

JSON adatmodell

# JSON adatmodell

Az előadás anyaga

Prof. Dr. Kovács László: Adatkezelés XML környezetbe,  
Jeszenszky Péter: XML és további irodalom alapján készült el

# Témakörök

1. JSON adatmodell
2. JSON vs. XML
3. JSON adattípusok
4. YAML formátum
5. Mintafeladatok

# Igényelt kompetenciák

- JSON adatmodell
- YMAL formátum
- Mintafeladatok
- Környezet: XML szerkesztő (Oxygen, EditIX, Eclipse, ....)

# XML – Előnyök - ismételés

- „Könnyen *alkalmazható webes rendszerekben.*
- *Keresés (web) helyett lekérdezéssel (DB) juthatunk információhoz.*
- *Univerzális adatcsere formátum, amely hozzájárul az üzleti alkalmazások szabványos kommunikációhoz.*
- Gyártófüggetlenség.
- Platformfüggetlenség.

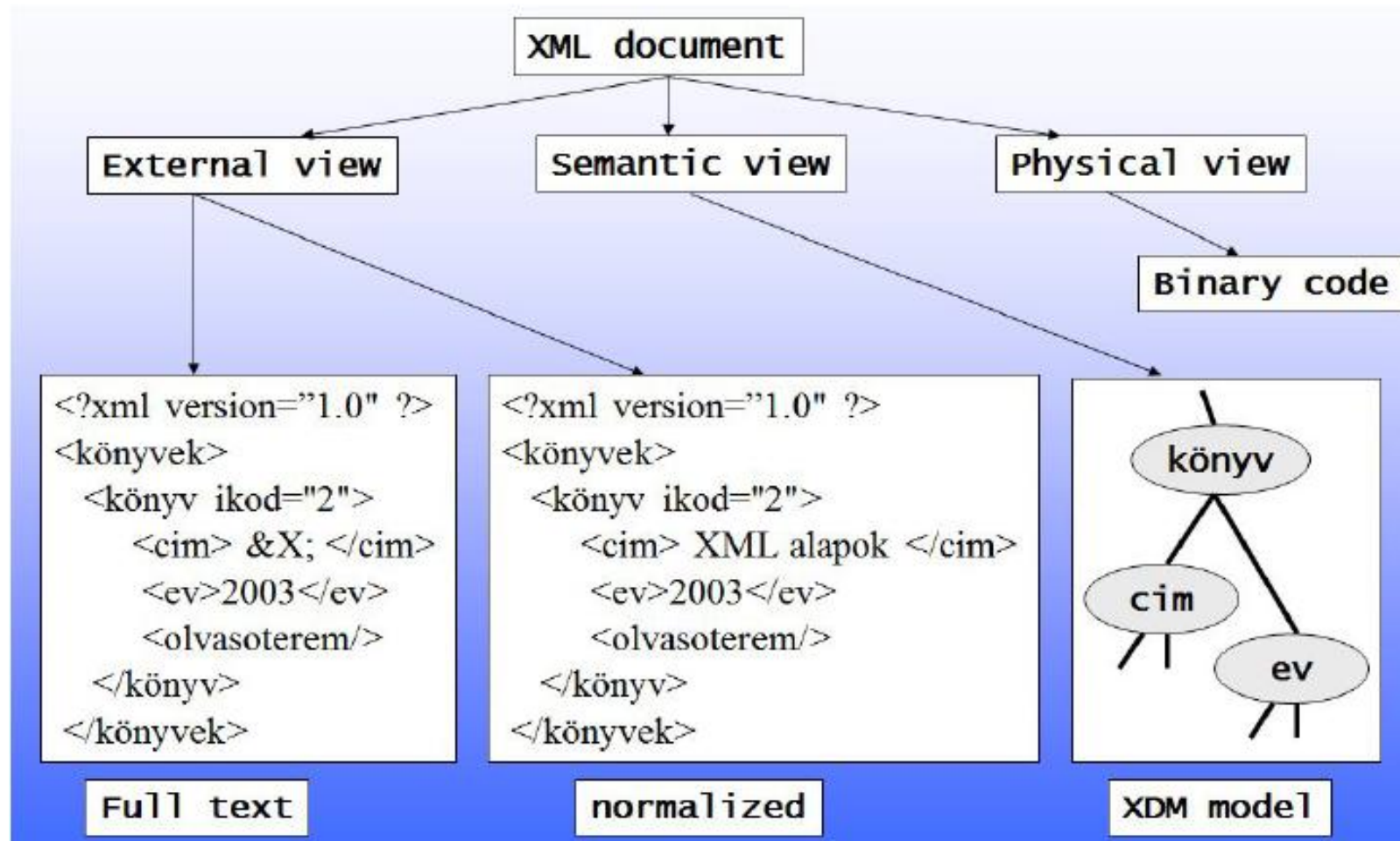
# XML - Előnyök - ismételés

- A számítógép képessé válik a *tartalom korlátozott megértésére*.
- Ezáltal lehetővé válik a tartalom *automatikus, gépi ellenőrzése*.
- Az iparban de-facto szabvány.

# XML - Hátrányok - ismételés

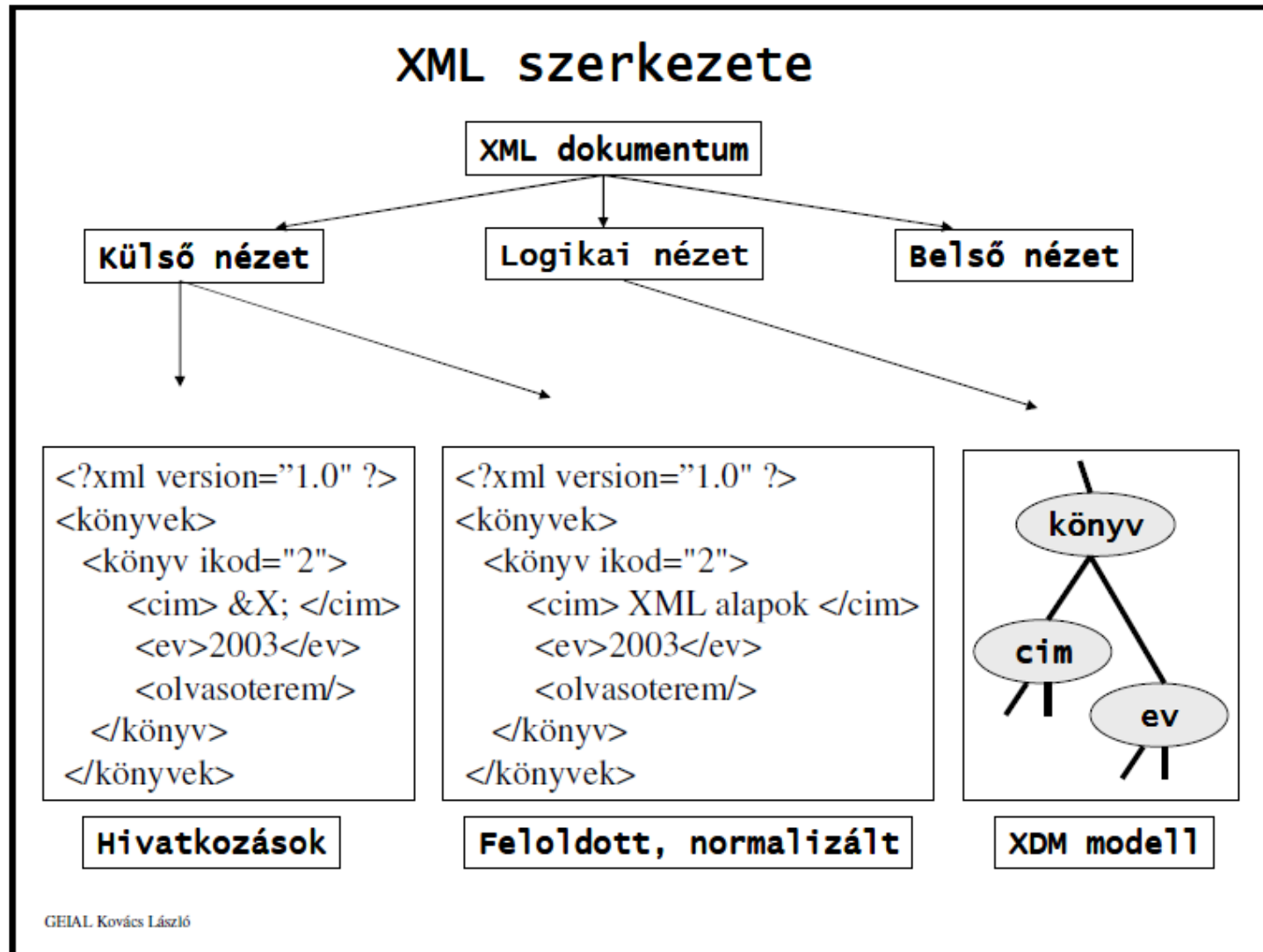
- Az eredeti szabvány *gyenge deklarációs rendszert* tartalmaz (DTD).
- A deklaráció *tervezési hibáinak javítása* igen költséges.
- *Bőbeszédű és nehézkesen* használható szintaxis.
- Nagy tárigény.
- Bonyolultság.
  - Se szeri, se száma az XML-hez *kötődő specifikációknak*.
- Mindezek ellenére fontos, együtt kell élni vele.

