

Web technológia

JavaScript, 2. rész

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

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

2022. november 15.

Objektumok:

- egységbe zárt (encapsulation) adattagok és metódusok (függvények)
- a rejtett `this` kötés kapcsolja össze az objektum tulajdonságait
- minden tulajdonság nyilvános

Objektum literál

```
1  const teglalap = {
2    a: 5,
3    b: 3,
4    kerulet: function() {
5      return 2 * (this.a + this.b)
6    }
7  }
```



```

30 // A nyílt fv. látja környezetének this kötését
31 function atlo3() {
32     return (() => Math.sqrt(this.a*this.a + this.b*this.b))()
33 }
34 // Egy hagyományos function-re viszont ez nem igaz
35 function atlo4() {
36     return (function() {
37         return Math.sqrt(this.a*this.a + this.b*this.b);
38     })();
39 }
40 console.log('Átlójának hossza: ${atlo3.call(teglalap)}') // 5.83
41 console.log('Átlójának hossza: ${atlo4.call(teglalap)}') // NaN

```

A prototípusnak is lehet prototípusa: származtatási hierarchia, fa struktúra, csúcsán: `Object.prototype`; ez a közös ő.

```
2 console.log(Object.getPrototypeOf([]) == Array.prototype); // true
3 console.log(Object.getPrototypeOf(Array.prototype) == Object.prototype);
4 // true
5 console.log(Object.getPrototypeOf(Object.prototype) == null); // true
```

```
7 // Őse az Object.prototype
8 const negyzet = {
9   a: 5,
10  kerulet: function() {
11    return 4 * this.a;
12  },
13  terulet() { // rövidítés
14    return this.a * this.a;
15  }
16};
```

1100

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523</
--	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-------

Prototípusnak szánt objektumba jellemzően csak olyan tulajdonságot tesznek, melynek értékét minden ebből származó objektumnak tartalmaznia kell.

Konstruktor: új objektum létrehozása adott prototípusból, az egyedre jellemző értékek hozzáadásával.

Objektum létrehozása konstruktorral

```
1  const negyzet = {  
2    kerulet() {  
3      return 4 * this.a;  
4    },  
5    terulet() {  
6      return this.a * this.a;  
7    }  
8  };  
9  
10 function konstruktor(oldalHossz) {  
11   const peldany = Object.create(negyzet);  
12   peldany.a = oldalHossz;  
13   return peldany;  
14 }
```

Objektum létrehozása konstruktorral

```
16 const negyzet2 = konstruktor(2);
17 const negyzet5 = konstruktor(5);
18 console.log('a=${negyzet2.a} '); // a=2
19 console.log('K=${negyzet2.kerulet()}, T=${negyzet2.terulet()} ');
20     // K=8, T=4
21 console.log('a=${negyzet5.a} '); // a=5
22 console.log('K=${negyzet5.kerulet()}, T=${negyzet5.terulet()} ');
23     // K=20, T=25
```

Majdnem ugyanez történik, ha a `new` kulcsszót írjuk egy függvény elé, azaz konstruktorként fog viselkedni.

Objektum létrehozása `new` kulcsszóval

```
1  function Negyzet(oldalHossz) {  
2      this.a = oldalHossz;  
3  }  
4  Negyzet.prototype.kerulet = function() {  
5      return 4 * this.a;  
6  }  
7  Negyzet.prototype.terulet = function() {  
8      return this.a * this.a;  
9  }  
10 const negyzet2 = new Negyzet(2);  
11 const negyzet5 = new Negyzet(5);
```

Objektum létrehozása new kulcsszóval

```
12 console.log('a=${negyzet2.a}, K=${negyzet2.kerulet()}'); // 2, 8
13 console.log('a=${negyzet5.a}, T=${negyzet5.terulet()}'); // 5, 25
14 console.log(Object.getPrototypeOf(negyzet2) === Negyzet.prototype);
15     // true
16 console.log(Object.getPrototypeOf(Negyzet) === Function.prototype);
17     // true
18 console.log(Object.getPrototypeOf(Function.prototype) ===
19     Object.prototype); // true
```

Minden függvénynek van prototype tulajdonsága, a konstruktorok is. Ez egy lecserélhető üres objektum, de a meglévőhöz is hozzáadhatók új tulajdonságok, mint a példában.

