Nagy Gusztáv

Web programozás alapismeretek

Nagy Gusztáv

Web programozás alapismeretek

Ad Librum Kiadó
Budapest, 2011

A könyv elektronikus változata, a fontosabb példaprogramok és webcímek elérhetőek a http://nagygusztav.hu/ oldalról.

A szerző a lehetőségei szerinti legnagyobb gondossággal járt el a könyv írása közben. De ettől még számítani kell szerkesztői pontatlanságra, sőt nem zárható ki a tárgyi tévedés sem. Mindezzel együtt a könyv alkalmas az alapismeretek megszerzésére.

A kiadvány létrejöttét a Kecskeméti Főiskola[2] Gépipari és Automatizálási Műszaki Főiskolai Kara[3] támogatta

Szakmai és nyelvi lektor: *Dr. Pap-Szigeti Róbert*[4]

Kiadó: *Ad Librum Kft.*

1107 Budapest, Mázsa tér 2-6.

http://www.adlibrum.hu
info@adlibrum.hu

ISBN: 978-615-5110-26-9

Nyomda: *Litofilm Kft.*

Borító terv: *Várhegyi Attila*

E könyv megrendelhető a kiadótól a http://www.adlibrum.hu/Webprogramozas címen.

1 http://creativecommons.org/licenses/by-nc-nd/2.5/hu/
2 http://www.kefo.hu/
3 http://www.gamf.hu/
4 http://pap-szigeti.hu/

Bevezetés

Ezzel a könyvvel arra vállalkozom, hogy a Kecskeméti Főiskola GAMF Karán tanuló műszaki informatikus hallgatók kezébe olyan írásos anyagot adjak, amely az előadások és gyakorlatok mellett további segítséget ad a web kliens és szerver oldali nyelvei, alapvetőbb technológiái megismerésére.

A könyv feltételezi az alapvető programozási és hálózati alapismeretek meglétét. Ennek hiányában az anyag elsajátítására több időt kell fordítani.

A könyvkiadásról

A könyv elkészülését négy tanév oktatási tapasztalatai és hat jegyzetként közreadott verzió előzte meg. Az előző verzióhoz képest a kisebb javítások és kiegészítések mellett a HTML 5, a CSS 3 és a PHP 5-ös verzió néhány fontosabb újdonsága is bekerült. Ezek még nem mindig használhatók a munka során, de hamarosan szükségünk lesz az ismeretükre.

A PDF formátumú jegyzeteimet eddig 10-20.000 alkalommal töltötték le, elég sok weboldalon ajánlják, és több egyetemen (pl. ELTE, BME, Óbudai Egyetem, Debreceni Egyetem, Szegedi Egyetem) is használják kötelező vagy ajánlott irodalomként.

A könyv „ára”

- **3.600 Ft**, ha a kiadótól rendeli meg a papír alapú verziót
- **0 Ft** a GAMF nappali vagy levelező tanrendű, a jegyzethez tartozó kurzust felvevő hallgatója
- **0 Ft** annak számára, aki ugyan letölti, de nem találja hasznosnak
- egy köszönő **e-mail** a nagy.gusztav@gmail.com címre azok számára, aki ugyan hasznosnak tartja, de fizetni nem tud érte
- **2.500 Ft** annak számára, aki hasznosnak találja és tud is érte fizetni

Ez utóbbi esetben a nagy.gusztav@gmail.com címre küldött e-mailben lehet jelezni a fizetési szándékot a számlázási cím megadásával.

Akik inspiráltak abban, hogy ezt az üzleti modellt alkalmazzam:

- **Jézus Krisztus**, akitől (közvetve vagy közvetlen) mindent kaptam,
- A szabad szoftveres közösség

Kecskemét, 2011. június

a szerző

1

Az alapok

Ezzel a könyvvel nem vállalkozhatunk arra, hogy minden szükséges előismeretet bemutassunk és minden határterülettel foglalkozzunk. De felvázolunk néhány olyan területet, amely a weboldalak készítői számára ismert kell legyen. Egyes témákat alaposabban meg fogunk vizsgálni, más témákhoz pedig további anyagok forrásait fogjuk megnézni.

1.1. A web és a látogató viszonya

Webfejlesztőként magunk is látogatók vagyunk. Nap mint nap különböző weboldalakat látogatunk meg. Ahhoz azonban, hogy jó weboldalakat tudjunk készíteni, olyan módon kell látnunk a weboldalakat, ahogy azt korábban nem tettük. Folyamatosan szem előtt kell tartanunk mérnöki szempontokat is.

1.1.1. Webes tipográfiai alapismeretek

Sokunkkal próbálták jól-rosszul megtanítani a szövegszerkesztési alapismereteket. Azonban a papíralapú szövegszerkesztéssel kapcsolatos tanulmányaink hátrányunkra válhatnak, ha nem értjük meg a papír és a weboldal mint különböző médiák közötti különbségeket.

A webes tipográfia korlátai[5]

A nyomdászoknak sokféle lehetőség áll a rendelkezésükre, amikor szóba kerül a tipográfia, mint például a betűkészletek puszta száma vagy az elrendezési lehetőségek széles skálája. A webes tipográfia ennél sokkal korlátozottabb, mivel olyan típusokkal és elrendezéssel kell dolgozzunk, amelyről tudjuk, hogy elérhető és használható lesz azokon a gépeken is, amelyeken az olvasók megnyitják a lapot, hiszen senki nem fejleszt csak saját magának weboldalt.

5 Paul Haine: *Tipográfia a weben* című cikke alapján
http://dev.opera.com/articles/view/11-tipografia-a-weben/

A webes tipográfia korlátai többek között a következők:

- Korlátozott betűkészlet
- Nincs elválasztás, így a sorkizárt elrendezés csúnya lesz keskenyebb oszlopok esetén
- Nem lehet tudni, hogy hol és hogyan nézik majd meg a munkát, így a dizájnereknek minden képernyőméretre, felbontásra és környezetre gondolniuk kell

Ennek megértésére a következők olvasása javasolt:

- Paul Haine: Tipográfia a weben
  http://dev.opera.com/articles/view/11-tipografia-a-weben/
- rrd (Radharadhya dasa): Web tipográfia 1, 2, 3
  http://webmania.cc/web-tipografia-1/
  http://webmania.cc/web-tipografia-2/
  http://webmania.cc/web-tipografia-3/

1.1.2. Hogyan olvasunk a weben?

Ha weboldal készítésére adjuk a fejünket, akkor jó, ha tisztában vagyunk a látogatói szokásokkal. E témát egyre többen és egyre átfogóbban kutatják. Itt most csak egy rövid ajánló erejéig foglalkozunk a témával. Legalább a következő cikkek elolvasása célszerű a továbbhaladás előtt:

- Hogyan olvasunk a weben?
  http://www.agent.ai/main.php?folderID=4&articleID=2217&ctag=articlelist&iid=1
- Kámán Veronika: A jelen forradalma: olvasás a weben
  http://krono.inaplo.hu/index.php/inter/weblibrary/816-a-jelen-forradalma-olvasas-a-weben
- Kovács Balázs: Írás és olvasás a weben
  http://www.carnation.hu/hirl_cikk.php?id=47&cid=1

1.1.3. Kereső(re) optimalizálás

Ha egy weboldalt fáradságos munkával elkészítünk, szeretnénk, ha minél több látogató megtalálná az oldalunkat. Aki elolvassa a cikkeinket, hozzászól a blogunkhoz, vásárol a termékeink közül.

A látogató „szerzése” minden honlapnak célja. Ezért e témával is foglalkozunk néhány ajánlott irodalom elejéig. A látogatószerzés klasszikus módja a keresőmotorokban (pl. Google) való megjelenés, mégpedig minél előkelőbb helyen az általunk hőn áhított keresőszavakra.

Bár a témával foglalkozó írások, weboldalak, vállalkozások nem mindig tesznek különbséget a keresőoptimalizálás és keresőmarketing között, itt ezt megtesszük.

- A *keresőre optimalizálás* a saját oldalunk fejlesztésével történik. Emiatt minden weboldal-tulajdonosnak szüksége van rá. Mi itt erre tudunk koncentrálni.
- A *keresőmarketing* sok egyéb eszközt (pl. hírlevél, fizetett hirdetések) is felhasznál, amelyek nem képezik a weboldalunk részét. (Ez a terület nem témája a könyvnek.)

Néhány hasznos információforrás:

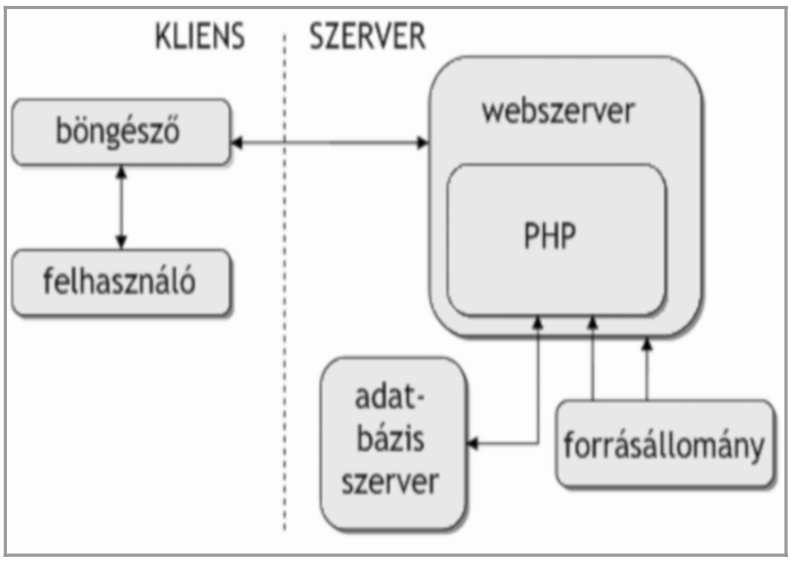
- Google keresőmotor-optimalizálási útmutató kezdőknek
  http://googlewebmastercentral.blogspot.com/2009/06/seo-starter-guide-now-available-in-40.html
- Jároli József: Mi a keresőoptimalizálás (Keresőmarketing)?
  http://webni.innen.hu/Keres_c5_91optimaliz_c3_a1l_c3_a1s
- Longhand: Keresőoptimalizálás
  http://longhand.hu/keresooptimalizalas

Érdemes azonban megjegyezni, hogy a keresőmarketing területén sok tévhit kering, és sok minőségen aluli cég kínálja szolgáltatásait. A tévhitekkel kapcsolatban két gondolatébresztőt említünk meg:

- Kungl István: SEO mítoszok
  http://www.seotools.hu/blog/seo-mitoszok
- Katona Zsuzsa: Rangsorolási tévhitek
  http://seo-training.eu/szakirodalom/google_ranking_tevhitek

1.2. A web működése

Az 1. ábra sokat segíthet a további információk megértésében.



1. ábra. A kliens-szerver architektúra

A *felhasználó*, aki a web szolgáltatásait ki akarja használni, megteheti ezt egy tetszőleges modern *webböngészővel*. (E két „szereplőt" együttesen a *kliens oldalnak* tekintjük.)

A felhasználó a böngészőt használva kezdeményezheti egyes weboldalak letöltését. A web kezdeti időszakában a webszerver azokat az állományokat tudta kiszolgálni, amiket a háttértárain elhelyeztek. (Ez tulajdonképpen *statikus tartalmat* eredményez, vagyis az ilyen tartalom jellemzően nem változik.) Bizonyos esetekben ez ma is így van: például egy honlapba illesztett kép nem fog megváltozni, akárhányszor töltjük is le, ezért a webszervernek a böngésző kérésére válaszul mindössze vissza kell azt adni.

Később egyre nagyobb igény lett a *dinamikus tartalmak* iránt, amikor a tartalom már a látogató tevékenységei, vagy más okok miatt színesebb, változóbb lehet. Ebben az esetben a webszerver nem önmaga válaszol a böngésző kérésére, hanem PHP, vagy más nyelvű program állítja elő a választ, amit a webszerver csak továbbít.

Tovább növelheti az oldal dinamizmusát, ha a tartalmak előállításához szükséges adatokat (legalább részben) *adatbázisban tároljuk*. Ekkor a PHP nyelvű forrásprogram az adatbázisszerverrel kapcsolatot épít fel, és adatbázisból származó információkat is felhasznál a válasz elkészítéséhez, illetve a felhasználók válaszait is eltárolja az adatbázisban.

1.2.1. Webszerver

A webkiszolgáló/webszerver egy kiszolgáló, amely elérhetővé teszi a rajta tárolt weblapokat a HTTP protokollon keresztül. A webszerverekhez webböngészőkkel lehet kapcsolódni.

Bár a webszerverek sok mindenben különböznek, az alapvető funkcióik azonosak. Minden webszerver HTTP kéréseket fogad a hálózatról, és HTTP válaszokat küld vissza. A HTTP válasz az esetek többségében egy HTML dokumentum, de lehet még egyszerű szöveges fájl, kép, vagy más típusú fájl is.

Kérés

A webszerverek a klienstől kapott kérésekben többek között *URL címet* kapnak, melyet aztán kétféleképpen értelmezhet a szerver a beállításaitól függően:

1. A tartománynév után álló relatív mappa és fájl struktúrát hozzárendelik egy gyökérmappához. (A gyökérmappa a webszerver beállításaiban van megadva, és az adatokat kérő kliens számára láthatatlan. További információk a 3.1.1. fejezetben.)
2. A tartománynév után álló relatív mappa és fájlstruktúra (vagy akár még a tartománynév is) teljesen független a kért címben szereplő struktúrától. Ebben az esetben szerver meghatározott szabályok szerint formázza a kért címet. Ennek segítségével egy mappára irányuló kérés teljesen más mappára vagy akár egy fájlra is mutathat és fordítva.

A kliens például az alábbi URL-t kéri: http://www.pelda.com/utvonal/fajl.html

A kliens webböngészője ezt értelmezve létrehoz egy kapcsolatot a www.pelda.com kiszolgálóval, és elküldi a következő HTTP 1.1 kérést:

```
GET /utvonal/fajl.html HTTP 1.1
Host: www.pelda.com
…
```

Válasz

1. A www.pelda.com címet megfelelteti a webszerver az adott gyökérmappához (pl. /var/www/pelda), amelyhez hozzáfűzi a /utvonal/fajl.html elérést – ezzel megtörtént a megfeleltetés a fájlrendszer erőforráshoz. A kért eredmény a szerveren tehát: /var/www/pelda/utvonal/fajl.html. Ezt követően a webszerver ellenőrzi, hogy kiszolgálható-e az adott kérés, ill. hogy létezik-e. Ha nem létezett, akkor 404-es hibakóddal tér vissza, ha hozzáférhető, akkor beolvassa, elvégzi rajta az esetleges további műveleteket, majd elküldi a kliensnek. A válasz természetesen szintén magában foglalja a megfelelő fejlécet.
2. A második megoldás esetében, az erőforrásokhoz történő megfeleltetés előtt a címet átformázza. Például:

```
www.pelda.com/toplista/kutyak+es+macskak
```

URL kérést a következőképpen alakíthatja át:

```
/var/www/pelda/toplista.php?cim=kutyak+es+macskak
```

Modulok

Lehetőség van a válaszok feldolgozása előtt, az esetlegesen a kérésben érkezett adatok feldolgozására és ennek eredményének visszaküldésére. Ilyenkor a szerver oldalon futó webszerver-modulok illetve a webszerver által meghívott CGI rutinok végzik el ezt a feladatot. A programrészletek (webszerver-modulok) rendszerint a HTML kódba vannak beágyazva és maga a webszerver-program hajtja ezeket végre. Ilyenek például a PHP, ASP, JSP.

1.2.2. Webtárhely

A mai weboldalak kis hányada igényli, hogy egy vagy esetleg több (ún. dedikált) szerver teljes egészében a weboldal kiszolgálását végezze. Éppen ezért a legtöbb honlap más honlapokkal osztozik egy webtárhely erőforrásain.

A *virtuális webtárhely szolgáltatás* alatt egy olyan internetes szolgáltatást értünk, ahol egy webszerver erőforrásait több felhasználó/honlap között osztják fel. Minden felhasználó egy a rendszer által dedikált tárhelyet foglal el, aminek nyilvános tartalma egyedi domén néven érhető el. Kisebb forgalmú weboldalt költséghatékonyan lehet bérelt webtárhelyen üzemeltetni. Tárhelyet ún. *tárhelyszolgáltatóktól*[6] bérelhetünk.

A webtárhely szolgáltatás általában tartalmaz egy *adminisztrációs felületet* (pl. cPanel[7]), hogy a bérlő a tárhelyét menedzselni tudja.

Osztott tárhelyszolgáltatók rendszerint az egyes szolgáltatásokat fizikailag elkülönített kiszolgáló rendszereken oldják meg, az ügyfélkiszolgáló és adminisztrációs rendszer, a levelező kiszolgáló, az adatbázis szerver, a webszerver fizikailag elkülönített kiszolgálókon működik. A legtöbb webkiszolgáló alacsony költségű Linux vagy BSD alapú *LAMP szerver*. (A LAMP a *Linux*, *Apache*, *MySQL*, *PHP* szavak rövidítését jelenti.)

Az egyes operációs rendszerekre épített szolgáltatások lényegében meghatározzák a felhasználó által elérhető technológiák csoportját is. Windows alapú webtárhely esetén a felhasználó választhat ASP.NET és Microsoft SQL Server, de akár PHP és MySQL Server támogatást is; míg LAMP szerver esetén csak PHP nyelvű weboldalakat készíthetünk MySQL Server támogatással.

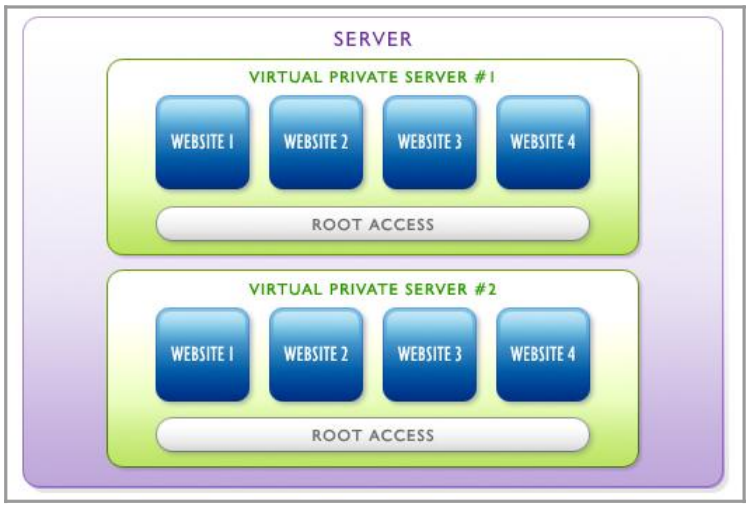
1.2.3. Virtuális szerver

A hagyományos osztott webtárhelyek helyett egyre népszerűbb megoldás a virtuális szerver alkalmazása.

A virtuális szerver, más néven VPS (Virtual Private Server) a valós szerver erőforrásainak egy bizonyos részét használhatja. Ezek az erőforrások lehetnek dedikáltak, azaz kizáróla-

6 http://tarhely.lap.hu/
7 http://www.cpanel.net/

gosan az adott virtuális szerver veheti igénybe a neki kiosztott erőforrásokat, fizikai hardver elemeket, vagy megosztottak: pl. két virtuális szerver használhatja az egyik processzszort, azaz osztoznak a teljesítményén, de lehetőségük van nagyobb kapacitás kihasználására is, amennyiben a másik fél kevésbé terheli (2. ábra).



2. ábra. VPS architektúra

A szerver virtualizációval teljesítményben és hardver felépítésben is (ezek virtuális hardverek, amik az egyes fizikai hardverekből vagy azok bizonyos hányadából állnak) fizikai géphez méltó konfigurációt hozhatunk létre, melynek kezelése gyakorlatilag megegyezik a valós szerverekével, ugyanakkor rengeteg előnnyel bír velük szemben.

1.2.4. HTTP protokoll

A HTTP (*HyperText Transfer Protocol*) egy információátviteli protokoll a világhálón. Az eredeti célja a HTML lapok publikálása és fogadása volt.

A HTTP egy *kérés-válasz alapú protokoll* kliensek és szerverek között. A kommunikációt mindig a *kliens kezdeményezi*. A HTTP klienseket gyűjtőnéven *user agent*-nek is nevezik. A user agent jellemzően, de nem feltétlenül webböngésző.

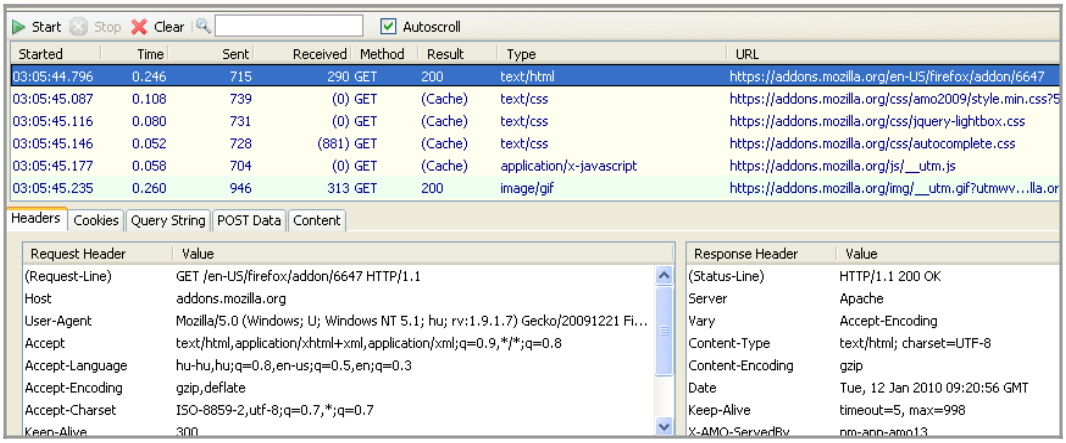
A HTTP általában a TCP/IP réteg felett helyezkedik el, de nem függ tőle.

HttpFox

A gyakorlatban a HTTP megismeréséhez a Firefox böngésző HttpFox[8] nevű kiegészítője nagyszerű lehetőséget nyújt.

8 https://addons.mozilla.org/en-us/firefox/addon/httpfox/

A 3. ábrán látszik, hogy a weboldal letöltéséhez (felül az első sor) milyen HTTP kérést küldött el a böngésző (lent balra), és milyen választ kapott a webszervertől (lent jobbra). A további sorok (fent) a HTML feldolgozása után szükséges CSS, JavaScript és kép állományok letöltését is mutatják.



3. ábra. A HttpFox működés közben

Kérés (`request`)

Egy HTTP kérés első sora mindig `metódus erőforrás verzió` alakú, például így:

```
GET /images/logo.gif HTTP/1.1
```

Ezt a sort követheti tetszőleges számú fejléc sor `kulcs: érték` alakban, például így:

```
Accept: text/plain,text/html
Accept-Language: en
```

A fejléc sorok végét egy *üres sor* jelzi, melyet az opcionális *üzenettest* követ. A sorokat a CRLF (azaz kocsi vissza + soremelés) karakterpárral kell elválasztani. A fejléc végét jelző üres sor csak ezt a két karaktert tartalmazhatja, nem lehet benne szóköz és tabulátor sem.

Metódusok

HTTP protokoll nyolcféle metódust definiál. A metódusok a megadott erőforráson végzendő műveletet határozzák meg.

`HEAD`	Ugyanazt adja vissza, mint a GET, csak magát az üzenettestet hagyja ki a válaszból.
`GET`	A megadott erőforrás letöltését kezdeményezi. Ez messze a leggyakrabban használt metódus.
`POST`	Feldolgozandó adatot küld fel a szerverre. Például HTML űrlap tartalmát. Az adatot az üzenettest tartalmazza.
`PUT`	Feltölti a megadott erőforrást.
`DELETE`	Törli a megadott erőforrást.
`TRACE`	Visszaküldi a kapott kérést. Ez akkor hasznos, ha a kliens oldal arra kíváncsi, hogy a köztes gépek változtatnak-e illetve mit változtatnak a kérésen.
`OPTIONS`	Visszaadja a szerver által támogatott HTTP metódusok listáját.
`CONNECT`	Átalakítja a kérést transzparens TCP/IP tunnellé. Ezt a metódust jellemzően SSL kommunikáció megvalósításához használják.

1. táblázat. HTTP metódusok

Biztonságosnak azokat a metódusokat nevezzük, amelyek csak információ lehívására szolgálnak és elvben nem változtatják meg a szerver állapotát. Más szóval mellékhatás nélküliek. Ilyenek például a HEAD, a GET, az OPTIONS és a TRACE. Fontos megjegyezni, hogy a gyakorlatban lehetnek a biztonságos metódusoknak is szerveroldali mellékhatásai.

Válasz (`response`)

A HTTP válasz első sora a státuszsor, amely `verzió státuszkód indoklás` alakú. A státuszkód egy három számjegyű szám, az indoklás egy angol nyelvű üzenet. Az előbbit inkább gépi, az utóbbit inkább emberi feldolgozásra szánták. Egy egyszerű példa státuszsorra:

```
HTTP/1.1 200 OK
```

A státuszkódok jelentését az RFC 2616[9] tartalmazza részletesen, az alábbi lista egy áttekintő osztályozást ad a kezdő számjegy alapján:

9 http://rfc-ref.org/RFC-TEXTS/2616/index.html

`1xx`	Informatív – Kérés megkapva.
`2xx`	Siker – A kérés megérkezett; értelmezve, elfogadva.
`3xx`	Átirányítás – A kérés megválaszolásához további műveletre van szükség.
`4xx`	Kliens hiba – A kérés szintaktikailag hibás vagy nem teljesíthető.
`5xx`	Szerver hiba – A szerver nem tudta teljesíteni az egyébként helyes kérést.

2. táblázat. HTTP válasz kódok

A státuszsor után fejléc sorok következhetnek a HTTP kérésnél látott módon, például így:

```
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

A fejléc sorokat itt is üres sor zárja, melyet az opcionális üzenettest követ.

Munkamenet (*session*)

A HTTP egy állapot nélküli protokoll. Az állapot nélküli protokollok előnye, hogy a szervernek nem kell nyilvántartania felhasználói információkat az egyes kérések kiszolgálása között. A HTTP eredetileg nem arra készült, hogy felhasználók jelentkezzenek be rajta keresztül szerverekre és ott munkamenetet (*session*-t) indítsanak. Történetileg azonban úgy alakult, hogy a HTTP terjedt el széles körben más, felhasználói bejelentkezést támogató protokollok helyett, ami arra kényszerítette a webfejlesztőket, hogy *kerülőutakon* járva tárolják a felhasználók munkamenet-állapotait, ha arra szükség van.

1.2.5. FTP protokoll

A *File Transfer Protocol*, a TCP/IP hálózatokon történő állományátvitelre szolgáló szabvány.

Gyakran van szükség arra, hogy valamilyen állományt hálózaton keresztül töltsünk le saját gépünkre, vagy egy állományt mások számára hozzáférhetővé tegyünk. Erre alkalmas az FTP, ami lehetővé teszi a különböző operációs rendszerű gépek között is az információcserét. A hozzáférési jog alapján kétféle kapcsolattípus létezik:

- letöltés, vagy feltöltés *nyilvánosan hozzáférhető* állományokból vagy állományokba,
- letöltés, vagy feltöltés olyan gépről, ahol *azonosítóval rendelkezünk*.

Azt a folyamatot, amikor egy távoli számítógépről fájlt mentünk a saját számítógépünk háttértárára, *letöltésnek* nevezzük; *feltöltésnek* nevezzük, ha a folyamat fordított irányban zajlik, és mi töltünk fájlt más gépére.

Az FTP kapcsolat *ügyfél/kiszolgáló alapú*, vagyis szükség van egy kiszolgáló (szerver) és egy ügyfélprogramra (kliens). Elterjedt protokoll, a legtöbb modern operációs rendszerhez létezik FTP-szerver és kliens program, sok webböngésző is képes FTP-kliensként működni.

1.2.6. Webcím (URL)

A webcím, más néven URL (amely a *Uniform Resource Locator,* vagyis *egységes erőforrás-azonosító* rövidítése), az Interneten megtalálható bizonyos erőforrások (például szövegek, képek) szabványosított címe. Először Tim Berners-Lee alkotta meg a World Wide Weben való használatra. A jelenleg használt formátumot részletesen leírja az IETF RFC 1738[10] szabványa.

A webcím az Internet történetének alapvető újítása. Egyetlen címben összefoglalja a dokumentum megtalálásához szükséges négy alapvető információt:

- a *protokollt*, amit a célgéppel való kommunikációhoz használunk;
- a szóban forgó gép vagy *tartomány nevét*;
- a hálózati *port számát*, amin az igényelt szolgáltatás elérhető a célgépen;
- a fájlhoz vezető *elérési utat* a célgépen belül.

Egy tipikus, egyszerű webcím így néz ki:

```
http://hu.wikipedia.org:80/wiki
```

Ennek részei:

- A `http` határozza meg a használandó protokollt. A protokoll neve után kettőspont (`:`) írandó.
- A `hu.wikipedia.org` adja meg a célgép tartománynevét. Ez elé két perjel (`//`) írandó.
- A `80` adja meg a célgép azon hálózati portszámát, amin kérésünket várja; ez elé kettőspont (`:`) írandó. Ezt a részt gyakran teljesen elhagyhatjuk, például esetünkben a HTTP protokoll alapértelmezett portszáma a 80.
- A `/wiki/Bit` a kért elérési út a célgépen. Ez a rész mindig a per-jellel (`/`) kezdődik.

A legtöbb böngésző nem is igényli, hogy a `http://` részt begépeljük egy weblap eléréséhez, hiszen az esetek döntő többségében úgyis ezt használjuk. Egyszerűen begépelhetjük a

10 http://tools.ietf.org/html/rfc1738

lap címét, például: `hu.wikipedia.org/wiki/Bit`. A címlap megtekintéséhez általában elég a tartomány nevét beírni, például `hu.wikipedia.org`.

A webcímek egyéb részeket is tartalmazhatnak, `http` esetében például az elérési út után, egy kérdőjel (*?*) mögé helyezve keresési kérdés szerepelhet, ami egy *get* metódusú HTML űrlapból vagy speciális módon összeállított hiperhivatkozásból származik. Az elérési út után, attól egy kettős kereszttel (*#*) elválasztva szerepelhet a hiperszöveg egy részére hivatkozó azonosító. Ez az azonosító nem része a webcímnek, de gyakran szerepel vele kapcsolatban.

Példák:

```
http://hu.wikipedia.org/w/wiki.phtml?title=Bit&action=history
http://hu.wikipedia.org/wiki/1999#Események
```

A webcím az URI[11] egy olyan fajtája, ahol az azonosítás a dokumentum helye alapján történik.

Ha egy weblapot egyértelműen meghatároz egy webcím, akkor *rá mutató hivatkozást hozhatunk létre*. Ez a helyzet nem mindig áll fenn, pl. egy menüpont megváltoztathatja a lapon belül az egyik keret[12] tartalmát anélkül, hogy ennek az új kombinációnak külön webcíme lenne. Ezen kívül egy weblap függhet ideiglenesen tárolt információtól is. Még ha van is egy weblapnak vagy keretnek önálló webcíme, ez nem mindig nyilvánvaló annak a számára, aki rá mutató hivatkozást kíván létrehozni. Egy keret webcíme nem látszik a címsávban, és létrehozható címsáv nélküli ablak is. A webcím ilyenkor a lap forrása, illetve egyes részeinek „tulajdonsága” alapján meghatározható.

Azon kívül, hogy rá mutató hivatkozást hozzunk létre, egyéb okokból is kíváncsiak lehetünk egy lap webcímére: ha tartalmát önállóan akarjuk megjeleníteni, illetve, ha bizonyos megszorításokat (eszköztár nélküli, vagy kicsi, méretezhetetlen ablak) meg akarunk kerülni.

1.3. A tervezés folyamata

Mint minden mérnöki feladat, a webfejlesztés sem képzelhető el tervezés nélkül egy bizonyos méret felett. Először is tisztáznunk kell a céljainkat.

1.3.1. A honlap célja

A szerző a megrendelővel való kommunikációt egy interjú formájában javasolja kezdeni, az alábbi kérdésekkel. A kérdésekre adott válaszok között persze nagy lehet az átfedés, az

11 Az URI az erőforrást kétféleképp azonosíthatja: hely szerint (URL) vagy név szerint (URN).

12 A keretek (frame-ek) segítségével az ablak több téglalapra felbontható, és minden részben önálló HTML állomány lesz megjelenítve. Ma már elavult és kerülendő technikának számít.

interjú célja az elérendő célok teljes körű feltérképezése. Egy konkrét esetben tehát bizonyos kérdéseket ki is hagyhatunk, ha azok feleslegesnek, értelmetlennek tűnnek. Más esetben további kérdések is szükségessé válhatnak.

- Mi a célunk a honlappal?
- Kik lesznek a látogatóink?
- Mit szeretnénk bemutatni?
- Mik a kulcsfontosságú funkciók?
- Milyen visszajelzéseket várunk a látogatóinktól?
- Részt vesznek-e a látogatóink a tartalom előállításában?
- Miben fog a tartalom és a szolgáltatás fejlődni (pl. 1 hónap, 1 év múlva)?
- Ki fogja az oldalt karbantartani?
- Milyen csoportos és személyes jogosultsági körökre lesz szükség?
- Hol és hogyan szeretnék a honlapot az interneten reklámozni?
- Látogatottsági statisztikákat szeretnénk-e megismerni?

Természetesen a kérdések akkor is alkalmazhatók, ha a saját vagy cégünk honlapját szeretnénk elkészíteni.

További részletek a szerző honlapján találhatóak:

- http://nagygusztav.hu/honlap-interju-mi-az-oldal-celja

Néhány hasonló forrás:

- Kanga Design – Gyakori kérdések
  http://kangadesign.hu/honlap-keszites-gyakori-kerdesek.html
- Wolf Gábor: Top 10 honlap hiba – és hogyan javítsd ki?
  http://www.marketingcommando.hu/cikkek/top-10-honlap-hiba.html
- 7even: Mit gondoljon át Weboldal indítás előtt
  http://www.7even.hu/weboldal-keszites/weboldal-inditas-elott

1.3.2. A honlap megtervezése

Ha már tudjuk, mi a célunk a honlapunkkal, akkor kezdjünk bele az oldal megtervezésébe. Ehhez meg kell válaszolnunk még a következő kérdéseket.

Milyen oldalaink lesznek?

A válasz egy konkrét, tételes lista legyen. Például:

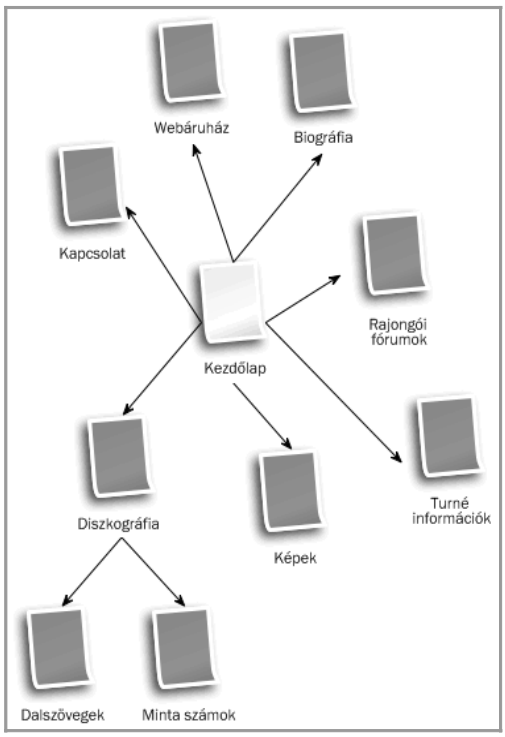
- kezdőoldal (hírekkel)
- kapcsolat
- termékkategóriák tartalomjegyzéke
- termékkategóriák oldalai
- termékek oldalai
- vendégkönyv oldal
- stb.

Milyen viszonyban állnak az oldalak egymással?

Itt az alá-fölé rendeltségi viszonyon kívül gyakori a mellérendelt kapcsolat is. A válasz az oldal navigációjának kitalálásában fog segíteni.

Illusztrációként nézzünk meg egy zenei együttes honlapjának oldaltérképét[13] (4. ábra).

13 Forrás: http://dev.opera.com/articles/view/6-informacios-architektura-egy-website-t/



4. ábra. Oldaltérkép

Ezután nevezzük el az oldalainkat, és írjuk hozzá rövid tartalmi összefoglalókat.

Hogyan épüljenek fel az oldalaink?

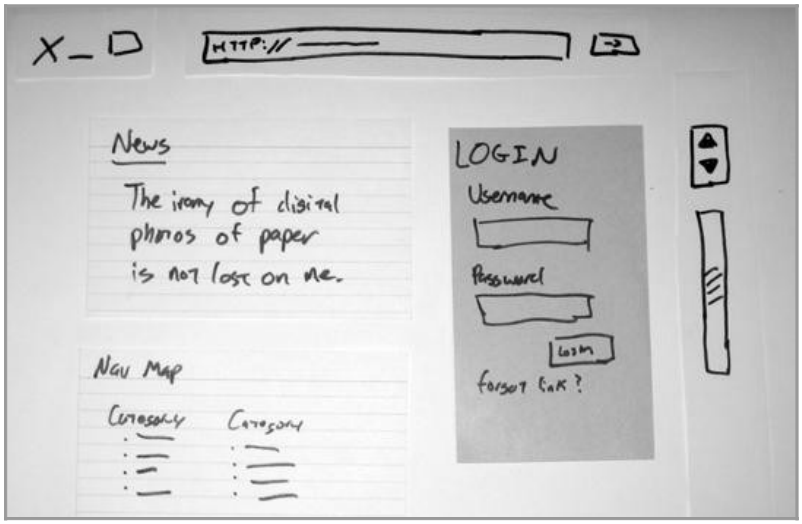
Az oldal funkcionális látványtervét tervezve el kell döntenünk, hogy

- a klasszikus 1, 2 és 3 oszlopos oldalelrendezés közül melyiket választjuk,
- hova kerüljön a logó, főcím, lábléc stb.
- hogyan épüljön fel a navigációs struktúra (felső menü, bal oldali menü, kenyérmorzsa menü[14], címkefelhő, stb.)
- a menük egy vagy többszintűek legyenek
- melyik szélső oszlopban milyen tartalmú dobozok és milyen sorrendben szerepeljenek

14 A kenyérmorzsa menü a felhasználói felületeken használt navigációs eszköz, amely a kiindulóponttól a felhasználó jelenlegi tartózkodási helyéig vezető utat mutatja. Pl.:
Címlap » ECDL » Vizsgaközpontok » GAMF Kar

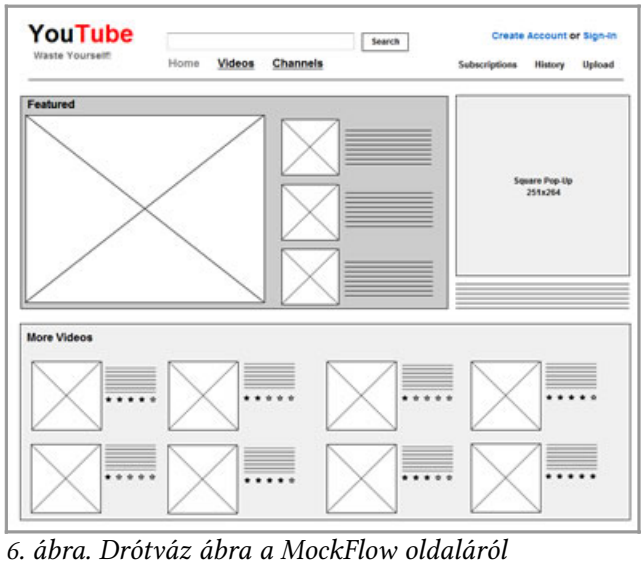
Jól át kell gondolnunk, hogy a látogatóink számára a leglogikusabb, legáttekinthetőbb struktúrát tudjuk nyújtani.

Az oldalelrendezés megtervezéséhez igen hasznos a *Paper prototyping* módszere (5. ábra).



5. ábra. Paper prototyping példa

Ezen kívül egyre elterjedtebb a drótváz (*wireframe*, *mockup*) eszközök használata is. Példaként a MockFlow egy Youtube tervét[15] nézzük meg (6. ábra).



6. ábra. Drótváz ábra a MockFlow oldaláról

15 Forrás: http://www.mockflow.com/samples/

További információk:

- http://dev.opera.com/articles/view/6-informacios-architektura-egy-website-t/
- http://weblabor.hu/blog/20100901/jo-weboldal-terv
- http://www.alistapart.com/articles/paperprototyping
- http://ergomania.blog.hu/2010/09/28/drotvaz_vagy_szoveg
- http://opendir.hu/?freedom=/tartalom/weboldal_tervezes/drotvazas_weboldal_tervezes_online_eszkozzel
- http://arth2o.com/blog/honlap-drotvaz-es-gui-prototipus

1.4. A fejlesztőkörnyezet kialakítása

Általános esetben a szerverkörnyezet kialakítása a rendszergazda feladatköréhez tartozik, mégis fontos, hogy alapszinten átlássuk a feladatokat, lehetőségeket.

Másrészt a saját fejlesztői gépünkön alkalmazott szoftverek is nagyban befolyásolják a munkánkat, így erről is ejtünk néhány szót.

1.4.1. Szerver operációs rendszer

Először is nézzük meg, melyek ma a webkiszolgálókban gyakran használt operációs rendszerek.

Linux

Az elterjedtebb, és nem kimondottan asztali (*desktop*) használatra szánt Linux[16] disztribúciók telepítésével működő web-, és adatbázis szervert kapunk. Akár régebbi, más felhasználások számára értéktelen hardverre is telepíthetünk Linuxot. Egy kisebb forgalmú honlapot tökéletesen képes kiszolgálni.

A nagy nevű disztribúciók friss verziói a mai csúcsgépek meghajtására és csúcs igények kiszolgálására alkalmas, szintén könnyen telepíthető lehetőséget nyújtanak a rendszergazdáknak.

Önálló szerver üzemeltetése nélkül, web-hosting szolgáltatás[17] igénybe vételével is többnyire Linux alapú szerverekkel találkozhatunk.

16 http://www.linux.hu/
17 http://tarhely.lap.hu/

BSD

Kevésbé közismert, de egyes rendszergazdák körében stabilitása miatt nagy népszerűségnek örvendő Unix[18] operációs rendszerek. Az alapvető változatok szerver feladatokra optimalizáltak, így nagyszerűen alkalmasak a webes kérések kiszolgálásához.

Windows

Bár éles webes környezetben nem jelentős a részesedése, azért előfordul az alkalmazása. A tanulás szakaszában azonban sokszor a legkézenfekvőbb megoldás az egyetlen (többnyire Windows operációs rendszerrel működő) számítógépünket szerverré alakítani. A fejezet további részében ezzel a témával fogunk foglalkozni.

Ha a fejlesztéshez Windows operációs rendszert alkalmazunk, akkor érdemes néhány technikai dologra figyelni.

- Először is az állománynevekben a Windows nem tesz különbséget kis-, és nagybetű között, ezért ha pl. HTML-ben vagy CSS-ben nem vagyunk következetesek, akkor Windows alatt még működni fog az oldal, de ha a kész munkát más operációs rendszert futtató gépre kell átvinni, akkor az problémás lesz. Javasolható, hogy webes fájl esetén nagybetű soha ne szerepeljen a fájlnévben.
- Az FTP kliens alkalmazások az ilyen jellegű hibák elkerülése érdekében általában lehetővé teszik az FTP-vel átvitt állományaink kisbetűsítését is.
- Hasonló okok miatt nem célszerű az állománynevekben ékezetes betűket vagy egyéb speciális karaktereket alkalmazni.
- Végül szintén fontos, hogy a könyvtárnevek megadásánál a `\` helyett mindig a `/` jelet használjuk, és soha ne adjunk meg Windows alatt érvényes teljes elérési utat, pl. kerülendő a `<img src="C:\www\kep.gif">` forma.

1.4.2. Szerver alkalmazások

Linux és BSD alatti telepítést most nem tárgyaljuk, a használt disztribúció dokumentációját érdemes fellapozni[19]. Itt a Windows alatti szerver környezet kialakítását fogjuk tárgyalni.

XAMPP integrált telepítő csomag

Mivel a szerver alkalmazások telepítése nem mindig egyszerű feladat, próbálkozhatunk előre csomagolt, és minden szükséges alkalmazást telepítő és bekonfiguráló programokkal

18 http://www.bsd.hu/

19 Pl. LAMP szerver:
http://sugo.ubuntu.hu/community-doc/jaunty/universe/apps/lamp-server.html

is. Ezek közül csak egyet mutatunk be részletesen, a többi alkalmazása hasonló. A szolgáltatások körében lehetnek jelentősebb eltérések is.

A telepítő-csomagok hátránya, hogy bár többnyire működnek, egy esetleges hiba előállása esetén a hiba megszüntetése eléggé bajos lehet, mivel kevésbé tudunk a csomag működésébe belelátni.

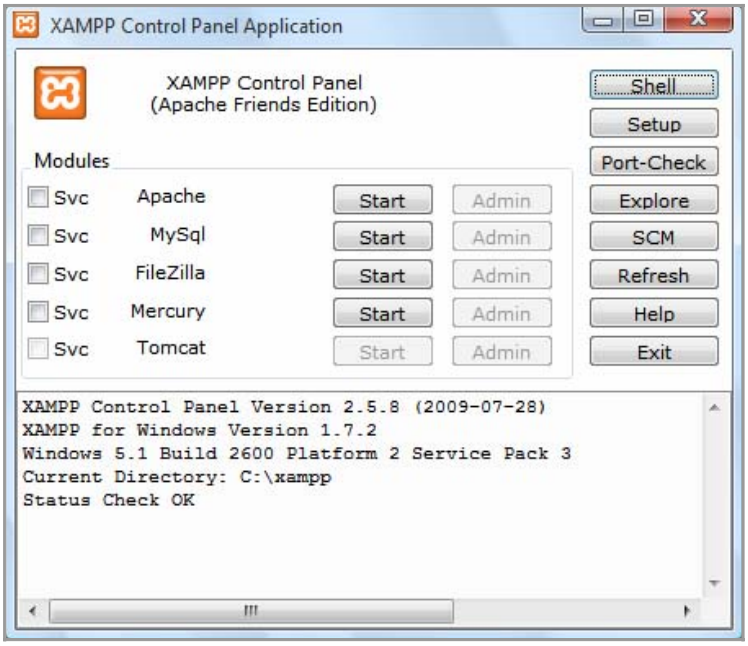
A szerző által leginkább ajánlott integrált telepítő csomag az *XAMPP*[20]. Az *XAMP for Windows 1.7.3* változata – többek között – a következő szoftvereket telepíti és konfigurálja:

- *Apache 2.2.14 (IPv6 enabled) + OpenSSL 0.9.8l*
- *MySQL 5.1.41 + PBXT engine*
- *PHP 5.3.1*
- *phpMyAdmin 3.2.4*

A letöltött telepítőprogram a Windows alatti telepítések szokásos kérdéseit teszi fel. Legfontosabb a telepítés helye. A telepítés után a Start menüből és parancssorból is vezérelhetjük az alkalmazásokat, de legegyszerűbb az *XAMPP Control Panel* alkalmazása (7. ábra).

A webszerver működését a böngésző cím sorába írt `localhost` cím lekérésével tudjuk tesztelni. Ekkor az `It works!` szöveg a működés bizonyítéka. A PHP és a MySQL működését legegyszerűbben a `localhost/phpmyadmin` lekérésével tesztelhetjük.

20 http://www.apachefriends.org/en/xampp-windows.html



7. ábra. XAMPP Control Panel

1.4.3. A fejlesztő gépe

Szerverként akár a saját gépünket használjuk, akár egy távoli szervert, a következő szoftverekre szükségünk lesz.

Böngésző

A weboldalt látogatók igen sokféle böngészőt használnak. Emiatt a webfejlesztőnek – legalábbis az elterjedtebb típusokat – ismernie és használnia kell tesztelési célokból.

Hogy pontosan milyen arányban használják a látogatók az egyes böngészőket és azok verzióit, igen sok vita tárgya. A szerző a weboldal jelenlegi adatait a Google Analytics alapján szokta figyelni, általános információként pedig a http://www.w3schools.com/browsers/browsers_stats.asp statisztikáit.

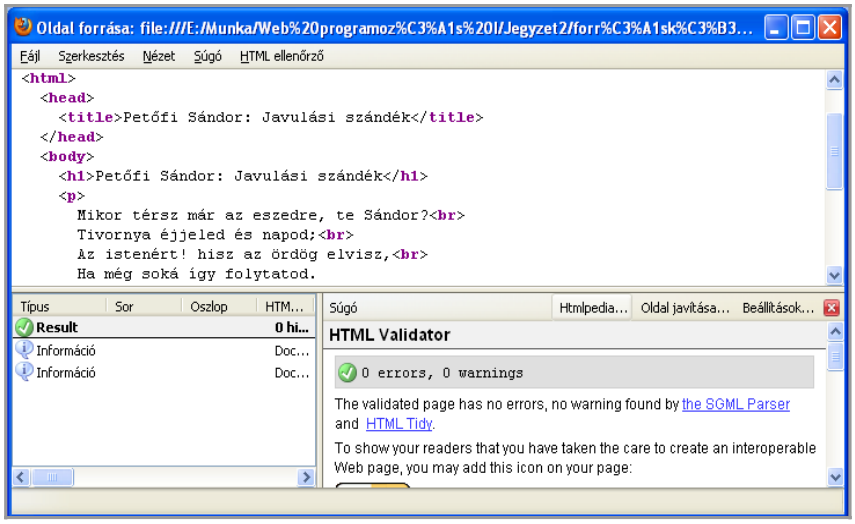
Ezen kívül fejlesztőként sem mindegy, melyik böngésző mennyi támogatást ad a fejlesztési folyamat gyorsításához. Ebből a szempontból legjobb választás a Firefox, amelyhez jó néhány fejlesztéshez használható kiegészítő tölthető le[21].

A szerző által gyakrabban használt kiegészítők:

21 https://addons.mozilla.org/hu/firefox/extensions/web-development/

HTML Validator[22]

A 8. ábrán látható, ahogy a színkiemelés alatt a szintaktikai hibák és a hibák részletes leírása is megtekinthetők. Kis ügyességgel (*Oldal javítása* gomb) akár a hibák egy részétől is megszabadulhatunk.

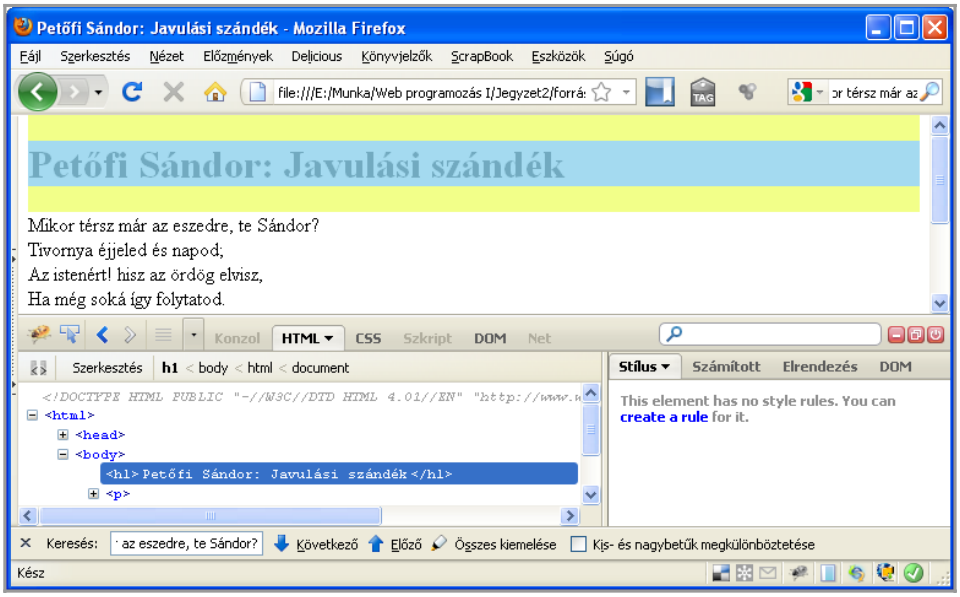


8. ábra. Forrás megtekintése a HTML Validator megjelenítésével

FireBug[23]

A 9. ábrán látható kiterjesztés az oldalt és a forráskódját teljesen szimultán mutatja, érdemes megfigyelni az egérkurzorral irányítható kék dobozokat.

22 http://users.skynet.be/mgueury/mozilla/
23 http://getfirebug.com/



9. ábra. A Firebug működés közben

A következő videókkal alaposabban megismerhetjük a Firebug szolgáltatásait:

- rrd (Radharadhya dasa): Firebug 10 percben
  http://webmania.cc/firebug-10-percben/
  http://webmania.cc/firebug-10-percben-2/
  http://webmania.cc/firebug-10-percben-3/

Web Developer[24]

A Web Developer segítségével szintén rengeteg információt gyűjthetünk az oldalról (10. ábra), de akár tesztelhetjük is CSS, JavaScript vagy képek nélkül az oldal kinézetét, működését.

24 https://addons.mozilla.org/hu/firefox/addon/60/

10. ábra. Web developer div információk

HTML Validator[25]

A HTML Validator az oldalunk HTML és CSS szabványosságát tudja ellenőrizni.

Fájlkezelő

A webfejlesztés közben igen gyakran van szükség állománykezelési feladatokra. Ehhez a Windows intéző helyett célszerű valamelyik két paneles szoftvert alkalmaznunk. Az állományok másolása, szinkronizálása, biztonsági mentések kezelése napi szintű feladat.

Legismertebb manapság a Total Commander[26], Linux alatt a Krusader[27], de vannak szép számmal további szoftverek is.

FTP kliens

Szükségünk lesz valamilyen FTP kliensre az állományok szerverre feltöltéséhez. Erre a célra megoldás lehet a *Total Commander*, de célszerű inkább megismerni a *FileZilla*[28] alkalmazást. A FileZilla egy szabad FTP kliens, amely könnyen használható, több szálon is képes fel-, és letölteni, folytatja a megszakadt letöltéseket, több operációs rendszeren és nyelven működik. Az egyre gyakoribb SFTP-hozzáférés miatt jó megoldás a *WinSCP*[29] használata is.

Programozói editor, IDE

A webfejlesztés során időnk jó részében forráskódok begépelését fogjuk végezni. Ezért nagyon nem mindegy, milyen támogatást kapunk ehhez az editorunktól. Nem igazán alkal-

25 https://addons.mozilla.org/hu/firefox/addon/249/
26 http://www.totalcommander.hu/
27 http://www.krusader.org/
28 http://filezilla-project.org/
29 http://winscp.net/

mas a feladatra a Jegyzettömb. De nem nagyon alkalmasak az ún. *WYSIWYG* (*What You See Is What You Get*) editorok sem, mert túlságosan megkötik a kezünket.

A szerző véleménye szerint a webfejlesztőnek olyan eszközökre van szüksége, amelyek úgy adnak támogatást, hogy a folyamatot a fejlesztő, és nem a program vezérli. Ezen szoftverek egyik legjobbika a *Komodo Edit*[30], amelyik a kódszínezés mellett kódkiegészítéssel és élő (FTP feletti) szerkesztéssel is megbirkózik.

Képszerkesztő program

A fejlesztő a valós életben többnyire kép(ek) formájában kapja meg az oldal látványtervét, valamilyen szöveges formátumban a szöveges részt, és nyers formában a tartalomhoz kapcsolódó fényképeket. Ebben az esetben a fejlesztő feladata az, hogy a látványterv alapján elkészítse az oldal HTML és CSS kódját, és „ráhúzza" minderre a dizájnt, a szöveget és a fényképeket. A grafikus programok szempontjából ez azt jelenti, hogy a dizájnt alkotó képet téglalap alakú részekre kell vágni, a fényképeket pedig méretre hozni, javítani, színkorrekciót, világosítást stb. végrehajtani. E feladatok ellátására részben a Paint is megfelel, de érdemes valamivel komolyabb szoftvert alkalmazni.

A *Paint.NET*[31] olyan ingyenes képszerkesztő, amely a számtalan hagyományos rajzeszköz mellett több különleges rajzszerszámmal rendelkezik Ilyenek a varázspálca, az utómunkálatra használható átszínező ecset vagy a klónbélyeg. A beépített, látványos effektusai segítségével egyszerűvé válik a retusálás. További érdekessége a rétegkezelés, a lapolvasók, a digitális fényképezőgépek és rajzpadok kezelése.

Linux alatt a *GIMP*-et alkalmazhatjuk valós idejű kötegfeldolgozó rendszerként, vagy mint nagy teljesítményű képszámoló, de használhatjuk egyszerűen képkonverternek is. A GIMP hihetetlenül jól bővíthető. Úgy lett tervezve, hogy plug-in bővítésekkel bármire képes legyen. Fejlett script-felülete lehetővé teszi, hogy a legegyszerűbb feladatoktól a legbonyolultabb képfeldolgozási eljárásokig minden könnyen elérhető legyen parancsfájlokból is.

30 http://www.activestate.com/komodo-edit
31 http://www.getpaint.net/

2

A tartalom és a kinézet

A weboldalak eredeti, és máig legfontosabb célja a tartalmak közzététele. Erre a HTML mellett a CSS nyelvet használjuk.

A HTML a tartalom szerkezetét, a CSS pedig a kinézetét írja le. E kettő szorosan összefügg, de megfelelő tervezéssel precízen el is választható egymástól.

A szerző kedvenc illusztrációs példája a CSS Garden[32] oldala, ahol ugyanazt a HTML struktúrát ezernyi különböző módon lehet megtekinteni, pusztám a dizájn (CSS állományok és a képek) cseréjével.

2.1. HTML alapok

A HTML nyelv az az alap, amivel minden webfejlesztőnek először meg kell ismerni, és alaposan tisztában kell lenni. Ez a fejezet segítséget ad a HTML lehetőségeinek megismeréséhez, de több nyelvi elem bemutatásától is eltekint. Ennek főbb okai:

- Bizonyos HTML jellemzők a mai napra elavultnak tekinthetők. Itt elsősorban a kinézet esztétikai megjelenésére kell gondolni. A CSS használatával ugyanis sokkal több és jobb lehetőségünk lesz a kinézet leírására. A HTML a mai gyakorlatban már tisztán csak az információra, és annak struktúrájára figyel. Ezt *szemantikus kódolás*nak is nevezzük.
- Bizonyos tagok, tulajdonságok a böngészők által nem egységesen támogatottak, így ezeket a gyakorlatban is csak kevesen használják.

2.1.1. Mi az a HTML?

- A HTML a *Hyper Text Markup Language* rövidítése
- A HTML állomány egyszerű szövegállomány, amely rövid jelölő tagokat tartalmaz
- A jelölő tagok alapján tudja a böngésző, hogyan kell megjelenítenie az oldalt

32 http://www.csszengarden.com/tr/magyar/

- A HTML állomány `html` kiterjesztéssel rendelkezik
- A HTML állományt egyszerű szöveges (editor) programokkal (pl. Jegyzettömb) is létrehozhatunk

2.1.2. Hogyan kezdjünk neki?

Windows operációs rendszer alatt indítsuk el a Jegyzettömböt (vagy inkább egy komolyabb editort, 1.4.3. fejezet), majd gépeljük be a következő szöveget:

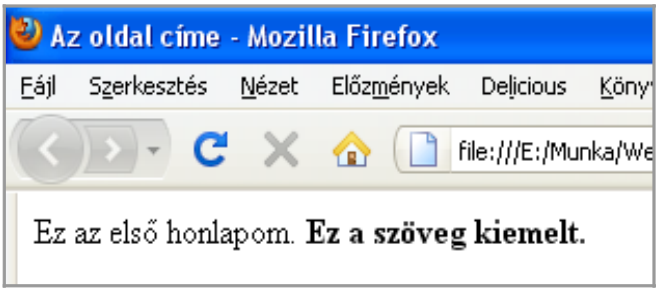
HTML

```
<html>
  <head>
    <title>Az oldal címe</title>
  </head>
  <body>
    Ez az első honlapom. <strong>Ez a szöveg kiemelt.</strong>
  </body>
</html>
```

Mentsük el a szöveget `oldal.html` néven!

Ebben és a következő néhány fejezetben még nincs feltétlen szükségünk webszerver elérésére a tanuláshoz. Később majd FTP kapcsolaton keresztül a webszerverre fogjuk az oldalainkat feltölteni, majd webszerver elérésével tesztelni azokat.

Nyissuk meg a böngészőnket, majd a *Fájl* menü *Webcím megnyitása* parancsát választva keressük meg az előbb elmentett `oldal.html` állományt! (További lehetőségünk, hogy a *Windows Intéző*ben, vagy *Total Commander*ben duplán kattintunk az állomány nevére. De az állomány böngészőre vonszolásával is célt érhetünk.) A 11. ábrához hasonlót kell látnunk a böngészőnkben:



11. ábra. Az első HTML oldal a böngészőben

A példa magyarázata

A dokumentum első tagja a `<html>`. A böngésző erről fogja tudni, hogy hol kezdődik a HTML oldal. Az utolsó tag a `</html>`, itt ér véget a dokumentum a böngésző számára.

A `<head>` és `</head>` tagok közötti rész a fejléc információ. Az itt leírt szöveget a böngésző nem jeleníti meg közvetlenül.

A `<title>` tagok közötti szöveget a böngésző a címsorában jeleníti meg, ahogy az ábrán is láttuk.

A `<body>` tagok közötti szöveg jelenik meg a böngésző ablakában.

A `<strong>` és `</strong>` tagok hatására a szöveg kiemelten (alapértelmezetten félkövéren) jelenik meg.

2.1.3. HTML szerkesztők

Léteznek olyan szerkesztőprogramok, amelyekkel tényleges HTML ismeretek nélkül is lehet HTML oldalakat létrehozni. Ezeket a programokat *WYSIWYG* (what you see is what you get) editoroknak hívjuk. Ezek azonban kerülendők, ha minőségi HTML oldalakat szeretnénk létrehozni. (Legalábbis a tanulás kezdeti fázisában.) Ezek a programok ugyanis kisebb-nagyobb mértékben „teleszemetelik” a kódot.

Érdemes inkább olyan szerkesztőprogramot választani, ahol a HTML kód fölött teljes ellenőrzéssel bírunk, ugyanakkor kiegészítő szolgáltatásokkal (pl. színkiemelés, tagok beszúrása gombnyomásra stb.) gyorsítani lehet a munkát. (Lsd. 1.4.3. fejezetben.)

2.1.4. Hogy nézzük meg egy oldal HTML kódját?

Gyakran előfordul, hogy a weben böngészve megtetszik egy oldal, és szeretnénk megnézni a forrását. (A szerző véleménye szerint ez az egyik legjobb módszer a tanulásra.) Hogyan tehetjük ezt meg?

Keressük meg a böngészőnk *Nézet* menüjét, majd *Forrás*, vagy *Oldal forrása* (vagy valami hasonló nevű, böngésző függő) menüpontot.

A szerző javasolja a fejlesztéshez a *Firefox* nevű böngészőt, amely eleve webfejlesztők számára lett kifejlesztve, és több ezer kiterjesztése (plug-in) közül jó néhány a HTML forrás könnyen áttekinthető megjelenítését szolgálja. (Az 1.4.3. fejezetben bemutattuk a legjobbakat.)

2.1.5. HTML tagok

A HTML állomány egyszerű szövegállomány, amely rövid jelölő *tag*okat tartalmaz.

A HTML tagok segítségével *elem*ek definiálhatók.

HTML tagok jellemzői

- A HTML tagok jelölik ki a HTML elemeket

- A HTML tagot a < és > írásjelek veszik körül (ezek az írásjelek az egyszerű szövegekben nem engedélyezettek)
- A HTML tagok általában párban vannak, mint a `<html>` és `</html>`
- A pár első tagja a kezdő, a második a záró tag
- A szöveg (tartalom) a kezdő és a záró tag között helyezkedik el
- A HTML tagok kis-, és nagybetűvel is írhatók

2.1.6. HTML elemek

A korábbi példában a következő példa egy elem:

```HTML
<strong>Ez a szöveg kiemelt.</strong>
```

A HTML elem kezdő tagja `<strong>`, az `Ez a szöveg kiemelt.` a tartalom, és `</strong>` a záró tag.

A következő is egy HTML elem:

```HTML
<body>
  Ez az első honlapom. <strong>Ez a szöveg kiemelt.</strong>
</body>
```

Az elem kezdő tagja `<body>` és a záró tagja `</body>`

2.1.7. Tag tulajdonságok

A tagok tartalmazhatnak tulajdonságokat (más néven attribútumokat, jellemzőket) is. Ezek a jellemzők járulékos információk az elem egészére nézve.

Az `img` tag segítségével képet tudunk az oldalunkkal beilleszteni. Ehhez azonban meg kell mondanunk, hogy hol találja a böngésző a kép állományt. Például, ha szeretnénk egy logót (`logo.png`) megjeleníteni a weboldalunkon, a következő szöveget kell begépelnünk:

```HTML
<img src="logo.png">
```

A tulajdonságok név-érték párokkal (`src="logo.png"`) adhatók meg, egymástól szóközzel elválasztva akár több is.

Melyik idézőjelet alkalmazzuk?

A nyelv elsősorban a (dupla) idézőjel alkalmazását írja elő. A böngészők az aposztróf jelet is elfogadják, mégis érdemes inkább a hagyományos idézőjelet alkalmazni.

2.1.8. Általános tulajdonságok

A HTML tagoknak különböző tulajdonságaik lehetnek. Később meg fogjuk ismerni a különböző HTML tagok speciális tulajdonságait. Ebben a fejezetben az általánosan használható tulajdonságok kerülnek a középpontba.

Alaptulajdonságok

- `class`: az elem osztálya, több tagnak lehet ugyanaz az értéke. Célja, hogy több elemet is azonos kinézetűre formázhassunk.
- `id`: a tag egyedi azonosítója, ugyanazon érték nem fordulhat elő többször.
- `style`: soron belüli stílus definíció.
- `title`: rövid súgó szöveg (ún. tooltip) definiálása.

Nyelvi tulajdonságok

- `dir`: az írás irányát definiálja
- `lang`: a nyelvet adja meg

Ezek segítségével akár egy bekezdésnek, vagy más elemnek is adhatunk egyedi információt a weboldal alapértelmezéséhez képest. (Pl. egy magyar oldalon idézünk egy francia verset.)

Billentyű tulajdonságok

- `accesskey`: gyorsbillentyűt rendel az elemhez.
- `tabindex`: tabulátorral történő elemváltás sorrendjét befolyásolja. (Űrlapok esetén használható.)

2.1.9. Megjegyzések

A megjegyzés tagot megjegyzések elhelyezésére használjuk. A böngésző nem veszi figyelembe a megjegyzésbe írt szöveget. (Természetesen a megjegyzés megtekinthető a forráskódban.)