# XML dokumentum szerkezete - ismétlés





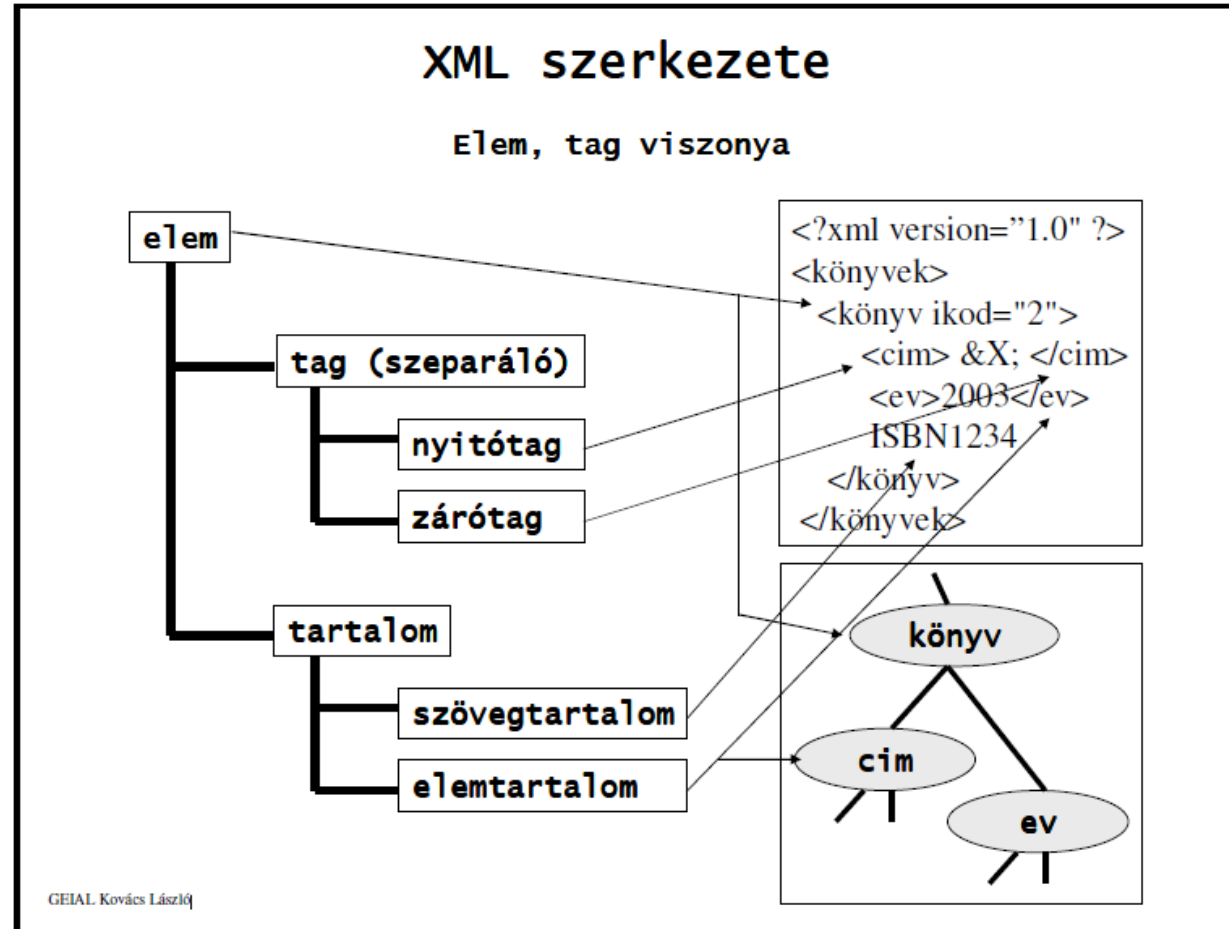
# XML dokumentum szerkezete - ismétlés



Forrás: KovácsL

# XML dokumentum szerkezete - ismétlés

Elem,  
tag  
viszonya



Forrás: KovácsL

# JSON adatmodell – új igény

## *Új igény*

- gyors, egyszerű adatcsere

## *Megvalósítás:*

- JSON: egyszerűsített XML.
- *Célja:* adatcsere.
- Napjainkban kibővül *séma* és *lekérdező nyelvekkel* is.

# JSON adatmodell - jellemzői

„A **JSON** (JavaScript Object Notation, JavaScript objektumjelölés)

- *Strukturált adatok* ábrázolására szolgál.
- *Szöveges formátum* - adatok *tárolására és továbbítására*.
- *Könnyűsúlyú szöveges* nyelvfüggetlen *adatcsere formátum*.
- Ember számára is *könnyen olvasható és írható*.
- Szoftverek által *könnyen generálható és feldolgozható*.

# JSON adatmodell - ECMAScript

Az ECMAScript programozási nyelvből származik.

URL: <http://www.json.org/>

- A jelenleg aktuális a 10-es számú kiadás.
- Ecma International, *ECMAScript 2019 Language Specification*, Standard ECMA-262, 10th Edition, June 2019.

A jelenleg fejlesztés alatt álló verzió az ECMAScript 2020.

# ECMA International

*Nemzetközi nonprofit szabványosító szervezet.*

- *Célterület:* infokommunikációs technológia (ICT), fogyasztói elektronika (CE).

Eredetileg 1961-ben alapították, jelenlegi nevén 1994 óta működik.

- *European Computer Manufacturers Association (ECMA)*

URL: <http://www.ecma-international.org/>

# JSON adatmodell - története, jellemzői

A JSON formátumot eredetileg *Douglas Crockford* specifikálta.

Douglas Crockford (amerikai számítógépes programozó és vállalkozó, aki részt vett a JavaScript nyelv fejlesztésében) volt az első, aki *meghatározta és népszerűsítette* a JSON formátumot.

A JSON-t a *State Software* cég használta 2001-től.

A JSON hivatalos *MIME-típusa* `application/json`.

A JSON fájlok kiterjesztése pedig `.json`.

# JSON adatmodell - jellemzői

*A JavaScript szkriptnyelvből alakult ki adatstruktúrák és asszociatív tömbök reprezentálására (JSON-ban objektum).*

*A JavaScripttel való kapcsolata ellenére nyelvfüggetlen, több nyelvhez is van értelmezője.*

*Példa: JSON string*

```
'{"name": „BL", "age":30, "car":null}'
```



# JSON felhasználása

- Elsősorban a *szerver* és a *webalkalmazások* közötti *adatátvitelre* használják.
- *Kialakulásával* könnyebbé tehető a *webszerveren* és a *böngészőben* futó kód közötti kommunikációja.
- *Böngészőbővítményeket* és *weboldalak* tartalmazó *JavaScript alapú alkalmazások* írásakor használják.
- *Modern programozási nyelvekkel* használható.

# JSON adatmodell - megvalósítás

JSON: egyszerűsített XML.

*Célja:* adatcsere, napjainkban kibővül *séma és lekérdező nyelvekkel* is.

*Kialakulása:* könnyebbé tegye a *webszerveren* és a *böngészőben* futó kód közötti kommunikációt.

A böngészőben futó *JavaScript kód* a kapott adatot a beépített `eval()` függvénnnyel ki tudja értékelni.

# JSON – hogyan használjuk?

**1. A szervertől kapunk egy ún. *JSON sztringet*.**

Pl.: `'{"nev":„Lilla", "kor":18, "tel":"0670", email":"nem tudom"}'`

**2. Rendeljük hozzá egy változóhoz:**

```
let ServerAnswer = '{"nev":„Lilla", "kor":18, "tel":"0670", email":"nem tudom"}',
```

# JSON – hogyan használjuk?

**3.** A *string*-et ezután egy *JavaScript függvénnnyel JavaScript objektummá* alakítjuk - azaz *parse*-juk.

```
JSON.parse()
```

```
Pl.: let obj = JSON.parse(serverAnswer);
```

A *parse* után kapott objektum: *név, kor, mobil, email*

**4.** Az attribútumnak van *értéke*, amelyhez hozzá tudunk férni:

```
let personNev = obj.nev; // „Lilla”
```

```
let personKor = obj.kor; // 18
```

# Fejlesztőkörnyezet - szabad szoftverek

## *Szabad és nyílt forrású szoftverek:*

- *Atom* (platform: Linux, macOS, Windows; licenc: *MIT License*)  
<https://atom.io/>; <https://github.com/atom/atom>  
Javasolt csomag: *Pretty JSON* <https://atom.io/packages/pretty-json>
- *Eclipse* (platform: Linux, macOS, Windows; licenc: *Eclipse Public License 2.0*) URL: <https://www.eclipse.org/>

## *Javasolt bővítmények:*

- *JavaScript Development Tools (JSDT)*  
<https://www.eclipse.org/webtools/jsdt/>

*JSON Editor Plugin*

<https://marketplace.eclipse.org/content/json-editor-plugin>

# Fejlesztőkörnyezet - szabad szoftverek

- *Notepad++* (platform: Windows; licenc: GPLv2)

<https://notepad-plus-plus.org/>

Javasolt bővítmény: *JSToolNpp*

- <http://www.sunjw.us/jstool/npp/>
- <https://github.com/sunjw/jstoolnpp>
- *Visual Studio Code* (platform: Linux, macOS, Windows; license: MIT License)
  - <https://code.visualstudio.com/>
  - <https://github.com/Microsoft/vscode>

# Fejlesztőkörnyezet - nem szabad szoftverek

*Nem szabad szoftverek:*

- *<Oxygen/> XML Editor* (platform: Linux, macOS, Windows)  
<https://www.oxygenxml.com/>  
URL: [https://www.oxygenxml.com/xml\\_editor/json\\_editor.html](https://www.oxygenxml.com/xml_editor/json_editor.html)
- *IntelliJ IDEA* (platform: Linux, macOS, Windows) – szabad lehet, ha..  
<https://www.jetbrains.com/idea/>  
<https://www.jetbrains.com/help/idea/json.html>

# JSON megjelenítés böngészőkben

*Firefox:* tartalmaz beépített JSON megjelenítőt.

- URL: [https://firefox-source-docs.mozilla.org/devtools-user/json\\_viewer/](https://firefox-source-docs.mozilla.org/devtools-user/json_viewer/)

*Google Chrome:* ajánlott kiterjesztések

- JSON Formatter  
<https://chrome.google.com/webstore/detail/jsonformatter/bcjindccaagfpajjmafaapmmgkkgkghgoa/>



# JSON adatmodell - sémai elemei

## *Séma elemei:*

- adatbázis,
- kollekció (tábla),
- dokumentum (rekord),
- *mező*: tömb, dokumentum.

Egymásba ágyazott struktúrák.

Séma mentes.

