Adatszerkezetek modellezése

Bonyolultabb adatszerkezet modellezésére összetett típust használunk, ami lehet tömb vagy az objektum, vagy a kettő együttes használata.

- Adott értéksorozat: *statikus tömb* v. *vektor* → *tömbliterál*
- Különböző típusú értékeket tartalmazó szerkezet: *record* v. *struct* → *objektummal*
- *Rekordok tömbje* → *objektumok tömbje v. tömböket tartalmazó objektumok tömbje*

Az adatszerkezetek fenti leírási módjának, amely során JavaScript objektum- és tömbliterállal adjuk meg az adatainkat, külön nevet is adtak, és ***JavaScript Object Notation*** vagy röviden ***JSON*** néven hivatkoznak rá.

A JSON formátum ma már szabványosnak számít, és erősebb formai megkötése van, mint amit a JavaScript ezeknél a literáloknál megenged. (lásd később)

Pl.: Hallgatói adatok tárolása

```
var hallgato = {
    nev: 'Mosolygó Napsugár',
    neptun: 'kod123',
    szak: 'Informatika BSc'
};
```

Részletesebben később!!!

```
var hallgatok = [
    {
        nev: 'Mosolygó Napsugár',
        neptun: 'kod123',
        szak: 'Informatika BSc',
        targyak: [
            'Programozás',
            'Webfejlesztés 2.',
            'Számítógépes alapismeretek'
        ]
    },
    {
        nev: 'Kék Ibolya',
        neptun: 'kod456',
        szak: 'Informatika BSc',
        targyak: [
            'Programozás',
            'Webfejlesztés 2.',
            'Diszkrét matematika',
            'Testnevelés'
        ]
    }
];
```

Függvények (Eljárások)

- a függvényeket a ***function*** kulcsszóval adjuk meg, az utasításokat utasításblokkba helyezni
- a visszatérési értéket a *return* utasítás után adható meg, de nem kötelező megadni
- A függvény *hívása* a függvény nevével történik megadva utána zárójelben a paraméterek aktuális értékét. Híváskor a zárójel akkor is kötelező, ha nincsenek aktuális paraméterek.
- nem kell megadni a paraméterek és a visszatérési érték típusát sem
- A paraméterek és a visszatérési érték is tetszőleges elemi vagy összetett érték lehet, hiányuk esetén az értékük *undefined* lesz, több paraméter is megadható mint amennyi a deklarációban szerepel
- A paramétereket az *arguments* tömb segítségével érhetjük el
- A függvények egymásba ágyazhatók

```
//Függvény általános formája
function fvnev(par1, par2) {
    utasítások;
    return visszatérési érték;
}

//Például
function negyzet(x) {
    return x * x;
}
negyzet(3);     // => 9
```

Függvények (Eljárások)

- van **rekurzív** *hívás*, azaz a függvény meghívhatja saját magát is

```
function factorialis(n) {
    if ( n==0 || n==1 ) return 1;
    else {
        var eredmeny = ( n * factorialis(n-1) ); return eredmeny;
    }
}

document.getElementById("objektum").innerHTML += factorialis(6)+ "<br>" +"<hr>";
```

720

- **Hivatkozás és meghívás**
  - Egy függvény meghívása úgy történik, hogy a neve után gömbölyű zárójelben megadunk valahány paramétert (akár egyet sem).
  - Ha nincsen zárójel a függvény neve után, akkor csak hivatkozunk a függvényre, azaz függvényreferenciát használunk.

  Pl:. *Az alábbi példában a duplaz változó megkapja a ketszerez függvény referenciáját(amire a ketszerez mutatott) Így a duplaz-t meghívva a ketszerez() függvény fut le.*

```
//Függvénydeklaráció
function ketszerez(a) {
    return a * 2;
}
//Függvényreferencia átadása
var duplaz = ketszerez;
//Függvény meghívása
duplaz(21); //42
```

Függvények (Eljárások)

- **A függvényliterál – névtelen függvény**
  - a függvény önmagában egy kifejezés, ami más kifejezésekben is megjelenhet
  - Az adattípusok azon formáját, amellyel kifejezésekben megjelenhet neveztük literálformának
  - a függvényekhez tartozó literálformát nevezzük *függvényliterálnak* nevezzük

```
function (par1, par2) {
    //JavaScript kód
}
```

- **Függvénykifejezés**
  - Ha a függvényliterált értékadásban használjuk, akkor a függvénydeklaráció mellett megjelenik a függvények egy másik létrehozási formája is, amely függvénykifejezést használ.