```
<!—Ez egy megjegyzés -->
```

A megjegyzésben nem fordulhat elő két kötőjel a > nélkül.

2.1.10. Karakter entitások

Bizonyos karakterek (mint például a < és >) speciális jelentésűek a HTML-ben, ezért nem használhatók a folyó szövegben. Ezenkívül vannak olyan speciális írásjelek, amelyek hagyományos billentyűzetről nem vihetőek be, vagy nem is szerepelnek az ASCII kódtáblában.(Pl. a © jel.) Ha egy ilyen speciális karaktert akarunk megjeleníteni, akkor karakter entitást kell alkalmaznunk.

A karakter entitás három részből áll: `&` az elején, `;` a végén, a kettő között pedig egy entitás név, vagy `#` és számkód. Ha például a < jelet szeretnénk megjeleníteni, akkor a dokumentumba az `<` vagy a `&#60;` karaktersorozatot kell gépelnünk. Az entitások kis-nagybetű érzékenyek.

Nem törhető szóköz

A gyakorlatban talán a legtöbbet alkalmazott karakter entitás a nem törhető szóköz. A HTML a több egymást követő elválasztó (ún. white space) karaktert csak egy szóközként jeleníti meg. Ilyen esetekben szokás a ` ` entitást egymás után többször alkalmazni, ugyanis ezeket ténylegesen figyelembe veszi a böngésző. Ez azonban nem felel meg a HTML eredeti céljának, és a mai technikai lehetőségeknek sem. (Stílus formázások segítségével ezek a problémák sokkal elegánsabban oldhatók meg.)

Ennek az entitásnak eredetileg az a célja (és a szerző véleménye szerint csak ilyen esetben szabadna alkalmazni), hogy a több szóból álló kifejezések (például tulajdonnév) esetén a sor végén ne törje szét a böngészőnk a kifejezést, hanem mindenképpen egy sorba kerüljenek. Például a következő név mindig egy sorba, tördelés nélkül fog kerülni:

```HTML
Nagy Gusztáv
```

Ékezetes karakterek

Az angol abc-ben nem szereplő karakterek (így a magyar nyelv ékezetes karakterei is) sokáig problémát okoztak a HTML szerkesztés során. Ezért korábban szokás volt az ékezetes karaktereket is entitások segítségével megadni. A mai napra azonban ezek a problémák lényegében megszűntek, ezért a szerző véleménye szerint teljesen indokolatlan az entitások alkalmazása. Helyette inkább a pontos karakterkódolásra érdemes figyelmet fordítani.

A karakterkódolás megadására először HTML meta tagok lehetőségét fogjuk megismerni. Később a PHP-ből küldött HTTP header alkalmazása még jobb megoldás lesz. További információk a 3.1.5. fejezetben találhatóak.

További karakter entitások

Leírás	*Jelentés*	*Entitás név*	*Entitás számkód*
	nem törhető szóköz	` `	` `
<	kisebb, mint	`<`	`&#60;`
>	nagyobb, mint	`>`	`&#62;`
&	és	`&`	`&#38;`
"	idézőjel	`&quot;`	`&#34;`
'	aposztróf	`&apos;`	`&#39;`
§	bekezdés	`&sect;`	`&#167;`
©	copyright	`&copy;`	`&#169;`

3. táblázat. További karakter entitások

Az entitások teljes listáját a HTML referenciában érdemes[33] keresni.

2.1.11. Szemantikus HTML

A szemantikus HTML nem más, mint a HTML tagok jelentésének betartása, rendeltetésszerű használata. Például, egy bekezdést nem használunk fel menü kialakításához.

Azért fontos ez, mert a keresőgépek, automatikus katalógusok, HTML értelmezők ez alapján tudják kategorizálni az oldal tartalmait. Például egy felolvasóprogram egy címsort ki tud emelni felolvasás közben is, vagy egy keresőgép tud idézetekre keresni egy bizonyos szerzőtől, ha jól meg van formázva a HTML.

A gyakrabban előforduló hibákkal szemben néhány jó megközelítés:

- A címsorokat a `<h*>` elemekkel adjuk meg, úgy ahogy a tartalom strukturálódik; a `<h1>`-el kezdjük, ez az oldal címe, majd jönnek a továbbiak
- Az oldalnak egyedi címet adunk a fejlécben (`title`): az egész webhely címe + oldal címe).
- A bekezdések szövegeit `<p>` elemekbe tagoljuk, az új bekezdésnél nem csak a kurzort visszük új sorba egy `<br>` elemmel, hanem valóban jelezzük a bekezdés végét és egy új kezdetét.

33 Egy olvasmányosabb összefoglaló található a http://htmlhelp.com/reference/html40/entities/latin1.html címen.

- A különböző listákat a lista elemekkel definiáljuk. (`<ul>`, `<ol>`, `<dl>`). Talán nem triviális, de a menüket is `<ul>`, `<li>` elemmel érdemes megadni. (A menüpontok is egyfajta listát jelentenek.)

Ide tartozik még, hogy a HTML 5 tovább erősíti a szemantikus HTML kialakítás lehetőségeit. Ezekről később lesz szó, pl. a 2.4.7. fejezetben.

2.1.12. Szabványosság

A HTML nyelv a kezdetektől fogva szigorú szabályokra épült. A *Microsoft Internet Explorer* és a *Netscape Navigator*[34] harcának egyik melléktermé ke, hogy a böngészők felismernek, értelmeznek olyan HTML oldalakat is, amelyek nem felelnek meg a szabványnak. Sőt a webfejlesztők ezekre a pontatlanságokra rá is szoktak, és ebből a korszakból sok máig is elérhető de elavult szemléletű, ismertető található a weben.

A HTML nyelv minden verziója, változata egy úgynevezett *Document Type Definition* (*DTD*) segítségével definiált. A böngészők számára segítség, ha a dokumentum elején pontosan leírjuk, hogy melyik verzióhoz tartjuk magunkat.

A HTML 4.01 változata 3 féle DTD-vel érvényesíthető.

A legpontosabb *(strict)* változat nem engedélyezi az elavult részek (többnyire formázó elemek és tulajdonságok) használatát, valamint a kereteket. A következő elemet kell a dokumentum első elemeként elhelyezni:

HTML

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Az átmeneti (*transitional*) DTD megengedi az elavult részek használatát is.

HTML

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

Keretek használata esetén használható harmadik DTD-t gyakorlatilag nem fogjuk használni.

A HTML 5-ös verziója leegyszerűsíti a DTD használatot. Csupán ennyit ír elő:

HTML 5

```
<!DOCTYPE html>
```

Az oldalunk HTML szabványosságának ellenőrzésére több lehetőségünk van:

- http://validator.w3.org/ online használata
- Firefox böngészőbe beépülő HTML Validator[35].

34 A Netscape 2007 utolsó heteiben – hosszú haldoklás után – befejezte pályáját. Átadta helyét a pusztán kísérleti célokból létrehozott, de villámgyorsan felemelkedő *Mozilla Firefoxnak.*

35 http://users.skynet.be/mgueury/mozilla/

2.1.13. HTML 5

Bár a HTML 5-ös változata[36] még nem végleges szabvány, folyamatosan erősödik a támogatása. Emiatt mindenképpen érdemes megismerkednünk a még fejlesztés alatt álló, de holnap már szabadon használható megoldásokkal. Többek között a 2.4.7., 2.7.3. és 2.10.2. fejezetekben olvashatnak ennek részleteiről.

2.1.14. XHTML

Az XHTML nyelvet a HTML 4 leváltására hozták létre egy évtizede, de ez nem terjedt el a gyakorlatban. A HTML 5 jelenlegi fejlődésének fényében úgy tűnik, hogy nem is fogunk vele a jövőben sokat találkozni. Ezt azonban csak később tudhatjuk meg. Nem felesleges azonban, ha tudunk a következő néhány alapelvről. Sőt, akár még hasznos is lehet az alkalmazásuk. A következő sorokban néhány pontban megnézzük, miben tér el egy XHTML dokumentum a HTML-től.

Egymásba ágyazás

Az elemek csak megfelelően egymásba ágyazva és mindig lezárva lehetnek.

A következő példában az egymásba ágyazással van probléma:

```
<b><i>Ez egy félkövér és dőlt szöveg</b></i>
```
HTML

A helyes megoldás:

```
<b><i>Ez egy félkövér és dőlt szöveg</i></b>
```
XHTML

A tagok neveit kisbetűvel kell írni

A tagok neveit kisbetűvel kell írni, ezért a következő hibás:

```
<BODY>
  <P>Ez egy bekezdés.</P>
</BODY>
```
HTML

A helyes változat:

```
<body>
  <p>Ez egy bekezdés.</p>
</body>
```
XHTML

Elemek lezárása

A HTML elemeket le kell zárni. Hibás:

36 http://www.w3.org/TR/html5/

```
<p>Ez egy bekezdés.
<p>Ez egy másik bekezdés.
```
HTML

A helyes:

```
<p>Ez egy bekezdés.</p>
<p>Ez egy másik bekezdés.</p>
```
XHTML

Záró pár nélküli elemek

Záró pár nélküli elemek esetén is le kell zárni. Itt azonban egy rövidített írásmód is alkalmazható:

```
Sortörés<br />
Elválasztó vonal<hr />
Kép <img src="happy.gif" alt="Happy face" />
```
XHTML

A / előtti szóköz csak a visszafele kompatibilitás miatt kell, az XHTML nem igényli. Tehát azért kell szóköz, hogy a csak HTML-t ismerő böngészők is meg tudják jeleníteni az oldalt.

A tulajdonság-értékeket mindig idézőjelbe kell tenni

Ez praktikus HTML-nél is, ha a tulajdonság értéke pl. szóközt tartalmaz. XHTML-ben azonban mindig kötelező:

```
<table width="100%">
```
XHTML

A tulajdonságok kötelezően tartalmaznak értéket is

Ha a HTML nem ír elő értéket (vagyis a tulajdonság lényegében logikai jelzőként működik), akkor az érték megegyezik a tulajdonság nevével.

```
<input checked="checked" />
<input readonly="readonly" />
<input disabled="disabled" />
<option selected="selected" />
<frame noresize="noresize" />
```
XHTML

A `name` tulajdonság helyett az `id` használandó

Hibás tehát:

```
<img src="picture.gif" name="kep" />
```
HTML

Helyes:

```
<img src="picture.gif" id="kep" />
```
XHTML

A régebbi böngészőkkel való kompatibilitás miatt mindkettőt is szokták alkalmazni:

```
<img src="picture.gif" id="kep" name="kep" />
```
XHTML

2.2. CSS alapok

2.2.1. Mi a CSS?

A CSS a Cascading Style Sheets rövidítése

- A stílusok a HTML megjelenítési elemei és attribútumai helyett használhatók, azoknál jóval több lehetőséget biztosítva
- A stílusok meghatározzák, hogy hogyan jelenjenek meg vizuálisan a HTML elemek
- A stíluslapok segítségével könnyen szét lehet választani az oldal tartalmát annak kinézetétől (a dizájntól)
- A stílusokat általában külön állományban tároljuk (`.css` kiterjesztéssel)
- Külső stíluslapokkal gyorsítani tudjuk a munkavégzést
- Több stílus is hatással lehet egy elem megjelenésére

Kedvcsináló

Mielőtt a CSS alapos megismeréséhez kezdenénk, mindenképpen célszerű a CSS Zen Garden[37] oldalát meglátogatni. Ez az oldal azzal a céllal jött létre, hogy a CSS-sel szembeni ellenérzéseket lerombolja, és bemutassa, milyen óriási lehetőségek vannak a dizájner kezében, ha a CSS-t komolyan és szakszerűen használja. Ezen az oldalon ugyanazt a HTML oldalt láthatjuk sok-sok különböző dizájnná formálva – csupán a CSS állomány (és természetesen a dekorációs képek) cseréjével. A teljesség igénye nélkül néhány képernyőkép (12-15. ábra):

37 Magyar oldal: http://www.csszengarden.com/tr/magyar/

12. ábra. CSS Zen Garden példa

13. ábra. CSS Zen Garden példa

14. ábra. CSS Zen Garden példa

15. ábra. CSS Zen Garden példa

A stílusok használatának okai

A HTML tagok eredetileg arra lettek megalkotva, hogy a dokumentum tartalmát definiálják. Amikor egy címet, bekezdést vagy táblázatot akarunk létrehozni, akkor használhatjuk pl. a `h1`, `p` és `table` tagokat. A tényleges megjelenítés a böngészőre van bízva, eldöntheti, hogy mit hogyan jelenítsen meg a tagok, az ablakméret, a felhasználó beállításai alapján.

Később a két nagy böngésző (a *Netscape Navigator* és az *Internet Explorer*) újabb és újabb HTML tagokat és tulajdonságokat adott a böngésző által felismert HTML nyelvhez (pl. a `font` tag, vagy a `color` tulajdonság). Így belekeveredtek a HTML nyelvbe a megjelenítést befolyásoló elemek. (Ráadásul a böngészők csak részben és később támogatták a vetélytárs újításait.)

A problémát a World Wide Web Consortium[38] (W3C), egy non-profit, szabványokat alkotó szervezet oldotta meg. A HTML 4.0-ás verziójával és a vele párhuzamosan fejlesztett CSS segítségével létrejött egy jól használható eszközpáros a webfejlesztők részére.

Mára minden jelentősebb böngésző támogatja a CSS-t, bár a támogatottság mértékében vannak eltérések.

A Microsoft (és több más, magát a saját területén monopolhelyzetben tudó cég) módszere, hogy a független szabványokat (a szerző véleménye szerint) tudatosan és szándékosan figyelmen kívül hagyja azt remélve, hogy a versenytársak helyzetét ezzel lehetetlenné teszi. Egészen a Firefox megjelenéséig és a felhasználók körében való viharos sebességű elterjedéséig (kb. 2005-ig) ez a taktika sikeresnek látszott. Később azonban kénytelen volt változtatni hozzáállásán, az ígéretek szerint az IE 9-es megelőzi a más böngészőket HTML 5 és CSS 3 támogatottságban. Kb. 2009-től a Google Chrome is komoly részesedést ért el. Ebben a felgyorsult versenyben az igazi győztesek mi, a felhasználók vagyunk.

A stíluslapokkal munkát spórolhatunk

A stíluslapok azt definiálják, hogy hogyan jelenjenek meg az egyes HTML elemek. A stíluslapokat külön `.css` kiterjesztésű állományban szokás elhelyezni. Így könnyedén lehet ugyanazt a megjelenítést adni a honlap összes oldalához, mindössze egyetlen CSS állomány szerkesztésével. Ha bármit változtatni kell a dizájnon, lényegében csak ezt az egyetlen állományt kell módosítanunk.

Egy elemre több stílusdefiníció is hatással van

A stílusok egy vagy több elemre, vagy akár az egész oldalra is hatással lehetnek (ez utóbbi a `body` elem formázásával). Megfordítva: egy elemre hatással lehet a soron belüli stílusa, a `head` elembeli formázás és akár külső CSS állomány is. Sőt egy HTML oldalhoz akár több külső CSS állományt is rendelhetünk.

Lépcsős elrendezés

Melyik stílus fog érvényesülni, ha több stílust is definiálunk ugyanahhoz a HTML elemhez?

A következő négy beállítás érvényesül egyre nagyobb prioritással (tehát ütközés esetén a későbbi felülírja az előzőt).

1. a böngésző alapbeállítása
2. külső stíluslap

38 http://www.w3.org/

3. `head` elemben definiált stílus
4. soron belüli stílus

Tehát a soron belüli stílus a legmagasabb prioritású, és felülír minden alacsonyabb szintű formázást.

Ezen kívül fontos még figyelembe venni egy általában jól használható *ökölszabályt*: *az a stílusdefiníció fog érvényesülni, amelyik a legszűkebb hatókörrel rendelkezik*. Pl. ha minden bekezdést barna színűre állítunk, de egy egyedi azonosítóval rendelkező bekezdést zöldre, akkor az utóbbi szabály rendelkezik szűkebb hatókörrel. Így ennél a bekezdésnél zöld lesz a szín.

Kezdő (sőt középhaladó) CSS használat esetén is nagyszerűen lehet boldogulni az igen bonyolult szabályok precíz ismerete nélkül, erre az ökölszabályra és a józan paraszti észre alapozva. Egy bevezetőt azonban mégis figyelmükbe ajánlunk Tommy Olsson tollából[39].

2.2.2. Hol legyenek a stílusdefiníciók?

A stílus információkat háromféle módon rendelhetjük hozzá a HTML dokumentumunkhoz. Célszerű azonban éles környezetben a külső stíluslapokat alkalmazni.

Külső stíluslap

A külső stíluslap alkalmazása a legideálisabb eset. Ekkor a HTML állomány az oldal informatív részét tartalmazza, a külső CSS állomány pedig összegyűjtve a megjelenítésre vonatkozó formázásokat.

A HTML oldalhoz a `head` elemben elhelyezett `<link>` tag segítségével csatolhatunk külső CSS állományt:

```HTML
<head>
  <link rel="stylesheet" type="text/css" href="stilus.css">
  ...
</head>
```

A CSS egy újabb megközelítése az `@import` használata:

```HTML
<head>
  <style type="text/css">
    @import url("stilus.css");
  </style>
  ...
</head>
```

Ezzel a szintaxissal akár a CSS állományba is importálhatunk további CSS állományokat, a C vagy PHP nyelvek `include` megoldásaihoz hasonlóan.

39 http://dev.opera.com/articles/view/28-inheritance-and-cascade/#thecascade

Külső stíluslap fájlt nem HTML oldalanként külön-külön, hanem akár egy egész portálhoz egyetlen (vagy néhány) fájlként szokás készíteni. Így a további oldalak látogatása esetén nincs szükség a CSS fájl újbóli letöltésére.

Beágyazott stíluslap

Beágyazott stíluslapot kész oldalnál akkor szokás alkalmazni, ha az oldal egyedi (a külső stíluslapot nem lehetne másik oldalhoz felhasználni), vagy nagyon egyszerű stílusról van szó. Persze a dizájn kialakításának fázisában is jó megoldás lehet.

Beágyazott stíluslapot a `head` elemben szereplő `style` tag segítségével definiálhatunk.

```HTML
<head>
  <style type="text/css">
    body {background-color: red}
    p {margin-left: 20px}
  </style>
  ...
</head>
```

Soron belüli stílus

Soron belüli stílust esetleg akkor szokás alkalmazni, ha olyan egyedi stílusról van szó, amelyik sehol máshol nem fordul elő.

A szerző véleménye szerint még ilyen esetekben sem érdemes ezt alkalmazni, a CSS alapos megismerésével és ügyes szervezéssel ezt el lehet kerülni.

Soron belüli stílus alkalmazásához a kiválasztott elemet kell `style` tulajdonsággal ellátni. A következő példa csak e bekezdés megjelenítését változtatja meg:

```HTML
<p style="color: red; margin-left: 20px">Ez egy bekezdés</p>
```

2.2.3. A CSS nyelvtana

A nyelvtan három elemet különböztet meg: kiválasztó, tulajdonság és érték:

```CSS
kiválasztó {tulajdonság: érték}
```

A kiválasztó legegyszerűbb esetben egy HTML tag, a tulajdonság azt határozza meg, hogy milyen jellemzőt akarunk módosítani, míg az érték a változást határozza meg. A tulajdonságot és az értéket egy kettősponttal kell egymástól elválasztani, és a kettőt együtt kapcsos zárójelbe tenni:

```CSS
body {color: black}
```

Ha az érték több szóból áll, *idézőjelbe* kell tenni:

```CSS
p {font-family: "sans serif"}
```

Ha egy kiválasztó esetén *többféle tulajdonságot* is módosítani szeretnénk, könnyedén megtehetjük, mindössze pontosvesszővel kell elválasztani a tulajdonság-érték párokat.

```
p {text-align:center; color:red}
```
CSS

A stílusdefiníciók jobb olvashatósága érdekében inkább *több sorba* érdemes tagolni:

```
p {
  text-align: center;
  color: black;
  font-family: arial
}
```
CSS

Egyszerre akár *több kiválasztóra* is érvényesíthetjük a formázást. Ekkor a kiválasztókat vesszővel elválasztott listaként kell felsorolni. A példában minden címet zölden szeretnénk megjeleníteni:

```
h1, h2, h3, h4, h5, h6 {
  color: green
}
```
CSS

Osztály alapú kiválasztás

Osztály kiválasztó segítségével más-más módon tudjuk megjeleníteni az egyes osztályokba sorolt elemek tartalmát. A példában a két különböző osztályhoz tartozó bekezdések más-más formázást kapnak:

```
p.right {text-align: right}
p.center {text-align: center}
```
CSS

Ez a két stílus a következő két bekezdés megjelenésére hatással lesz:

```
<p class="right">
  Ez egy jobbra igazított bekezdés.
</p>
<p class="center">
  Ez egy középre igazított bekezdés.
</p>
```
HTML

A `p` tagoknak nem kötelező megadni `class` tulajdonságot, vagy akár más is lehet a `class` értéke, de ezekben az esetekben a fenti stílusoknak nem lesz hatása a bekezdésekre.

Az osztály szintű kiválasztást nem kötelező taghoz kötni, lehet tagoktól független, általános osztály kiválasztót is definiálni. A példa minden center osztályú elemet középre igazít. (Már amelyik HTML elemnél egyáltalán van értelme a középre igazításnak.)

```
.center { text-align: center }
```
CSS

A formázás minden center osztályú tagot középre igazít:

```HTML
<h1 class="center">
  Ez egy középre igazított cím.
</h1>
<p class="center">
  Ez egy középre igazított bekezdés.
</p>
```

Érdemes megemlíteni, hogy a `class` tulajdonság több értéket is kaphat, egymástól szóközzel elválasztva:

```HTML
<h1 class="center alahuz">
```

Ilyen esetben már az egyik kiválasztó (`.center`) használata esetén is találat lesz. Ha csak azokat a tagokat akarjuk kiválasztani, amelyek több `class` tulajdonságot is tartalmaznak, akkor a `.center.alahuz` szintaxist kell használnunk. Ekkor az egyik osztályba igen, de a másikba nem tartozó elemek nem lesznek kiválasztva.

Azonosító alapú kiválasztás

A HTML elemeknek megadhatjuk az egyedi `id` tulajdonságot. Így az egyedi `id`-vel rendelkező elemhez speciális formázást határozhatunk meg. CSS-ben a # segítségével tudunk elemet `id` alapján kiválasztani.

A következő példában a `menu` azonosítójú elem betűszínét zöldre állítjuk:

```CSS
#menu {color: green}
```

A `para1` azonosítójú bekezdést középre igazítva és piros színnel definiáljuk:

```CSS
p#para1 {
  text-align: center;
  color: red
}
```

Univerzális kiválasztó

A * szelektorral minden elemet egységesen tudunk formázni:

```CSS
* { border: 1px solid #000000; }
```

Tulajdonság kiválasztó

A következő példa csak azokra a képekre fog vonatkozni, amelyek alt tulajdonsága meg van adva:

```CSS
img[alt] { border: 1px solid #000000; }
```

A következő példa még a tulajdonság értékét is figyelembe veszi:

```CSS
img[src="alert.gif"] { border: 1px solid #000000; }
```

Gyermek kiválasztó

Gyakori, hogy az elemek túlzott `id` és `class` tulajdonsággal való ellátása helyett inkább az elemek hierarchiájára (más megfogalmazásban leszármazására, mint a családfánál) építünk.

Pl. a következő példa csak a `h3` elemen belül közvetlenül elhelyezkedő `strong` tagra lesz érvényes. Tehát nem vonatkozik a `h3` elem más részeire, és nem vonatkozik a nem közvetlenül `h3`-ban levő `strong` elemekre sem.

```css
h3 > strong { color: blue; }
```
CSS

Leszármazott kiválasztó

Az előzőhöz igen hasonló lehetőség nem csak a közvetlen szülő-gyermek kapcsolatra, hanem az akár több szintű öröklődésre, vagy máshogy fogalmazva a tartalmazásra épít. Nézzünk egy példát:

```html
<div>
  <em>Szia!</em>
  <p>Ez a bekezdés
    <em>köszönt</em> téged.
  </p>
</div>
```
HTML

Ekkor a `div > em` kiválasztó csak az első, míg a `div em` kiválasztó mindkét `em` tagot formázza.

Testvér kiválasztó

Gyakran van szükség arra, hogy egymást közvetlen követő elemek (vagyis testvérek) irányából fogalmazzuk meg a kiválasztásunkat. Pl. egy `h2`-es címet követő bekezdések közül az elsőt máshogy szeretnénk formázni, mint a továbbiakat. Ekkor jó megoldás lehet a `h2 + p` kiválasztó, amely a `h2` utáni `p` tagot címzi meg.

Megjegyzés

A CSS fájlba azért szoktunk megjegyzéseket tenni, hogy a későbbi megértést és módosítást könnyebbé tegye. CSS megjegyzésként egyedül a C nyelvből ismert `/*...*/`megjegyzés használható:

```css
/* Ez itt egy megjegyzés */
p {
  text-align: center;
/* Ez itt egy másik megjegyzés */
  color: black;
  font-family: arial
}
```
CSS

A gyakorlatban szoktuk még arra is használni a megjegyzéseket, hogy egyes formázásokat ideiglenesen kikapcsoljunk vele.

2.2.4. Szervezési elvek

Egy komolyabb dizájn kialakításakor igen összetett CSS kódunk lesz. Nem egyszerű úgy leírni a kódot, hogy az egy későbbi továbbfejlesztés esetén, esetleg más fejlesztő számára is jól olvasható, egyértelmű lesz. De önfegyelemmel, és hasznos alapelvek kialakításával, betartásával mégis elérhető. Nézzünk meg néhány ötletet ezzel kapcsolatban.

A CSS lépcsős formázása

Bár a gyakorlatban még nem terjedt el, nagyobb CSS állományok esetén az áttekinthetőséget jelentősen javítani tudja a CSS forrás (a C nyelven megszokotthoz hasonló) behúzása. Például:

CSS

```
body {...}
  p {...}
    p.balra {...}
  table {...}
    table#egyik td {...}
```

Természetesen ez a leírási mód nem teljesen mechanikus, többféleképpen is lehet a logikai csoportosítást szervezni.

Tematikus szétválasztás

Bonyolultabb esetekben szokás a CSS állományokat tipográfiai (a szövegekhez szorosan kapcsolódó) és dizájn (a menük, blokkok, stb. megjelenítéséhez kapcsolódó) szempontok mentén külön állományokra bontani.

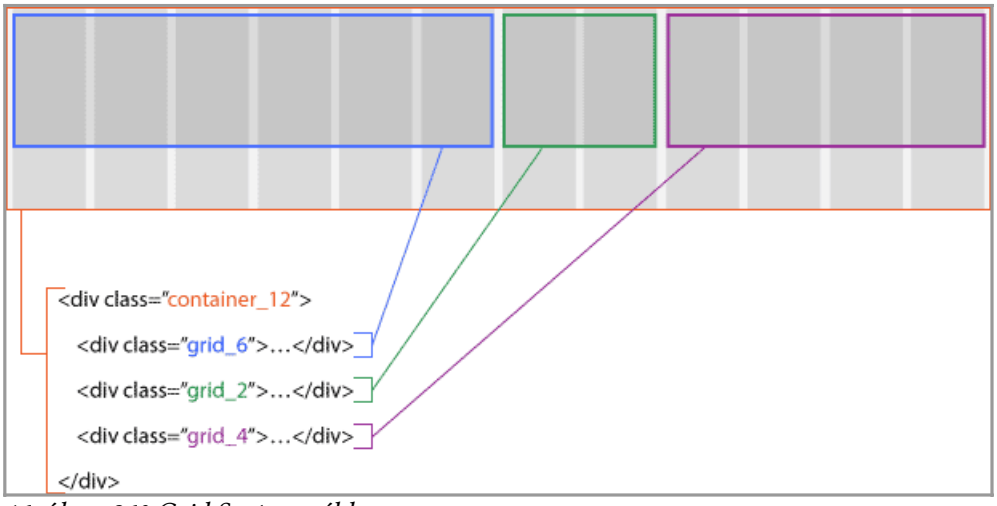
CSS keretrendszerek

Egyesek által felmagasztalt, mások által lenézett megoldás a CSS keretrendszerek alkalmazása. A támogatói az újrafelhasználható megoldásokkal indokolják döntéseiket. Sokszor tényleg egyszerűbb egy jól összeállított alapra építeni, mint egy új rendszert kiépíteni.

A sok lehetőség közül a *960 Grid System*[40] az egyik legígéretesebb megoldás. Jól mutatja, hogy a tervezés a kiadványszerkesztésnél szokásos gondolkozásmódra épít. Ezt demonstrálja a következő ábra[41] is:

40 http://960.gs/

41 Forrás: http://www.webdesignerdepot.com/2010/03/fight-div-itis-and-class-itis-with-the-960-grid-system/



16. ábra. 960 Grid System példa

2.2.5. Média típusok

Ma már egy weboldalt többféle környezetben használhatunk. Egy weboldalt megnézhetünk képernyőn, kézi-számítógépen, mobiltelefonon, vagy akár kinyomtatva is.

Célszerű minden médiához a hozzá legjobban illő formázást elkészíteni. Pl. nyomtatásban a talpas, képernyőn a talp nélküli betűket tartjuk olvashatóbbnak. Bizonyos elemeket (pl. menüket) felesleges kinyomtatni, míg weblapról kifelé mutató linkek `href` értékét nagyon hasznos megoldás.

Nézzünk példát a betűtípusok eltérő kezelésére. Nyomtatásban és képernyőn más-más méretet és betűtípust alkalmazhatunk a következő stílusokkal:

CSS
```
@media screen {
  p.test {font-family: verdana, sans-serif; font-size: 14px}
}

@media print {
  p.test {font-family: times, serif; font-size: 10px}
}

@media screen, print {
  p.test {font-weight: bold}
}
```

Másik gyakran alkalmazott lehetőség, amikor a HTML kódhoz külön-külön stíluslap állományokat készítünk:

HTML
```
<link rel="stylesheet" type="text/css"
  href="screen.css" media="screen">
<link rel="stylesheet" type="text/css"
  href="print.css" media="print">
```

Nézzünk néhány gondolatot Matkovský Péter *CSS lépésről-lépésre: nyomtatás* cikke[42] alapján.

Ne nyomtassunk ki felesleges részeket:

```
#header, #footer, #menu { display: none; }
```
CSS

Az oldal fix szélességű részei is gondot okozhatnak, használjunk %-os megadást:

```
#container, #main { width: 100%; }
```
CSS

Címsorok után soha, de `h1` előtt mindig legyen oldaltörés:

```
h1, h2, h3, h4, h5, h6 {page-break-after: avoid;}
h1 {page-break-before: always;}
```
CSS

Nyomtassunk ki a kifelé mutató linkeket:

```
a:after {content: " [" attr(href) "] ";}
```
CSS

Látszódjon a link a nyomtatásban (min egyfajta kiemelt szöveg):

```
a:link, a:visited, a:hover, a:active {
  text-decoration: underline; color: black;
}
```
CSS

Gondoljunk a színekre is:

```
body {color: black; background: white;}
```
CSS

A fontosabb médiatípusok:

`all`	minden eszköz
`aural`	felolvasó szoftver
`handheld`	kézi megjelenítő
`print`	lapozható (nyomtatás)
`projection`	vetítés (mint egy kirakati reklám, vagy az S5[43] módszer)
`screen`	képernyő

4. táblázat. Médiatípusok

42 http://weblabor.hu/cikkek/cssnyomtatas
43 http://meyerweb.com/eric/tools/s5/

2.2.6. Validátor

A CSS kód ellenőrzésére az online is használható W3C CSS Validation Service[44] egy nagyon jó eszköz. Arra azért figyelni kell, hogy csak érvényes HTML kóddal lehet a CSS érvényességét vizsgálni.

2.2.7. CSS 3

Az elmúlt években a HTML-hez hasonlóan a CSS területén sem volt sok változás. A CSS 2 óta megszoktuk, hogy a weboldalak szépek is lehetnek. Nagyon szépek. Hamarosan kialakultak olyan technikák, amelyek erősen építettek a grafikus szoftverek tudására és a kisebb-nagyobb egyedi trükkökre. Ma ezek (mint pl. a lekerekített sarkok, színátmenetek, árnyékok stb.) már annyira általánosan használtak, hogy a laikus internetező nem is sejti, mennyi kreativitás és munka szükséges ezek használatához.

Ma a CSS 3 területén is a változás korát éljük. Egyre-másra jelennek meg az újabb verziók, javítások a nagyobb böngészőkből. Sokat ígérnek, és egyre többet meg is valósítanak a CSS 3 lehetőségeiből. Egyelőre még csak játék, de hamarosan napi munkává válhat, ezért nem hagyhatjuk ki a témáink közül. A 2.12. fejezetben részletesen megnézünk néhány újdonságot.

2.3. Címsorok és formázásuk

A címek a `h1 h2 h3 h4 h5 h6` tagok segítségével adhatók meg. `h1` a legnagyobb (legfelsőbb szintű) címet jelenti, `h6` pedig a legkisebbet. (Általában egy oldalon 2-4 szintet indokolt alkalmazni, ekkor pl. a `h1`, `h2`, `h3` és `h4` alkalmazható.) A következő példa bemutatja a címek szokásos strukturálását.

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Nagy Gusztáv oldala</title>
  </head>
  <body>
    <h1>Nagy Gusztáv oldala</h1>
    <h2>Drupal 6 alapismeretek</h2>
    <p>Megjelent első könyvem, melyet szeretettel ajánlok ...</p>
    <h2>iuste.biz – Tisztességes vállalkozás mozgalom</h2>
    <p>Fél éve csatlakoztam a Tisztességes vállalkozás mozgalomhoz.</p>
    <p>Számomra ez nem csak egy plecsni. Jól kifejezi, milyen alapokra
        építem a vállalkozásomat. Számlát adok, szeretek adózni, és nem
        csapom be az ügyfeleimet.</p>
  </body>
</html>
```

44 http://jigsaw.w3.org/css-validator/

Általában egyszer alkalmazzuk a `h1`-es címet, ezután néhányszor a `h2`-est, szükség esetén a további szintekkel.

A 17. ábra jelzései jól mutatják, hogy szemantikus értelemben minden címhez tartozik egy tartalmi, kifejtő rész. A tartalmi részben a címek egy szinttel alacsonyabbak a főcímüknél, és mindig tartalmaznak folyó szöveget (ami már nem cím), pl. `p` elemeket.

17. ábra. Címek szemantikus használata

A címsorokhoz a böngésző alapértelmezetten térközöket is kapcsol, amit majd CSS segítségével módosíthatunk.

A következő CSS formázások nem csak címekre, hanem sokféle más tagra is alkalmazhatóak lesznek.

2.3.1. Háttér

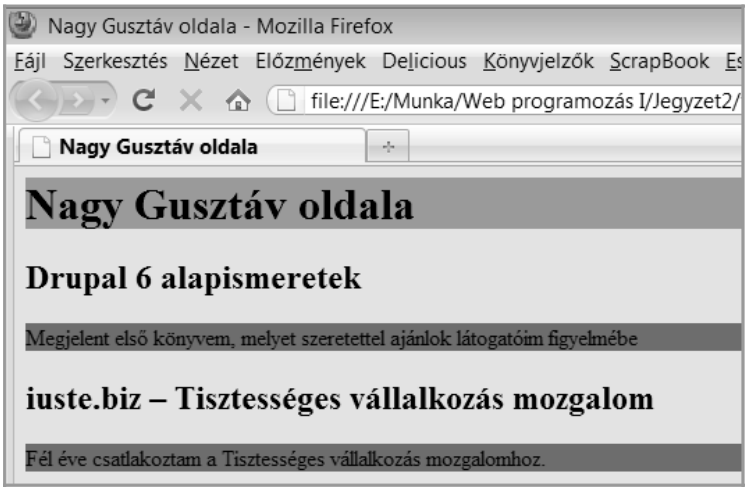
Egy elem hátterét háttérszín és háttérkép segítségével változatosan formázhatjuk. A háttérkép ismétlődhet is függőlegesen, vízszintesen, és lehet a böngészőablakhoz vagy a görgetett tartalomhoz ragasztott.

Háttérszín

A `background-color` tulajdonság segítségével meghatározhatjuk az elemek háttérszínét. A korábban megismert színmegadási módok közül itt is tetszőlegesen választhatunk. További lehetőség a `transparent` megadása, ami átlátszó hátteret fog jelenteni.

```
body {background-color: yellow}
h1 {background-color: #00ff00}
h2 {background-color: transparent}
p {background-color: rgb(250,0,255)}
```
CSS

Az eredmény:



18. ábra. Háttérszínek alkalmazása

Háttérkép

Az egyszínű háttér helyett akár látványos háttérképet is elhelyezhetünk az elemek hátteréként a `background-image` tulajdonság segítségével.

```
background-image: url('bgdesert.jpg')
```
CSS

Ismétlődés

A háttérkép alapértelmezés szerint kitapétázva jelenik meg, ha az elem mérete ezt szükségessé teszi. Természetesen ez megváltoztatható, mind vízszintes, mind függőleges irányban megtilthatjuk az ismétlődést. A következő példában csak *y* irányban ismétlődik a háttérkép:

```
background-image: url('bgdesert.jpg');
background-repeat: repeat-y;
```
CSS

A mindkét irányú ismétlődés kikapcsolásához a `no-repeat` értéket kell adnunk.

Pozíció

A háttérkép pozíciója is megváltoztatható: az alapértelmezett bal felső sarok helyett máshol is lehet a háttérkép.

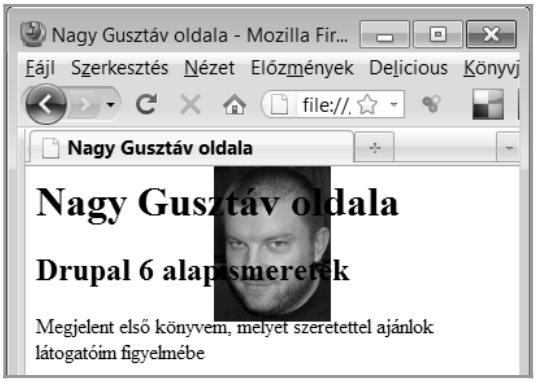
Ennek még akkor is van jelentősége, ha a teljes hátteret kitapétázva látjuk, ugyanis a kezdő tapétát van értelme máshol és máshol megadni.

Az érték megadásnál először a függőleges, majd a vízszintes pozíciót kell megadnunk. A 3-3 konstans (`top`, `center`, `bottom` és `left`, `center`, `right`) mellett százalékos és pixeles pozicionálás is lehetséges.

A következő példában az ablak felső részén, középen jelenik meg a kép.

CSS
```
body{
  background-image: url('nagygusztav.jpg');
  background-repeat: no-repeat;
  background-position: top center;
}
```

Az eredmény:



19. ábra. Háttérkép

Még a kettő közül az egyiket is elhagyhatjuk, ha az egyetlen megadott érték egyértelművé teszi a fejlesztő szándékát.

Háttérkép ragasztva

Elsősorban hosszabb görgethető tartalom esetén van jelentősége annak, hogy a háttérkép nem csak a görgetősávval együtt mozogva, hanem fixen pozicionálva is kérhető. Sok érdekes megoldás érhető el ezzel az apró trükkel.

CSS
```
background-image: url('nagygusztav.jpg');
background-repeat: no-repeat;
background-attachment: fixed;
```

Mindent bele

Ha többféle háttértulajdonságot is állítunk, tömörebb írásmódot eredményez az összevont `background` tulajdonság. Így egyszerre akár mindent is beállíthatunk.

```css
background: #00ff00 url('nagygusztav.jpg') no-repeat fixed center;
```

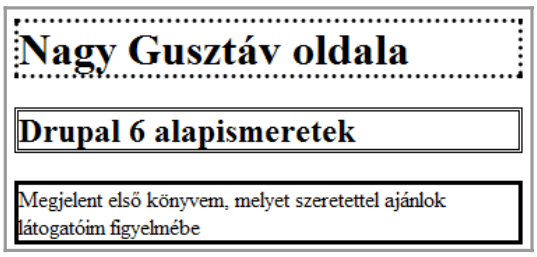
2.3.2. Szegélyek

A CSS `border` tulajdonságaival szegélyeket rajzolhatunk az elemek köré. A következő példa néhány fontosabb vonalstílust mutat be.

Nem minden böngésző az előírásoknak megfelelően rajzolja a szegélystílust.

```css
h1 {border-style: dotted}
h2 {border-style: double}
p {border-style: solid}
```

Az eredmény:



20. ábra. Segélyek használata

Az elemek szegélyét nem csak egységesen állíthatjuk be. Többféle lehetőségünk van az eltérő megadásra.

Egyrészt egyenként is állíthatjuk a szegélyek stílusát a `border-top-style`, `border-right-style`, `border-bottom-style` és `border-left-style` segítségével.

Másrészt a fenti megadásnál nem csak egy értéket, hanem 2, 3 vagy 4-et is hozzárendelhetünk a definícióhoz. Ennek logikáját a következő részben, a színekkel kapcsolatban fejtjük ki.

Szegélyszín

A színeket többféle módon közelíthetjük meg. Itt most a monitorok képmegjelenítésénél használatos, éppen ezért a webfejlesztésben is általános *RGB* megközelítést alkalmazzuk.

Az RGB színek három összetevőből állnak, mint *piros* (*red*), *zöld* (*green*), és *kék* (*blue*). Mindhárom érték egy bájton tárolt előjel nélküli számként határozható meg, vagyis értéke

`0` és `255` között (hexadecimálisan `00` és `FF` között) lehet. Ezzel a módszerrel tehát a 3 bájton ábrázolható 16 millió szín közül bármelyik kiválasztható.

Ezen kívül 16 alapszín névvel is rendelkezik (`aqua`, `black`, `blue`, `fuchsia`, `gray`, `green`, `lime`, `maroon`, `navy`, `olive`, `purple`, `red`, `silver`, `teal`, `white`, és `yellow`).

A szegélyek színét beállíthatjuk a `border-color` tulajdonsággal. Az alábbi példák minden oldal szegélyét meghatározzák.

Minden oldal egyforma színű:

```
border-color: #0000ff;
```
CSS

A fenti és lenti szegély `#ff0000` színű , míg a bal és jobb oldali `#0000ff` színű:

```
border-color: #ff0000 #0000ff;
```
CSS

A fenti szegély `#ff0000`, az oldalsók `#00ff00` és a lenti `#0000ff` színűek:

```
border-color: #ff0000 #00ff00 #0000ff;
```
CSS

A négy oldal az óramutató járásának megfelelően (kezdve a fenti szegéllyel) a felsorolt 4 értéket veszi fel:

```
border-color: #ff0000 #00ff00 #0000ff rgb(250,0,255)};
```
CSS

Végül érdemes megemlíteni, hogy ilyen speciális RGB színeknél lehetőség van a színkód „felezésére” is. Ennek feltétele, hogy mindhárom színkód dupla jelű legyen. Így az utolsó példánkat így is írhatnánk:

```
border-color: #f00 #0f0 #00f rgb(250,0,255)};
```
CSS

A szegély vastagsága

A szegély vastagságát a `border-width` tulajdonsággal állíthatjuk be. A következő példa folyamatos vonallal szegélyezi a bekezdést, de a szegély vastagságát megnöveli az alapértelmezett `1px`-hez képest:

```
p {
  border-style: solid;
  border-width: 15px
}
```
CSS

A négy oldal szegélyeit nem csak egyszerre, hanem akár külön-külön is lehet állítani, például a bal oldalit. Itt a szegély minden oldalon folyamatos lesz, de csak a bal oldalon lesz vastagabb:

```
p {
  border-style: solid;
  border-left-width: 15px
}
```
CSS

Közös deklaráció

Az eddigi szegély tulajdonságok (hasonlóan a korábbi tulajdonságcsoportokhoz) összevonhatók a `border`, vagy a `border-top`, `border-right`, `border-bottom` és a `border-left` tulajdonság segítségével. Például az alsó szegély több tulajdonságának beállítása:

```
p {border-bottom: medium solid #ff0000}
```
CSS

2.3.3. Térközök a szegélyen belül és kívül

A CSS szintaktikailag két nagyon hasonló tulajdonság-csoportot tartalmaz. A `margin` tulajdonsággal a szegélyen (`border`) kívüli, a `padding` tulajdonsággal pedig a tartalom és a szegély közötti belső margót lehet beállítani. A szintaktikai hasonlóság miatt ebben az alfejezetben csak a külső margó (`margin`) szintaxisa fog következni, de minden példa hasonlóan leírható lenne a belső margóra (`padding`) is.

Margók esetén is van lehetőségünk, hogy egyszerre mind a négy oldal értékét, akár különbözőre is beállíthassuk, vagy csak egyetlen oldalét változtassuk meg. A teljesség igénye nélkül következzék néhány példa.

Csak a bal oldali margót definiálja:

```
margin-left: 2cm;
```
CSS

Minden oldalét definiálja, de más-más értékkel:

```
margin: 2cm 5px 2em 5%;
```
CSS

Minden oldali margót egységesen nulláz:

```
margin: 0;
```
CSS

Az egyes böngészők bizonyos alapértelmezett margóbeállításokat eltérően értelmezhetnek. Az ebből eredő kellemetlenségek és bosszúságok elkerülése érdekében a szerző szokása, hogy egy új oldal CSS állományát valahogy így kezdi:

```
body, p, h1, h2, h3, table, tr, th, td, img {
  margin: 0;
  padding: 0;
}
```
CSS

Eric Meyer *Reset CSS*[45] megoldása még tovább megy. Mindenképpen érdemes átgondolni.

45 http://meyerweb.com/eric/tools/css/reset/

2.4. Az oldalszerkezet kialakítása

Ha a tervezés során kidolgoztuk az oldalunk funkcionális elemeit (5. és 6. ábra), akkor olyan oldalszerkezetet kell kialakítanunk, amely a terveknek megfelel. Ebben a fejezetben ennek az alapvetőbb eszközeit fogjuk bemutatni.

A div és span tagok

A `div` és `span` tagok célja az, hogy rá lehessen akasztani valamilyen CSS formázást, ha nincs más alkalmas tag (vagy az túl erőltetett lenne) a HTML kódban.

A két tag között egyetlen különbség, hogy a `div` blokk szintű (mint pl. a `p` vagy a `table`), míg a `span` soron belüli (mint pl. a `strong` vagy az `a`) elem. Az oldal kialakításakor a HTML 4-gyel bezárólag a `div` tag tömeges használata számított szakszerű megoldásnak. A HTML 5 térnyerésével erre már sok esetben nincs szükség, pontosabban van jobb (szemantikusabb) megoldásunk is. Erre a 2.4.7. fejezetben visszatérünk.

2.4.1. Méretek

Az elemek szélesség (`width`) és magasság (`height`) tulajdonsága segítségével az elem mérete befolyásolható.

Elsősorban doboz-jellegű elemeknél van értelme használni: például egy kép méretét, vagy a navigációs sáv szélességét gyakran állítjuk be ilyen módon.

A `width` és a `padding` megadása esetén a `width` a tartalom (content) tényleges szélességét jelenti. Nézzünk egy példát:

CSS
```
div#doboz {
  width: 200px;
  padding: 10px;
}
```

A szabvány szerint ebben az esetben a tényleges szélesség 200 pixel lesz. (Egyes korábbi böngészők itt más logikát alkalmaztak.)

2.4.2. Megjelenítés

A display tulajdonság egy elem más elemekhez viszonyított megjelenését befolyásolja. A tulajdonság három legalapvetőbb értéke a `block`, `inline` és `none`.

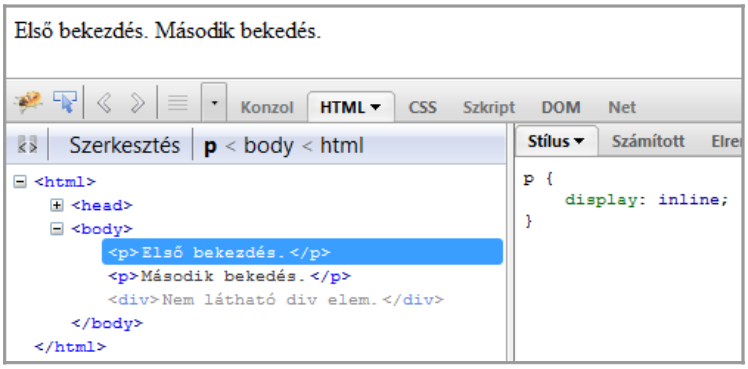
Egyes HTML elemeknek (pl. `h1`..`h6`, `p`, `ul`, `ol`, `div`) a `block` az alapértelmezése, míg másoknak (pl. `small`, `img`, `span`) az `inline`.

Nézzünk egy egyszerű példát a bekezdések `inline`-ként való használatára, és a `div` elemek eltüntetésére:

CSS
```
p {display: inline}
div {display: none}
```

HTML
```
<p>Első bekezdés.</p>
<p>Második bekedés.</p>
<div>Nem látható div elem.</div>
```

Az eredmény:



21. ábra. Soron belüli formázás

Egy másik – gyakorlati szempontból érdekes – példa, ami szerint az oldal főmenüjét felsorolt listával is szokás létrehozni, és a CSS szintjén `inline` listaelemekkel megoldani a menüpontok egymás mellé kerülését. Ezt a példát a 2.4.3. fejezetben még alaposabban megvizsgáljuk.

Felmerülhet a kérdés, hogy mi értelme van egy elem láthatóságát kikapcsolni. Sok érdekes eset közül csak egy faszerkezetű navigációs szituációt gondoljunk át. (A *Windows Intéző* bal oldali panelére érdemes gondolni.) Itt praktikus, ha egyes faelemeket, vagy akár nagyobb részeket is be lehet csukni, hogy a navigáció áttekinthetőbb legyen. Ezt JavaScripttel támogatva egyszerűen meg lehet tenni: kattintás hatására egy bizonyos elem display tulajdonságát kell `none` vagy `block` értékre váltani az eltüntetéshez vagy megjelenítéshez.

2.4.3. A lebegtetés

Eredetileg a képeknél alkalmazott, de az oldalkialakításnál is nélkülözhetetlen lehetőség a `float` tulajdonság alkalmazása. Ennek segítségével a soron belül megjelenő elemet ki tudjuk emelni a sor-folytonosságból, és a környező tartalom körül tudja futni az elemet.

Kép lebegtetése

Nézzünk egy képes példát először CSS nélkül. A kép a bekezdés belsejében, sor-folytonosan szerepel (pontosan úgy, mintha egy nagy méretű betű lenne a szövegben):

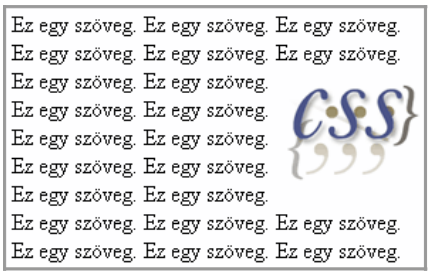
HTML

```
<p>
  Ez egy szöveg. Ez egy szöveg. Ez egy szöveg. Ez egy szöveg.
  Ez egy szöveg. Ez egy szöveg.
  <img src="logocss.gif" width="95" height="84">
  Ez egy szöveg. Ez egy szöveg. Ez egy szöveg. Ez egy szöveg.
  Ez egy szöveg. Ez egy szöveg. Ez egy szöveg. Ez egy szöveg.
...
```

Az eredmény:

22. ábra. Kép lebegtetés nélkül

Ha a kép elem megkapja a `float: right` formázását, a kép jobb oldalt jelenik meg, körbefuttatva a következő tartalommal:



23. ábra. Kép lebegtetve

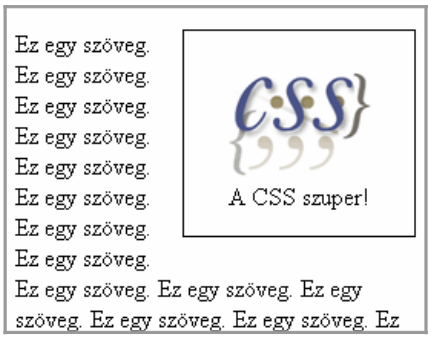
Ha a kép és a körbefuttatott tartalom közt nagyobb helyet szeretnénk kihagyni, akkor ezt a kép `margin` tulajdonságaival tehetjük meg.

Ha a képhez feliratot is szeretnénk társítani, akkor ez egy apró trükkel megvalósítható. A megoldás lényege, hogy nem közvetlenül a képet lebegtetjük, hanem egy `div` elem segítségével egységbe zárjuk a képet és a feliratát, és ezt a csoportosító `div` elemet lebegtetjük.

CSS
```
div {
  float:right;
  width:120px;
  margin:0 0 15px 20px;
  padding:15px;
  border:1px solid black;
  text-align:center;
}
```

HTML
```
<div>
  <img src="logocss.gif" width="95" height="84"><br>
  A CSS szuper!
</div>
<p>Ez egy szöveg. Ez egy szöveg. ...
```

Az eredmény:



24. ábra. Lebegtetés aláírással

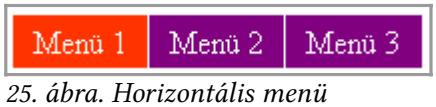
Az eddigiek alapján egy iniciálé lebegtetése egyszerű feladat. Tipp: a bekezdés első betűjét tegyük egy `span` elembe, hogy tudjunk formázást hozzákapcsolni.

Horizontális menü

Nézzük most meg a korábban beígért menüs példát. A menü egyszerű HTML lista:

HTML
```
<ul>
  <li><a href="#">Menü 1</a></li>
  <li><a href="#">Menü 2</a></li>
  <li><a href="#">Menü 3</a></li>
</ul>
```

A következő eredményt szeretnénk látni. (A képen nem látszik, hogy az egérkurzor a `Menü 1` felett van.)



25. ábra. Horizontális menü

Nézzük meg közelebbről a példa fontosabb pontjait.

```
ul {
  float:left;
  width:100%;
```
CSS

A menü az alatta levő szövegtől elválik, birtokolja a teljes vízszintes sávját.

```
  padding:0;
  margin:0;
  list-style-type:none;
}
```
CSS

Csak maga a szöveg felesleges térközök és jelölő nélkül.

```
a {
  float:left;
  width:6em;
```
CSS

Fix szélességű gombok érhetők el. Érdemes azonban vigyázni ezzel a módszerrel, mert ha a szöveg nem fér ki, a dizájn szétesik.

```
  text-decoration:none;
```
CSS

Ne legyen aláhúzva a link.

```
  color:white;
  background-color:purple;
  padding:0.2em 0.6em;
  border-right:1px solid white;
}
```
CSS

A menün kívüli háttérszín és a jobb oldali szegély színe itt meg kell, hogy egyezzen.

```
a:hover { background-color:#ff3300}
```
CSS

Az egérkurzorra színváltással reagál a menüpont. Érdemes megfigyelni, hogy a link a teljes li területét elfoglalja, ezért a teljes li elem linkként működik.

```
li { display:inline }
```
CSS

Ez a sor oldja meg, hogy a menüpontok egymás mellé kerüljenek.

Felesleges táblázatok nélküli oldalkialakítás

A webfejlesztés elmúlt éveiben sok zsákutcát megjártak a szakma művelői. A keretek indokolatlan használatánál talán már csak a táblázatos oldalkialakítás okozott több bonyodalmat és felesleges munkát.

Régi „jól bevált”, és sokak által még a mai napig is legjobbnak ítélt módszer a menük oldalra, fejléc felülre, lábléc alulra stb. pozicionálásához a táblázatok használata. A módszer rövidesen elkezdett burjánzani, megjelentek a 2-3-4 szinten egymásba ágyazott táblázatok, az egyesített cellák, a csak térköz kialakításához létrehozott sorok és oszlopok – vagy ami még ennél is rosszabb – a „távtartó gif-ek”. Aki már megpróbált egyszer egy ilyen szerkezetű oldalt megérteni, esetleg a dizájnt megváltoztatni, az lehet, hogy néhány ősz hajszál-

lal „gazdagodott”. A CSS 2-es verziója óta semmi szükség az ilyen elavult és értelmetlen technikákra.

Példaként nézzünk egy alap oldalelrendezést fejléccel, lábléccel és baloldali (például menü kialakítására alkalmas) sávval. A következőt szeretnénk elérni:

26. ábra. Tipikus két hasábos oldalszerkezet

A HTML szerkezet kialakításánál alapvetően a fentről lefelé, azon belül balról jobbra haladó tervezést érdemes követni. (Természetesen összetettebb esetben ez a sorrend nem ilyen egyszerű, és legtöbb esetben többféle megoldás is adható. Másrészt az is egy fontos szempont, hogy a lényegi információtól haladjunk a kevésbé lényeges felé.) Az oldal HTML szerkezete:

HTML

```
<body>
  <div class="container">
    <div class="header">
      <h1 class="header">Praesent...</h1>
    </div>
    <div class="left">
      <p>Phasellus wisi nulla...</p>
    </div>
    <div class="content">
      <h2>Aenean nummy odio orci</h2>
      <p>Phasellus wisi nulla...</p>
      <p>Adipiscing elit praesent...</p>
    </div>
    <div class="footer">Praesent...</div>
  </div>
</body>
```

A `container` nevet gyakran alkalmazzák az oldal fő tárolójának azonosításához. Érdemes még azt is megfigyelni, hogy a `left` és `content` doboz nincsenek egy közös dobozba összefogva, bár bizonyos esetekben ez is szükséges lehet.

CSS
```
div.container {
  width: 100%;
  margin: 0px;
  border: 1px solid gray;
  line-height: 150%;
}
```

A `container` szélessége alapvetően az egész oldal szélességét határozza meg. Látszik, hogy az oldal ki fogja tölteni a teljes ablakszélességet. A `margin` és `border` tulajdonságok már ismerősek, csak erre a dobozra lesznek hatással, míg a `line-height` öröklődni fog a tartalmazott dobozok irányába. Ehhez hasonlóan színeket, betűtípusokat szokás ilyen módon, egységesen megadni.

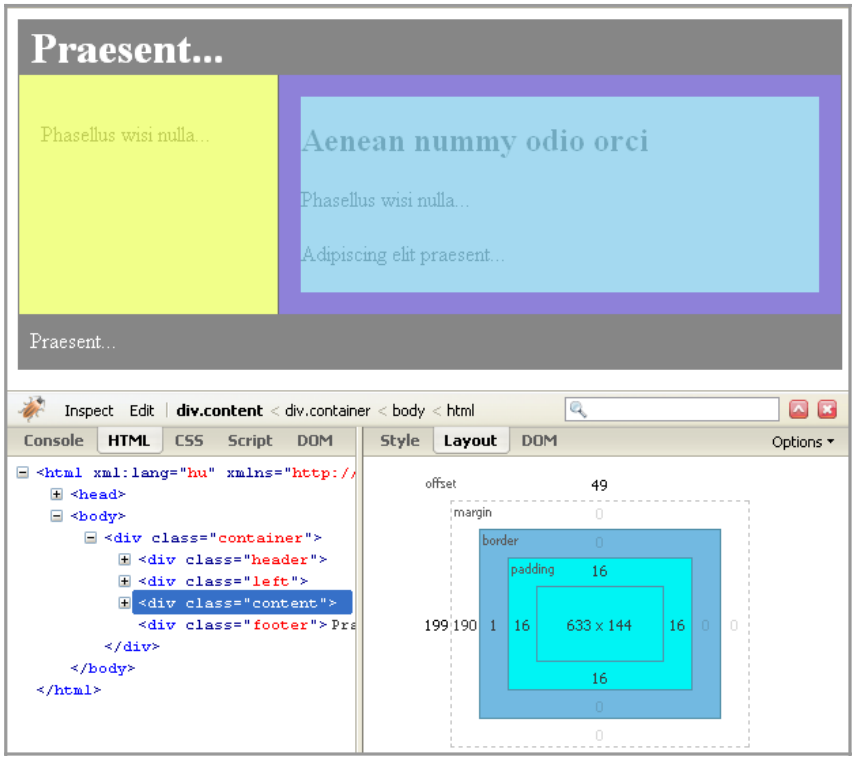
CSS
```
div.header, div.footer {
  padding: 0.5em;
  color: white;
  background-color: gray;
  clear: left;
}
```

Az utolsó sor megakadályozza, hogy a két doboz bal oldalán lebegő (`float`) elem legyen.

CSS
```
h1.header {
  padding: 0;
  margin: 0;
}
div.left {
  float: left;
  width: 160px;
  margin: 0;
  padding: 1em;
}
div.content {
  margin-left: 190px;
  border-left: 1px solid gray;
  padding: 1em;
}
```

A példa talán legérdekesebb részéhez értünk: a `left` és `content` egymás mellé helyezéséhez. Alap tulajdonságokkal ez a két doboz egymás alá kerülne, de a `left` elem `float` formázása lehetővé teszi, hogy a forráskódban utána következő `content` doboz ne alatta, hanem mellette (is) jelenjen meg. Ezen kívül még fontos, hogy a content elem bal margója (`margin-left`) is be lett állítva, így a `left` és `content` dobozok soha nem fogják egymást zavarni.

A megértéshez érdemes a működő példát a *FireBug* kiegészítő *Layout* nézetével is megnézni. Az ábra azt a pillanatot mutatja, amikor a `content` doboz van kijelölve (ez a bal alsó részen jól látszik). Megfigyelhető a 190 pixel széles, sárgával színezett bal oldali margin terület, és a konkrét számérték is a jobb alsó Layout elemben.



27. ábra. Az oldal felépítés vizsgálata Firebug-gal

A táblázatos oldalkialakítási módszer után a tisztán CSS-re építő megoldás logikája furcsa, nehézkes lehet. Hosszú távon azonban busásan meg fogja hálálni a befektetett energiát.

Az előző példánál maradva talán az szokta a legtöbb nehézséget okozni, hogy a táblázat celláinál megszokott háttérbeállítások itt nem úgy működnek, hiszen itt nem két egyforma magasságú celláról van szó. Ilyen jellegű probléma esetén van egy egyszerű megoldás: az egymás mellé kerülő dobozokat egy közös tartalmazó dobozba helyezzük, és a háttér ügyes beállításával el lehet azt a hatást érni, amit szeretnénk. Tehát nem az egymás melletti dobozok, hanem az őket közösen tartalmazó doboz hátterét kell beállítanunk.

Végül érdemes megjegyezni, hogy gyakran mindkét (vagy mindhárom) egymás mellé kerülő dobozt lebegtetjük. Ekkor nem szükséges a nagy margó alkalmazása. Hátránya viszont, hogy a dobozok előbb említett „közös hátterét” nehezebb formázni.

2.4.4. Pozicionálási sémák

A `position` tulajdonság segítségével az alapértelmezett statikus beállítás helyett relatív és abszolút pozicionálást is kérhetünk.

Relatív pozíció

Az elemeket alapértelmezett (`static`) helyzetüktől el tudjuk mozgatni a relatív pozicionálás segítségével vízszintes és függőleges irányban. A relatív eltolás mértékét a `left`, `right`, `top` és `bottom` tulajdonságokkal határozhatjuk meg. Az így eltolt elem „eredeti” helye üresen marad, oda más elem nem fog becsúszni. (Ez lényeges eltérés a lebegtetett elemekhez képest.)

A következő árnyékkal ellátott cím példa szemantikusan ugyan nem szerencsés, de jól szemlélteti a relatív pozicionálás lehetőségeit. A megoldás lényege, hogy a cím szövege két példányban szerepel a HTML forrásban, de vizuálisan majdnem egymást elfedve, és más színnel jelennek meg. Nézzük a példát:

HTML
```
<h1 class="shadow">Főcím</h1>
<h1 class="main">Főcím</h1>
<p>
  Bekezdés. Bekezdés.
  Bekezdés...
</p>
```

CSS
```
h1 {
  font-size:  2em;
}
h1.main
{
  color:     black;
  position:  relative;
  top:       -1.9em;
  left:      -0.1em;
}
h1.shadow {
  color:     #bbb;
}
```

Az eredmény:

28. ábra. Relatív pozicionálás

A megoldás hátránya is jól látszik a képen: a címet követő bekezdés nem követi közvetlenül a címet. Természetesen ez a hátrány további trükkökkel kiküszöbölhető.

Abszolút pozíció

A relatív pozíció esetén csak eltoltuk az elemet a helyéről, de az eltolás a környezetére nem volt más hatással. Ezzel szemben az abszolút módon pozicionált elem nem tart fenn egy területet. A megadott pozíciók itt a tartalmazó dobozon belüli abszolút pozíciók, és a következő elemek számára nem is léteznek.

Az előző példa így megoldható abszolút pozicionálással is a korábbi hátrány nélkül.

CSS
```
h1 {
  font-size: 2em;
}
h1.main {
  color:     black;
  top:       6px;
  left:      6px;
  position:  absolute;
}
h1.shadow {
  color:     #bbb;
  margin:    2px;
}
```

Az eredmény:

29. ábra. Abszolút pozicionálás

Érdemes még külön kiemelni, hogy az abszolút pozíció a szülők hierarchiájában a hozzá legközelebb eső, abszolút vagy relatív pozicionált elemhez képest pozicionál, ennek hiányában pedig a `body`-hoz. Tehát ha a szülője nem rendelkezik `position` tulajdonsággal, de annak szülője igen, akkor a szülő szülőjéhez képest számít a `top`, `left`, `right` vagy `bottom` eltolás. Emiatt nem ritka, hogy a kívánt szülőt a következő formázással látjuk el.

CSS
```
#valamelyikszulo {
  position: relative;
  top: 0;
  left: 0;
}
```

Őt nem akarjuk módosítani, de kiindulási alapként számítunk rá.

Fix pozíció

A fix pozíció az előző speciális esete. Az elem itt is kikerül a pozicionálási sémából, de itt nem a tartalmazó doboz, hanem a látótér (képernyő) a viszonyítási pont.

Nagyszerűen alkalmazható pl. a hagyományos keretek (frame-ek) kiváltására[46].

Az Internet Explorer korábbi verziói nem támogatják a fix pozicionálást.

2.4.5. Z-index

Sok esetben az elemek nem takarják egymást. Ha mégis, akkor alapértelmezetten a (HTML forráskódban) későbbi elem takarja el a korábbit. Ha ez nem megfelelő, akkor a `z-index` értékek meghatározásával manipulálhatjuk a vizuális takarást.

2.4.6. Beágyazott keretek

Az oldalszerkezet kialakításának általában kerülendő, de időnként hasznos megoldása az *inline frame*, vagyis az `iframe` tag használata. Az `iframe` segítségével egy téglalap alakú terület forráskódját külön állományban kezelhetjük az oldal egyéb részeitől.

```HTML
<iframe src="masikoldal.html" width="80%" height="110"></iframe>
```

A `masikoldal.html` önmagában is használható HTML oldal kell legyen.

Youtube[47] videót így ágyazhatunk a saját weboldalunkba:

```HTML
<iframe width="425" height="349"
   src="http://www.youtube.com/embed/Cy5ZZFIRY50"
   frameborder="0" allowfullscreen></iframe>
```

2.4.7. A HTML 5 újdonságai

A struktúra kialakítása területén igen sok újdonsággal készül a HTML 5-ös. Nézzünk meg[48] néhány fontosabb megoldást.

A HTML 4 használata esetén igen gyakori a következő ábrán is látható elrendezés. Jól megfigyelhető, hogy a `div` elemek, valamint a `class` és `id` attribútumok sora szükséges az oldal kialakításához.