JSON and BSON formátumok.

A screenshot of a code editor window titled "Output". The editor displays a JSON schema snippet. The schema defines a "class" object containing an array of "student" objects. Each "student" object has four fields: "vezeteknev" (last name), "keresztnev" (first name), "becenev" (nickname), and "kor" (age). Three example student objects are shown in the array: one with last name "Fekete", first name "Peter", nickname "Petya", and age 22; another with last name "Kek", first name "Dora", nickname "Dorka", and age 20; and a third with last name "Zsoldos", first name "Andrea", nickname "Andi", and age 18. The code is color-coded: strings are green, numbers are blue, and punctuation is black. The editor has a dark theme and a toolbar at the top right with icons for saving, deleting, copying, and other actions.

```
1 {  
2   "class": {  
3     "student": [  
4       {  
5         "vezeteknev": "Fekete",  
6         "keresztnev": "Peter",  
7         "becenev": "Petya",  
8         "kor": 22  
9       },  
10      {  
11        "vezeteknev": "Kek",  
12        "keresztnev": "Dora",  
13        "becenev": "Dorka",  
14        "kor": 20  
15      },  
16      {  
17        "vezeteknev": "Zsoldos",  
18        "keresztnev": "Andrea",  
19        "becenev": "Andi",  
20        "kor": 18  
21      }  
22    ]  
23  }  
24 }
```

# JSON adatmodell – hierarchikus felépítés

## *Hierarchikus adatmodell*

- XML-hez hasonló séma, de
- *Nincs attribútum elem,  
nincs névtér,...*

## *Egyszerű példa:*

```
{  
  _id: <ObjectId>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```

# JSON adatmodell - szintaktikája

## Szerkezeti elemek szintaktikája:

- *struktúra*: { }
- *mező*: name : value
- *tömb*: [ v , v , ] (fontos a pozíció)

A dokumentumok azonosítása

egy *kulcsmezővel* történik:

`_id:mező`



```
1 {  
2   "class": {  
3     "student": [  
4       {  
5         "vezeteknev": "Fekete",  
6         "keresztnév": "Peter",  
7         "becenev": "Petya",  
8         "kor": 22  
9       },  
10      {  
11        "vezeteknev": "Kek",  
12        "keresztnév": "Dora",  
13        "becenev": "Dorka",  
14        "kor": 20  
15      },  
16      {  
17        "vezeteknev": "Zsoldos",  
18        "keresztnév": "Andrea",  
19        "becenev": "Andi",  
20        "kor": 18  
21      }  
22    ]  
23  }  
24 }
```

# JSON adatmodell

*A kulcsmező lehet a felhasználó által adott vagy rendszer által generált.*

A N:M kapcsolat modellezése:

- Hivatkozás az **\_id** mezőre
- Az értékek típusosak:  
(number, string)

```
{
  _id: "joe",
  name: "Joe Bookreader",
  addresses: [
    {
      street: "123 Fake Street",
      city: "Faketon",
      state: "MA",
      zip: "12345"
    },
    {
      street: "1 Some Other Street",
      city: "Boston",
      state: "MA",
      zip: "12345"
    }
  ]
}
```

# Összehasonlítás XML - JSON

## *Hasonlóságok:*

- egyszerű szövegformátum,
- emberek által olvasható-írható,
- hierarchikus felépítésű,
- JavaScriptben parse-olható,
- adatokat küldhetünk AJAX hívásokkal.

## *Különbségek:*

- nincs *end tag*,
- rövidebb,
- *gyorsabban írható és olvasható*,
- a JavaScriptben beépített `eval()` függvénnnyel parse-olható,
- *tömböket* használ,
- nincsenek *nyelvi kulcsszavak*.

# JSON vs. XML

No .	JSON	XML
1.	A JSON a JavaScript Object Notation	Az XML az eXtensible Markup Language
2.	A JSON egyszerűen olvasható és írható.	Az XML kevésbé egyszerű, mint a JSON.
3.	A JSON-t könnyű megtanulni.	Az XML kevésbé egyszerű, mint a JSON.
4.	A JSON <i>adatorientált</i> .	Az XML <i>dokumentumorientált</i> .
5.	A JSON <i>nem nyújt megjelenítési képességeket</i> .	Az XML lehetővé teszi az adatok megjelenítését, mivel ez egy jelölőnyelv.

# JSON vs. XML

6.	A JSON támogatja a tömböt.	Az XML nem támogatja a tömböt.
7.	A JSON kevésbé biztonságos, mint az XML.	Az XML biztonságosabb.
8.	A JSON fájlok emberileg olvashatóbbak, mint az XML.	Az XML fájlok kevésbé olvashatók az ember számára.
9.	A JSON csak a <i>text and number data type</i> támogatja.	Az XML számos adattípust támogat, pl.: szöveget, számot, képeket, diagramokat, grafikonokat stb.

# JSON vs. XML

## **A JSON és az XML közös jellemzői:**

- Egyszerűség (egyértelműen a JSON a nyerő).
- Az ember számára is könnyen írható és olvasható formátumok (szöveg alapú szabvány).
- Szoftverek által könnyen generálható és feldolgozható (itt is egyértelműen a JSON a nyerő).
- Formátum (egyértelműen a JSON a nyerő).



# JSON vs. XML

A fő különbség az, hogy a

- JSON *adat-orientált*,
- XML *dokumentum-orientált*.

*Adatszerkezetek* ábrázolásához a JSON tökéletes választás.

- *Előnye:* az XML-hez képest, hogy kevésbe bőbeszédű.

# JSON vs. XML

- *Dokumentum-középpontú* alkalmazásokhoz az XML-t használjuk.
- *Előnye:* a JSON-hoz képest, hogy kiterjeszthető, és hogy kiforrottabb infrastruktúra áll hozzá rendelkezésre (XML Schema, XSLT, XQuery, ...).

# XML vs. JSON - példa

```
Sample
1 <hallgato id="KPRLNM.ME">
2   <hnev>Kovács János</hnev>
3   <szulev>2004-04-05</szulev>
4   <szak evf="5">Programtervező informatikus</szak>
5 </hallgato>
```

```
Output
1 {
2   "hallgato": {
3     "hnev": "Kovács János",
4     "szulev": "2004-04-05",
5     "szak": "Programtervező informatikus"
6   }
7 }
```

# JSON adattípusok

Négy *primitív adattípus* ábrázolását teszi lehetővé:

- number,
- string,
- boolean,
- null.

Két *strukturált típus*:

- Array,
- Object.

# JSON alap adattípusai – primitív adattípus

- *Number*: lehet *lebegőpontos* és *egész* (C és Java hasonló)

Pl.: { "age" : 30 }

- *String*: idézőjelek közé zárt *Unicode karakter*, szükség szerint visszaper-jellel kivédve. Pl.: { "name" : "John" }

*A karakter egy hosszúságú karakterláncnak felel meg.*

Hasonlít a C vagy Java karakterláncaihoz.

# JSON alap adattípusai – primitív adattípusok

- *Boolean*: értéke `true` (igaz) vagy `false` (hamis).

Pl.: `{"sale":true}`

- *JSON null*: a JSON értéke `null` lehet (üres érték)

Példa:

`{"middlename":null}`

# JSON alap adattípusai – strukturált típusok

*Tömb*: értékek rendezett halmaza.

- A tömb [ ] , - nyitó és zárójel operátorokat használunk.
- Az értékeket - , (vessző)-vel választjuk el egymástól.
- Az értékeknek *nem kell azonos típusúnak* lenniük.

Példa:

```
{  
  „hallgatok”: [ "John",  "1234",  "Peter" ]  
}
```

# JSON alap adattípusai - strukturált

*Objektum: név-érték* párok rendezetlen halmaza.

Egy objektum {nyitó és záró kapcsosjel } zárul.

A ':' karakter választja el a *kulcsot* és az *értéket*.

A *név-érték* párok , (vessző)-vel tagoltak.

A *névnek*: sztringeknek kell lenni és különbözni egymástól.

*Érték lehet*: idézőjelek közé írt *karakterlánc*, *szám*, *logikai igaz/hamis*, *null*, *objektum* vagy *tömb*.



# JSON szintaxis

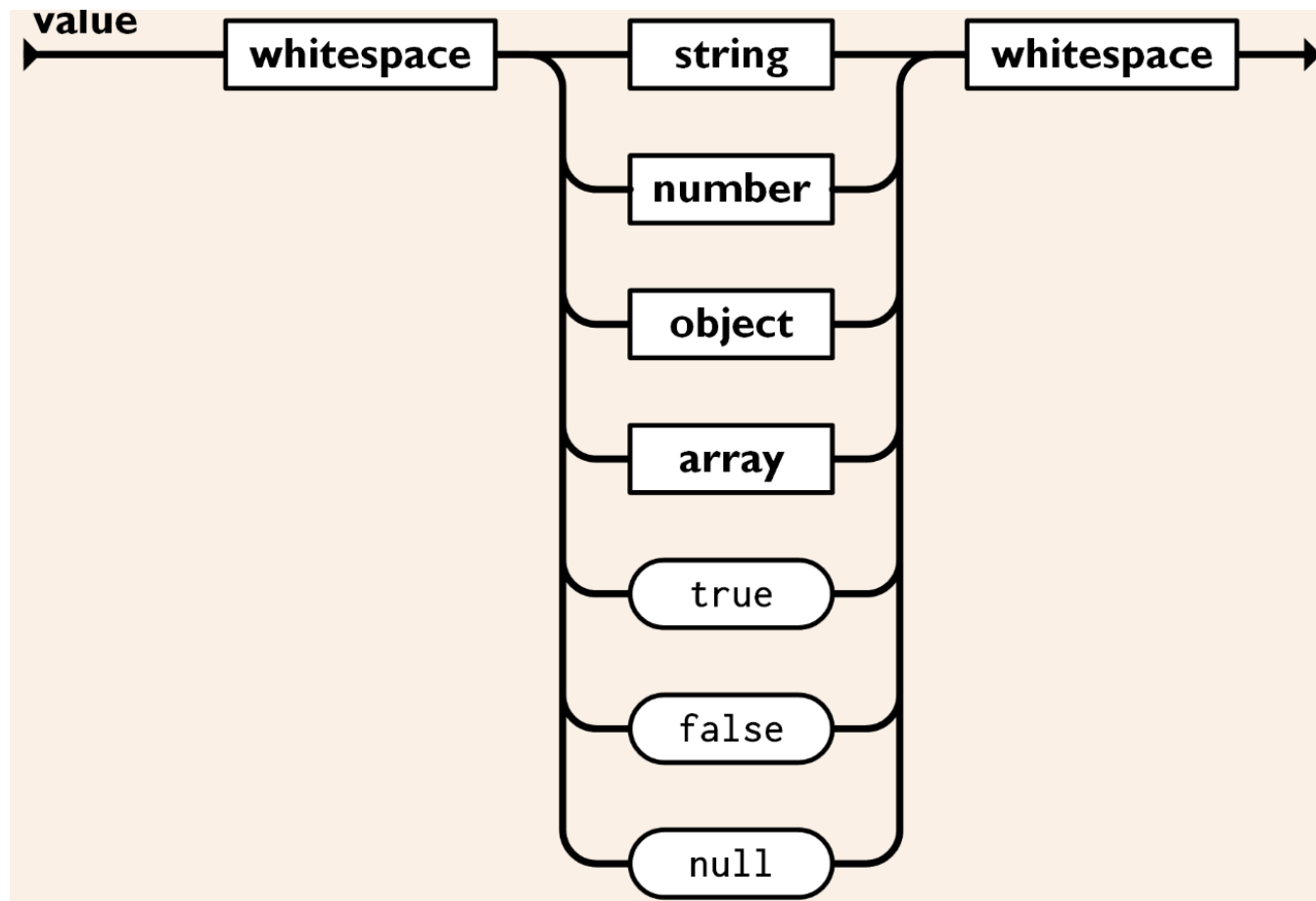
```
Sample
3 <class>
4
5 <student id = "01">
6   <vezeteknev>Fekete</vezeteknev>
7   <keresztnev>Peter</keresztnev>
8   <becenev>Petya</becenev>
9   <kor>22</kor>
10 </student>
11
12 <student id = "02">
13   <vezeteknev>Kek</vezeteknev>
14   <keresztnev>Dora</keresztnev>
15   <becenev>Dorka</becenev>
16   <kor>20</kor>
17 </student>
18
19 <student id = "03">
20   <vezeteknev>Zsoldos</vezeteknev>
21   <keresztnev>Andrea</keresztnev>
22   <becenev>Andi</becenev>
23   <kor>18</kor>
24 </student>
25 </class>
```

```
Output
1 {
2   "class": {
3     "student": [
4       {
5         "vezeteknev": "Fekete",
6         "keresztnev": "Peter",
7         "becenev": "Petya",
8         "kor": 22
9       },
10      {
11        "vezeteknev": "Kek",
12        "keresztnev": "Dora",
13        "becenev": "Dorka",
14        "kor": 20
15      },
16      {
17        "vezeteknev": "Zsoldos",
18        "keresztnev": "Andrea",
19        "becenev": "Andi",
20        "kor": 18
21      }
22    ]
23  }
24 }
```

# Tokenek

- A JSON szöveg tokenek olyan sorozata, mely megfelel a JSON érték nyelvtani szabálynak.
- Tokenek:
  - Szerkezeti tokenek a {, }, [, ], : és , karakterek.
  - Sztringek
  - Számok
  - Literális tokenek a true, false és null karakterláncok.
- Tokenek előtt és után megengedettek *whitespace* karakterek, melyek nem lényegesek.
  - *Whitespace* karakter: HT (U+0009), LF (U+000A), CR (U+000D), szóköz (U+0020)
  - A tokenek közül csak a sztringekben megengedettek *whitespace* karakterek. (Forrás: DE)

# JSON értékei - értékek

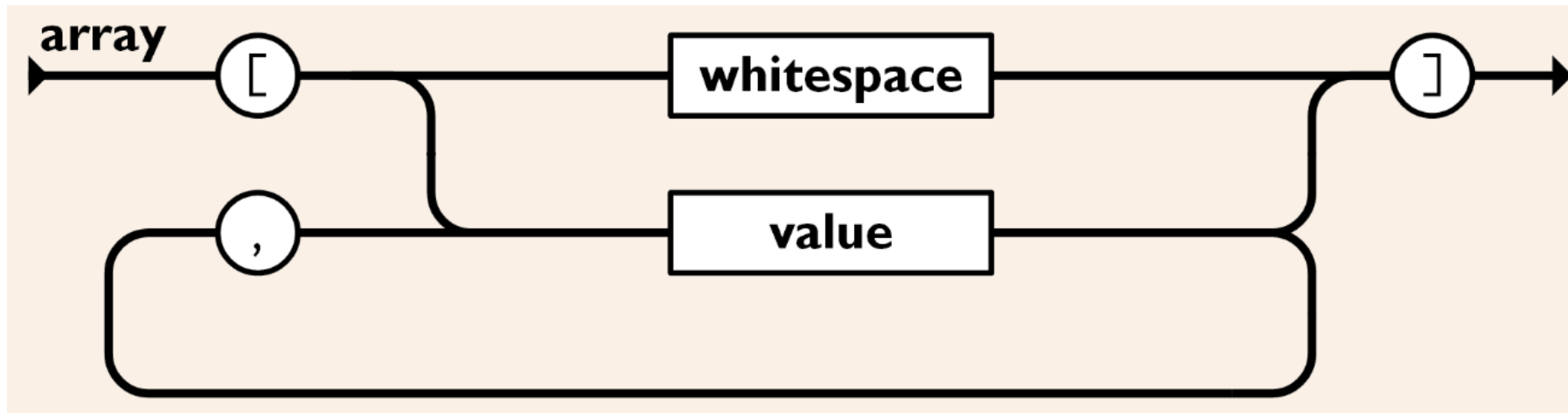


# Sztringek

- *Unicode karakterek sorozatai*, melyeket *idezőjelek* (U+0022) határolnak.
- *Bármely karaktert* tartalmazhatják, azonban az alábbiakat csak levédve:
  - *idézőjel* (U+0022), *backslash* (U+005C), *vezérlő karakterek* (U+0000–U+001F)
- Speciális karakterek megadásához rendelkezésre állnak az *escape* szekvenciák: `\", \\, \t, \n, \r, ...`

# Tömbök

- Tetszőleges *számú érték rendezett sorozata* (lehet üres).
  - Az elemek *különböző típusúak* is lehetnek.
  - Hivatkozás az elemre a elem számmal pl. [1] lehet



## Tömbök - példa

- ["Athos", "Porthos", "Aramis", "d'Artagnan"]
- [9, 14, 19, 25, 26, 28]
- ["Pi", 3.141593, null, true]
- [[45.7370889, 16.1133866], [48.5852340, 22.8981217]]

(Forrás: DE)

# JSON Array - példa

## *JSON Array of String*

```
[ "Vasárnap" , "hétfő" , "kedd" , "szerda" ,  
  "csütörtök" , "péntek" , "szombat" ]
```

## *JSON Array of Numbers*

```
[12, 34, 56, 43, 95]
```

## *JSON Array of Booleans*

```
[true, true, false, false, true]
```

# JSON Array - példa

## *JSON Array of Objects*

```
{  
  „hallgatok”: [  
    { "name": „Olga", "email": „olga@gmail.com", „kor”:40  },  
    { "name": „Dóra", "email": „dori10@gmail.com", „kor”:33},  
    { "név": „Lilla", "email": "lilla@gmail.com", "kor":22},  
    { "name": „Sara", "email": „sara@gmail.com", „kor”:11}]  
}
```



# JSON Array - példa

## JSON többdimenziós tömb

```
[  
  [ "a" , "b" , "c" ],  
  [ "m" , "n" , "o" ],  
  [ "x" , "y" , "z" ]  
]
```

# JSON Comments

A JSON nem támogatja a *megjegyzéseket*. Ez nem szabvány.

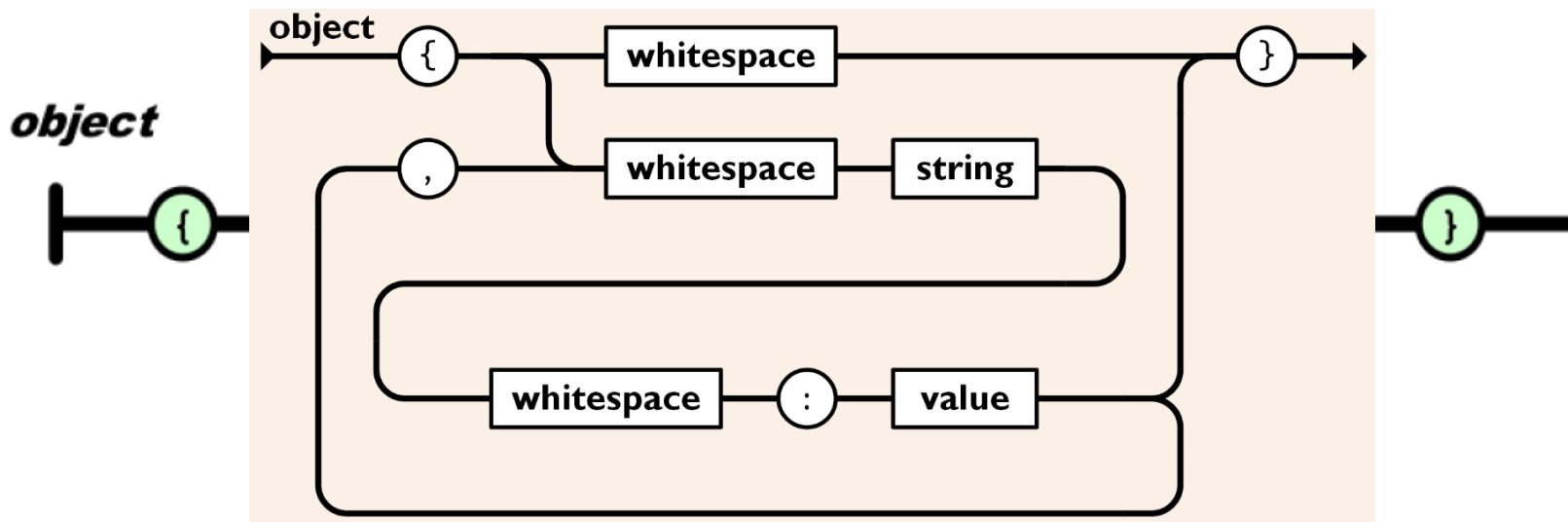
De hozzáadhat egy un. *extra attribútumot* a megjegyzéshez a JSON objektumban.

