JavaScript gyakorlás - megoldások

Változók definiálása

Const vs let kulcsszavak

Milyen kulcsszavakat (`const` vagy `let`) írnál a kérdőjelek helyére? Miért?
Megoldás:

```
const f = (scores) => {
 const min = 30;
 let sum = 0;

 scores.forEach((score) => (sum += score));

 if (sum < min) {
   console.log('Failed');
 }

 return sum;
};

f([5, 3, 2]);
f([7, 12, 20]);
```

Magyarázat: A `const` kulcsszóval létrehozott változóknak pontosan egyszer adhatunk értéket, tehát akkor használjuk, amikor egy változónak már létrehozáskor tudjuk az értékét, és később nem akarjuk módosítani. A `let` kulcsszó megengedőbb, az így létrehozott változóknak akárhányszor adhatunk értéket. Ha a helyzet megengedi, akkor mindig használjuk a `const` kulcsszót, mert ezáltal a kódunk olvashatóbbá válik.

Érték típusok

Sztringek - template string szintakszis

```
const firstName = 'John';
const age = 25;
console.log(
 `Hello, my name is ${firstName} and I'm ${age} ${
   age === 1 ? 'year' : 'years'
 } old`
);
```

Megoldás: `Hello, my name is John and I'm 25 years old`
Magyarázat: Template string szintakszis esetén a stringeket ún. backtick-ek (`) között definiáljuk. Gyakran előfordul, hogy a stringbe egy kifejezés értékét szeretnénk

behelyettesíteni, template stringek esetén behelyettesítendő kifejezés értékét ${} - közé írhatjuk, ide bármilyen JavaScript kódrészlet kerülhet, pl.: változó értékének felolvasása (1. és 2. behelyettesítés a példában), függvényhívás eredménye vagy egyszerűbb JavaScript kifejezések (3. behelyettesítés).

Sztringek - műveletek

Írjunk egy függvényt, amely két sztringet vár paraméterül, a függvény fűzze össze a két sztringet és térjen vissza az eredménnyel. Hívjuk meg ezt a függvényt, majd írjuk ki a visszatérési értéket a console-ra, írjuk még ki az eredmény sztring hosszát, valamit az 5. karaktert. Mit kapunk, ha az összefűzött sztring nincs 5 karakter hosszú?

Megoldás és magyarázat:

```
const a = 'alma';
const b = 'fa';

const concat = (s1, s2) => s1 + s2; // Stringeket a + kulcsszóval fűzhetünk össze

const result = concat(a, b);
console.log(result); // almafa
console.log(result.length); // 6
console.log(result.charAt(4)); // f -> ne felejtsük, 0-tól kezdődik a számozás, ha 5 karakternél rövidebb lenne a sztring, akkor egy üres sztringet kapnánk
```

Ha még kíváncsi vagy, hogy milyen műveleteket lehet egy string típusú értékkel végezni, akkor itt találod ezek listáját:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Függvények - arrow function szintakszis

Írjuk át egyszerűbb szintaktikai formára az alábbi formára, milyen karaktereket hagyhatunk el?

```
const f = (a) => {
 return a + 4;
}
```

Megoldás:

```
const f = a => a + 4;
```

Magyarázat:
Arrow function függvény szintakszis esetén az alábbi egyszerűsítések állnak rendelkezésre:

- Ha a függvénynek csak egy paramétere van, akkor a paraméter körüli zárójelek elhagyhatók
- Ha a függvényünk csak egyetlen utasításból áll, akkor a függvény törzs körüli kapcsos zárójelek elhagyhatók, ilyenkor az egyetlen utasítás végére nem is kell pontosvesszőt tennünk (a példában a pontosvessző nem a függvény törzsében levő utasítás végét jelzik, hanem a változó definíció végét zárja le)

- Ha az előbb egyszerűsítést alkalmazzuk, akkor a függvény visszatérési értéke automatikusan az egyetlen utasítás visszatérési értéke lesz

Tömbök - műveletek

Hozzunk létre egy tömböt, amelyben olyan objektumok találhatók, amelyek kulcsai: “firstName”, “lastName”, “age”. Létrehozáskor egy elemet tegyünk a tömbbe, majd a “push” metódus segítségével még két elemet adjunk a tömbhöz. Írjuk ki a tömb hosszát, és a legelső elemet a console-ra. A “splice” metódus segítségével töröljük a 2. elemet a tömbből.