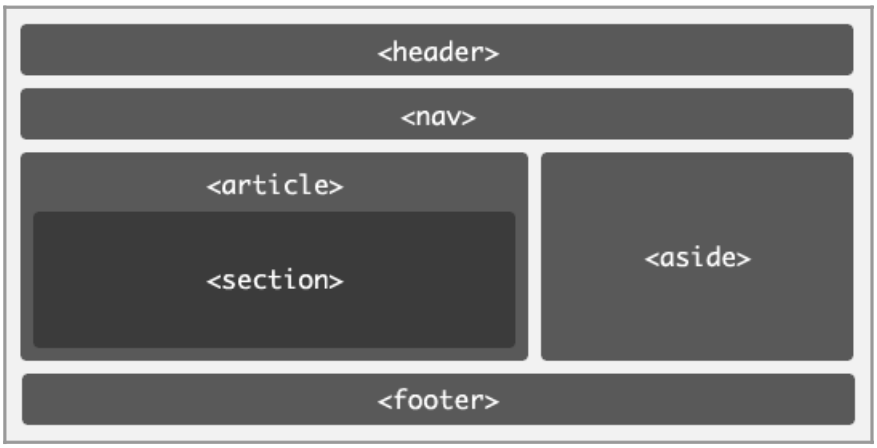
46 Egy nagyon szép megoldás: http://www.456bereastreet.com/archive/200609/css_frames_v2_fullheight/

47 http://www.youtube.com/

48 A fejezet több példája A List Apart: *Articles: A Preview of HTML 5* oldalról (http://www.alistapart.com/articles/previewofhtml5) származik.

30. ábra. Tipikus két hasábos oldalelrendezés

A HTML 5 használatával ezt a furcsa torlódást megszüntethetjük, ha a következőképpen tervezünk:



31. ábra. A div elemek helyett header, nav, section, article, aside és footer

Ez a következő HTML kóddal valósítható meg:

HTML 5

```
<body>
  <header>...</header>
  <nav>...</nav>
  <article>
    <section>
      ...
    </section>
  </article>
  <aside>...</aside>
  <footer>...</footer>
</body>
```

Szekció

A szekció (`section` elem) egy olyan strukturális elem, amely akár egymásba is ágyazva írhatja le a tartalmi struktúrát. Természetesen címek (`h1-h6` elemek) a szekció címét is megadhatják.

HTML 5

```
<section>
  <h1>Level 1</h1>
  <section>
    <h1>Level 2</h1>
    <section>
      <h1>Level 3</h1>
      ...
    </section>
  </section>
  ...
</section>
```

Fejrész

A `header` elem navigációs célokat szolgál. Alkalmazható egy szekció vagy az egész oldal (`body`) fejrészének leírásához. Tartalmazhat `h1..h6` elemeket is.

A teljes weboldal fejléce:

HTML 5

```
<body>
  <header>
    <h1>A Preview of HTML 5</h1>
    <p class="byline">By Lachlan Hunt</p>
  </header>
...
```

Egy kisebb egység fejrésze lehet:

HTML 5

```
<header>
  <h1>Example Blog</h1>
  <h2>Insert tag line here.</h2>
</header>
```

Lábrész

A `footer` elem a lábrész leírásához használható.

HTML 5

```
<footer>&copy; 2011 Example Inc.</footer>
```

Navigáció

A menüket `ul-li` listaként volt célszerű eddig is készíteni. Most már érdemes a `nav` elemet is alkalmazni:

HTML 5
```
<nav>
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Products</a></li>
    <li><a href="/services">Services</a></li>
    <li><a href="/about">About</a></li>
  </ul>
</nav>
```

Oldalsávok

A legtöbb honlap tartalmaz egy vagy több oldalsávot. A HTML 5 az `aside` elemet javasolja alkalmazni ilyen esetekben.

HTML 5
```
<aside>
  <h1>Archives</h1>
  <ul>
    <li><a href="/2007/09/">September 2007</a></li>
    <li><a href="/2007/08/">August 2007</a></li>
    <li><a href="/2007/07/">July 2007</a></li>
  </ul>
</aside>
```

A fő tartalom

A weboldalak fő tartalmi elemeinek (pl. blog esetén egy blogbejegyzés, az egyes kommentek) kialakításához a HTML 5 az `article` elemet javasolja.

HTML 5
```
<article id="comment-2">
  <header>
    <h4><a href="#comment-2" rel="bookmark">Comment #2</a>
        by <a href="http://example.com/">Jack O'Niell</a></h4>
    <p><time datetime="2007-08-29T13:58Z">August 29th,
       2007 at 13:58</time></p>
  </header>
  <p>That's another great article!</p>
</article>
```

Végezetül elmondhatjuk, hogy a HTML 5 komoly változásokat hoz ezen a területen.

2.5. Szövegek készítése

Bár egyre kevesebbet olvasunk a weben, a jól megfogalmazott és célszerűen megjelenített szövegre még mindig szükségünk van. A következőkben az 1.1.1. fejezet elveit tudjuk megvalósítani a HTML és CSS eszközeivel.

2.5.1. Bekezdések

Bekezdéseket a `p` taggal lehet definiálni:

```HTML
<p>Fél éve csatlakoztam a Tisztességes vállalkozás mozgalomhoz.</p>
<p>Számomra ez nem csak egy plecsni. Jól kifejezi, milyen alapokra
    építem a vállalkozásomat. Számlát adok, szeretek adózni, és nem
    csapom be az ügyfeleimet.</p>
```

Ugyanúgy, mint a papír alapú szövegszerkesztés esetén, itt sem soronként, hanem gondolati egységenként szokás a bekezdéseinket széttagolni.

A bekezdések hosszára a látogató nagyon érzékeny, így legalább 2-3 bekezdésenként érdemes egy címmel kiemelni a lényeget, illetve a bekezdés se legyen hosszabb 6-8 sornál.

A bekezdések tördeléséről általában nem nekünk kell gondoskodnunk, hanem hagyhatjuk a böngésző hatáskörében. A következő példában hiába szerepel a forráskódban az újsor és több szóköz karakter, a böngésző minden elválasztó karakter-sorozatot (az ún. *white space*-eket) egy szóközként értelmez és jelenít meg, a 17. ábra szerint.

```HTML
<p>Fél éve csatlakoztam
    a Tisztességes
    vállalkozás
    mozgalomhoz.</p>
<p>Számomra ez          nem csak egy plecsni. Jól kifejezi, milyen alapokra
    építem a vállalkozásomat. Számlát adok, szeretek adózni, és nem
    csapom be az ügyfeleimet.</p>
```

A tényleges tördelést mindig a böngésző ablakmérete és a benne levő szöveg határozza meg.

2.5.2. Sortörések

A `br` tag használható, ha szeretnénk új sort, de nem akarunk új bekezdést kezdeni. A `br` kikényszeríti a sortörést. Hibás gyakorlat viszont a bekezdésekre bontás helyett 2-2 `br` tagot írni.

Pl. a következő vers esetén szabad használni:

```HTML
<p>
  Mikor térsz már az eszedre, te Sándor?<br>
  Tivornya éjjeled és napod;<br>
  Az istenért! hisz az ördög elvisz,<br>
  Ha még soká így folytatod.
</p>
```

Érdemes megfigyelni, hogy az utolsó sor után már nincs szükség a `br` tagra.

32. ábra. Sortörés alkalmazása

A `br` elem üres, és nincs záró tagja sem.

2.5.3. Kiemelési lehetőségek

A HTML a története során felszedett olyan tagokat, amelyek ma már nem ajánlottak, sőt a HTML 5-be bele se kerültek. Ezért a hagyományos, kinézettel foglalkozó formázó tagok helyett a szemantikailag értelmezhető tagokat kell használnunk.

Szövegkiemelés

- `em` Kiemeli a szöveget
- `strong` Erősebb kiemelés
- `sub` Alsó indexet definiál
- `sup` Felső indexet definiál
- `ins` Beszúrt szöveget jelöl
- `del` Törölt szöveget jelöl

Az utolsó két tag használatára érdemes egy kis magyarázatot adni. Egy olyan weboldalnál, ahol a szövegen kisebb – de annál lényegesebb – változtatásokat ejtünk, nem fog feltűnni azoknak a látogatóknak, akik korábban már látták a tartalmat. De ha a `del` taggal jelöljük

a törölt, és szükség esetén `ins` taggal az újonnan beszúrt szöveget, akkor teljesen egyértelmű lesz minden olvasó számára. Nézzünk egy példát a `del` használatára:

```HTML
<h4>Időpont:</h4>
<p>2010. <del>június 21</del> június 20 - <del>július 2</del> július 1. (2 héten át hétfőtől péntekig) délelőtt 9.00-11.00 vagy délután 18.00-20.00 óráig.</p>
```

Az eredmény:

Időpont:

2010. ~~június 21~~ június 20 - ~~július 2~~ július 1. (2 héten át hétfőtől péntekig) délelőtt 9.00-11.00 vagy délután 18.00-20.00 óráig.

33 ábra. A `del` tag használata

Számítógép kimenet

- code Forráskódot definiál
- pre Előformázott szöveget definiál: az elválasztó (white space) karaktereket nem a HTML-ben szokásos, hanem direkt módon értelmezi

Idézet, kiemelés és definíciós tagok

- `abbr` Rövidítést definiál
- `acronym` Mozaikszót definiál
- `address` Cím elemet definiál
- `bdo` Szöveg írásirányt határoz meg
- `blockquote` Hosszú (akár több bekezdéses) idézetet jelöl
- `q` Rövid (bekezdésen belüli) idézetet jelöl
- `cite` Idézetet jelöl
- `dfn` Definíciót jelöl

Néhány egyszerűbb példa a tagok használatához:

```HTML
<address>
  Nagy Gusztáv<br>
  Kecskemét
</address>

<abbr title=
  "United Nations">UN</abbr>
```

```
<acronym title="World Wide
  Web">WWW</acronym>

<bdo dir="rtl">
  Ez a szöveg jobbról
  olvasható
</bdo>

<blockquote>
  Az ilyen hosszabb idézeteket
  a böngésző térközökkel is
  kiemeli
</blockquote>
```

2.5.4. Szövegek megjelenítése

A CSS szövegtulajdonságai segítségével a szövegek vizuális megjelenítését lehet testre szabni.

A szöveg színe

A szöveg színét a `color` tulajdonság határozza meg:

```
h1 {color: #00ff00}
h2 {color: #dda0dd}
p {color: rgb(0,0,255)}
```
CSS

Betűtávolság

A betűk közötti távolság a `letter-spacing` tulajdonsággal módosítható. A pozitív érték ritkítást eredményez, a negatív pedig sűrítést:

```
h1 {letter-spacing: -3px}
h4 {letter-spacing: 0.5cm}
```
CSS

Szótávolság

A betűtávolsághoz hasonló módon adható meg a `word-spacing` tulajdonság segítségével.

A szöveg igazítása

A következő példa bemutatja, hogyan lehet a szöveget balra, középre, jobbra vagy sorkizártan igazítani:

```
h1 {text-align: center}
h2 {text-align: left}
h3 {text-align: right}
p {text-align: justify}
```
CSS

Természetesen az igazítás meghatározásának csak blokk szintű elem esetén van értelme. Az alapértelmezett a balra igazítás.

A sorkizárt igazítással érdemes csínján bánni, mivel az automatikus elválasztás hiánya miatt a szóközök nagyon csúnyán megnyúlhatnak. Keskeny szövegblokk esetén különösen kerülendő.

A középre zárt igazítást is csak indokolt esetekben szabad alkalmazni. Pl. címek, ábra feliratok esetén.

A szöveg dekorációja

A következő példa bemutatja, hogy hogyan lehet a szövegünket fölé húzott, áthúzott, alá-húzott vonallal megjeleníteni:

```css
h1 {text-decoration: overline}
h2 {text-decoration: line-through}
h3 {text-decoration: underline}
a  {text-decoration: none}
```

A szöveg behúzása

A bekezdés első sorát a következő módon tudjuk `1 cm`-el behúzni:

```css
p {text-indent: 1cm}
```

Kis-, és nagybetű formázás

A következő példa a nagybetűs, kisbetűs, majd kis-kapitális formázást mutatja be:

```css
p.uppercase {text-transform: uppercase}
p.lowercase {text-transform: lowercase}
p.capitalize {text-transform: capitalize}
```

Elválasztó karakterek értelmezése

Ahogy már a korábbiakban volt szó róla, a HTML oldalakon az ún. elválasztó karakterek (white-spaces) számától és típusától (szóköz, tabulátor vagy újsor) függetlenül mindig egy szóköznek számítanak. Ez alól – HTML szintjén – az egyetlen kivétel a `pre` tag alkalmazásával érhető el.

Praktikus, hogy az elválasztó karakterek értelmezését ennél finomabban is tudjuk szabályozni a CSS `white-space` tulajdonság segítségével. Az alapértelmezett kikapcsolt (`none`) mellett lehetőség van az előformázott értelmezés (`pre`) és a több sorra tördelést megtiltó (`nowrap`) beállításra.

Az eddigi lehetőségeken túl az írásirányt is beállíthatjuk a `direction` tulajdonsággal.

Betűtípus megadása

A következő példában a `h3` címeknek és a bekezdéseknek más-más betűtípust használunk.

```css
h3 {font-family: times}
p {font-family: courier}
p.sansserif {font-family: sans-serif}
```

A betűtípusok közül érdemes általánosan használt, a képernyőn jól olvasható típusokat megadni. Kevésbé elterjedt betűtípus esetén érdemes elterjedtebb alternatívákat is felsorolni, hogy a böngésző nem ismert típus esetén is tudjon hasonlót választani.

Betűméret

A betűk méretét a `font-size` tulajdonsággal állíthatjuk be. A megadásnál kötelező mértékegységet is alkalmazni, és általában érdemes relatív megadási módot alkalmazni, hogy a felhasználó a saját igényei szerint tudja azt kicsinyíteni vagy nagyítani.

```css
p {font-size: 1,1em}
h1 {font-size: 130%}
```

A számszerű megadáson túl szövegesen is megadhatjuk a méretet: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`, `smaller` és `larger`. Az így megadott méretek is relatívak.

Betűstílus

A dőlt betűstílus alkalmazására láthatunk egy példát:

```css
p {font-style: italic}
```

Betűvastagság

A betűk vastagsága a `font-weight` tulajdonsággal befolyásolható. Szöveges konstansok (`normal`, `bold`, `bolder` és `lighter`) mellett `100, 200, … 900` értékek használhatók.

```css
p.normal {font-weight: normal}
p.vastagabb {font-weight: bold}
p.legvastagabb {font-weight: 900}
```

Betűformázások összevonása

Az összes betűformázás összevonható akár egyetlen deklarációvá:

```css
p {font: italic small-caps 900 12px arial}
```

Első gyermek

A `:first-child` látszólagos osztály egy adott elem első gyermekét képes kiválasztani. Nézzük a következő példát:

```css
a:first-child {
  text-decoration:none
}
```

Ennek hatására minden első gyermekként előforduló a elemre érvényes lesz a fenti formázás. Például:

```HTML
<p>Nézze meg a <a href="http://www.gamf.hu">GAMF</a> és
  az <a href="http://informatika.gamf.hu"> Informatika tanszék</a>
  honlapját!</p>
```

Ha az adott elemben (itt p) nem link (a) az első gyermek elem, akkor az adott környezetben ennek a formázásnak semmilyen hatása nem lesz. A következő verzióban a strong elem átvette az első gyermek szerepét, ezért a formázásnak itt nem lesz hatása:

```HTML
<p>
  <strong>Nézze meg</strong> a
  <a href="http://www.gamf.hu">GAMF</a>
...
```

Első betű és első sor

A :first-letter kiválasztó segítségével az első betű, míg a :first-line segítségével az első sor kaphat speciális formázást. Nézzünk példaként egy iniciálét, valamint egy félkövér első sort:

```HTML
<p>
  Tetszőleges szövegű bekezdés
</p>
```

```CSS
p:first-letter {
  color: #ff0000;
  font-size:300%;
  vertical-align: top;
  float: left;
}
p:first-line {
  font-variant: small-caps;
}
```

2.6. Linkek

A HTML linkeket (hivatkozásokat) használ arra, hogy az oldalunkhoz más tartalmakat kapcsolhassunk.

2.6.1. HTML szintaxis

A link (a) tag és a href tulajdonság

Egy link hivatkozni tud egy tetszőleges webes erőforrásra, pl. egy HTML oldalra, egy képre, zenére stb.

A link szintaxisa a következő:

```HTML
<a href="url">Megjelenő szöveg</a>
```

A `href` tulajdonsághoz rendelt érték határozza meg, hogy a böngésző hogyan reagáljon a link kiválasztására. (Itt nem csak kattintás jöhet szóba, hiszen billentyűzetről is lehet linket kiválasztani a TAB segítségével, és akár gyorsbillentyű (`accesskey`) is rendelhető egy linkhez.) A kezdő és a záró tag közötti szöveg (vagy akár bonyolultabb tartalom) lesz kattintható, és (alapértelmezetten) kék színnel aláhúzott link.

A következő példa egy linket definiál a Weblabor honlapjára:

```HTML
<p>A <a href="http://weblabor.hu/">Weblabor</a> honlapja.</p>
```

A `target` tulajdonság

Alapértelmezetten egy link kiválasztása esetén az új oldal az aktuális helyett jelenik meg. Ez azonban módosítható.

A következő link egy új ablakban nyitja meg a Weblabor honlapját:

```HTML
<a href="http://weblabor.hu/" target="_blank">Weblabor</a>
```

A `name` tulajdonság

Egy hivatkozás alapértelmezetten a HTML oldal tetejét jelenti. Néha azonban praktikus, ha egy oldalon belül is pontosítani tudjuk a link célját. Erre szolgál ez a tulajdonság.

A következő módon tudunk létrehozni egy ugrási célpontot:

```HTML
<a name="term107">PHP</a>
```

Ez az elem vizuálisan nem fog megjelenni (pl. aláhúzással), hiszen ez a kapcsolat végpontja lehet, és nem a kezdőpontja.

Ha erre a pontra akarunk hivatkozni egy linkkel, akkor a következő módon kell alkalmaznunk:

```HTML
<a href="http://weblabor.hu/szojegyzek#term107">
```

Természetesen akár az oldalon belül is lehet ilyen linkeket alkalmazni:

```HTML
<a href="#tipp">Ugrás</a>
```

Hasznos tippek

Ha egy alkönyvtárra akarunk hivatkozni, az URL végére tegyük ki a `/` karaktert. Így a webkiszolgáló egy felesleges próbálkozást megspórolva gyorsabban fogja megtalálni a keresett könyvtár tartalmát. (Először a könyvtárnevet állománynévként próbálja értelmezni, ha nincs mögötte / karakter.)

Hosszú oldal esetén tegyünk az elejére egy tartalomjegyzéket a fontosabb fejezetekre mutató belső linkekkel. Ezen kívül szokás az oldal végén (vagy akár több helyen is) az oldal tetejére (elejére) mutató linket alkalmazni (erre a `href="#"` használható). Ennek a megoldásnak az a hátránya, hogy a vissza gomb hatására is az oldalon belül fog a felhasználó ugrálni.

2.6.2. Linkek formázása

Valószínűleg legismertebb látszólagos osztályok a linkekhez kapcsolódnak. Hagyományosan más-más színnel szokás jelezni az egyszerű linkeket, a már meglátogatott linkeket, az éppen az egérkurzor alatt levő és a kattintás közben levő linkeket:

CSS
```
a:link    {color: #FF0000}
a:visited {color: #00FF00}
a:hover   {color: #FF00FF}
a:active  {color: #0000FF}
```

Ma talán a `:hover` látszólagos kiválasztóval találkozhatunk a legtöbbször, és nem is csupán a betűszínek, hanem akár komolyabb viselkedés is megvalósítható vele.

Kimenő linkek formázása

Ma már egyre több weboldal segít abban, hogy a weboldalról kifelé mutató linkek könnyen felismerhetőek legyenek a mögötte található kis kép alapján. Ekkor természetesen nem célszerű a HTML `img` tagot használni. Nézzünk egy konkrét példát:

HTML
```
<a title="" class="ext" target="_blank">Kecskeméti Főiskola</a>
```

CSS
```
a.ext {
  background: url("nyil.gif") no-repeat right center;
  padding-right: 15px;
}
```

2.7. Multimédia

A HTML története a szövegek strukturált megjelenítésénél indult. Ma már nem olvasni, hanem hallani és látni szeretnénk a weben. Ezért a képek mellett az audio és videó tartalmak beillesztése mindennapos feladatnak számít.

2.7.1. Képek

A HTML nyelvben az `img` tag segítségével tudunk képeket definiálni. Ez az elem üres, és nincs záró tagja sem (hasonlóan a `br` elemhez).

A kép megjelenítéséhez először is meg kell adni a `src` tulajdonságot, vagyis a kép állomány helyét és nevét. A szintaxis a következő:

```HTML
<img src="url">
```

Az `url` lehet abszolút vagy relatív megadású is. Abszolút:

```HTML
<img src="http://valami.hu/kep.gif">
```

Abszolút, a `base href`-ben megadott helytől, ennek hiányában a domain gyökerében keres:

```HTML
<img src="/kep.gif">
```

Relatív, a HTML állomány könyvtárában keresi:

```HTML
<img src="kep.gif">
```

Az `alt` tulajdonság

Az `alt` tulajdonság alternatív szöveg definiálását teszi lehetővé. A szabvány szerint tehát ennek a szövegnek akkor kell a böngészőben láthatóvá válni, ha a kép valamilyen oknál fogva nem jeleníthető meg (pl. még nem töltődött le, nem érhető el, vagy eleve ki van kapcsolva a képek letöltése).

A Microsoft Internet Explorer akkor is megjeleníti ezt a szöveget, ha az egérkurzort visszük a kép fölé, de ez eltér a HTML eredeti céljától. Erre a szabvány a `title` tulajdonságot írja elő.

```HTML
<img src="hajo.gif" alt="1 árbócos vitorlás hajó">
```

Méret megadása

A böngésző ugyan a méret megadása nélkül is meg tudja jeleníteni a képet, mégis célszerű a `width` és `height` tulajdonságokat megadni. Lassú kapcsolat vagy sok nagy kép esetén kimondottan zavaró lehet, amikor egy újabb kép letöltődésekor – az ekkor ismertté vált méret adatok alapján – a félig megjelent oldal „ugrál”.

```HTML
<img src="hajo.gif" width="144" height="50">
```

Természetesen a kép fizikai méretétől eltérő méretek is megadhatók, ekkor kicsinyítés, nagyítás, vagy akár torzítás is lehet az eredmény. Ezt azonban nem szabad a képszerkesző program alternatívájaként használni. Rendkívül illetlen magatartás a 10-20 KB-os bélyegképek helyett több MB-os képeket letöltetni a látogatóval, majd 100px-es méretben megjeleníteni.

Kép használata linkként

Link aktív felületéhez szöveg mellett vagy helyett kép is rendelhető. Erre mutat példát a következő kód:

CSS

```
<p>
   <a href="lastpage.htm">
      <img border="0" src="kovetkezo.gif"
        width="65" height="38">
   </a>
</p>
```

2.7.2. Flash lejátszó beágyazása

Multimédia tartalmak közzétételére legegyszerűbb megoldás, ha elhelyezzük a webszerveren, és valahol az oldalon egy linket készítünk rá. Ekkor a látogató kattintására az állomány teljes egészében letöltődik a saját gépére, és elindul a zene- vagy médialejátszó. Ennek néhány előnye mellett inkább a hátrányát szoktuk érezni: sokat kell várni a lejátszás elindulására, és lehet, hogy nem is akarjuk az egészet megnézni.

Ma már teljesen általános megoldás, hogy a legtöbb látogató számítógépén rendelkezésre álló Flash beépülő alkalmazást használjuk erre az esetre. Ekkor a weboldalunkhoz egy ingyenesen használható, vagy megvásárolt Flash alapú lejátszót kell illesztenünk a weboldalunkba.

Az `object` és `param` tagok

Az `object` tag beágyazott objektumot definiál. Az `object` tipikusan multimédia tartalom beágyazására használható.

A `param` tag segítségével futásidejű paramétereket adhatunk át a beágyazott objektumnak.

Az `object` tagok akár egymásba is ágyazhatók, a belső `object` tag akkor lesz figyelembe véve, ha a külső `object` nem futtatható. (Pl. letiltott Flash lejátszó esetén egy állókép jelenjen meg.

Flowplayer

A méltán népszerű Flowplayer[49] lejátszó dokumentációja alapján pl. a következőhöz hasonló kód használható:

49 http://flowplayer.org/

```HTML
<object
    width="500" height="375"
    type="application/x-shockwave-flash"
    id="swf12970890251"
    data="/flowplayer3/flowplayer-3.2.5.swf">
  <param name="swliveconnect" value="default">
  <param name="play" value="true">
  <param name="loop" value="true">
  <param name="menu" value="false">
  <param name="quality" value="autohigh">
  <param name="scale" value="showall">
  <param name="align" value="l">
  <param name="salign" value="tl">
  <param name="wmode" value="opaque">
  <param name="bgcolor" value="#FFFFFF">
  <param name="version" value="9">
  <param name="allowfullscreen" value="true">
  <param name="allowscriptaccess" value="sameDomain">
  <param name="base" value="/files/">
  <param name="src" value="/flowplayer3/flowplayer-3.2.5.swf">
  <param name="width" value="500">
  <param name="height" value="375">
  ...
</object>
```

Természetesen nem mindig szükséges ennyi paraméter átadására. A pontos jelentés pedig az adott lejátszó dokumentációjából olvasható ki.

A kódban látszó redundanciák oka a böngészők különbözőségében keresendő.

JW Player

A szintén népszerű JW Player[50] a következő beágyazó kódot javasolja:

```HTML
<embed
  flashvars="file=/data/bbb.mp4&autostart=true"
  allowfullscreen="true"
  allowscripaccess="always"
  id="player1"
  name="player1"
  src="player.swf"
  width="480"
  height="270"
/>
```

Videómegosztó kód beágyazása

Igen gyakori, hogy nem a saját tárhelyünkön, hanem valamelyik videómegosztó oldalon elhelyezett videókat szeretnénk az oldalunkba beágyazni. Példaként nézzük meg a *Ustream*[51] által nyújtott kód felépítését:

50 http://www.longtailvideo.com/support/jw-player/jw-player-for-flash-v5

51 http://www.ustream.tv/

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" width="480"
height="296" id="utv32777" name="utv_n_585188"><param name="flashvars"
value="loc=
%2F&amp;autoplay=false&amp;vid=12499218&amp;locale=en_US&amp;hasticket=fals
e&amp;id=12499218&amp;v3=1"><param name="allowfullscreen"
value="true"><param name="allowscriptaccess" value="always"><param
name="src" value="http://www.ustream.tv/flash/viewer.swf"><embed
flashvars="loc=
%2F&amp;autoplay=false&amp;vid=12499218&amp;locale=en_US&amp;hasticket=fals
e&amp;id=12499218&amp;v3=1" width="480" height="296" allowfullscreen="true"
allowscriptaccess="always" id="utv32777" name="utv_n_585188"
src="http://www.ustream.tv/flash/viewer.swf" type="application/x-shockwave-
flash"></object>
```

HTML

A népszerű *Youtube*[52] videómegosztó egy teljesen más megoldást nyújt a számunkra. Erre láthattunk egy példát a 2.4.6. fejezetben.

2.7.3. HTML 5 újdonságok

Manapság igen gyakori a weboldalakban elhelyezett videó vagy hang állomány. A letöltés nélküli lejátszáshoz eddig elengedhetetlenek voltak a Flash alapú lejátszó alkalmazások. A HTML 5 célul tűzte ki a média állományok natív támogatását.

A teljesség igénye nélkül néhány példa.

Videó beillesztése:

```
<video src="video.ogv" controls poster="poster.jpg" width="320"
height="240">
    <a href="video.ogv">Download movie</a>
</video>
```

HTML 5

Audio beillesztése:

```
<audio src="music.oga" controls>
    <a href="music.oga">Download song</a>
</audio>
```

HTML 5

Videó beágyazása vezérlő gombokkal:

```
<video src="video.ogg" id="video"></video>
<script>
  var video = document.getElementById("video");
</script>
<p><button type="button" onclick="video.play();">Play</button>
   <button type="button" onclick="video.pause();">Pause</button>
   <button type="button" onclick="video.currentTime = 0;">
   << Rewind</button>
```

HTML 5

A HTML 5 multimédia szolgáltatásai még nagyon képlékenyek. De a böngészőgyártók folyamatos vetélkedése szemlátomást jót tesz a fejlődésnek.

52 http://youtube.com/

2.8. Listák

A listákat igen széles körben alkalmazhatjuk a weblapok struktúrájának kialakításánál.

2.8.1. HTML szintaxis

A HTML támogatja a számozott, felsorolt és definíció-listák létrehozását.

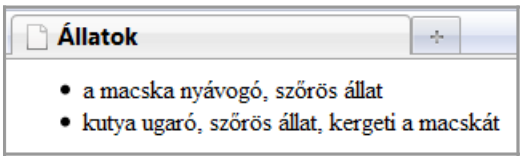
Felsorolt lista

A felsorolt listák olyan elemeket tartalmaznak, amelyeket nem kell számozással azonosítanunk, ehelyett egy felsorolási szimbólum (alapértelmezetten egy fekete kör) jelzi vizuálisan a listaelemek kezdetét. Tehát ilyenkor a felsorolt elemek sorrendje nem lényeges, nem hordoz információt.

A felsorolt lista az `ul` elemmel írható le, a lista elem pedig az `li` elemmel.

HTML
```
<ul>
  <li>a macska nyávogó, szőrös állat</li>
  <li>kutya ugaró, szőrös állat, kergeti a macskát</li>
</ul>
```

Az eredmény:



34. ábra. Felsorolt lista

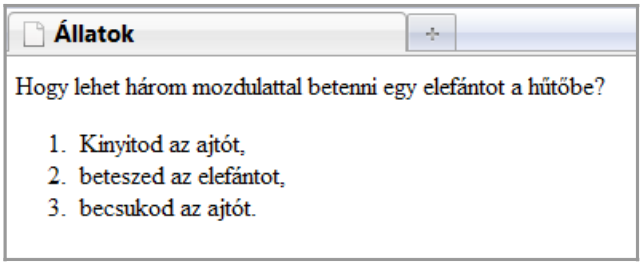
Számozott lista

A számozott listák elemei (többnyire) számmal azonosítottak.

A számozott listát `ol` taggal kell létrehozni, a lista elemek az előzőhöz hasonlóan `li`-vel definiálhatók.

HTML
```
<p>Hogy lehet három mozdulattal betenni egy elefántot a hűtőbe?</p>
<ol>
  <li>Kinyitod az ajtót,</li>
  <li>beteszed az elefántot,</li>
  <li>becsukod az ajtót.</li>
</ol>
```

Az eredmény:



35. ábra. Számozott lista

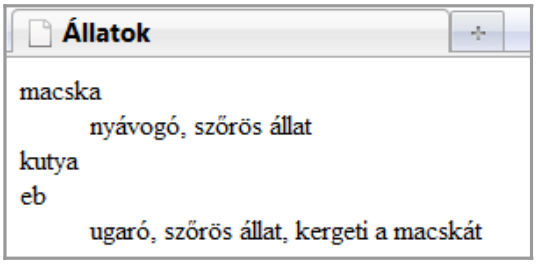
Definíciós lista

A definíciós lista nem csupán egy felsorolás, hanem a definiált elemek és a definícióik sorozata. Egy definícióhoz akár több elem is megadható.

A definíciós lista `dl` taggal, a definiált elem `dt` taggal, maga a definíció pedig `dd` taggal definiálható.

HTML
```
<dl>
  <dt>macska</dt>
  <dd>nyávogó, szőrös állat</dd>
  <dt>kutya</dt>
  <dt>eb</dt>
  <dd>ugaró, szőrös állat, kergeti a macskát</dd>
</dl>
```

Az eredmény:



36. ábra. Definíciós lista

További példák

Felsorolt és számozott listák esetén a szimbólumok választhatók. A következő lista az angol abc kisbetűit írja ki:

HTML
```
<ol type="a">
 <li>alma</li>
 <li>banán</li>
 <li>citrom</li>
</ol>
```

A számozásnál megadhatjuk a kezdő értéket is a `start` attribútum segítségével.

```
<ol start="4">
```
HTML

A listák egymásba is ágyazhatók, így hierarchikus szerkezeteket is kialakíthatunk.

```
<ul type="square">
  <li>Kávé</li>
  <li>Tea
    <ul>
      <li>Fekete tea</li>
      <li>Zöld tea</li>
    </ul>
  </li>
  <li>Tej</li>
</ul>
```
HTML

Érdemes azonban megfigyelni, hogy a belső lista nem két `li` elem között, hanem egy `li` elem belsejében, a `/li` tag előtt szerepel. Vagyis az egymásba ágyazott tagok lánca:

`ul` – `li` – `ul` – `li`.

2.8.2. Listák formázása

A CSS lista formázásai segítségével a lista előtti térközök, a lista elem vagy lista kép állítható be.

Lista jelölők

Felsorolt listák esetén az egyes listaelemek sorrendisége nem jelent információt, ezért minden listaelem előtt ugyanazt a jelölőt szokás alkalmazni. Érdemes megfigyelni a következő példában, hogy a formázás a lista (`ul`) és nem az egyes listaelemek (`li`) tekintetében történik.

```
ul.disc   {list-style-type: disc}
ul.circle {list-style-type: circle}
ul.square {list-style-type: square}
ul.none   {list-style-type: none}
```
CSS

Számozott listák esetén jóval több lehetőségünk van, bár ezek közül is csak néhány tartozik a gyakrabban használtak közé.

```
ol.decimal {list-style-type: decimal}
ol.lroman  {list-style-type: lower-roman}
ol.uroman  {list-style-type: upper-roman}
ol.lalpha  {list-style-type: lower-alpha}
ol.ualpha  {list-style-type: upper-alpha}
```
CSS

Az előre adott lehetőségeket magunk is tovább bővíthetjük azzal, hogy tetszőleges képet alkalmazhatunk listajelölőnek:

```
list-style-image: url('arrow.gif')
```
CSS

A listajelölő pozíciója

A listajelölő a szöveg belsejében, vagy az előtt is szerepelhet:

```css
ul.inside {
  list-style-position: inside
}
ul.outside {
  list-style-position: outside
}
```

Közös deklaráció

A listaelemek esetén is használható a már jól megszokott, rövidebb írásmódot lehetővé tevő közös deklaráció. Erre is láthatunk egy példát:

```css
list-style: square inside url('arrow.gif')
```

2.9. Táblázatok

A táblázatok segítségével mátrix-szerű, kétdimenziós adathalmazt tudunk megjeleníteni. Érdemes azonban figyelemmel lenni arra, hogy nagy méretű táblázatok kezelése még nagy monitoron se egyszerű, ezért érdemes tesztelni kisebb felbontás esetén is. Nagy táblázatok helyett érdemesebb lehet több kisebb táblázatot, vagy felsorolást használni.

2.9.1. HTML szintaxis

Táblázatokat a `table` tag segítségével lehet létrehozni. Egy tábla sorokat tartalmaz (`tr` tag), és minden sor cellákat (`th` vagy `td` tag). A tábla cellái szöveget, képet, bekezdést, listát, űrlapokat, újabb táblázatokat is tartalmazhatnak.

Nézzünk egy egyszerű, `2x2` cellás táblázatot:

```html
<table border="1">
  <tr>
    <td>1. sor, 1. cella</td>
    <td>1. sor, 2. cella </td>
  </tr>
  <tr>
    <td>2. sor, 1. cella </td>
    <td>2. sor, 2. cella </td>
  </tr>
</table>
```

Az eredmény:

1. sor, 1. cella	1. sor, 2. cella
2. sor, 1. cella	2. sor, 2. cella

37. ábra. Táblázat

Táblázat szegélyek

Alapértelmezetten a táblázatok szegélyek nélkül jelennek meg. Van azonban lehetőség arra, hogy szegélyek is megjelenjenek az oldalon: az előző példában is látható `border` tulajdonsággal lehet beállítani a szegély szélességét. (A szám képpontban értendő.)

Hamarosan látni fogjuk, hogy CSS-ben ennél jobb lehetőségeink lesznek a kinézet befolyásolására.

Táblázat fejléc

A táblázat első (néhány) sorába, első (néhány) oszlopába szokás fejléc információkat elhelyezni. Ez magyarázza az alatta vagy mellette található értékek jelentését. Ebben az esetben az első sort celláit a `th` tagokkal kell megadni:

HTML

```
<table border="1">
  <tr>
    <th>1. fejléc</th>
    <th>2. fejléc</th>
  </tr>
  <tr>
    <td>1. sor, 1. cella</td>
    <td>1. sor, 2. cella</td>
  </tr>
  <tr>
    <td>2. sor, 1. cella</td>
    <td>2. sor, 2. cella</td>
  </tr>
</table>
```

Az eredmény:

1. fejléc	2. fejléc
1. sor, 1. cella	1. sor, 2. cella
2. sor, 1. cella	2. sor, 2. cella

38. ábra. Táblázat fejléc

Táblázat cím

Méltatlanul keveset használt elem a `caption`, amivel a táblázat címét tudjuk korrekt módon megadni. Lehetőség van annak kiválasztására is, hogy a négy lehetséges oldal közül hol jelenjen meg a cím. A `caption` elemnek meg kell előznie a sorokat:

```HTML
<table border="1">
  <caption>Táblázat címe</caption>
  <tr>
    <th>1. fejléc</th>
    <th>2. fejléc</th>
  </tr>
  ...
```

Az eredmény:

Táblázat címe

1. fejléc	2. fejléc
1. sor, 1. cella	1. sor, 2. cella
2. sor, 1. cella	2. sor, 2. cella

39. ábra. Táblázat címmel

Cellák összevonása

A `colspan` és `rowspan` tulajdonságok segítségével cellákat lehet egyesíteni:

```HTML
<table border="1">
  <tr>
    <th>Név</th>
    <th colspan="2">Telefon</th>
  </tr>
  ...
```

Az eredmény:

Név	Telefon	
Bill Gates	555 77 854	555 77 855

40. ábra. Összevont cellák

Tippek

A `thead`, `tbody` és `tfoot` elemek a céljuk szerint nagyon hasznos elemek lennének, de sajnos a böngészők igen változó módon támogatják őket, ezért a gyakorlatban nem is szokás alkalmazni.

A táblázat cellák további táblázatokat is tartalmazhatnak, amivel összetett szerkezetek alakíthatók ki.

2.9.2. Táblázatok formázása

A táblázatok formázására használhatóak a korábban már megismert szegély, háttér, térköz formázásokat igen gyakran használjuk. Itt csak néhány specialitásra térünk ki.

Szegélyek

A HTML `border` tulajdonság helyett általában a következő módon szoktuk a szegélyt formázni:

```css
table, th, td {
  border: 1px solid black;
}
```

Ezen kívül szinte minden esetben alkalmazzuk a `table` tag `border-collapse` attribútumát, amivel a szokatlan kinézetű (pl. 40. ábra) önálló cellaszegélyek helyett a szövegszerkesztőkben megszokott kinézetet írhatjuk elő.

```css
table {
  border-collapse:collapse;
}
```

Méretek

A táblázatok alapértelmezés szerint csak azt a minimális helyet foglalják el, amely szükséges a tartalom megjelenítéséhez. A könnyebb olvashatóság érdekében a cellákon szokás belső szegélyt (`padding`) beállítani, illetve egyéb módokon is befolyásolni a táblázat méretét. Néhány szokásos megoldás:

```css
table { width:100%; }
th { height:50px; }
td {line-height: 24px; }
```

Cella igazítás

A cellák tartalmát mind vízszintes, mind függőleges irányban igazíthatjuk. Érdemes azonban egységes kinézetet tervezni, és nem a „tarkaságra” törekedni.

Nézzünk itt is néhány megoldást:

```css
td {
  text-align:right;
  vertical-align:bottom;
}
```

Zebra táblák

Az alap formázások lehetőségein túl további segítséget adhatunk a látogatóknak, hogy a táblázatot könnyebben tudják olvasni. Ennek igen elterjedt példája a zebra táblák alkalmazása.

Nézzünk egy rövid példát[53].

HTML

```
<table>
  <tr class="paratlan">
    td>1. sor, 1. cella</td>
    <td>1. sor, 2. cella</td>
  </tr>
  <tr class="paros">
    td>2. sor, 1. cella</td>
    <td>2. sor, 2. cella</td>
  </tr>
  <tr class="paratlan">
    td>3. sor, 1. cella</td>
    <td>3. sor, 2. cella</td>
  </tr>
  <tr class="paros">
    td>4. sor, 1. cella</td>
    <td>4. sor, 2. cella</td>
  </tr>
</table>
```

CSS

```
table {
  border-collapse:collapse;
  border: 1px solid #aaa;
}
td {
  border: 1px solid #aaa;
  padding: 2px;
}
tr.paros td {
  background-color: #eee;
}
tr.paratlan  td {
  background-color: #fff;
}
```

Az eredmény:

53 Forrás: A List Apart Magazine, http://www.alistapart.com/articles/zebratables/

1. sor, 1. cella	1. sor, 2. cella
2. sor, 1. cella	2. sor, 2. cella
3. sor, 1. cella	3. sor, 2. cella
4. sor, 1. cella	4. sor, 2. cella

41. ábra. Zebra tábla formázás

2.10. Űrlapok

Az űrlapokat arra használhatjuk, hogy különböző módokon lehetőséget adjunk a látogatónak visszajelzésre, vagyis adatok megadására.

2.10.1. HTML szintaxis

A `form` elem más űrlap elemeket tartalmaz. Ezek az űrlap elemek teszik lehetővé az adatbevitelt.

A leggyakrabban használt elem az `input`. A `type` tulajdonságával állítható be, hogy pontosan milyen adatbeviteli módot szeretnénk.

Szöveges mezők

A szöveges mezők lehetővé teszik, hogy betűkből, számokból, írásjelekből álló karaktersorozatot lehessen begépelni.

```HTML
<form>
  Vezetéknév:
  <input type="text" name="vezeteknev">
  <br>
  Keresztnév:
  <input type="text" name="keresztnev">
</form>
```

Az eredmény:

Vezetéknév:

Keresztnév:

42. ábra. Űrlap

Az űrlap elemek soron belüli (`inline`) elemek. Ha egymás alá akarjuk tenni, akkor azt valamilyen módon külön meg kell adnunk a HTML, vagy még szebb, ha a CSS segítségével.

Érdemes megfigyelni, hogy maga a `form` elem vizuálisan nem látható, csak a benne elhelyezett elemek. Ha (ahogy ebben a példában is) nem adjuk meg a mezők szélességét, a legtöbb böngésző alapértelmezetten 20 karakter szélesen jeleníti meg. Ez azonban nem korlátozza a ténylegesen begépelhető szöveg hosszát.

Jelszavak begépeléséhez `password` típusú (`type="password"`) beviteli mezőt szokás létrehozni. Ez viselkedésében csak annyiban tér el a `text` típustól, hogy a begépelt szöveg helyett * (egyes böngészőkben vagy operációs rendszerek alatt •) karakterek jelennek meg, és vágólapra nem lehet kimásolni a tartalmát.

Az elrejtés csak a képernyőre vonatkozik, a hálózaton egyszerű adatként utazik a jelszó.

Rádiógombok

A rádiógombokat akkor használhatjuk, ha a látogatónak néhány választható elem közül kell választási lehetőséget adni. Az elemek közül mindig csak az egyik lehet aktív. Érdemes megfigyelni a következő listában, hogy a `name` tulajdonság azonossága rendeli a rádiógombokat logikailag egy csoporttá, vagyis egymást kizáró választási lehetőségekké. Tehát ebből a szempontból sem a vizuális elrendezés számít.

HTML

```
<form>
  <input type="radio" name="nem"
    value="no">Nő</input>
  <br>
  <input type="radio" name="nem"
    value="ferfi">férfi</input>
</form>
```

Az eredmény:

43. ábra. Rádiógombok

Az ábrán látszik, hogy alapértelmezetten egyik elem sincs kiválasztva. Ha valamelyik választási lehetőséget alapértelmezettnek tekintjük, akkor ezzel könnyíthetjük is az űrlap kitöltését:

HTML

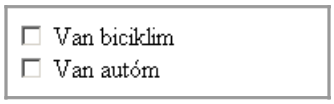
```
<input type="radio" name="nem" value="no" checked>Nő</input>
```

Jelölőnégyzetek

A jelölőnégyzetek arra szolgálnak, hogy a felhasználó egy vagy több elemet is ki tudjon választani a rendelkezésre álló lehetőségek közül. Más megközelítésben úgy is lehetne fogalmazni, hogy egy jelölőnégyzet ki- vagy bekapcsolt állapotban lehet, függetlenül más beviteli elemektől.

```HTML
<form>
  <input type="checkbox" name="bicikli">
  Van biciklim
  <br>
  <input type="checkbox" name="auto">
  Van autóm
</form>
```

Az eredmény:



44. ábra. Jelölőnégyzetek

Itt is van lehetőségünk az alapértelmezetten ki nem választott állapot helyett bejelölve megjeleníteni a jelölőnégyzetet:

```HTML
<input type="checkbox" checked name="bicikli">
```

A `label` elem

Érdemes megfigyelni, hogy rádiógomb és jelölőnégyzet esetén a kattintható terület a kör, illetve négyzet alakú területre korlátozódik. Az elemek mellett megjelenő szövegek a böngésző számára függetlenek a jelölő elemektől, csupán a vizuális helyzet jelzi nekünk az összefüggést.

A `label` elem használatával ez a függetlenség megszüntethető: a szöveget a jelölőelemmel aktív kapcsolatba hozhatjuk. Ez azt jelenti, hogy lehet kattintani a szövegre is.

```HTML
<form>
  <input type="radio" name="nem" value="no" id="no">
  <label for="no">Nő</label>
  <br>
  <input type="radio" name="nem" value="ferfi" id="ferfi">
  <label for="ferfi">férfi</label>
</form>
```

Az előző verzióhoz képest fontos kiegészítés, hogy a `value` mellett az `id` is megkapta az azonosító szövegeket, mivel a `label` tag `for` tulajdonsága az `id` alapján azonosítja az elemeket.

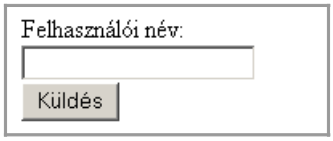
Űrlap adatok elküldése

Az esetek jelentős részében a felhasználó azért tölt ki egy űrlapot, hogy adatokat tudjon a szerver felé küldeni valamilyen hatás érdekében.

Az eddigi példákból két fontos rész kimaradt. Először is a felhasználó számára szokás biztosítani egy küldés (vagy valami hasonló feliratú) gombot, hogy erre kattintva kezdeményezhesse az adatok elküldését. Másrészt a `form` tag – egyébként kötelezően kitöltendő – `action` tulajdonsága határozza meg, hogy melyik oldalnak kell a kérést feldolgoznia.

```HTML
<form name="input" action="feldolgoz.php" method="get">
  Felhasználói név:
  <input type="text" name="nev">
  <input type="submit" value="Küldés">
</form>
```

Az eredmény:



45. ábra. Nyomógomb

Ha a felhasználó begépeli a nevet, majd kattint a `Küldés` gombra, akkor a szöveg továbbításra kerül a `feldolgoz.php` számára.

Nyomógombokat a `button` taggal is létre lehet hozni.

Lenyíló lista

Bizonyos esetekben rádiógombok helyett célszerű inkább a lenyíló listák alkalmazása. (Elsősorban terjedelmi, áttekinthetőségi okokra kell gondolni.)

Két tag használatára lesz mindenképpen szükség: először is a `select` tag adja meg a lista kereteit, míg az `option` tagok a választható elemeket.

```HTML
<form>
  <select name="autok">
    <option value="audi">Audi</option>
    <option value="fiat">Fiat</option>
    <option value="skoda">Skoda</option>
    <option value="suzuki" selected>Suzuki</option>
  </select>
</form>
```

A `selected` tulajdonság lehetővé teszi az alapértelmezett elem kijelölését. Ha ez nem szerepel a forrásban, akkor az első elem lesz a kiválasztott betöltődéskor.

Lehetőség van olyan lista létrehozására is, ahol egyszerre akár több elem is kiválasztható:

```HTML
<select name="autok" multiple>
```

A `size` attribútum segítségével a lenyíló lista több soros listává alakítható.

```HTML
<select name="autok" size="5">
```

Végül megemlítjük a hosszabb, csoportosítással áttekinthetővé tehető listák esetén hasznos `optgroup` tagot. Az alábbi példa a négy választási lehetőséget két vizuális csoportba sorolja.

```HTML
<select>
  <optgroup label="Olasz autók">
    <option value="fiat">Fiat</option>
    <option value="ferrari">Ferrari</option>
  </optgroup>
  <optgroup label="Német autók">
    <option value="vw">VW</option>
    <option value="audi">Audi</option>
  </optgroup>
</select>
```

Több soros szöveges mezők

Lehetőség van hosszabb szöveg begépelését, szerkesztését lehetővé tevő beviteli mezőt is létrehozni. Erre szolgál a `textarea` elem. A következő példán a méretek megadásán túl a kezdőszöveg is definiált:

```HTML
<textarea rows="10" cols="30">Kezdőszöveg</textarea>
```

Mezőcsoportok

Hosszabb űrlapok esetén hasznos lehetőség, hogy az egyes mezőket vizuálisan csoportosíthatjuk a `fieldset`, és akár címmel is elláthatjuk a `legend` taggal. Nézzünk egy rövid példát egyetlen csoporttal. (A gyakorlatban egymás után több csoportot szoktunk létrehozni.)

```HTML
<form>
  <fieldset>
    <legend>Személyes</legend>
    Név: <input type="text" size="30"><br>
    E-mail cím: <input type="text" size="30"><br>
    Születési év: <input type="text" size="10">
  </fieldset>
</form>
```

Az eredmény:

46. ábra. Mezőcsoport hosszabb űrlapokhoz

2.10.2. Új lehetőségek a HTML 5-ben

A HTML 5 sok hiányt pótol ezen a rendkívül fontossá vált területen. A HTML 5 nélkül különböző kliens és szerver oldali megoldásokkal próbáltuk pótolni azokat a hiányosságokat,

amelyek a jelenlegi űrlapok használata esetén mutatkoznak. De nézzük meg, melyek azok a lehetőségek, amelyek könnyíthetik a munkánkat.

Kezdjük először az újfajta űrlap elemekkel.

Szám mező

Gyakran van szükségünk arra, hogy számadatot kérjünk be a látogatótól. A következő példa mutatja, hogy milyen lehetőségeink vannak.

```
<input type="number" min="10" max="20" step="2">
```
HTML 5

Jól látszik, hogy az intervallum határait és a lépésközt is megadhatjuk.

47. ábra. Szám mező

Csúszka használata

Akár számadatok esetén, akár más szituációban érdekes megoldás lehet a csúszka használata.

```
<input type="range" min="0" max="10" step="2" value="6">
```
HTML 5

Arra azonban érdemes figyelni, hogy itt a csúszka számszerű értéke nem jelenik meg közvetlenül. JavaScript segítségével viszont összeköthetjük a csúszka működését más űrlap elemekkel.

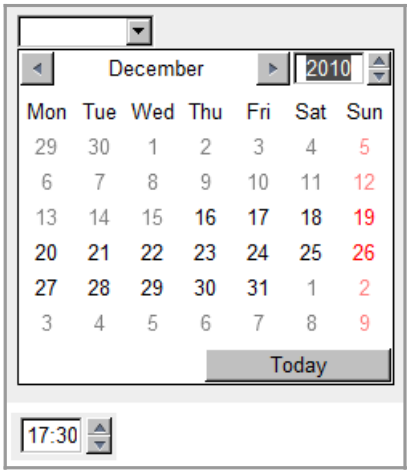
48. ábra. Csúszka

Dátum/idő megadása

Nagyon körülményes megoldásokat kellett alkalmaznunk a dátum és időpont megadásokhoz a korábbiakban. De erre is van egy nagyszerű megoldásunk:

```
<input type="date" … >
<input type="time" … >
```
HTML 5

Az eredmény:



49. ábra. Date és time bemenet

Ezen kívül használható a `datetime`, `month` és `week` típusú mező is.

Szín megadása

A szín megadására is jó lehetőségünk van:

```
<input type="color">
```

HTML 5

További lehetőségek

A következő példák jól mutatják az igényeket. A böngészők megoldása már nem ilyen egyértelmű.

```
<input type="tel">
<input type="email">
<input type="url">
```

HTML 5

Automatikus fókusz

Automatikusan megkapja a fókuszt a következő elem:

```
<input type="text" autofocus … >
```

HTML 5

Validálás

A JavaScript használata helyett további lehetőségeket is kaptunk.

Kötelező kitölteni a következő mezőt:

```
<input type="text" … required >
```

HTML 5

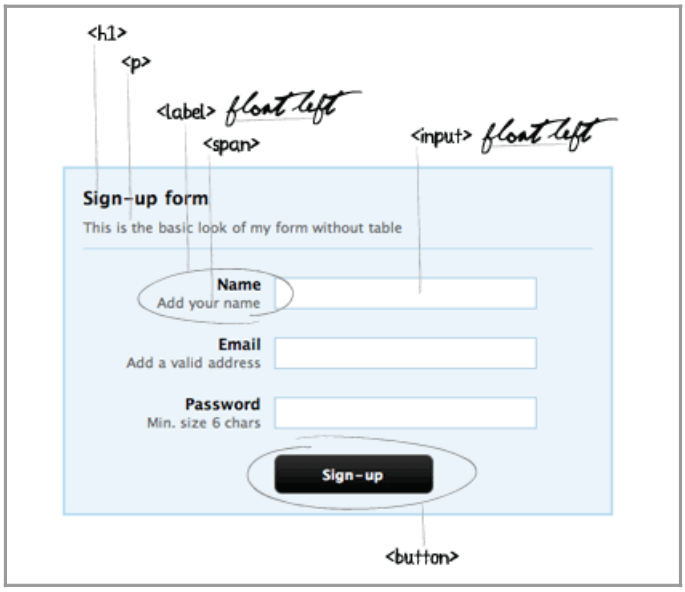
Lehetőség van reguláris kifejezések használatára is:

```
<input type="text" … pattern="[az]{3}[0-9]{3}" >
```

HTML 5

2.10.3. Űrlapok formázása

Az űrlapok formázásánál nem az új technikai lehetőségeket, hanem a szemléletmódot érdemes hangsúlyozni. Nézzünk meg egy rövid, de szemléletes példát *Antonio Lupetti* tollából[54]. Ezt szeretnénk elérni:



50. ábra. Form dizájn

A HTML kód:

54 http://woork.blogspot.com/2008/06/clean-and-pure-css-form-design.html

HTML

```
<div id="stylized" class="myform">
  <form id="form" name="form" method="post" action="index.html">
    <h1>Sign-up form</h1>
    <p>This is the basic look of my form without table</p>
    <label>Name
      <span class="small">Add your name</span>
    </label>
    <input type="text" name="name" id="name">
    <label>Email
      <span class="small">Add a valid address</span>
    </label>
    <input type="text" name="email" id="email">
    <label>Password
      <span class="small">Min. size 6 chars</span>
    </label>
    <input type="text" name="password" id="password">
    <button type="submit">Sign-up</button>
    <div class="spacer"></div>
  </form>
</div>
```

A CSS kód:

CSS

```
body {
  font-family:"Lucida Grande", "Lucida Sans Unicode", Verdana,
    Arial, Helvetica, sans-serif;
  font-size:12px;
}

p, h1, form, button {
  border:0;
  margin:0;
  padding:0;
}

.spacer {
  clear:both;
  height:1px;
}

.myform {
  margin:0 auto;
  width:400px;
  padding:14px;
}

#stylized {
  border:solid 2px #b7ddf2;
  background:#ebf4fb;
}

#stylized h1 {
  font-size:14px;
  font-weight:bold;
  margin-bottom:8px;
}
```

```
#stylized p {
  font-size:11px;
  color:#666666;
  margin-bottom:20px;
  border-bottom:solid 1px #b7ddf2;
  padding-bottom:10px;
}

#stylized label {
  display:block;
  font-weight:bold;
  text-align:right;
  width:140px;
  float:left;
}

#stylized .small {
  color:#666666;
  display:block;
  font-size:11px;
  font-weight:normal;
  text-align:right;
  width:140px;
}

#stylized input {
  float:left;
  font-size:12px;
  padding:4px 2px;
  border:solid 1px #aacfe4;
  width:200px;
  margin:2px 0 20px 10px;
}

#stylized button{
  clear:both;
  margin-left:150px;
  width:125px;
  height:31px;
  background:#666666 url(img/button.png) no-repeat;
  text-align:center;
  line-height:31px;
  color:#FFFFFF;
  font-size:11px;
  font-weight:bold;
}
```

Természetesen sok más megoldás is hasznos és megfelelően látványos lehet. A témában érdemes még tutorialokat[55] megismerni, és hasznos lehet a *Wufoo Form Gallery*[56] megismerése is. Végül a pForm[57] automata generálási lehetőségeit is érdemes kipróbálni.

55 Pl. http://www.smashingmagazine.com/2006/11/11/css-based-forms-modern-solutions/
56 http://wufoo.com/gallery/
57 http://www.phpform.org/

2.11. Fejrész

A `head` elem olyan információkat tartalmaz, amelyek a HTML dokumentum egészére vonatkoznak. A fejrész a `base`, `link`, `meta`, `title`, `style` és `script` tagokat tartalmazhatja.

A `base` elem az oldal relatív linkjeinek működésére lesz hatással. Ha nincs a `base` elem `href` tulajdonsága megjelölve, akkor a relatív hivatkozások az aktuális HTML fájl könyvtárához képest lesznek értelmezve. A `href` megadása esetén a megadott URL lesz az indulási hely.

A `meta` elem meta-információt definiál az oldalhoz. Néhány példa:

```
<meta name="keywords" content="HTML, CSS, XHTML, JavaScript">
```
HTML

A kereső robotok számára fontos kulcsszavakat emeli ki.

```
<meta name="description" content="Web programozás jegyzet">
```
HTML

Rövid összefoglaló az oldaltól.

Ezeknek az információknak korábban igen nagy jelentőségük volt a keresőoptimalizálás szempontjából. A keresőoldalak egy korábbi generációja (a Google előtti korszak) lényegében a `meta` tagokra építette a keresési szolgáltatását. Ezt a fejlesztők egy idő után arra használták, hogy az oldalak hamis képet mutassanak magukról, és ezzel jobb helyet érjenek el a találati listákon. A Google nagy ötlete volt, hogy nem erre a könnyen manipulálható információra, hanem a bejövő linkekre épít. Így ma a `meta` tagok jelentősége jóval kisebb, mint ahogy azt sokan gondolják.

```
<meta name="revised" content="Hege Refsnes, 6/10/99">
```
HTML

Az utolsó módosítás napja.

Végül a karakterkódolás megadása:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```
HTML

2.12. A CSS3 néhány megoldása

A CSS3 fejlődése jelentősen felgyorsult. De így is nehéz megjósolni, hogy mikor használhatjuk ezeket a megoldásokat teljes értékűen.

Most csak néhány alapvetőbb példát fogunk megnézni.

Lekerekített sarkok

```
-moz-border-radius: 5px;
-webkit-border-radius: 5px;
```
CSS 3

Szegély képpel

```
border-image: url('image-border.png')29 29 29 29 round round;
border: double orange 1em;
```
CSS 3

Szöveg árnyék

```
text-shadow: 2px 2px 2px #000;
```
CSS 3

Doboz árnyék

```
box-shadow:10px 10px 20px #000000;
```
CSS 3

Átlátszóság

```
<img src="opacity.png"
  style="margin: 10px; opacity:0.25; width:179px; height:76px;"/>
<img src="opacity.png"
  style="margin: 10px; opacity:0.50; width:179px; height:76px;"/>
<img src="opacity.png"
  style="margin: 10px; opacity:0.75; width:179px; height:76px;"/>
```
CSS 3

Több háttérkép

```
background: url(top.png) top left no-repeat,
url(banner.png) 11px 11px no-repeat,
url(bottom.png) bottom left no-repeat,
url(middle.png) left repeat-y;
```
CSS 3

Több hasáb

```
-moz-column-gap: 2em;
-moz-column-count: 3;
```
CSS 3

A sokféle forrás közül talán az Opera tutorialjait[58] és a *Modern CSS3: Alapvető jellemzők*[59] című cikket érdemes elolvasni.

58 http://dev.opera.com/articles/tags/css3/

59 http://www.webragacs.hu/2010/05/modern-css3-1-resz-alapveto-jellemzok/

3

Szerver oldali működés

Ennek a fejezetnek a célja, hogy a dinamikus weboldalak előállításához szükséges alapismereteket bemutassa.

A fejezet tanulmányozásához szükséges lehet, hogy az 1.2. és 1.4. fejezeteket ismét átnézzük.

Szerver oldali környezetekben igen nagy a választék, de több ok miatt is a klasszikus Apache-PHP-MySQL alapú webfejlesztéssel fogunk ismerkedni e könyvben. Természetesen más környezetek használata esetén is hasonló módon kell megismernünk a környezetet, és hasonló lépéseket kell megtennünk a weboldal fejlesztése során.

3.1. A szerver konfigurálása

Akár az 1.4.2. fejezetben bemutatott XAMPP-ot használjuk, akár Linux alapú szerverrel kell dolgoznunk, hasonló konfigurációs lépések megtételére lesz lehetőségünk és szükségünk. Saját gép esetén – rendszergazdai jogosultságokkal – nem okozhat problémát a szükséges konfigurációs lépések elvégzése. Más esetben (pl. megosztott, bérelt tárhelyszolgáltatás esetén) a rendszergazda, illetve szolgáltató tudja a szükséges lépéseket elvégezni.

A teljesség igénye nélkül nézzünk meg néhány fontosabb konfigurációs lehetőséget.

3.1.1. Az Apache konfigurálása

Az Apache webszerver legfontosabb konfigurációs állománya a *httpd.conf*. Ez XAMPP esetén az *xampp/apache/conf/httpd.conf* útvonalon érhető el. Ezt az állományt kell megnyitnunk egy programozói editorral, és szerkesztenünk. Az állomány elmentése után pedig újra kell indítanunk a webszervert, hogy a változtatások érvényesüljenek.

XAMPP esetén ezt az XAMPP Control Panel segítségével célszerű megoldani (7. ábra).

Nézzünk bele, hogyan is kell szerkesztenünk. Az állomány első sorai:

```
#
# This is the main Apache HTTP server configuration file.  It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/2.2/> for detailed information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.2/mod/directives.html>
# for a discussion of each configuration directive.
```

Jól látszik, hogy – Unixos szokásnak megfelelően – a konfigurációs állomány is sok információt tartalmaz, de a http://httpd.apache.org/docs/2.2/mod/directives.html weboldalt is meglátogathatjuk további információkért.

A példából azt is érdemes még megfigyelni, hogy a # karakterrel kezdődő sorok megjegyzéseket tartalmaznak az Apache szempontjából, vagyis ezekkel a sorokkal az Apache nem fog foglalkozni. Ezért láthatjuk azt a későbbiekben is, hogy egyébként korrekt beállításokat kikommentálva találunk. Ezzel az Apache számára azt mondjuk, hogy ne foglalkozzon az adott sorokkal.

A következőkben néhány változó jelentését nézzük meg.

ServerRoot

A `ServerRoot` meghatározza, hogy mi az Apache szerver telepítési útvonala. Érdemes a példán megfigyelni, hogy az elérési utakat (Windows esetén is) `/` jelekkel kell leírni.

```
ServerRoot "C:/xampp/apache"
```

LoadModule

A `LoadModule` sorok betöltik azokat az opcionális Apache modulokat, amelyekre éppen szükségünk van az oldal működéséhez. Pl. A következő sor a könyvtárstruktúra direkt használata helyett egy logikai útvonaltervezést is lehetővé tesz a `.htaccess` állományok használatával.

```
LoadModule rewrite_module modules/mod_rewrite.so
```

Ezt a megoldást szokás *beszédes URL*-nek is nevezni.

Érdemes Nullstring *Beszédes URL-ek .htaccess, mod_rewrite azaz a rewrite engine ereje 1.* cikkét[60] is elolvasni.

DocumentRoot

A `DocumentRoot`-ba kell azokat az állományokat tennünk, amelyeket a webszerver által elérhetővé akarunk tenni.

```
DocumentRoot "C:/xampp/htdocs"
```

60 http://nullstring.blog.hu/2008/01/23/beszedes_url_ek_htaccess_mod_rewrite_azaz_a_rewrite_engine_ereje_1

Pl. XAMPP esetén az ebbe a könyvtárba helyezett `proba.html` állományt a http://localhost/proba.html útvonalon tudunk elérni a böngészőnkből.

`Directory`

A `Directory` részben már nem csak egyszerű változómegadást láthatunk. Azt határozzuk meg, hogy a megadott könyvtár tartalmát hogyan szolgálja ki a webszerver. Pl. XAMPP esetén az alapbeállítás:

```
<Directory "C:/xampp/htdocs">
  Options Indexes FollowSymLinks Includes ExecCGI
  AllowOverride All
  Order allow,deny
  Allow from all
</Directory>
```

Ezek a beállítások a teljes könyvtárstruktúra (pl. a C:\xampp\htdocs és alkönyvtárai) alapértelmezett beállításai lesznek.

- az `Indexes` miatt kilistázza egy könyvtár állományait, ha az nem tartalmaz `index.html` vagy `index.php` állományt
- az `AllowOverride All` megengedi, hogy az alkönyvtárakban felülírjuk az alapbeállításokat

`IfModule`

Az `IfModule` konfigurációs sorok csak akkor lesznek figyelembe véve, ha az adott modult az Apache betöltötte. Pl.

```
<IfModule dir_module> ...</IfModule>
```

`DirectoryIndex`

A `DirectoryIndex` azt határozza meg, hogy egy könyvtár (pl. http://localhost/ a C:\xampp\htdocs könyvtár, míg a http://localhost/xampp a `C:\xampp\htdocs\xampp` könyvtár) lekérésekor mi történjen. A következő példa esetén először az `index.php` majd az `index.php4`, majd a többi nevű állományt próbálja betölteni a webszerver.

```
DirectoryIndex index.php index.php4 index.php3 index.cgi index.pl
index.html index.htm index.shtml index.phtml
```

Ha bármelyiket megtalálja, akkor a keresés abbamarad. De ha egyiket se találja meg, akkor vagy kilistázza a könyvtár állományait (`Indexes` beállítás esetén), vagy hibaüzenetet kapunk (403-as HTTP hiba: `Forbidden`).

További információk olvashatók pl. *SzabiLinux* oldalán[61].

61 http://szabilinux.hu/apache2/konfig.html

3.1.2. A PHP konfigurálása

Ha az XAMPP megfelelően telepítve van, a http://localhost/xampp/phpinfo.php címen alapvető információkat tudhatunk meg az Apache, PHP és egyéb konfigurációról. Ha távoli tárhelyszolgáltatót használunk, akkor célszerű pl. egy `i.php` nevű állományt létrehozni a következő tartalommal, és FTP-vel feltölteni a tárhely gyökér (vagy `public_html`) könyvtárába:

```php
<?php phpinfo() ?>
```

PHP

Természetesen a használat után töröljük a szerverről ezt az állományt.

Ekkor a böngészőben a http://azendomainem.hu/i.php címen ugyanezeket az információkat láthatjuk:

PHP Version 5.2.9

System	Windows NT NAGYAMILO 6.1 build 7600
Build Date	Feb 25 2009 15:51:41
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-snapshot-template=d:\php-sdk\snap_5_2\vc6\x86\template" "--with-php-build=d:\php-sdk\snap_5_2\vc6\x86\php_build" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=D:\php-sdk\oracle\instantclient10\sdk,shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini

51. ábra. A phpinfo() eredménye (részlet)

Igen sok hasznos információt találhatunk, de most legérdekesebb a számunkra az ábra utolsó sora. Ebben látszik, hogy hol található a `php.ini` állomány, amelyet a PHP konfigurálásához szerkesztenünk kell.

A `php.ini` nem az egyetlen, de a legfontosabb konfigurációs lehetőségünk. Nézzük meg néhány kezdő sorát:

```
[PHP]

;;;;;;;;;;;
; WARNING ;
;;;;;;;;;;;
; This is the default settings file for new PHP installations.
; By default, PHP installs itself with a configuration suitable for
; development purposes, and *NOT* for production purposes.
; For several security-oriented considerations that should be taken
; before going online with your site, please consult php.ini-recommended
; and http://php.net/manual/en/security.php.
```

Jól látszik, hogy itt a `;` jelzi a megjegyzéseket, részletes dokumentációt pedig a http://php.net/manual/en/security.php címen találhatunk.

Az állomány néhány beállítása következik, itt is változó-szerű szintaxissal.

short_open_tag

A `short_open_tag` meghatározza, hogy a `<?php` kezdő karaktersorozat helyett használható-e a `<?` is. Egy esetleges jövőbeli költözés miatt érdemes inkább a mindenhol használható hosszabb szintaxist alkalmazni, tehát nem építeni erre a beállításra.

