1. HTML DOM

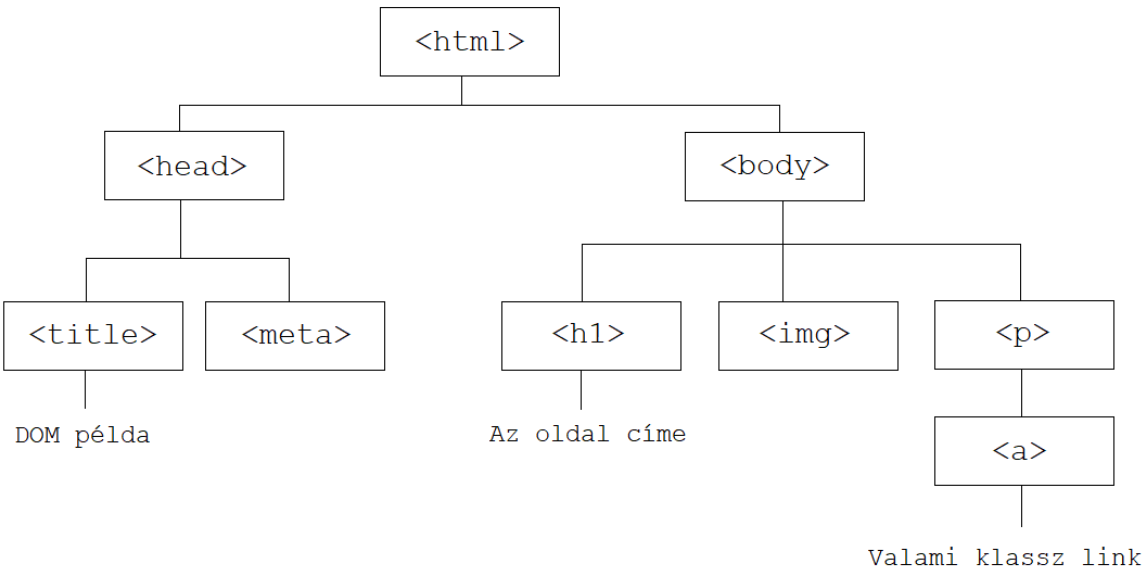
A webfejlesztésben a **HTML** nyelvet használjuk weboldalak létrehozására. Ennek a nyelvnek a segítségével mondhatjuk meg, hogy mi az, amit egy weboldalon látni szeretnénk (pl. szövegek, képek, táblázatok, űrlapok, multimédia stb.), illetve lehetőségünk van a weboldalon megjelenő tartalom strukturálására is különféle szakaszok, tartalmi egységek kialakításával.

A HTML dokumentumok úgy épülnek fel, hogy HTML objektumokat (úgynevezett **tageket**) ágyazunk egymásba. Ezek az objektumok egy hierarchikus fastruktúrát alkotnak a dokumentumban.

Amikor egy weboldal betöltődik, akkor a böngésző a weboldalon található HTML objektumokból elkészíti az úgynevezett **dokumentum-objektum modell**t, avagy röviden a **DOM**-ot. A **DOM fa** (DOM tree) segítségével könnyen szemléltethetjük a weboldalon található HTML elemek hierarchikus viszonyait. Nézzünk példát egy DOM fára!

Példa: Egy egyszerű HTML kód és az ahhoz tartozó DOM fa

```html
<!DOCTYPE html>
<html lang="hu">
   <head>
      <title>DOM példa</title>
      <meta charset="UTF-8"/>
   </head>
   <body>
      <h1>Az oldal címe</h1>
      <img src="my-image.png" alt="Valami klassz kép"/>
      <p>
         <a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">Valami klassz link</a>
      </p>
   </body>
</html>
```



Megjegyzés: A `DOCTYPE` nem egy HTML tag, ezért a DOM fában sem szerepel.