Megoldás és magyarázat:

```
const array = [{ firstName: 'John', lastName: 'Doe', age: 7 }];

array.push({ firstName: 'Sara', lastName: 'Smith', age: 22 });
array.push({ firstName: 'Adam', lastName: 'Rush', age: 42 });

console.log(array.length); // 3
console.log(array[0]); // Ne felejtsük, 0-tól számozzuk az elemeket

array.splice(2, 1); // Első paraméter: törlendő elem index-e, második paraméter: hány
elemet akarunk törölni

console.log(array.length); // 2
```

Ha többet szeretnénk tudni a tömbökön végezhető műveletekről, akkor itt található a teljes lista:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Primitív vs referencia típusok

Osztályozzuk az alábbi érték típusokat aszerint, hogy primitív vagy referencia típusnak számítanak-e:

- string
- Object
- Function
- Array
- bool
- Date
- number

Megoldás:

- Primitív
  - string
  - number
  - bool
- Referencia
  - Object
  - Function
  - Array
  - Date

Összefoglaló a különféle értéktípusokról JavaScriptben:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#javascript_types

Ciklusok és elágazások

Tömbökön iterálás

Keressünk három féle módot, amivel egy tömbön végigiterálhatunk.

Megoldás és magyarázat:

```
const numbers = [4, 5, 12, 3, 77, 8];

numbers.forEach((n) => console.log(n)); // forEach, ez a legújabb az iterációk közül, ez a legegyszerűbb, ha lehet ezt válasszuk

for (let i = 0; i < numbers.length; i++) {
 console.log(numbers[i]);
} // for ciklus, tömb iterációknál nem szoktuk használni, inkább akkor, ha x-szer kell egymás után végrehajtani valamit

let i = 0;
while (i < numbers.length) {
 console.log(numbers[i]);
 i++;
} // while ciklus, akkor érdemes használni, ha nem csak annyi a kilépési feltétel, hogy végégértünk a tömbön, hanem pl.: megtaláltuk a keresett elemet (a gyakorlatban az újabb tömbfüggvények mellett ritkán van rá szükség)
```

If-else

Írjuk át az alábbi kifejezést if-else helyett ?: operátorra:

```
const f = (y) => {
 let res;
 if (y >= 0) {
   res = 2 * y;
 } else {
   res = -2 * y;
 }

 return res;
};

console.log(f(3));
console.log(f(-5));
```

Megoldás:

```
const f = (y) => {
 const res = y >= 0 ? 2 * y : -2 * y;
 return res;
};

console.log(f(3));
console.log(f(-5));
```

Truthy vagy falsy

Mit ír ki?

Megoldás:

```
const truthyOrFalsy = (n) => {
 if (n) {
   console.log('truthy');
 } else {
   console.log('falsy');
 }
}

truthyOrFalsy(''); //falsy
truthyOrFalsy([]); // truthy
truthyOrFalsy(0); // falsy
```

Magyarázat:
JavaScriptben minden érték leképezhető true vagy false értékekre, ezek az értékek akkor kapnak jelentőséget, ha az értéket “bool contextusban” használjuk (pl.: if-else ágban), a true-ra értékelődő értékeket truthy értékeknek hívjuk, false-ra értékelődőket pedig falsy-nak nevezzük.

- Truthy értékek listája: https://developer.mozilla.org/en-US/docs/Glossary/Truthy
- Falsy értékek listája: https://developer.mozilla.org/en-US/docs/Glossary/Falsy

Logikai operátorok

! operátor - mit ír ki, miért?

```
console.log(!!5);
```

Megoldás: `true`
Magyarázat:

A “!” logikai operátor negálja a mögötte található bool értéket, mivel ez “bool contextust” igényel, ezért első lépésben bool értékké kell konvertálni azt az értéket, amire az operátort alkalmazzuk, jelen esetben az 5-öt, az 5 truthy érték, a bool konverzió után true-t kapunk. Erre kell alkalmazni a ! operátort, vagyis negálni az értéket, ami false-t eredményez. Mivel két !!-et írtunk, ezért az előző művelet eredményére ismét alkalmazni kell a negálást, ami pedig true-t fog eredményezni. Tehát a “!!” alkalmazásával egy érték bool leképzését kapjuk meg.