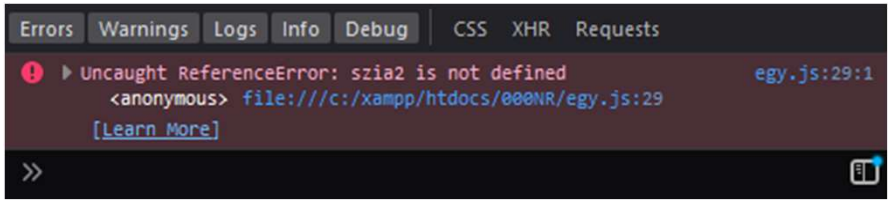
```
//Függvénydeklaráció
function osszead(a, b) {
    return a + b;
}

//Függvénykifejezéssel
var osszead = function (a, b) {
    return a + b;
};

//Hívásuk
osszead(10, 32);    //42
```

Függvények (Eljárások)

- **Önkioldó függvény**
  - A függvényhivatkozás helyettesíthető a függvényliterállal, azaz a függvény helyben definiálásával. Ekkor a definiált függvényt rögtön meg is hívjuk, a szakirodalom az ilyen függvényeket *önkioldó függvényeknek* nevezi..

```
//Függvény definiálása
var szoroz = function (a, b) {
    return a * b;
}

//Függvény meghívása
szoroz(6, 7);   //42
//vagy hangsúlyozandó, hogy itt függvényreferencia van
(szoroz)(6, 7); //42

//Függvényhivatkozást függvényliterállal helyettesítve
(function (a, b) {
    return a * b;
})(6, 7);       //42
```

```
szia();

function szia(){
    alert('Szia !')
}

szia2();

szia2 = function(){
    alert ('Szia2 !')
}
```



- **Megjegyzés:**
  - *A függvénydeklaráció és a függvénykifejezés közötti döntéskor egy dologra figyelni kell, meghívhatunk egy függvényt, mielőtt függvényt deklarálnánk. A függvénykifejezéssel létrehozott függvény nem hívható meg a létrehozása előtt.*

  - A JavaScript emelés (Hoisting) arra a folyamatra utal, amelynek során az értelmező a kód végrehajtása előtt áthelyezi a függvények, változók vagy osztályok deklarációját a hatókörük tetejére. Az emelés lehetővé teszi a funkciók biztonságos kódbeli használatát, mielőtt deklarálnák őket.

Függvények - Paraméterátadás

A függvény tipikusan a paraméterein keresztül kapja meg a bemeneti adatokat, és a visszatérési értékén keresztül adja meg a feldolgozás eredményét

- Paraméterátadás:
  az egyszerű típusú értékek (number, boolean) - érték szerint,
  az összetett típusú adatok (objektum, tömb, függvények) - referencia szerint adódnak át.
- Példa: *A függvényre mint paraméter*:
  - *a függvény paraméterre a formális paraméterlistán megadott néven hivatkozunk*
  - *Tetszőleges számú függvényparaméter adható meg*
  - *Az aktuális paraméter függvény és a használt formális paraméter függvény paramétereinek számának egyeznie kell!!! Mivel nincs a paraméterszám egyeztetés , és a meg nem adottak paraméterek undefined értékek lesznek*