1.1. HTML elemek DOM-beli viszonyai

Ha az A objektum (nem feltétlen közvetlenül) tartalmazza a B objektumot, akkor azt mondjuk, hogy az A objektum a B objektum **ős**e, a B objektum pedig A-nak **leszármazott**ja. Amennyiben ez a tartalmazás **közvetlen**, akkor A-t a B **szülő**jének, B-t pedig az A **gyerek**ének nevezzük.

Néhány példa a fenti kódból és az ahhoz tartozó DOM fából:

- A `<body>` objektum leszármazottjai a `<h1>`, `<img>`, `<p>` és `<a>` objektumok, illetve a „Valami klassz link” és „Az oldal címe” szöveges csomópontok.
- A `<body>` objektum gyerekei a `<h1>`, `<img>` és `<p>` objektumok. Másképp mondva: a `<h1>`, `<img>` és `<p>` objektumok szülője a `<body>`.
- A `<body>` objektumnak az `<a>` objektum **nem** gyereke, csak leszármazottja, hiszen itt a tartalmazás nem közvetlen (van még a fában egy `<p>` elem is közöttük).

Ha az A és B objektumok szülője megegyezik, akkor A és B egymás **testvér**ei. Például a fenti kódban és az ahhoz tartozó DOM fában a `<h1>`, `<img>` és `<p>` objektumok egymás testvérei, hiszen mindhárom objektum szülője a `<body>`.

A DOM fa tetején lévő, szülővel nem rendelkező elemet **gyökérelem**nek nevezzük. A teljes HTML DOM-ban a gyökérelem mindig a `<html>` objektum lesz (ugyanis ebbe ágyazunk be minden további HTML elemet).

1.2. A DOM egyik gyakorlati jelentősége: DOM alapú CSS szelektorok

Amikor a weboldalunk tartalmát CSS-ben formázzuk, akkor használhatunk olyan szelektorokat (kijelölőket) is, amelyek a DOM-beli viszonyaik alapján jelölnek ki HTML objektumokat. Néhány példa DOM alapú CSS szelektorokra, a teljesség igénye nélkül:

- `div p`: kijelöl minden olyan `<p>`-t, amely egy `<div>` leszármazottja
- `div > p`: kijelöl minden olyan `<p>`-t, amely egy `<div>` gyereke
- `div ~ p`: kijelöl minden olyan `<p>`-t, amely egy `<div>` testvére, és ezen `<div>` után szerepel
- `p:first-child`: kijelöl minden olyan `<p>`-t, amely a szülőjének legelső gyereke
- `p:nth-child(n)`: kijelöl minden olyan `<p>`-t, amely a szülőjének `n`-edik gyereke
- `p:last-child`: kijelöl minden olyan `<p>`-t, amely a szülőjének utolsó gyereke
- `p:first-of-type`: kijelöl minden olyan `<p>`-t, amely a `<p>` típusú testvérei közül a legelső
- `p:nth-of-type(n)`: kijelöl minden olyan `<p>`-t, amely a `<p>` típusú testvérei közül az `n`-edik
- `p:last-of-type`: kijelöl minden olyan `<p>`-t, amely a `<p>` típusú testvérei közül az utolsó.

1.3. A DOM egy másik gyakorlati jelentősége: JavaScript DOM manipuláció

A webes világban gyakran előfordul, hogy „menet közben”, dinamikusan szeretnénk a DOM fát manipulálni (pl. szeretnénk egy objektumot módosítani vagy törölni, esetleg egy új objektumot akarunk beszúrni a fába). Erre biztosítanak lehetőséget a JavaScript **DOM manipulációs műveletei**.