```
short_open_tag = On
```

safe_mode

A `safe_mode` egy próbálkozás a PHP futtatókörnyezet egyes biztonsági problémáinak kiküszöbölésére. Ha megoldható, kapcsoljuk ki:

```
safe_mode = Off
```

disable_functions

A `disable_functions` sorral megadhatjuk, hogy melyik PHP függvények ne legyenek elérhetőek a futó PHP kód számára. Tárhelyszolgáltatók előszeretettel alkalmazzák, biztonsági okokra hivatkozva. Egyes esetekben tiltani szokták pl. a `show_source`, `system`, `shell_exec`, `passthru`, `exec`, `phpinfo`, `popen`, `proc_open` függvényeket.

max_execution_time

A `max_execution_time` beállítása meghatározza, hogy meddig futhat a PHP kódunk. Ha ezt az időkeretet túllépjük, a futás meg lesz szakítva. Tipikus a 30-60 másodperc körüli beállítás:

```
max_execution_time = 60
```

memory_limit

A memory_limit beállítása meghatározza, hogy mennyi memóriát használhat maximálisan a PHP értelmező a kódunk futtatásához. Ha ezt túllépjük, a futás meg lesz szakítva. Tipikus a 32-256Mb körüli beállítás:

```
memory_limit = 32M
```

Az error_reporting beállítás azt befolyásolja, hogy milyen jellegű hibaüzeneteket fogunk a böngészőben látni a kódunk futása közben. Fejlesztés közben érdemes mindent bekapcsolni:

```
error_reporting  =  E_ALL & ~E_NOTICE
```

Élő környezetben viszont inkább kapcsoljuk ki a publikus hibaüzeneteket, és inkább a log fájlokat figyeljük.

post_max_size és upload_max_filesize

A post_max_size az egyetlen kéréskor, POST metódussal küldött maximálisan elfogadható adatmennyiséget írja le. Az upload_max_filesize egyetlen feltöltendő állomány maximális méretét határozza meg. Az alapbeállítások:

```
post_max_size = 64M
upload_max_filesize = 64M
```

A két érték között van ugyan kapcsolat, de azért nem triviális. Pl. egy több állomány feltöltését lehetővé tevő űrlap esetén lehet, hogy állományonként csak 8Mb-ot engedélyezünk, de összességében egy 32MB-os korlátot is szabunk:

```
post_max_size = 32M
upload_max_filesize = 8M
```

extension_dir és extension

Az extension_dir megadja, hogy hol találhatóak a PHP-hez telepített beépülő csomagok.

```
extension_dir = "C:\xampp\php\ext\"
```

Az extension-ök a betöltendő beépülőket adja meg. Néhány gyakori beépülő:

```
extension=php_gd2.dll
extension=php_mbstring.dll
extension=php_mysql.dll
```

session.save_path és session.auto_start

A session.save_path beállítása a munkamenetek tárolására szolgáló állományok helyét határozza meg. A session.auto_start a munkamenetek automatikus indítását tiltja le a következő kódban:

```
session.save_path = "C:\xampp\tmp"
session.auto_start = 0
```

3.1.3. A *phpMyAdmin* konfigurálása

XAMPP esetén a http://localhost/phpmyadmin címen tudjuk elérni a MySQL adatbázisok konfigurálására szolgáló adminisztrátori felületet. XAMPP esetén különlegesebb konfigurálásra nincs szükség. De ha mégis, akkor az *xampp\phpMyAdmin\config.inc.php* címen tudjuk megtenni.

Egy rövid példa elejéig nézzünk bele ebbe a PHP nyelvű kódba:

```php
/* Authentication type and info */
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = '';
$cfg['Servers'][$i]['AllowNoPasswordRoot'] = true;
```

Jól látszik, hogy a phpMyAdmin az alapértelmezetten létrejövő `root` nevű, és jelszó nélkül használható felhasználót használjuk.

3.1.4. A *MySQL* konfigurálása, jogosultságkezelés

A MySQL legalapvetőbb konfigurációját XAMPP esetén a *c:\xampp\mysql\bin\my.ini* állomány szerkesztésével, majd a MySQL (újra)indításával érhetjük el.

Következzen néhány beállítás példaként.

Ezen a porton lesz elérhető a MySQL szerver:

```
port = 3306
```

Itt tárolódnak az adatbázisok:

```
datadir="C:/xampp/mysql/data"
```

Hova logolja a hibaüzeneteket:

```
log_error="mysql_error.log"
```

Jogosultságkezelés

A *MySQL* jogosultságkezelése rendkívül fontos szerepet tölt be. Adataink biztonsága nagy jelentőségű – bármilyen adatbázisról és alkalmazásról legyen is szó.

A MySQL jogosultságkezelése az automatikusan települő, `mysql` nevű adatbázis tartalmára épül. Ez alapján minden kapcsolódás során figyelembe veszi a következőket:

- honnan kapcsolódunk

- kik vagyunk
- mit tehetünk

Az adatbázis tábláinak a következő szerepe van:

- `user`: Ki és honnan kapcsolódhat. Definiálja a globális jogokat is.
- `db`: Melyik adatbázishoz ki férhet hozzá. Definiálja az adatbázis szintű jogokat is.
- `host`: Melyik adatbázishoz honnan lehet csatlakozni.
- `tables_priv` és `columns_priv`: Tábla és mező szintű jogosultságokat definiál.

Ez alapján a hitelesítés két lépésben történik.

1. A MySQL megállapítja, hogy van-e jogunk csatlakozni a megadott névvel, jelszóval és helyről a kívánt adatbázishoz.
2. Minden SQL parancs esetén megvizsgálja, hogy van-e jogunk az adott parancsot lefuttatni.

Adatbázisok és felhasználók

Tárhelyszolgáltatók esetében igen gyakori, hogy egyetlen adatbázisunk lehet, és hozzá egyetlen felhasználónk. Ebben az esetben tehát az adatbázishoz az egy felhasználóval csatlakozunk, és a felhasználó is csak ehhez az egy adatbázishoz kapott jogosultságokat.

Ezt a szokást más esetben is célszerű használni. Olyankor is, ha egyébként lenne lehetőségünk több adatbázishoz ugyanazt a MySQL felhasználói azonosítót használni.

3.1.5. Karakterkódolás: Használjunk mindenhol UTF-8-at

Mivel a magyar nyelv használata weboldalak készítésénél nem triviális, érdemes bevezetőként ezzel a témával is foglalkoznunk.

Mivel ma, 2011-ben az UTF-8 karakterkódolás egyre dominánsabbnak (bár közel sem problémamentesnek) tekinthető, érdemes egyre inkább megismerni, még ha a korlátaival is kell még szembesülnünk.

Hodicska Gergely 2006-os nagyszerű cikke *Karakterkódolási problémák kiküszöbölése*[62] címmel még nem az UTF-8-ról szól, de az alapelvek miatt mindenképpen érdemes elolvasni.

Bevezetőként még érdemes azt elmondani, hogy a PHP jelenlegi legfrissebb (5.3.x) verziója sem kezeli jól az UTF-8 kérdést. Ha azonban követjük a következő alapelveket, többnyire kézben tarthatjuk a problémát.

62 http://weblabor.hu/cikkek/karakterkodolasiproblemakkikuszobolese

Konfigurációs állományok

A *httpd.conf*-hoz adjuk hozzá:

```
AddDefaultCharset UTF-8
```

A *php.ini*-hez adjuk hozzá:

```
default_charset = "utf-8"
```

A *my.ini*-hez adjuk hozzá:

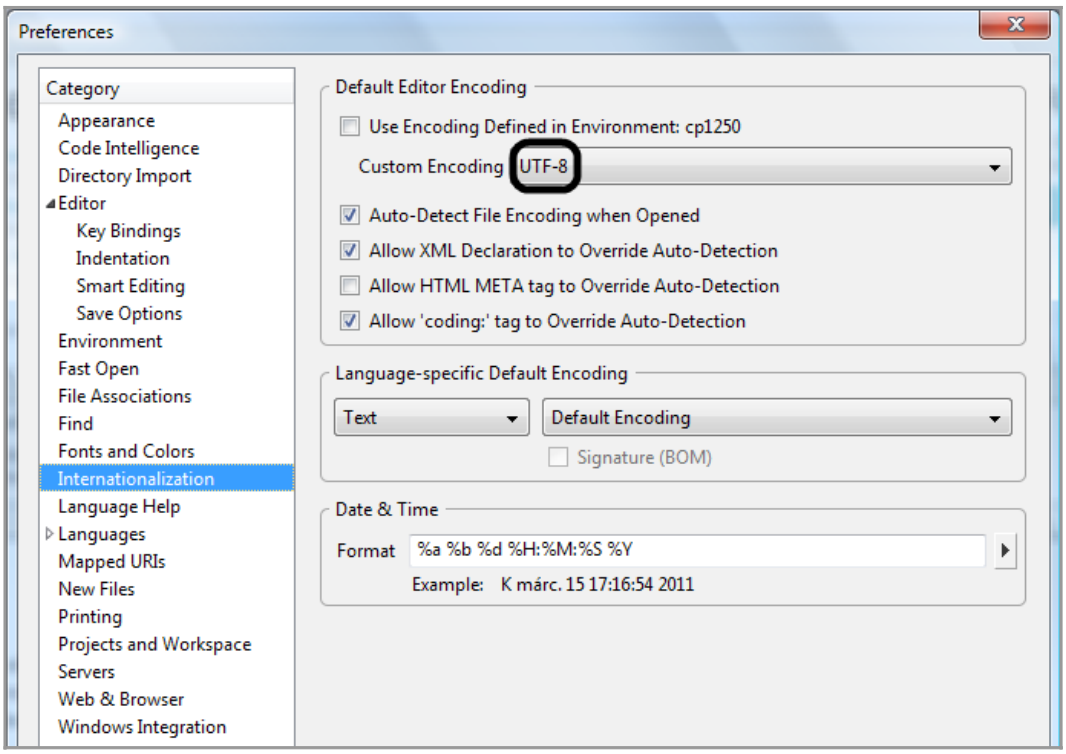
```
[client]
default-character-set=utf8

[mysqld]
character-set-server=utf8
default-character-set=utf8
default-collation=utf8_unicode_ci
init-connect='SET NAMES utf8'
character-set-client = utf8
```

Minden szerver alkalmazást (szolgáltatást) indítsunk újra, és már használhatjuk is.

HTML, PHP forráskód mentése

A forráskód készítésekor az editorunkkal tudatnunk kell, hogy milyen karakterkódolással kell elmentenie az állományt. Ez Komodo Edit esetén két módon tehetjük meg:

Az *Edit* / *Preferences* / *Internationalization* / *Custom encoding* étréke legyen UTF-8. Ez minden további állomány alapértelmezése lesz.

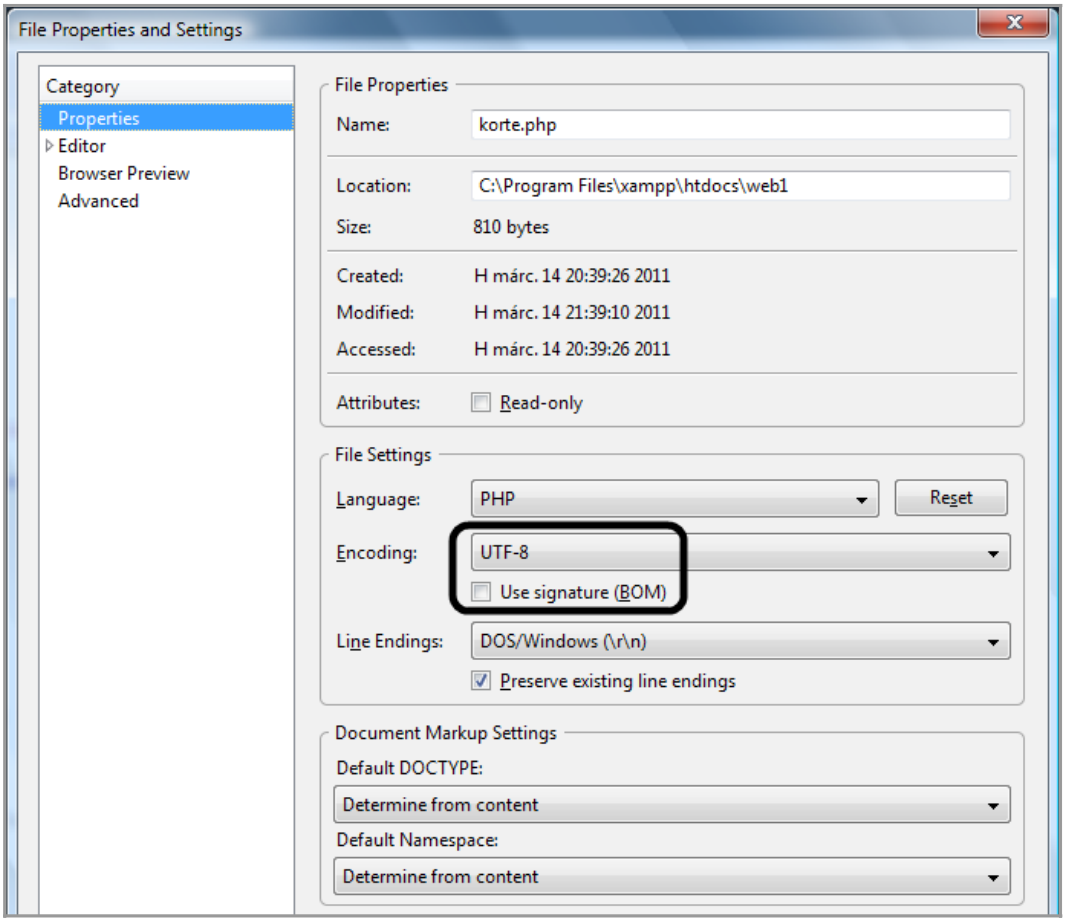


52. ábra. UTF-8 alapértelmezett beállítása Komodo Edit esetén

Ha egy megnyitott állományt, akarunk UTF-8 kódolással menteni, akkor az Edit / *Current File Settings* / *Properties* / *Encoding* értékét állítsuk UTF-8-ra, *BOM* (*byte order mark*) nélkül, és mentsük az állományt.

„Egyes Windows-os programok a fájl elejére írt 0xEF,0xBB,0xBF bájtsorozattal jelzik, hogy UTF-8 kódolású fájlról van szó; ezt néha UTF-8 BOM kódolásnak hívják. [...] UTF-8-ban ennek a karakternek elméletileg nincs jelentése, így használható a kódolás jelzésére, azonban ez megtöri az ASCII-kompatibilitást, így nem javasolt. Az így kódolt PHP fájlok például a weboldal elején megjelenő ï»¿ karaktersorozatot eredményeznek." [63]

63 Forrás: http://hu.wikipedia.org/wiki/UTF-8#BOM



53. ábra. UTF-8 beállítása az aktuális állományra, Komodo Edit esetén

Ha azt akarjuk, hogy a böngésző UTF-8-ként dolgozza fel a választ, a korábban már látott `meta` tagot használhatjuk:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
```
HTML

Ennél még jobb megoldás, ha a forráskód legelején, még a `doctype` és a `html` tag előtt kiküldjük a következő `header` függvényt:

```
<?php
  header('Content-Type: text/html; charset=utf-8');
?><!DOCTYPE html>
```
PHP

Problémák a sztringek kezelésével

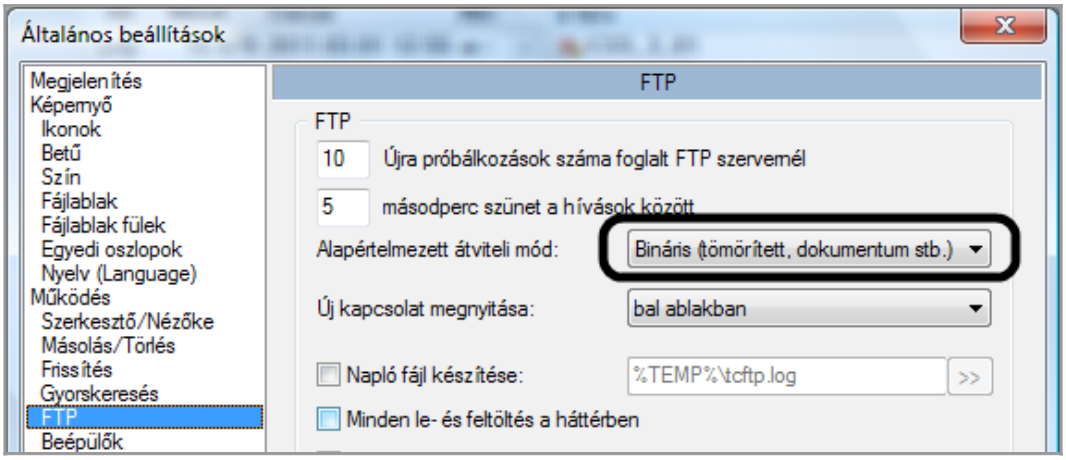
Ha UTF-8 kódolást használunk, a hagyományos karakter-kezelési funkciók (pl. az indexelés) nem megfelelően fognak működni. De gondban leszünk az `strlen`, `strtoupper`, `strpos` és hasonló függvényekkel is. Jelenleg megoldást kiegészítők formájában találhatunk.

Pl. a *Multibyte String* használata esetén az `mb_strlen` és hasonló függvényeket[64] kell használnunk. Emellett egyre inkább elterjedő megoldásnak tűnik az *iconv*[65].

FTP feltöltés

Arról sem érdemes megfeledkeznünk, hogy érdemes az állományainknál garantálni, hogy az FTP feltöltés során se változzon semmi a forráskódunkban. Ezért az FTP alkalmazásunkat érdemes úgy beállítani, hogy az bináris módban töltse fel az állományokat.

Total Commander esetén a Beálltások / Általános beállítások / Működés / FTP / *Alapértelmezett átviteli mód* a *Bináris* legyen.



54. ábra. Bináris átviteli mód beállítása a Total Commanderben

Ezekkel a beállításokkal az UTF-8 kódolás többnyire elérhető.

Adatbázis-kapcsolat

Adatbázis-használat esetén az adatbázis létrehozása és a PHP-MySQL kommunikáció során is érdemes beállítani az utf-8-at.

64 További információk: http://www.php.net/manual/en/ref.mbstring.php
65 http://www.php.net/manual/en/ref.iconv.php

3.2. PHP alapok

A PHP (*PHP: Hypertext Preprocessor*) erőteljes szerver-oldali szkriptnyelv, jól alkalmazható dinamikus és interaktív weboldalak elkészítéséhez. A versenytársakkal szemben (mint például az ASP) széles körben alkalmazott és alkalmazható, szabad és hatékony alternatíva.

A PHP használata esetén a PHP kódot közvetlenül a HTML kódba ágyazhatjuk be. A nyelv szintaxisa nagyon hasonlít a C/C++, Java, Javascript vagy Perl nyelvekhez. Leggyakrabban az Apache webszerverrel használjuk együtt különböző operációs rendszerek alatt.

Egy PHP állomány szöveget, HTML tagokat és PHP kódokat (szkripteket) tartalmazhat.

A PHP kód még a szerveren lefut, mielőtt a webszerver a látogató kérését kiszolgálná. A látogató kérésére egyszerű HTML állománnyal válaszol.

Futása során kommunikálhat adatbázis-szerverrel (pl. MySQL, Informix, Oracle, Sybase, PostgreSQL, ODBC) is.

A PHP állomány szokás szerint `.php` kiterjesztésű, bár a webszerver beállításai akár ettől eltérő megoldásokat is lehetővé tesznek. (Előfordul pl., hogy az éppen aktuális beállítások szerint a `.html` állományokat is értelmezi a PHP motor.)

A PHP szabadon letölthető és használható.

3.2.1. Szintaxis

Egy weboldal PHP forrását nem tudjuk a böngészőnk segítségével megnézni. Ott mindig csak a PHP által előállított HTML kimenet látható.

A fejezet példaprogramjai – helytakarékossági okokból – a legtöbb esetben nem teljes vagy nem szabványos HTML oldalakat készítenek.

A HTML oldalba ágyazott PHP kód kezdetét és végét egyértelműen jelölni kell a PHP értelmező számára. Ennek többféle módja közül leginkább a következő ajánlható:

```php
<?php ... ?>
```
PHP

Szokás még használni egy rövidebb jelölést is, de ez nem biztos, hogy minden konfiguráció esetén működik. A 3.1.2. fejezetben olvashattunk a `short-open-tag`-ról.

```php
<? ... ?>
```
PHP

A következő „Hello World” példa jól mutatja, hogyan ágyazzuk be a PHP kódot a HTML oldalba:

```php
<html>
  <body>
<?php
  echo "Hello World";
?>
  </body>
</html>
```

Majdnem minden PHP utasításnak `;`-vel kell végződnie. Kivétel: a PHP záró tag előtt elhagyható. Két egyszerű lehetőségünk van arra, hogy a HTML állományba kimenetet szúrjunk be: `echo` és `print`. A két függvény közül szabadon választhatunk, de az olvashatóság kedvéért nem illik keverni a kettőt egy kódon belül. Emellett egy ettől eltérő megközelítést is használhatunk:

Sablonszerű megközelítés

A sablonrendszerekről szóló 3.11. fejezet példái elsősorban a következő példához hasonlóan, `print` és `echo` nélkül kezelik a kimenetet:

```php
<acronym class="datum" title="<?= $datum['hosszu'] ?>">
  <span class="honap"><?= $datum['honap'] ?></span>
  <span class="nap"><?= $datum['nap'] ?></span>
</acronym>
```

A példa lényege, hogy nem a PHP-be illesztjük a HTML-t készítő kódsorokat, hanem fordítva: a fix HTML szövegbe szúrjuk be a PHP kifejezéseket (elsősorban változók beillesztéséhez szükséges utasításokat). Ehhez az érték kiírására szolgáló szintaxis használható:

```php
<?= kifejezés ?>
```

A kifejezés kiértékelődik, és értéke a tag helyére kerül beszúrásra.

Ez a szintaxis csak akkor használható, ha a `php.ini`-ben a `short_open_tag` be van kapcsolva (3.1.2. fejezet).

3.2.2. Megjegyzések

A PHP kódban egysoros (sorvégi) és többsoros megjegyzéseket is használhatunk:

```php
<html>
<body>
<?php
  // Egy soros megjegyzés
  /*
    Ez egy több soros
    megjegyzés blokk
  */
?>
</body>
</html>
```

Nem szabad összekeverni a HTML és a PHP megjegyzések szerepét. A HTML megjegyzés a kliens gépén is megjelenik, „fogyasztja” a sávszélességet, általában nem javasolt. Ezzel

szemben a PHP megjegyzések nem kerülnek el a klienshez, az ilyen megjegyzéseket a PHP nem továbbítja a kimenetre. Célja a forráskód későbbi könnyebb megértése.

Egy példa a szerző korábbi honlapjáról:

PHP
```
<? /* Az előzetes megjelenítésére, a hozzászólások számával */ ?>
<div class="hir">
<? if ($lehethozza || $tartalom) { ?>
  <h2><a href="<?= $index ?>?<?= $blognev ?>/<?= $url ?>">
    <?= $hirCim ?></a></h2>
<? } else { ?>
  <h2><?= $hirCim ?></h2>
<? }?>
```

3.2.3. Változók

A PHP nyelvben minden változó neve `$` karakterrel kezdődik. A változók alapvetően szövegeket, számokat és tömböket tartalmazhatnak.

A következő példa a `$txt` változót használja a kiírandó szöveg tárolására:

PHP
```
<html>
<body>
<?php
  $txt="Hello World";
  echo $txt;
?>
</body>
</html>
```

Ha két vagy több szöveges kifejezést (sztringet) össze akarunk fűzni, akkor a `.` (pont) operátor használható:

PHP
```
<html>
<body>
<?php
  $txt1="Hello World";
  $txt2="1234";
  echo $txt1 . " " . $txt2;
?>
</body>
</html>
```

A programunk a következő kimenetet állítja elő:

```
Hello World 1234
```

Változónevek

A változónevek az ABC betűiből, számokból és az aláhúzás karakterből állhatnak, de nem kezdődhetnek számmal. A nevek szóközt nem tartalmazhatnak, így az összetett változónevek esetén az aláhúzás karaktert (`$my_string`) , vagy a közbülső szó nagy kezdőbetűjét (`$myString`) alkalmazzuk.

Változók típusai

A PHP nyelv a következő típusok használatát teszi lehetővé:

- logikai (`boolean`), értéke `true` (igaz) vagy `false` (hamis) lehet
- egész számok (`integer`), pl. `5`, `-23`
- lebegőpontos számok (`double`), pl. `3.14`, `2.0`
- sztringek (`string`), pl. `"alma"`
- tömbök (`array`), amik kulcsok és értékek rendezett sorozata
- objektumok (`object`), amik egy osztály példányai
- erőforrások, pl. egy adatbázis vagy állomány azonosítója
- NULL, ami egy érték nélküli változó

A PHP nyelv *gyengén típusos*, így nincs szükség a változó típusának előzetes megadására. A típus legtöbbször az értékadás pillanatában állítódik be, az értékadás során használt érték típusától függően.

`gettype`

A `gettype` függvény segítségével lekérdezhetjük egy változó típusát. A következő példánkban ezt egyszerűen kiírjuk:

PHP
```
$a = 5;
$b = 6.2;
print gettype($a);
print gettype($b);
```

A futás eredménye:

```
integer
double
```

Implicit típuskonverzió

Bizonyos műveletek végzése közben automatikus típuskonverzió történhet. Pl. a következő kódban az összeadás elvégzése érdekében az egész változó értéke `3.0` valós értékké konvertálódik, hogy az összeadás két azonos típus között tudjon megtörténni. Így a végeredmény `5.4` valós érték lesz.

PHP
```
$a = 3;
$b = 2.4;
print $a + $b;
```

Érdemes itt külön kihangsúlyozni, hogy az `$a` változó értéke vagy típusa ettől nem változik meg, a konverzió az értékének egy ideiglenes másolatán történik.

Külön érdemes megemlíteni egy speciális esetet. Ha összeadásnál sztring operandust használunk, akkor nem kapunk hibaüzenetet. Ehelyett számmá próbálja konvertálni azokat, hogy az összeadást el tudja végezni rajtuk. Így a következő kód kimenete integer típusú és 5 értékű lesz:

```php
print '2 alma' + '3 körte';
```

Explicit típuskonverzió

Ha egy kifejezés típusa nem megfelelő a számunkra, direkt módon is előírhatjuk a konverziót. A következő példában az összeadás előtt a $ változó értéke (egészen pontosan a másolata) egésszé konvertálódik, és az összeadás két egész szám között történik meg. Így az eredmény is 5 egész szám lesz.

```php
print $a + (integer)$b;
```

A következő példában explicit és implicit típuskonverzió is történik. A példa kód 2 tizedesjegyre kerekíti az $a valós számot:

```php
$a = 0.33333333;
print (integer)($a*100) / 100.0;
```

Nézzük sorba a kiértékelés lépéseit:

1. $a*100: a 100 implicit valós számmá konvertálódik
2. (integer) 33.333333: explicit egész számmá konvertálódik
3. 33 / 100.0: a 33 implicit valós számmá konvertálódik

A végeredmény 0.33 valós érték lesz.

Érdemes végül megjegyezni, hogy az $a és $b változók itt sem változtak meg. A következő megoldás viszont a változó típusát is megváltoztatja.

settype

Időnként szükség lehet arra, hogy egy változó típusát megváltoztassuk. Mivel ez sokszor automatikusan megtörténik, ritkán van szükség a settype használatára. De nézzünk mégis egy példát:

```php
$a = 5.2;
settype($a, 'integer');
```

Mivel a double típusú változó típusát egészre állítottuk, kerekítés történ, és a változó értéke 5 lesz.

Változó változók

Érdemes még egy érdekes példán keresztül megnézni, hogy mit is jelent a változók kiértékelése.

```php
$a = 'b';
$$a = 'alma';
```

A második sor ekvivalens ezzel:

```php
$b = 'alma';
```

Létezés vizsgálata, és megszüntetése

Időnként szükség lehet arra, hogy megvizsgáljuk: a változó létezik-e. Erre használható az `isset` függvény. Más esetben megszüntetni szeretnénk egy változót. Erre az `unset` függvény alkalmazható.

3.2.4. Sztringek használata

A PHP nyelvben nagyon nagy jelentősége van a szövegek kezelésének, hiszen a PHP kód futtatásának végcélja egy szövegkimenet előállítása.

Sztring létrehozása aposztróffal

A legkönnyebben úgy adhatunk meg egy egyszerű sztringet, hogy aposztrófok (' karakterek) közé tesszük.

```php
echo 'Arnold egyszer azt mondta: "I\'ll be back"';
echo 'You deleted C:\\*.*?';
```

Jól megfigyelhető, hogy ilyenkor az idézőjeleknek nincs különös hatása, az aposztróf leírásához viszont a \ speciális (ún. escape) karakter használata kötelező. A \ karakter az utána következő karakterrel speciális jelentést hordoz, ezért a `C:\` kiírása érdekében is használnunk kell. A kimenet:

```
Arnold egyszer azt mondta: "I'll be back"
You deleted C:\*.*?
```

Sztring létrehozása idézőjellel

Az idézőjeles sztringek legfontosabb jellemzője az, hogy a változók behelyettesítésre kerülnek.

Ha dollár (`$`) jelet talál a PHP egy sztringben, megkeresi az összes ezt követő azonosítóban is használható karaktert, hogy egy érvényes változónevet alkosson. Ezért kapcsos zárójeleket kell használnunk, ha pontosan meg szeretnénk határozni, meddig tart egy változó.

```php
$ingatlan = 'ház';
echo "kertes $ingatlan kerítéssel";
// működik, szóköz nem lehet változónévben
echo "páros $ingatlanszám";
// nem működik, az 's' és további karakterek lehetnek változónévben
echo "páros ${ingatlan}szám";
// működik, mert pontosan meg van adva a név
echo "páros {$ingatlan}szám";
// működik, mert pontosan meg van adva a név
```

Sztring létrehozása 'heredoc' szintaxissal

Egy másfajta módszer a sztringek megadására a heredoc szintaxis. A `<<<` jelzés után egy azonosítót kell megadni, majd a sztringet, és végül az azonosítót még egyszer, ezzel zárva le a sztringet.

A lezáró azonosítónak mindenképpen a sor legelső karakterén kell kezdődnie. Ugyancsak figyelni kell arra, hogy ez az azonosító is az általános PHP elemek elnevezési korlátai alá tartozik.

```php
$str = <<<VAS
Példa egy stringre, amely
több sorban van, és heredoc
szintaxisú
VAS;
```

Példánkban a *Vége A Stringnek* kezdőbetűit, csupa nagy betűvel alkalmaztuk.

Sztring karaktereinek elérése és módosítása

A string karaktereire nullától számozott indexekkel lehet hivatkozni, a string neve után megadott kapcsos zárójelek között.

```php
$str = 'Ez egy teszt.';
$first = $str{0};
$third = $str{2};
```

Ezzel a módszerrel nem csak kiolvasni, hanem módosítani is tudjuk a sztring karaktereit:

```php
$str{0} = 'A';
```

Ennek hatására az `$str` szting 0. karakterét írjuk felül.

Végül érdemes itt ismét megemlíteni, hogy UTF-8 karakterkódolás használata esetén az indexelés nem fog megfelelően működni. Helyette a *Multibyte String* függvényeket[66] vagy az *iconv* függvényeket[67] használhatjuk.

66 http://www.php.net/manual/en/ref.mbstring.php
67 http://www.php.net/manual/en/ref.iconv.php

3.2.5. Operátorok és kifejezések

Az operátorok műveleteket végeznek az operandusaikkal. Pl. a `4 + 5` kifejezésben a `+` az operátor, a `4` és `5` pedig az operandus, az eredmény pedig `9`. Operandus lehet egy változó is.

A művelet eredménye maga is lehet újabb operandus valamelyik másik operátor műveletében. Pl `2 * 3 + 4` esetén a szorzás eredménye (`6`) az összeadás bal operandusa lesz.

A fenti példáink mind kifejezések is egyben, de kifejezés önmagában egy konstans (még pontosabban literál) is. A kifejezésekkel sok helyen találkozhatunk a programozásban, egyelőre az értékadó operátorok jobb oldalán fogunk legtöbbször. Pl. a következő példánkban `$b + $c` egy kifejezés, ennek kiszámítása után pedig az eredmény (vagyis a kifejezés értéke) az `$a` változóban kerül tárolásra.

```php
$a = $b + $c;
```

Következzen egy gyors összefoglaló az operátorokról.

Értékadó operátorok

Az értékadó operátorok a bal oldali operandus értékét változtatják meg. Az első (=) operátor új értéke nem függ a korábbi értékétől, míg a többi operátor a korábbi értékéhez képest számolja ki az új értéket.

Operátor	Példa	Jelentés
=	$x=$y;	$x=$y;
+=	$x+=$y;	$x=$x+$y;
-=	$x-=$y;	$x=$x-$y;
*=	$x*=$y;	$x=$x*$y;
/=	$x/=$y;	$x=$x/$y;
%=	$x%=$y;	$x=$x%$y;
.=	$x.=$y;	$x=$x.$y;

5. táblázat. Értékadó operátorok

Ha a két operandus típusa nem egyezik meg, a jobb oldal megfelelő konvertált másolata fog értékül adódni.

Aritmetikai operátorok

Az aritmetikai operátorok a matematikából ismert feladatukat látják el. Többnyire egész vagy valós típusú operandusokkal szoktuk őket használni.

Operátor	Művelet	Példa	Eredmény
+	összeadás	$x=2; $x+2	4
-	kivonás	$x=2; 5-$x	3
*	szorzás	$x=4; $x*5	20
/	osztás	15/5 5/2	3.0 2.5
%	maradékos osztás maradéka	5%2 10%8 10%2	1 2 0
++	növelés	$x=5; $x++ ++$x	$x=6
--	csökkentés	$x=5; $x-- --$x	$x=4

6. táblázat. Aritmetikai operátorok

Bizonyos esetekben automatikus típuskonverzió történhet. Ez többnyire akkor fordul elő, ha a két operandus nem azonos típusú. Ekkor olyan automatikus (implicit) konverzió történik, amely nem jár adatvesztéssel. Pl. az `$x=4; $x*5` esetben nincs, de az `$x=4; $x*5.2` esetben már van szükség konverzióra. Ilyenkor az eredmény se egész, hanem valós típusú lesz.

Az osztás esetén akkor lesz valós az eredmény, ha mindkét operandus egész. Vagyis a C nyelvtől eltérően nem maradékos osztás történik.

A növelés és csökkentés operátor csak egész típusú változó operandussal használható. Az operátor mind a változó előtt, mind utána írható. A két írásmód eredménye között akkor van különbség, amikor az operátort egy összetett kifejezés részeként szerepel.

```php
$a = 2;
$b = 3;
$c = $a++ + $b++;
```

PHP

Ekkor $a és $b értéke is nő, de mindkettő csak az összeadás elvégzése után. Így `$c` értéke `5` lesz.

Ha módosítjuk a `$c = $a++ + ++$b;` alakra, akkor `$b` növelése még az összeadás elvégzése előtt megtörténik, így `$c` értéke `6` lesz. Végeredményben tehát az operátor helye a növelés/csökkentés időbeliségét befolyásolja.

Összehasonlító operátorok

Gyakran van szükségünk arra, hogy két kifejezés viszonyát (relációját) meg tudjuk határozni. Az összehasonlító műveleteket eredménye mindig logikai érték (`true` vagy `false`). Leggyakrabban feltételes elágazásoknál, ciklusok feltételeként találkozhatunk velük.

Operátor	Művelet	Példa	Eredmény
`==`	egyenlő	`5==8`	`false`
`!=`	nem egyenlő	`5!=8`	`true`
`>`	nagyobb	`5>8`	`false`
`<`	kisebb	`5<8`	`true`
`>=`	nagyobb-egyenlő	`5>=8`	`false`
`<=`	kisebb-egyenlő	`5<=8`	`true`
`===`	típus és érték egyenlő	`'5'===5`	`false`

7. táblázat. Összehasonlító operátorok

A *nagyobb*, *kisebb*, *nagyobb-egyenlő* és *kisebb-egyenlő* műveletek számok vagy sztringek esetén értelmezhetőek.

Sztringek esetén az eredmény karakterkészlet-függő.

Az *egyenlő* és *nem egyenlő* operátorok bármilyen, akár különböző típusú operandusokon is elvégezhetőek. Ilyenkor implicit típuskonverzió történik, ami néha meglepő eredményt produkál. Ha az ilyen meglepetéseket el szeretnénk kerülni, jól jöhet a *típus és érték egyenlő* operátor. Ez csak akkor ad igaz eredményt, ha eleve megegyező típusokkal használjuk, és az érték is megegyezik.

Logikai operátorok

A logikai operátorok bonyolultabb feltételek felépítése során elengedhetetlenek lesznek. Általában az operandusok eleve logikai típusúak. Ha mégsem, akkor konverzió fog történni. Logikai `false` értékké konvertálódik pl.

- 0, 0.0 számok
- üres sztring ('')
- üres tömb
- üres objektum

Nézzük meg a logikai operátorokat:

Operátor	Művelet	Példa	Eredmény
&& and	és	$x=6 $y=3 ($x < 10 && $y > 1) ($x < 10 and $y > 1)	 true true
|| or	vagy	$x=6 $y=3 ($x==5 || $y==5) ($x==5 or $y==5)	 false false
!	nem	$x=6 $y=3 !($x==$y)	 true

8. táblázat. Logikai operátorok

Az operátorok precedenciája

Az operátorok végrehajtási sorrendjét az eddig megismerteken kívül az „erősségük” és a végrehajtás iránya határozza meg. Nézzük meg a sorrendet a legerősebbtől a leggyengébb irányába:

1. `++, --, (típuskonverzió)`
2. `/, *, %`
3. `+, -`
4. `<, <=, >, >=`
5. `==, !=, ===`
6. `&&`
7. `||`
8. `=, +=, -=, /=, *=, %=, .=`
9. `and`
10. `or`

Ha tehát egy kifejezésben különböző szintű operátorokat alkalmazunk, akkor a fenti sorrendben fogja őket a PHP kiértékelni. Az azonos csoportba tartozó operátorok egyforma erősségűnek tekinthetők, közöttük nem teszünk különbséget. Ilyenkor általában balról jobbra haladva haladunk a műveletvégzésben.

A zárójelek segítségével a fenti sorrend manipulálható.

Rövidzár kiértékelés

Bizonyos esetekben a logikai operátor második operandusa nem értékelődik ki. Például a következő esetben:

```php
($a < 3) && ($b == 5)
```

Az `&&` operátor csak akkor ad vissza `true`-t, ha mindkét operandus `true`. Tehát, ha `$a < 3` `false`, akkor `&&` bal oldala `false`, és a kifejezés eredménye (`false`) a jobb oldali operandus kiértékelése nélkül születik meg. Ilyen esetekben a PHP nem értékeli ki a jobb oldali operandust.

Ehhez hasonlóan, ha a `||` operátor bal oldali operandusa `true`, felesleges a jobboldalt kiértékelni, így az nem is fog megtörténni.

3.2.6. Tömbök

A tömb egy vagy több értéket tárolhat egyetlen változónévvel.

Ha PHP-vel dolgozunk, akkor előbb vagy utóbb valószínűleg szeretnénk több hasonló változót létrehozni. Több változó használata helyett az adatokat egy tömb elemeiként is tárolhatjuk. A tömb minden elemének egyedi azonosítója (indexe, kulcsa) van, így könnyen hozzáférhető.

A tömböknek három különböző típusa van:

- *Sorszámozott tömb* – Olyan tömb, amelynek egyedi numerikus (vagyis szám) azonosítókulcsai vannak
- *Asszociatív tömb* – Olyan tömb, amelynek egyedi szöveg (vagyis sztring) azonosítókulcsai vannak
- *Többdimenziós tömb* – Olyan tömb, amely egy vagy több tömböt (is) tartalmaz

Hamarosan pontosítani fogjuk: igazából egyféle tömb van, de háromféle tipikus felhasználási eset szokott előfordulni.

Sorszámozott tömbök

A sorszámozott tömb minden eleméhez egy numerikus azonosítókulcsot tárol. A sorszámozott tömb létrehozásának több módja van.

Nézzünk két példát, amelyek ugyanazt a tömböt hozzák létre. Az első példában az azonosítókulcsok (indexek) automatikusan kerülnek hozzárendelésre az elemekhez:

```php
$names = array("Peter","Quagmire","Joe");
```

A második példában azonosítókulcsot manuálisan rendelünk hozzá:

```php
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
```

Az azonosítókulcsokat felhasználhatjuk szkriptekben:

```php
echo $names[1] . " and " . $names[2] .
" are ". $names[0] . "'s neighbors";
```

Az előző kódrészlet kimenete:

```
Quagmire and Joe are Peter's neighbors
```

Tömbök feltöltésekor az index megadása sem kötelező. Ha elhagyjuk, akkor a legnagyobb használt index után folytatja a számozást. Az előző példát folytatva a 3-as index alá kerül `Jack`:

```php
$names[] = "Jack";
```

Ahogy az eddigiek alapján sejthető, az index-számok nem csak folytonosak lehetnek.

Asszociatív tömbök

Az asszociatív tömb minden azonosítókulcsához egy értéket rendelünk hozzá. Ha egyedi értékeket szeretnénk tárolni, nem feltétlenül a sorszámozott tömb a legjobb megoldás erre. Az asszociatív tömb értékeit kulcsként használhatjuk, s értékeket rendelhetünk a kulcsokhoz.

Az asszociatív tömbökre először az űrlapból kapott adatok feldolgozásakor merült fel igény (3.5. fejezet).

Ebben a példában tömböt használunk, hogy különböző személyekhez az életkorukat rendeljünk:

```php
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Ez a példa az előzővel azonos értékű, de a tömb létrehozásának más módját mutatja:

```php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

Az azonosítókulcsot így használhatjuk szkriptben:

```php
echo "Peter is " . $ages['Peter'] . " years old.";
```

Az előző kód kimenete:

```
Peter is 32 years old.
```

Bizonyos értelemben pontosabb lenne úgy fogalmazni, hogy a PHP csak az asszociatív tömböket ismeri, és a kulcs lehet szám is.

Többdimenziós tömbök

Egy többdimenziós tömbben a fő tömb minden eleme lehet maga is egy tömb. Az altömb minden eleme lehet (de nem kötelezően) tömb, és így tovább.

Ebben a példában olyan többdimenziós tömböt hozunk létre, melynek azonosítókulcsai automatikus hozzárendeléssel rendelkeznek:

PHP
```
$families = array (
  "Griffin"=>array (
    "Peter",
    "Lois",
    "Megan"
  ),
  "Quagmire"=>array (
    "Glenn"
    ),
  "Brown"=>array (
    "Cleveland",
    "Loretta",
    "Junior"
  )
);
```

A fenti tömb így néz ki a kimenetre írva (`print_r($families)`):

```
Array (
  [Griffin] => Array (
    [0] => Peter
    [1] => Lois
  ),
  [Quagmire] => Array (
    [0] => Glenn
  ),
  [Brown] => Array (
    [0] => Cleveland
    [1] => Loretta
    [2] => Junior
  )
)
```

A példából jól látszik, hogy az egyes altömbök elemszámának se kell egyenlőnek lenni.

A 3.6.2. fejezetben részletesebben bemutatásra kerülő példa oldal alapja, hogy a következő tömb alapján generálódik a felső menü:

```php
$oldalak = array(
  '/' =>
    array('fájl' => 'kezdo', 'szoveg' => 'Kezdőlap'),
  'magamrol' =>
    array('fájl' => 'magamrol', 'szoveg' => 'Magamról'),
  'szakmai_oneletrajz' =>
    array('fájl' => 'szakmai_oneletrajz', 'szoveg' =>
      'Szakmai önéletrajz'),
  'galeriak' =>
    array('fájl' => 'galeriak', 'szoveg' => 'Galériák'),
  'kapcsolat' =>
    array('fájl' => 'kapcsolat', 'szoveg' => 'Kapcsolat'),
);
```

A külső tömb kulcsai az URL részletei, a belső tömbök kulcsai pedig a szöveges oldalt tartalmazó állomány nevét, és a menüpontok címét tartalmazzák.

3.2.7. Szuper-globális változók

A 3.3.3. fejezetben látni fogjuk, hogy egy kódblokkban általában a helyben létrehozott változókat szoktuk használni. De a kivétel itt is erősíti a szabályt: egyes változók globálisak ugyan, de nem kell feltétlenül a `global` kulcsszót használnunk, hogy bárhol használni tudjuk őket. Ezeket a mindenhonnan egyformán létező változókat csupa nagybetűvel elnevezve, készen kapjuk a PHP értelmezőtől. Később a többségükkel meg is fogunk ismerkedni:

`$GLOBALS`, `$_SERVER`, `$_GET`, `$_POST`, `$_FILES`, `$_COOKIE`, `$_SESSION`, `$_REQUEST`, `$_ENV`.

3.3. Vezérlési szerkezetek

Egy komolyabb programozási feladat nem oldható meg oly módon, hogy előre leírjuk egyenként az egymás után végrehajtandó összes lépést. E fejezet bemutatja, hogy hogyan tudjuk a kódot „tömöríteni” a ciklusok segítségével és hogyan tudunk szituációtól függően másként reagálni az elágazások segítségével. Végül megnézzük, hogyan tudunk újrahasznosítható kódrészeket létrehozni függvények segítségével.

3.3.1. Elágazások

Az `if`, `elseif` és `else` kulcsszavak használatosak a PHP-ben feltételhez kötött utasítások végrehajtására.

Sok esetben merül fel egy-egy programozási feladat kapcsán annak szükségessége, hogy különféle lehetőségekhez más-más viselkedést rendeljünk. Ennek megoldására használhatjuk a feltételes kifejezéseket.

`if-else` - Akkor használjuk, ha a feltétel igaz volta esetén egy adott kódot, hamis volta esetén egy másik kódot szeretnénk futtatni.

`elseif` - Az `if-else` kifejezéssel együtt használjuk, ha azt szeretnénk, hogy egy adott kódrészlet akkor fusson, ha több feltétel közül csak egy teljesül.

Az `if-else` szerkezet

Egy feltétel teljesülése esetén futtat egy kódrészletet, amennyiben a feltétel nem teljesül, akkor egy másikat. Szintaxis:

```php
if (feltétel)
  végrehajtandó kód, ha a feltétel teljesül;
else
  végrehajtandó kód, ha a feltétel nem teljesül;
```

A következő példa kimenete a `Have a nice weekend!` szöveg lesz, ha épp péntek van, egyébként pedig a `Have a nice day!`:

```php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
else
  echo "Have a nice day!";
```

Ha egy utasításnál többet szeretnénk futtatni a feltétel teljesülése/nem teljesülése esetén, az utasításokat kapcsos zárójelek közé kell írnunk:

```php
$d=date("D");
if ($d=="Fri") {
  echo "Hello!<br>";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
}
?>
```

Az `elseif` szerkezet

Ha több feltételből egy teljesülése esetén szeretnénk más-más kódot futtatni, az `elseif` szerkezetet használjuk. Szintaxis:

```php
if (feltétel1)
  végrehajtandó kód, ha a feltétel1 teljesül;
elseif (feltéte2l)
  végrehajtandó kód, ha a feltétel2 teljesül;
else
  végrehajtandó kód, ha egyik feltétel sem teljesül;
```

A középső `elseif` szakasz akár többször is szerepelhet.

Az alábbi példa kimenete `Have a nice weekend!` ha ma péntek van, `Have a nice Sunday!` ha ma vasárnap van, egyébként pedig `Have a nice day!`

```php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
```

Feltételes operátor

Néha előfordul, hogy az `if-else` szerkezet helyett egyetlen operátort, a feltételes operátort használjuk. Ez csak akkor lehet alternatívája az `if-else` használatának, ha mindkét ágon valamilyen értéket akarunk előállítani, és azt a továbbiakban felhasználni.

Nézzünk egy egyszerű példát. A következő kód a `$bar` logikai változó értékét szövegesen állítja elő tesztelési célokból.

```php
$bar = true;
$str = "TEST ". ($bar ? 'true' : 'false') ." TEST";
```

A `switch` szerkezet

A `switch` utasítást a PHP-ben akkor használjuk, hogyha más-más kódot szeretnénk végrehajtani egy változó vagy kifejezés lehetséges értékei alapján. A `switch` használatával bizonyos esetekben elkerülhetőek a hosszú `if-elseif-else` blokkok. A `switch` utasítás általános formája:

```php
switch (kifejezés) {
case cimke1:
  a végrehajtandó kód, ha a kifejezés = cimke1
  break;
case cimke2:
  a végrehajtandó kód, ha a kifejezés = cimke2
  break;
default:
  a végrehajtandó kód
  ha a kifejezés különbözik
  cimke1 és cimke2-től;
}
```

Működése:

- egy egyszerű kifejezés (leggyakrabban egy változó) egyszer kiértékelődik
- a kifejezés értéke összehasonlításra kerül mindegyik `case` kifejezéssel a blokkon belül
- ha egyezést talál, akkor az adott `case`-ben található kód végrehajtódik
- miután ez megtörtént, a `break` utasítás leállítja az utasítások végrehajtását
- a `default` utasítás arra használható, hogy le tudjuk kezelni azokat a kifejezéseket, értékeket is, amelyek egy `case` utasításban sem voltak kezelve

Egy példa:

```php
switch ($x) {
  case 1:
    echo "Number 1";
    break;
  case 2:
    echo "Number 2";
    break;
  case 3:
    echo "Number 3";
    break;
  default:
    echo "No number between 1 and 3";
}
```

Figyelni kell arra, hogy minden egyes `case` ágat zárjunk le egy `break`-kel, különben a következő `case` ág is lefut.

Más nyelvektől eltérően a `case` értékei nem kell, hogy konstansok legyenek. Tetszőleges kifejezés alkalmazható.

Alternatív szintaxis

A PHP a kapcsos zárójelek helyett egy alternatív szintaxist is biztosít. Elsősorban akkor szokás használni, ha keverni akarjuk a PHP és HTML kódot. A következő példa középső sora csak akkor fog a kimenetre kerülni, ha a feltétel igaz.

```php
<?php if ($a == 5): ?>
A is equal to 5
<?php endif; ?>
```

Persze használhatjuk tisztán PHP kódban is:

```php
<?php
if ($a == 5):
  echo "a equals 5";
  echo "...";
elseif ($a == 6):
  echo "a equals 6";
  echo "!!!";
else:
  echo "a is neither 5 nor 6";
endif;
?>
```

Egy másik példa a `switch` használatára:

```php
<div>
<?php switch($variable):
case 1: ?>
<div>Newspage</div>
<?php break;?>
<?php case 2: ?>
<div>Forum</div>
<?php break;?>
<?php endswitch;?>
</div>
```

3.3.2. Ciklusok

Mikor programozunk, nagyon gyakran fordul elő, hogy egy adott programrészt (a ciklusmagot) többször szeretnénk lefuttatni. Ezt elérhetjük ciklusok használatával.

PHP-ben a következő ciklusok használhatók:

- `while` – amíg a feltétel igaz, lefuttatja a programrészt
- `do-while` – először lefuttatja az adott programrészt, és addig ismétli, amíg a feltétel igaz
- `for` – az adott programrészt meghatározott számban futtatja le
- `foreach` – az adott programrészt a tömb minden elemére lefuttatja

Az ismételten lefuttatott programrészt *ciklusmagnak* nevezzük.

A `while` ciklus

A `while` utasítás lefuttatja a ciklusmagot, amíg az adott feltétel igaz. Szintaxis:

```php
while (feltétel)
  ciklusmag
```

Az alábbi példa egy olyan ciklust mutat be, amely addig fut, amíg az `$i` változó kisebb vagy egyenlő, mint 5. Az `$i` értéke a ciklusmag minden lefutása végén 1-el növekszik:

```php
$i=1;
while($i<=5) {
  echo "The number is " . $i . "<br>";
  $i++;
}
```

A `do-while` ciklus

A `do-while` utasítás legalább egyszer lefuttatja az adott ciklusmagot, majd annyiszor ismétli azt, amíg a feltétel igaz. Szintaxis:

```php
do {
  ciklusmag;
} while (feltétel);
```

Az alábbi példaprogram legalább egyszer növeli az `$i` változó értékét, majd addig növeli `$i`-t, amíg annak értéke kisebb, mint 5.

```php
$i=0;
do {
  $i++;
  echo "The number is " . $i . "<br>";
} while ($i<5);
```

A `for` ciklus

A `for` utasítást akkor használjuk, amikor előre tudjuk, hányszor kell futtatni egy adott utasítást vagy utasítás listát. Szintaxis:

```php
for (inicializálás; feltétel; növelés) {
  ciklusmag;
}
```

A `for` utasításnak három eleme van. Az első ad értéket a változóknak, a második tartalmazza a feltételt és a harmadikban van az inkrementáló (növelő) utasítás, ami a ciklust vezérli. Ha több mint egy változó is szerepel az értékadásban vagy az inkrementáló részben, vesszővel válasszuk el őket. A feltétel igaznak vagy hamisnak bizonyul a futás során.

Az alábbi példa ötször írja a képernyőre a `Hello World!` szöveget:

```php
for ($i=1; $i<=5; $i++) {
  echo "Hello World!<br>";
}
```

Érdemes átgondolni, hogy a fenti példa könnyedén átírható `while` ciklussá is:

```php
$i=1;
while ( $i<=5) {
  echo "Hello World!<br>";
  $i++;
}
```

Érdemes még megemlíteni, hogy a `for` ciklus nem csak ebben a speciális, ciklusváltozóhoz kötődő esetben használható. Bármilyen `while` ciklus is átírható `for` ciklussá.

A `foreach` ciklus

A `foreach` utasítást tömbelemekre alkalmazzuk.

A `$value` minden ciklusban az adott tömbelem értékét kapja meg, és a tömb mutató egygyel továbblép – így a következő ciklusban már a következő tömbelemre mutat. Szintaxis:

```php
foreach ($array as $value) {
  ciklusmag
}
```

A `$value` értéke alapértelmezetten egy másolata a tömb elemének. Így annak módosítása nem fogja a tömb adott elemét megváltoztatni.

A következő példa azt a ciklust mutatja be, amely a képernyőre írja az adott tömb értékeit.

```php
$arr=array("one", "two", "three");
foreach ($arr as $value)
  echo "Value: " . $value . "<br>";
```

Bizonyos esetekben szükségünk lehet nemcsak a tömb elemeire, hanem a kulcsokra is. Ekkor a következő szintaxis alkalmazható:

```php
foreach (tömb as kulcs => tömbelem) {
  ciklusmag
}
```

Ebben az esetben a kulcs használatával az eredeti tömbelem is elérhető, akár módosítható is.

A korábban bemutatott `$oldalak` nevű kétdimenziós tömb (3.2.6. fejezet) alapján a következő kód generálja a menüt. Az `if` utasítás az aktív menüpont esetén plusz `class` tulajdonságot is generál.

```php
<div id="topnav">
  <ul>
<? foreach ($oldalak as $url => $oldal) { ?>
    <li<?= (($oldal == $keres) ? ' class="aktiv"' : '') ?>>
      <a href="<?= ($url == '/') ? '.'
                                 : ('?oldal=' . $oldal['fájl'])
        ?>"><?= $oldal['szoveg'] ?></a>
    </li>
<? } ?>
  </ul>
</div>
```

A tömbök mélyebb kezelése

E könyv lehetőségein túlmutat, de mindenképpen érintenünk kell a tömbök mélyebb összefüggéseit, belső működését. Most egyetlen példán keresztül pillantsunk bele abba, hogy a tömbök bejárása igazából sokkal több lehetőséget rejteget, mint az eddigi példáink. Az alábbi két ciklus teljesen ekvivalens:

```php
$arr = array("one", "two", "three");
reset($arr);
while (list($key, $value) = each($arr)) {
    echo "Key: $key; Value: $value<br>\n";
}
foreach ($arr as $key => $value) {
    echo "Key: $key; Value: $value<br>\n";
}
```

A `reset` függvény a tömbmutatót az elejére állítja, az `each` a következőre lépteti. A `list` függvény többszörös értékadást tesz lehetővé egy tömb alapján.

A példában nem szerepel, de érdekes lehetőségeket tartogat a `key`, `current`, `next` és `prev` függvény is.

Referencia használata

Ha a tömb aktuális elemét szeretnénk módosítani, akkor referenciával is bejárhatjuk a tömböt:

```php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$value) {
  $value = $value * 2;
}
```

A futás után a tömb elemei a duplájukra nőttek.

A referencia nélkül csak a tömbelem egy ideiglenes másolatán történne meg a módosítás.

Kilépés a ciklusból

A `break` utasítással ki lehet lépni egy ciklusból, a hagyományostól eltérő módon is. A ciklusmag tetszőleges pontján, ha ráfut a vezérlés a `break` utasításra, akkor egy vagy több ciklusból is kiléphetünk.

```php
foreach($products as $product) {
  foreach($product['Data'] as $data) {
    ...
    break 2;
    ...
  }
}
```

A 2-es szám azt jelzi, hogy mindkét ciklusból ki kell lépnie.

A ciklusmag végrehajtásának megszakítása

A ciklus teljes megszakítása helyett időnként csak az aktuális ciklusmag futtatását akarjuk megszakítani. A `continue` utasítás átugorja a ciklusmag hátralevő részét. A következő kód az `$i` 2-es értéke esetén nem fut rá a `print` függvényre:

```php
for ($i = 0; $i < 5; ++$i) {
    if ($i == 2)
        continue
    print "$i\n";
}
```

Természetesen itt is van lehetőség többszörös ciklus esetén többszörös ciklusmagból kiugrani:

```php
$i = 0;
while ($i++ < 5) {
  echo "Outer<br>\n";
  while (1) {
    echo " Middle<br>\n";
    while (1) {
      echo "  Inner<br>\n";
      continue 3;
    }
    echo "This never gets output.<br>\n";
  }
  echo "Neither does this.<br>\n";
}
```

3.3.3. Függvények használata

A PHP-ben közel ezer beépített függvény érhető el.

Ebben a fejezetben megmutatjuk, hogyan készítsünk saját függvényeket.

Függvények létrehozása

Egy függvény tulajdonképpen egy kód blokk, amit akkor futtathatunk, amikor szükségünk van rá.

Minden függvény létrehozása a `function` kulcsszóval kezdődik.

A függvény nevét lehetőleg úgy válasszuk meg, hogy utaljon rá, milyen feladatot végez el.

Egy egyszerű függvény, amely a szerző nevét írja ki, ha meghívjuk:

```php
function nevKiiras() {
  echo "A nevem Nagy Gusztáv.";
}
```

Függvények használata

A függvényt a PHP kód tetszőleges későbbi pontján meghívhatjuk:

```php
function nevKiiras() {
  echo "A nevem Nagy Gusztáv.";
}
nevKiiras();
```

Függvények paraméterezése

Első függvényünk (`nevKiiras`) egy igen egyszerű függvény. Pusztán egy egyszerű szöveget ír ki.

Ha szeretnénk több funkcióval ellátni egy függvényt, paramétereket adhatunk hozzá. A paraméter olyan mint egy változó. A paramétereket a zárójelen belül adhatjuk meg.

A következő példa különböző keresztneveket ír ki azonos vezetéknevek után.

```php
function nevKiiras($nev) {
  echo "A nevem Nagy $nev.<br>";
}
nevKiiras("Gusztáv");
nevKiiras("Dániel");
nevKiiras("Ábel");
```

A kód kimenete a következő:

```
A nevem Nagy Gusztáv.<br>
A nevem Nagy Dániel.<br>
A nevem Nagy Ábel.<br>
```

A következő függvénynek két paramétere van:

```php
function nevEsFoglalozasKiiras($nev,$foglalozas) {
  echo "A nevem Nagy $nev, $foglalozas vagyok.<br>";
}
nevEsFoglalozasKiiras("Gusztáv", "tanszéki mérnök");
nevEsFoglalozasKiiras("Dániel", "tanuló");
nevEsFoglalozasKiiras("Ábel", "tanuló");
```

A kód kimenetet a következő lesz:

```
A nevem Nagy Gusztáv, tanszéki mérnök vagyok.<br>
A nevem Nagy Dániel, tanuló vagyok.<br>
A nevem Nagy Ábel, tanuló vagyok.<br>
```

Paraméterátadás alapértelmezett értékkel

A függvények paraméterei közül egyet vagy többet elhagyhatunk, ha a következő módon használjuk:

```php
function nevEsFoglalozasKiiras($nev,$foglalozas = "tanuló") {
  echo "A nevem Nagy $nev, $foglalozas vagyok.<br>";
}
nevEsFoglalozasKiiras("Gusztáv", "tanszéki mérnök"); // 1
nevEsFoglalozasKiiras("Dániel"); // 2
nevEsFoglalozasKiiras("Ábel"); // 3
```

Ha a híváskor megadjuk a 2. (`$foglalozas`) paramétert (`1`), akkor az lesz az értéke. Ha nem adjuk meg (`2` és `3`) , akkor pedig az alapértelmezett `"tanuló"` értéket veszi fel.

A kód kimenete a korábbival megegyezik.

Paraméterátadás referenciával

Ha meghívunk egy függvényt, akkor a paraméterként átadott érték másolatát használhatjuk. Nézzünk egy szám faktoriálisát kiszámoló és kiíró függvényt:

```php
function faktorKiir($n) {
  $fakt = 1;
  while ($n > 1)
    $fakt *= $n--;
  echo $fakt;
}
faktorKiir(5);
```

Ha meghíváskor `5`-öt adunk át neki, akkor a függvényen belül `$n` először `5` lesz, majd a ciklusmagban folyamatosan csökken, amíg el nem éri az `1`-es értéket. Eközben pedig `$fakt` értéke `1*5*4*3*2`, azaz `120` lesz.

Ha a faktoriálisok sorozatát szeretnénk kiírni, akár egy ciklusban is hívogathatjuk a függvényünket:

```php
for ($i = 1; $i<5; $i++) {
  echo $i . " faktorialisa: " . faktorKiir($i) . "<br>";
}
```

Természetesen ez nem túl hatékony a megoldás szempontjából, de jól működik, és jól illusztrálja a paraméterátadás mechanizmusát: A `$i` változónak mindig csak egy másolata lesz elérhető a függvényen belül, és ott az `$n` módosítása nem lesz hatással az `$i` értékére.

Néha azonban *előnyös* lenne, ha mégse egy másolatot, hanem az eredeti változót érnénk el. Példaként nézzünk egy (nem szokványos) abszolút érték függvényt. A paraméterként kapott változót átalakítja az abszolút értékére:

```php
function abszolut(&$szam) {
  if ($szam < 0)
    $szam = -$szam;
}
$a = 23;
abszolut($a);
$b = -12;
abszolut($b);
abszolut(-8); // hibás!
```

Ekkor `$a` értéke nem fog változni, de `$b` értéke igen.

Végül érdemes megjegyezni, hogy az utolsó sor fordítási hibát fog okozni.

Függvények visszatérési értéke

A függvények használhatnak visszatérési értékeket is.

```php
function osszead($x,$y) {
  $osszeg = $x + $y;
  return $osszeg;
}
echo "1 + 16 = " . osszead(1,16)
```

A fenti kód kimenete ez lesz:

```
1 + 16 = 17
```

A korábbi faktoriális számító függvényünk helyett érdemes a következő verziót használnunk, mert nem mindig egyből a kimenetre szánjuk a számítás eredményét. Segítségével pl. kiszámíthatjuk az ismétlés nélküli kombinációk számát. A következő kód kiszámítja, hogy 5 különböző kártyalapból hány féleképpen tudunk 3-at kiválasztani, ha a kiválasztás sorrendje nem változik:

PHP
```
function faktor($n) {
  $fakt = 1;
  while ($n > 1)
    $fakt *= $n--;
  return $fakt;
}
echo faktor(5) / (faktor(3) * faktor(5 - 3));
```

Érvényességi kör

Egy függvényen belül létrehozott változó lokális a függvényre nézve. A változó tehát a függvényből kilépve nem látszik. De meglepő módon a fordítottja is igaz: egy függvényben nem látszanak a függvényen kívül létrehozott (globális, vagy más függvénybeli lokális) változók sem. A globális változókat mégis elérhetővé tudunk tenni a `global` kulcsszó használatával.

PHP
```
$a = 1;
$b = 2;
function Sum() {
  global $a, $b;
  $b = $a + $b;
}
```

A `global` sor nélkül a lokálisan létrejövő `$a` és `$b` változóval történne meg az értékadás, majd a függvényből kilépve ezek értéke el is veszne.

Egy másik megoldás a `$GLOBALS` változó használata. Ez ugyanis a globálisan létrehozott összes változót tartalmazza, és bárhonnan elérhető, a következő példának megfelelően. A kódnak ugyanaz a hatása, mint az előzőnek.

PHP
```
$a = 1;
$b = 2;
function Sum() {
   $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}
```

A globális elérések rontják a program áttekinthetőségét, ezért nem érdemes a paraméterátadást ezzel helyettesíteni.

Statikus változók

A függvényen belül létrejövő lokális változó normál esetben nem őrzi meg az értékét. A `static` kulcsszó használatával azonban pont ezt tudjuk elérni.

```php
function hivasSzamol( ) {
  static $db = 0;
  $db++;
  echo "Ezt a függvényt $db alkalommal hívtuk meg.<br>";
}
hivasSzamol();
hivasSzamol();
hivasSzamol();
```

A függvény kimenete:

```php
Ezt a függvényt 1 alkalommal hívtuk meg.<br>
Ezt a függvényt 2 alkalommal hívtuk meg.<br>
Ezt a függvényt 3 alkalommal hívtuk meg.<br>
```

3.4. Adatbázis-kapcsolat

A PHP fejlesztők leggyakrabban MySQL adatbázis-szervert alkalmaznak. Ezért ebben a fejezetben a MySQL használatának alapjait fogjuk áttekinteni.

A téma megismeréséhez mindenképpen szükségesek az adatbázis-kezelési alapismeretek[68].

