TÁMOP-4.1.1.F-14/1/KONV-2015-0009
„A GÉPÉSZETI ÉS INFORMATIKAI ÁGAZATOK DUÁLIS ÉS MODULÁRIS KÉPZÉSEINEK KIALAKÍTÁSA A PÉCSI TUDOMÁNYEGYETEMEN „

Dr. Nagy Enikő

Web-programozás I

Pécs
2015

A tananyag a TÁMOP-4.1.1.F-14/1/KONV-2015-0009 azonosító számú, „A gépészeti és informatikai ágazatok duális és moduláris képzéseinek kialakítása a Pécsi Tudományegyetemen” című projekt keretében valósul meg.

PÉCSI TUDOMÁNYEGYETEM
UNIVERSITY OF PÉCS

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

Web-programozás I

Dr. Nagy Enikő
Szakmai lektor: Gyurák Gábor
Nyelvi lektor: Veres Mária

Pollack Press
7624 Pécs, Boszorkány u. 2.

Felelős kiadó:

ISBN szám

Pécsi Tudományegyetem
Műszaki és Informatikai Kar

Pécs, 2015

TARTALOMJEGYZÉK

Web-programozás I

ÁBRÁK JEGYZÉKE

TÁBLÁZATOK JEGYZÉKE

Web-programozás I

1. Bevezetés

A magyar felsőfokú informatikusképzésről elmondható, hogy a képzésfejlesztés folytonos kihívások elé állítja az oktatásban szereplő szakembereket. Az Európai Unió által deklarált globális kérdéskörök – közéjük tartozik az informatika is – hatással vannak a felsőoktatásra. A felsőoktatási szektor versenyképességének javítása ugyanis az informatikai képzési eljárások folyamatos fejlesztését igényli, és ez kemény feladatokat jelent a szakembereknek. Az információs technológia hallatlan fejlődésével új kommunikációs csatornák, új internetes felületek jönnek létre, amelyek folytonos változásban tartják az informatikai képzési területet.

Ezenkívül mindennapivá váltak a jelenkori információs társadalom környezetének változásai, és ehhez kapcsolódóan az elektronikus környezeté is. A hálózat alapú kommunikációs megoldások jelentősége megnőtt, amely nagy hatással van munkánkra, életünkre. Az online világnak meghatározó a szerepe már a hétköznapjainkban is. Eddigi tananyag-fejlesztési munkáimmal, elkészült jegyzeteimmel ebből következően az volt a célom, hogy segítséget nyújtsak az elsősorban nem informatikai szektorban elhelyezkedni szándékozó hallgatóknak az eligazodásban, az új ismeretek átadásában.

Mára viszont elmondható, hogy szinte nincs is olyan munkakör, ahol ne lenne szükség informatikai tudásra. A jegyzetkészítésnél tehát azt a fajta segítségnyújtási koncepciót választottam, amellyel azt szeretném elérni, hogy olyan segédanyagot adjak a kedves olvasók kezébe, amelynek áttanulmányozása segíthet mind az informatikai, mind a nem informatikai szektorban történő munkák elvégzésében.

Konkrétabban megfogalmazva, ezt a jegyzetet olyan, a webprogramozásban kezdő hallgatóknak ajánlom, akik szeretnének megismerkedni a statikus és a dinamikus honlapok fejlesztésével, legyen az magán vagy akár üzleti jellegű honlap. Ez a tananyag olyan alapozó ismereteket foglal magába, melyekkel leendő informatikusok is, és más szakon tanulók, de a téma iránt érdeklődők is megbirkózhatnak.

Ennek a jegyzetnek tehát az elsődleges célja, hogy hozzásegítse a hallgatót a webprogramozás alapjainak elsajátításához.