2015-től lehet használni a `class` és `constructor` kulcsszavakat. A háttérben ettől még ugyanaz történik (syntactic sugar), továbbra sem léteznek valódi osztályok a nyelvben. Konstans adatokat nem lehet az objektumhoz adni.

class, constructor

```
1  class Negyzet {  
2      constructor(oldalHossz) {  
3          this.a = oldalHossz;  
4      }  
5      kerulet() {  
6          return 4 * this.a;  
7      }  
8      terulet() {  
9          return this.a * this.a;  
10     }  
11 }
```

class, constructor

```
13  const negyzet2 = new Negyzet(2);
14  const negyzet5 = new Negyzet(5);
15  console.log('a=${negyzet2.a}, K=${negyzet2.kerulet()}'); // 2, 8
16  console.log('a=${negyzet5.a}, T=${negyzet5.terulet()}'); // 5, 25
17  console.log(Object.getPrototypeOf(negyzet2) === Negyzet.prototype);
18    // true
19  console.log(Object.getPrototypeOf(Negyzet) === Function.prototype);
20    // true
21  console.log(Object.getPrototypeOf(Function.prototype) ===
22    Object.prototype); // true
```

A class kifejezésben is szerepelhet, nem csak utasításban.

class kifejezés

```
24 // osztály kifejezés
25 const kocka = new class {
26     constructor(oldalak) {
27         this.oldalak = oldalak;
28     }
29     dobas() {
30         return Math.floor(Math.random()*this.oldalak) + 1;
31     }
32 }(6);
33 console.log(kocka.dobas()); // [1, 6]
```

Az Object-tól örökölt metódusok egyedileg felüldefiniálhatók.

Felüldefiniálás

```
35 console.log(kocka.toString()); // [object Object]
36 kocka.toString = function() {
37     return 'Ez egy ${this.oldalak} oldalszámú "kocka".';
38 }
39 console.log(kocka.toString()); // Ez egy 6 oldalszámú "kocka".
40 console.log(String(kocka)); // Ez egy 6 oldalszámú "kocka".
41 delete kocka.toString;
42 console.log(kocka.toString()); // [object Object]
```


Objektumokat használni asszociatív tömbként nem biztonságos:

- 1 öröklött tulajdonságok is megjelennek kulcsként
- 2 implicit típuskonverzió miatt kulcsnak látszódhat egy érték, ami nem az

Asszociatív tömb objektummal

```
1  const hallgato = {
2    "a1b2c3": "Kovács István",
3    "abc123": "Nagy Ilona"
4  };
5  console.log('a1b2c3" neve ${hallgato["a1b2c3"]}') // Kovács István
6  console.log('Van "abc123" kódú hallgató? ${"abc123" in hallgato}'); // true
7  console.log('Van "toString" kódú hallgató? ${"toString" in hallgato}');
8    // true !!!
9  console.log('a1b2c3" saját tulajdonság? ' +
10    `${hallgato.hasOwnProperty("a1b2c3")}`); // true
11  console.log('"toString" saját tulajdonság? ' +
12    `${hallgato.hasOwnProperty("toString")}`); // false
13  console.log(Object.keys(hallgato)); // ["a1b2c3", "abc123"]
```

Az első problémát megoldja, ha nincs prototípusa az objektumnak.

Asszociatív tömb objektummal

```
15  const hallgato2 = Object.create(null);
16  hallgato2["a1b2c3"] = "Kovács István";
17  hallgato2["abc123"] = "Nagy Ilona";
18  console.log('Van "abc123" kódú hallgató? ${"abc123" in hallgato2}');
19  // true
20  console.log('Van "toString" kódú hallgató? ${"toString" in hallgato2}');
21  // false
```

A másodikon viszont nem segít, sőt mellékhata is lehet (pl. nyomkövetésnél)

Megoldás: Map (és Set), ld. később

Asszociatív tömb objektummal

```
23  const asszociativ = {  
24    "123": "érték"  
25  };  
26  console.log(asszociativ["123"]); // érték  
27  console.log(asszociativ[123]); // érték !!!
```

Az objektumok kulcsaiként eddig mindig String-ek álltak. Ez azonban akkor is lehetővé teszi pl. egy metódus felüldefiniálását vagy lecserélését, ha nem szándékoztuk.