3.4.1. MySQL alapok

A MySQL ma a legnépszerűbb, bár nem az egyetlen nyílt forrású adatbázis szerver alkalmazás. A szintaxis ugyan eltér, de a logikai háttér hasonló más rendszerek esetén is. A fejezet végén néhány más megoldást is bemutatunk.

Kapcsolódás egy MySQL adatbázishoz

Mielőtt hozzáférnénk és dolgoznánk az adatbázis adataival, egy kapcsolatot kell létrehozni az adatbázishoz. PHP-ben ezt a `mysql_connect()` függvénnyel lehet megvalósítani. Szintaxis:

```php
mysql_connect(servername, username, password);
```

- `servername`: Opcionális. Meghatározza azt a szervert, amihez kapcsolódni akarunk. Az alapbeállítás a következő: `localhost:3306`
- `username`: Opcionális. Meghatározza a MySQL felhasználói nevet, amivel bejelentkezünk.
- `password`: Opcionális. Meghatározza a MySQL felhasználó jelszavát. Az alapbeállítás: `""` (üres sztring).

68 A szerző a témában nagyra becsült tanára, Dr. Katona Endre *Adatbázisok* c. előadásjegyzetét ajánlja az olvasók figyelmébe. Letölthető: http://www.inf.u-szeged.hu/~katona/adatb.htm

Több paraméter is megadható, de a fentiek a legfontosabbak.

A következő példában elmentjük a kapcsolatot egy változóba (`$con`) későbbi használatra. A `die` rész akkor fog végrehajtódni, ha a kapcsolódás nem sikerül:

```php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Nem sikerult kapcsolatot kiepiteni: ' . mysql_error());
}
// ...
```

A kapcsolat bontása

A kapcsolat bontásra kerül, ahogy a szkript futása véget ér. Ha ezelőtt szeretnénk bontani a kapcsolatot, akkor a `mysql_close` függvényt használva tehetjük meg.

```php
mysql_close($con);
```

3.4.2. Adatbázisok és táblák létrehozása

Egy adatbázis egy vagy több táblát tartalmazhat.

A `CREATE DATABASE` parancsot használhatjuk adatbázis létrehozására a MySQL-ben. Szintaxis:

```sql
CREATE DATABASE database_name
```

Hogy a PHP végrehajtsa a fenti parancsot, használnunk kell a `mysql_query` függvényt. Ez elküld egy lekérdezést vagy parancsot a MySQL szervernek.

A *lekérdezés* szót az SQL SELECT utasítása esetén, a *parancs* szót pedig minden más utasítás esetén szokás használni.

A következő példában létrehozunk egy adatbázist `my_db` néven:

```php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Nem sikerult kapcsolatot kiepiteni: ' . mysql_error());
}
if (mysql_query("CREATE DATABASE my_db",$con)) {
  echo "Database created";
} else {
  echo "Sikertelen adatbazis letrehozas: " . mysql_error();
}
mysql_close($con);
```

A `mysql_query` függvény második paramétere opcionális. Akkor kell feltétlenül használnunk, ha egyszerre több adatbázis-kapcsolatot kell igénybe vennünk.

Tábla létrehozása

A `CREATE TABLE` parancs használatával létre tudunk hozni egy táblát a MySQL adatbázisban. Szintaxis:

SQL
```
CREATE TABLE table_name (
  column_name1 data_type,
  column_name2 data_type,
  column_name3 data_type,
...
)
```

A `mysql_query` függvénynek paraméterként kell átadni a `CREATE TABLE` parancssort.

Az adatbázist ki kell választani, mielőtt a táblát létrehozzuk. Az adatbázist a `mysql_select_db` függvénnyel tudjuk kiválasztani. A következő példa megmutatja, hogyan lehet egy `person` nevű táblát létrehozni három mezővel. A mezők nevei: `FirstName`, `LastName` és `Age`:

PHP
```
mysql_select_db("my_db", $con);
$sql = "CREATE TABLE person (
  FirstName varchar(15),
  LastName varchar(15),
  Age int
)";
mysql_query($sql, $con);
mysql_close($con);
```

MySQL adattípusok

A különböző MySQL adattípusok legfontosabb jellemzőivel folytatjuk:

Numerikus adattípusok

```
int(size), smallint(size), tinyint(size), mediumint(size),
bigiynt(size)
```

Csak egész számokat tartalmaz. A maximális számjegyek számát meg lehet határozni a `size` paraméterrel:

```
decimal(size,d), double(size,d), float(size,d)
```

Tizedesvesszőt tartalmazó számok. A maximális számjegyek számát meg lehet határozni a `size` paraméterben. A tizedesvesszőtől jobbra lévő számjegyek maximális számát a `d` paraméterben lehet megadni.

Szöveges adattípusok

```
char(size)
```

Fix hosszúságú sztringet tartalmaz (betűk, számok és speciális karakterek). A fix hosszúságot zárójelben lehet megadni.

```
varchar(size)
```

Változó hosszúságú sztringet tartalmaz (betűk, számok és speciális karakterek). A maximális hosszúságot zárójelben lehet megadni

```
tinytext
```

Változó sztringet tartalmaz egy megadott maximális hosszúsággal, ami 255 karakter.

```
text, blob
```

Változó sztringet tartalmaz egy megadott maximális hosszúsággal, ami 65535 karakter.

```
mediumtext, mediumblob
```

Változó sztringet tartalmaz egy megadott maximális hosszúsággal, ami 16777215 karakter.

```
longtext, longblob
```

Változó sztringet tartalmaz egy megadott maximális hosszúsággal, ami 4294967295 karakter.

Dátum típusú adatok

```
date(yyyy-mm-dd), datetime(yyyy-mm-dd hh:mm:ss), timestamp(yyyymmddhhmmss),
time(hh:mm:ss)
```

Dátumot és/vagy időt tartalmaz

Összetett típusok

```
enum(value1, value2, ect)
```

ENUM az ENUMERATED lista rövid formája. 1-65535 értéket tud raktározni listázva a ( ) zárójelben. Ha egy olyan értéket akarunk beszúrni, ami nincs a listában, úgy egy üres érték fog beszúródni.

```
set
```

A SET (halmaz) hasonló az ENUM-hoz. Azonban a SET 64 listázott tételt tartalmazhat több választási lehetőséggel.

Elsődleges kulcsok és `autoincrement` mezők

A legtöbb táblának célszerű tartalmaznia egy elsődleges kulcs mezőt.

Tipikus kivétel a kizárólag több-több kapcsolatot megvalósító ún. *kapcsoló táblák*.

Az elsődleges kulcs arra szolgál, hogy egyedileg azonosítani lehessen a sorokat a táblában. Minden elsődleges kulcs értékének egyedinek kell lennie a táblán belül. Továbbá az elsődleges kulcsú mező nem lehet `null`, mert az adatbázis (általában) megkíván egy értéket, amely azonosítja a rekordot.

Az adattáblák azon mezőit, amelyek alapján gyakran kell a mezőket visszakeresni, indexelni szokás. Az elsődleges kulcs mező mindig indexelve van.

A következő példa beállítja a `personID` mezőt elsődleges kulcsként. Az elsődleges kulcsú mező gyakran egy ID (azonosító) szám és gyakran van használva az AUTO_INCREMENT beállítással. Ez automatikusan növeli a mező értékét, ha egy új rekord adódik hozzá az eddigiekhez. Hogy biztosítva legyen, hogy az elsődleges kulcs mező nem üres, kötelezően hozzá kell adni a NOT NULL beállítást a mezőhöz.

Példa:

PHP
```
$sql = "CREATE TABLE person (
  personID int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(personID),
  FirstName varchar(15),
  LastName varchar(15),
  Age int
)";
mysql_query($sql,$con);
```

3.4.3. Adatok bevitele adatbázisba

Az `INSERT INTO` paranccsal adatokat illeszthetünk be egy adatbázis táblába. Szintaxis:

SQL
```
INSERT INTO table_name
VALUES (value1, value2,...)
```

Azt is meghatározhatjuk, hogy melyik oszlopba akarjuk az adatot beilleszteni:

SQL
```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,...)
```

Az utóbbi a formát akkor érdemes használni, ha nem minden mezőnek akarunk értéket adni, vagy nem ugyanabban a sorrendben akarjuk az értékeket felsorolni. Az itt értéket nem kapó mezők automatikusan növelt (*auto increment*) vagy alapértelmezett (*default*) értéket is kaphatnak. Ha ilyen előírás nem történt a tábla létrehozásakor, akkor `NULL` (definiálatlan) értéket kap.

Ahhoz, hogy a PHP végrehajthassa a fenti parancsokat, a `mysql_query` függvényt kell meghívni.

Az előző fejezetben egy `Person` nevű táblát hoztunk létre, három oszloppal: `Firstname`, `Lastname` és `Age`. Ugyanezt a táblát használjuk ebben a fejezetben is. A következő példa két új rekordot fűz a `Person` táblához:

PHP
```
mysql_query("INSERT INTO person (FirstName, LastName, Age)
  VALUES ('Peter', 'Griffin', '35')");
mysql_query("INSERT INTO person (FirstName, LastName, Age)
  VALUES ('Glenn', 'Quagmire', '33')");
mysql_close($con);
```

Adatok beillesztése űrlapról egy adatbázisba

Most egy HTML űrlapot fogunk készíteni, melyet új adat bevitelére fogunk használni a `Person` táblába.

Íme a HTML űrlap:

HTML
```
<html>
<body>
  <form action="insert.php" method="post">
    Firstname: <input type="text" name="firstname">
    Lastname: <input type="text" name="lastname">
    Age: <input type="text" name="age">
    <input type="submit" value="Elküld">
  </form>
</body>
</html>
```

Amikor egy felhasználó az `Elküld` gombra kattint a HTML űrlapon, az adatok elküldésre kerülnek az `insert.php` fájlnak. Az `insert.php` fájl egy adatbázishoz kapcsolódik, a `mysql_query` függvény végrehajtja az `INSERT INTO` parancsot, és az új adat beszúrásra kerül az adatbázis táblába.

Az `insert.php` oldal kódja a következő:

PHP
```
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Nem sikerult kapcsolatot kiepiteni: ' . mysql_error());
}
mysql_select_db("my_db", $con);
$sql="INSERT INTO person (FirstName, LastName, Age)
    VALUES
    ('$_POST[firstname]','$_POST[lastname]','$_POST[age]')";
if (!mysql_query($sql,$con)) {
  die('Hiba: ' . mysql_error());
}
echo "1 rekord létrejött";
mysql_close($con)
```

Ez a példa még nem foglalkozik a biztonsági kérdésekkel. *Ellenőrzés nélkül soha nem szabad az adatokat felhasználnunk!* A 3.5.3. fejezetben ezekkel a problémákkal is megismerkedünk.

3.4.4. Lekérdezés

A `SELECT` parancs adatok kiválasztására szolgál egy adatbázisból. Alapvető (nem teljes) szintaxis:

SQL
```
SELECT column_name(s)
FROM table_name
WHERE conditions
```

Ahhoz, hogy a PHP végrehajthassa a fenti parancsokat, a `mysql_query` függvényt kell meghívni.

Az alábbi példa kiválasztja az összes adatot, amely a `Person` táblában van tárolva (A `*` karakter kiválasztja az összes adatot a táblában):

```php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Nem sikerult kapcsolatot kiepiteni: ' . mysql_error());
}
mysql_select_db("my_db", $con);
$result = mysql_query("SELECT * FROM person");
while($row = mysql_fetch_array($result)) {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br>";
}
mysql_close($con);
```

A `mysql_query` függvény által visszaadott erőforrást a `$result` változóban tároljuk. Ezzel az erőforrással érhetőek el a lekérdezés eredményeként kapott rekordok.

A következő példában a `mysql_fetch_array` függvényt használjuk, hogy az első sort megkapjuk tömbként az adathalmazból. Minden későbbi meghívás a `mysql_fetch_array`-ra a következő sorral tér vissza az adathalmazból. Ahhoz, hogy kiírjuk az összes sort, a `$row` változót használjuk (`$row['FirstName']` és `$row['LastName']`).

A `mysql_fetch_array` függvény `false` értéket ad, ha már nincs több rekord. Ezért is alkalmazható ilyen egyszerűen egy `while` ciklusban.

A `mysql_fetch_array` függvény a rekord mezőit asszociatív indexekkel és sorszámokkal is elérhetővé teszi. A példánkban a `$row['FirstName']` helyett `$row[1]`-et is írhattunk volna. Ez a megoldás azonban kevésbé javasolt.

A fenti kód kimenete a következő lesz:

```
Peter Griffin
Glenn Quagmire
```

Eredmények megjelenítése HTML táblában

A következő példa kiválasztja ugyanazokat az adatokat, mint a fenti példa, de az eredményeket HTML táblában fogja megjeleníteni:

```php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Nem sikerult kapcsolatot kiepiteni: ' . mysql_error());}
mysql_select_db("my_db", $con);
$result = mysql_query("SELECT * FROM person");
echo "<table border='1'>
  <tr>
    <th>Vezeteknev</th>
    <th>Keresztnev</th>
  </tr>";
while($row = mysql_fetch_array($result)) {
  echo "<tr>";
  echo "  <td>" . $row['FirstName'] . "</td>";
  echo "  <td>" . $row['LastName'] . "</td>";
  echo "</tr>";
}
echo "</table>";
mysql_close($con);
```

PHP

3.4.5. Rekord feltételek

Ha olyan adatokat szeretnénk kiválasztani, ami valamilyen feltételeknek megfelel, akkor a `SELECT`-hez hozzá kell adnunk egy `WHERE` záradékot. Szintaxis:

```sql
SELECT column FROM table
WHERE condition(s)
```

SQL

A következő operátorok használhatók a `WHERE`-rel:

Operátor	Leírás
`=`	Egyenlő
`!=`	Nem egyenlő
`>`	Nagyobb
`<`	Kisebb
`>=`	Nagyobb, vagy egyenlő
`<=`	Kisebb, vagy egyenlő
`BETWEEN`	Tartományba esés
`LIKE`	Mintával való egyezés

9. táblázat. SQL operátorok

Az SQL parancs PHP-ben való futtatásához a `mysql_query` függvényt kell használnunk.

A következő példa kiválasztja az összes sort a `Person` táblából, ahol `FirstName='Peter'`:

PHP
```
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Nem sikerult kapcsolatot kiepiteni: ' . mysql_error());
}
mysql_select_db("my_db", $con);
$result = mysql_query("SELECT * FROM person
  WHERE FirstName='Peter'");
while($row = mysql_fetch_array($result)) {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br>";
}
```

Eredmény:

```
Peter Griffin
```

3.4.6. A rekordok rendezése

Az `ORDER BY` kulcsszó a lekérdezett adatok rendezésére szolgál. Szintaxis:

SQL
```
SELECT column_name(s)
FROM table_name
ORDER BY column_name
```

A következő példa az eredményt rendezi az `Age` mező szerint:

PHP
```
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Nem sikerult kapcsolatot kiepiteni: ' . mysql_error());
}
mysql_select_db("my_db", $con);
$result = mysql_query("SELECT * FROM person ORDER BY age");
while($row = mysql_fetch_array($result)) {
  echo $row['FirstName'];
  echo " " . $row['LastName'];
  echo " " . $row['Age'];
  echo "<br>";
}
mysql_close($con);
```

Eredmény:

```
Glenn Quagmire 33
Peter Griffin 35
```

Rendezés növekvő és csökkenő sorrendben

Ha az `ORDER BY` kulcsszót használjuk, a rendezés alapértelmezetten növekvő (pl. `1` után `9` és `a` után `p`).

A csökkenő sorrendbe való rendezéshez használjuk a `DESC` szót (pl. `9` után `1` és `p` után `a`):

```sql
SELECT column_name(s)
FROM table_name
ORDER BY column_name DESC
```

Rendezés két vagy több oszlop alapján

A rendezés lehetséges egynél több oszlop alapján is. Ilyenkor a második (és az esetleges további) oszlopot csak akkor lesz figyelembe véve, ha az első mező szerint sorrend nem dönthető el (azaz azonos értékűek).

```sql
SELECT column_name(s)
FROM table_name
ORDER BY column_name1, column_name2
```

3.4.7. Adatok módosítása

Az `UPDATE` utasítás az adatok módosítására szolgál az adatbázis táblában. Szintaxis:

```sql
UPDATE table_name
SET oszlop_nev = uj_ertek
WHERE oszlop_nev = valamilyen_ertek
```

Az előbbi utasítássorozat lefordítására a `mysql_query` függvényt kell használnunk.

A következő példa módosít pár adatot a `Person` táblában:

```php
mysql_query("UPDATE Person SET Age = '36'
    WHERE FirstName = 'Peter' AND LastName = 'Griffin'");
mysql_close($con);
```

A feltöltés után a `Person` tábla kinézete ilyen lesz:

FirstName	LastName	Age
Peter	Griffin	36
Glenn	Quagmire	33

10. táblázat. A Person tábla

A tábla többi rekordját nem akartuk megváltoztatni. Adatvesztéssel járna, ha a `where` záradékot lefelejtenénk, vagy hibásan adnánk meg.

3.4.8. Adatok törlése az adatbázisból

A `DELETE FROM` kifejezés használatos sorok törlésére az adatbázis egy adott táblájából. Szintaxis:

```sql
DELETE FROM table_name
WHERE column_name = some_value
```

A fenti példa PHP szkriptből való végrehajtására a `mysql_query` függvényt használhatjuk.

Az alábbi kódrészlet töröl minden olyan sort a táblából, ahol a `LastName` mező értéke `Griffin`.

```php
mysql_query("DELETE FROM Person WHERE LastName='Griffin'");
mysql_close($con);
```

A végrehajtás után a tábla tartalma:

FirstName	LastName	Age
Glenn	Quagmire	33

`where` záradék

A where záradékban gyakran használt logikai operátor még az `in`, ritkábban az `all`, `any`, `exists` (ezek főleg beágyazott `select`-nél) az `is null` és a `not` is.

3.4.9. Adatbázis absztrakció

Ha a tárhelyszolgáltató nem nyújt MySQL szolgáltatást, vagy a MySQL nem tudja megfelelően kiszolgálni a nagy terhelést, megoldás lehet másik adatbázis-kezelő szerver használata. Többnyire a PostgreSQL[69], Microsoft SQL Server[70], vagy az Oracle Database[71] szokott a megoldás lenni. Esetleg a másik irányban, a még szerényebb szintű szolgáltatás esetén is szükséges lehet a váltás: ekkor az SQLite[72] megoldása is helyettesítheti az adatbázisszervert.

Ezekhez is használhatunk PHP-ben natív illesztő eszközöket, a PHP Manual hosszasan sorolja[73] a lehetőségeket. Ma azonban egyre elterjedtebb megoldás az adatbázis absztrakció alkalmazása.

Nézzünk át az absztrakció néhány fontos előnyét:

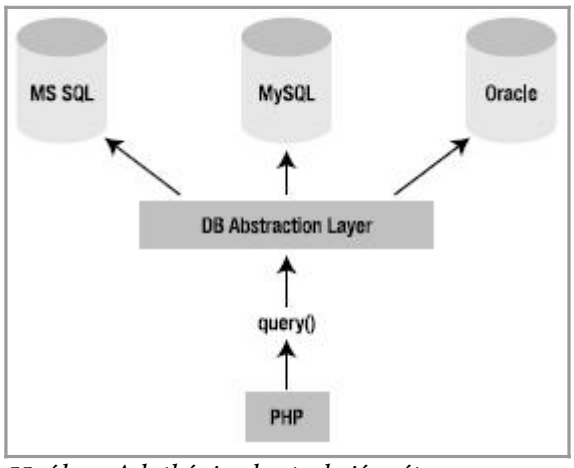
- függetlenedés az adatbázis-kezelő rendszertől (pl. MySQL helyett Oracle használata)
- függetlenedés az adatbázis-kezelő verziótól (az újabb verzió előnyeinek kihasználásához nem kell lemondani a régebbi verziókkal való kompatibilitásról sem)
- az egyes rendszerek specialitásaival nem kell foglalkoznunk, nem kell specialistává válnunk

69 http://www.postgresql.org/
70 http://www.microsoft.com/sqlserver
71 http://www.oracle.com/us/products/database/index.html
72 http://www.sqlite.org/
73 http://hu.php.net/manual/en/refs.database.vendors.php

- az alkalmazás tisztább logikai felépítése (könnyebb tervezés, kódolás, karbantartás)

Az adatbázis absztrakciós réteg

Az adatbázis absztrakció lényege az *adatbázis absztrakciós réteg* fogalmán keresztül érthető meg. Enélkül egy adatbázis alapú alkalmazás a forráskód tetszőleges részein szétszórva tartalmaz olyan kódrészeket, amelyek az adott natív adatbázissal való kommunikációért felelősek. Ezzel szemben az adatbázis absztrakciós réteg segítségével a kommunikáció központilag menedzselve valósulhat meg.



55. ábra. Adatbázis absztrakciós réteg

Erre is létezik többféle megoldás, de talán a legelterjedtebb a *PDO (PHP Data Objects)*.

PDO

Néhány példán keresztül nézzük meg az alapvetőbb PDO[74] szolgáltatásokat.

A megértéshez szükséges lehet a PHP objektumorientált lehetőségeinek (3.8. fejezet) ismeretére.

Kapcsolódás adatbázishoz:

```php
$db = new PDO('mysql:host=localhost;dbname=test', 'user', 'pass');
```

Rekordszintű bejárás:

```php
foreach($db->query('select * from products') as $row) {
  // ...
}
```

Rekord beszúrása, és az új ID kiolvasása:

```php
$db->exec("insert into products (name) values ('Sample')");
$id = $db->lastInsertId();
```

74 http://www.php.net/manual/en/intro.pdo.php

Rekordok módosítása:

```php
$db->exec("update products set name = 'Sample' where id = 3");
```

Preparált lekérdezések

Preparált lekérdezés váza:

```php
$statement = $db->prepare('select * from products');
$statement->execute();
$row = $statement->fetch();
```

A következő példák az SQL befecskendezéses támadás (*SQL injection*[75]) elleni védelem lehetőségeit mutatják be. Alap technika a preparált lekérdezések használata.

Preparált lekérdezés :

```php
$statement = $db->prepare(
    'select * from products where id = ? and category = ?');
$statement->execute(
    array($_GET['id'], $_GET['category']));
while($statement->fetch() as $row) {
  // ...
}
```

Preparált beszúrás:

```php
$statement = $db->prepare(
    'insert into products (name) values(:name)');
$statement->execute(
     array('name' => $_POST['name']));
$newProductId = $db->lastInsertId();
```

Preparált módosítás:

```php
$statement = $db->prepare(
    'update products set name=:name where id=:id');
$statement->execute(
    array('name' => $_POST['name'], 'id' => 123));
```

A PDO használatával kapcsolatban további példákat fogunk megnézni a 3.12. fejezet példáiban.

3.5. Űrlapok használata

Az űrlapok a felhasználótól érkező információk klasszikus, és legfontosabb forrásai.

A PHP `$_GET` és `$_POST` változói használatosak az információ űrlapokból való továbbítására.

75 Bevezetőként a *Weboldalak biztonsága 1: SQL Injection* cikket ajánljuk. Forrás: http://pezia.hu/content/2009/03/08/weboldalak_biztonsága_1_sql_injection

Minden HTML oldalon található űrlapelem automatikusan elérhető a PHP szkriptek számára.

3.5.1. A GET paraméterátadás

A `$_GET` változót arra használjuk, hogy értékeket gyűjtsünk az űrlapról a GET metódussal. A GET-tel mindenki számára látható módon tudunk adatokat küldeni az űrlapból (megjelenik a böngésző címsorában, az URL-ben).

Az elküldhető információ mennyisége korlátozva van.

A pontos korlát több szoftvertől is függ, de maximum 2.000 karakterrel érdemes számolni[76].

Példa:

```HTML
<form action="welcome.php" method="get">
  Név: <input type="text" name="name">
  Életkor: <input type="text" name="age">
  <input type="submit">
</form>
```

Amikor a felhasználó a `submit` gombra kattint, az URL-ben megjelennek az adatok:

```
http://azenoldalam/welcome.php?name=Peter&age=37
```

A `welcome.php` fájl arra használja a `$_GET` változót, hogy elérje az űrlap adatait. Az űrlap mezők nevei automatikusan a `$_GET` tömb azonosítói lesznek:

```PHP
<html>
<body>
  Üdvözöllek, <?php echo $_GET["name"]; ?>!<br>
  Te <?php echo $_GET["age"]; ?> éves vagy.
</body>
</html>
```

Miért használjuk a `$_GET`-et?

Amikor használjuk a $_GET változót, akkor az összes változó neve és értéke megjelenik az URL-ben. Tehát ennek a metódusnak a használata nem ajánlott jelszó és egyéb bizalmas információk küldésekor. Viszont pont emiatt lehetséges könyvjelző elhelyezése. Ez sok esetben hasznos, máskor kifejezetten hátrányos lehet.

A `$_GET` űrlap nélkül

Elsőre talán meglepőnek tűnik, de a `$_GET` tömb elemei nem csak űrlap kitöltésével jöhetnek létre. Semmi akadálya annak, hogy az URL eleve tartalmazzon kulcs-érték párokat, például a következő link eleve ilyen:

```HTML
<a href=”index.php?id=23”>Másik oldal</a>
```

76 Konkrétabb információk: http://www.boutell.com/newfaq/misc/urllength.html

Ebben az esetben űrlap nélkül is lesz tartalma a `$_GET` tömbnek.

Hasonló okokból szokás űrlapba *rejtett* (`hidden`) mezőt elhelyezni. A kitöltött adatokkal együtt ezek az adatok is el fognak jutni a `$_GET` tömbbe.

URL-kódolás

Az URL egy erőforrások megcímzésére alkalmas eszköz kötött karakterkészlettel. Bizonyos karakterek (pl. ékezetes betűk, egyes írásjelek) közvetlenül nem szerepelhetnek az URL-ben, de kódolt formában már igen.

A kódolt karakterek egy `%` jellel és a két számjeggyel leírt hexadecimális értékükkel írhatók le. Például a szóköz kódolva `%20`.

Űrlapok adatainak GET metódussal való elküldése esetén az űrlap adatok az URL-ben kerülnek továbbításra. Ilyen esetben is fontos szerepet kap a kódolás, hiszen egy begépelt űrlapmezőben bármilyen karakter előfordulhat.

A $_REQUEST változó

A PHP $_REQUEST tartalmazza a $_GET, $_POST és $_COOKIE változók elemeit.

Arra használjuk, hogy megkapja a GET és POST metódussal történt adatküldés eredményét.

Példa:

```php
Üdvözöllek, <?php echo $_REQUEST["name"]; ?>!<br>
Te <?php echo $_REQUEST["age"]; ?> éves vagy.
```

PHP

Biztonsági okokból a $_REQUEST használata általában kerülendő. Meg kell ugyanis győződnünk arról, hogy a fontos adatok tényleg onnan érkeztek-e, ahonnan mi várjuk, és nem valahonnan máshonnan.

3.5.2. A POST paraméterátadás

A `$_POST` változó neveket és értékeket tartalmazó tömb, melyek a HTTP POST metódussal lettek továbbítva. Az az információ, melyet egy űrlapról küldenek POST metódussal, láthatatlan a többi felhasználó részére, és nincs korlátozva az információ mennyiségét illetően. Példa:

```html
<form action="welcome.php" method="post">
  Név: <input type="text" name="name">
  Életkor: <input type="text" name="age">
  <input type="submit">
</form>
```

HTML

Amikor a felhasználó a `submit` gombra kattint, az URL nem fog semmilyen űrlap adatot tartalmazni, és a valahogy így fog kinézni:

```
http://azenoldalam/welcome.php
```

Így a `welcome.php` fájl most már használhatja a `$_POST` változót, hogy az űrlap adatokat elérhesse. Az űrlap mezők nevei automatikusan a `$_POST` tömb azonosító kulcsai lesznek:

PHP
```
<html>
<body>
  Üdvözöllek, <?php echo $_POST["name"]; ?>!<br>
  Te <?php echo $_POST["age"]; ?> éves vagy.
</body>
</html>
```

Miért használjunk `$_POST`–ot?

- A HTTP POST-al küldött változók nem láthatók az URL-ben
- A változóknak nincs hosszúsági korlátjuk

Egyébként, mivel az URL-ben nem láthatók a változók, nem lehetséges az oldalakat könyvjelzővel ellátni. (Ez általában nem is lenne praktikus.)

3.5.3. Adatfeldolgozás

A felhasználó által bevitt adatok érvényességét minden esetben vizsgálni kell. A kliens oldali vizsgálat (4.9.1. fejezet) a látogató számára gyorsabb, és csökkenti a szerver terheltségét, de önmagában sosem elegendő, hiszen egy rosszindulatú felhasználó azt könnyedén ki tudja játszani. Ezért az adatok szerver oldali vizsgálata is szükséges, különösen, ha az űrlapnak egy adatbázishoz kell hozzáférnie.

Mit is kaptunk?

Az egyes mezőtípusok esetén talán nem mindig triviális, hogy minek is kellene lennie az egyes szituációkban. Ezért egy egyszerű tesztelési lehetőség a `print_r` függvény használata:

PHP
```
<pre>
<?php
  print_r($_GET);
?>
</pre>
```

Többes adatok

A `select` típusú űrlap elem és `multiple` tulajdonság használata esetén nem csak egy egyszerű értéket kellene visszakapunk, hanem egy tömböt. Ilyen esetben az űrlapot így érdemes felépítenünk:

```HTML
<form action="form.php" method="post">
  <select name="test[]" multiple="multiple">
    <option value="one">egy</option>
    <option value="two">kettő</option>
    <option value="three">három</option>
    <option value="four">négy</option>
    <option value="five">öt</option>
  </select>
  <input type="submit" value="Küldés">
</form>
```

Ekkor a szerver oldalon minden adatot megkapunk, egy tömbként:

```PHP
$test=$_POST['test'];
if ($test){
  foreach ($test as $t)
    echo 'Kiválasztva: ',$t,'<br>';
}
```

Ha `get` paraméterátadást használunk, az URL így fog kinézni:

```
form.php?test[]=two&test[]=three
```

Űrlapok kódolása

Ha UTF-8 karakterkódolással küldjük ki az oldalainkat, akkor célszerű az űrlapok esetén is megadni ezt az információt:

```HTML
<form action="..." method="post" accept-charset="UTF-8">
```

Adatok érvényesítése

A felhasználótól érkező adatokban soha nem bízhatunk meg. Emiatt többféle ellenőrzést kell végeznünk a teljes biztonság érdekében.

Nézzünk meg egy egyszerű űrlapot:

```HTML
<form action="myform.php" method="post">
  <p>Your Name: <input type="text" name="yourname"><br>
    E-mail: <input type="text" name="email"></p>
  <p>Do you like this website?
    <input type="radio" name="likeit" value="Yes" checked="checked"> Yes
    <input type="radio" name="likeit" value="No"> No
    <input type="radio" name="likeit" value="Not sure"> Not sure</p>
  <p>Your comments:<br>
    <textarea name="comments" rows="10" cols="40"></textarea></p>
  <p><input type="submit" value="Send it!"></p>
</form>
```

Ha az űrlap adatokat egyszerűen eltároljuk és/vagy később felhasználjuk, meglepetéseket tapasztalhatunk. Nézzük meg a `myform.php` egyszerű verzióját:

```php
<html>
<body>
  Your name is: <?php echo $_POST['yourname']; ?><br>
  Your e-mail: <?php echo $_POST['email']; ?><br>
  <br>
  Do you like this website? <?php echo $_POST['likeit']; ?><br>
  <br>
  Comments:<br>
  <?php echo $_POST['comments']; ?>
</body>
</html>
```

Ekkor ilyen eredményre számítunk:

```
Your name is: John Doe
Your email: john@doe.com
Do you like this website? Yes
Comments:
This is my comment...
```

De mi történik akkor, ha a látogató egy kis JavaScript kódot ír be a beviteli mezőbe:

```html
<script>location.href('http://www.SPAM.com')</script>
```

Ha ezt kiírjuk a fenti `myform.php` kóddal, az összes látogatót elküldjük a fenti webcímre. Ez megengedhetetlen. A következő kód a `htmlspecialchars` függvényt használja a támadások kivédésére:

```php
<?php
  $yourname = htmlspecialchars($_POST['yourname']);
  $email    = htmlspecialchars($_POST['email']);
  $likeit   = htmlspecialchars($_POST['likeit']);
  $comments = htmlspecialchars($_POST['comments']);
?>
<html>
<body>
  Your name is: <?php echo $yourname; ?><br>
  Your e-mail: <?php echo $email; ?><br>
  <br>
  Do you like this website? <?php echo $likeit; ?><br>
  <br>
  Comments:<br>
  <?php echo $comments; ?>
</body>
</html>
```

Ekkor a fenti JavaScript kódból ez a közömbösített kód lesz:

```
&lt;script&gt;location.href('http://www.SPAM.com')&lt;/script&gt;
```

Ezen kívül további függvények is szóba jöhetnek az adatok megtisztítására. Ezek egységes használatára nézzünk egy példát:

```php
function check_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
$yourname = check_input($_POST['yourname']);
$email    = check_input($_POST['email']);
...
```

Kötelező mezők kezelése

Fejlesszük tovább a `check_input` függvényt. Ha egy mezőt kötelezően ki akarjuk töltetni, akkor ezt az opciót egy második paraméterrel beépíthetjük a függvénybe: ha átadjuk a szöveges paramétert, akkor ezzel jelezzük, hogy kötelező a kitöltés, és ekkor ez lesz a kiadandó hibaüzenet is:

```php
function check_input($data, $problem='')
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    if ($problem && strlen($data) == 0)     {
        die($problem);
    }
    return $data;
}
```

A következő felhasználás esetén az első és negyedik mező kitöltése kötelező, a középsőké nem:

```php
$yourname = check_input($_POST['yourname'], "Enter your name");
$email    = check_input($_POST['email']);
$likeit   = check_input($_POST['likeit']);
$comments = check_input($_POST['comments'], "Write your comments");
```

Persze a hibaüzenetek lekezelésére (`die` függvény) ez csak egy első megoldás, tovább kell fejlesztenünk.

Reguláris kifejezések

A szintaxis-ellenőrzés régi bevált módszere a reguláris kifejezések használata. Itt most néhány példát fogunk megnézni. A hangsúly nem a reguláris kifejezések működésén, hanem azok PHP felhasználásán lesz.

Az e-mail cím ellenőrzésére egy lehetséges megoldás:

```php
$email = htmlspecialchars($_POST['email']);
if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$email)) {
    die("E-mail address not valid");
}
```

URL ellenőrzése:

```php
$url = htmlspecialchars($_POST['website']);
if (!preg_match("/^(https?:\/\/+[\w\-]+\.[\w\-]+)/i",$url)) {
    die("URL address not valid");
}
```

Számjegyek:

```php
if (preg_match("/\D/",$age)) {
    die("Please enter numbers only!");
}
```

Angol abc betűi:

```php
if (preg_match("/[^a-zA-Z]/",$text)) {
    die("Please enter letters a-z and A-Z only!");
}
```

Űrlapok feldolgozása helyben

Az adatok ellenőrzésére bevett módszer, hogy az űrlap adatait az űrlapot is tartalmazó szkript dolgozza fel, így nincs szükség arra, hogy az adatokat egy újabb oldalra küldjük. Ebben az esetben a felhasználó a hibaüzeneteket ugyanazon az oldalon láthatja, ahol az űrlap található, így megkönnyítve a hiba felfedezését és kijavítását.

Nézzük az alapokat. Legyen az oldal neve `form-action.php`.

```php
<?php
  if(isset($_POST['submit'])) {
    $name = $_POST['name'];
    echo "User Has submitted the form and entered this name : <b> $name
</b>";
    echo "<br>You can use the following form again to enter a new name.";
}
?>
<html>
<head><title>Using PHP_SELF</title></head>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
  <input type="text" name="name"><br>
  <input type="submit" name="submit" value="Submit Form"><br>
</form>
</body>
</html>
```

Először is meglepő, hogy nem az űrlappal kezdjük a forráskódot. Itt a háttérben már körvonalazódik az a – később egyre fontosabbá váló – logika, ami szerint a lehető legjobban válasszuk szét a program logikáját a megjelenítési részektől.

A kód elején el kell döntenünk, hogy most először akarja látni a látogató az oldalt, és üres űrlappal kell várnunk, vagy pedig az űrlap már kitöltve érkezett vissza a látogatótól. Az első esetben nem fog semmilyen POST adat érkezni. A második esetben viszont igen: a `Submit Form` gomb lenyomásakor már a POST adatok is elküldésre kerültek. Emiatt az `isset` függvény alkalmas a két eset megkülönböztetésére.

A `form action` paraméterét szokás a fenti módon kitölteni. A `$_SERVER['PHP_SELF']` ugyanannak az oldalnak a címe, amely a fenti PHP kódot tartalmazza.

Itt is számítanunk kell azonban egy betörési lehetőségre. Ha a látogató a böngészője cím sorába beírja a következő címet, a $_SERVER['PHP_SELF'] változó egy ügyesen elhelyezett támadókódot tartalmaz.

```
http://www.yourdomain.com/form-action.php/%22%3E%3Cscript%3Ealert('xss')
%3C/script%3E%3Cfoo%22
```

A PHP futás eredménye ekkor:

HTML
```
<form name="test" method="post" action="form-action.php"/>
<script>alert('xss')</script><foo"">
```

Az XSS támadás elkerülése érdekében inkább így használjuk:

PHP
```
<form name="test"
  action="<?php echo htmlentities($_SERVER['PHP_SELF']); ?>"
  method="post">
```

Hibaüzenetek és javítási lehetőségek

A látogatók sokszor nem tudják hiba nélkül kitölteni az űrlapot. Emiatt fontos az alábbi elvek figyelembe vétele:

1. az űrlap eleve tartalmazzon minden információt, ami a kitöltéssel kapcsolatos
2. kliens oldalon ellenőrizzük az adatokat JavaScript segítségével
3. beszédes hibaüzenetekkel lássuk el a látogatót, a hiba helyét könnyen beazonosítható módon
4. adjuk vissza a látogatónak a megadott adatait, hogy ne kelljen mindent elölről kezdenie

A korábbi példáink a hibaüzeneteket eléggé mostohán kezelték. Többször használtunk olyan kódot, amely az első hibánál leáll. De a használhatóság miatt inkább össze kell gyűjtenünk a hibaüzeneteket, és a megfelelő helyen kiírni.

Nézzünk egy minimális példát a hibaüzenetek összegyűjtésére. A változók az űrlapról érkező adatokat tartalmazzák:

PHP
```
$errorMessage = '';
if(empty($varMovie)) {
  $errorMessage .= "<li>You forgot to enter a movie!</li>";
}
if(empty($varName)) {
  $errorMessage .= "<li>You forgot to enter a name!</li>";
}
if(empty($varGender)) {
  $errorMessage .= "<li>You forgot to select your Gender!</li>";
}
```

Jól látszik, hogy a futás végére az `$errorMessage` vagy üres marad, és ekkor nem volt probléma, vagy nem marad üres, és felsorolásként tartalmazza a hibaüzeneteket, és már csak ki kell írnunk egy `ul` elembe. Még szebb megoldás lenne, ha közvetlenül a mező mellett jelenne meg a hibaüzenet. (A 3.10.1. fejezetben látni fogunk erre is példát.)

Ha az űrlap feldolgozása során azt láttuk, hogy nem tökéletes, és újból a látogató felé kell továbbítani, akkor *vissza kell küldenünk a korábban kitöltött adatokat*. Erre egy egyszerű példa:

PHP
```
<input type="text" name="Coal"
  <?php if (isset($_POST['Coal'])) { ?>
    value="<?php echo $_POST['Coal']; ?>"
  <?php } ?>
/>
```

Ha a látogató már kitöltötte a `Coal` mezőt, az `isset($_POST['Coal'])` igaz lesz, és kiírjuk a `value` értékeként.

Érdemes azt is átgondolni, hogy az `if`-nek inkább olvashatósági jelentősége van, nélküle is megfelelő lenne a kimenet:

- ha a feltétel teljesülne, akkor nincs változás,
- ha a feltétel nem teljesül, akkor a `value` üres értékkel jelenik meg.

3.5.4. Állományok feltöltése

PHP-vel lehetséges fájlok szerverre történő feltöltése is. Leggyakrabban képeket szokás feltölteni, amelyek a szövegbe illesztve fognak fontos szerepet betölteni. De fájlmegosztás céljából (pl. egy PDF dokumentumot), vagy adminisztrációs célokból (adatok tömeges importálása CSV állományból) történő fájlfeltöltés is előfordulhat.

Fájlfeltöltő űrlap készítése

Nézzük meg az alábbi fájl feltöltésre használt HTML űrlapot:

HTML
```
<html>
<body>
  <form action="upload_file.php" method="post"
    enctype="multipart/form-data">
    <label for="file">Filename:</label>
    <input type="file" name="file" id="file">
    <br>
    <input type="submit" name="submit" value="Submit">
  </form>
</html>
</body>
```

Jegyezzük meg a következőket a HTML űrlappal kapcsolatban:

- A `form` `enctype` attribútuma határozza meg, hogy melyik tartalom típust használjuk, amikor az űrlapot elfogadjuk. A `multipart/form-data` akkor használatos, ha az űrlap bináris adatot vár, mint pl. egy fájl tartalma feltöltéskor.
- Az input elem `file` típusa határozza meg, hogy a függvény bemenete fájlként legyen feldolgozva. Például böngészőben való megjelenítéskor lesz egy `Tallózás` gomb a bemenet mező mellett.

Feltöltés engedélyezése felhasználók számára nagy kockázattal jár. Kizárólag megbízható felhasználók számára tegyük lehetővé fájlok feltöltést, és feltöltés után is ellenőrizzük a fájlt.

Feltöltő szkript készítése

Az `upload_file.php` fájl tartalmazza a fájl feltöltésére és az ellenőrzésre szolgáló programot:

PHP
```
if ($_FILES["file"]["error"] > 0) {
  echo "Error: " . $_FILES["file"]["error"] . "<br>";
} else {
  echo "Upload: " . $_FILES["file"]["name"] . "<br>";
  echo "Type: " . $_FILES["file"]["type"] . "<br>";
  echo "Size: " . ($_FILES["file"]["size"] / 1024)
       . " Kb<br>";
  echo "Stored in: " . $_FILES["file"]["tmp_name"];
}
```

A `$_FILES` globális PHP tömb használatával fájlokat tölthetünk fel egy kliens gépről a távoli szerverre. Az első index az űrlap mező neve, a második lehet `name`, `type`, `size`, `tmp_name` vagy `error`, a következőképpen:

- `$_FILES["file"]["name"]`: a feltöltött fájl neve
- `$_FILES["file"]["type"]`: a feltöltött fájl típusa
- `$_FILES["file"]["size"]`: a feltöltött fájl mérete byte-okban
- `$_FILES["file"]["tmp_name"]`: a szerveren tárolt fájl másolatának átmeneti neve
- `$_FILES["file"]["error"]`: a fájl feltöltése során kapott hibakód

Ez a fájlfeltöltés egy nagyon egyszerű módja. Biztonsági okokból korlátozásokat kell bevezetni a felhasználóknál a feltöltésekre vonatkozóan.

A feltöltés korlátozása

Ebben a szkriptben a fájlfeltöltést korlátokhoz kötjük. A felhasználó kizárólag `.gif` vagy `.jpeg` fájlokat tölthet fel, és a fájlméretnek `20 Kb` alatt kell maradnia:

```php
if ((($_FILES["file"]["type"] == "image/gif")
    || ($_FILES["file"]["type"] == "image/pjpeg"))
      && ($_FILES["file"]["size"] < 20000)) {
  if ($_FILES["file"]["error"] > 0) {
    echo "Error: " . $_FILES["file"]["error"] . "<br>";
  } else {
    echo "Upload: " . $_FILES["file"]["name"] . "<br>";
    echo "Type: " . $_FILES["file"]["type"] . "<br>";
    echo "Size: " . ($_FILES["file"]["size"] / 1024)
         . " Kb<br>";
    echo "Stored in: " . $_FILES["file"]["tmp_name"];
  }
} else {
  echo "Invalid file";
}
```

A feltöltött fájl elmentése

A fenti példák elkészítik a szerveren a feltöltött fájlok egy-egy átmeneti másolatát a PHP `temp` mappában.

A `temp` mappa egy logikai kifejezés, ténylegesen a `php.ini` beállításától függ. XAMPP esetén `c:\xampp\tmp`.

Az átmeneti fájlok törlésre kerülnek, amint a szkript futása véget ér. Ahhoz, hogy a fájlt eltároljuk, át kell másolnunk máshová. Ehhez a `move_uploaded_file` függvényt kell használnunk:

```php
if (($_FILES["file"]["type"] == "image/gif")
      || ($_FILES["file"]["type"] == "image/pjpeg")
    && ($_FILES["file"]["size"] < 20000)) {
  if ($_FILES["file"]["error"] > 0) {
    echo "Return Code: " . $_FILES["file"]["error"]
         . "<br>";
  } else {
    echo "Upload: " . $_FILES["file"]["name"] . "<br>";
    echo "Type: " . $_FILES["file"]["type"] . "<br>";
    echo "Size: " . ($_FILES["file"]["size"] / 1024)
         . " Kb<br>";
    echo "Temp file: " . $_FILES["file"]["tmp_name"]
         . "<br>";
    if (file_exists("upload/" . $_FILES["file"]["name"])) {
      echo $_FILES["file"]["name"] . " already exists. ";
    } else {
      move_uploaded_file($_FILES["file"]["tmp_name"],
      "upload/" . $_FILES["file"]["name"]);
      echo "Stored in: " . "upload/"
           . $_FILES["file"]["name"];
    }
  }
} else {
  echo "Invalid file";
}
```

A fenti szkript leellenőrzi, létezik-e már a fájl, ha pedig nem, bemásolja azt `upload` nevű mappába.

Ezzel a feltöltés folyamata eredményessé vált.

3.5.5. Levélküldés

A PHP lehetővé teszi, hogy közvetlenül szkriptből küldjünk e-maileket. Néhány ok, amiért e-mailt szoktunk küldeni egy weboldalról:

- regisztráció, hírlevél feliratkozás stb. esetén megerősítő levél
- hírlevél a feliratkozott látogatóknak
- értesítés valamilyen esemény bekövetkezéséről

A PHP `mail` függvényt arra használják, hogy segítségével szkripten belül e-maileket küldjenek. Szintaxis:

```php
mail(to, subject, message, headers, parameters)
```

- `to`: Kötelezően megadandó. Az e-mail címzettjét/címzetteit határozza meg.
- `subject`: Kötelezően megadandó. Az email témája. Ez a paraméter nem tartalmazhat új sor (`LF`, `\n`) karaktereket.
- `Message`: Kötelezően megadandó. Az elküldendő üzenetet határozza meg. Minden új sort LF (`\n`) karakterrel kell elválasztani. Egy sor terjedelme nem haladhatja meg a 70 karaktert.
- `headers`: Opcionális. További fejléceket adhatunk meg, mint például `From`, `Cc` és `Bcc`. A további fejléceket `CRLF` (`\r\n`) karakterrel kell elválasztani.
- `parameters`: Opcionális. További paramétereket adhatunk meg a `sendmail` programnak

A `mail` függvény használatához a PHP-nek szüksége van egy feltelepített és működő levelezőrendszerre, vagy SMTP kapcsolatra. A használandó programot a `php.ini` fájl konfigurációs beállításai határozzák meg (3.1.2. fejezet).

Egyszerű megoldás

A legegyszerűbb levélküldési mód PHP-ben a szöveges levél küldése.

Az alábbi példában először deklaráljuk a változókat (`$to`, `$subject`, `$message`, `$from`, `$headers`), majd a `mail` függvényben felhasználjuk ezeket a változókat e-mailküldésre:

```php
$to = "valaki@levele.com";
$subject = "Teszt levél";
$message = "Szia! Ez egy egyszerű levél.";
$from = "valakimas@levele.com";
$headers = "From: $from";
mail($to,$subject,$message,$headers);
echo "A levél elment.";
```

PHP levél űrlapból

A PHP segítségével létre tudunk hozni egy visszajelző (más néven kapcsolatfelévteli) űrlapot weboldalunkon. A következő példában egy szöveges üzenetet küldünk el a megadott e-mail címre.

```php
<html>
<body>
<?php
if (isset($_POST['email'])) {
  $email = $_POST['email'] ;
  $subject = $_POST['subject'] ;
  $message = $_POST['message'] ;
  mail( "valaki@levele.com", "Subject: $subject",
  $message, "From: $email" );
  echo "Köszönjük üzenetét.";
} else {
  echo "<form method='post' action='mailform.php'>
  Email: <input name='email' type='text'><br>
  Tárgy: <input name='subject' type='text'><br>
  Üzenet:<br>
  <textarea name='message' rows='15' cols='40'>
  </textarea><br>
  <input type='submit'>
</form>";
}
?>
</body>
</html>
```

Először megvizsgáljuk, hogy az email beviteli mező ki van-e töltve. Ha nincs (például ha ez az első látogatás az oldalon), akkor megjelenítjük a HTML űrlapot.

Ha ki van töltve, elküldjük az űrlapból az emailt. Ha az űrlap kitöltése után nyomják meg az elküld gombot, a lap újratöltődik, megnézi, hogy az email beviteli mező ki van-e töltve, majd elküldi az e-mailt.

Biztonságos levélküldés

Az előző fejezetben bemutatott `mail` függvénynek van egy gyenge pontja: lehetőséget ad befecskendezéses támadás (injection) megvalósítására.

A probléma a fent bemutatott kóddal, hogy illetéktelen felhasználók be tudnak szúrni adatokat a levél fejlécébe a beviteli űrlapon keresztül.

Mi történik, ha a felhasználó a következő szöveget írja be az email beviteli mezőbe az űrlapon?

```
someone@example.com%0ACc:person2@example.com
%0ABcc:person3@example.com,person3@example.com,
anotherperson4@example.com,person5@example.com
%0ABTo:person6@example.com
```

A `mail` függvény a szokott módon a fenti szöveget helyezi el a levél fejlécében, és így a fejléc többlet `Cc:`, `Bcc:`, és `To:` mezőket tartalmaz. Amikor a felhasználó az elküld gombra kattint, a fenti összes címre elküldi a levelet. Így a támadó olyan helyről küldött e-mailt, amely eddig nem volt letiltva spam gyanú miatt.

Emiatt mindenképpen *célszerű reguláris kifejezésekkel ellenőrizni*, hogy az egyes mezők megfelelnek-e a kívánalmaknak.

3.6. Állománykezelés

Mint minden programozási környezetben, PHP-ben is igen gyakori, hogy egyszerre több állománnyal dolgozunk. Egyrészt a forráskódot szokás külön állományokba szervezni, másrészt igen gyakran dolgozunk bináris vagy szöveges adatállományokkal is.

3.6.1. Forráskód beillesztése

Az `include` függvény használata esetén a PHP lényegében bemásolja a sor helyére a paraméterként megadott fájlt, és – ha tartalmaz PHP kódot – lefuttatja azt.

Az `include` függvény

Tegyük fel, hogy az oldalainkra ugyanazt a menüt akarjuk beszúrni:

PHP
```
<html>
<body>
  <a href="default.php">Címlap</a> |
  <a href="about.php">Magamról</a> |
  <a href="contact.php">Kapcsolat</a>
...
```

Ha mindhárom oldalra (`default.php`, `about.php`, `contact.php`) ugyanezt a menüt akarjuk alkalmazni, akkor hagyományos megoldással mindhárom oldalra kézzel be kell illesztenünk. Ez azonban több okból sem szerencsés. Ehelyett tegyük a fenti kódot egy külön `menu.php` fájlba, majd minden további oldalról csak illesszük be. Például a `default.php`:

PHP
```
<?php include("menu.php"); ?>
  <h1>Üdvözlöm honlapomon!</h1>
  <p>További szöveg...</p>
</body>
</html>
```

Természetesen ha megnézzük a böngészőben az állományt, akkor ott a beillesztett tartalmat fogjuk látni:

```
<html>                                                                PHP
<body>
  <a href="default.php">Címlap</a> |
  <a href="about.php">Magamról</a> |
  <a href="contact.php">Kapcsolat</a>
  <h1>Üdvözlöm honlapomon!</h1>
  <p>További szöveg...</p>
</body>
</html>
```

A `require` függvény

Ez a függvény nagyon hasonlít az előzőre: mindössze a hibakezelésben van köztük eltérés. Az `include` `warning`-ot, vagyis figyelmeztetést ad, ha a beszúrt állományban szintaktikai hiba van, de a hívás utáni kód tovább fut, míg a `require` `error`-t, vagyis hibát ad, és a hívó kód futását is megszakítja.

Tegyük fel, hogy a `wrongFile.php` hibás kódot tartalmaz.

```
<html>                                                                PHP
<body>
<?php
  include("wrongFile.php");
  echo "Hello World!";
?>
</body>
</html>
```

Az `include` miatt a futás nem szakad meg a hibánál, a hibaüzenetek után az `echo` is lefut:

```
Warning: include(wrongFile.php) [function.include]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5
Warning: include() [function.include]:
Failed opening 'wrongFile.php' for inclusion
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5
Hello World!
```

Ha az `include`-ot kicseréljük `require`-re, akkor a futás megszakad, és a következő eredményt kapjuk:

```
Warning: require(wrongFile.php) [function.require]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5
Fatal error: require() [function.require]:
Failed opening required 'wrongFile.php'
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5
```

A céljainknak megfelelően kell eldöntenünk, hogy melyik megoldásra van szükségünk. Pl. ha az állományban a későbbiekben szükséges függvény- vagy osztálydefiníciók találhatóak, akkor a `require` használata célszerű.

Egyedi beillesztés

Van, amikor szándékosan illesztünk be egy állományt többször (pl. egy ciklus belsejében), de van, amikor a többszörös beillesztés káros lenne (pl. függvény- vagy osztálydefiníció esetén), vagyis el kell azt kerülni.

A többszörös beillesztés elkerülése végett alkalmazhatjuk az `include_once` és `require_once` függvényeket, amelyek hatására az esetleges többszörös beillesztés nem fog megtörténni.

3.6.2. Egyszerű Front Controller megoldások

A *Front Controller* tervezési minta a weboldalhoz érkezett kérés feldolgozásához és a kérések kiszolgálásának vezérléséhez használt módszer. (A Front Controller nagyon közel áll az *MVC* minta *Controller* eleméhez.)

A tervezési mintákról bővebben a 3.10. fejezetben beszélünk. A 3.10.2. fejezetben egy komplexebb Front Controller megoldást is bemutatunk.

Statikus megoldás

A statikus megoldás lehet pl. a következő:

PHP
```
<?php // index.php
switch ( @$_GET['action'] ) {
  case 'edit':
    include ('actions/edit.php'); break;
  case 'post':
    include ('actions/post.php'); break;
  case 'delete':
    include ('actions/delete.php'); break;
  case 'default':
    include ('actions/view.php'); break;
} ?>
```

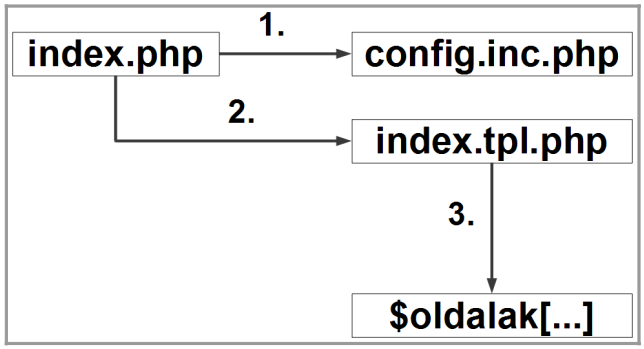
Jól látható, hogy itt a lehetőségeket előre beleégetjük a forráskódba.

Persze ez nem mindig probléma, további paraméterezéssel jó megoldás lehet. A 3.12. fejezet példája is hasonló.

Dinamikus megoldás

A dinamikus megoldás nem igényli a lehetőségek (az előző példában akciók) felsorolását, hanem valamely más forrásból (pl. fájlrendszer, adatbázis, konfigurációs adatok) deríti ki az érvényes kérések körét.

Nézzünk meg egy egyszerű változatot, ami konfigurációs fájlban, tömbben tárolja az egyes oldalak adatait. (A példa könnyen továbbfejleszthető abba az irányba, hogy tömb helyett adatbázisból olvassuk ki az oldalaink adatait.) A példa 3 egyedi és több hasonló forrásfájlból épül fel. Az 56. ábra a forrásállományok `include` struktúráját, és a beillesztések sorrendjét (1-3) mutatja.



56. ábra. A forrásállományok include-olása

A `config.inc.php`

Ebben a fájlban olyan konfigurációs adatokat helyezünk el, amelyek minden oldallekérés esetén szükségesek lesznek.

A fejléc a `title` és az oldal főcíme számára is tartalmaz információt:

```php
$fejléc = array(
  'cím' => 'Nagy Gusztáv',
  'mottó' => 'Szakmai és személyes oldal'
);
```

Az `$oldalak` tömb a főmenüben szereplő menüpontokat és a hozzájuk tartozó oldalak adatait tartalmazza. Első eleme a kezdőoldalra utal.

```php
$oldalak = array(
  '/' =>
    array('fájl' => 'kezdo', 'szoveg' => 'Kezdőlap'),
  'magamrol' =>
    array('fájl' => 'magamrol', 'szoveg' => 'Magamról'),
  'szakmai_oneletrajz' =>
    array('fájl' => 'szakmai_oneletrajz', 'szoveg' =>
      'Szakmai önéletrajz'),
  'galeriak' =>
    array('fájl' => 'galeriak', 'szoveg' => 'Galériák'),
  'kapcsolat' =>
    array('fájl' => 'kapcsolat', 'szoveg' => 'Kapcsolat'),
);
```

Végül a hibaoldal számára is definiálunk információkat:

```php
$hiba_oldal =
  array ('fájl' => '404',
    'szoveg' => 'A keresett oldal nem található');
```

Természetesen ha további, a főmenün kívüli oldalak is lesznek a honlapunkon, akkor újabb tömb segítségével azon is definiálhatók lennének. (Persze ekkor az `index.php` is összetettebb kell legyen.)

Az `index.php`

Az `index.php` feladata a honlap feldolgozási folyamatának indítása, vezérlése. Először is a konfigurációs adatokat töltjük be:

```php
include('config.inc.php'); // 1.
```

A `$keres` változóba állítjuk elő, hogy melyik oldal lekérése is történt. Először az oldal GET paraméter hiányára készülünk fel, majd az `$oldal` tömb alapján érvényesítjük a kérést. A `current` tömbfüggvény az `$oldalak` aktuális elemét adja vissza. Mivel a tömböt nem jártuk be, az aktuális elem a tömb legelső eleme, vagyis a címlap.

```php
$keres = current($oldalak);
if (isset($_GET['oldal'])) {
  if (isset($oldalak[$_GET['oldal']])) {
    $keres = $oldalak[$_GET['oldal']];
  } else {
    $keres = $hiba_oldal;
    header("HTTP/1.0 404 Not Found");
  }
}
```

Látszik, hogy hibás kérésre is fel kell készülnünk.

Végül az `index.tpl.php` sablon fájl fejezi be a feladatot:

```php
include('index.tpl.php'); // 2.
```

Az `index.tpl.php`

Az `index.tpl.php` fájl a honlap oldalról oldalra változatlan sablonját, valamint a változó részek logikáját valósítja meg. A következő `head` rész példája jól mutatja a két rész együttesét:

```php
<head>
  <meta http-equiv="Content-Type"
      content="text/html; charset=utf-8">
  <title>
    <?= $fejléc['cím'] .
      ( (isset($fejléc['mottó'])) ? ('|' . $fejléc['mottó']) : '' ) ?>
  </title>
  <link rel="stylesheet" href="style.css" type="text/css">
</head>
```

A minden oldalon egyformán megjelenő HTML tagok között szereplő PHP rész a `config.inc.php`-ből származó adatokat helyettesíti be.

Persze egyszerű behelyettesítés mellett a PHP valamivel összetettebb feladatot is megoldhat. Például a következő ciklus és feltételes utasításokkal pont az igényeknek megfelelő eredményt produkálhatunk:

PHP
```
<div id="header">
  <h1>
    <?= $fejléc['cím'] ?>
  </h1>
```

Ha a mottó is meg van adva, akkor kiírjuk a `h2` tagokkal együtt:

PHP
```
<? if (isset($fejléc['mottó'])) { ?>
  <h2>
    <?= $fejléc['mottó'] ?>
  </h2>
<? } ?>
</div>
```

A felső menüt az `$oldalak` tömb alapján generáljuk. Ha az aktuális tömbelem (`$oldal`) egyenlő a kérésben (`$keres`) szereplő oldallal, akkor a kimenetbe az `aktiv class` tulajdonságot is belegeneráljuk:

PHP
```
<div id="topnav">
  <ul>
<? foreach ($oldalak as $url => $oldal) { ?>
    <li<?= (($oldal == $keres) ? ' class="aktiv"' : '') ?>>
      <a href="<?= ($url == '/') ? '.' : ('?oldal=' . $oldal['fájl']) ?>">
          <?= $oldal['szoveg'] ?></a>
    </li>
<? } ?>
  </ul>
</div>
```

Az egyes oldalak

Az egyes oldalak változó törzse jelen példánkban az oldal könyvtárban található. Az `index.tpl.php` következő része illeszti be az oldalba:

PHP
```
<div id="body">
<?
  include("oldal/{$keres['fájl']}.tpl.php");
?>
</div>
```

Példánk végén érdemes átgondolni e verzió sablon elvű megközelítését.

Természetesen érdemes a példa hiányosságait is áttekinteni:

- az adatok nem adatbázisból, hanem fájlrendszerből származnak, és
- csak egyszerű főoldal és főmenü struktúrát valósít meg.

Érdemes a témában *Chris Corbyn* objektumorientált megoldását (3.10.2. fejezet) is tanulmányozni, amely egy egyszerű vendégkönyv alkalmazás vázlatát mutatja be.

3.6.3. Fájlok egészként kezelése

A PHP igen sok lehetőséget nyújt arra, hogy az állomány egészével kapcsolatos műveletet végezzünk. A teljesség igénye nélkül néhány fontosabb példa következik.

Állományok vizsgálata

A `file_exists` függvény segítségével lekérdezhetjük, hogy egy állomány létezik-e:

```php
$filename = '/path/to/foo.txt';
if (file_exists($filename)) {
    echo "The file $filename exists";
} else {
    echo "The file $filename does not exist";
}
```

A következő függvények is hasonló módon információ lekérdezésére használhatóak:

- `is_dir`: az állomány könyvtár-e
- `is_file`: az állomány állomány-e (vagyis nem könyvtár)
- `is_readable`: az állomány olvasható-e
- `is_writable`: az állomány írható-e

Állományok törlése

Az `unlink` segítségével egy állományt „törölhetünk” a fájlrendszerből.

Itt érdemes kicsit elidőznünk azon, hogy a Unix filozófiája szerint egy állománynak több egyenértékű neve (és ezzel együtt elérési útvonala) lehet egy fájlrendszeren belül. Ha egy több névvel rendelkező állomány egyik nevét töröljük, akkor a többi nevén még elérhető marad. Emiatt nem biztos, hogy fizikai törlés lesz a függvény használatának következménye.

```php
$file = "test.txt";
if (!unlink($file))
  echo ("Error deleting $file");
else
  echo ("Deleted $file");
```

Könyvtárkezelés

Könyvtárat létrehozni az `mkdir`, törölni az `rmdir` függvénnyel tudunk:

```php
mkdir("testing");
```

A könyvtár nevének megadásán kívül egyéb paramétereket is használhatunk a jogosultságok megadására és rekurzív létrehozásra:

```php
mkdir("/path/to/my/dir", 0700);
```

Könyvtárak bejárása

Ha szükségünk van arra, hogy egy könyvtárban található összes állományról tudomást szerezzünk, az `opendir`, `readdir` és `closedir` függvényekre lesz szükségünk.

```php
$dir = opendir("images");
while (($file = readdir($dir)) !== false)
  echo "filename: " . $file . "<br>";
closedir($dir);
```

Sikeres megnyitás esetén a `$dir` erőforrás változót adja vissza az `opendir` függvény. Ezután a `readdir` függvényt addig hívogatjuk, amíg sztringet, és nem `false` értéket ad vissza. Végül a `closedir` szabadítja fel a foglalt erőforrásokat.

3.6.4. Fájlok tartalmának kezelése

PHP-ben a fájlkezelés logikája a hagyományos C (és nem a C++) nyelvre épül.

Az `fopen` függvényt fájlok megnyitására használjuk a PHP-ben. A függvény első paramétere tartalmazza a megnyitandó fájl nevét, a második azt határozza meg, hogy milyen módon nyissuk meg a fájlt.

```php
<html>
<body>
<?php
  $file=fopen("welcome.txt","r");
?>
</body>
</html>
```

A fontosabb megnyitási módok:

Mód	Leírás
r	Csak olvasás. A fájl elején kezdődik.
w	Csak írás. Megnyitja és törli a fájl tartalmát; vagy egy új fájlt készít ha a fájl nem létezik.
a	Hozzáfűzés. Megnyitja a fájlt és a végére kezd írni; vagy egy új fájlt készít ha a fájl nem létezik.

11. táblázat. Az fopen megnyitási módjai

Ha az `fopen` nem tudja megnyitni a fájlt, `false`-t ad vissza.

Az alábbi példa egy üzenetet ad vissza, ha az `fopen` nem tudja megnyitni a megadott fájlt:

```php
<html>
<body>
<?php
  $file=fopen("welcome.txt","r") or exit("Unable to open file!");
?>
</html>
</body>
```

A kód az `or` rövidzár kiértékelése (3.2.5. fejezet) miatt csak az `fopen` `false` visszaadott értéke esetén futtatja az `exit` függvényt.

A fájl bezárása

Az `fclose` függvényt fájlok bezárására használjuk. Illik – más erőforrásokhoz hasonlóan – a fájlokat a lehető legkorábban „elengedni”.

```php
<?php
  $file = fopen("test.txt","r");
  //...
  fclose($file);
?>
```

Fájlvég ellenőrzés

Az `feof` függvény ellenőrzi a „fájlvég” (`EOF`) elérését.

Az `feof` függvény hasznos ismeretlen hosszúságú állományon való munkánál.

```php
if (feof($file)) echo "End of file";
```

Fájl soronkénti beolvasása

Az `fgets` függvényt egy fájl egy sorának beolvasására használjuk. A függvény hívása után az ún. fájl mutató a következő sorra mutat.

Az alábbi példa sorról sorra beolvas egy fájlt, míg el nem éri annak végét:

```php
$file = fopen("welcome.txt", "r") or
    exit("Unable to open file!");
while(!feof($file)) {
  echo fgets($file). "<br>";
}
  fclose($file);
```

Fájl karakterenkénti beolvasása

Az `fgetc` függvény egy fájl egy karakterének beolvasására szolgál. A függvény hívása után a fájlmutató a következő karakterre mutat.

Az alábbi példa karakterenként olvas be egy fájlt, míg el nem éri annak végét.

```php
$file=fopen("welcome.txt","r") or exit("Unable to open file!");
while (!feof($file)) {
  echo fgetc($file);
}
fclose($file);
```

Bináris fájlkezelés

PHP-ben viszonylag ritka a közvetlen bináris fájlkezelés, legtöbbször szöveges állományokkal dolgozunk. Amikor mégis szükség van rá, akkor pl. a *GD Library*-t[77] használjuk képek létrehozására, manipulálására. (Pl. bélyegkép készítés, képek vízjelezése, stb.) Más feladatokra hasonló magasabb szintű megoldások léteznek.