Ezek után már csak az a kérdés merülhet fel, hogy mit értünk webprogramozáson? A programfejlesztésnek olyan aspektusait, amelyeknek eredményei webes felületekre készülnek. A webböngésző, a webszerver, a HTTP-protokoll, a HTML nyelv, az URL-címzés, a kliens, a JavaScript, a PHP stb. olyan kifejezések, amelyeknek egy webprogramozó alapszókincsének részévé kell válniuk, és a programozónak a megértésük olyan szintjét kell birtokolnia, amellyel képes webes alkalmazások fejlesztésére.

Bízom benne, hogy munkámmal hatékonyan hozzájárulok ahhoz, hogy egy átlagos felhasználó számára olyan kiterjedt ismereteket nyújt az anyag, amelyekre szüksége lesz webes felületen futó távoli alkalmazások megértéséhez és fejlesztéséhez.

a szerző

2. A tantárgy célkitűzése

A felsőfokú alapképzési szakok képzési és kimeneti követelményrendszerében szerepel, hogy alapfokozat birtokában a mérnök informatikusoktól legalább két kompetencia elvárt arra vonatkozóan, hogy alkalmas legyen számítógépes és távközlő hálózatok telepítésére, konfigurálására, hibaelhárítására, üzemeltetésére, továbbfejlesztésére; kliens szerverrendszerek programozására, webprogramozásra, vállalati információs rendszerek folyamatalapú funkcionális tervezésére és készítésére, valamint döntéstámogató rendszerek tervezésére, készítésére, működtetésére. Ebből következően, jelen tananyaggal a webprogramozáshoz szükséges kompetenciák, ismeretek és attitűdök kifejlesztése az alapvető cél.

A webprogramozás tantárgy célkitűzéseit tekintve tehát a következőket fogalmazom meg:

- Alapvető cél – a bevezető ismereteken túl – megérteni a webes felületek és alkalmazások működését.
- Cél, hogy megismerjük az internet működését, és az ezzel kapcsolatos technikai megvalósításokat, protokollokat.
- Olyan szakkifejezések, fogalmak ismereteit, azaz szakszókincsét kell elsajátítanunk, mely nélkül nem válhatunk sikeres szakemberré.
- A jegyzetben célul tűzzük ki a kliens és szerver oldali programozás megértését, az ehhez tartozó kompetenciák kifejlesztését.
- Kliens oldalról a HTML5 leíró nyelv alapjainak megismerése, és ehhez kapcsolódóan a használati rutin megszerzése elsődleges célként szerepel.
- A statikus és dinamikus weboldalak közötti különbségek, hasonlóságok és összekapcsolódások megértése, és ide tartozik a JavaScript programozásba való bevezetés is.
- Jelen jegyzetben viszont nem kívánunk hangsúlyt fektetni a honlapok formailag professzionális megjelenését szolgáló CSS stíluslapok megismerésére. Erre a bevezető kurzus utáni, ismereteket elmélyítő tanfolyamon kerülhet sor.
- A szerver oldali programozás megismeréséhez és megértéséhez a PHP programozási nyelvvel kapcsolatos alapfeladatok megoldása a cél.
- Végül, de nem utolsósorban, célkitűzéseink között szerepel, hogy a hallgatók olyan webprogramozási alaprutint és ismereteket szerezzenek, melyeknek birtokában képesek lesznek majd önfejlesztéssel saját célokat kialakítani, mi több, azokat teljesíteni is.

3. Követelmények

Jelen tananyag elsajátításához elengedhetetlenek a következők:

A tananyag feldolgozása során feltételezzük, hogy a hallgató birtokában van a középiskolai informatikai ismeretanyagnak, a közoktatásról szóló informatikai vonatkozású tananyag előírásai alapján.

Számítástechnikai alapismeretek, számítógépes hálózatok működésének alapjai, valamint dokumentumkezelési alapok és készségek szükségesek az anyag megértéséhez, készségszintű elsajátításához.

Ezen kívül, mivel programozási feladatok megoldására kerül sor, szükség lesz alapvető algoritmikus gondolkodásmódra. Ide tartozóan feltételezzük a programozási alapokkal kapcsolatos ismereteket.