```
{
  "alkalmazott" : {
    "név" : „Lili” ,
    "fizetés" : 560000 ,
    "megjegyzés" : "Kedves Sara"
  }
}
```

# JSON Objects

Tetszőleges számú *név-érték* párból állnak.

- A *név* tetszőleges *string*, az *érték* tetszőleges *JSON érték*.
- A *név-érték* párokra a **tag (*member*)** elnevezést is használjuk.



# JSON Objects - példa

*JSON objektum*

```
{  
  „hallgato” : {  
    „név” :      ”Dóra” ,  
    "fizetés" :   560000,  
    "házas" :    ”férjezett”  
  }  
}
```

*JSON objektum karakterláncokkal*

```
{  
  "name" : „dora” ,  
  "email" : „dora2022@gmail.com”  
}
```

# JSON Objects - számokkal

A JSON *lebegőpontos formátumban* támogatja a számokat.

A szám lehet:

- *egész* (0-9),
- *tört* ((33, .532 stb.) és
- *kitevő* (e, e +, e-, E, E +, E-).

```
{  
  "egész szám" : 34,  
  "tört" : 1.2145,  
  "exponent" : 6.61789e + 0  
}
```

# JSON Objects – logikai érték

JSON objektum *logikai elemekkel*

```
{  
  "első" : igaz,  
  "második" : hamis  
}
```

JSON beágyazott objektum

```
{  
  „keresztnev” : „Sara” ,  
  „vezeteknev” : „Kis” ,  
  „eletkor” : 27 ,  
  „cim” : {  
    „utca” : „Feher 2” ,  
    „varos” : „Miskolc” ,  
    „iranyitoszam” : „3515”  
  }  
}
```

# Karakterkódolás

*RFC (Request For Comments) 8259:*

JSON szöveg különböző rendszerek közötti átvitelekor az *UTF-8 karakterkódolást* kell használni.

# XML vs. JSON – melyiket használjuk?

Több ezer karakter rögzítése esetén pl.: *5000 karakter XML-be*, akkor a mérleg a JSON felé tolódik (kevesebb adat jut át az egyik szerverről a másikra vagy vissza).

Tehát, az *adat továbbítása szempontjából* a JSON oldalán szól.

A *JSON validálásának* ellenőrzése szintén *több URL* rendelkezik. Például:

[URL: https://JSONLint.com/](https://JSONLint.com/)

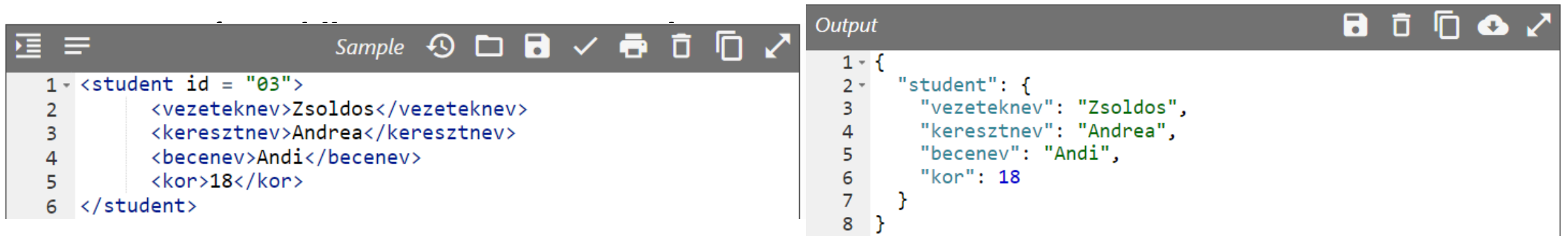
Itt ellenőrizhető a JSON megfogalmazás.



# XML vs. JSON

*Fontos:* az átalakítást mindig egy *parse-l* végezze.

- Önerőből NE írjunk JSON-t – erre valók a *parse-k*.



The screenshot displays a web application for converting XML to JSON. On the left, under the 'Sample' tab, is an XML document for a student. On the right, under the 'Output' tab, is the resulting JSON structure.

```
1 <student id = "03">
2   <vezeteknev>Zsoldos</vezeteknev>
3   <keresztnev>Andrea</keresztnev>
4   <becenev>Andi</becenev>
5   <kor>18</kor>
6 </student>
```

```
1 {
2   "student": {
3     "vezeteknev": "Zsoldos",
4     "keresztnev": "Andrea",
5     "becenev": "Andi",
6     "kor": 18
7   }
8 }
```

<http://www.utilities-online.info/xmltojson/?save=bbda4cb9-17d0-4277-9829-4f8219d7dbc2-xmltojson#.Xd-KR-hKhPY>

# JSON

A JSON-be a feltűnhet, hogy *számokat* idézőjel nélkül tárolja, ha *szövegként szeretnénk* értelmezni, akkor „” kell tenni.

Az XML parse-nél külön meg kell mondani.

*XML to JSON Converter:*

<https://www.freeformatter.com/xml-to-json-converter.html>

# XML to JSON konverter

URL: <https://codebeautify.org/xmltojson>

```
Sample
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!-- <!DOCTYPE kurzusfelvetel SYSTEM "kurzusfelvetel.dtd"
4 >-->
5
6 <!-- <kurzusfelvetel tanev="2022_2023 I." egyetem="ME">-->
7
8 <kurzusfelvetel tanev="2022/2023I" egyetem="ME" xmlns:xsi
9   ="http://www.w3.org/2001/XMLSchema-instance" xsi
10  :noNamespaceSchemaLocation="kurzusfelvetel1.xsd">
11
12   <hallgato id="KPRLNM.ME">
13     <hnev>Kovács János</hnev>
14     <szulev>2004-04-05</szulev>
15     <szak evf="5">Programtervező informatikus</szak>
16   </hallgato>
17
18   <kurzusok>
19     <kurzus id="GEIAL332-B">
20       <kurzusnev>Adatkezelés XML környezetben</kurzusnev>
21       <kredit>5</kredit>
22       <hely>XXXII.</hely>
23       <idopont>Kedd 12:00-13:30</idopont>
24     </kurzus>
25   </kurzusok>
26 </kurzusfelvetel>
27
```

Ln: 41 Col: 17 size: 1.39 KB

```
Output
1 {
2   "kurzusfelvetel": {
3     "hallgato": {
4       "hnev": "Kovács János",
5       "szulev": "2004-04-05",
6       "szak": "Programtervező informatikus"
7     },
8     "kurzusok": {
9       "kurzus": [
10        {
11          "kurzusnev": "Adatkezelés XML környezetben",
12          "kredit": 5,
13          "hely": "XXXII.",
14          "idopont": "Kedd 12:00-13:30",
15          "oktato": "Dr. Bednarik László"
16        },
17        {
18          "kurzusnev": "Adatbázis rendszerek I.",
19          "kredit": 5,
20          "hely": "In101",
21          "idopont": "Kedd 08:00-09:30",
22          "oktato": "Dr. Kovács László"
23        }
24      ]
25    }
26  }
27
```

Ln: 35 Col: 1 size: 899 B

# XML to JSON konverter

<http://www.utilities-online.info/xmltojson/?save=bbda4cb9-17d0-4277-9829-4f8219d7dbc2-xmltojson#.Xd-KR-hKhPY>

## XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="autok5.xsl" ?>
<autok>
  <auto rsz="ABC-100">
    <tipus> Fiat </tipus>
    <ar> 21233 </ar>
    <szin>piros</szin>
    <tulaj>
      <nev>Zoli</nev>
      <varos>Eger</varos>
    </tulaj>
  </auto>
  <auto rsz="ABC-101">
    <tipus> Skoda </tipus>
    <ar> 44233 </ar>
    <szin>fehér</szin>
    <tulaj>
      <nev>Peti</nev>
      <varos>Miskolc</varos>
    </tulaj>
  </auto>
  ..
  ..
```

## JSON

```
{
  "autok": {
    "auto": [
      {
        "-rsz": "ABC-100",
        "tipus": " Fiat ",
        "ar": " 21233 ",
        "szin": "piros",
        "tulaj": {
          "nev": "Zoli",
          "varos": "Eger"
        }
      },
      {
        "-rsz": "ABC-101",
        "tipus": " Skoda ",
        "ar": " 44233 ",
        "szin": "fehér",
        "tulaj": {
          "nev": "Peti",
          "varos": "Miskolc"
        }
      }
    ]
  }
}
```

# JSON Schema

A JSON-séma egy *deklaratív nyelv*, amely lehetővé teszi a JSON-dokumentumok *annotációját* és *validálást*.

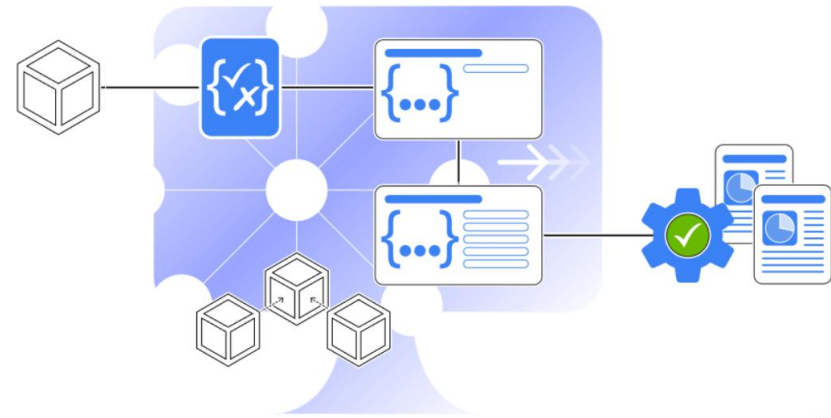
*JSON dokumentumok* érvényesítéséhez *JSON-alapú sémanyelvet* használunk.