Néhány valós életbeli példa arra, amikor a DOM fával szeretnénk műveleteket végezni:

- Van egy HTML-ben és CSS-ben megírt űrlapunk, amit a felhasználó kitölt. A kitöltést követően szeretnénk kliensoldali ellenőrzéseket végezni, hogy helyesek-e a megadott adatok (pl. e-mail cím formátuma megfelelő, a kötelezően bejelölendő jelölőnégyzeteket kipipálták stb.). Ekkor JavaScriptben DOM műveletekkel lekérjük az űrlapon megadott értékeket és feldolgozzuk őket.
- Készítettünk egy egyszerű böngészős számológépet HTML és CSS segítségével, ehhez akarunk működésbeli logikát társítani. Amikor a felhasználó rákattint a „Kiszámol” gombra, akkor JavaScriptben DOM műveletekkel lekérdezzük a beírt kifejezést, kiszámoljuk azt, és az eredményt DOM művelettel kiírjuk a HTML dokumentumba, hogy a felhasználó is láthassa azt.
- Lekérdezünk adatokat egy szervertől, és meg szeretnénk azokat jeleníteni a weboldalunkon. Mivel a HTML oldal már rég betöltődött addigra, amire az adatok megérkeztek, ezért azokat úgy tudjuk megjeleníteni a weboldalon, hogy utólag szúrogatjuk be őket a DOM fába.
- Szeretnénk elérni, hogy egy gombra kattintva a felhasználó válthasson világos és sötét téma között. Ha a weboldal jelenleg világos témájú, akkor a gombra kattintva sötét stílust adunk az oldalnak. Ha a weboldal sötét témájú, akkor a gombra kattintva világos témájúvá tesszük az.

2. A JavaScript DOM manipulációs lehetőségei

A DOM tulajdonképpen egy objektumorientált reprezentációja a weboldalnak. A weboldalon szereplő elemek **Node**-ok (csomópontok) a DOM fában, amelyek számos **property**-vel (adattaggal) és **metódus**sal rendelkeznek. Ezeket JavaScriptből egyszerűen el tudjuk érni.

Ebben a fejezetben megismerkedünk a JavaScript néhány fontosabb DOM manipulációs lehetőségével. Sokszor fogjuk használni a **`document`** objektumot, ami lényegében a böngésző által megnyitott HTML dokumentumot reprezentálja, és hozzáférést biztosít a DOM fához.

2.1. HTML objektumok megkeresése a DOM fában

- `document.getElementById(id)`: visszaadja az adott `id` értékkel rendelkező elemet (egyetlen elemet ad vissza, hiszen az `id` értéke szabályosan egyedi a HTML dokumentumon belül)
- `document.getElementsByTagName(tag)`: visszaadja az adott tagnévvel rendelkező elemeket (minden esetben egy kollekciót ad vissza, amit 0-tól kezdődően indexelünk)
- `document.getElementsByClassName(class)`: visszaadja az adott `class` értékkel rendelkező elemeket (minden esetben egy kollekciót ad vissza, amit 0-tól kezdődően indexelünk)

Példa: Keressük meg az alábbi HTML kódban a **második** bekezdést a fenti három DOM metódussal, és írassuk ki a szöveges tartalmukat a konzolra! Egy elem szöveges tartalmát az `innerText` property-vel kérdezhetjük le.

```
<!DOCTYPE html>
<html lang="hu">
   <head>
      <title>DOM példa</title>
      <meta charset="UTF-8"/>
   </head>
   <body>
      <p id="first" class="underline">Ez itt az első bekezdés.</p>
      <p id="second" class="bold">Ez itt a második bekezdés.</p>
      <p id="third" class="underline">Ez itt a harmadik bekezdés.</p>

      <!-- Egy lehetséges megoldás -->

      <script>
        console.log(document.getElementById("second").innerText);           // id
        console.log(document.getElementsByTagName("p")[1].innerText);       // tagnév
        console.log(document.getElementsByClassName("bold")[0].innerText); // class
      </script>
   </body>
</html>
```

Egy másik lehetőség elemek megkeresésére a DOM fában:

- `document.querySelector(s)`: visszaadja az `s` CSS szelektor által kijelölt **legelső** elemet
- `document.querySelectorAll(s)`: visszaadja az `s` CSS szelektor által kijelölt **összes** elemet (egy kollekciót ad vissza, amit 0-tól kezdődően indexelünk).