Példaként nézzünk meg egy egyszerű képkonvertáló GD kódot:

```php
$png_image = imagecreatefrompng( "input.png" );
imagejpeg( $png_image, "output.jpg" );
imagedestroy( $png_image );
```

3.7. Felhasználókezelés

Amióta dinamikus weboldalakat fejlesztünk, nagy jelentősége lett annak, hogy a látogatók személyét be tudjuk azonosítani. Nem mindegy, hogy egy külső látogató, vagy a weboldal tulajdonosa, adminisztrátora, vagy szerkesztője látogatja az oldalt.

Ebben a fejezetben megnézünk néhány alapvető technikát a témával kapcsolatban.

3.7.1. Sütik kezelése

A süti (cookie) egy információcsomag, amelyet a szerver küld a böngészőnek, majd a böngésző visszaküld a szervernek minden, a szerver felé irányított kérés alkalmával.

A süti bármilyen, a kiszolgáló által meghatározott információtartalmat hordozhat. Az eljárás célja az állapot bevezetése az alapvetően állapotmentes HTTP tranzakcióba. Sütik hiányában minden egyes weboldal (vagy erőforrás) lekérése elszigetelt esemény, gyakorlatilag független a honlap többi oldalának lekérésétől.

Ha a böngésző visszaküld egy sütit, a szerver oldali kódnak lehetősége van összekapcsolni az aktuális kérést a korábbiakkal. Leggyakrabban egy adott weboldal regisztrált felhasználóinak azonosítására, „bevásárlókosár” nyilvántartására vagy látogatók nyomonkövetésére használják.

77 http://php.net/manual/en/book.image.php

A legtöbb böngésző egyszerű szöveges fájlban vagy fájlokban tárolja a sütik tartalmát, hogy azok a böngésző kikapcsolása és újraindítása után is elérhetőek maradjanak. Ezeket sütifájloknak nevezik.

Hogyan készítsünk sütit?

Süti készítésére a `setcookie` függvényt használhatjuk.

Fontos megértenünk, hogy a `setcookie` függvényt még a `doctype` és a `html` tag előtt kell meghívnunk.

A HTTP protokoll szerint a szerver először ún. fejlécet küld, ennek a fejlécnek lesz része a sütink is. A HTML oldal a fejléc után kerül küldése, egy elválasztó üres sor után. Ha tehát akár egyetlen betűt is elküldünk az oldalból, akkor már nem küldhetünk fejlécet. Ha mégis megtesszük, a következő figyelmeztetést kapjuk:

```
Warning: Cannot modify header information – headers already sent ...
```

Nézzünk egy kommunikációs példát. A böngésző HTTP kérést küld a szerver felé:

```
GET /index.html HTTP/1.1
Host: www.oldalam.com
```

A szerver válasza:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
Set-Cookie: name2=value2; Expires=Wed, 09 Jun 2021 10:18:14 GMT

<!DOCTYPE html ...
```

Ezután egy újabb HTTP kérés esete:

```
GET /spec.html HTTP/1.1
Host: www.oldalam.com
Cookie: name=value; name2=value2
```

Nézzük a PHP szintaxist:

```
setcookie(name, value, expire, path, domain);
```
PHP

Az alábbi példában létrehozunk egy user nevű sütit és az `Alex Porter` értéket rendeljük hozzá, továbbá az érvényességi időt egy órában határozzuk meg.

```
<?php
  setcookie("user", "Alex Porter", time()+3600);
?>
<html>
<body>
</body>
</html>
```
PHP

A süti értéke automatikusan kódolásra kerül küldéskor, és automatikusan dekódolódik kiértékeléskor. Kódolás nélküli küldéshez a `setrawcookie` függvény használható.

A süti kiolvasása

A süti értéke a `$_COOKIE` tömbben kerül tárolásra. Az alábbi példában a `user` nevű süti értékét, majd az egész tömb tartalmát írjuk ki a képernyőre.

PHP
```
<?php
  echo $_COOKIE["user"];
  print_r($_COOKIE);
?>
```

A következő kódrészletben az `isset` függvény segítségével ellenőrizzük, hogy érkezett e süti.

PHP
```
<html>
<body>
<?php
  if (isset($_COOKIE["user"]))
    echo "Welcome " . $_COOKIE["user"] . "!<br>";
  else
    echo "Welcome guest!<br>";
?>
</body>
</html>
```

Süti törlése

A süti törlése gyakorlatilag azt jelenti, hogy a sütit lejártnak állítjuk be.

Példa törlésre (az érvényesség lejártának beállítása egy órával ezelőttire):

PHP
```
<?php
  setcookie("user", "", time()-3600);
?>
```

3.7.2. Munkamenet-kezelés

A munkamenet-kezelést (session) arra használjuk, hogy információt tároljon a felhasználó beállításairól, vagy annak megváltozásáról.

Amikor egy asztali (desktop) alkalmazással dolgozunk, megnyitjuk, változtatunk valamit, majd bezárjuk. A számítógép tudja, hogy ki végzi a műveletet. Tudja, amikor elindítjuk az alkalmazást, és amikor bezárjuk. De az interneten ez nem így van. A webszerver nem tudja, hogy kik vagyunk és mit csinálunk, mert a HTTP protokoll nem tartja meg a kérések közt az állapotát. Úgy is mondjuk, hogy a HTTP protokoll *állapotmentes*.

A munkamenet-kezelés megoldja ezt a problémát. Engedélyezi a felhasználókra vonatkozó információk tárolását a szerveren (felhasználó név, vásárlási tételek, stb.). A session információk ideiglenesek, és törlődnek, miután a felhasználó elhagyta az oldalt. Ha állandó tárolásra van szükség, akkor az adatokat adatbázisba kell elmenteni.

Sütik és/vagy munkamenet?

A munkamenetek úgy működnek, hogy létrehozunk egy egyedi azonosítót (UID) minden látogatónak, és változókban tároljuk. Az UID többnyire sütiben, esetleg az URL-ben kerül továbbításra GET paraméterként.

Itt érdemes azt is kiemelni, hogy a sütik segítségével minden adatot oda-vissza kellene küldözgetni a kliens és a szerver között. Munkamenet-kezelés esetén elegendő egy munkamenet-azonosítót küldeni, a többi adat a szerveren kerül tárolásra.

Munkamenet indítása

Mielőtt információkat tárolnánk, el kell indítani a munkamenetet.

A `session_start` függvénynek a `doctype` és a `html` tag előtt kell szerepelnie:

```php
<?php session_start(); ?>
<html>
<body>

</body>
</html>
```

Az előbbi kód regisztrálja a session-t a szerveren, engedélyezi a felhasználói információk mentését, és az UID-t továbbítja a felhasználó számára.

Adatmentés munkamenetbe

A session változók tárolása és visszaolvasása a $_SESSION változóval lehetséges:

```php
<?php
  session_start();
  $_SESSION['views']=1;
?>
<html>
<body>
<?php
  echo "Pageviews=". $_SESSION['views'];
?>
</body>
</html>
```

Kimenet:

```
Pageviews=1
```

Az előző példában készítettünk egy egyszerű oldal látogatottság számlálót: megszámoljuk, hogy ugyanaz a felhasználó ugyanazon az oldalon hányszor járt. Az `isset` függvény ellenőrzi, hogy a `views` munkamenet-változó kapott-e már értéket. Ha igen, akkor növeljük eggyel. Ha a `views` nem létezik, akkor létrehozzuk, és beállítjuk `1`-re:

```php
session_start();
if(isset($_SESSION['views']))
  $_SESSION['views']=$_SESSION['views']+1;
else
  $_SESSION['views']=1;
echo "Views=". $_SESSION['views'];
```

A session törlése

A session adatok törlése az `unset` vagy a `session_destroy` függvényekkel történik.

Az `unset` függvényt egy session változó törlésére használjuk. Ekkor a munkamenet egyéb adatai megmaradnak.

```php
unset($_SESSION['views']);
```

A session teljes törlése a `session_destroy` függvénnyel lehetséges:

```php
session_destroy();
```

A `session_destroy` függvény újraindítja a session-t, és az összes tárolt session adatot elveszítjük.

Korábban a `session_register` és a `session_unregister` függvények voltak használatosak a munkamenet-változók kezelésére. Ez azonban elavultnak tekinthető, használjuk mindig közvetlenül a `$_SESSION` tömböt.

3.8. Objektumorientált PHP

Az elmúlt évek fejlődési iránya nem csak a hagyományos, hanem a webes fejlesztés területén is az *objektumorientált tervezés és programozás*. A PHP nyelv ugyan lemaradva, de ma már lehetővé teszi az OOP magas szintű használatát is.

Azon olvasók kedvéért, akik az OOP-vel még nem találkoztak, mindenképpen érdemes egy kis bevezetővel kezdenünk.

Mivel a PHP objektumorientált képességei erősen a Java 5-ös megoldásaira építenek, további bevezetésként érdemes elolvasni a szerző *Java programozás*[78] című könyvének 2. fejezetét is.

A fejezet *Tim Huegdon* nagyszerű cikksorozata[79] alapján készült.

3.8.1. Az OOP alapjai

Az OOP egy olyan paradigma, amely a hagyományosnak tekinthető procedurális megközelítés továbbfejlesztése. A korábbi programtervezési módszerekkel megterveztük az algoritmust, és az adatszerkezetet, de a kettő között gyakran elég laza kapcsolat volt. Az OOP

78 http://nagygusztav.hu/java-programozas

79 http://nefariousdesigns.co.uk/archive/2006/05/object-oriented-concepts/

legalapvetőbb jellemzője az *egységbezárás*, amely a kód és adat sokkal szorosabb egységét fejezi ki. Az egységbezárás az objektumok és osztályok szintjén is megvalósul.

Objektumok és osztályok

Egy objektumorientált program tulajdonképpen objektumokból áll össze. Az objektum állapotát *adattagokkal*, viselkedését pedig *metódusokkal* (*tagfüggvényekkel*) tudjuk leírni. Az objektumok egymással kommunikálni elsősorban a metódusaikon keresztül tudnak.

Az *osztály* az egyes objektumok elvi tervrajzát, működését definiálja. Az elnevezés onnan. származik, hogy az egyes objektumok hasonlóságait a tervezés során felismerhetjük. Ezt szoktuk osztályozásnak is hívni.

A programkódban tehát először létre hozzuk az osztályok forráskódját, majd az osztály példányai létrehozva, az objektumokat hálózatként használhatjuk a feladat megoldása érdekében.

Az objektumok létrehozását *példányosítás*nak hívjuk. A példány létrejöttekor automatikusan meghívódó függvény neve *konstruktor*. Az objektum élettartamának végén szükséges takarítást elvégző függvényt *destruktor*nak hívjuk.

Öröklődés

Az objektumok, és maguk az osztályok is kapcsolatban lehetnek egymással.

Az öröklődés azt fejezi ki, hogy a leszármazott megfelel az ősének, de tovább finomítja (specializálja) az őséhez képest a lehetőségeit.

Példaként nézzük egy pár gyümölcsöt:

```
az alma gyümölcs
a narancs gyümölcs
a banán gyümölcs
```

Ebben az esetben a 3 fajta gyümölcs rendelkezik minden tulajdonsággal és viselkedési lehetőséggel, ami minden más gyümölcsre jellemző, de vannak csak rá jellemző tulajdonságai is. Vagyis a gyümölcs osztály leszármazottai speciálisabbak, mint az őseik.

Nézzünk egy másik egyszerű példát. Egy űrlap lehetséges mezőit szeretnénk modellezni. Egy egyszerű (pszeudo) nyelven ezt így is leírhatnánk:

```
class formElement
{
  Attributes:
    id
    name
    class
}
class input extends formElement
{
  Attributes:
    value
    type
}
class textarea extends formElement
{
  Attributes:
    cols
    rows
}
```

Példánkban az `input` és `textarea` a `formElement` leszármazottai.

Objektumok közötti relációk

Az egyes objektumok közötti kapcsolatokra is vessünk néhány pillantást.

Az *asszociáció* azt jelenti, hogy az egyes osztályok között függőségek vannak. Irányítatlan asszociáció esetén mindkét fél tud a másikról. A gyakorlatban gyakrabban előforduló irányított asszociáció esetén csak az egyik fél tud a másikról.

A következő (pszeudo nyelvű) példa jól kifejezi, hogy a kapcsolat a tulajdonos irányában állandó, nem hiányozhat.

```
class person {
}
class car {
  Attributes:
    driver
  Methods:
    // Konstruktor:
    car(driver) {
      this.driver = driver;
    }
}
me = new person();
myMotor = new car(me);
```

Többalakúság

A többalakúság (vagy polimorfizmus) azt jelenti, hogy az egyes osztályok példányai másként viselkedhetnek egy adott helyzetben.

Példaként a következő kód más-más HTML kimenetet gyárt az egyes űrlap elemek esetén:

```
class formElement {
  Attributes:
    id
    name
    class
  Methods:
    getHtml() {
      // alap HTML kimenet
    }
}
class textarea extends formElement {
  Attributes:
    cols
    rows
  Methods:
    getHtml() {
      // speciális textarea kimenet
    }
}
```

Ebben a példában a leszármazott osztály felülírja az ősosztályában meglévő viselkedést.

3.8.2. Osztályok használat

Nézzük meg PHP szintjén is néhány példán keresztül az alapfogalmakat, alapvetőbb technikákat.

PHP-ben először létre kell hoznunk (definiálnunk kell) az *osztályt*, hogy utána abból példányosítva az *objektumok* is létrejöhessenek.

PHP

```
class myClass {
  var $attribute1;
  var $attribute2;
  function method1() {
    // törzs
    return $something;
  }
  function method2() {
    // törzs
  }
}
```

A fenti osztályból példányt létrehozni a `new` operátorral tudunk:

PHP

```
$myObject = new myClass();
$myObject2 = new myClass();
$myObject3 = new myClass();
$myObject4 = new myClass();
```

A fenti objektumok tagjaihoz (akár adattag, akár tagfüggvény) csak a nevük leírásával nem tudunk hozzáférni. Minden esetben minősített nevet kell alkalmaznunk:

```php
// Tulajdonságok
$myObject->attribute1 = 'Sonic';
$myObject->attribute2 = 'Knuckles';
// Metódusok:
$returned = $myObject->method1();
$myObject->method2();
```

A fenti, PHP 4-es verziójának megfelelő példa még nem tartalmaz láthatósági információkat, vagyis minden publikusan látható az osztályon kívülről is.

A $this változó

Ha nem kívül, hanem az osztály törzsén belül szeretnénk a tagokra utalni, akkor – más nyelvekhez képest meglepő módon – szintén kötelező a minősített hivatkozás alkalmazása. Ilyenkor a `$this` változó segítségével érhetjük el a tagokat.

```php
class myClass {
  var $attribute1;

  function method1() {
    // attribute1 értékét adja vissza
    return $this->attribute1;
  }
}
```

A konstruktor

A konstruktor egy olyan speciális metódus, amely automatikusan meghívódik az objektum példányosítása során, és beállíthatja az objektum induló állapotát.

A következő osztály az adatbázissal való kapcsolatot kezeli. A konstruktor megkapja a csatlakozáshoz szükséges információkat.

```php
class database {
  var $str_schema;
  var $str_host;
  var $str_user;
  var $str_password;
  // Konstruktor:
  function database($str_schema, $str_host, $str_user, $str_password) {
    // Adattagok beállítása a paraméterek alapján
    $this->str_schema   = (string) $str_schema;
    // Jó szokás a típuskényszerítés alkalmazása
    $this->str_host     = (string) $str_host;
    $this->str_user     = (string) $str_user;
    $this->str_password = (string) $str_password;
  }
}
```

A fenti példa szintén a PHP 4-es logikáját követi. Az 5-ös verzióban a konstruktor neve más lesz.

Az objektum példányosítása így történhet:

```php
$db = new database('myschema', 'localhost', 'username', 'password');
```

PHP 5-ben már inkább a következő szintaxist használjuk a konstruktor létrehozására:

```
function __construct($str_schema, $str_host, $str_user,
    $str_password) {
  // Adattagok beállítása a paraméterek alapján
  $this->str_schema   = (string) $str_schema;
  // Jó szokás a típuskényszerítés alkalmazása
  $this->str_host     = (string) $str_host;
  $this->str_user     = (string) $str_user;
  $this->str_password = (string) $str_password;
}
```

PHP 5

A destruktor

PHP-ben viszonylag ritkán használjuk a destruktort, amely az objektum felszabadítása előtt hívódik meg. PHP 5-ben így használhatjuk:

```
function __destruct() {
  // A mysql kapcsolat bezárása
  if ($this->res_connection) {
    mysql_close($this->res_connection);
  }
}
```

PHP 5

3.8.3. Öröklődés

Az öröklődés a kód újrahasznosításának egy gyakori módja. Az alábbi `formElement` osztály adattagjait és tagfüggvényét örökli a leszármazott `dateInput` osztály.

```
class formElement {
  var $str_id;
  var $str_name;
  var $str_class;
  function isRequired() {
    // Érvényesítő kód
  }
}
class dateInput extends formElement {
  var $str_value;
  // Speciálisabb érvényesítés
  function isValidDate() {
    // Érvényesítő kód
  }
}
```

PHP

Így a `myDate` objektum tagjai között elérhetők mind a saját, mind az ősétől örökölt tagok.

```
$myDate = new dateInput();
$myDate->str_id = 'mydate';
$myDate->str_name = 'mydate';
$myDate->str_value = '2006/06/09';
$myDate->isRequired();
$myDate->isValidDate();
```

PHP

A konstruktorok futása

Öröklődés esetén a leszármazott konstruktorból meg kell hívnunk a szülő valamelyik konstruktorát. PHP 4-es esetén:

PHP

```
class fruit {
  // Konstruktor:
  function fruit() {
  }
}
class apple extends fruit {
  function apple() {
    // A szülő osztály konstruktorának hívása
    $this->fruit();
  }
}
```

PHP 5-ös esetén az ősosztály konstruktorát a `parent` hatókörben tudjuk elérni:

PHP 5

```
class fruit {
  // Konstruktor:
  function __construct() {
  }
}
class apple extends fruit {
  function __construct() {
    // Szülő osztály konstruktorának hívása
    parent::__construct();
  }
}
```

3.8.4. Asszociáció

Az egyes objektumok között gyakori a tartalmazási vagy hasonló kapcsolat. Pl. a fal téglákból áll:

PHP

```
class brick {
  var $sample_attribute;
}
class wall {
  var $brick1;
  var $brick2;
  var $brick3;
  var $brick4;
  // Konstruktor:
  function wall() {
    $this->brick1 = new brick();
    $this->brick2 = new brick();
    $this->brick3 = new brick();
    $this->brick4 = new brick();
  }
}
```

A tartalmazási kapcsolat onnan is látszik, hogy a fal objektum létrejöttekor automatikusan létrejönnek a tartalmazott téglák is. Akár egyből használhatjuk is őket:

```php
$myWall = new wall();
echo $myWall->brick1->sample_attribute;
echo $myWall->brick2->sample_attribute;
echo $myWall->brick3->sample_attribute;
echo $myWall->brick4->sample_attribute;
```

Nézzünk egy másik példát. Az autó objektumhoz szükséges a vezető, nélküle nem jöhet létre:

```php
class person {
  // ...
}
class car {
  var $driver;
  // Konstruktor:
  function car(&$driver) {
    $this->driver = (object) $driver;
  }
}
// A person példányostása
$me = new person();
// car létrehozása a vezető megadásával
$myCar = new car($me);
```

Érdemes megfigyelni a konstruktor referencia szerinti paraméterátadását. Így a `car` konstruktor nem egy másolatot kap a vezetőből, hanem a referenciáját (álnevét).

3.8.5. Láthatóság

A PHP 5-ös objektummodellje abban is újdonságot hozott, hogy háromféle láthatóságot vezetett be a tagokra nézve:

- `public`: kívülről korlátlanul elérhető (ez a PHP 4-es hozzáállása is, a `var` szó is ezt jelenti)
- `private`: csak a saját osztályon belül látható, még a leszármazottakban sem
- `protected`: a saját és a leszármazott osztályból látható

Adattagokat alapvetően privát, esetleg végett, míg konstruktorokat, metódusokat általában publikus láthatósággal hozunk létre.

Nézzünk egy egyszerű példát:

PHP 5

```
class myClass {
  public $public = 'Public';
  protected $protected = 'Protected';
  private $private = 'Private';
  function printHello() {
    echo $this->public;
    echo $this->protected;
    echo $this->private;
  }
}
$obj = new myClass();
echo $obj->public; // Működik
echo $obj->protected; // Hiba!
echo $obj->private; // Hiba!
// Public, Protected és Private kiírása:
$obj->printHello();
class myClass2 extends myClass {
  protected $protected = 'Protected2';
  function printHello() {
    echo $this->public;
    echo $this->protected;
    echo $this->private;
  }
}
$obj2 = new myClass2();
echo $obj->public; // Működik
echo $obj2->private; // Definiálatlan
echo $obj2->protected; // Hiba!
// Public, Protected2, és nem Private
$obj2->printHello();
```

A továbbiakban már lesz lehetőségünk arra is, hogy a ma egyre elterjedtebb objektumorientált megközelítést alkalmazzuk a gyakorlatban is.

3.9. Hibakezelés

Amikor webalkalmazásokat készítünk, a hibakezeléssel nagyon fontos törődni. Ha az alkalmazásunk nem végez korrekt ellenőrzéseket, az oldal hibásan fog működni, és biztonsági réseket tartalmazhat.

Ez a fejezet a PHP-ben leggyakrabban alkalmazott hibakezelő módszereket mutatja be:

- egyszerű `die` kifejezés
- hibakezelő függvények
- kivételkezelés

3.9.1. Alapvető hibakezelés: a `die` függvény használata

Az első példa egy egyszerű szkriptet mutat be, amely egy szövegfájlt nyit meg:

```
$file=fopen("welcome.txt","r");
```
PHP

Ha a fájl nem létezik, akkor egy ehhez hasonló hibaüzenetnek kell megjelennie:

```
Warning: fopen(welcome.txt) [function.fopen]: failed to open stream:
No such file or directory in C:\webfolder\test.php on line 2
```

Ennek elkerülésére tesztelnünk kell, hogy a fájl valóban létezik-e, mielőtt megpróbálnánk hozzáférni:

```
if (!file_exists("welcome.txt")) {
  die("File not found");
} else {
  $file=fopen("welcome.txt","r");
}
```
PHP

Ha a fájl nem létezik, akkor egy hibaüzenetet fogunk kapni:

```
File not found
```

A fenti módon megakadályozható vele, hogy olyan információkat szivárogtassunk ki, amelyek rosszindulatú támadásokhoz adhatnak támpontot.

A szkript leállítása nem mindig a megfelelő út. Vessünk egy pillantást két alternatív hibakezelő megoldása. Az első függvény alapú, a második objektumorientált megközelítést alkalmaz.

3.9.2. Alapértelmezett hibakezelő függvény készítése

A hibakezeléssel kapcsolatban az egyik legnagyobb veszély, hogy elfeledkezünk egyes hibalehetőségek lekezeléséről. Ezért praktikus, ha minden lehetséges hibát egységesen kezelünk le. Az alapötlet az, hogy megkérjük a futtatókörnyezetet: minden hiba esetén hívja meg ugyanazt a függvényt. Ezt a hibakezelő függvényt tehát nem a mi kódunk fogja közvetlenül meghívni. Az ilyen függvényt *callback* függvénynek szokás hívni.

Egy alapértelmezett hibakezelő függvény készítése meglehetősen egyszerű. Egyszerűen egy speciális függvényt kell létrehozni, ami meghívható, ha hiba merül fel futás közben.

Ennek a függvénynek tudnia kell kezelni legalább két paramétert (hiba szintje és hibaüzenet), de elfogadhat maximum öt paramétert. (Az opcionálisak: fájl, a sor száma és a hibakörnyezete.)

Szintaxis:

```
error_function (error_level,error_message, error_file, error_line,
        error_context)
```
PHP

A paraméterek jelentése:

- `error_level`: Kötelező. Megadja, hogy milyen hibákat akarunk lekezelni. Egy számértéknek kell lennie a 12. táblázatnak megfelelően.

- `error_message`: Kötelező. A hibaüzenetet tartalmazza.
- `error_file`: Opcionális. A fájlnevet tartalmazza, ahol a hiba felmerült.
- `error_line`: Opcionális. A sor számát tartalmazza, ahol a hiba felmerült.
- `error_context`: Opcionális. Egy tömböt tartalmaz, amiben benne van minden változó és annak értéke, melyek használatban voltak, amikor a hiba felmerült.

Hiba értesítési szintek

Ezek a hiba értesítési szintek azok a hibafajták, melyek orvoslására a felhasználó által létrehozott hibakezelőket használni lehet:

Érték	Konstans	Leírás
2	`E_WARNING`	Nem végzetes futásidejű hibák. A szkript végrehajtása nem lesz megszakítva.
8	`E_NOTICE`	Futásidejű figyelmeztetés. A szkript talált valamit, ami lehetséges hibaforrás, de az értesítés akkor is előfordulhat, ha a szkript hibamentesen fut.
256	`E_USER_ERROR`	Végzetes felhasználó által generált hiba. Ez olyan, mint egy `E_ERROR`, melyet a programozó állít be a `trigger_error` függvényt használva.
512	`E_USER_WARNING`	Nem végzetes felhasználó által generált figyelmeztetés. Ez olyan, mint egy `E_WARNING`, melyet a programozó állít be a `trigger_error` függvényt használva.
1024	`E_USER_NOTICE`	Felhasználó által generált értesítés. Ez olyan, mint egy `E_NOTICE`, melyet a programozó állít be a `trigger_error` függvényt használva.
4096	`E_RECOVERABLE_ERROR`	Elkapható végzetes hiba. Ez olyan, mint egy `E_ERROR`, de ez elkapható egy felhasználó által definiált hibakezelővel.
8191	`E_ALL`	Minden hiba és figyelmeztetés, kivéve a `E_STRICT` szintet.

12. táblázat. Hiba értesítési szintek

Hozzunk létre egy függvényt a hibák kezelésére:

```php
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}
```

A fenti kódrészlet egy egyszerű hibakezelő függvény. Amikor megkapja a vezérlést, megkapja a hiba szintjét és egy hiba üzenetet. Ezután a kimenetre küldi a hibaszintet és üzenetet, és a szkript futása befejeződik. Természetesen ez csak egy vázlatos megoldás. Ilyen esetben célszerű lehet egy log fájlba írni a hibakezeléshez szükséges információkat.

Most tehát, miután írtunk egy hibakezelő függvényt, el kell döntenünk, milyen esetekben legyen meghívva.

A hibakezelő beállítása

Az alapértelmezett hibakezelő a PHP-ban egy beépített függvény. Most azonban a beépített megoldás helyett a fenti függvényt fogjuk használni.

Lehetséges úgy megváltoztatni a hibakezelőt, hogy csak bizonyos hibákra vonatkozzon, és a szkript különböző hibákat különbözőképpen tudjon kezelni. Mi most ebben a példában a saját hibakezelőt fogjuk használni az összes hiba esetén:

```php
set_error_handler("customError");
```

Mivel azt akarjuk, hogy a hibakezelő függvényünk minden hibát kezeljen, a `set_error_handler` csak egy paramétert kívánt, a második csak opcionális, és a hiba szintjét határozza meg.

Teszteljük a hibakezelőt, olyan változót megpróbálva kiíratni, ami nem létezik:

```php
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr";
}
set_error_handler("customError");
echo($test);
```

A kód kimenete ehhez hasonló lesz:

```
Error: [8] Undefined variable: test
```

Hiba előidézése

Egy szkriptben, melyben a felhasználók adatokat vihetnek be, hasznos lehet a hibák előidézése, amikor egy nem megfelelő bevitel adódik. A PHP-ban ezt a `trigger_error`-ral lehetséges.

Ebben a példában hiba történik, ha a test változó nagyobb, mint `1`:

```php
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below");
}
```

A kód kimenete ehhez hasonlólesz:

```
Notice: Value must be 1 or below
in C:\webfolder\test.php on line 6
```

Egy hibát meg lehet hívni bárhol a szkripten belül, és egy második paraméter hozzáadásával meg lehet határozni, hogy milyen hiba szintet akarunk előidézni.

Ebben a példában egy `E_USER_WARNING` üzenet jelenik meg, ha a test változó nagyobb, mint 1. Egy ilyen esetben a szokásos hibakezelőnket használjuk, és befejezzük a szkript futtatását:

```php
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}
set_error_handler("customError",E_USER_WARNING);
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
```

A kód kimenete ehhez hasonló lesz:

```
Error: [512] Value must be 1 or below
Ending Script
```

Most, hogy megtanultunk saját hibakezelőt készíteni, és hogy hogyan lehet meghívni őket, vessünk egy pillantást a hiba naplózásra.

Hiba naplózása

A `php.ini` `error_log` konfigurációjában beállított módon lehetőség van a hiba naplózásának finomhangolására.

Hibaüzenetek a saját email címre küldése egy jó módszer arra, hogy értesüljünk a különböző hibákról.

Egy hiba üzenet elküldése e-mailben

Az alábbi példában egy e-mailt fogunk elküldeni egy hibaüzenettel, és leállítjuk a szkriptet, ha egy bizonyos hiba megjelenik:

```php
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Webmaster has been notified";
  error_log("Error: [$errno] $errstr",1,
  "someone@example.com","From: webmaster@example.com");
}
set_error_handler("customError",E_USER_WARNING);
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
```

A kód kimenete ehhez hasonló lesz:

```
Error: [512] Value must be 1 or below
Webmaster has been notified
```

A levél, amit kapunk, ehhez hasonló lesz:

```
Error: [512] Value must be 1 or below
```

Ezt nem tanácsos használni minden hiba esetén. A rendszeres hibákat inkább a szerveren naplózzuk, a PHP alapértelmezett naplózási rendszerét használva.

3.9.3. Kivételkezelés

A kivételkezelés objektumorientált nyelvek esetén gyakran használt hibakezelési megoldás.

A fejezet megértéséhez szükséges a 3.8. fejezet ismerete.

Mi is a kivétel?

A PHP 5-tel egy új objektumorientált lehetőség is adott a hibák kezelésére.

A kivételkezelés használatakor az megváltoztatja az utasítások végrehajtásának normál menetét, ha egy meghatározott hiba bekövetkezik. Ezt a feltételt kivételnek nevezzük.

Általában a következő történik kivétel kiváltódásánál:

- A jelenlegi állapot mentésre kerül.
- A kód végrehajtása átadódik egy előre meghatározott kivételkezelő kódra.

A kivételeket csak a hibák felmerülésekor kell lekezelni, és nem lehet arra használni, hogy a kód egy másik helyére ugorjunk velük.

A kivételek használata

Amikor kivétel keletkezik, az azt követő kód nem hajtódik végre, és a PHP megpróbálja megtalálni a hozzá tartozó kivételkezelő `catch` blokkot.

Ha egy kivételt egyetlen kezelő sem kap el, egy hiba fog keletkezni, egy "Uncaught Exception" (Lekezeletlen kivétel) üzenettel.

Dobjunk (throw) egy kivételt anélkül, hogy elkapnánk:

```php
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}
checkNum(2);
```

A fenti kód a következő hibát fogja generálni:

```
Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in C:\webfolder\test.php on line 6
```

try, throw és catch

Ha el szeretnénk kerülni a fenti hibát, létre kell hoznunk a megfelelő kódot, hogy kezelni tudja a kivételt. Nem csak a saját, hanem a futtatókörnyezetet által eldobott kivételek esetén is.

A kivételkezelő kódnak tartalmaznia kell a következőket:

- A try blokkba kell helyezni azt a kódot, amelyikben a kivétel létrejöhet. Pl. egy függvényhívásnak, ami kivételt dobhat, benne kell lennie egy try blokkban. Ha nincs kivétel dobva, a kód normális módon fog folytatódni. Azonban kivétel keletkezése esetén a kivételkezelő catch blokk fog lefutni.
- A throw utasítással tudunk egy kivétel objektumot eldobni, jelezve ezzel a hiba bekövetkezését.
- A catch blokk elkapja a kivétel objektumot, amivel elérhetővé válik a hiba helye és jellege.

A következő példában a checkNum függvényben kivételt dobunk:

```php
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}
try {
  checkNum(2);
  echo 'If you see this, the number is 1 or below';
}
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}
```

A fenti kód a következő hibát fogja okozni:

```
Message: Value must be 1 or below
```

A fenti kód dob egy kivételt, és el is kapja azt:

1. A `checkNum` függvény azt nézi, hogy a szám nagyobb-e 1-nél. Ha nagyobb, akkor kivételt dob.
2. A `checkNum` függvényt meghívjuk a `try` blokkban.
3. A `checkNum` függvény kivételt dob.
4. A `catch` blokk lekezeli a kivételt, és elérhetővé teszi a kivétel információit.
5. A hibaüzenet megjelenik az `$e->getMessage()` meghívása alapján.

A legalapvetőbb szabály: *Ha kivételt dobunk, azt el is kell kapnunk.*

További bevezetésként érdemes elolvasni a szerző *Java programozás*[80] című könyvének 18. fejezetét is.

3.10. Tervezési minták

A webfejlesztés során használt tervezési minták (*design patterns*) nagyban segíthetik a munkánkat, hiszen mások bevált ötletei alapján valószínűleg gyorsabb és hatékonyabb alkalmazást készíthetünk, mint csupán a saját ötleteinkre és tapasztalatainkra támaszkodva.

A weben igen sok (angol nyelvű) forrás található a témakörben. A szerző véleménye szerint az egyik legjobb a *phpPatterns*[81] oldala. Ezen kívül a Zend *PHPPatterns: Instructions*[82] cikke is nagyon hasznos.

Mi a tervezési minta?

Ha egy feladat újra előkerül a fejlesztés folyamán, akkor valószínűleg a megoldás hasonló lesz a korábbihoz. A tervezési minták olyan objektumközpontú megoldásokat jelentenek, amelyek már bizonyítottak a gyakorlatban. Ezek felhasználása rugalmasabban módosítható és könnyebben, jobban újrahasznosítható alkalmazásokat készíthetünk.

Mivel jelen könyv nem vállalkozhat az objektum-orientált programozás és tervezés komolyabb lehetőségeit bemutatni, elsősorban az alapelvek és néhány egyszerűbb megoldás következik. A fejezet megértéséhez szükséges a 3.8. fejezet ismeretére.

A 3.6.2. fejezetben már megismerkedtünk a Front Controller minta alapjaival. E fejezetben néhány további tervezési mintát fogunk megvizsgálni.

80 http://nagygusztav.hu/java-programozas
81 http://www.phppatterns.com
82 http://devzone.zend.com/node/view/id/3

3.10.1. Stratégia

A *Strategy* egy objektumba zárt algoritmus. Az egységbezárás miatt az algoritmus jól elkülöníthető a kód más részeitől, az öröklődés miatt pedig a stratégia könnyen lecserélhető egy másikra.

Validátor példa

A fejezet példája[83] segítségével az űrlap által küldött adatok érvényességének ellenőrzését (validálását) fogjuk elvégezni.

Nézzünk egy hagyományos, nem túl praktikus megközelítést:

PHP
```
if ( isset ($_POST['submit']) ) {
    if ( $_POST['user'] < '6' ) {
        echo ('Username is too short');
    } else if ( $_POST['pass'] != $conf ) {
        echo ('Passwords do not match');
    } else if ( $_POST['fullmoon'] == true ) {
        // ...
    }
}
```

Látszik, hogy a kód kissé zavaros, nehezen áttekinthető és módosítható. Ezen kívül a hibaüzeneteket is azonnal megjeleníti. (Ezt célszerűbb lenne az űrlap elem közelében megtenni.)

Nézzük meg egy jobb megoldást. A `Validator` (logikailag[84]) egy absztrakt osztály, a lényegi munka a leszármazottakban fog történni.

PHP
```
class Validator {
    var $errorMsg;
    function Validator () {
        $this->errorMsg=array();
        $this->validate();
    }
    function validate() { }
    function setError ($msg) {
        $this->errorMsg[]=$msg;
    }
    function isValid () {
        if ( isset ($this->errorMsg) ) {
            return false;
        } else {
            return true;
        }
    }
    function getError () {
        return array_pop($this->errorMsg);
    }
}
```

83 A példa a http://www.phppatterns.com/docs/design/strategy_pattern oldalról származik.

84 A példa írásakor elterjedt 4-es PHP verzió még nem ismeri az absztrakt osztály fogalmát.

Jól látszik, hogy az ellenőrzés során a hibák felismerése a hibaüzenetek megfogalmazását is jelenti. A hibák az `$errorMsg` tömbben gyűlnek.

A következő osztály a felhasználónév érvényességét ellenőrzi:

PHP
```
class ValidateUser extends Validator {
    var $user;
    function ValidateUser ($user) {
        $this->user=$user;
        Validator::Validator();
    }
    function validate() {
        if (!preg_match('/^[a-zA-Z0-9_]+$/',$this->user )) {
            $this->setError(
                'Username contains invalid characters');
        }
        if (strlen($this->user) < 6 ) {
            $this->setError('Username is too short');
        }
        if (strlen($this->user) > 20 ) {
            $this->setError('Username is too long');
        }
    }
}
```

Érdemes megfigyelni, hogy ez a kód épít a PHP azon specialitására, ami szerint az ősosztály konstruktorát nem hívja meg automatikusan a leszármazott konstruktora. (Ebben az esetben az osztály logikája nem is működne.)

A jelszó és az e-mail cím ellenőrzése hasonló logikára épül:

PHP
```
class ValidatePassword extends Validator {
    var $pass;
    var $conf;
    function ValidatePassword ($pass,$conf) {
        $this->pass=$pass;
        $this->conf=$conf;
        Validator::Validator();
    }
    function validate() {
        if ($this->pass!=$this->conf) {
            $this->setError('Passwords do not match');
        }
        if (!preg_match('/^[a-zA-Z0-9_]+$/',$this->pass )) {
            $this->setError(
                'Password contains invalid characters');
        }
        if (strlen($this->pass) < 6 ) {
            $this->setError('Password is too short');
        }
        if (strlen($this->pass) > 20 ) {
            $this->setError('Password is too long');
        }
    }
}
```

```
class ValidateEmail extends Validator {
    var $email;
    function ValidateEmail ($email){
        $this->email=$email;
        Validator::Validator();
    }
    function validate() {
        $pattern=    "/^([a-zA-Z0-9])+([\.a-zA-Z0-9_-])*@([a-zA-Z0-9_-])+
(\.[a-zA-Z0-9_-]+)+/";
        if(!preg_match($pattern,$this->email)){
            $this->setError('Invalid email address');
        }
        if (strlen($this->email)>100){
            $this->setError('Address is too long');
        }
    }
}
```

Az ellenőrzés megvalósítása az osztályaink felhasználásával már egyszerű:

PHP

```
<?php
if ( $_POST['register'] ) {
 require_once('lib/Validator.php');
 $v['u']=new ValidateUser($_POST['user']);
 $v['p']=new ValidatePassword($_POST['pass'],$_POST['conf']);
 $v['e']=new ValidateEmail($_POST['email']);
    foreach($v as $validator) {
        if (!$validator->isValid()) {
            while ($error=$validator->getError()) {
                $errorMsg.="<li>".$error."</li>n";
            }
        }
    }
    if (isset($errorMsg)) {
        print (
            "<p>There were errors:<ul>n".$errorMsg."</ul>");
    } else {
        print ('<h2>Form Valid!</h2>');
    }
} else { ?>
<h2>Create New Account</h2>
<form action="<?php echo ($_SERVER['PHP_SELF']); ?>" method="post">
<p>Username: <input type="text" name="user"></p>
<p>Password: <input type="password" name="pass"></p>
<p>Confirm: <input type="password" name="conf"></p>
<p>Email: <input type="text" name="email"></p>
<p><input type="submit" name="register" value="Register"></p>
</form>
<?php } ?>
```

3.10.2. Front controller

A 3.6.2. fejezet egyszerűbb megoldása után nézzünk a Front controller tervezési mintáról is egy korszerűbb, objektumorientált megoldást. Jelen fejezet példája Chris Corbyn *A lightweight and flexible front controller for PHP 5* cikkéből[85] származik.

85 http://www.w3style.co.uk/a-lightweight-and-flexible-front-controller-for-php-5

Példaként egy egyszerű vendégkönyv jellegű alkalmazást fogunk megismerni.

index.php

Objektumorientált megközelítésben a Front controller általában egy osztály szokott lenni. Ez az osztály végzi a teljes vezérlést. Az `index.php` tulajdonképpen csak ennek előkészítésére szolgál, hosszabb kódot nem szokott tartalmazni:

```php
<?php
  define("PAGE_DIR", dirname(__FILE__) . "/pages");
  require_once "FrontController.php";
  FrontController::createInstance()->dispatch();
```

FrontController.php

A `FrontController` osztály példányosítása nem a szokásos módon, az osztályon kívül, hanem egy gyártó metódussal (`createInstance`) valósul meg.

```php
<?php
class FrontController {
  public static function createInstance() {
    if (!defined("PAGE_DIR")) {
      exit("Critical error: Cannot proceed without PAGE_DIR.");
    }
    $instance = new self();
    return $instance;
  }
```

A `dispatch` függvény adja az osztály tényleges logikáját. Először is jól látszik, hogy a get paraméterek között page és action értékekre számít. Az alapértelmezett `page` érték `home` lesz, az alapértelmezett `action` pedig `index`.

```php
  public function dispatch() {
    $page = !empty($_GET["page"]) ? $_GET["page"] : "home";
    $action = !empty($_GET["action"]) ? $_GET["action"] : "index";
```

Összeállítja a `$page` paraméter alapján a betöltendő osztály nevét. A kezdőbetűt nagybetűsíti az objektumorientált konvencióknak megfelelően.

```php
    $class = ucfirst($page) . "Actions";
```

Összeállítja a `$file` változóba a betöltendő állománynevet (pl. `pages/home/HomeActions.php`), majd megnyitja.

```php
    $file = PAGE_DIR . "/" . $page . "/" . $class . ".php";
    if (!is_file($file)) {
      exit("Page not found");
    }
    require_once $file;
```

Szintén a paraméterek alapján áll össze a meghívandó metódus neve. Ha létezik, meghívjuk (pl. `$controller->doIndex()`), és ezzel az osztály befejezte működését.

```php
    $actionMethod = "do" . ucfirst($action);
    $controller = new $class();
    if (!method_exists($controller, $actionMethod)) {
      exit("Page not found");
    }
    $controller->$actionMethod();
    exit(0);
  }
}
```

pages/guestbook/GuestbookActions.php

Példaként nézzük a vendégkönyv funkcionalitást megvalósító kód vázlatát.

```php
<?php
class GuestbookActions {
  public function doIndex() {
    // index (listázás) megvalósítás
  }
  public function doCreatePost() {
    // bejegyzés létrehozása megvalósítás
  }
}
```

Nézzünk 1-1 példát a két akció meghívására:

- http://localhost/index.php?page=guestbook&action=index
- http://localhost/index.php?page=guestboook&action=createPost

3.10.3. MVC

A fejezet elején meg kell jegyeznünk, hogy a 3. fejezet hátralevő része jelentősen komplexebb a könyv egyéb részeinél. Ha az Olvasó számára nehezen vagy nem érthető, akkor érdemes egy átolvasás után az alaposabb megismerését elhalasztani, és később visszatérni rá.

Az MVC[86] (*model – view – controller*, *modell – nézet - vezérlő*) egy jól használható, kipróbált módszer arra, hogy hogyan válasszuk szét a felhasználói felületet és az alkalmazás logikát. Az elsődleges cél az, hogy a felhasználói felület megjelenítéséért felelős kódot teljesen elkülönítsük. Ezáltal annak módosítása, kiegészítése nem vonja maga után az alkalmazás logikát megtestesítő kód módosítását, vagy megismétlését.

A módszer lényege az, hogy a hagyományos eljárás alapú programok adatbevitel-adatfeldolgozás-eredmény megjelenítése feladatokat leképezzék a grafikus felhasználói felülettel rendelkező programokban:

adatbevitel⟶	adatfeldologzás⟶	eredmény megjelenítése
controller ⟶	model⟶	view

86 Barkóczi Roland szakdolgozata (http://blog.aer.hu/) alapján

A *vezérlő* dolgozza fel a felhasználói adatbevitelt. Függvényhívásokká képezi le azokat. Ezek fogják előidézni az adatok módosítását, törlését, vagy a nézetek megváltozását. Például ha a felhasználó kiválasztja a menü egyik elemét, akkor egy vezérlő fogja meghatározni, hogy ennek hatására mi is történjen.

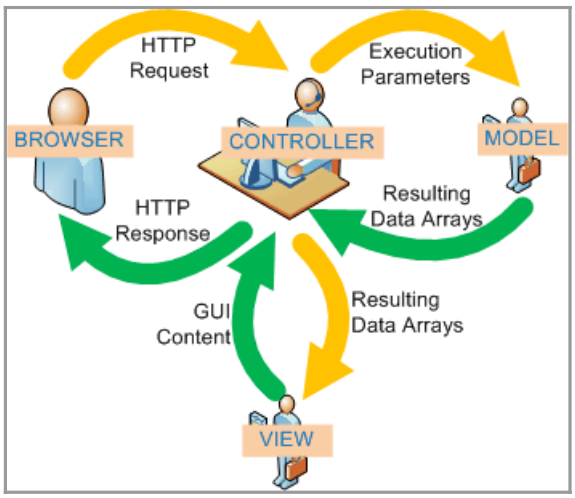
A *modell* reprezentálja az alkalmazás logikát, feladata az adatok kezelésével kapcsolatos feladatok elvégzése. Ez az egység felelős pl. egy számla áfa tartalmának és végösszegének kiszámolásáért. A modell tudja, hogy melyik vevő fogja kifizetni a számlát, és ha szükséges, azt is, hogy éppen születésnapja van-e ennek a vevőnek.

A *nézet* feladata a felhasználói felület megjelenítése. Ez az űrlapokat, táblázatokat, linkeket, gombokat, szövegeket jelent. Ezek az elemek megjelenhetnek egy szabványos HTML oldalon, de akár egy RSS csatornán is.

Vegyük egy konkrét gyakorlati PHP fejlesztői példát. Feladatunk egy felmérés eredményeinek megjelenítése táblázatban.

Hagyományos módszerrel a PHP forrásunk tartalmaz egy SQL lekérdezést, majd az adatokat egy ciklussal kiírja egy táblázatba. Ha az ügyfelünk ugyanezen adatokat egy grafikonon is látni szeretné, akkor létrehozunk egy másik fájlt, ami szintén tartalmazni fogja az SQL kérést, és az adatok megjelenítését.

Az MVC tervezési minta ezzel szemben tartalmazni fog egy modellt, ami felelős az adatbázis kezeléséért, és két nézetet, ami felelős az adatok megjelenítéséért. Azt, hogy melyik nézetet kell megjeleníteni, a vezérlő dönti el a felhasználó kérése alapján. Ha további nézetekre van szükségünk (pl. kördiagramm vagy esetleg egy szöveges magyarázatokkal ellátott kimutatásra), akkor csak a megfelelő nézetfájlokat kell elkészíteni, amelyek a modellből származó adatokat használják fel.



57. ábra. Az MVC felépítése

Egy vázlatos megoldás

Sajnos e könyv terjedelme és a téma komplexitása nem teszi lehetővé, hogy egy komolyabb megoldást alaposan bemutassunk. Ezért egy vázlatos megoldást fogunk megismerni Henrique Barroso *How to create a simple MVC framework in PHP*[87] cikke alapján. A következőkben egy *egyszerű blog oldal* vázlatát fogjuk látni.

Fájlstruktúra

Az MVC megoldásoknál nagy jelentősége van az alkalmazás könyvtár és állománystruktúrájának. Nézzük először a megoldásunk könyvtárszerkezetét. Gyökérkönyvtárként a `my_mvc` könyvtárat fogjuk használni. Így a böngészőnkben a http://localhost/my_mvc/ útvonalon tudjuk elérni.

- ./my_mvc
- ./my_mvc/application
- ./my_mvc/application/controller
- ./my_mvc/application/model
- ./my_mvc/application/view

.htaccess

Átirányítjuk az összes kérést az `index.php`-nek:

87 http://www.henriquebarroso.com/how-to-create-a-simple-mvc-framework-in-php/

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /my_mvc/
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule ^(.*)$ index.php/$1 [L]
</IfModule>
<IfModule !mod_rewrite.c>
    ErrorDocument 404 /index.php
</IfModule>
```

Természetesen a `mod_rewrite` modult engedélyezni kell az Apache beállításainál (3.1.1. fejezet).

index.php

Az `index.php` látja el az alapvető vezérlési feladatokat. A `.htaccess` miatt minden kérés ide fog érkezni, és ez alapján tölti be a további állományokat, és hozza létre az MVC megfelelő objektumait.

Definiáljuk az oldal url-jét és a szerveren belüli alap útvonalát:

```php
define("BASE_PATH", "http://localhost");
$path = "/my_mvc";
```

Az `$url` változóba kinyerjük a `$path` nélküli URL-t.

```php
$url = $_SERVER['REQUEST_URI'];
$url = str_replace($path,"",$url);
```

Az `$url`-ből tömbbé szeleteljük a `/` és `\` jelek mentén.

```php
$array_tmp_uri = preg_split('[\\/]', $url, -1, PREG_SPLIT_NO_EMPTY);
```

Az `$url` három szelete fontos szerepet tölt be az URL MVC szerinti kezeléséhez. Az első a példányosítandó osztály neve, a második az objektum függvénye, a harmadik pedig az átadott további paraméter.

```php
$array_uri['controller'] = $array_tmp_uri[0];
$array_uri['method'] = $array_tmp_uri[1];
$array_uri['var'] = $array_tmp_uri[2];
```

A következő sorok a vezérlő felépítését és beüzemelését valósítják meg:

```php
require_once("application/base.php");
$application = new Application($array_uri);
$application->loadController($array_uri['controller']);
```

application/base.php

Az `Application` osztály a vezérlők ősosztálya. Érdekes módon nem csak a leszármazottaiból fogunk példányt létrehozni, hanem belőle is. A megfelelő leszármazott példányosítását is ő valósítja meg. Ezt a megoldást a *Builder* (*Építő*) tervezési mintára hasonlít.

Az `Application` példánya a paraméterek tömbjét és a modell objektumot tartalmazza majd.

```
class Application                                                    PHP
  var $uri;
  var $model;
```

A konstruktor csupán a paramétereket menti el:

```
  function __construct($uri) {                                       PHP
    $this->uri = $uri;
  }
```

A `$class` változó szintén a paraméterből származik. A példa kód `blog` lesz, de továbbfejleszthető további vezérlőkel. Először is betölti a vezérlő osztályt.

```
  function loadController($class) {                                  PHP
    $file = "application/controller/".$this->uri['controller']."php";
    if(!file_exists($file)) die();
      require_once($file);
      $controller = new $class();
```

Ekkorra már van vezérlő példányunk is. Ha a 2. paraméter is korrekt, meghívjuk a neki megfelelő metódust. Különben az alapértelmezetten `index` működés lesz.

Itt az `add` és az `index` működések lesznek elérhetőek, ahogy a `Blog` osztály kódjában hamarosan látni fogjuk.

```
      if(method_exists($controller, $this->uri['method'])) {         PHP
        $controller->{$this->uri['method']}($this->uri['var']);
      } else {
        $controller->index();
      }
  }
```

A nézet a paraméter alapján töltődik be. De előtte a 2. paraméter `wddx` előtaggal önálló változókká lesz bontva, hogy a betöltött sablon állomány ez alapján végezhesse a behelyettesítést.

```
  function loadView($view,$vars="") {                                PHP
    if(is_array($vars) && count($vars) > 0)
      extract($vars, EXTR_PREFIX_SAME, "wddx");
    require_once('view/'.$view.'.php');
  }
```

A modell betöltése és példányosítása is a paramétertől függően történik. Példánkban `model_blog` paraméterrel fogjuk meghívni.

```
  function loadModel($model) {                                       PHP
    require_once('model/'.$model.'.php');
    $this->$model = new $model;
  }
}
```

application/controller/blog.php

A `Blog` osztás specializálja az ős lehetőségeit. Itt a modell betöltését láthatjuk:

```php
class Blog extends Application {
  function __construct() {
    $this->loadModel('model_blog');
  }
```

Az index az alkalmazásunk `blog` vagy `blog/index` útvonalon lesz elérhető. A modell a teljes listázást szimulálja. (Az adatbázissal való kommunikáció már nem lesz része a példának.)

```php
  function index() {
    $articles = $this->model_blog->select();
    $data['articles'] = $articles;
    $this->loadView('view_blog',$data);
  }
```

Az `add` egy új blog beküldését valósítja meg. Ez is csak a megoldás vázát fogja tartalmazni: az űrlapot igen, de az adatok elmentését már nem tartalmazza.

```php
  function add($title="") {
    $data['title'] = $title;
    $this->loadView('view_blog_add',$data);
  }
}
```

application/model/model_blog.php

A modell – ahogy említettük – igen vázlatos:

```php
class model_blog extends Application {
  function __construct() {
    // ide az adatbázis kapcsolat, stb. jöhet
  }
  function select() {
    // ide adatbázisból történő lekérések kerülhetnének
    return array("title 1","title 2","title3");
  }
}
```

application/view/view_blog.php

A sablon állomány blogok (jelenleg csak címek) listázásához:

```php
<html>
  <head>
    <title>My Blog</title>
  </head>
  <body>
    <h1>My Blog</h1>
    <?foreach($articles as $article):?>
      <h3><?=$article?></h3>
    <?endforeach?>
  </body>
</html>
```

application/view/view_blog_add.php

A sablon állomány új blog beszúrásához:

```php
<html>
  <head>
    <title>My Blog</title>
  </head>
  <body>
    <h1>My Blog</h1>
    <h3>Title:</h3>
    <input type="text" value="<?=$title?>">
  </body>
</html>
```

Ahogy láthatjuk, egy vázlatos, de ugyanakkor az MVC logikáját jól bemutató példát ismerhettünk meg. Érdemes azonban azt is elmondani, hogy a PHP rugalmassága és a fejlesztők fantáziája miatt ugyanezen alapelveket jelentősen eltérő más megközelítésekben is megvalósíthatóak.

3.11. Sablonrendszerek

A webfejlesztésben sokszor az MVC-nél egyszerűbb módszert alkalmazunk, vagy a nézet felépítését is a sablonozásra bízhatjuk. Ennek lényege, hogy az alkalmazás logikát és a megjelenítési kódot a lehető legtisztábban válasszuk el egymástól. Így bármelyik komponens változása esetén a másik komponenshez csak kis részben, vagy egyáltalán nem kell hozzányúlni.

Az alkalmazás-logika és a megjelenítési kód szétválasztásának további előnye, hogy az alkalmazás logikáját megvalósító kódon dolgozó fejlesztő, és az oldal megjelenésével foglalkozó dizájner akár párhuzamosan, egymás zavarása nélkül is végezheti a dolgát. Ráadásul egymás munkáját is elég alapfokon ismerni az együttműködéshez.

3.11.1. Smarty

A Smarty[88] az elmúlt években méltán vált az első számú (és sok fejlesztő számára az egyetlen) sablon-motorrá. A mai változata már több mint egy sablonmotor, talán pontosabb lenne sablon-keretrendszernek nevezni.

Jellemzők

Bevezetésül a Smarty néhány fontos jellemzőjét fogjuk megvizsgálni.

Gyorstárazott

A Smarty lehetőséget nyújt az oldalaink lefordított állapotának eltárolására két különböző szinten is. Természetesen ez a lehetőség finoman konfigurálható, nem csak globálisan, hanem egyes oldalakra nézve is ki-be kapcsolható.

Konfigurációs állományok

Egy vagy több oldalon is felhasználhatók a külön állományokban tárolt konfigurációs információk. Ráadásul a konfigurációs beállításokat a dizájner önállóan is menedzselni tudja, nincs szükség a programozó beavatkozására.

Biztonságos

A sablonok nem tartalmaznak PHP kódot. Ezzel csökkentjük ugyan a dizájner lehetőségeinek körét, de ez a legtöbb esetben inkább előnyös, mint hátrányos. Ráadásul van arra is lehetőség, hogy ellenőrzött módon kiterjesszük a Smarty lehetőségeit, tehát a „probléma” áthidalható.

Könnyen kezelhető és fenntartható

A dizájnernek nem kell a PHP bonyolult szintaxisával megismerkedni, helyette egy (a HTML-hez sok tekintetben közelebb álló) sablon-nyelvet kell alkalmazni.

Változó módosítás

A használt változók értékét könnyedén módosíthatjuk a felhasználásuk előtt. Például nagybetűssé konvertálás, csonkítás, szóközök szűrése stb.

Sablon függvények

A sablon függvények segítségével összetettebb lehetőségeket is kap a dizájner az egyszerű megjelenítés helyett.

88 http://www.smarty.net/

Szűrők

A programozó teljesen kontroll alatt tudja tartani a kimenetet. Ezt elő- vagy utószűrők segítségével teheti meg. Pl. a 4.7.4. fejezetben bemutatott e-mail cím elrejtés csak akkor működőképes, ha a @ karakter nem jelenik meg a kimenetben. Egy utószűrővel meg lehet oldani, hogy az esetleg a szövegben levő @ karaktereket automatikusan – a korábbiakban leírtaknak megfelelően – elkódolja.

Kiegészítések

Az alaprendszerhez sokféle kiegészítő tölthető le a webről, és alkalmazható, vagy akár magunk is fejleszthetünk kiegészítőt.

Munkafolyamat Smartyval

A dizájner és a programozó sablonrendszer használatával nagyjából a következő módon dolgozhatnak a követelményspecifikáció elkészítése után:

- A dizájner minden egyes oldaltípushoz készít egy tisztán HTML mintát.
- A programozó PHP-ben megvalósítja az alkalmazás logikát.
- Kettőjük kommunikációjából előáll a dizájn és az alkalmazás logika csatolófelülete (interfésze), ami a változók formájában megjelenő adatok és a dizájn-elemek kapcsolatát fejezi ki.
- Ez után a kisebb (vagyis az interfészt nem befolyásoló) változások keretén belül a programozó és a dizájner ismét egymástól függetlenül végezheti a feladatát.

Smarty alapok

A Smarty fő célja az alkalmazás logika és a prezentációs logika szétválasztása.

Nézzünk példaként egy cikkeket publikáló honlapot. Az egyes cikkeket megjelenítő oldalak szerkezete hasonló lesz. Lesz az oldalnak fejléce, lábléce, címe, írója, megjelenési dátuma, tartalma stb. Érdemes megfigyelni, hogy az előző példa logikailag fogalmazta meg az oldalt alkotó elemeket. Nem beszélt `table` és `p` elemekről, se a megjelenítés vizuális, esztétikai részéről. Ráér majd a dizájner azzal foglalkozni, hogy ez HTML (vagy akár RSS, WAP) kód szintjén hogyan fog megvalósulni.

A Smarty a sablon fájl alapján egy PHP állományt fog előállítani. Mivel ez költséges művelet, gyorstárazza az eredményt, hogy a későbbi kérések gyorsabban kiszolgálhatók legyenek.

Telepítés

A Smarty magja

Ha letöltjük a Smarty legfrissebb változatát a letöltési oldalról[89], akkor egy tömörített állományt találunk. Ennek `libs` alkönyvtára tartalmazza a rendszer magját. Ennek a könyvtárnak a rendszerünkbe másolása jelenti a telepítés első lépését. A könyvtár tartalmát soha ne módosítsuk!

A Smarty `libs` könyvtárában található állományait az alkalmazásunkból el kell tudni érnünk. Éppen ezért vagy egy globálisan elérhető helyre, vagy pl. a portál gyökérmappájába célszerű helyeznünk. Fontos szerepet tölt még be a `SMARTY_DIR` változó is. Szokás szerint ez a változó tartalmazza a Smarty telepítés helyét.

Nézzük meg a lehető legegyszerűbb példát:

```php
require_once('Smarty.class.php');
$smarty = new Smarty();
```

A Smarty használatáról csak akkor beszélhetünk, ha létrehozunk egy példányt a `Smarty` osztályból.

További könyvtárak

A Smarty számára további négy könyvtárra lesz még szükség:

- `templates/`
- `templates_c/`
- `configs/`
- `cache/`

Ezek a könyvtárak tipikusan a weboldalunk gyökérmappájába kerülnek. A `templates` és `configs` könyvtárak állományait tipikusan a dizájner hozza létre: sablonok és konfigurációs állományok kerülnek beléjük.

A webszervert futtató felhasználónak írási joggal kell rendelkezni a `templates_c` és `cache` könyvtárakra, ugyanis ezekben tárolja a sablon állományokból fordított PHP állományokat, valamint a gyorstárazott HTML oldalainkat.

A Smarty belsőleg kétszintű átmeneti tárolást tartalmaz. Az első szintet az jelenti, hogy amikor először tekintünk meg egy sablont, a Smarty tiszta PHP kódra fordítja azt, és menti az eredményt. Ez a tárolási lépés megakadályozza, hogy a sablonkódokat az első kérelem után is minden alkalommal fel kelljen dolgozni. A második szint, hogy a Smarty az éppen megjelenített (jellemzően HTML) tartalom átmeneti tárolását is megvalósítja.

89 http://www.smarty.net/download

Hello Világ!

Eljutottunk az első teljes, működő alkalmazásunkhoz. Nézzük először a sablon állományt (`templates/index.tpl`):

```
<html>
<head>
<title>Info</title>
</head>
<body>
<pre>
User Information:
Name: {$name}
Address: {$address}
</pre>
</body>
</html>
```

A sablont megjelenítő kód:

```php
require_once(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty();
$smarty->assign('name','Nagy Gusztáv');
$smarty->assign('address', 'Fő u. 15.');
$smarty->display('index.tpl');
```

A példán már látszik is a sablonelvű fejlesztés legalapvetőbb eleme: a sablon oldal `{$name} és {$address}` változóját szeretnénk kicserélni az aktuális névre és címre. Erre a `smarty` objektum `assign` metódusánál láthatunk két hozzárendelést: az objektum tulajdonképpen gyűjti az ilyen típusú hozzárendeléseket. A sablon állomány feldolgozása és a sablonváltozók beillesztgetése történik meg a `display` metódus segítségével.

A kimenet a következő lesz:

```html
<html>
<head>
<title>Info</title>
</head>
<body>
<pre>
User Information:
Name: Nagy Gusztáv
Address: Fő u. 15.
</pre>
</body>
</html>
```

Hol legyen a logika?

Jelen könyvben nem részletezzük, hogyan, de a Smarty-ban elágazásokat, ciklusokat is használhatunk.

Egyesek a vezérlési szerkezetek ellen azzal érvelhetnek, hogy magában a sablonban egyáltalán nem ajánlott működési logikát elhelyezni. Az nem biztos, hogy jó, ha a megjelenítés-

ből teljesen kivonjuk a logikát, vagy azt jelenti, hogy a kimenet előállításához valójában nem tartozik logika. A megjelenítési kódnak az alkalmazásba helyezése pedig nem jobb, mint az alkalmazáslogika beépítése a megjelenítési kódba. A sablonrendszerek használatának éppen az a célja, hogy mindkét említett helyzetet elkerüljük.

Mindazonáltal a sablonokban logikát elhelyezni sok veszélyt rejt. Minél több szolgáltatás érhető el a sablonokban, annál nagyobb a kísértés, hogy maga az oldal tartalmazzon nagy mennyiségű kódot. Amíg ez a megjelenítésre korlátozódik, tartjuk magunkat az MVC mintához.

Az MVC nem arról szól, hogy a nézetből eltávolítunk minden logikát – a cél az, hogy az területre jellemző működési kódot vegyük ki belőle. A megjelenítési és működési kód között azonban nem minden esetben könnyű különbséget tenni. Számos fejlesztő számára nem csupán az a cél, hogy elválasszák a megjelenítést az alkalmazástól, hanem az, hogy a megjelenítési kód is minél kisebb legyen. A Smarty ezt a problémát nem oldja meg.

3.11.2. A PHP mint sablonnyelv

Vitathatatlan, hogy a sablonrendszerek nagyon fontos szereket töltenek be a webfejlesztésben. A Smarty mellett sok kisebb rendszer is létezik, amelyek a Smarty bonyolultsága, vagy éppen egy más típusú (pl. XML alapú) sablonnyelv igénye miatt más utakon járnak. Vannak azonban olyan fejlesztők is – köztük a szerző is –, akik szerint olyan esetben, amikor a dizájner ismeri a PHP nyelvet, felesleges lehet egy külön sablonnyelvet megtanulni és alkalmazni.

Ezzel nem a sablon logika különválasztásának, hanem csupán a külön sablon nyelv szükségességét kérdőjelezzük meg. A tanulás szempontjából tehát igenis nagyon hasznos, ha megismerjük pl. a Smartyt, de ezután akár egy sablonnyelv, akár a PHP is alkalmas lehet a feladat megoldására, ha a fejlesztők fegyelmezetten külön tudják választani a megjelenítési és alkalmazási kódot.

Érdemes itt megjegyezni, hogy pl. a Drupal és a Wordpress tartalomkezelő rendszerek estén is lehetőség van PHP alapú sablon (vagy más néven smink) kezelésre. Van azonban arra is példa[90], hogy tartalomkezelő rendszer a Smartyt használja beépített módon.

Egy minimális megoldás

A 3.6.2. fejezet `index.tpl.php` sablon állománya már megmutatta, mi a legegyszerűbb módja a PHP sablonnyelvként való használatának. A változók már léteznek, amikor `include`-oljuk a sablon állományt.

90 A legjelentősebb példa: *CMS Made Simple.* http://www.cmsmadesimple.hu/

A kimenet pufferelése

A következő példa megértéséhez szükségünk lesz a kimenet pufferelés működésének megértésére.

Ahogy azt már többször említettük, a HTTP header kiküldése után automatikusan elküldésre kerülnek a kliens oldalra a szánt (kimeneti) karakterek. Bizonyos esetekben praktikus lehet ezt a folyamatot manipulálni. A kimenet ilyenkor egy átmeneti pufferben gyűlik, ahonnan a pufferelés befejezésével tényleg elküldjük a kimenetet a kliens oldalra, esetleg eldobjuk, ha nincs rá szükségünk.

A túlzott használata nem igazán jó programozói magatartást alakít ki, de tudatos alkalmazása hasznos és megengedhető.

Sablon osztály

Az előző minimális példánál valamivel többet nyújt a következő sablon osztály. Ez az osztály *massassi*[91] megoldása alapján egy kissé testre szabott megoldás.

PHP
```
class Sablon {
  var $dir = 'sablon/';
```

A sablon állományok helye a fájlrendszerben.

PHP
```
  var $fajl;
  var $cserek = array();
```

A sablon fájl neve és a cserélendő párok.

PHP
```
  function Sablon ($f) {
    $this -> fajl = $f;
  }
```

A konstruktor mindössze a fájl nevét tárolja el.

PHP
```
  function berak($nev, $ertek) {
    if (is_object($ertek)) {
      $this -> cserek[$nev] = $ertek->cserel();
    } else {
      $this -> cserek[$nev] = $ertek;
    }
  }
```

A leggyakrabban használt metódus, hiszen ezzel definiálhatók a cserélendő párok.