- Webhely: <https://json-schema.org/>
- Aktuális verzió: 2020. 12.
- A legszélesebb körben támogatott verzió a 2018-ban kiadott draft-07 verzió.

# JSON Schema jellemzői

- A JSON-séma *szabályokat és megszorításokat* határozhat meg.
- Ha a JSON-dokumentumok megfelelnek ezeknek a korlátozásoknak, könnyebbé válik a *strukturált adatok cseréje az alkalmazások között*.
- A JSON-dokumentumok a *szerverek és alkalmazások közötti adatok tárolására és átvitelére* szolgálnak.

# JSON Schema -előnyei



## *Előnyei:*

- Leírja a meglévő data format(s).
- Ember és gép által olvasható dokumentációt biztosít.
- Ellenőrzi azokat az adatokat, amelyek:
  - Automatizált tesztelés.
  - Kliens által használt adatok minőségének ellenőrzése.

# JSON Schema - jellemzői

A **JSON-séma** egy olyan szabvány, amely lehetővé teszi a JSON (JavaScript Object Notation) dokumentumok *szerkezetének és tartalmának leírását*.

- A JSON séma használható *JSON adatok érvényesítésére*, így ellenőrizheti, hogy az *adatok megfelelnek-e a kívántnak, típusoknak és értékhatároknak*.



# JSON Schema - egyszerűsített séma

## *Jellemzők:*

- adattípus korlátozás,
- értékkorlátozás,
- szerkezeti kényszer,
- kötelező elem,
- opcionális elem,
- referenciák

JSON Validate

Schema in json format:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type" : "**",  
}
```

# JSON Schema - jellemzői

## *JSON schema type definitions:*

- atomic: "type" : "datatype"
- object: "type": "object",  
          "properties": { content }
- array: "type": "array",  
          "items": { content }
- content: fieldname1: type1,  
          fieldname2:type2,...

## *Constraints:*

- "additionalProperties" :  
  false
- "required" :  
  ["fname1","fname2",...]

## *Elementary datatypes:*

- Integer, number, string, boolean

# JSON Schema - jellemzői

## JSON Schema

```
1 {  
2   "$schema": "http://json-schema.org/draft-04/schema#",  
3   "type" : "object",  
4   "properties": {  
5     "nev" : {  
6       "type" : "string"  
7     },  
8     "kor" : {  
9       "type" : "integer"  
10    }  
11  },  
12  
13  "additionalProperties": false  
14  
15 }
```

## JSON Content

```
1 { "nev": "Zoli", "kor": 23 }  
2  
3  
4  
5  
6  
7  
8  
9
```

# JSON Schema - példa

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "tags": {
      "type": "array",
      "minItems": 2,
      "maxItems": 4,
      "items": {
        "enum": [
          "Open Source", "Java", "JavaScript", "JSON", "REST"
        ]
      }
    }
  },
  "additionalProperties": false,
  "required": ["tags"]
}
```

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "email": {
      "type": "string",
      "pattern": "^[\\w|-\\.]+@[\\w]+\\. [A-Za-z]{2,4}$"
    },
    "firstName": {
      "type": "string"
    },
    "lastName": {
      "type": "string"
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "favoriteTopic": {
      "type": "string"
    }
  },
  "additionalProperties": false,
  "required": ["email", "firstName", "lastName"],
  "dependencies": {
    "favoriteTopic": ["tags"]
  }
}
```

# JSON séma fő elemei

- *\$schema*: A JSON Schema verziójának URL-je (pl. `"http://json-schema.org/draft-07/schema#"`).
- *type*: milyen típusú értéket várunk (pl. *object*, *array*, *string*, *number*, stb.).
- *properties*: Az objektum *kulcsait* és az *értékeket* definiálja.
- *required*: Az objektum azon *mezőit sorolja fel*, akiknek *kötelező szerepelniük*.

# JSON séma fő elemei

- *additionalProperties*: Meghatározza, hogy adott-e további tulajdonságok szerepeltetése az objektumban.
- *items*: Ha egy tömböt definiálunk, az *items* a mező írja le, hogy *milyen típusú elemek lehetnek* benne.
- *enum*: Felsorolja azokat az értékeket, melyeket egy *mezőben* *rendeltem*.
- *minLength*, *maxLength*, *minimum*, *maximum* : Különféle megszorítások (pl. minimális, maximális értékek, hosszúságok stb.).

# Maven függőségek - példa

Először hozzá kell adnunk a szükséges függőségeket a *pom.xml* fájlhoz, ha Maven-t használunk:

## 1. lépés

```
<dependencies>
  <dependency>
    <groupId>org.everit.json</groupId>
    <artifactId>org.everit.json.schema</artifactId>
    <version>1.14.1</version>
  </dependency>
  <dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20230227</version>
  </dependency>
</dependencies>
```

A könyvtárak segítségével válik a JSON séma és a JSON adat validálása.

# JSON-schema

## JSON

```
{
  "nev": "Fekete Péter",
  "kor": 30,
  "email": "peter.fekete@gmail.com",
  "aktiv": true,
  "baratok": ["Lili", "Mark"]
}
```

A JSON egy objektum ( type: "object"), vannak különböző tulajdonságai ( properties).

*név*: Egy kötelező mező, stringnek kell lennie, és legalább 1 karakter hosszúnak.

*kor*: Szintén kötelező, egész szám, értéke nem lehet kisebb mint 0.

*email*: Kötelező string típusú mező, érvényes email címnek kell lennie.

*aktiv*: Egy opcionális logikai érték (boolean), amely a felhasználó aktív állapotát jelzi.

*baratok*: Egy opcionális tömb, nyelv string típusú barátok nevei szerepelnek.

additionalProperties: false: Ez biztosítja, hogy ne lehessen egyéb, nem definiált tulajdonságokat kell hozzáadni az objektumhoz.

## JSON-schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "nev": {
      "type": "string",
      "minLength": 1
    },
    "kor": {
      "type": "integer",
      "minimum": 0
    },
    "email": {
      "type": "string",
      "format": "email"
    },
    "aktiv": {
      "type": "boolean"
    },
    "baratok": {
      "type": "array",
      "items": {
        "type": "string"
      }
    }
  },
  "required": ["nev", "kor", "email"],
  "additionalProperties": false
}
```



# JSON-séma

## 2. lépés

Az alábbi JSON séma leírja az érvényesíteni kívánt adatokat:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "nev": {
      "type": "string",
      "minLength": 1
    },
    "kor": {
      "type": "integer",
      "minimum": 0
    },
    "email": {
      "type": "string",
      "format": "email"
    },
    "aktiv": {
      "type": "boolean"
    },
    "baratok": {
      "type": "array",
      "items": {
        "type": "string"
      }
    }
  },
  "required": ["nev", "kor", "email"],
  "additionalProperties": false
}
```

# Java kód a validáláshoz

**3. lépés:** Java nyelven végrehajtani a JSON validálást a megadott séma alapján.

# Java kód - magyarázat

- *SchemaLoader.load()*: A JSON séma betöltésére szolgál.
- *schema.validate()*: Ez a *metódus* a tényleges érvényesít. Ha a JSON adat nem felel meg a séma szabályainak, kivételt dob.
- *ValidationException*: Ha a *validálás sikertelen*, a program *kivételt kezel*, és *kiírja a hibát*, amely részletezi, hogy miért érvénytelen a JSON.

# Kimenet

- Ha a validáció sikeres (pl. ha a JSON adat megfelel a schemának), a következő üzenet jelenik meg:

**A JSON adat érvényes**

# BSON - Binary JSON

BSON - (*bee – sahn*) bináris *adatcsere* formátum.

- Specifikáció: <http://bsonspec.org/>

A *MongoDB* NoSQL adatbázis-kezelő rendszer használja.

URL: <https://www.mongodb.org/>

- *Adattárolás és hálózati adatátvitel.*

**BSON** { 01000010  
01010011  
01001111  
01001110 }

<https://bsonspec.org/>

# BSON - Binary JSON

A JSON adattípusainak kiterjesztése:

Pl.: dátum típus, időbélyeg, BinType, ...

- Nincs azonban *number* adattípus, helyette *int32*, *int64* és *double* adattípusok használata.

- *Basic types*

<code>byte</code>	1 byte (8-bits)
<code>int32</code>	4 bytes (32-bit signed integer, two's complement)
<code>int64</code>	8 bytes (64-bit signed integer, two's complement)
<code>uint64</code>	8 bytes (64-bit unsigned integer)
<code>double</code>	8 bytes (64-bit IEEE 754-2008 binary floating point)
<code>decimal128</code>	16 bytes (128-bit IEEE 754-2008 decimal floating point)

# BSON - Binary JSON

BSON is támogatja a *dokumentumok* és *tömbök beágyazását* más *dokumentumokba* és *tömbökbe*.

*BSON jellemzői:*

- *Könnyűsúlyú:* hálózati adatátvitel.
- *Átjárható:* MongoDB használják.
- *Hatékony:* *kódolás* és *dekódolás* gyorsan végrehajtható, mert *C nyelv* adattípusait használja.

# JSON vs. BSON közötti különbségek

JSON	BSON
A JSON a JavaScript objektum jelölése.	A BSON egy bináris Javascript objektum jelölés.
Ez egy szabványos fájlformátum.	Ez egy bináris fájlformátum.
A JSON néhány alapvető adattípust tartalmaz, pl.: karakterláncot, számokat, logikai értéket és nullát.	A BSON tartalmaz néhány további adattípust, pl. dátumot, időbélyeget stb.
Adatbázisok, mint az <i>AnyDB</i> , a <i>Redis</i> stb., JSON formátumban tárolják az adatokat.	A <i>MongoDB</i> adatait BSON formátumban tárolják.
A JSON kevesebb helyet igényel a BSON-hoz képest.	A BSON több helyet igényel, mint a JSON.



# JSON vs. BSON közötti különbségek

Viszonylag kevésbé gyorsabb, mint a BSON.	Gyorsabb a BSON-hoz képest.
Adatátvitelre szolgál.	Adatok tárolására szolgál.
JSON-fájlból való olvasás során, <i>végig kell mennie a teljes tartalma</i> .	A BSON-ban az <i>indexet</i> használják,
A JSON formátumot nem kell elemezni, mivel az már ember által olvasható.	Elemezni kell, mivel a gépek könnyen értelmezhetik.
A JSON objektumok és tömbök kombinációja, ahol az objektum kulcs-érték párok gyűjteménye, míg a tömb az elemek rendezett listája.	A BSON további információkat nyújt, pl.: a karakterlánc hosszát és az objektum altípusait. A BinData és a dátum a BSON által támogatott további adattípusok.

# GSON – Google JSON

A GSON egy nyílt forráskódú Java-könyvtár, amely Java objektumokat *JSON-ba serializál és deserializál*.



- A GSON a Google JSON *parser* és *Java generátor*.
- A Google belső használatra fejlesztette ki a GSON-t, de később nyílt forráskódú.

## *Download Gson Archive*

- Download: `gson-2.3.1.jar` – Javában használóknak.

*GSON User Guide* – hasznos információk

<https://sites.google.com/site/gson/gson-user-guide>

# BOON

- A *Boon* egy Java alapú eszköz, amellyel a JSON adatokat *hatékonyan és gyors módon kódolható vagy dekódolható*.
- Használhatjuk a *Boon JSON parser-t*, ha befoglaljuk a *Boon JAR* fájlt a Java alkalmazásba.

További hasznos információ:

[https://www.tutorialspoint.com/boon/boon\\_quick\\_guide.htm](https://www.tutorialspoint.com/boon/boon_quick_guide.htm)

<https://jenkov.com/tutorials/java-json/gson-installation.html>

# BOON

## Download Boon Archive

- *Download:* `boon-0.34.jar`

URL: <https://mvnrepository.com/artifact/io.fastjson/boon>

# JSON kérdések

Több kérdés is felmerülhet a JSON-nél?

- Kérdés van egyszerűbb megoldás a JSON-nél?
- Miért kellene még mindig operátorok?
- Miért nem lehet e nélkül elkészíteni a feladatot?

Válasz: LEHET

# YAML formátum

*YAML formátum* (YAML - Nem Markup Language), amely egy újabb módja az *adatok tárolásának*.

A fájl kiterjesztése: `.yaml`

A YAML egy adat **sorosító nyelv** (szerializáció), - közvetlenül *olvasható és írható* emberi szemmel.

*Láncolt lista és körkörös hivatkozások* kezelésére is képes.

*Célja:* a memóriában tárolt adatok egyszerű lemezre mentése és visszatöltése.

# JSON - YAML formátum konvertálás

JSON-ről YAML-ra is van lehetőség konvertálni.

<https://www.json2yaml.com/>