Példa: A fenti két metódus segítségével CSS szelektorokat is megadhatunk az elemek keresésekor

```
<!DOCTYPE html>
<html lang="hu">
   <head>
      <title>DOM példa</title>
      <meta charset="UTF-8"/>
   </head>
   <body>
     <div>
       <!-- Ennek a bekezdésnek a szövegét szeretnénk kiíratni a konzolra... -->
       <p class="bold">Hello!</p>
     </div>

     <script>
       console.log(document.querySelector("div p.bold").innerText);    // Hello!
       console.log(document.querySelectorAll("p.bold")[0].innerText); // Hello!
     </script>
   </body>
</html>
```

2.2. HTML elemek tartalmának módosítása

Ahhoz, hogy egy HTML objektum tartalmát módosítsuk JavaScriptben, csupán az `innerText` vagy `innerHTML` property-k valamelyikének kell értéket adni.

A különbség a két property között, hogy az `innerHTML` értékeként megadott szöveg HTML-ként lesz értelmezve (ezért itt használható a szokásos HTML markup), míg az `innerText` értéke minden esetben egyszerű szövegként jelenik meg.

Példa: Bekezdések tartalmát módosító kód és annak kimenete

```
<!DOCTYPE html>
<html lang="hu">
   <head>
      <title>DOM példa</title>
      <meta charset="UTF-8"/>
   </head>
   <body>
     <p id="first">Ez itt az első bekezdés.</p>
     <p id="second">Ez itt a második bekezdés.</p>

     <script>
       const p1 = document.getElementById("first");   // 1. bekezdés megkeresése
       const p2 = document.getElementById("second");  // 2. bekezdés megkeresése
       p1.innerText = "Itt nem ez volt eredetileg..."; // szöveges tartalom módosítása
       p2.innerHTML = "<b>Valami félkövér szöveg...</b>"; // HTML tartalom módosítása
     </script>
   </body>
</html>
```

Itt nem ez volt eredetileg...

Valami félkövér szöveg...

2.3. HTML elemek beszúrása a DOM fába

Ahhoz, hogy egy új elemet hozzunk létre és szúrjunk be a DOM fába, a következő lépéseket tesszük meg:

1. A `document.createElement(tag)` metódussal létrehozzuk az új HTML elemet.
2. Beállítjuk az elem szöveges tartalmát és/vagy attribútumait.
3. Beszúrjuk az újonnan létrehozott elemet a DOM fába a szülő objektum `append()` vagy `appendChild()` metódusának meghívásával.

Az `append()` és `appendChild()` metódusok mindketten arra szolgálnak, hogy egy DOM-beli objektumhoz gyerekobjektumokat fűzünk hozzá. Két fontos különbség a két metódus között:

- Az `append()` több gyereket is hozzáfűzhet egy objektumhoz, míg az `appendChild()` csak egyet.
- Az `append()` `Node`-okat és stringeket is hozzáfűzhet egy objektumhoz gyerekként, míg az `appendChild()` kizárólag `Node`-okkal működik.

Példa: Szúrjunk be JavaScriptben egy új elemet az alábbi HTML listába!

```
<!DOCTYPE html>
<html lang="hu">
   <head>
      <title>DOM példa</title>
      <meta charset="UTF-8"/>
   </head>
   <body>
     <ul>
       <li>Első listaelem</li>
       <li>Második listaelem</li>
     </ul>

     <script>
       const ul = document.getElementsByTagName("ul")[0];  // lista megkeresése
       const newItem = document.createElement("li");       // listaelem létrehozása
       newItem.innerText = "Harmadik listaelem";    // listaelem szövegének beállítása
       ul.append(newItem);                          // listaelem hozzáfűzése a listához
     </script>
   </body>
</html>
```

- Első listaelem
- Második listaelem
- Harmadik listaelem