# JS ismétlés

Csak azokat írom ide, amiket plusszban mondtam a 05.18-i órához képest

# Script tag

## Helye

A `<script>` taget elhelyezhetjük az `<head>` vagy a `<body>` elemben, attól függően, mikor szeretnénk, hogy a JavaScript kód betöltődjön és fusson. Általában a `<body>` végére helyezzük, hogy biztosak legyünk benne, hogy az oldal összes eleme betöltődött, mielőtt a JavaScript kód lefut.

## Külső script fájl importálása

A külső JavaScript fájlok használata lehetővé teszi a kód szervezettebb tárolását és újrahasználatát. Ebben az esetben a `<script>` tag src attribútumát használjuk, hogy hivatkozzunk a külső fájlra.

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Külső JavaScript</title>
    </head>
    <body>
        <h1>Üdvözlet!</h1>
        <script src="script.js"></script>
    </body>
</html>
```

Ha van src attribútum beállítva, akkor a script tag belseje ignorálva lesz.

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Külső JavaScript</title>
    </head>
    <body>
        <h1>Üdvözlet!</h1>
        <script src="script.js">
            console.log("Ez nem fut le");
        </script>
    </body>
</html>
```

# strict mode

A "strict mode" (szigorú mód) egy JavaScript futási mód, amely szigorúbb szabályokat vezet be a kód írására és futtatására vonatkozóan. A "strict mode" bevezetése az ECMAScript 5 (ES5) szabványban történt.

A "strict mode"-ot egy speciális kifejezéssel lehet engedélyezni: "use strict";. Ezt a kifejezést vagy a teljes szkript elején, vagy egy függvény elején kell elhelyezni. Ha a teljes szkript elején helyezzük el, akkor az egész szkript szigorú módban fog futni. Ha csak egy függvény elején helyezzük el, akkor csak az a függvény fog szigorú módban futni. (Függvényekről később tanulunk majd)

**Példa:**

```
"use strict";

x = 3.14; // Hiba, mert x nincs deklarálva
```

# Template literals

## 1. Template Literals

A template literals egy újfajta szintaxis, amelyet az ECMAScript 6 (ES6) vezetett be. Lehetővé teszik, hogy egyszerűbben és olvashatóbban írjunk szövegformázott stringeket és interpoláljunk változókat a szövegbe. A template literals használatához backtick (`) karaktereket használunk a szöveg körül, nem pedig a hagyományos idézőjeleket (' vagy ").

## 2. Változó Interpoláció

A template literals egyik legnagyobb előnye, hogy lehetővé teszi a változók és kifejezések beillesztését a szövegbe közvetlenül a `${}` szintaxis használatával. Ebben az esetben a `${age}` kifejezés interpolálja az `age` változó értékét a stringbe.

## 3. Példa Használat

Tegyük fel, hogy van egy változónk, amely az életkort tárolja:

```
let age = 30;
```

Amikor a fenti template literal kódot futtatjuk:

```
let age = 30;
let message = `I'm ${age} years old`;
console.log(message);
```

Ez az alábbi stringet adja eredményül:

```
I'm 30 years old
```

# Boolean konverzió

A Boolean konverzió azt jelenti, hogy egy értéket logikai típusúvá alakítunk, azaz `true` vagy `false` értékké. JavaScriptben ezt a `Boolean` konstruktor vagy a logikai negálás (`!`) kétszeri használatával érhetjük el (erről nem beszéltünk órán).

## Igaz (true) értékek

A következő értékek mindig `true`-vá konvertálódnak:

1. **Nem üres stringek**:
   - Például: `"hello"`, `"false"`, `"0"`, `" "`
2. **Nem nulla számok**:
   - Például: `1`, `-1`, `3.14`, `Infinity`, `-Infinity`
3. **Logikai érték true**:
   - Például: `true`

### Példák:

```
Boolean("hello"); // true
Boolean(123); // true
Boolean(true); // true
```

# Hamis (false) értékek

A következő értékek mindig `false`-vá konvertálódnak:

1. **Üres string**:
   - Például: `""`
2. **Nulla szám**:
   - Például: `0`, `-0`
3. **Null érték**:
   - Például: `null`
4. **Undefined érték**:
   - Például: `undefined`
5. **NaN (Not-a-Number)**:
   - Például: `NaN`
6. **Logikai érték false**:
   - Például: `false`

## Példák:

```
Boolean(""); // false
Boolean(0); // false
Boolean(null); // false
Boolean(undefined); // false
Boolean(NaN); // false
Boolean(false); // false
```

# További Példák és Magyarázatok

1. **Nem üres stringek**:

- Még az olyan stringek is, amelyek látszólag hamis értékek, mint például `"false"` vagy `"0"`, true-ként konvertálódnak, mert nem üresek.

```
Boolean("false"); // true
Boolean("0"); // true
```

2. **Nem nulla számok**:

   - Bármely szám, ami nem nulla, true-ra konvertálódik.

```
Boolean(1); // true
Boolean(-100); // true
Boolean(Infinity); // true
Boolean(-Infinity); // true
```

3. **Logikai negálás használata**:

   - Kétszeri logikai negálással (`!!`) bármely értéket Boolean típusúvá konvertálhatunk.

```
!!"hello"; // true
!!0; // false
!!{}; // true
!!null; // false
```

# ÉS (&&) és VAGY (||) Műveletek

JavaScriptben az ÉS (&&) és a VAGY (||) logikai műveletek lehetővé teszik, hogy több feltételt kombináljunk.

## ÉS Művelet (&&)

Az ÉS művelet akkor ad vissza `true` értéket, ha mindkét operandusa `true`. Ha az egyik operandus `false`, akkor a kifejezés értéke `false` lesz.

**Példák:**

```
true && true; // true
true && false; // false
false && true; // false
false && false; // false

// Példák nem logikai értékekkel:
let a = 10;
let b = 20;

a > 5 && b > 15; // true, mert mindkét kifejezés igaz
a > 15 && b > 15; // false, mert az első kifejezés hamis
```

## VAGY Művelet (||)

A VAGY művelet akkor ad vissza `true` értéket, ha bármelyik operandusa `true`. Ha mindkét operandus `false`, akkor a kifejezés értéke `false` lesz.

**Példák:**

```
true || true; // true
true || false; // true
false || true; // true
false || false; // false

// Példák nem logikai értékekkel:
let x = 10;
let y = 20;

x > 15 || y > 15; // true, mert a második kifejezés igaz
x > 15 || y > 25; // false, mert mindkét kifejezés hamis
```