```
{  
  "Person": {  
    "name": „Lilla”,  
    "age": 20  
  }  
}
```

---

```
Person:  
  name: Lilla  
  age: 20
```

*A 3 kötőjel jelzi a fájl kezdetét.*

Legnagyobb különbség a formátumok között az a *nyelvtani*.

Mindegyik hűen ábrázolja egy objektum struktúráját.

# YAML formátum jellemzői

- Az `.yaml` fájlok '---' 3 kötőjellel kezdődnek, jelezve a dokumentum kezdetét.
- A kulcsérték-párokat: *kettőspont* választja el.
- A listák - **kötőjellel** kezdődnek.

*Validálása*

URL: <https://codebeautify.org/xml-to-yaml>

*Konvertáló*

URL: <https://codebeautify.org/xml-to-yaml>



# XML - YAML konvertálás - mintapélda

```
Sample
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!--<!DOCTYPE orarend SYSTEM "BL_orarend.dtd">-->
4
5 <!--<orarend>-->
6
7 <orarend xmlns:xs="http://www.w3.org/2001/XMLSchema
  -instance"
8   xs:noNamespaceSchemaLocation="BL_orarend.xsd">
9
10
11   <ora id="1" tipus="előadás">
12     <targy>Adatkezelés XML-es környezetben</targy>
13     <idopont>
14       <nap>kedd</nap>
15       <tol>14</tol>
16       <ig>16</ig>
17     </idopont>
18     <helyszin>In103</helyszin>
19     <oktato>Bednarik László</oktato>
20     <szak>Mérnök-informatikus BSc</szak>
21   </ora>
22   <ora id="2" tipus="gyakorlat">
23     <targy>Adatkezelés XML-es környezetben</targy>
```

Ln: 44 Col: 10 size: 1.27 KB

```
Output
1 orarend:
2   ora:
3     - targy: Adatkezelés XML-es környezetben
4       idopont:
5         nap: kedd
6         tol: 14
7         ig: 16
8       helyszin: In103
9       oktato: Bednarik László
10      szak: Mérnök-informatikus BSc
11     - targy: Adatkezelés XML-es környezetben
12       idopont:
13         nap: szerda
14         tol: 10
15         ig: 12
16       helyszin: In101
17       oktato: Bednarik László
18      szak: Mérnök-informatikus BSc
19     - targy: Adatkezelés XML-es környezetben
20       idopont:
21         nap: szerda
22         tol: 12
23         ig: 14
24       helyszin: In103
```

Ln: 27 Col: 0 size: 643 B

# JSON - YAML konvertálás - mintapélda

JSON ✓	YAML
1 {	1 ---
2   "vizsgak": {	2 vizsgak:
3     "vizsga": [	3   vizsga:
4       {	4     - kurzus: XML
5         "kurzus": "XML",	5       hely: In/101
6         "hely": "In/101",	6       idopont:
7         "idopont": {	7         nap: szerda
8           "nap": "szerda",	8         tol: 10
9           "tol": 10,	9         ig: 12
10          "ig": 12	10        oktato: BL
11        },	11        jegy: 5
12        "oktato": "BL",	12     - kurzus: Angol
13        "jegy": 5	13       hely: In/102
14      },	14       idopont:
15      {	15         nap: szerda
16        "kurzus": "Angol",	16         tol: 10
17        "hely": "In/102",	17         ig: 12
18        "idopont": {	18        oktato: FK
19          "nap": "szerda",	19        jegy: 5
20          "tol": 10,	20     - kurzus: DB1
21          "ig": 12	21       hely: In/102
22        },	22       idopont:
23        "oktato": "FK",	23         nap: szerda
24        "jegy": 5	24         tol: 10
25      },	25         ig: 12
26      {	26        oktato: KL
27        "kurzus": "DB1",	27        jegy: 5
28        "hely": "In/102",	28
29        "idopont": {	
30          "nap": "szerda",	
31          "tol": 10,	
32          "ig": 12	

# JSON - TEXT konvertálás - mintapélda

```
Sample
1 {
2   "class": {
3     "student": [
4       {
5         "vezeteknev": "Fekete",
6         "keresztnev": "Peter",
7         "becenev": "Petya",
8         "kor": 22
9       },
10      {
11        "vezeteknev": "Kek",
12        "keresztnev": "Dora",
13        "becenev": "Dorka",
14        "kor": 20
15      },
16      {
17        "vezeteknev": "Zsoldos",
18        "keresztnev": "Andrea",
19        "becenev": "Andi",
20        "kor": 18
21      }
22    ]
23  }
24 }
```

Ln: 24 Col: 1      size: 485 B

```
Output
1 vezeteknev keresztnev becenev kor
2 Fekete Peter Petya 22
3 Kek Dora Dorka 20
4 Zsoldos Andrea Andi 18
5
```

Ln: 5 Col: 0      size: 112 B

# YAML formátum

A YAML formátummal *gyorsan tudunk adatokat továbbítani* az interneten az egyik helyről a másikra.

Közvetlenül is tudunk *XML-ről YAML konvertálni*:

<https://jsonformatter.org/xml-to-yaml>

<https://www.json2yaml.com/>

<https://codebeautify.org/xml-to-yaml>

# JSON – JAVA, JAVACRIPT, PHP, AJAX

A különböző programozási nyelvek lehetőséget adnak *a JSON* objektumok *készítésére/konvertálására*.

- **JSON with Java**
- JSON with PHP
- JSON with Python
- JSON with Perl
- JSON with Ruby
- **JSON with JavaScript**

# Java JSON

A **json.simple** könyvtár lehetőséget biztosít, hogy *JSON-adatokat olvassunk és írjunk Java-ban*.

Tehát, *kódolhatjuk és dekódolhatjuk* a JSON objektumot a Java-ban a **json.simple** könyvtár használatával.

# Java JSON – package instal

Instal *json.simple.jar*

A *json.simple* telepítéséhez be kell állítania a *json.simple.jar* környezeti változót, vagy hozzá kell adnia a Maven függőséget.

URL: <http://www.java2s.com/Code/Jar/j/Downloadjsonsimple11jar.htm>

**2.** A maven dependency hozzáadásához a következő kódot kell az *pom.xml* fájlba beírni.

<https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple>