Tulajdonságok karakterlánccal megadva

```
1  const negyzet = {  
2    a: 5  
3  };  
4  // negyzet.toString is jó lenne  
5  negyzet["toString"] = function() {  
6    return 'A négyzet oldalhossza: ${this.a}';  
7  }  
8  negyzet["toString"] = function() { // felülírás  
9    return 'Oldalhossz: ${this.a}, kerület: ${4*this.a}';  
10 }  
11 console.log(String(negyzet)); // Oldalhossz: 5, kerület: 20
```

Ezzel szemben a létrehozott szimbólumok mindig egyediek.

Tulajdonságok szimbólummal megadva

```
13  const negyzet2 = {
14      a: 5
15  };
16  const toString1 = Symbol("toString");
17  const toString2 = Symbol("toString");
18  negyzet2[toString1] = function() {
19      return 'A négyzet oldalhossza: ${this.a}';
20  }
21  negyzet2[toString2] = function() {
22      return 'Oldalhossz: ${this.a}, kerület: ${4*this.a}';
23  }
24  console.log(String(negyzet2[toString1]())); // A négyzet oldalhossza: 5
25  console.log(String(negyzet2[toString2]())); // Oldalhossz: 5, kerület: 20
26  console.log(toString1 == toString2); // false
```

A `for/of` ciklusokkal azok a gyűjtemények járhatók be, melyek megvalósítják az `iterator` interfészt, azaz rendelkeznek egy `Symbol.iterator` nevű metódussal, ami visszadja a tényleges bejárást biztosító iterátor objektumot.

Ennek `next()` metódusával lehet elkérni a `value` és `done` kulcsokkal rendelkező objektumokat. Az első adattag a tényleges, soron következő értéket adja meg, a másodikból kiderül, hogy elérhető-e még egyáltalán további adat.

Legismertebb megvalósítások a nyelvben: `Array`, `String`.

Beépített objektumok iterátorral

```

1  for(elem of ['A', 'B', 'C']) {
2    console.log(elem); // A B C
3  }
4  for(betu of "JS") {
5    console.log(betu); // J S
6  }
7  const JSiterator = "JS"[Symbol.iterator]();
8  console.log(JSiterator.next()); // { value: "J", done: false }
9  console.log(JSiterator.next()); // { value: "S", done: false }
10 console.log(JSiterator.next()); // { value: undefined, done: true }
```

Készítsünk a **Python** és a **PHP** mintájára olyan Intervallum objektumot, ami a [tol, ig) intervallumból ad vissza egymástól lepes-nyire lévő értékeket!

Saját objektum iterátorral

```
12 class Intervallum {
13     constructor(tol, ig, lepes=1) {
14         this.tol = tol;
15         this.ig = ig;
16         this.lepes = lepes;
17     }
18     [Symbol.iterator]() {
19         return new IntervallumIterator(this);
20     }
21 }
```

```
class IntervallumIterator {
  constructor(intervallum) {
    this.aktualis = intervallum.tol;
    this.intervallum = intervallum;
    this.elojel = intervallum.lepes >= 0 ? +1 : -1;
  }
  next() {
    if (this.aktualis * this.elojel < this.intervallum.ig) {
      const eredmeny = { value: this.aktualis, done: false };
      this.aktualis += this.intervallum.lepes;
      return eredmeny;
    } else {
      return { value: undefined, done: true };
    }
  }
}
```


Saját objektum iterátorral

```
40 for(elem of new Intervallum(0, 10, 2)) {  
41     console.log(elem); // 0 2 4 6 8  
42 }  
43 for(elem of new Intervallum(5, 0, -1)) {  
44     console.log(elem); // 5 4 3 2 1  
45 }
```

JS-ben valamennyi adattag nyilvánosan elérhető. Néha viszont nem szeretnénk (redundáns) adattagokat létrehozni, abban folyamatosan adatokat tárolni és azt frissíteni, amikor az objektum változik. Ilyenkor létrehozhatunk olyan metódusokat, melyek adattagnak tűnnek a kívüljár számára, és a számított adatokat szolgáltatják, amikor szükség van rájuk.

Hasonlóképpen létrehozhatunk adattagnak tűnő metódusokat tárolt értékek beállítására, input ellenőrzésére is.