PHP
```
  function berakTomb($tomb) {
    if (is_array($tomb)) {
      $this -> cserek = ($this -> cserek) + $tomb;
    }
  }
```

Hasznos függvény, ha a cserélendő párok már eleve tömbben helyezkednek el.

91 http://www.massassi.com/php/articles/template_engines/

```php
  function cserel() {
    extract($this -> cserek);
    $cimsor;
    ob_start();
    include($this -> dir . $this -> fajl . '.tpl.php');
    $tartalom = ob_get_contents();
    ob_end_clean();
    return $tartalom;
  }
```

A tényleges sablon műveletet, a cserék megvalósítását végzi. Érdemes megfigyelni, menynyire egyszerűvé teszi a kódot a kimenet pufferelése (`ob_*` függvények).

```php
  function torol() {
    $this -> cserek = array();
  }
}
```

Végül a `töröl` függvény lehetővé teszi, hogy újabb sablon objektum példányosítása nélkül is újra használhassuk a sablonállományunkat, pl. több azonos sablon alapján gyártott blokk generálására.

Blogbejegyzés

Nézzünk egy konkrét példát a nagygusztav.hu oldal 2006-os verziójából. A következő sablon állomány (`ujBejegyzesekBlokk.tpl.php`) egy blogbejegyzés előzetest fog készíteni a kezdőoldalra.

```php
<? /* Csak az előzetes megjelenítésére, hozzászólások számával */ ?>
<div class="hir">
<? if ($lehethozza || $tartalom) { ?>
  <h2><a href="<?= $index ?>?<?= $blognev ?>/<?=
        $url ?>"><?= $hirCim ?></a></h2>
<? } else { ?>
  <h2><?= $hirCim ?></h2>
<? }?>

<? if ($datum) {?>
  <acronym class="datum" title="<?= $datum['hosszu'] ?>">
    <span class="honap"><?= $datum['honap'] ?></span>
    <span class="nap"><?= $datum['nap'] ?></span>
  </acronym>
<? } ?>

<?= $hirSzoveg ?>
<? if ($lehethozza || $tartalom) { ?>
<p class="hozzaszolasszam">
<? if ($tartalom) { ?>
  <a href="<?= $index ?>?<?= $blognev ?>/<?= $url ?>">
        Teljes bejegyzés&raquo;</a>
<? }?>
```

```
<? if ($lehethozza) { ?>
Eddig
<?  if ($hozzaszolasSzam == 0) { ?>
 nincs
<?  } else {?>
  <?= $hozzaszolasSzam ?>
<?  } ?>
<a href="<?= $index ?>?<?= $blognev ?>/<?=
    $url ?>#hozza">hozzászólás</a>.
<? }?>
</p>
<?  } ?>
</div>
```

A következő kód használja a sablont:

```
$ujBejegyzesek = $this -> adatbazis -> ujBejegyzesek(10);
foreach ($ujBejegyzesek as $k => $e) {
  $ujBejegyzesek[$k]['blogcim'] = $this ->
      joOldalak[$e['blognev']]['cim'];
}
$sablon = new Sablon('ujBejegyzesekBlokk');
$sablon -> berakTomb($GLOBALS['portal_adatok']);
$sablon -> berak('ujBejegyzesek', $ujBejegyzesek);
$szoveg .= $sablon -> cserel();
```

PHP

Ezzel el is jutottunk egy jól használható sablonkezelés alapjaihoz.

Végül Medovarszki Mihály: *PHP, mint sablonmotor egyszerűen* című cikkét[92] is ajánljuk a téma kapcsán.

3.12. Tartalomkezelő rendszerek

Ma a weboldal-tulajdonosok általában fontosnak tartják, hogy a weboldaluk tartalmát, sőt akár a strukturális felépítését is maguk tudják kezelni, mindenféle fejlesztői beavatkozás nélkül. Ilyen esetekben jó megközelítés lehet a tartalomkezelő rendszerek alkalmazása.

A Wikipédia[93] szerint *„a tartalomkezelő rendszer (angolul Content Management System, CMS) olyan szoftverrendszer, amely nem strukturált információk, mint például az internetes portálok, akár több felhasználó általi elkészítését, kezelését, és tárolását segíti. Továbbá gondoskodik a tartalmak strukturált megjelenítéséről, statisztikák készítéséről, kiegészítő funkciók integrálásáról."*

Ma már sokszor nincs arra szükség, hogy saját webalkalmazás, vagy tartalomkezelő rendszer fejlesztésével elégítsük ki a megrendelő igényeit. Hiszen vannak nyílt forrású, nagyszerűen bevált tartalomkezelő rendszerek. (A szerző a *Drupalt*[94] ajánlja a keresők számára,

92 http://medovarszki.blog.prog.hu/archives/10-PHP,-mint-sablonmotor-egyszeruen.html
93 http://hu.wikipedia.org/wiki/Tartalomkezelő_rendszer
94 Http://drupal.hu

és a *Drupal 6 alapismeretek*[95] című könyvét.) De sok esetben az is jó megoldás, ha egy általános célú és nagy tudású (ezzel együtt nagyobb erőforrás-igényű) rendszer helyett egy pehelysúlyú, az adott feladatra tervezett CMS-t használunk.

Jelen fejezet célja, hogy egy rövid betekintést nyújtson a tartalomkezelő rendszerek fejlesztésébe. Erre a célra *Matt Doyle*: *Build a CMS in an Afternoon with PHP and MySQL* című nagyszerű cikkét[96] vesszük alapul. A forráskód letölthető Matt honlapjáról is.

A fejezet során néhány alapvető továbbfejlesztési lehetőségre is ki fogunk térni.

3.12.1. Célok

Kezdjük a célok megfogalmazásával. A mini tartalomkezelő rendszerünknek a következőket kell tudnia.

A publikus felület (*front end*):

- Kezdőoldal az 5 legfrissebb cikkel
- Listázó oldal az összes cikkel
- Egy cikk megtekintésére szolgáló oldal

Az adminisztrációs felület (*back end*):

- Adminisztrátor belépés, kilépés
- Cikkek listája
- Új cikkek listája
- Létező cikk szerkesztése
- Új cikk létrehozása

Az egyes cikkeknek legyen címük, összefoglalójuk, törzsük és beküldési dátumuk.

Végül néhány továbbfejlesztési lehetőség:

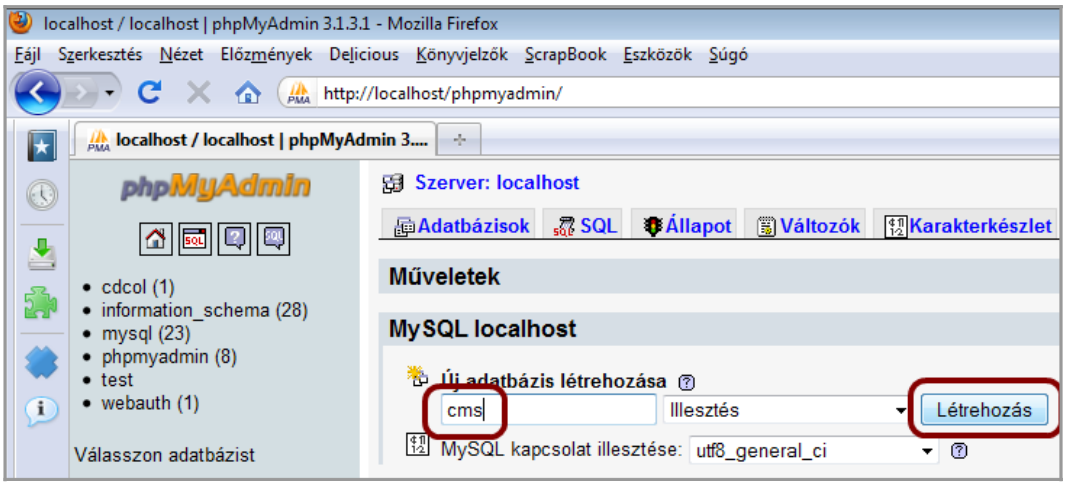
- Hozzászólások lehetősége
- Látogatók regisztrációja
- Több szerző kezelése

95 http://nagygusztav.hu/drupal-6-alapismeretek
96 http://www.elated.com/articles/cms-in-an-afternoon-php-mysql/

3.12.2. Adatbázis felépítése

Minden alkalmazásnak, így egy tartalomkezelő rendszernek is szüksége van arra, hogy az adatokat hosszú távon meg tudjuk őrizni. Erre ma a legelterjedtebb megoldás a 3.4. fejezetben bemutatott relációs adatbázis-kezelők használata.

Az alkalmazásunkat érdemes önálló adatbázisba helyezni. Pl. lokális gépen hozzuk létre a PhpMyAdmin segítségével egy `cms` nevű adatbázist a `localhost/phpmyadmin` webcímen:



58. ábra. Adatbázis létrehozása

Jelen verzióban egyetlen `articles` adattáblát használunk, melyben a cikkeket fogjuk tárolni. Ennek létrehozására használhatjuk a következő SQL parancsot:

```SQL
CREATE TABLE articles (
  id              smallint unsigned NOT NULL PRIMARY KEY auto_increment,
  publicationDate date NOT NULL,
  title           varchar(255) NOT NULL,
  summary         text NOT NULL,
  content         mediumtext NOT NULL
);
```

Nézzük sorba a mezők jelentését:

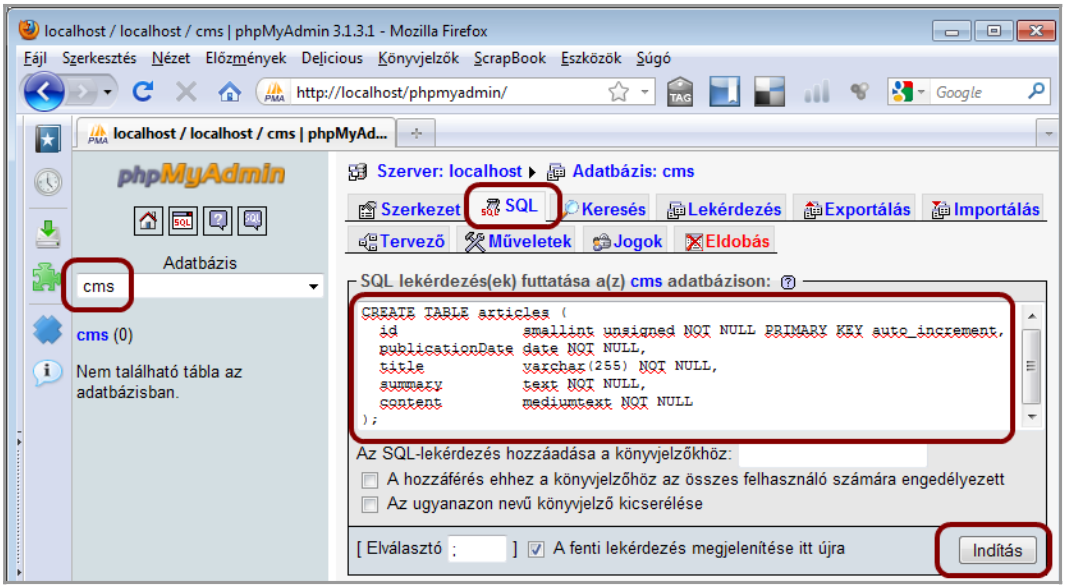
- Az `id` mezőt minden olyan táblánál létre szokás hozni elsődleges kulcsként, amelyik alapvető adattárolásra szolgált. (Pl. a több-több kapcsolatot megvalósító tábláknál nem szokás.) A jelen verziónkban az `id`-nek még nincs akkora jelentősége, de ha pl. a hozzászólások lehetőségét is be akarjuk építeni, akkor a hozzászólások tárolására szolgáló `comments` táblában egy `articles_id` nevű mezőt vezethetünk be külső kulcsként.

- A `publicationDate` csak a közzététel dátumát tartalmazza. Ha pontos időpontot is szeretnénk tárolni, a `datetime` típust érdemes használnunk.
- A `title` mező a cikk címét fogja tárolni. Ezt a listázó oldalakon, de az egyes cikkek oldalain is fel fogjuk használni.
- A `summary` mező a front-end listázó oldalakon fog a cím alatt megjelenni.
- A `content` mező a bejegyzés szövegét tartalmazza.

Több lehetőségünk is van, hogy a táblánkat létrehozzuk.

SQL futtatás

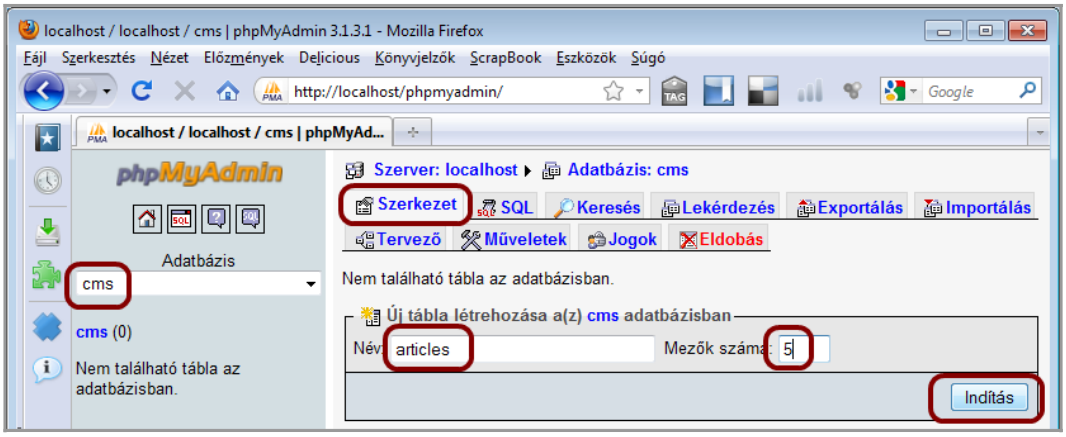
A `localhost/phpmyadmin` címen kiválasztjuk az adatbázisunkat, majd az SQL fülön futtatjuk az SQL parancsot (59. ábra)



59. ábra. SQL parancs futtatása

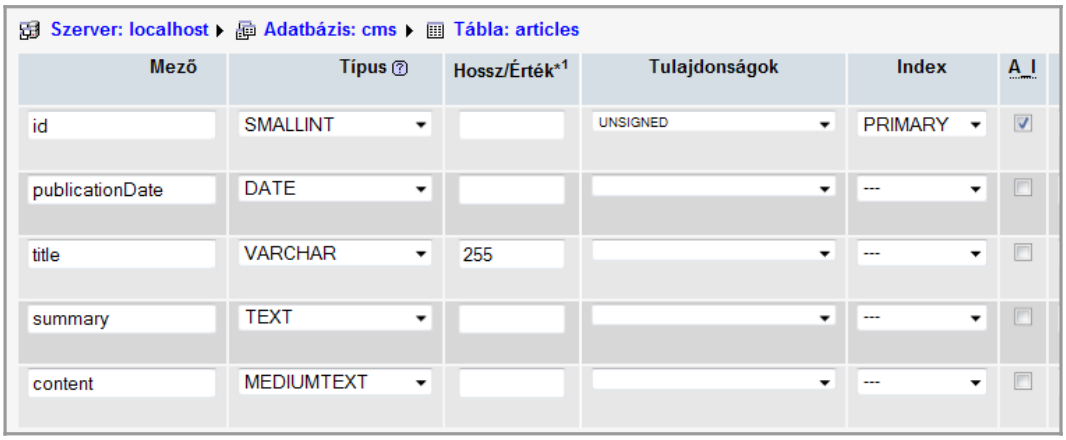
Varázsló használata

A `localhost/phpmyadmin` címen kiválasztjuk az adatbázisunkat, majd a *Szerkezet* fülön az *Új tábla* nevét és a *Mezők számát* megadva elindítjuk a varázslót (60. ábra). Mivel itt igen sok lehetőség van, csak megfelelő hozzáértés esetén válasszuk ezt a mdot.



60. ábra. Adattábla létrehozása varázsóval

Az 61. ábra a mezők beállításait mutatja. (Az ábra csak a beállítandó oszlopokat tartalmazza.)



61. ábra. Tábla mezőinek létrehozása

PHP futtatás

A 3.4.2. forráskódjának megfelelően futtathatjuk is az SQL parancsot.

3.12.3. Konfiguráció

Webalkalmazások készítésénél általában létre szoktunk hozni egy konfigurációs állományt, ahol a futtatási környezet alap információit helyezünk el a programunk számára. Ha esetleg költöztetni kell az alkalmazást, elegendő ehhez az állományhoz hozzányúlni.

A `config.php` tartalma a következő legyen.

A fejlesztés idejére kapcsoljuk be a figyelmeztetéseket:

```php
ini_set( "display_errors", true );
```

A következő `define` utasítások konstansokat hoznak létre, hogy a forráskódok további részében már ne kelljen változtatni egy esetleges szerver költöztetés esetén.

Először az adatbázishoz kapcsolódás részletei következnek. Felhasználónév és jelszó az XAMPP alapértelmezése szerint:

```php
define( "DB_DSN", "mysql:host=localhost;dbname=cms" );
define( "DB_USERNAME", "root" );
define( "DB_PASSWORD", "" );
```

A követező konstansok az osztályokat és a sablonokat tartalmazó könyvtárak helyét definiálják:

```php
define( "CLASS_PATH", "classes" );
define( "TEMPLATE_PATH", "templates" );
```

Mennyi cikk legyen a címlapon:

```php
define( "HOMEPAGE_NUM_ARTICLES", 5 );
```

Az adminisztrátor felhasználóneve és jelszava:

```php
define( "ADMIN_USERNAME", "admin" );
define( "ADMIN_PASSWORD", "mypass" );
```

Abban az esetben, ha több felhasználót is szeretnénk megkülönböztetni, akkor majd egy `users` nevű táblát érdemes használni a felhasználók adatainak tárolására.

Biztonsági szempontból nem közvetlenül a jelszót, hanem valamilyen kódolt változatát szokás tárolni. Ma a legelterjedtebb megoldás az md5 függvény használata, esetleg további komplexitásnövelő, egyedi ötletek beépítésével. Ezekre szép példákat láthatunk az `md5` függvény dokumentációjában[97].

Amikor a belépéskor bekérjük a jelszót, ugyanazt a kódolást végezzük el a kapott jelszóval, és végül ezt hasonlítjuk a tárolt kódolt alakkal. Egyezés esetén a jelszó helyességét biztosnak vehetjük.

Végül az `Article` osztály definícióját futtatjuk. Erre minden esetben szükség lesz a jelenlegi verzióban:

```php
require( CLASS_PATH . "/Article.php" );
```

3.12.4. Az `Article` osztály

A cikkek kezelése, az adatbázissal való kommunikáció egyetlen osztály felelőssége. Ez az osztály fogja megoldani egy új cikk esetén az adatbázisba történő mentést, a cikkek lekérését adatbázisból, stb.

97 http://php.net/manual/en/function.md5.php

A későbbi front-end és back-end kód ezt az osztályt fogja használni.

Adattagok

A `classes/Article.php` kódja:

```
<?php
class Article
```
PHP 5

Az osztály adattagjai az adatbázis mezőinek felelnek meg:

```
  public $id = null;
  public $publicationDate = null;
  public $title = null;
  public $summary = null;
  public $content = null;
```
PHP 5

Magyarázatra egyedül az `$id` szorul. Ennek értéke arra utal, hogy az objektumnak megfelelő rekord az adatbázisban már létezik-e. Akkor lesz `null`-tól eltérő, ha a rekord létezik.

A konstruktor

A konstruktor vagy az adatbázisból, vagy a felhasználótól származó adatokat alapul véve hoz létre egy objektumot. Az adatokat a `$data` tömbben kapja, és megfelelő ellenőrzést végez rajtuk.

```
  public function __construct( $data=array() ) {
    if ( isset( $data['id'] ) ) $this->id = (int) $data['id'];
    if ( isset( $data['publicationDate'] ) )
        $this->publicationDate = (int) $data['publicationDate'];
    if ( isset( $data['title'] ) )
        $this->title =
            preg_replace ("/[^\.\,\-\_\'\"\@\?\!\:\$ a-zA-Z0-9()]/",
                "", $data['title'] );
    if ( isset( $data['summary'] ) )
        $this->summary =
            preg_replace ( "/[^\.\,\-\_\'\"\@\?\!\:\$ a-zA-Z0-9()]/",
                "", $data['summary'] );
    if ( isset( $data['content'] ) )
        $this->content = $data['content'];
  }
```
PHP 5

Az űrlapból származó adatokkal hívjuk a konstruktort:

```
  public function storeFormValues ( $params ) {
    $this->__construct( $params );
```
PHP 5

Ha a dátum beviteli mező ki lett töltve, akkor a szövegből dátumot próbálunk előállítani:

```
    if ( isset($params['publicationDate']) ) {                    PHP 5
      $publicationDate = explode ( '-', $params['publicationDate'] );
      if ( count($publicationDate) == 3 ) {
        list ( $y, $m, $d ) = $publicationDate;
        $this->publicationDate = mktime ( 0, 0, 0, $m, $d, $y );
      }
    }
  }
```

Egy cikk betörlése

A `getById` függvény az `$id` alapján tölti be az adatbázis megfelelő rekordját egy új objektumba.

```
  public static function getById( $id ) {                         PHP 5
```

A PDO (PHP Data Objects[98]) segítségével adatbázis típustól függetlenül tudjuk a kommunikációt megvalósítani. A `$conn` objektum a PDO osztály egy példánya lesz.

```
    $conn = new PDO( DB_DSN, DB_USERNAME, DB_PASSWORD );          PHP 5
```

Az SQL parancs összeállításánál igen fontos, hogy a változó részek ne nyers adatként, hanem ellenőrzötten kerüljenek az adatbázisba. Itt – egész számként – a preparált lekérdezés használata szép megoldás:

```
    $sql = "SELECT *, UNIX_TIMESTAMP(publicationDate)             PHP 5
                AS publicationDate
            FROM articles WHERE id = :id";
    $st = $conn->prepare( $sql );
    $st->bindValue( ":id", $id, PDO::PARAM_INT );
```

A preparált lekérdezés kész a futtatásra (`execute`). Lekérjük az első találatot a `fetch` függvénnyel. (Itt az SQL lekérdezésnek csak egy rekord kimenete lehet.)

```
    $st->execute();                                               PHP 5
    $row = $st->fetch();
    $conn = null;
```

Végül létrehozzuk az objektumot:

```
    if ( $row ) return new Article( $row );                       PHP 5
  }
```

Cikkek listázása

A `getList` függvény lekéri a cikkek listáját az adatbázisból. A két paramétere opcionális. Az első a cikkek darabszámát korlátozza, a második a rendezettségre van hatással.

```
  public static function getList( $numRows=1000000,              PHP 5
        $order="publicationDate DESC" ) {
    $conn = new PDO( DB_DSN, DB_USERNAME, DB_PASSWORD );
```

98 http://php.net/manual/en/book.pdo.php

Az SQL lekérdezés összeállításánál a `mysql_escape_string` függvényt használjuk az *SQL injection* támadások elkerülése érdekében.

```
    $sql = "SELECT SQL_CALC_FOUND_ROWS *,                                PHP 5
            UNIX_TIMESTAMP(publicationDate) AS publicationDate
        FROM articles
        ORDER BY " . mysql_escape_string($order) . " LIMIT :numRows";
    $st = $conn->prepare( $sql );
    $st->bindValue( ":numRows", $numRows, PDO::PARAM_INT );
    $st->execute();
```

A `$list` tömbbe fogjuk gyűjteni az egyes cikkek objektumait.

```
    $list = array();                                                     PHP 5
    while ( $row = $st->fetch() ) {
      $article = new Article( $row );
      $list[] = $article;
    }
```

Végül lekérdezzük a talált sorok számát, lezárjuk az adatbázis-kapcsolatot, és egy tömbként visszaadjuk az objektumokat és a találatok számát.

```
    $sql = "SELECT FOUND_ROWS() AS totalRows";                           PHP 5
    $totalRows = $conn->query( $sql )->fetch();
    $conn = null;
    return ( array ( "results" => $list, "totalRows" => $totalRows[0] ) );
  }
```

Egy új cikk beszúrása

Az `insert` függvény egy objektum adatait írja ki az adatbázisba.

```
  public function insert() {                                             PHP 5
```

Ha az objektum már rendelkezik `$id`-vel, akkor nem szabad beszúrni, mert az `$id`-t csak adatbázisból kaphatta, és akkor nem szabad újból beszúrni.

```
    if ( !is_null( $this->id ) )                                         PHP 5
        trigger_error ( "Article::insert(): Attempt to insert an
              Article object that already has its ID property set
              (to $this->id).", E_USER_ERROR );
```

Összeállítjuk és lefuttatjuk az SQL lekérdezést.

```
    $conn = new PDO( DB_DSN, DB_USERNAME, DB_PASSWORD );                 PHP 5
    $sql = "INSERT INTO articles
          ( publicationDate, title, summary, content )
          VALUES
          ( FROM_UNIXTIME(:publicationDate), :title, :summary, :content )";
    $st = $conn->prepare ( $sql );
    $st->bindValue( ":publicationDate", $this->publicationDate,
             PDO::PARAM_INT );
    $st->bindValue( ":title", $this->title, PDO::PARAM_STR );
    $st->bindValue( ":summary", $this->summary, PDO::PARAM_STR );
    $st->bindValue( ":content", $this->content, PDO::PARAM_STR );
    $st->execute();
```

Végül a beszúráskor kapott `id`-t eltárioljuk.

```
    $this->id = $conn->lastInsertId();                                    PHP 5
    $conn = null;
  }
```

Cikk módosítása

Az `update` függvény segítségével az objektum állapotában történt változásokat az adatbázisba is visszavezethetjük. Ha az id még nincs kitöltve, akkor nincs a cikk az adatbázisban, nincs mit felülírni:

```
  public function update() {                                              PHP 5
    if ( is_null( $this->id ) )
      trigger_error ( "Article::update(): Attempt to update an Article"
          . " object that does not have its ID property set.",
          E_USER_ERROR );
```

Adatbázis kapcsolat, SQL parancs összeállítása hasonlóan a korábbi példákhoz:

```
    $conn = new PDO( DB_DSN, DB_USERNAME, DB_PASSWORD );                  PHP 5
    $sql = "UPDATE articles SET"
           . " publicationDate=FROM_UNIXTIME(:publicationDate),"
           . " title=:title, summary=:summary, content=:content"
           . " WHERE id = :id";
    $st = $conn->prepare ( $sql );
    $st->bindValue( ":publicationDate", $this->publicationDate,
        PDO::PARAM_INT );
    $st->bindValue( ":title", $this->title, PDO::PARAM_STR );
    $st->bindValue( ":summary", $this->summary, PDO::PARAM_STR );
    $st->bindValue( ":content", $this->content, PDO::PARAM_STR );
    $st->bindValue( ":id", $this->id, PDO::PARAM_INT );
    $st->execute();
    $conn = null;
  }
```

Cikk törlése

A `delete` függvény törli a cikket az adatbázisból az `id` alapján.

```
  public function delete() {                                              PHP 5
    if ( is_null( $this->id ) )
      trigger_error ( "Article::delete(): Attempt to delete an Article"
          . " object that does not have its ID property set.",
          E_USER_ERROR );
    $conn = new PDO( DB_DSN, DB_USERNAME, DB_PASSWORD );
    $st = $conn->prepare ( "DELETE FROM articles WHERE id = :id LIMIT 1" );
    $st->bindValue( ":id", $this->id, PDO::PARAM_INT );
    $st->execute();
    $conn = null;
  }
}
?>
```

Ezzel az `Article` osztály kódját áttekintettük.

3.12.5. Front-end

Nézzük meg, hogyan találkozik a látogató a weboldal publikus részével. A következő `index.php` állomány épít az `Article` osztályra, és a kimenetet sablonok segítségével állítja elő.

Először is a *Front Controller* mintát (3.6.2. fejezet) valósítja meg:

PHP 5

```
<?php
require( "config.php" );
$action = isset( $_GET['action'] ) ? $_GET['action'] : "";
switch ( $action ) {
  case 'archive':
    archive();
    break;
  case 'viewArticle':
    viewArticle();
    break;
  default:
    homepage();
}
```

Archívum

URL: `index.php?action=archive`

Az `archive` függvény betölti az összes cikk bevezető adatait az `Article::getList` függvénnyel, majd az `archive.php` sablont hívja be (62. ábra).

PHP 5

```
function archive() {
  $results = array();
  $data = Article::getList();
  $results['articles'] = $data['results'];
  $results['totalRows'] = $data['totalRows'];
  $results['pageTitle'] = "Article Archive | Widget News";
  require( TEMPLATE_PATH . "/archive.php" );
}
```

62. ábra. Archív cikkek

Cikk megjelenítése

URL példa: `index.php?action=viewArticle&articleId=7`

A `viewArticle` függvény egy cikk megjelenítését (63. ábra) oldja meg, ha megfelelő GET paramétert kapott. Különben a címlapot jeleníti meg.

PHP 5

```
function viewArticle() {
  if ( !isset($_GET["articleId"]) || !$_GET["articleId"] ) {
    homepage();
    return;
  }
  $results = array();
  $results['article'] = Article::getById( (int)$_GET["articleId"] );
  $results['pageTitle'] = $results['article']->title . " | Widget News";
  require( TEMPLATE_PATH . "/viewArticle.php" );
}
```

63. ábra. Cikk megjelenítése

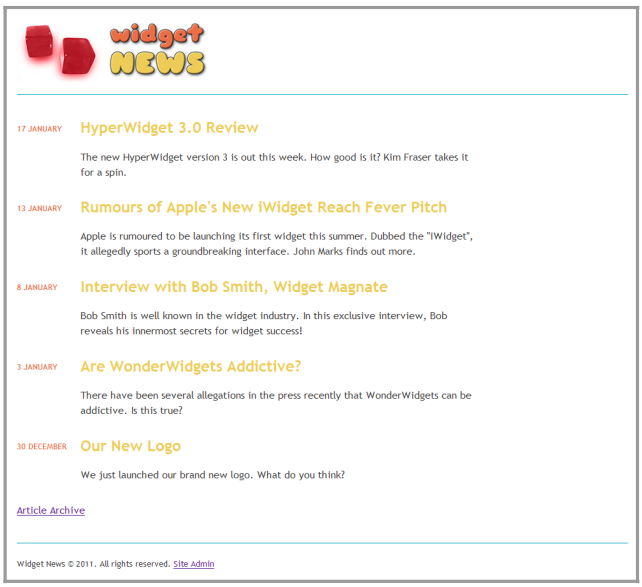
A címlap

URL: `index.php`

A `homepage` függvény a címlap számára kéri le legfrissebb néhány cikk adatait, és jeleníti meg a `homepage.php` sablon segítségével (64. ábra).

PHP 5

```
function homepage() {
  $results = array();
  $data = Article::getList( HOMEPAGE_NUM_ARTICLES );
  $results['articles'] = $data['results'];
  $results['totalRows'] = $data['totalRows'];
  $results['pageTitle'] = "Widget News";
  require( TEMPLATE_PATH . "/homepage.php" );
}
```



64. ábra. A címlap néhány cikkel

A látogatók lehetőségeit ezzel áttekintettük.

3.12.6. Back-end

Az adminisztrátori felület hasonlóan működik a publikus felülethez. Az `admin.php` végzi a vezérlést. Fontos eltérés a felhasználó kezelése, a kilépés, belépés, stb.

A felhasználó belépett állapotát a `username` munkamenet változó jelzi.

PHP 5
```
<?php
require( "config.php" );
session_start();
$action = isset( $_GET['action'] ) ? $_GET['action'] : "";
$username = isset( $_SESSION['username'] ) ? $_SESSION['username'] : "";
```

Ha a felhasználó nincs még belépve, és nem a kilépés vagy belépés folyamatában van, akkor átirányítjuk a beléptetésre, a `login` űrlapra.

PHP 5
```
if ( $action != "login" && $action != "logout" && !$username ) {
  login();
  exit;
}
```

Ha a felhasználó már belépett, akkor foglalkozunk a GET paraméter alapján a kérésével.

```
switch ( $action ) {
  case 'login':
    login();
    break;
  case 'logout':
    logout();
    break;
  case 'newArticle':
    newArticle();
    break;
  case 'editArticle':
    editArticle();
    break;
  case 'deleteArticle':
    deleteArticle();
    break;
  default:
    listArticles();
}
```

PHP 5

Beléptetés

URL: `admin.php?action=login`

A `login` függvény végzi az űrlap megjelenítését (65. ábra) és feldolgozását is.

```
function login() {
  $results = array();
  $results['pageTitle'] = "Admin Login | Widget News";
```

PHP 5

65. ábra. Login űrlap

Ha a látogató már visszaküldte az űrlapot, akkor ellenőrizzük a nevet és jelszót. Ha korrekt adatokat kapott, a munkamenetbe bejegyzi a felhasználónevet, és átküldi az adminisztrációs oldalra.

```
  if ( isset( $_POST['login'] ) ) {
    if ( $_POST['username'] == ADMIN_USERNAME &&
         $_POST['password'] == ADMIN_PASSWORD ) {
      $_SESSION['username'] = ADMIN_USERNAME;
      header( "Location: admin.php" );
    }
```

PHP 5

Ha nem jók az adatok, hibaüzenet kíséretében jeleníti meg az űrlapot.

PHP 5
```
    else {
      $results['errorMessage'] =
          "Incorrect username or password. Please try again.";
      require( TEMPLATE_PATH . "/admin/loginForm.php" );
    }
  }
```

Először üresen jeleníti meg a beléptető űrlapot a látogató előtt:

PHP 5
```
  else {
    require( TEMPLATE_PATH . "/admin/loginForm.php" );
  }}
```

Kiléptetés

URL: `admin.php?action=logout`

A kilépés linkre kattintva a `logout` függvény törli a felhasználó nevét a munkamenetből, így a továbbiakban nem tekintjük belépett felhasználónak.

PHP 5
```
function logout() {
  unset( $_SESSION['username'] );
  header( "Location: admin.php" );
}
```

Új cikk beküldése

URL: `admin.php?action=newArticle`

A `newArticle` függvény vezérli a folyamat minden lépését. legelőször a legutolsó `else` ág fog lefutni, és létrehozni az üres űrlapot (66. ábra). Érdemes megfigyelni, hogy az `editArticle.php` sablont használja az új cikk beviteléhez is.

66. ábra. Cikk beküldése

A `$results['formAction']` értéke az űrlap `action` paramétereként lesz felhasználva. Ebből tudja majd később, hogy nem szerkeszteni, hanem beküldeni szeretnénk egy új cikket.

PHP 5
```
function newArticle() {
  $results = array();
  $results['pageTitle'] = "New Article";
  $results['formAction'] = "newArticle";
```

Ha menthetőek az adatok, menti az `insert` függvénnyel, majd átirányítja a

PHP 5
```
  if ( isset( $_POST['saveChanges'] ) ) {
    $article = new Article;
    $article->storeFormValues( $_POST );
    $article->insert();
    header( "Location: admin.php?status=changesSaved" );
  }
```

Ha a `Cancel` gombra kattint, akkor eldobjuk az oldalt.

PHP 5
```
  elseif ( isset( $_POST['cancel'] ) ) {
    header( "Location: admin.php" );
  }
```

Végül itt következik az üres űrlap, amit először lát az adminisztrátor.

```
  else {                                                                PHP 5
    $results['article'] = new Article;
    require( TEMPLATE_PATH . "/admin/editArticle.php" );
  }
}
```

Cikk szerkesztése

URL példa: `admin.php?action=editArticle&articleId=1`

Az `editArticle` függvény a megadott kódú cikket engedi szerkeszteni a 66. ábra űrlapján. természetesen ekkor a cím más, és az űrlap mezők eleve kitöltöttek.

```
function editArticle() {                                                PHP 5
  $results = array();
  $results['pageTitle'] = "Edit Article";
  $results['formAction'] = "editArticle";
```

Ha már visszaküldte a látogató az űrlapot mentési szándékkal, először betöltjük az adatbázisból a jelenlegi állapotot. Sikertelenség esetén hiba oldalra küldjük az adminisztrátort. Egyébként mentünk.

```
  if ( isset( $_POST['saveChanges'] ) ) {                               PHP 5
    if ( !$article = Article::getById( (int)$_POST['articleId'] ) ) {
      header( "Location: admin.php?error=articleNotFound" );
      return;
    }
    $article->storeFormValues( $_POST );
    $article->update();
    header( "Location: admin.php?status=changesSaved" );
  }
```

Ha a `Cancel` gombra kattint, akkor eldobjuk az oldalt.

```
  elseif ( isset( $_POST['cancel'] ) ) {                                PHP 5
    header( "Location: admin.php" );
  }
```

Végül itt következik az üres űrlap, amit először lát az adminisztrátor.

```
  else {                                                                PHP 5
    $results['article'] = Article::getById( (int)$_GET['articleId'] );
    require( TEMPLATE_PATH . "/admin/editArticle.php" );
  }
}
```

Cikk törlése

URL: `admin.php?action=deleteArticle&articleId=1`

A `deleteArticle` függvény törli a megkapott azonosítóval rendelkező cikket. Ellenőrzésként itt is betölti a cikkel az adatbázisból.

```
function deleteArticle() {                                          PHP 5
  if ( !$article = Article::getById( (int)$_GET['articleId'] ) ) {
    header( "Location: admin.php?error=articleNotFound" );
    return;
  }
  $article->delete();
  header( "Location: admin.php?status=articleDeleted" );
}
```

Cikkek listázása

URL: `admin.php`

A `listArticles` függvény a listázás mellett adminisztrációs üzeneteket és állapotinformációkat is megjelenít.

```
function listArticles() {                                           PHP 5
  $results = array();
  $data = Article::getList();
  $results['articles'] = $data['results'];
  $results['totalRows'] = $data['totalRows'];
  $results['pageTitle'] = "All Articles";
```

Hibaüzenet esetén megjeleníti:

```
  if ( isset( $_GET['error'] ) ) {                                  PHP 5
    if ( $_GET['error'] == "articleNotFound" )
      $results['errorMessage'] = "Error: Article not found.";
  }
```

Állapotinformáció esetén megjeleníti:

```
  if ( isset( $_GET['status'] ) ) {                                 PHP 5
    if ( $_GET['status'] == "changesSaved" )
      $results['statusMessage'] = "Your changes have been saved.";
    if ( $_GET['status'] == "articleDeleted" )
      $results['statusMessage'] = "Article deleted.";
  }
```

Végül a sablon állomány végzi a tényleges kiírást.

```
  require( TEMPLATE_PATH . "/admin/listArticles.php" );             PHP 5
}
?>
```

3.12.7. A kinézet

A mini CMS-ünk szép megoldásai közül nem hiányozhat a jól kitalált sablonkezelés sem.

Terjedelmi okokból a sablon állományok teljes kódját nem tudjuk itt idézni. Azok értelmezését is az olvasóra bízzuk. Az eddigi forráskódok ismeretében a sablonok feldolgozása könnyen áttekinthető.

3.12.8. Nyílt forrású tartalomkezelő rendszerek

A fejezet eddigi részében egy vázlatos, de a fontos alapfunkciókat ellátó CMS rendszert ismerhettünk meg. Jól látszik, hogy ez az állapot nagyon messze van meg attól, hogy komplex weboldalakat készítsünk vele. Érdemes azonban megismerkedni a jelenleg is elérhető, azonnal felhasználható, nyílt forrású tartalomkezelő rendszerekkel. A szerző mindenképpen ajánlja bevezetésként a *Drupal 6 alapismeretek*[99] című könyvét.

3.13. Keretrendszerek

A tartalomkezelő rendszerek mellett igen hasznos megismernünk egy vagy több keretrendszer (*framework*) használatát is.

Először is nézzük, mi a különbség a tartalomkezelő rendszerek és a keretrendszerek között.

Tartalomkezelő rendszer	Keretrendszer
végfelhasználónak készül	programozónak készül
programozási ismeret nélkül is használható	erős programozási ismeretek szükségesek
gyors megoldás, azonnal használható	igen komoly programozói munka szükséges hozzá
alap feladatokra van kitalálva	bármilyen feladatra alkalmas lehet

13. táblázat. A tartalomkezelő rendszerek és keretrendszerek összehasonlítása

3.13.1. Yii

Terjedelmi okokból itt csak azt tudjuk ajánlani, hogy a ma egyik legkorszerűbbnek számító, és *Méhész Imre*[100] nagyszerű munkásságának köszönhetően magyar szakirodalommal is rendelkező *Yii framework*[101] tanulmányozásával kezdje a témát e sorok olvasója.

Mi is a Yii?[102]

A Yii egy nagyteljesítményű, komponens alapú PHP keretrendszer nagyszabású webalkalmazások fejlesztéséhez. Maximális újrahasznosíthatóságot tesz lehetővé a webprogramozás terén, és a fejlesztési folyamatot is jelentősen meggyorsítja.

99 Letölthető a http://nagygusztav.hu/drupal-6-alapismeretek címről.
100 http://mehesz.net/
101 http://www.yiiframework.com/
102 http://yiihun.blogspot.com/2011/01/mi-is-yii.html alapján

A Yii egy általános webprogramozási keretrendszer, ami voltaképpen felhasználható bármilyen webalkalmazás fejlesztéséhez. Mivel könnyűsúlyú (light-weighted) és kifinomult gyorsítótárazási megoldásokat nyújt, kifejezetten alkalmas nagy forgalmú alkalmazások, úgy mint portálok, fórumok, tartalomkezelők (CMS), e-kereskedelmi rendszerek, stb. fejlesztésére.

Mint a legtöbb PHP keretrendszer, a Yii egy MVC kertrendszer. A Yii felülmúlja a többi a keretrendszert a következőkben: hatékony, képességekben gazdag, és világosan dokumentált. A Yii a kezdetektől fogva komoly webalkalmazások fejlesztésére lett tervezve. Nem egy projekt melléktermeke vagy netán harmadik féltől származó kódok összefércelt változata, hanem a szerzők webalkalmazások fejlesztésében és vizsgálatában szerzett tekintélyes tapasztalatának eredménye és a legnépszerűbb webprogramozási keretrendszerek és alkalmazások tükröződése.

A Yii megismeréséhet ét forrást ajánlunk:

- Méhész Imre: Wiki à la Yii
  http://weblabor.hu/cikkek/wiki-a-la-yii
- A Yii (PHP) keretrendszer, magyarul
  http://yiihun.blogspot.com/

4

Kliens oldali működés

A JavaScript egy olyan kliens oldali szkriptnyelv, amely mára lehetővé tette az ún. *RIA* (*Rich Internet Application*), vagyis a hagyományos asztali (*desktop*) alkalmazásokhoz hasonlóan hatékony és kényelmes alkalmazások fejlesztését.

Példaként érdemes pl. a Google néhány szolgáltatásával megismerkedni. A *Gmail*, a *Google naptár* vagy a *Google dokumentumok* funkcionalitása a JavaScript nélkül elképzelhetetlen lenne.

Természetesen nem minden weboldal kell, hogy RIA legyen. De a weboldalak milliói használják űrlapok helyes kitöltésének ellenőrzéséhez, sütik kezeléséhez, a felhasználói élmény növeléséhez.

4.1. Alapok

Mi a JavaScript?

- Szkriptnyelv (tehát értelmezett, a böngésző értelmezi)
- Interaktivitást (működést) ad a weboldalhoz
- HTML forráskódba építhető, vagy attól elszeparálható a kód
- Események kezelésére alkalmazható
- A neve ellenére nincs szoros kapcsolatban a Java nyelvvel
- C++ és Java szintaxisra alapoz
- A szabvány leírását az ECMAScript specifikáció tartalmazza

4.1.1. Beillesztés a HTML kódba

A `script` elem segítségével tudunk a HTML oldalba szkripteket elhelyezni. Ennek `type` tulajdonságában kell megadni, hogy milyen szkriptnyelvet szeretnénk használni.

```HTML
<html>
  <body>
    <script type="text/javascript">
      document.write("Hello Világ!")
    </script>
  </body>
</html>
```

Az oldalt böngészőben megjelenítve a `Hello Világ!` szöveg jelenik meg a képernyőn.

Ha megnézzük a böngészőben az oldalunk forrását, akkor látható, hogy ott az eredeti, a webszerverről letöltött verzió látszik, és nem a JavaScript által „módosított”.

Jól látszik a példán, hogy JavaScriptben elhagyhatjuk a sorokat lezáró pontosvesszőket.

A `document.write` módszerét a mai korszerű megközelítésben nem illik alkalmazni.

Nagyon egyszerű kódok kivételével nem szokás a JavaScript kódot a body tagba tenni. Összetettebb működést inkább önálló függvények formájában szoktunk elhelyezni a `head` tagba, vagy még inkább külső JavaScript állományba. Nézzük meg ezek módjait:

```HTML
<head>
  <script type="text/javascript">
    function message() {
      alert("This alert box was called with the onload event")
    }
  </script>
</head>
<body onload="message()">
...
</body>
```

A példán látszik, hogy a `body` tag betöltődésekor (`onload` esemény) fog lefutni a `message` nevű függvény, ami egy figyelmeztető dialógusablakot dob fel. Ha a `message` függvény kódját egy `message.js` állományban helyeztük volna, akkor a `head` részbe a következő kód lenne szükséges:

```HTML
<script type="text/javascript" src="message.js"></script>
```

A `noscript` tag

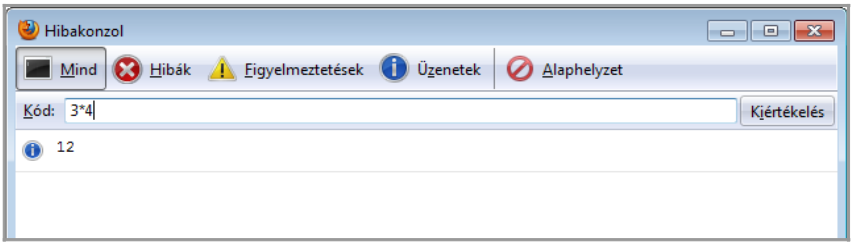
Ha a böngésző valamilyen ok miatt nem tudja végrehajtani a `script` elem tartalmát (pl. a JavaScript ki van kapcsolva), akkor a `noscript` elem tartalma fog megjelenni.

```HTML
<script type="text/javascript">
  document.write("A böngésző támogatja a JavaScriptet.")
</script>
<noscript>
  A JavaScript nem támogatott.
</noscript>
```

4.1.2. Hogyan kezdjünk neki?

A JavaScript tanulásához nincs feltétlenül szükség a webszerver használatára. Ezért a böngésző (pl. Firefox) és a programozói editor (pl. Komodo Edit) elegendő hozzá. Az elmentett HTML oldalt ugyanúgy megnyithatjuk a böngészőben, mint korábban a HTML és CSS tanulása során.

Érdemes azonban arra is gondolni, hogy JavaScript futása közben szintaktikai hibát is találhat a böngésző. Ilyen esetben jól jön, ha a Firefoxot használjuk. Itt ugyanis az *Eszközök* / *Hibakonzol* menüponttal (67. ábra) egy nagyon jól használható segítséget kapunk. A futási hibaüzeneteket is visszakereshetjük, de akár egyszerűbb kódot is lefuttathatunk a segítségével.



67. ábra. Firefox hibakonzol működése

Ezen kívül a Firebug kiegészítő *Konzol* füle még több szolgáltatást nyújt.

Természetesen más böngészők esetén is elérhetőek hasonló megoldások.

4.1.3. Esemény tulajdonságok

A böngésző programok lehetőséget adnak arra, hogy különböző (többnyire felhasználói) események esetén szkriptek segítségével programozható tevékenységet lehessen definiálni. Például egy HTML elemen való kattintás hatására az oldal egy része megváltozik.

Ablak események

Egyedül a body elem esetén használhatók.

- `onload`: akkor fut le a szkript, ha az oldal betöltődött
- `onunload`: akkor fut le a szkript, ha a felhasználó az oldalt bezárja, vagy más oldalra lép tovább

Űrlap elemek eseményei

Egyedül űrlap elemek esetén használhatók.

- `onchange`: akkor fut le a szkript, ha az elem tartalma megváltozott
- `onsubmit`: akkor fut le a szkript, ha az űrlap elküldését kéri a felhasználó
- `onreset`: akkor fut le a szkript, ha az űrlap kezdőállapotba áll
- `onselect`: akkor fut le a szkript, ha az elemet kiválasztja a felhasználó
- `onfocus`: akkor fut le a szkript, ha az elem megkapja a fókuszt
- `onblur`: akkor fut le a szkript, ha az elem elveszti a fókuszt

Billentyűzet események

- `onkeydown`: akkor fut le a szkript, ha egy billentyűt lenyomott a felhasználó
- `onkeyup`: akkor fut le a szkript, ha egy billentyűt felengedett a felhasználó
- `onkeypress`: akkor fut le a szkript, ha egy billentyűt lenyomott és felengedett a felhasználó (Folyamatos nyomva tartás esetén nem csak egy, hanem több esemény is bekövetkezhet, az operációs rendszer beállításainak függvényében.)

Egér események

- `onclick`: akkor fut le a szkript, ha egérkattintás történt
- `ondblclick`: akkor fut le a szkript, ha dupla egérkattintás történt
- `onmousedown`: akkor fut le a szkript, ha az egérgombot lenyomta a felhasználó
- `onmousemove`: akkor fut le a szkript, ha a felhasználó az adott elem felett mozgatja a kurzort
- `onmouseover`: akkor fut le a szkript, ha a felhasználó az adott elemre mozgatja a kurzort
- `onmouseout`: akkor fut le a szkript, ha a felhasználó az adott elemről elmozgatja a kurzort
- `onmouseup`: akkor fut le a szkript, ha az egérgombot felengedi a felhasználó

4.1.4. Dialógusablakok

Időnként szükségünk lehet egy honlapon arra, hogy a felhasználó figyelmét ráirányítsuk valamilyen fontos információra, vagy valamilyen reakciót várjunk egy dialógusablak segítségével. Ezek a dialógusablakok típusuktól függően más-más függvénnyel érhetők el.

Nézzük meg a fontosabb lehetőségeket példákon keresztül.

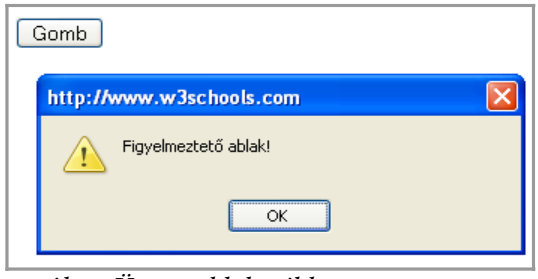
Üzenetablak

Az üzenetablak csupán egy egyszerű információt közöl, ezért az `alert` függvénynek is csak egy paramétert kell megadnunk. A felhasználó mindössze tudomásul veheti a közölt üzenetet.

JavaScript

```
<head>
  <script type="text/javascript">
  function disp_alert() {
    alert("Figyelmeztető ablak!")
  }
  </script>
</head>
<body>
  <form>
    <input type="button"
      onclick="disp_alert()"
      value="Gomb">
  </form>
</body>
```

Az eredmény:



68. ábra. Üzenetablak példa

Kérdés

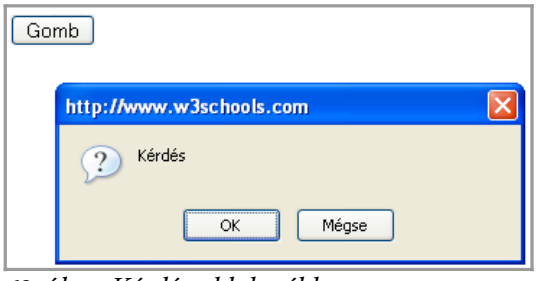
Bizonyos esetekben hasznos, ha a felhasználótól külön dialógusablakból kérünk választ egy eldöntendő kérdésre. Bizonyos esetekben ez praktikusabb, mint egy oldalon elhelyezett űrlap. Érdemes azonban arra is figyelni, hogy csak tényleg indokolt esetben alkalmazzunk üzenetablakokat!

A `confirm` függvény visszatérési értéke jelzi, hogy a felhasználó melyik gombot választotta.

JavaScript

```
<head>
  <script type="text/javascript">
    function disp_confirm() {
      if (confirm("Kérdés")) {
        document.write("OK")
      } else {
        document.write("Mégse")
      }
    }
  </script>
</head>
<body>
  <form>
    <input type="button"
      onclick="disp_confirm()"
      value="Gomb">
  </form>
</body>
```

Az eredmény:



69. ábra. Kérdés ablak példa

Egyszerű adatbevitel

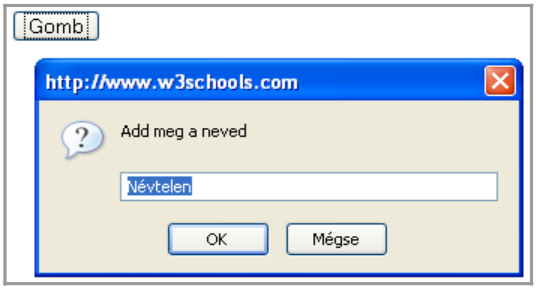
A `prompt` függvénnyel egy egysoros szöveg bevitelét kérhetjük a felhasználótól. A függvény első paramétere az üzenet szövege, a második pedig az alapértelmezett válasz (nem kötelező megadni). Itt is a függvény visszatérési értékén keresztül juthatunk a felhasználó válaszához. (Ha a felhasználó a `Mégse` gombot választotta, `null` a visszaadott érték.)

JavaScript

```
<head>
  <script type="text/javascript">
    function disp_prompt() {
      var name=prompt(
     "Add meg a neved","Névtelen")
      if (name!=null && name!=""){
        document.write(
            "Szia " + name + "!")
      }
    }
  </script>
</head>
```

```
<body>
  <form>
    <input type="button"
      onclick="disp_prompt()"
      value="Gomb">
  </form>
</body>
```

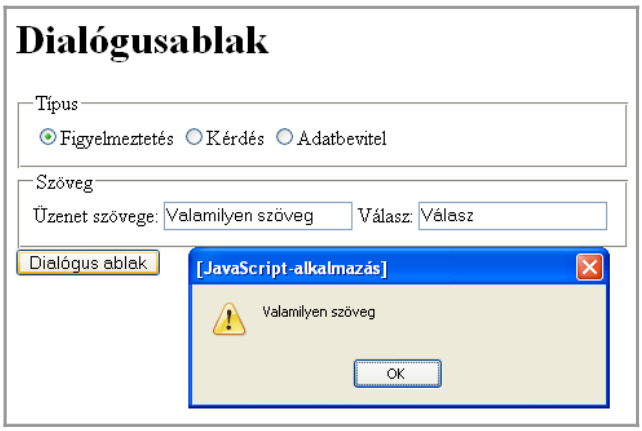
Az eredmény:



70. ábra. Egyszerű adatbevitel példa

Dialógusablakok dinamikus létrehozása

A következő példa a dialógusablakok tesztelését teszi lehetővé. Nézzük először a működését. Bemutatunk egy űrlapot, amelyik segítségével különböző dialógusablakok hozhatók létre, majd a válasz is értelmezhető lesz.



71. ábra. Dialógusablakok dinamikus létrehozása

A `Típus` segítségével a 3 -féle dialógus-típus közül választhatunk, az `Üzenet szövege` kerül a dialógusablakba, a `Válasz` mező pedig a függvény visszatérési értékét jeleníti majd meg. Nézzük az űrlap kódját:

```
<h1>Dialógusablak</h1>                                                    JavaScript
<form name="urlap" method="get" action="#">
  <fieldset>
    <legend>Típus</legend>
    <label><input checked="checked" type="radio" name="tipus"
        value="alert">Figyelmeztetés</label>
    <label><input type="radio" name="tipus" value="confirm">Kérdés</label>
    <label><input type="radio" name="tipus" value="prompt">
        Adatbevitel</label>
  </fieldset>
  <fieldset>
    <legend>Szöveg</legend>
    <label>Üzenet szövege: <input type="text" name="szoveg"
           value=""></label>
    <label>Válasz: <input type="text" name="valasz"
           value="Válasz"></label>
  </fieldset>
  <input onclick="uzenet();" type="button" name="gomb"
      value="Dialógus ablak" ></input>
</form>
```

Az érdemi JavaScript kód a head részben található uzenet függvényben lesz:

```
function uzenet() {                                                       JavaScript
  switch (true) {
    case document.urlap.tipus[0].checked:
      document.urlap.valasz.value =
          alert(document.urlap.szoveg.value)
      break
    case document.urlap.tipus[1].checked:
      document.urlap.valasz.value =
          confirm(document.urlap.szoveg.value)
      break
    case document.urlap.tipus[2].checked:
      document.urlap.valasz.value =
          prompt(document.urlap.szoveg.value)
      break
  }
}
```

Annak vizsgálására, hogy melyik rádiógomb is a kiválasztott, a rádiógomb checked tulajdonságának figyelését kell megoldanunk. Érdemes megfigyelni, hogy milyen logika szerint érjük el az egyes űrlapelemeket:

```
document.urlap.tipus[0].checked                                           JavaScript
```

A függvények válasza hasonlóan rugalmasan elhelyezhető a válasz mezőben:

```
document.urlap.valasz.value =                                             JavaScript
    alert(document.urlap.szoveg.value)
```

A 4.7.1. fejezetben még további magyarázatot láthatunk e kódok működésére.

4.2. Változók

A változók olyan tárolók, amelyekbe adatokat helyezhetünk. A változó értékét meg tudjuk változtatni a szkript futása során. A változóra a nevével tudunk hivatkozni, és az értékét bármikor lekérdezni vagy módosítani.

A változó nevek (mint ahogy az egész JavaScript nyelv) kis-nagybetű érzékenyek, és betűvel vagy aláhúzás karakterrel kezdődnek. A további karakterek között szám is lehet. Általában törekszünk a rövid, de kifejező nevek használatára, pl. `carname`.

4.2.1. Változó deklaráció

A `var` kulcsszó segítségével a következő módon tudunk változót létrehozni:

```JavaScript
var x;
var carname;
```

Ebben az esetben a változóknak még nincs értéke, vagyis a speciális `undefined` értéket veszi fel.

A deklarációt kezdőértékadással (inicializálással) együtt is írhatjuk:

```JavaScript
var x=5;
var carname="Volvo";
```

Változók élettartalma, láthatósága

Ha egy változót függvényen belül hozunk létre, a változó csak a függvényen belül érhető el. Ha kilépünk a függvényből, a változó megszűnik. Ezeket a változókat *lokális változók*nak nevezzük. Ha több különböző függvényben használjuk ugyanazt a változónevet, akkor azok egymástól független változók lesznek, mindegyik csak abban a függvényben látható, ahol létrehoztuk.

Ha egy változót a függvényeinken kívül deklarálunk, akkor az oldal minden függvényéből el tudjuk érni azt. A változó akkor jön létre, amikor deklaráljuk, és akkor szűnik meg, ha az oldalt bezárja a felhasználó. Az ilyen változókat *globális változóknak* nevezzük.

Végül még egy érdekesség: ha a változót a `var` kulcsszó nélkül deklaráljuk, akkor minden esetben globális lesz. Ezt azonban érdemes kerülni.

Konstansok

Több más nyelvhez hasonlóan a JavaScriptben is a `const` kulcsszóval van lehetőségünk arra, hogy egy változó értéke csak egyszer beállítható legyen.

```JavaScript
conts PI = 3.14;
```

4.2.2. Típusok

Egyszerű példák esetén nem túl sokat kell foglalkoznunk a típusok kérdésével, hiszen a JavaScript más szkriptnyelvekhez hasonlóan dinamikus típusrendszerű, vagyis a változó aktuális értéke határozza meg a típusát, ami bármikor megváltoztatható, pl. egy más típusú kifejezés értékadásával.

A változók típusa lehet objektum, primitív érték (`undefined`, `null`, `boolean`, `string`, `number`) vagy metódus (függvény objektum).

Nézzünk néhány példát:

- a `43` vagy `3.14` szám (`Number`) típusú
- a `true` és a `false` logikai (`Boolean`) típusú
- a `"Hello"` sztring (`String`) típusú
- a `null` és az `undefined` speciális esetekben szerepet kapó értékek

A sztringek tulajdonképpen objektumok, pl. lekérdezhetjük a sztring hosszát:

```
var s = "alma"
var h = s.length
```
JavaScript

Típuskonverzió és típus lekérdezés

A JavaScript nyelvben is szükségünk van időnként arra, hogy típuskonverziót hajtsunk végre. Egy gyakori eset, hogy valamilyen input (pl. a `prompt` függvény által visszaadott) szöveget kell számmá konvertálnunk. Erre a következő megoldást használhatjuk:

```
var szoveg = "23"
var szam = Number(szoveg)
```
JavaScript

Ha tudni szeretnénk egy változó típusát, akkor a `typeof` operátort használhatjuk:

```
document.write(typeof(szoveg)) // string
document.write(typeof(szam)) // number
```
JavaScript

Az operátor a következő sztringek valamelyikét adja vissza: `number`, `string`, `boolean`, `object`, `function`, `undefined`.

4.2.3. Literálok

Konkrét értékeket a következő módon tudunk JavaScriptben leírni.

Számok

Egész számokat többnyire 10-es számrendszerben adunk meg. Pl.

```
var a = 0
var b = 123
var c = -5000
```
JavaScript

Lebegőpontos számok esetén a tizedes pont nem hagyható el:

```
var pi = 3.14
var p = .2
```
JavaScript

Logikai típus csak kétféle értéket vehet fel:

```
var szemuveges = true
var kellfolytatni = false
```
JavaScript

Tömb

Tömb literálok a következő módon hozhatóak létre:

var szamok = [2, 3, 3, , 5]

A példából látszik, hogy a tömb elemei kihagyhatók, ezen értékük `undefined` lesz.

Szrting

Szringeket idézőjelekkel vagy aposztrófokkal tudunk létrehozni. Használhatóak a szokásos speciális karakterek is.

```
var s1 = 'alma'
var s2 = "korte"
var s3 = "c:\\temp"
```
JavaScript

Objektum

Objektum literálok kulcs-érték párok felsorolásával adhatóak meg:

```
var myRotator = {
  path: 'images/',
  speed:  4500,
  images: ["smile.gif", "grim.gif", "frown.gif", "bomb.gif"]
}
```
JavaScript

Ez így a C nyelv struktúráinak felel meg, komplexebb példákat később, a 4.6. fejezetben fogunk megnézni.

4.3. Kifejezések és operátorok

A kifejezések valamilyen értékel állítanak elő és/vagy valamilyen műveletet hajtanak végre. Nézzünk két egyszerű operátort, amelyeket gyakran használunk:

- Az = operátor (értékadó operátor) értéket rendel egy változóhoz.

- Az + operátor (összeadó operátor) összeadja a két operandus értékét.

A következő kód kiszámolja a 7-es értéket, és eltárolja az x változóban:

JavaScript
```
y=5;
z=2;
x=y+z;
```

4.3.1. Operátorok

Nézzük meg a fontosabb operátorokat. Ez nagyrészt csak ismétlés lesz a PHP operátorok ismeretében.

Aritmetikai operátorok

Az aritmetikai operátorok matematikai műveleteket hajtanak végre.

operátor	*név*	*példa*	*eredmény*
+	összeadás	x=2 y=2 x+y	4
-	kivonás	x=5 y=2 x-y	3
*	szorzás	x=5 y=4 x*y	20
/	osztás	15/5 5/2	3 2.5
%	maradék képzés	5%2 10%8 10%2	1 2 0
++	növelés 1-el	x=5 x++	x=6
--	csökkentés 1-el	x=5 x--	x=4

Értékadó operátorok

Az értékadó operátorok a bal oldali operandus értékét változtatják meg a jobb oldali kifejezés értékére (első sor), vagy módosítja a jobb oldali értékkel (további sorok).

Ha `x=10` és `y=5`, akkor:

operátor	*példa*	*eredmény*
`=`	`x=y`	`x=5`
`+=`	`x+=y`	`x=15`
`-=`	`x-=y`	`x=5`
`*=`	`x*=y`	`x=50`
`/=`	`x/=y`	`x=2`
`%=`	`x%=y`	`x=0`

Összehasonlító operátorok

Az összehasonlító operátorok logikai kifejezést állítanak elő.

operátor	*név*	*példa*	*eredmény*
`==`	egyenlő	`5==8`	`false`

`===`	egyenlő típus és érték	`X=5;`
		`y="5";`
		`x==y`
		`x===y`
			
			`true`
			`false`

`!=`	nem egyenlő	`5!=8`	`true`
`>`	nagyobb, mint	`5>8`	`false`
`<`	kisebb, mint	`5<8`	`true`
`>=`	nagyobb, vagy egyenlő, mint	`5>=8`	`false`
`<=`	kisebb, vagy egyenlő, mint	`5<=8`	`true`

Logikai operátorok

A logikai operátorok logikai értékekből újabb logikai értéket állítanak elő.

operátor	*név*	*példa*	*eredmény*
&&	és	x=6 y=3 (x < 10 && y > 1)	true
||	vagy	(x==5 || y==5)	false
!	nem	!(x==y)	true

Sztring operátor

Sztringek összefűzésére a + operátor egyszerűen alkalmazható:

```
txt1="Ez egy nagyon "
txt2="szép nap!"
txt3=txt1+txt2
```
JavaScript

Ekkor `txt3` értéke: `Ez egy nagyon szép nap!`

Sztringek és számok összeadása, kivonása

Szöveg és szám vegyes összeadása esetén (eltérően a PHP nyelvtől) a konverzió sztring irányban történik, és az eredmény is sztring lesz.

```
x=5+5;
document.write(x);
```
JavaScript

Az eredmény: `10`.

```
x="5"+"5";
document.write(x);
```
JavaScript

Az eredmény: `55` (sztring típusú).

```
x=5+"5";
document.write(x);
```
JavaScript

Az eredmény: `55` (sztring típusú).

```
x="5"+5;
document.write(x);
```
JavaScript

Az eredmény: `55` (sztring típusú).

Hogy még érdekesebb legyen, a kivonás esetén a PHP-nál megszokott konverzió történik:

```
x="5"-2;
document.write(x);
```
JavaScript

Az eredmény: `3` (szám típusú).

Feltételes operátor

A háromoperandusú feltételes operátor egyszerű esetekben egy `if-else` utasítást is kiválthat. Szintaxis:

```
változónév = (feltétel) ? kifejezéshaigaz : kifejezéshahamis
```
JavaScript

A `feltétel` teljesülése esetén `változónév` értéke `kifejezéshaigaz` értéke lesz, különben pedig `kifejezéshahamis` értéke.

A következő példa a `Jó reggelt` újabb verziója:

```
var d=new Date()
var time=d.getHours()
var message = "Jó " + ((time<9)?"reggelt":"napot") + "!"
document.write(message)
```
JavaScript

4.4. Vezérlési szerkezetek

Egy egyszerű lineáris kód csak nagyon egyszerű feladatokat tud megoldani. Legtöbbször szükségünk lesz különböző vezérlési szerkezetekre, hogy a kódunk megoldhassa a feladatát. A JavaScript nyelv alapvetően a megszokott vezérlési szerkezeteket nyújtja.

4.4.1. Elágazások

Kód írása közben nagyon gyakran alkalmazunk elágazásokat, hogy különböző esetek között különbségeket tudjunk tenni.

A JavaScript a C-hez és Javához hasonlóan többféle elágazást ismer.