```
<függőség>  
  <groupId> com.googlecode.json-simple </groupId>  
  <artifactId> json-simple </artifactId>  
  <version> 1.1 </version>  
</dependency>
```

# Java JSON API

Az *org.json.simple 2.1.2.jar* csomag a *JSON API* fontos osztályait *tartalmazza*.

Hasznos információk:

URL: <https://stleary.github.io/JSC>

[parser]	
DeserializationException	class
DeserializationException\$1	class
DeserializationException\$Problems	class
ItemList	class
Jsonable	class
JsonArray	class
JSONAware	class
Jsoner	class
Jsoner\$1	class
Jsoner\$DeserializationOptions	class
Jsoner\$SerializationOptions	class
Jsoner\$States	class
JSONObject	class
JSONStreamAware	class
JSONValue	class
Yylex	class
Yytoken	class
Yytoken\$1	class
Yytoken\$Types	class



# Java JSON API

*Az `org.json.simple 1.1.1.jar` csomag a `JSON API` fontos osztályait *tartalmazza*.*

- `JSONValue`
- `JSONObject`
- `JSONArray`
- `JSONString`
- `JSONNumber`

URL: <https://stleary.github.io/JSON-java/index.html>

# Java JSON API - JSONObject

*JSONObject* kulcs- és értékpárok rendezetlen gyűjteménye.

*Támogatott főbb metódusok:*

- *get(String key)* – megkapja a kulccsal társított objektumot,
- *opt(String key)* – megkapja a kulccsal társított objektumot, különben *null*,
- *put(String key, Object value)* – beszúr vagy lecserél egy kulcs-értékpárt az aktuális *JSONObject*-ben.

# Java JSON API – JSONObject - példa

A *put()* metódus argumentumában megadhatjuk a *kulcsot* és az *értéket*:

```
JSONObject bl = new JSONObject();
```

```
// put() metódust meghívása
```

```
bl.put("name", „BB”);
```

```
bl.put(„kor”, "22”);
```

```
bl.put(„város”, „Miskolc”);
```

```
System.out.println(bl);
```

# Java JSON API – JSONObject Map osztály

Létrehozunk egy *MAP*, majd argumentumként átadhatjuk a *JSONObject* konstruktorának – példa *(eredmény un.)*:

Ehhez be kell importálni:

```
import java.util.Map;  
Map<String, String> map = new HashMap<>();  
map.put("name", „BB”);  
map.put(„kor”, "22");  
map.put(„város”, „Miskolc”);  
JSONObject bl = new JSONObject(map);
```

# Java JSON API – JSONArray

*JSONArray az értékek rendezett gyűjteménye.*

Az értékek lehetnek:

- *szám, karakterlánc,*
- *logikai érték,*
- *JSONArray, JSONObject, JSONObject.NULL* etc.

*JSONArray van egy konstruktora, amely fogad egy karakterláncot, és elemzi azt JSONArray létrehozásához.*

# Java JSON API – JSONArray

*JSONArray osztály metódusai:*

- *get(int index)* – a megadott index értéket adja vissza (0 és teljes hossz – 1 között),
- *opt(int index)* – az indexhez tartozó értéket adja vissza (0 és teljes hossz – 1 között).
- *put(Object value)* – objektumérték hozzáfűzése a *JSONArray*-hez.

# Java JSON API – JSONArray - példa

Létrehozunk egy *JSONArray* objektumot, majd *hozzáadunk* és *lekérhetünk* elemeket a *put()* és *get()* metódusokkal:

```
JSONArray kk = new JSONArray() ;
```

```
kk.put(Boolean.TRUE) ;
```

```
kk.put („szöveg”) ;
```

```
JSONObject k1 = new JSONObject() ;
```

```
k1.put ("name", „BB”) ;
```

```
k1.put („kor", "22”) ;
```

```
k1.put („varos", „miskolc”) ;
```

```
kk.put(k1) ;
```

# Java JSON – JSON kódolás - mintapélda

*Írjon egy Java programot, mely a JSON objektumot JSON string-be kódolja a Java-ba – JSON mappába.*



# Java JSON tömbkódolás a List használatával

Írjon egy Java programot, mely *JSON tömböt* kódolja a *Java List* segítségével – JSON tömbbe.

URL: <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

# JSON string dekódolása a Java-ban

Írjon egy Java programot, mely *JSON string dekódolását* végzi Java-ban.

# JSON tömb dekódolása Java-ban

*Példa:* a **JSONObject**-et és a **JSONArray**-t használja, ahol a

- `JSONObject` egy `java.util.Map`,
- a `JSONArray`, pedig egy `java.util.List`,

így a *Map* vagy *List* szabványos műveleteivel érheti el őket.

Adott a következő karakterlánc:

```
String s = "[0,{\"1\":{\"2\":{\"3\":{\"4\":[5,{\"6\":7}]}}}}],„
```

Írattassa ki a tömb 2. elemét ill. a mező 1. elemét.

# JSON tömb dekódolása Java-ban

Írjon egy Java programot, mely JSON tömb dekódolja a *Java* segítségével, ahol `JsonObject` (`java.util.Map`) és `JSONArray` (`java.util.List`)

Írattassa ki

- a tömb 2. elemét ,
- a mező 1. elemét.
- {},
- [5]
- [5,2]

# JSON file olvasása Java-ban

Készítsen egy Java programot, amely egy *JSON dokumentumot olvas be* és a feldolgozás után megjeleníti a konzolon.

Használja a `JSONParser`, a `JSONObject` és a `JSONArray` osztályokat.

Hasznos infók: <https://www.tutorialspoint.com/how-can-we-read-a-json-file-in-java>

# JSON file írása Java-ban

Készítsen egy Java programot, amely egy JSON dokumentumot *ír ki egy fájlba és a konzolra*.

Használja a `JSONObject` és a `JSONArray` osztályokat.

URL: <https://howtodoinjava.com/java/library/json-simple-read-write-json-examples/>

# JSON file írása Java-ban

Tervezés menete:

- *Első példány*
- *Második példány*
- *Példányok hozzáadása a listához*
- *Write JSON file*

# JSON használata JavaScriptben

A JSON a *JavaScript objektumok* formátuma.

- A `parse()` metódus használható a *JSON* szöveg *JSON objektumra* történő konvertálására.
- A `toJSONString()` metódus használható a *JSON objektum JSON szövegre* történő konvertálására.

```
var myJSONObject = {"bindings": [  
    {"ircEvent": "PRIVMSG", "method": "newURI", "regex": "^http://.*"},  
    {"ircEvent": "PRIVMSG", "method": "deleteURI", "regex": "^delete.*"},  
    {"ircEvent": "PRIVMSG", "method": "randomURI", "regex": "^random.*"}  
    ]  
};
```

```
var myObject = eval('(' + myJSONtext + ')');
```



# JSON – JavaScript dokumentum

Adott a következő XML dokumentum! *xmlNeptunkod.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <element>
    <author>Nyékyné Dr. Gaizler Judit</author>
    <language>Java 2 I-II. - Útikalauz programozóknak 5.0</language>
  </element>
  <element>
    <author>Reiter István</author>
    <language>C# programozás lépésről lépésre</language>
  </element>
</book>
```

# JSON - JavaScript

Konvertálja át a *xmlNeptunkod.xml* dokumentumot  
xmlNeptunkod.json dokumentumra.

---

```
{
  "book": [
    {
      "language": "Java XML and JSON",
      "author": "Nyékyné Dr. Gaizler Judit"
    },
    {
      "language": "C# programozás lépésről lépésre",
      "author": "Reiter István"
    }
  ]
}
```

# JSON parse - Text JS objektummá konvertálás

`JSON.parse()` használjuk – *JSON Text-be* megírt adatok *JS objektummá* kerülnek elemzésre.

Írjon egy JS programot, amely, egy adott TXT-t konvertálja át *JS objektummá*:

```
' { "name": "BB", "code": 3515, "city": "Miskolc-  
Egyetemváros" } '
```

JSON parse - JSON tömbbe megírt adatok JS tömbbe kerülnek elemzésre

Írjon egy JS kódot, amely a JSON tömböt, JS tömbbé konvertálja.

Adott a következő JSON tömb. Írattassa ki a **tömb 3. elemét!**

```
' [ "Toyota", "Trabant", "WW", "KIA" ] ,
```

Eredmény:

**JSON tömbbe megírt adatok JS tömbbe kerülnek elemzésre**

KIA

# JSON parse - Dátumok elemzése

*A dátum objektumok nem engedélyezettek a JSON-ban.*

Ha dátumot kell megadnia, írja be *karakterláncként*, később konvertálhatjuk a **dátumot objektummá**.

Írjon egy JS kódot, amely egy adott *karakterláncot konvertál dátummá*.

*Adott a következő karakterlánc*

```
'{"name":"BB", "birth":"2000-11-10",  
"code":3515, "city":"Miskolc-Egyetemváros"}';
```

# JSON parse - Dátumok elemzése

Írjon egy kis programot, amely az adott karakterláncot konvertálja dátum objektummá.

*Eredmény:*

# JSON parse - Függvények elemzése

*A függvények nem engedélyezettek a JSON-ban.*

Ha függvényt kell beillesztenie, írja be karakterláncként, majd később vissza konvertáljuk dátummá.

Írjuk be a code helyére: `"code": "function() {return 3515;}"`

*Karakterlánc konvertálása függvénnnyé*

```
'{"name": "BB", "birth": "2000-11-10",  
"code": 3515}', "city": "Miskolc-Egyetemváros"}';
```

# JSON parse - Függvények elemzése

Írjon egy kis programot, amely *code* nevű mező értékét egy függvény segítségével adja vissza.

Eredmény:



# JSON `stringify()` használata

A JSON elterjedt használata az adatok cseréje  
webszerverrel/webszerverről.

*Amikor adatokat küldünk egy webszervernek, az adatoknak  
karakterláncnak kell lenniük.*

*Alakítson át egy JavaScript-objektumot - sztringgé a  
JSON.stringify() segítségével.*

*JS object konvertálása JSON karakterláncná - JSON.stringify().*

# JSON objektum konvertálása JSON karakterlánccá

Adott a következő JSON objektum:

```
{name:"BB", code:3515, city:"Miskolc-  
Egyetemváros"};
```

Írjon egy kis programot, amely a *JSobjektum*-ot konvertálja JSON string-é – használja a *JSON.stringify()*

# Felhasznált irodalom

- Kovács László: Adatkezelés XML környezetbe  
<http://moodle.iit.uni-miskolc.hu/login/index.php>
- JSON Tutorial  
<https://www.javatpoint.com/java-json-example>
- JSON Basics Tutorial  
<https://www.tutorialspoint.com/json/index.htm>