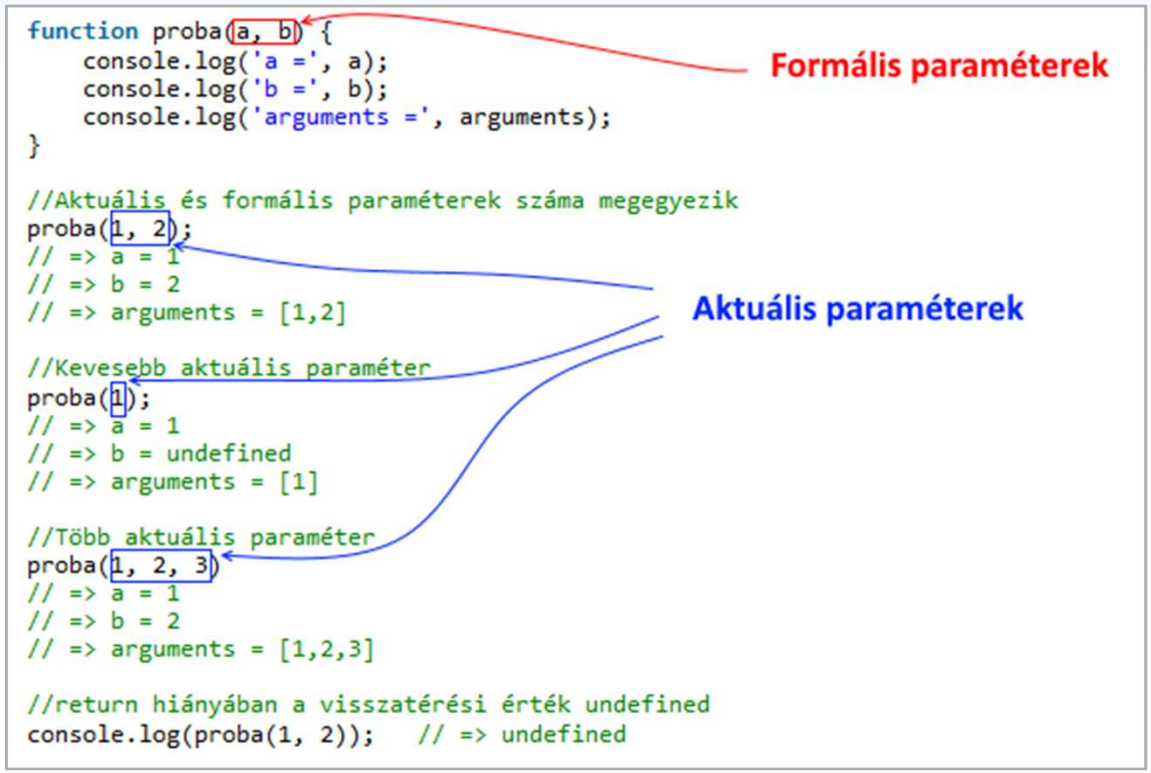
```
function cserel_valtozo (f, x, y){
    return f(y,x);
}

function kisebb(a, b){
    return a<b;
}

BkisebbA = cserel_valtozo(kisebb, 20, 10); // ekkor azt kapjuk vissza, hogy a "b" értéke kisebb-e "a"-nál
document.getElementById("objektum").innerHTML +=  BkisebbA + "<br>";
```

Formális paraméterek

Aktuális paraméterek

Függvények - Paraméterátadás

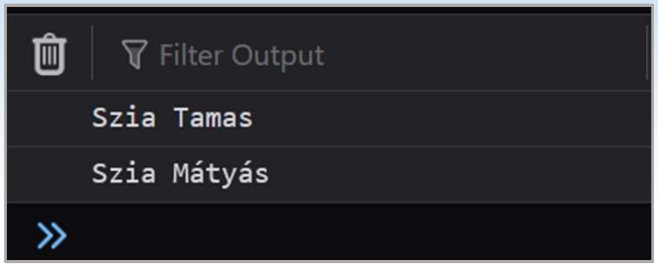
- **Formális és aktuális paraméterek viszonya**
  - Az aktuális és formális paraméterek száma nem kell, hogy megegyezzen.
  - Ha kevesebb aktuális paramétert adunk meg, akkor az értéket nem kapó formális paraméterek értéke *undefined* lesz.
  - Ha több aktuális paramétert adtunk meg, mint ahány formálisat soroltunk fel, akkor a függvényen belül egy *arguments* nevű tömbszerű objektumon keresztül lehet őket elérni. Az aktuális paraméterek számát ennek *length* tulajdonságával lehet lekérdezni.
  - Visszatérési érték hiányában a függvény *undefined-dal* tér vissza.