`if` utasítás

Akkor használjuk, ha egy adott feltétel beteljesülése esetén szeretnénk valamit végrehajtani. Szintaxis:

```
if (feltétel) {
  utasítás
}
```
JavaScript

A következő kód `Jó reggelt` kíván, ha még nincs 9 óra.

```JavaScript
var d=new Date()
var time=d.getHours()
if (time<9) {
  document.write("<strong>Jó reggelt!</ strong >")
}
```

A következő példa az ebédidőre figyelmeztet.

```JavaScript
var d=new Date()
var time=d.getHours()
if (time==12) {
  document.write("<strong >Ebédidő!</ strong >")
}
```

`if-else` utasítás

Akkor van szükségünk erre az utasításra, ha a feltételünk teljesülése mellett a nem teljesülés esetén is kell valamilyen feladatot ellátni. Szintaxis:

```JavaScript
if (feltétel) {
  utasítás ha igaz
} else {
  utasítás ha hamis
}
```

Egészítsük ki a `Jó reggelt` példánkat:

```JavaScript
var d=new Date()
var time=d.getHours()
if (time<9) {
  document.write("<strong>Jó reggelt!</ strong >")
} else {
  document.write("Jó napot!")
}
```

`switch` utasítás

Ez az utasítás akkor alkalmazható nagyszerűen, ha egy adott kifejezés különböző értékei esetén más-más feladatot kell a kódnak végrehajtani. Szintaxis:

```JavaScript
switch(n) {
case 1:
  utasítás 1
  break
case 2:
  utasítás 2
  break
default:
  utasítás
}
```

Először az `n` kifejezés kiértékelése történik meg, majd a lehetséges `case` esetek között próbál a böngésző megegyezőt találni. Ha esetleg nincs, akkor a `default` címke fog aktiválódni. A `break` pedig azért szükséges, hogy a programunk ne tudjon a másik esetre rácsorogni.

A következő példa a hét napjától függően a munkahelyi hangulatot képes megjeleníteni.

```JavaScript
var d=new Date()
theDay=d.getDay()
switch (theDay) {
  case 5:
    document.write("Végre péntek!")
    break
  case 6:
  case 0:
    document.write("Jó kis hétvége!")
    break
  default:
    document.write("Mikor lesz hétvége?")
}
```

4.4.2. Ciklusok

A ciklusok lehetővé teszik, hogy egy kódot többször végrehajtsunk.

`for` ciklus

A `for` ciklust akkor szokás alkalmazni, ha a ciklus elkezdése előtt lehet tudni, hányszor kell majd a ciklusnak lefutnia. Jellemző szintaxis:

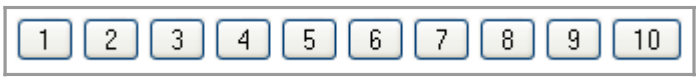
```JavaScript
for (változó=kezdőérték; változó <= végérték; változó++) {
  ciklusmag
}
```

Fontos szerepet tölt be a változó, amelyet ciklusváltozónak is szokás hívni. A ciklusmag lefutásának száma a ciklusváltozó kezdő és végértékétől, valamint a változó növelési módjától függ.

A következő példa egy számkitaláló játék alapja lehet.

```JavaScript
for (i = 1; i <= 10; i++) {
  document.write("<input type=button value=" + i +
                 " onclick=\"alert("+ i +")\"> ")
}
```

Az eredmény:



72. ábra. Dinamikus nyomógombok

`while` és `do-while` ciklusok

Ezeket a ciklusokat akkor szokás alkalmazni, ha nem tudjuk előre, hányszor kell a ciklusmagnak lefutnia. Sokszor ciklusváltozóra nincs is szükség, a ciklus futási feltétele valamilyen más módon áll össze.

A `while` ciklus magja addig fut, amíg a feltétel igaz. (Akár „végtelen” ciklus is lehet logikailag. A gyakorlatban a böngésző egy idő után leállítja a túl sokáig futó szkriptet.)

```
while (feltétel) {
  cikusmag
}
```
JavaScript

A `do-while` ciklus logikája csak abban más, hogy mindenképpen lefut egyszer a ciklusmag, és csak ez után értékelődik ki a feltétel:

```
do {
  cikusmag
} while (feltétel);
```
JavaScript

Vezérlésátadás

Nem kevés fejtörést okozott a programozás történetében a strukturálatlan nyelvek `goto` utasítása. Ezért a strukturált nyelvek szűk keretek között teszik csak lehetővé a vezérlésátadó utasítások használatát.

A C alapú nyelvekben a `break` és `continue` utasítások használhatók a ciklus teljes, vagy csak az aktuális ciklusmag futás megszakítására. (A `break` a `switch-case` szerkezetben már bemutatta kiugrási feladatát.)

A következő példa a korábbihoz hasonlóan gombokat hoz létre, de a 3-al osztható számokat kihagyja:

```
for (i = 1; i <= 10; i++) {
  if (i%3 == 0)
    continue;
  document.write("<input type=button value=" + i +
                 " onclick=\"alert("+ i +")\"> ")
}
```
JavaScript

Az eredmény:

73. ábra. Dinamikus nyomógombok

`for-in` ciklus

A `for` ciklust gyakran használjuk arra, hogy valamilyen tároló elem, például tömb minden elemével csináljunk valamit. Ilyen esetekben a szintaxist egyszerűsíteni tudja a `for-in` ciklus. Szintaxis:

```
for (változó in objektum) {
  ciklusmag
}
```
JavaScript

A ciklusmag annyiszor fog lefutni, amennyi az objektum elemeinek száma. Sőt a ciklusmagban a változó kifejezéssel közvetlenül hivatkozhatunk az éppen bejárt elemre.

A következő példa egy tömb elemeit írja ki egymás alá:

```
var mycars = new Array()
var x
mycars[0] = "Saab"
mycars[1] = "Volvo"
mycars[2] = "BMW"
for (x in mycars) {
  document.write(mycars[x] + "<br>")
}
```
JavaScript

4.5. Függvények

A függvények[103] az újra felhasználható kód egyszerű eszközei. Egy egyszer megírt függvényt a kód különböző helyein tetszőlegesen sokszor meghívhatjuk. (Egy jól megírt függvényt akár egymástól jelentősen eltérő weboldalakon is használhatunk.) Másrészt a függvények teszik lehetővé, hogy a HTML kódtól minél jobban elválasszuk a viselkedési réteget megvalósító JavaScriptet. (Hasonlóan, mint ahogy a megjelenítést meghatározó CSS-t is elkülönítjük.)

A függvények használatának egyik legegyszerűbb esete, amikor valamilyen esemény bekövetkeztekor egy eseménykezelő függvénynek adjuk át a vezérlést. A korábbi példák is jól mutatják ezt a megközelítést.

Érdemes azonban kitérni arra is, hogy a Javascript nyelv *meglepő újdonságokat tartalmaz* az objektum-orientált nyelvek ismerői számára is. Ez a jelentős eltérés talán itt, a függvények témájától kezdve mutatható ki leginkább. A Javascript függvényeit hozzárendelhet-

103 A fejezet készítésénél használt, és további információkért ajánlott *Galiba Péter: JavaScript függvények* című nagyszerű cikke. http://weblabor.hu/cikkek/javascript-fuggvenyek

jük változókhoz és objektum tulajdonságokhoz, átadhatjuk őket paraméterként más függvényeknek, valamint egy függvény – illetve metódus – is térhet vissza függvénnyel.

Függvénydeklaráció

A függvények létrehozására leginkább ismert megoldás a függvénydeklaráció.

A deklaráció szintaxisa:

```
function függvénynév(par1,par2,...,parX) {
  függvénytörzs
}
```
JavaScript

A `function` kulcsszó a PHP nyelvhez hasonlóan nem elhagyható. A kerek zárójelek között a függvények paramétereit adhatjuk meg. (Természetesen paraméter nélküli függvényeket is készíthetünk, de a zárójeleket ekkor is alkalmaznunk kell.)

A függvénydeklaráció a globális szinten, de akár egy másik függvény törzsében is szerepelhet. Ez utóbbira egy példa:

```
function kulso () {
  function belso () {
    // függvény törzs
  }
}
```
JavaScript

A `belso` függvény akkor jön létre, amikor a `kulso` függvény lefut.

Függvénykifejezés

A függvények létrehozásának másik lehetőségével is gyakran találkozhatunk. Bárhol használhatjuk, ahol változó szerepelhet. Ráadásul gyakran nem is adunk neki függvénynevet, mert nincs rá szükség. A következő név nélküli függvény akkor fog lefutni, ha az oldal teljes egészében betöltődik:

```
window.onload=function(){
..// kód ...
}
```
JavaScript

Visszatérési érték

Egyes függvények visszatérési értéket is előállíthatnak. A visszatérési érték a `return` utasítással adható meg. A kód a `return` hatására a függvényből kilépve a vezérlést is visszaadja az őt hívó kódnak.

A következő egyszerű függvény a két paramétere szorzatát adja vissza:

```
function prod(a,b) {
  return a*b
}
```
JavaScript

A függvényt például a következő módon tudjuk meghívni:

```
var product=prod(2,3)
```
JavaScript

A függvény 6-os visszatérési értéke bekerül a `product` változóba.

Változó számú paraméter

Vannak olyan esetek, amikor egy függvényt kevesebb vagy több paraméterrel meghívva is működőképes kódot szeretnénk készíteni. Erre az esetre a függvényen belül elérhető `arguments` objektum ad jó megoldást. A C nyelv main függvényéhez hasonló megoldás:

```
function foo() {
  var argv = foo.arguments;
  var argc = argv.length;
  for (var i = 0; i < argc; i++) {
    alert("Argument " + i + " = " + argv[i]);
   }
}
```
JavaScript

4.6. Objektumok

A JavaScript objektumorientált nyelv. Mielőtt azonban a részletekbe mennénk, érdemes megjegyezni, hogy a C++, Java, PHP stb. nyelvekkel szemben itt valódi objektumorientáltságról beszélhetünk, míg az előbb említett nyelveket talán pontosabb lenne osztály-orientált nyelveknek nevezni. A JavaScript nyelvet szokás még *objektum szemléletűnek* is nevezni.

4.6.1. Alapok

A böngésző JavaScript kód futtatásakor jó néhány objektumot bocsájt a rendelkezésünkre, amiken keresztül a weboldal egyes jellemzőit, részeit érhetjük el, vagy akár manipulálhatjuk is.

Kezdő JavaScript programozóként tehát nem sokban fog különbözni a dolgunk más nyelvi környezetekhez képet, komoly feladatok megoldásához már igen mély JavaScriptes OOP ismeretekre lesz szükségünk.

Az objektum különböző adatok gyűjteménye. Tulajdonságokkal és metódusokkal rendelkezik.

Tulajdonságok

A következő példa bemutatja, hogy hogyan lehet egy objektum tulajdonságát lekérdezni:

```
var txt="Hello World!"
document.write(txt.length)
```
JavaScript

A futás eredménye `12`.

Metódusok

A metódusok valamilyen tevékenységet hajtanak végre az objektumon. A következő példa egy sztring objektum `toUpperCase` metódusát használja a nagybetűs kiírás érdekében:

```
var str="Hello world!"
document.write(str.toUpperCase())
```
JavaScript

Az eredmény: `Hello world!`

4.6.2. Objektumok létrehozása

Egy objektum név-érték párok gyűjteménye, ahol a név a más nyelvekben megszokott adattag vagy tagfüggvény neve, az értéke pedig egy primitív érték, objektum, vagy akár egy függvény is lehet. Objektum példányt is *több módszerrel hozhatunk létre.*

Objektum inicializáló

Ez a lehetőség a C nyelv struktúrájához vagy a PHP nyelv asszociatív tömbjéhez hasonlít a leginkább.

```
var motor = {szin: "piros", kerek: 2}
```
JavaScript

Konstruktor függvény

A JavaScriptben nincsenek klasszikus értelemben vett osztályok. Az objektumokat közvetlenül hozzuk létre a `new` operátor segítségével. Minden objektum az `Object` típusra vezethető vissza, így egy üres objektumot pl. így hozhatunk létre:

```
var objektum = new Object();
```
JavaScript

Az objektumokat a konstruktor függvények példányosításával hozhatjuk létre. A következő kód egy egyetlen adattaggal rendelkező objektumok hoz létre.

```
function tipus(tul) {
  this.tulajdonsag = tul;
}
var objektum = new tipus(5); // objektum példányosítása
alert(objektum.tulajdonsag);
```
JavaScript

A PHP-ből is ismert `this` kifejezéssel hoztuk létre az objektum egyetlen adattagját.

A létrehozott `objektum` változó típusa `Object`.

Metódus létrehozása

A metódus az objektum egy olyan tagja, amely függvény típusú.

Metódust akár a konstruktorban, akár utólag is hozzárendelhetjük egy objektumhoz. Például egy névtelen függvény formájában:

```
motor.f = function() { /*...*/ }
```
JavaScript

4.6.3. Objektumként viselkedő változók

A JavaScript nyelv jó néhány beépített típust tesz elérhetővé. Ebben a fejezetben néhány fontosabbal ismerkedünk meg.

Tömbök

A JavaScript nyelvben a tömbök is objektumok. Ez több helyen is tetten érhető a következő példánkban. Hozzunk létre három tömböt:

```
var arrayOne = new Array("One","Two","Three","Four","Five");
var arrayTwo = new Array("ABC","DEF","GHI");
var arrayThree = new Array("John","Paul","George","Ringo");
```
JavaScript

A tömbök a PHP-hez hasonlóan szövegeket is elfogadnak indexként:

```
var anArray = new Array();
anArray[0] = "Fruit";
anArray["CostOfApple"] = 0.75;
```
JavaScript

A `new` operátor mellett az is árulkodik, hogy a tömböknek létezik `concat` metódusuk a tömbök összefűzésére, és `length` tagjuk a méretük lekérdezésére.

```
var joinedArray = arrayOne.concat(arrayTwo, arrayThree);
document.write("joinedArray has " + joinedArray.length + " elements<br>");
document.write(joinedArray[0] + "<br>")
document.write(joinedArray[11] + "<br>")
```
JavaScript

A tömb objektumoknak igen sok hasznos szolgáltatásuk van. Pl. egy tömböt rendezhetünk, majd megfordíthatjuk az elemeit:

```
var arrayToSort =
    new Array("Cabbage", "Lemon", "Apple","Pear","Banana");
var sortedArray = arrayToSort.sort();
var reverseArray = sortedArray.reverse();
```
JavaScript

A tömböket nem csak összefűzni, hanem szeletelni is tudjuk a `slice` metódussal:

```
var fullArray = new Array("One","Two","Three","Four","Five");
var sliceOfArray = fullArray.slice(1,4);
document.write(sliceOfArray[0] + "<br>");
```
JavaScript

Dátumok

A következő példánkban a beolvasott szöveget dátummá konvertáljuk a `Date` példányosításával, és így lehetőségünk lesz a dátumokkal való műveletekre is. Érdemes a `getDate` és a `setDate` függvény használatát megfigyelni.

```JavaScript
var input = prompt("Dárum (nap hónap év)", "31 Dec 2003");
var originalDate = new Date(input);
var addDays = Number(prompt("Eltelt napok","1"))
originalDate.setDate(originalDate.getDate() + addDays)
document.write(originalDate.toString())
```

Nem csak az egész dátumot, hanem egyes részeit is manipulálhatjuk:

```JavaScript
var someDate = new Date("31 Jan 2003 11:59");
document.write("Perc = " + someDate.getMinutes() + "<br>");
document.write("Év = " + someDate.getFullYear() + "<br>");
document.write("Hónap = " + someDate.getMonth() + "<br>");
document.write("Dátum = " + someDate.getDate() + "<br>");
```

Csak a percet változtatjuk:

```JavaScript
someDate.setMinutes(34);
document.write("Minutes = " + someDate.getMinutes() + "<br>");
```

Matematikai műveletek

A JavaScript nyelvben a `Math` objektum sokféle esetben lehet hasznos segítség.

Háromféle kerekítő algoritmus:

```JavaScript
var numberToRound = prompt("Please enter a number","")
document.write("round() = " + Math.round(numberToRound));
document.write("<br>");
document.write("floor() = " + Math.floor(numberToRound));
document.write("<br>");
document.write("ceil() = " + Math.ceil(numberToRound));
```

A következő példa egy kockadobást szimulál:

```JavaScript
var diceThrow = Math.round(Math.random() * 5) + 1;
document.write("A dobas: " + diceThrow);
```

4.7. A dokumentum elérése és módosítása

A JavaScript szóhasználatában a dokumentum elsősorban a HTML oldalt, illetve a HTML forráskódból felépített belső adatstruktúrát jelenti.

A böngészőmotor a HTML forráskód alapján felépít egy olyan adatstruktúrát, amely alapján a képernyőn vagy egyéb média eszközön megjeleníti az oldal tartalmát. Ez az adatstruktúra a DOM névre hallgat, és jelentős fejlődésen ment keresztül. A JavaScript korai

megközelítése a *dinamikus HTML* (röviden *DHTML*) kifejezéssel írta le azt, hogy ezt az adatszerkezet nem csak lekérdezhető, hanem manipulálható is. Nézzük először ezt a hagyományos megközelítést. (Egyes területeit ma is gyakran használjuk.) Majd utána térünk ki a DOM-ban rejlő mai lehetőségekre.

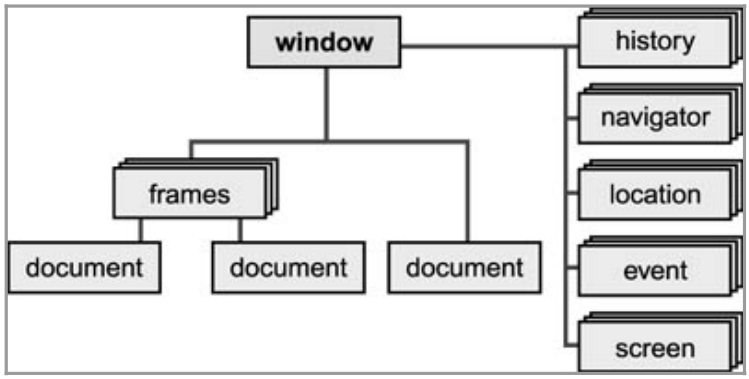
4.7.1. Dinamikus HTML

A böngészők a JavaScript kódunk számára speciális objektumokon keresztül teszik elérhetővé a dokumentum egyes részeinek elérését, manipulálását. Ennek egy nagyon egyszerű példájával már többször találkoztuk. A `document` objektum a HTML forráskódból felépített adatszerkezetet jelenti. Így annak `write` metódusa be fogja szúrni a dokumentum megadott pontjára a paramétereként kapott szöveget:

```
<script type="text/javascript">
  document.write(Date());
</script>
```
JavaScript

Ha nem szöveget adunk át a metódusnak, akkor szöveggé lesz konvertálva.

A következő objektumhierarchia néhány elemével fogjuk folytatni:



74. ábra. JavaScript objektumhierarchia

`Window` objektum

A `Window` objektum az aktuális böngészőablakot (vagy böngészőfület) reprezentálja. Ezzel tudunk pl. új felugró ablakot nyitni:

```
window.open ("http://oldalam.hu","oldalam");
window.open ("http://oldalam.hu","oldalam","status=1,toolbar=1");
```
JavaScript

Ezt ma ritkán használjuk, mert a böngészők, vagy kiegészítőik általában blokkolják a kéretlen reklámok miatt.

`Document` objektum

A `document` objektum a HTML `body` tagjának tartalmát reprezentálja.

- A `title` tulajdonság az oldal címét tartalmazza. Ezt akár meg is változtathatjuk JavaScript kóddal.
- A `lastModified` változó a dokumentum utolsó módosításának időpontját tartalmazza. Természetesen dinamikusan (pl. PHP-vel) generált oldal esetén a generálás időpontja fog szerepelni.
- A `location` csak olvasható mező tartalmazza a dokumentum teljes URL-jét.
- A `referrer` tulajdonság szintén csak olvasható mező. Annak az oldalnak az URL-jét tartalmazza, amelyről a látogató egy linket követve érkezett. Ha nem linken keresztül érkezett, akkor üres a mező.
- A `forms` és a `links` tömbök a dokumentum űrlap és hivatkozás objektumait tartalmazzák.
- A `name` attribútummal rendelkező elemeket közvetlenül is elérhetjük a tartalmazási hierarchiának megfelelően.

Nézzünk egy példát az utolsóra. A 71. ábra forráskódjának egy részlete jól mutatja, hogy `urlap` a `form` `name` tulajdonsága, ami tartalmaz `tipus` nevű rádiógombokat, valamint `valasz` és `szoveg` nevű `input` mezőket.

JavaScript
```
switch (true) {
  case document.urlap.tipus[0].checked:
    document.urlap.valasz.value =
        alert(document.urlap.szoveg.value)
```

`History` objektum

A `history` objektum az előzőleg meglátogatott oldalakat tartalmazza. A következő példa a sokak által szeretett, de nem a legszerencsésebb `Vissza` linket hozza létre:

JavaScript
```
<a href="javascript:history.go(-1)">Vissza</a>
```

`Navigator` objektum

A `navigator` objektum a böngészőnkről tud információkat kinyerni. Pl. a `navigator.appName` a böngésző nevét, a `navigator.cookieEnabled` a sütik használhatóságát tartalmazza. A böngésző pontos típusának meghatározására is alkalmazható, bár ez mai szemmel elég trükkös módon[104].

104 Egy érdekes példát láthatunk *Peter-Paul Koch: Browser detect* című cikkében.
Forrás: http://www.quirksmode.org/js/detect.html

Ahogy említettük, az eddigi technikákat ma már ritkábban használjuk, inkább a következő megközelítés alkalmazandó.

4.7.2. DOM

A *Document Object Model* egy olyan szabvány, amelynek segítségével a böngésző a HTML dokumentum struktúráját faszerkezetben ábrázolja és teszi elérhetővé, manipulálhatóvá a JavaScript számára.

A DOM három részre osztható: Core DOM, XML DOM és HTML DOM. Ezen kívül megkülönböztetünk Level 1/2/3 DOM-okat. Mi azonban csak egy nagyon rövid bevezetéssel fogunk a témára pillantani.

A fastruktúra minden pontja egy objektum, ami egy elemet reprezentál. Az elemek tartalmazhatnak szöveget és tulajdonságokat is, amit az objektumokból ki is lehet nyerni.

A következő példa az oldalon való kattintás hatására sárga hátteret állít be:

```
<head>                                                    JavaScript
  <script type="text/javascript">
    function ChangeColor() {
      document.body.bgColor="yellow"
    }
  </script>
</head>
<body onclick="ChangeColor()">
  Klikkeljen az oldalon!
</body>
```

A `document` objektum tartalmazza az összes további objektumot, amelyek együttesen a HTML dokumentumot reprezentálják. A `document.body` objektum a body tagnak felel meg.

Az objektumok tulajdonságai egyszerűen elérhetők, illetve manipulálhatók. Az előző példában látott `document.body.bgColor` az oldal (`body`) háttérszínét képviseli.

A következő példa eredményében ugyanazt nyújtja, mint az előző, azonban a megközelítése más. A `style` tulajdonság a CSS-ben beállítható tulajdonságokat foglalja össze:

```
document.body.style.background="yellow"                   JavaScript
```

`getElementById`

Az egyik leggyakrabban alkalmazott metódus, hiszen ennek segítségével lehet a HTML dokumentum egy adott elemét elérni, azon például valamilyen manipulációt elérni. A következő példából fontos megfigyelni, hogy csak az id-vel rendelkező elemeket érhetjük el a segítségével. A következő példa egy link elemet keres meg, majd több jellemzőjét megváltoztatja:

```
<a id="micro" href="http://www.microsoft.com"> Microsoft</a>   HTML
```

A JavaScript kód:

```
var link = document.getElementById('micro')
link.innerHTML="Visit W3Schools"
link.href="http://www.w3schools.com"
link.target="_blank"
```
JavaScript

A `getElementById` metódussal megkapott objektum a HTML forráskód alapján épült fel, amiből lekérdezhetünk információkat, vagy akár módosíthatjuk is a mezőit.

Az `innerHTML` mező segítségével a kezdő és záró tag között szereplő teljes HTML kódot lekérdezhetjük vagy módosíthatjuk. Bár hasznos lehetőség, csak egyszerűbb esetekben, főleg tiszta szöveg cserénél érdemes használni.

További érdekes szolgáltatások: például egy elem megkaphatja a fókuszt a `focus` metódus meghívásával, vagy éppen elvesztheti a `blur` metódus hatására. A `focus` praktikus lehet, ha a JavaScriptet űrlap adatok ellenőrzésére használjuk. Ekkor a hibaüzenet megjelenítése után vissza lehet küldeni a kurzort, a hibát tartalmazó beviteli mezőre, így egy kattintást megspórol a felhasználó.

Talán feleslegesnek tűnik az ilyen aprólékos kódolás, de sokszor az ilyen apró praktikus segítségek miatt érzi azt a felhasználó, hogy ez egy barátságos oldal.

Végül nézzünk meg egy kicsit komplexebb DOM példát[105], amely az `innerHTML` helyett korszerűbb megközelítés a dokumentum dinamikus megváltoztatására. A következő kód betűméret választó „gombokat” (`div` elemet, és abban `a` elemeket) hoz létre.

```
function fontresizeShow() {
  var div = document.createElement("div");
  div.id = "fontresize";
  var aSmall = document.createElement("a");
  var aNormal = document.createElement("a");
  var aLarge = document.createElement("a");
  aSmall.onclick =
      function() { StyleActivate('small'); return(false); };
  aSmall.href = "#";
  aSmall.className = "small";
  aSmall.appendChild(document.createTextNode('A'));
  aNormal.onclick =
      function() { StyleActivate('normal'); return(false); };
  aNormal.href = "#";
  aNormal.className = "normal";
  aNormal.appendChild(document.createTextNode('A'));
  aLarge.onclick =
      function() { StyleActivate('large'); return(false); };
  aLarge.href = "#";
  aLarge.className = "large";
  aLarge.appendChild(document.createTextNode('A'));
  div.appendChild( aSmall );
  div.appendChild( aNormal );
  div.appendChild( aLarge );
  document.body.appendChild( div );
}
```
JavaScript

105 Bártházi András: *Dinamikus betűméret választó* című cikke nagyszerű írás a témában.
Forrás: http://weblabor.hu/cikkek/betumeretvalaszto

4.7.3. DOM megoldások

Ebben a fejezetben néhány olyan függvényt fogunk megismerni, amely a DOM korszerű használatát mutatja be. Ezen kódok bármely weboldalon felhasználhatók – általában – változtatás nélkül.

`getElementsByClass`

A `getElementById` DOM függvény mintájára hasznos lenne, ha egy adott osztályú elemeket könnyen ki tudnánk gyűjteni, akár elemtípusokra (pl. `p` vagy `h3`) szűkítve is. A következő egyszerű verzió csak a keresett osztály nevét várja. Lehetne szűkíteni a keresést, például csak egy típusú elemekre, vagy adott azonosítójú elem leszármazottaiban, stb.

```
function getElementsByClass(name, element = "*") {
  var found = 0;
  var elems = new Array();
```
JavaScript

Találatok megszámolásához és tárolásához két segédváltozó.

```
  var alltags = document.getElementsByTagName(element);
```
JavaScript

Ha a második paramétert is megadjuk a híváskor, az alapértelmezett `*` helyett csak bizonyos elemekben keres. A visszaadott érték lehet `null` is, ezért ezt ki kell szűrni:

```
  if (alltags) {
    for (i=0; i < alltags.length; i++) {
        if (alltags[i].className==name) {
          elems[found++]=alltags[i];
        }
    }
  }
  return(elems);
}
```
JavaScript

`addEvent`

Ez a függvény egy adott böngésző objektum egy adott eseményéhez csatol hozzá egy adott függvényt, melyeket paraméteréül kap.

```
function addEvent(obj, evType, fn) {
  if (obj.addEventListener) {
    obj.addEventListener(evType, fn, true);
    return true;
  } else if (obj.attachEvent) {
    var r = obj.attachEvent("on"+evType, fn);
    return r;
  } else {
    return false;
  }
}
```
JavaScript

Bár a megoldás böngészőfüggő, azért könnyen átlátható. (A DOM az `addEventListener`-t tartalmazza, szokás szerint az Internet Explorer jár külön utakon.)

addLoadEvent

Ez a metódus az előző probléma speciális esetének is tekinthető. Az előző ugyanis a body tagra nem minden böngésző alatt működik, míg a következő megoldás igen.

```JavaScript
function addLoadEvent(func) {
  var oldonload = window.onload;
  if (typeof window.onload != 'function') {
    window.onload = func;
  }
```

Először is megnézzük, hogy van-e eseménykezelő rendelve az onload-hoz. Ha nem, egyszerű a dolgunk.

```JavaScript
  else {
    window.onload = function() {
      oldonload();
      func();
    }
  }
}
```

Ellenkező esetben egy új függvénybe kell összefoglalni az eddigi és az új kódot, hogy az új függvényt eseménykezelővé téve mindkettőt futtassa.

4.7.4. Diszkrét JavaScript

A JavaScript segítségével sok hasznos funkcionalitást adhatunk hozzá oldalainkhoz. Vannak azonban olyan szempontok is, amelyek az okos használatra ösztönöznek.

Ha a JavaScript fut a fejlesztő oldalán, akkor hajlamos azt feltételezni, hogy mindenki másnál is futni fog, pedig ez nincs így. Sok felhasználó fél a biztonsági kockázatoktól, a mobin böngészője nem támogatja, vagy valami más ok miatt kapcsolja ki a JavaScriptet. Ezen kívül a kereső robotok sem értelmezik a JavaScript kódokat, így például a JavaScripttel megoldott navigációt a Google nem fogja figyelembe venni, vagyis az oldal nem lesz kereshető a Google-lel.

A *Diszkrét JavaScript* kifejezés azt az alapelvet jelenti, hogy a csak JavaScripttel használható (tolakodó) szolgáltatások helyett olyan oldalakat készítsünk, amelyek JavaScript nélkül is működnek, esetleg kényelmetlenebbül, extrák nélkül, de mégis működnek. A JavaScript csak a plusz kényelmi szolgáltatásokat adja.

Előugró ablak példa

A téma rövid bevezetéseként nézzünk meg egy feladat többféle megoldását. A feladat egy előugró (popup) ablak nyitása lesz.

Az első megoldás kézenfekvőnek tűnek, de az előző megfontolások miatt nem célravezető:

```
<a href="javascript:window.open('popup.html','popup');">
  popup nyitás</a>
```
JavaScript

A következő megoldás szintén elérhetetlen JavaScript nélkül:

```
<a href="#"
  onclick="window.open('popup.html','popup');return(false);">
  popup nyitás</a>
```
JavaScript

A link a dokumentum elejére mutat (#). JavaScript használata esetén az előugró ablak megnyílik (`window.open`), majd a `return(false)` miatt a böngésző nem fogja a tényleges link célt (#) figyelembe venni. JavaScript nélkül azonban az oldal elejére ugrik a böngésző.

Egy jó megoldás lehet a következő:

```
<a href="popup.html"
  onclick="window.open('popup.html','popup');return(false);">
  popup nyitás</a>
```
JavaScript

Ekkor a JavaScript gondoskodik az előugró ablak megnyitásáról, és `false` visszatérési értékkel megakadályozza, hogy a főoldal a `popup.html`-re ugorjon. JavaScript nélkül pedig a főablak jeleníti meg a `popup.html`-t.

Még diszkrétebb

A diszkrét JavaScript elvével még ennél is tovább mehetünk. A JavaScript kód HTML oldalba keverése ugyanúgy nem praktikus, mint ahogy a HTML tartalom és a megjelenítés mikéntjét definiáló CSS összekeverése sem az. Célszerű tehát a JavaScriptet a lehető legjobban elkülöníteni a HTML dokumentumtól.

Ennek az elvnek megnyilvánulása az a célunk is, hogy a JavaScript kódunk jelentős része függvényekben és osztályokban, azok pedig külön .js állományokban legyenek.

A következő verzió ugyan feleslegesen bonyolultnak tűnhet, de nem szabad elfelejteni, hogy a megoldás gerincét alkotó elveket és függvényeket már mások kidolgozták, nekünk elég felhasználni az ő munkájukat, és hátradőlni a karosszékünkben. :)

A HTML kódon nem is látszik, hogy itt JavaScriptről lenne szó:

```
<a href="popup.html" class="popup">popup nyitás</a>
```
HTML

Az egész HTML kódban csak a `.js` állomány betöltése mutatja, hogy itt JavaScript kód fog futni.

A feladatot megoldó JavaScript állomány tartalmaz több függvényt, és a következő sort:

```
addEvent(window, 'load', classPopupHandler);
```
JavaScript

Ennek a sornak az a feladata, hogy a `window` objektum `load` eseményéhez hozzákapcsoljuk a `classPopupHandler` metódust. (Ennek kevésbé szép megoldása lenne a `body onload="classPopupHandler"` a HTML kódban).

Az `addEvent` metódus és egy segédmetódusa a fejezet későbbi részében kerülnek bemutatásra, itt csak az eseménykezelő függvényt minden egyes `popup` osztályú linkhez kapcsoló `classPopupHandler` metódus és a tényleges eseménykezelő doPopup metódus következik.

JavaScript
```
function classPopupHandler() {
  var elems=getElementsByClass('popup');
  for(i=0;i<elems.length;i++) {
    if (elems[i].href && elems[i].href!='') {
      addEvent(elems[i],'click',doPopup);
    }
  }
}
```

Először is a (korábban bemutatott) `getElementsByClass` metódus egy tömbbe kigyűjti a `popup` osztályú elemeket. A ciklusban a megadott `href` értékkel bíró tagokat szűrjük, és hozzárendeljük a `click` eseményhez a `doPopup` metódust.

A kevésbé diszkrét megoldás ugye pont azt jelentené, hogy a HTML kód tartalmazza minden helyen az `onc-lick="..."` szöveget.

Következzék a tényleges eseménykezelő, ami a kattintáskor fog lefutni:

JavaScript
```
function doPopup(ev) {
  ev || (ev = window.event);
```

Az `ev` esemény objektum elvileg a függvény paramétereként áll rendelkezésre, de Internet Explorer alatt ezt nem kapjuk meg, ezért a `window` objektumból kell kinyerni.

JavaScript
```
  var source;
  if (typeof ev.target != 'undefined') {
    source = ev.target;
  } else if (typeof ev.srcElement != 'undefined') {
    source = ev.srcElement;
  } else { return(true); }
```

Megpróbáljuk kideríteni, hogy melyik objektum váltotta ki az eseményt. A többféle próbálkozás itt is a különböző böngészők miatt szükséges.

JavaScript
```
  window.open(source.href,'popup');
```

A tényleges feladat, az ablak megnyitása.

JavaScript
```
  if (ev.preventDefault) {
    ev.preventDefault(); ev.stopPropagation();
   } else {
    ev.cancelBubble = true; ev.returnValue = false;
  }
  return false;
}
```

Végül arról is gondoskodnunk kell, hogy az eseményt még egyszer ne kezelje le a böngésző, vagyis a főablak maradjon eredeti állapotában.

E-mail cím elrejtése

Nagyon felelőtlen dolog egy e-mail címet kitenni egy weboldalra. Az ún. spam robotok pont azt keresik, hogy a weben hol vannak kint levő e-mail címek, és a találatokat egy központi szerverre gyűjtik. Az ilyen e-mail gyűjteményeket aztán eladják, és jön a megszámlálhatatlanul sok kéretlen levél. Webfejlesztőként gondoskodnunk kell tehát az illetéktelen hozzáférés megakadályozásáról.

Korábban elterjedt megoldás volt a képek alkalmazása: a képszerkesztővel elkészített e-mail címet a látogató el tudja olvasni, de a spam robot nem (hacsak nem foglalkozik karakterfelismeréssel). Másik elterjedt megoldás a szöveges kódolás, ahol a pont és kukac írásjelek szövegesen jelennek meg. Itt a biztonság ugyan megvan, de kényelmetlen egy ilyen esetben begépelni az e-mail címet, fennáll a tévesztés lehetősége. Ebben a környezetben jön jól a következő megoldás, ami szintén diszkrét JavaScriptre épít.

A megoldás lénye, hogy az e-mail címet egy meghatározott algoritmus szerint kódoljuk, amit a böngészőben futó JavaScript visszaalakít eredeti formájára. Arra is gondolva, hogy a látogató nem biztos, hogy engedélyezi a JavaScript alkalmazását, a kódolt címnek az emberi felhasználó számára értelmezhetőnek kell lenni. A HTML kód tehát lehet például:

```HTML
<a href="mailto:nagy.gusztav#KUKAC#gamf.kefo.hu">
    nagy.gusztav#KUKAC#gamf.kefo.hu</a>
```

Ha a böngészőben fut a JavaScript, akkor a `#KUKAC#` karaktersorozat minden előfordulását visszacseréljük a `@` karakterre.

```JavaScript
function whatCorrector() {
  var replaces = 0;
  while (document.body.innerHTML.indexOf('#'+'KUKAC#')!=-1
         && replaces<10000) {
```

Az `innerHTML` hozzáfér a teljes body szöveghez. Az `indexOf` a cserélendő karaktersorozatot kutatja. Egy biztonsági intézkedés az esetleg valami probléma miatt fennálló végtelen ciklus elkerülése érdekében, hogy legfeljebb 10.000 futtatás engedélyezett.

```JavaScript
    document.body.innerHTML =
      document.body.innerHTML.replace(
        '#'+'KUKAC#',String.fromCharCode(64));
```

A `replace` a tényleges cserét végzi, bár egy további trükk, hogy a `@` karakter nem szerepel a kódban, hanem a `String.fromCharCode(64)` segítségével állítjuk elő.

```JavaScript
    replaces++;
  }
}
```

Végül a csere elindítása:

```JavaScript
addLoadEvent(whatCorrector);
```

4.8. Eseménykezelés

JavaScript segítségével készíthetünk dinamikus weblapokat. Alapvetően ez azt jelenti, hogy a honlap különböző felhasználói (és böngésző) eseményeket képes érzékelni, és azokra valamilyen módon reagálni.

Jellemző események a következők:

- egérkattintás
- a weboldal betöltődött, és a feldolgozása megtörtént
- az egeret érzékeny területen mozgatjuk
- listából egy elem kiválasztásra kerül
- űrlap elküldés
- billentyűleütés

Az egyes események bekövetkezése esetén egy függvénnyel szokás a szükséges funkciót végrehajtani.

Az `Event` objektum

Egérrel vagy billentyűvel kiváltott felhasználói események kezelésekor az eseménykezelő függvény egy esemény objektumot kap paraméterként. Ebből az objektumból lehet kinyerni például, hogy melyik egérgomb volt lenyomva, vagy éppen milyen koordinátákon van az egérkurzor.

4.8.1. `onload` és `onUnload`

Ezek az események akkor következnek be, ha az oldal betöltése befejeződött, illetve mikor a felhasználó az oldal elhagyását kezdeményezi.

Nézzünk néhány hasznos példát (haszontalant, feleslegeset sajnos sokat találhatunk a weben):

- A sütikben tárolt felhasználói beállításokat (pl. a felhasználó által preferált betűméret) szeretnénk betölteni és elmenteni.
- Oldal elhagyása előtt ellenőrizhetjük, hogy az űrlapban történt-e változás. (Ha van begépelt adat, és véletlen kattintunk egy linkre, elveszhet a begépelt szövegünk.) Ha van változás, csak kérdés után engedjük el az oldalról a felhasználót.

- Oldal betöltődése után (onload) a spammer robotok miatt elkódolt e-mail címeket visszakódolhatjuk. (Ha saját, egyedi elkódolást alkalmazunk, a spammer robot programok nem fogják azt ismerni, tehát a kitett e-mail címet nem tudják ellopni.)

4.8.2. onFocus, onBlur és onChange

Ezek az események elsősorban űrlapok ellenőrzésére, vagy az űrlapok használatának kényelmesebbé tételére szolgálnak.

Az `onFocus` akkor következik be, ha az elem megkapja a fókuszt (például kattintás vagy a tab billentyű hatására).

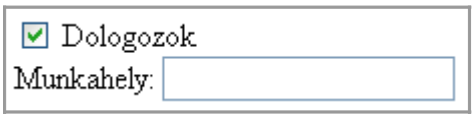
Az `onBlur` az elem elhagyásakor, az `onChange` pedig bármilyen, a bevitt adatban történő változás esetén következik be. Nézzünk egy példát az e-mail cím helyes szintaxisának ellenőrzésére:

```
<input type="text" size="30" id="email"
  onchange="checkEmail()">
```
JavaScript

Összetettebb logikájú űrlapokon hasznos lehet, ha az összefüggéseket vizuálisan is jelezzük a felhasználónak. Például ha a felhasználó egy jelölőnégyzeten kattintva jelzi, hogy van munkája, adjunk lehetőséget a munkahely mező kitöltésére. Egy ehhez hasonló esetben megoldás lehet a következő példa:

```
<input type="checkbox" onchange=
  "document.getElementById('info').style.visibility =
   this.checked ? 'visible' : 'hidden'"> Dologozok
<br>
<div id="info" style="visibility:hidden">
  Munkahely: <input name="munkahely" type="text">
</div>
```
JavaScript

Az eredmény:



75. ábra. onchange esemény példa

4.8.3. onSubmit

Űrlap elküldése előtt az összes ellenőrzést el lehet végezni ennek az eseménykezelőnek a segítségével. Az eseménykezelő logikai értékével jelezhetjük a böngészőnek, hogy az űrlap elküldhető-e:

```
<form method="post" action="check.php"
  onsubmit="return checkForm()">
```
JavaScript

A `checkForm` függvény visszatérési értékét az űrlap adatai érvényességének függvényében kell beállítanunk.

4.8.4. `onClick`, `onMouseDown` és `onMouseUp`

Az `onClick` esemény akkor következik be, ha a felhasználó kattint az elemen. Csak akkor érdemes ezt az eseményt kezelni, ha tényleg a kattintáshoz kell kötni egy feladatot, például egy Tili-toli játéknál a mozgatandó kép kijelölésére, vagy egy fényképalbum esetén a mozaikképre kattintva egyből megjelenjen a nagy méretű kép a nézőke dobozban. Az utóbbi példa vázlata:

JavaScript
```
<img src="sbergamot.jpg" alt=" "
  onmousedown="document.images['large'].src='bergamot.jpg'">
<img src="sfoxglove.jpg" alt=" "
  onmousedown="document.images['large'].src='foxglove.jpg'">
<img src="spansy.jpg" alt=" "
  onmousedown="document.images['large'].src='pansy.jpg'">
<br>
<img src="space.jpg" alt=" " id="large">
```

Még speciálisabban alkalmazható az `onMouseDown` és `onMouseUp` esemény kezelése, ami az egérgomb lenyomásakor és felengedésekor következik be. Erre ma tipikus példa a vonszolás esete, amikor az egérgombbal történő megragadáskor és elengedéskor kell valamit tenni.

Korábban sok helyen alkalmazott felesleges és rossz gyakorlat, hogy a HTML link (`a` tag) helyett is valamelyik eseményt alkalmazzák.

Szintén elavult megoldás az oldalunk menüpontjait képekkel megvalósítani és a képeket az `onMouseDown` és `onMouseUp` események esetén cserélgetni. (CSS-el is megoldható a csere, másrészt az aktivitást jelző kép lassú kapcsolat esetén nem jelenik meg azonnal, ami nagyon zavaró.)

4.9. Felhasználói élmény

A felhasználói élmény elérése a JavaScript korszerű használatának egyik legfontosabb oka. E fejezetben néhány egyszerűbb példán keresztül megvizsgáljuk, milyen módon lehet a felhasználó munkáját könnyebbé, akár élményszerűvé tenni.

4.9.1. Kliens oldali űrlap ellenőrzés

Felhasználóként rendkívül kiábrándító tud lenni, ha egy komolyabb űrlap kitörlése értelmetlenül felesleges munkát okoz.

Egy űrlap áttekinthető megtervezése, a magyarázó szövegek pontos megfogalmazása gyakran nehéz feladat. De még jól megtervezett űrlapok esetén is előfordulhat, hogy a felhasználó hibát követ el, egy hosszabb űrlap esetén akár többet is.

Rendkívül *rontja a felhasználói élményt*, ha a hibákat csak hosszas procedúra után lehet kijavítani: pl. az elküldött adatokban levő első hibát a webalkalmazás hibaüzenettel jelzi, és így a felhasználó akár jó néhányszor kénytelen az adatokat javítani-újraküldeni, hogy egyáltalán a hibaüzenetekkel szembesülhessen. Ez a látogató számára nagyon kényelmetlen.

Egy űrlap esetén az a minimum, hogy az összes hibaüzenetet egyszerre (és a beviteli elem környékén) mutatjuk meg, de még jobb a megoldásunk, ha az adatok elküldése előtt jelezzük a hibákat. A jelzésre egy praktikus lehetőség, ha a beviteli elem környékén pl. egy pipa megjelenésével, vagy éppen a háttérszín vagy a szegély más színre állításával jelezzük az adott elem helyes kitöltését.

Az itt következő példa a szerző honlapjának egy 2006-os változatán a hozzászólás beküldésére szolgáló űrlap lesz. Először is érdemes megfigyelni, hogy a felhasználóval nem egy kitalálós játékot játszunk, hanem a lehető legrészletesebb magyarázatot adjuk a beviteli mezőkhöz:

76. ábra. Új hozzászólás űrlap

Az ábrán talán nem egyértelműen látszik, hogy a `Küldés` gomb induló állapotában nem használható, szürke színű. Csak akkor válik használhatóvá, ha a felhasználó minden szükséges mezőt korrekten kitöltött.

HTML kód

HTML

```
<h2>Új hozzászólás</h2>
<form id="hozzaszolas" action="index.php" method="post">
  <label for="szoveg">(kötelező, minimum 10 betű):</label>
  <textarea id="szoveg" name="szoveg"></textarea>
  <label for="nev"><strong>Név</strong>
    (kötelező, minimum 3 betű):</label>
  <input id="nev" name="nev" type="text" size="20" value="">
  <label for="email"><strong>E-mail cím</strong>:</label>
  <p>(Az e-mail cím megadása nem kötelező. Megadása esetén
     megjelenik az oldalon, de nem lesz kiszolgáltatva a spam
     robotok számára.</p>
  <input id="email" name="email" type="text" size="20"
    value="">
  <label for="human">Humán ellenőrzés.
    <strong>Negyvenhét</strong> számmal:</label>
  <input id="human" name="human" type="text" size="5"
    value="">
  <button name="hozzaszolaskuld" id="hozzaszolaskuld"
    type="submit" value="kuld">Küldés</button>
</form>
```

Érdemes megfigyelni, hogy a JavaScript kód diszkrét, nem is látszik közvetlenül a HTML kódban. Ha a JavaScript nincs engedélyezve, akkor a kényelmi szolgáltatás nélkül ugyan, de az űrlap elküldhető.

A diszkrét működés érdekében a gomb alapértelmezetten lenyomható, a következő JavaScript kód teszi használhatatlanná (`disabled`):

JavaScript

```
document.getElementById('hozzaszolaskuld').disabled = true;
```

Ezen kívül minden billentyűfelengedésre beregisztráljuk az `urlapEllenoriz` függvényt:

JavaScript

```
addEvent(document.getElementById('szoveg'),
         'keypress', urlapEllenoriz);
addEvent(document.getElementById('nev'),
         'keypress', urlapEllenoriz);
addEvent(document.getElementById('human'),
         'keypress', urlapEllenoriz);
```

A következő függvény csak a minimális szöveghosszakat ellenőrzi, ízlés szerint továbbfejleszthető pl. e-mail cím szintaxis-ellenőrzésére.

JavaScript

```
function urlapEllenoriz() {
  document.getElementById('hozzaszolaskuld').disabled =
    (document.getElementById('szoveg').value.length<10) ||
    (document.getElementById('nev').value.length<3) ||
    (document.getElementById('human').value.length<1);
}
```

4.9.2. Hosszú listák böngészése helyett

A GAMF-on működő BME Nyelvvizsgahely korábbi honlapján minden nyelvvizsga előtt tájékozódniuk kell a vizsgázóknak, hogy pontosan milyen körülmények között (mikor, hol stb.) fog lezajlani a vizsga. Mivel elég sok vizsgázó volt, nem triviális, hogy az információt hogyan is nyújtsuk a vizsgázóknak. (A 77. ábrán csak az információk töredéke látszik.)

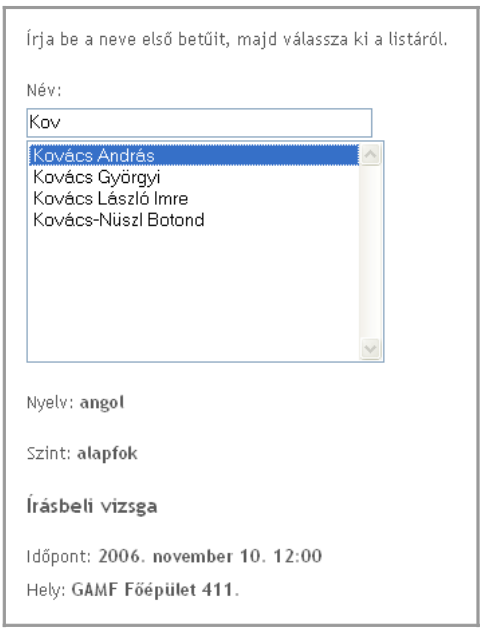
	A	B	C	D	E	F	G	H	I	J
1	Név	Nyelv	Szint	Tipus	Ib idő	Ib óra	Ib hely	Magnó idő	Magnó óra	Magnó hely
2	[illegible]	Angol	Alap	A				2007.03.30	14::00	GAMF Főép.41
3	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
4	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
5	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
6	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
7	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
8	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
9	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
10	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
11	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
12	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
13	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.41
14	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.41
15	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.41
16	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.41
17	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.41
18	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.41
19	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.41
20	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.41

77. ábra. Vizsga eredmények

Néhány hagyományos megoldási lehetőség:

- valamilyen offline médium használata (pl. egy Excel táblázat), ebben történik a szervezési információk előállítása és publikálása is
- hosszú táblázatos oldal kialakítása, amiben már lehet szövegesen is keresni (nagyon hosszú tartalom esetén a lapozás szükségessé válhat, ami szintén nem túl szép megoldás)

Az itt következő megoldás az előző megközelítések nehézségeit próbálja kiküszöbölni. A Végeredmény működés közben:



78. ábra. Dinamikus lista

A vizsgázónak elegendő a neve két-három betűs részét beírnia ahhoz, hogy a sok név közül csak azt a néhányat lássa, amelyikből már könnyedén ki tudja választani a saját nevét. (A 78. ábrán a `Kov` begépelése látszik.)

Második lépésként egyetlen kattintás elegendő, hogy a számára fontos információk jelenjenek meg az oldalon (időpont, helyszín stb.).

Az űrlap felépítése

A megoldás logikája röviden az, hogy a név minden egyes karakterének leütése vagy törlése után automatikusan meghívódik a `gepel` függvény, ami a találati lista megjelenítéséért felelős.

A névlistán való kattintás esetén a `valaszt` függvény gondoskodik a keresett adatok megjelenítéséről.

A következő kódrészletek nem a forráskódban előforduló sorrendben, hanem a végrehajtás sorrendjében kerülnek bemutatásra.

```
<form method="get" action="#" onSubmit="return false;">
```
JavaScript

Az űrlap tényleges adatküldést nem fog megvalósítani. Erről az `onSubmit="return false;"` gondoskodik.

```
  <label accesskey="n">                                          JavaScript
    Név:<br>
    <input type="text" id="nev" name="nev" value=""
      onKeyUp="gepel();"></input>
  </label><br>
```

Minden karakter felengedésre (`onKeyUp`) meghívódik a `gepel` függvény.

```
  <select size="10" id="lista" name="lista"                      JavaScript
    onChange="valaszt();">
  </select>
</form>
```

A listára kattintáskor a `valaszt` fut le.

Szükséges még bemutatni, hogy hol is fognak megjelenni a keresett adatok. Például az írásbeli vizsga adatai:

```
<div id="irasbeli" style="display: none">                        JavaScript
  <h2>Írásbeli vizsga</h2>
  <p>Időpont: <span id="iido"></span><br>
    Hely: <span id="ihely"></span></p>
</div>
```

Alapértelmezetten a doboz nem jelenik meg (`display: none`), hiszen nem biztos, hogy a vizsgázóknak lesz egyáltalán írásbeli vizsgája. Másrészt érdemes megfigyelni a (most még) üres `span` elemeket: itt fognak megjelenni a keresett adatok.

Adatok tárolása

Mielőtt a `gepel` függvényt megvizsgálnánk, át kell gondolnunk, hogy honnan fogja venni a kódunk a megjelenítendő adatokat. Több megközelítés is szóba jöhet, mi itt csak a JavaScript változó használatát vizsgáljuk meg.

Egyszerűbb esetekben és kisebb adatmennyiség (néhány kb) esetén tökéletesen megfelel az adatok tömbben való tárolása. Jelen megoldás ezt az egyszerű módszert alkalmazza:

```
var vizsgazok = new Array(                                       JavaScript
  new Array("Antal Alma","a","a","C","7","1","8:45","8"),
  new Array("Bera Berta","n","k","B","","","","","  "),
  ...
)
```

Jelen fejezetben azzal nem foglalkozunk, hogy ezt a tömböt hogyan is állítjuk elő. Mint lehetőség, meg kell azonban említeni, hogy szerver oldalon nem csak HTML, hanem akár JavaScript kódot is generálhatunk, így nincs annak sem akadálya, hogy PHP-vel az adatbázisból származó adatokat JavaScript tömbbé írjuk ki, hogy a fenti eredményt kapjuk. Még egyszerűbb esetekben a tömb „kézi” módszerekkel is előállítható.

Komolyabb adatmennyiség esetén a 4.10. fejezetben bemutatásra kerülő AJAX alapú megoldás jöhet szóba.

A `gepel` függvény

A `gepel` függvény néhány fontosabb részlete:

```
function gepel(){                                                    JavaScript
  var nev = document.getElementById('nev')
  var lista = document.getElementById('lista')
```

Kinyerjük a név begépelésére és a névlista megjelenítésére szolgáló objektumot.

```
  var str = ''                                                       JavaScript
```

Az `str` változóban fogjuk felépíteni a névsort. Ha nem üres a név mező, listát generálunk:

```
  if (nev.value.length>0) {                                          JavaScript
    for (i=0; i<vizsgazok.length; i++) {
      var vnev = vizsgazok[i][0];
```

Ha az aktuális név tartalmazza a minta szöveget (`nev.value`), akkor jó nevet találtunk:

```
      if (vnev.indexOf(nev.value) != -1) {                           JavaScript
        str += '<option value="o' + i + '">' + vnev + '<'
               + '/option>\n'
      }
    }
```

Ha üres a név mező, törölni kell mindent:

```
  } else {                                                           JavaScript
    var szobeli = document.getElementById('szobeli')
    szobeli.style.display = 'none'
    var irasbeli = document.getElementById('irasbeli')
    irasbeli.style.display = 'none'
  }
```

Végül egy „technikai” érdekesség: Az Internet Explorer hibásan működik, ha az `innerHTML` adattagot `select` elemre használjuk. Ezért – a nem szabványos – `outerHTML`-t kell alkalmaznunk helyette:

```
  if (lista.outerHTML) {                                             JavaScript
    lista.outerHTML =
      '<select size="10" id="lista" name="lista"'
      .' onChange="valaszt();">' + str + '</select>'
  } else {
    lista.innerHTML = str
  }
```

A `valaszt` függvény

Az előző függvény alapján sok részfeladat már könnyen elkészíthető.

A kiválasztott név kinyerése:

```
  var nev = lista.options[lista.selectedIndex].text                  JavaScript
```

Megkeressük, hogy a tömb melyik indexén találhatóak a keresett adatok:

```
for (i=0; i<vizsgazok.length; i++) {
  if (vizsgazok[i][0] == nev) {
    break
  }
}
```
JavaScript

A példán jól látszik, hogy a kód csak egyedi nevek esetén működik hibátlanul.

A függvény további része a talált adatok megfelelő beillesztésére szolgál.

4.10. AJAX bevezető

Jesse James Garrett 2005 elején publikált cikkében[106] alkalmazta először az AJAX kifejezést. A JavaScript történetében mérföldkőnek számít az AJAX technológia megalkotása és robbanásszerű elterjedése. Ha nem ismerős ez a fogalom, akkor először érdemes pl. a GMail hagyományos nézetet és az egyszerű HTML módját összehasonlítani.

Sajnos ez a papír alapú média nem képes érzékeltetni megfelelően a használhatósági és felhasználói élménybeli különbségeket.

Az AJAX az *Asynchronous JavaScript and XML* kifejezés rövidítéseként jött létre. Nagyon egyszerűen fogalmazva az AJAX lehetővé teszi, hogy az oldal úgy reagáljon a felhasználói interakciókra, hogy nem kell hozzá (mindig) újratölteni az egész weboldalt. A látogató nem fog másodpercekig várni 1-1 kattintás után, hogy az előbb a szerveren, majd a kliensen is érvényesüljön. Ehelyett a kliens él, mozog, reagál, és közben a szerverrel is kommunikál.

Nézzünk meg egy nagyon egyszerű helyzetet. A látogató regisztrálni szeretne, és ezért felhasználói nevet választ magának. Hagyományos JavaScript módszerrel itt annyi kényelmi szolgáltatást adhatunk, hogy a szintaxis ellenőrzést elvégezhetjük (például a minimum hosszúság megvan-e, és csak megengedett karaktereket adott-e meg). Azt azonban, hogy a név már foglalt, csak az űrlap elküldése után tudjuk jelezni. AJAX segítségével akár „azonnal” jelezhető a foglaltság.

AJAX kommunikáció

A hagyományos webalkalmazások a felhasználó által bevitt adatokat egy űrlap formájában küldik el a kiszolgálónak. Miután az feldolgozta az adatokat, egy teljesen új oldalt küld vissza a böngészőnek. Mivel minden felhasználói adatbevitel után a kiszolgáló új oldalt küld, ezek az alkalmazások általában lassúak és kevésbé felhasználóbarátok.

AJAX-szal azonban a weboldalak úgy küldhetnek és fogadhatnak adatokat a kiszolgálónak, hogy nem szükséges az egész oldalt újra letölteni. Ez úgy lehetséges, hogy az oldal a háttérben küld egy HTTP kérést a kiszolgáló felé, majd JavaScript segítségével csak a weboldal egy részét módosítjuk, ha megérkezik a válasz.

106 http://www.adaptivepath.com/publications/essays/archives/000385.php

Habár bármilyen formátumot használhatunk az adatok küldésére/fogadására (akár egyszerű szöveget is), legelterjedtebb az XML használata. (Bár a JSON[107] népszerűsége is egyre növekszik.)

4.10.1. Bevezető példa

A következő példa azt mutatja be, hogyan tud egy weboldal kommunikálni a szerverrel az oldal teljes újratöltése nélkül. Nézzünk egy egyszerű szöveges beviteli mezőt, amelyik egy név begépelését teszi lehetővé.

HTML

A kezdetben üres javaslatlista a `txtHint` span mezőbe fog kerülni, az éppen begépelt szövegnek megfelelően. (Például az „A” begépelésére megjelenik az összes `A` betűvel kezdődő név.)

HTML

```
<form>
  Keresztnév:
  <input type="text" id="txt1" onkeyup="showHint(this.value)">
</form>
<p>Javaslat: <span id="txtHint"></span></p>
```

`showHint`

Minden egyes billentyűleütés (pontosabban felengedés: `onkeyup`) esetén lefut a `showHint` JavaScript függvény. Érdemes megfigyelni az átadott paramétert: a `this.value` az aktuális input elem (`this`) begépelt értékét (`value`). A függvény néhány kötelező ellenőrzéssel kezd, a lényegi kód a függvény végén található:

JavaScript

```
function showHint(str) {
  if (str.length==0){
    document.getElementById("txtHint").innerHTML=""
    return
  }
  xmlHttp=GetXmlHttpObject()
  if (xmlHttp==null) {
    alert ("Browser does not support HTTP Request")
    return
  }
  var url="gethint.php"
  url=url+"?q="+str
  xmlHttp.onreadystatechange=stateChanged
  xmlHttp.open("GET",url,true)
  xmlHttp.send(null)
}
```

Ha a beviteli mező üres (`str.length==0`), akkor nincs javaslat (`innerHTML=""`).

107 http://www.json.org/

Ha ilyenkor a teljes listát kiadná, akkor a lényeg veszne el. Egy ilyen helyzetben pont az az AJAX megoldás előnye, hogy nem kell előre letölteni a több ezer választási lehetőséget egy select tagba, hanem minden esetben csak egy jóval szűkebb találati listát.

Következő lépés az AJAX működés lelkét adó (példánkban `xmlHttp`) objektum létrehozása. (Mivel ez a lépés böngészőfüggő, később részletesen visszatérünk erre.)

Ha létrejött a kommunikációs objektum, akkor az URL összeállítása következik. Itt egyelőre annyi lényeges, hogy a megszólításra kerülő `gethint.php` a paraméterek alapján tud arról, hogy itt mi a kérés: az `A` betű begépelése esetén az url a következő lesz:

```
gethint.php?q=A
```

Természetesen itt a tényleges kódolás előtt az összes lehetséges helyzet alapján rögzíteni kell valamilyen paraméterátadási módszert, és ehhez kell alkalmazkodni a kérés kliens oldali összeállításakor és szerver oldali feldolgozásakor is. (Ezt úgy is mondhatjuk, hogy a kliens és szerver közti kommunikációra egy interfészt vagy protokollt dolgozunk ki.) A példában jól látszik, hogy a szerver oldalon a q paraméter értéke alapján kell a javaslatokat összeállítani.

A szerver oldalon például egy adatbázis táblában tárolt nevek esetén egy egyszerű, a következőhöz hasonló lekérdezésre lesz szükség:

```SQL
SELECT nev FROM nevek
WHERE nev like "A%"
```

Az utolsó három sor kicsit több magyarázatot igényel. Az AJAX aszinkron működése azt jelenti, hogy a szerver felé küldött kérésre a JavaScript nem várja (nem is várhatja) meg a választ, hanem a felhasználót hagyja tovább dolgozni (esetünkben gépelni). A kommunikációs objektumunk számára tehát egy ún. callback (visszahívható) függvény nevét adja meg az `xmlHttp.onreadystatechange=stateChanged` sor. Amikor a szervertől visszaérkezik a válasz, akkor a JavaScript kódunk erről úgy fog értesülni, hogy a `stateChanged` függvény kapja meg a vezérlést. Másként fogalmazva: a `stateChanged` függvényt kell képessé tennünk arra, hogy a visszaérkező választ feldolgozza.

Az `xmlHttp.open` függvény a kérést írja le: `GET` metódussal szeretnénk a kérést küldeni az url számára, a `true` paraméter pedig az aszinkronitást írja elő.

Aszinkronitás nélkül a böngésző lemerevedne a válasz megérkezéséig. Ez (különösen lassabb hálózati kapcsolat esetén) nem a felhasználói élményt növelné, hanem inkább korlátozná a munkát.

Végül az `xmlHttp.send` függvény végzi a kérés tényleges elküldését.

`stateChanged`

```JavaScript
function stateChanged() {
  if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete"){
    document.getElementById("txtHint").innerHTML=
        xmlHttp.responseText
  }
}
```

A függvény többször is meghívódik a válasz teljes megérkezéséig, de az esetek túlnyomó többségében elég akkor foglalkozni vele, amikor a teljes válasz megérkezett (`readyState==4`).

Ahogy eddig is látszott, ez a példa az alapok rövid áttekintését tűzi ki célul, és nem a teljes és komplex megoldást. A válasz esetén is látható, hogy itt nem XML válasz érkezik a szervertől, hanem szöveges (akár HTML). Egyszerűbb esetekben ez tökéletes megoldás lehet.

4.10.2. A böngészők AJAX támogatása

Az `XMLHttpRequest` objektum nélkül nincs AJAX.

Sajnos a böngészők nem teljesen egységesen kezelik az (egyébként szabványban rögzített) `XMLHttpRequest` objektumot. Ez azonban nem olyan nagy probléma, a következőhöz hasonló kóddal a probléma jól kézben tartható:

JavaScript
```
function GetXmlHttpObject(handler) {
  var objXMLHttp=null
  if (window.XMLHttpRequest) {
    objXMLHttp=new XMLHttpRequest()
  } else if (window.ActiveXObject) {
    objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
  }
  return objXMLHttp
}
```

A függvény először a Mozilla alapú böngészőkön alkalmazható megoldással kezd, majd Explorerben ActiveX vezérlőt próbál létrehozni. A kódból az is következik, hogy az AJAX használatához a JavaScripten túl az ActiveX engedélyezésére is szükség van Explorer esetén.

Az előző példa egyszerű szöveges (*AHAH*[108]) kommunikációt valósított meg. Összetettebb esetekben többnyire XML vagy JSON alapú kommunikáció szükséges.

108 Asychronous HTML and HTTP

Tartalomjegyzék

Hogyan tovább?

Ha eddig eljutottál – és nem csak hátralapoztál –, kedves Olvasó, akkor nyakig benne vagy a webfejlesztés mocsarában. Mert aki belekóstol és beleszeret az alkotó munka örömeibe, azt fogva tartja, magával ragadja ez a fantasztikus világ. Én pedig veszem a bátorságot, és – nem tiszteletlenségből, hanem mint szakmabelit – tegezni foglak e rövid fejezet erejéig.

Az alapok megismerése után jól látszik, hogy az ismeretlen nem csökkent, hanem még nagyobb területeket sejtet az ismeret határain túl. Ha csak számba vesszük a négy alaptémánk következő lépéseit, látszik, hogy sok felé van mit bővíteni a tudásunkon.

A HTML 5 és a CSS3 szabvány még nem végleges, de nap mint nap újabb tutorialok, trükkök, megoldások jelennek meg a szakmai közösség tollából. Talán soha nem volt ilyen látványos verseny a böngészők fejlesztői között, szinte hetente jelennek meg az újabb buildek valamelyik böngésző háza tájról. A szabványok rengeteg újdonságával kell a közeljövőben megismerkednünk.

A JavaScript területén is jelentős verseny zajlik a böngésző gyártók között. Nem csak a teljesítményben, hanem a verziószámokban is nagy a pörgés. Ezen kívül az elmúlt 5-6 évben a *Prototype*[109], *jQuery*[110] és társai nagyban gyorsítják, látványossá és élvezetessé teszik a fejlesztők munkáján kívül a látogatók tevékenységét is. Itt is sok továbblépési lehetőség van a tanulásban.

A szerver oldalon a bemutatott keretrendszerek, tartalomkezelő rendszerek, a tervezési minták mellett még a PHP-n kívüli világ is hátra van: a Perl, Ruby, Python, Java és .Net bármelyike évekre elegendő tanulni valót nyújt.

Végül két személyes javaslatot hagy adjak a szakmai továbblépéshez:

- Napi szinten olvasd a Weblabort[111], csatlakozz a szakmai közösség életébe a saját aktivitásoddal is.
- Ismerd meg közelebbről a Drupalt[112]! A Drupal ismeret és az eddig elsajátított webfejlesztési alapismeretek nagyszerű szakmai hátteret adnak a munkakereséshez, a karrierépítéshez vagy a vállalkozáshoz, cég alapításhoz.

Sok sikert kívánok!

Nagy Gusztáv

109 http://www.prototypejs.org/
110 http://jquery.com/
111 http://weblabor.hu/
112 http://drupal.hu/