Időnként egy adat nem példányhoz kötődik, hanem az „osztályhoz” (ld. `Math.PI`). Ezeket célszerű csak egyszer tárolni → statikus adattag. Hasonlóan, ha egy metódus nem dolgozik a példány adataival, megjelölhetjük statikusként (pl. `Math.sin()`). Hívásakor a prototípus/osztály nevével minősítjük. Jellemzően objektumok létrehozására, másolására használják őket.

getter, setter, static

```
1  class USgallon {
2      static gallon2liter = 3.785411784;
3      constructor(gallon) {
4          this.gallon = gallon;
5      }
6      get liter() {
7          return this.gallon * USgallon.gallon2liter;
8      }
9      set liter(value) {
10         this.gallon = value / USgallon.gallon2liter;
11     }
12     static fromLiter(value) {
13         return new USgallon(value / USgallon.gallon2liter);
14     }
15 }
```

getter, setter, static

```
17 console.log('1 liter = ${USgallon.fromLiter(1).gallon} US gallon ');
18    // 1 liter = 0.26417205235814845 US gallon
19 let g = new USgallon(1);
20 console.log('1 US gallon = ${g.liter} liter ');
21    // 1 US gallon = 3.785411784 liter
22 g.liter = 10;
23 console.log('10 liter = ${g.gallon} US gallon ');
24    // 10 liter = 2.6417205235814842 US gallon
```

Származtatás megvalósításához, az ős megnevezésére megjelent az `extends` kulcsszó. Az ősre `super` segítségével lehet hivatkozni.

Származtatás

```
1  class Negyzet {  
2      constructor(a) {  
3          this.a = a;  
4      }  
5      get kerulet() {  
6          return 4 * this.a;  
7      }  
8      get terület() {  
9          return this.a * this.a;  
10     }  
11     toString() {  
12         return 'Négyzet a=${this.a}, K=${this.kerulet}, T=${this.terulet}';  
13     }  
14 }
```

Származtatás

```
16 class Teglalap extends Negyzet {
17     constructor(a, b) {
18         super(a);
19         this.b = b;
20     }
21     get kerulet() {
22         return 2 * (this.a+this.b);
23     }
24     get terület() {
25         return this.a * this.b;
26     }
27     toString() {
28         return 'Teglalap a=${this.a}, b=${this.b}, ' +
29             'K=${this.kerulet}, T=${this.terulet}';
30     }
31 }
```

Származtatás

```
33 const negyzet = new Negyzet(2);  
34 const teglalap = new Teglalap(3, 5);  
35 console.log(String(negyzet)); // Négyzet a=2, K=8, T=4  
36 console.log(String(teglalap)); // Téglalap a=3, b=5, K=16, T=15  
37 console.log(teglalap instanceof Teglalap); // true  
38 console.log(teglalap instanceof Negyzet); // true  
39 console.log(teglalap instanceof Object); // true
```

Az instanceof operátorral ellenőrizhető egy objektum (akár közvetett) prototípusa.

Lekerekített téglalap (`lekerekített.js`)

Készítsen Teglalap „osztályt”, melynek konstruktora megkapja paraméterként a síkidom szélességét, magasságát, és egy logikai értéket, melyből kiderül, hogy rajzolásnál csak a körvonalat kell megrajzolni, vagy a téglalap belsejét is ki kell tölteni * karakterekkel! A `rajz()` metódus adja vissza a rajzot egy String formájában! Származtasson ebből egy `Lekerekített` nevű osztályt, amelynek konstruktora egy további paramétert fogad, a sarkok lekerekítési sugarát!

A String objektum

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
```

A String objektum

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

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

`split([elvalasztó[, max]])`

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

`join([elvalasztó])`

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 *kitalto* 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

álvéletlen szám a $[0; 1)$ intervallumból

trigonometrikus függvények (paraméterek radiánban!)

trigonometrikus függvények inverz függvényei

exponenciális fv., természetes alapú logaritmus

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 \rightarrow 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

```
toDateString() Pl. Mon Oct 03 2022
```

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

toString()

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()`

Tömbelemek összekeverése (**kever.js**)

Írjon olyan függvényt, ami összekeveri egy tömb elemeit! Ötlet: [Fisher-Yates shuffle](#)