${c}'); // a=1, c=4
```

Paraméterezés

- Nem ellenőrzi híváskor sem a paraméterek számát, sem azok típusát! → felesleges paramétereket figyelmen kívül hagyja, a hiányzók értéke `undefined`
- A `return` nélküli, vagy a `return` után kifejezést nem tartalmazó függvények visszatérési értéke `undefined`
- Tetszőleges számú paramétert fogadó fv. is készíthető (ld. később)

parameter1.js

```
1 const negyzet = alap => alap*alap;  
2 console.log(negyzet(3, 4, 5)); // 9  
3 console.log(negyzet(3)); // 9  
4 console.log(negyzet()); // NaN  
5 console.log(negyzet("Micimackó")); // Nan
```

parameter2.js

```
1  const negyzet = alap => {  
2    if (typeof(alap)=="number") {  
3      return alap*alap;  
4    } else {  
5      return;  
6    }  
7  }  
8  
9  console.log(negyzet(3)); // 9  
10 console.log(negyzet()); // undefined  
11 console.log(negyzet("Micimackó")); // undefined
```


A függvények *értékek*:

- függvények átadhatók más függvénynek paraméterként,
- függvény visszatérési értéke lehet függvény,
- függvény beágyazható másik függvénybe.

paramFv1.js

```
1  const összead = (a, b) => a + b;  
2  const muvelet = function(a, op, b) {  
3    console.log( `${a} + ${b} = ${op(a, b)} ` );  
4  }  
5  muvelet(3, összead, 5); // 3 + 5 = 8
```

Névtelen (*anonymous*) függvények (paramFv2.js)

```
1  const muvelet = function(a, op, b) {  
2    console.log(`${a} + ${b} = ${op(a, b)}`);  
3  }  
4  muvelet(3, (a, b) => a + b, 5); // 3 + 5 = 8  
5  muvelet(4,  
6    function(a, b) {  
7      return a + b;  
8    },  
9    7  
10 ); // 4 + 7 = 11
```

Függvények definiálása és azonnali hívása (paramFv3.js)

```
1  (function(a, op, b) {  
2      console.log(`${a} + ${b} = ${op(a, b)} `);  
3  })(3, (a, b) => a + b, 5); // 3 + 5 = 8  
4  
5  ((a, op, b) => {  
6      console.log(`${a} + ${b} = ${op(a, b)} `);  
7  })(4, (a, b) => a + b, 7); // 4 + 7 = 11
```

Zárványok (*closure*)

- Mi történik, ha egy *külső* függvény *lokális* változóit eléri egy *belső* függvény, amit *meghívunk* azután, hogy az őt létrehozó *külső* függvényből kiléptünk?

Zárványok (*closure*)

- Mi történik, ha egy *külső* függvény *lokális* változóit eléri egy *belső* függvény, amit *meghívunk* azután, hogy az őt létrehozó *külső függvényből kiléptünk*?
- A függvény megőrzi futtatási környezetét

Függvény, mint visszatérési érték (zarvany.js)

```
1  const hatvany = (kitevo) => {  
2    return alap => {  
3      let h = 1;  
4      for(let k=1; k<=kitevo; k++) {  
5        h *= alap;  
6      }  
7      return h;  
8    };  
9  };  
10 const negyzet = hatvany(2);  
11 const kob = hatvany(3);  
12 console.log(negyzet(3)); // 9  
13 console.log(kob(5)); // 125
```

Rekurzív hatványozás (rekurzio.js)

```
1  const hatvany = function(alap, kitevo) {  
2      if(kitevo == 0) return 1;  
3      if(kitevo == 1) return alap;  
4      let h = hatvany(alap, (kitevo-kitevo%2)/2);  
5      h *= h;  
6      if(kitevo%2) {  
7          h *= alap;  
8      }  
9      return h;  
10 }  
11 console.log(hatvany(5, 3)); // 125
```

Fibonacci-számok (fibonacci.js)

Fibonacci-sorozat: másodrendben rekurzív sorozat. Képzeltbeli nyúlcsalád növekedése: hány pár nyúl lesz n hónap múlva, ha

- az első hónapban csak egyetlen újszülött nyúl-pár van,
- az újszülött nyúl-párok két hónap alatt válnak termékennyé,
- minden termékeny nyúl-pár minden hónapban egy újabb párt szül,
- és a nyulak örökké élnek.

$$F_n = \begin{cases} 0, & \text{ha } n = 0 \\ 1, & \text{ha } n = 1 \\ F_{n-1} + F_{n-2} & \text{ha } n > 1 \end{cases}$$

Készítse el azt a fibonacci függvényt, melynek paramétere a sorozat valamely elemének indexe (n), visszatérési értéke a sorozat megfelelő eleme!

Négyzetgyökvonás (gyok.js)

Készítse el a gyok függvényt, mely Newton módszerrel meghatározza és visszatérési értéként szolgáltatja paraméterének négyzetgyökét!

A módszer iteratív: egy sorozat egymást követő tagjait kell kiszámolni, melyek általában nagyon gyorsan konvergálnak a keresett eredményhez. A sorozat első elemét célszerű lenne a megoldás közeléből választani, de az egyszerűség kedvéért legyen ez nálunk mindig 10. Ha az utolsóként meghatározott tag értéke 10^{-6} -nál nem nagyobb mértékben tér el az utolsó előttiként kiszámolttól, akkor ezt az utolsóként kiszámolt értéket tekintjük a megoldásnak. A Newton módszer szerint a sorozat tagjait általánosan a következőképpen határozzuk meg:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Konkréten a négyzetgyökvonás esetén, ha pl. az $x^2 = 612$ (itt 612 a gyok függvény aktuális paraméterének feleltethető meg) zérushelyét keressük, azaz $f(x) = x^2 - 612$ akkor $f'(x) = 2x$.

Ebből adódik, hogy $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 10 - \frac{10^2 - 612}{2 \cdot 10} = 35.6$ majd

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 35.6 - \frac{35.6^2 - 612}{2 \cdot 35.6} = 26.3955056, \text{ stb.}$$

Szinusz függvény (sin.js)

Írja meg azt a `sin` függvényt, amely visszaadja a paraméterként kapott, radiánban mért szög szinuszát!

A keresett érték meghatározható a szinusz függvény sorba fejtésével:

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \text{ azaz } \sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

A függvénynek természetesen nem kell végtelen sok tagot, illetve azok összegét meghatároznia. Elegendő, ha a függvény $\epsilon = 10^{-6}$ pontossággal kiszámítja az eredményt.