Függvények - Paraméterátadás

- Alapértelmezett paraméterek
  - Az alapértelmezett paraméterek az ES6 specifikáció óta használhatók. Így abban az esetben, ha az argumentumhoz nincs megadva érték, a rendszer az alapértelmezett értéket fogja használni.

```
szia3 = function(nev = "Tamas"){
    console.log(`Szia ${nev}`);
}

szia3();
szia3("Mátyás");
```

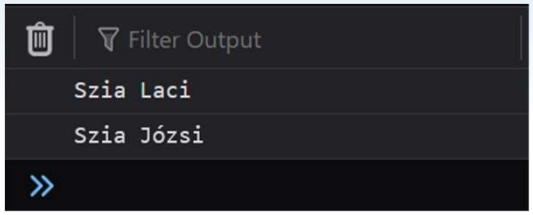


- **Megjegyzés:**
  - ***Sablon karakterláncok :*** *A sablon karakterláncok alternatívát kínálnak a karakterlánc-összefűzésre. Azt is lehetővé teszik, hogy változókat szúrjunk be egy karakterláncba.*
  - *Ezeket szokás sablonkarakterláncnak, sablonliterálnak vagy karakterláncsablonnak nevezni.*
  - A karakterláncot akkor a tompa ékezet → ` (AltGr + 7) határolja.
  - A sablonkarakterláncok tiszteletben tartják a whitespace karaktereket, megkönnyítve az e-mail sablonok, kódpéldák vagy bármi más szóközt tartalmazó szöveg létrehozását.

Nyíl Függvények (Arrow functions)

- A nyíl függvények lehetővé teszik, hogy rövidebb (function és return kulcsszavak nélküli) függvény szintaxist írjunk.
- Használatuk korlátozott, nem használható minden helyzetben.
  - Nem használhatók this és super kulcsaszvakkal, és nem használható metódusként.
  - Nemhasználható new.target kulcsszóval sem.
  - Nem alkalmas call, apply, és bind kulcsszavakkal, amelyek általában hatókör megállapításán alapulnak.
  - Nem használható konstruktorként.

```
szia3 = function(nev){
    console.log(`Szia ${nev}`);
}
szia4 = nev => console.log(`Szia ${nev}`);

szia3("Laci");
szia4("Józsi");
```



Használat:

```
/*
 * Egyszerű szintaxis
 */
 param => expression;              //egy paraméter + mincs visszatérési érték
 (param1, paramN) => expression;   //több paraméter + nincs visszatérési érték
 (a=400, b=20, c) => expression;   // paraméter alapértelmezett értékekkel + nincs visszatérési érték

 param => {                        //egy paraméter + több utasítás + visszatérési érték
   let a = 1;
   return a + param;
 };

 (param1, paramN) => {             //több paraméter + több utasítás + visszatérési érték
   let a = 1;
   return a + param1 + paramN;
 };
```

Globális függvények

A JavaScript számos globális elérhető függvényt definiál. a böngésző által nyújtott függvények segítségével,

- Not a Number = NaN → ***isNaN()***
  - Ha egy matematikai művelet eredménye kivezet a számok halmazából, akkor a JavaScript egy speciális számot ad eredményül, ami igazából „nem-szám”, a NaN (Not a Number) értéket. A NaN toxikus hatású, azaz matematikai kifejezésben megjelenve, az is NaN-t ad eredményül. Egyedüli biztos vizsgálata az isNaN() függvénnyel lehetséges.

```
var sokPenz = 100 * 'kevés pénz';
sokPenz;         // => NaN
sokPenz / 10;    // => NaN
isNaN(sokPenz); // => true
```

- Végtelen =Infinity → **isFinite()**
  - Ha az ábrázolható értéktartományon kívülre vezet egy matematikai műveletek, akkor két speciális számot kaphatunk, a pozitív végtelent (Infinity vagy +Infinity) vagy a negatív végtelent (-Infinity). Vizsgálata az isFinite() függvénnyel lehetséges.

```
var a = 10/0;
a;              //Infinity
isFinite(a);    //false
```

Globális függvények

- Konverziók → **parseInt (*szöveg, szr alapszáma*) és a parseFloat(*szöveg*)**
  - Szöveges érték explicit átalakítása számmá .
  - Mindkét függvény első paramétereként az átalakítandó szöveget kell megadni.
  - A parseInt()-nek második paraméterként azt is meg lehet adni, hogy a szöveget milyen számrendszerű számnak értelmezze
  - Mindkét függvény addig próbálja értelmezni a szöveget, amíg a formátumának megfelelő számokat talál az elején. Ha a szöveg egyáltalán nem alakítható át, akkor *NaN* lesz az eredmény.

```
//parseInt
parseInt('123');     //123
parseInt('123', 10); //123
parseInt('0101', 2); //5
parseInt('alma', 10); //NaN
parseInt('5alma', 10); //5

//parseFloat
parseFloat('4.54'); //4.54
parseFloat('3.1415 a pi'); //3.1415
```

Globális függvények

- Szövegkódolás - **encodeURI***[Component]***() és decodeURI***[Component]***()**
  - Webes alkalmazásokban az információ gyakran kerül tárolásra vagy továbbításra szöveges formátumban. Annak érdekében, hogy a speciális karakterek se okozzanak gondot, megfelelően kódolni szükséges őket.
  - Az *encodeURI()* és *decodeURI()* függvények teljes URI-k kódolására és dekódolására szolgálnak, feltételezvén, hogy bizonyos karakterek (/, :, ;, ?, #, &) az URI részei, ezért ezeket kihagyják a kódolási folyamatból.
  - Az *encodeURIComponent()* és *decodeURIComponent()* függvények az URI egyes részeinek kódolására és dekódolására való valók, ezeknél a fent említett speciális karaktereknek nincs speciális funkciója, azok a szöveg részét képezik.

Megjegyzés: Szövegek oda-visszakódolásánál találkozhatunk még az *escape()* és *unescape()* függvényekkel is. Ezeket azonban a szabvány már nem tartalmazza, elavultak, és használatuk nem javasolt.

```
//Szöveg kódolása és dekódolása
var kod = encodeURIComponent('árvíztűrőtükörfúrógép');
kod;    //"%C3%A1rv%C3%ADzt%C5%B1r%C5%91t%C3%BCk%C3%B6rf%C3%BAr%C3%B3g%C3%A9p"
var dekod = decodeURIComponent(kod);
dekod;  //"árvíztűrőtükörfúrógép"

//URI kódolása
encodeURI('http://példa.hu/index.php?sz=Bogyó és Babóca');
//"http://p%C3%A9lda.hu/index.php?sz=Bogy%C3%B3%20%C3%A9s%20Bab%C3%B3ca"

//URL küldése paraméterként
var url = encodeURIComponent('http://példa.hu/index.php?sz=Bogyó és Babóca');
//"http%3A%2F%2Fp%C3%A9lda.hu%2Findex.php%3Fsz%3DBogy%C3%B3%20%C3%A9s%20Bab%C3%B3ca"
encodeURI('http://valami.hu/index.php?hova=') + url;
//"http://valami.hu/index.php?hova=http%3A%2F%2Fp%C3%A9lda.hu%2Findex.php%3Fsz%3DBogy%C3%B3%20%C3%A9s%20Bab%C3%B3ca"
```

Beolvasás - Kiírás

A JavaScript nyelv önmagában nem tartalmaz beolvasó és kiíró utasításokat. Ezeket mindig az a környezet biztosítja, amiben a JavaScript értelmezésre kerül. Beolvasásra, kiírásra JavaScriptben három lehetőség van:

- a böngésző által nyújtott függvények segítségével,
- a böngészőbe beépülő konzol segítségével, vagy
- a böngészőbe betöltődő dokumentumon keresztül.

A böngésző beolvasó-kiíró függvényei

A böngészők három beépített függvényt biztosítanak a felhasználóval való kapcsolattartásra:

- Az **alert()** függvény egy felugró ablakban írja ki a paraméterként megkapott értéket. Viszonylag ritkán használjuk
- A **confirm()** függvény egy olyan ablakot dob fel, amelyen egy „OK” és „Mégsem” gomb helyezkedik el a paraméterként megadott szöveg alatt. A függvény igaz értékkel tér vissza az előbbi, hamis értékkel az utóbbi megnyomása esetén. Gyakran használjuk azokban az esetekben, amikor egy művelet megerősítésre vár.
- A **prompt()** függvény egy szöveges beviteli mezőt tartalmazó ablakot jelenít meg. Első paramétere az ablakon megjelenő szöveg, az opcionális második pedig a beviteli mező alapértelmezett értékét adja meg. A függvény a beviteli mezőbe írt szöveggel tér vissza. Nagyon ritkán használjuk.

Beolvasás - Kiírás

A konzol lehetőségei

A JavaScript konzolokra csak kiírni lehet. Kiíró utasításból többet is tartalmazhat egy konzol attól függően, hogy milyen típusú üzenetet írunk ki, szeretnénk-e a kiírásokat csoportosítani, vagy időt mérni. Ezek az utasítások konzolonként változhatnak. A console.log() parancs azonban mindenhol elérhető. Felhasználóval történő kommunikációra nem használjuk, csak fejlesztéskor.

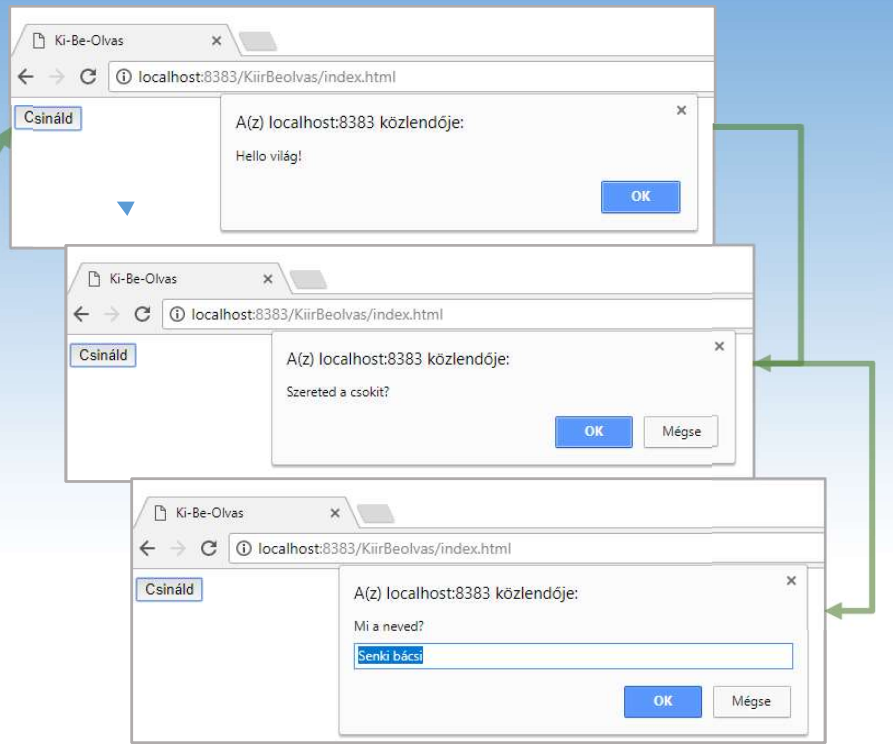
Kommunikáció a dokumentumon keresztül

Mivel a felhasználó a böngészőbe weboldalakat tölt be, ezért az egyértelműen legfelhasználóbarátabb kommunikáció a betöltött dokumentumon keresztül zajlik. Beolvasásra a HTML űrlapelemei szolgálnak, kiírásra pedig a dokumentum adott elemei tartalmának módosítása. ← ***Lásd később részletesebben és az eddigi példákban!!!***

A régi viszonylag elavultnak mondható lehetőség, amely csak és kizárólag az oldal betöltése közben használható. A document.write() vagy document.writeln() paranccsal JavaScriptből generálhatunk HTML tartalmat az oldalba. Ez a megoldás csupán kiírásra szolgál.

Beolvasás - Kiírás

```
function  doit(){
   alert('Hello világ!');

   var mehet = confirm('Szereted a csokit?');
   console.log(mehet);

   var nev = prompt('Mi a neved?', 'Senki bácsi');
   console.log(nev);
```




```
<html>
    <head ...7 lines />
    <body>
        <div id="lap"></div>
        <input type="submit" value="Csináld" name="doit"  onclick="doit()"/>

        <h1>Celsiusból Fahrenheit</h1>
        <script type="text/javascript">
            var cels = prompt('Celsius fok:', 0)
            var fahr = cels * 9 / 5 + 32;
            document.writeln('<p>Celsius = ' + cels + '</p>');
            document.writeln('<p>Fahrenheit = ' + fahr + '</p>');
        </script>

    </body>
</html>
```