Mivel alapvetően internetes felületekkel foglalkozunk, ezért fontos, hogy a hallgató jártas legyen az internetes böngészésben, elektronikus tartalmak felhasználói szintű kezelésében.

Alapkövetelményként említendő továbbá az alapfokú angolnyelv-tudás, illetve a Közös Európai Referenciakeret nyelvtudás fokozatait tekintve a minimum A2-es szint.

Infrastrukturális megközelítésből a tananyag elsajátításához szükségünk lesz internetkapcsolatra, és a webes fejlesztésekhez szükséges telepítésekre alkalmas asztali számítógépre vagy laptopra.

4. Alapvető tudnivalók a webről

A hétköznapi életben sokszor találkozhatunk azzal a jó tanáccsal, ha valamit tudni szeretnénk, hogy „Nézd meg az interneten!” , „Keress rá a neten!”, vagy esetleg „Guglizd meg!”. Ezek a hétköznapi kifejezések bizonyítékai annak a természetessé váló ténynek, hogy napjainkra kétségtelenül jellemző az internet evidens használata, már nemcsak szakemberek, hanem laikusok számára is. Amikor azonban túllépünk a hétköznapi használaton, információkeresési tevékenységeken, és olyan szakismereteket szeretnénk szerezni, amellyel nemcsak böngészni, hanem hozzátenni, jó értelembe véve beleszólni és dinamikus szereplőivé, fejlesztőivé szeretnénk válni az internetes világnak, akkor első lépésként meg kell értenünk annak működését.

Ennek a fejezetnek így az a célja, hogy megalapozzuk az internet működéséről ismereteinket, megértsük az alapvető technológiákat, protokollokat. Ehhez hozzátartozik, hogy megértsük a fontos és alapvető informatikai szakkifejezéseket, szakszavakat, illetve azok jelentéseit, amelyek használatának be kell ivódnia a szakmai szókincsünkbe ahhoz, hogy hatékony webfejlesztővé válhassunk.

Ezen túlmenően a fejezet elméleti szinten elvezeti az olvasót a kliens–szerver kapcsolat létrejöttétől kezdve, azok működésén át a kliens és szerver oldali programozási nyelvek használatáig. Ezzel rávezetve a hallgatót a gyakorlati munka megkezdésére, a weboldal fejlesztési tevékenységének megtanulásához.

A számítógépek fizikai összekapcsolódása

Első lépésként nézzük meg, hogy egy felhasználó által kezelt gép fizikailag hogyan tud kapcsolatba lépni a világ bármely részén lévő másik géppel. Ahhoz, hogy a számítógépek kommunikálni tudjanak egymással, össze kell kapcsolni őket. Az összekapcsolt gépek által létrejött világháló a hálózatok hálózataként működik. Ha lemodellezzük ezeket a kapcsolati hálókat, akkor úgy kell elképzelni, hogy valamilyen, úgynevezett hálózati eszközzel csatlakoztatják a gépeket egymáshoz. Legáltalánosabb csatlakozási forma szerint csillag alakzatban rendezik a gépeket, amely csillag közepén egy csomópont van, amihez minden egyes gép csatlakozik. Ebben a csomópontban helyezkedik el a switch (kapcsoló), aminek feladata, hogy a kapcsolatot fenntartsa a csatlakoztatott gépek között. Nehéz lenne azonban megvalósítani, hogy a világ összes számítógépét egy ilyen csillagba rendezzük, így felmerül a kérdés, hogy mit tudunk kezdeni több ilyen csomóponthoz rendelt sok-sok hálózattal. Természetesen léteznek a csillag elrendezési formán kívül más kapcsolódási topológiák is, de mivel ezt másik kurzus tárgyalja, nem szeretnék kitérni ennek a részletezésére. Ahhoz, hogy egy számítógép olyan számítógéppel tudjon kommunikálni, amelyik nem ugyanahhoz a switch-hez kapcsolódik, szükség van egy újabb eszközre, ami elvezeti a gépet a kiválasztotthoz. Ennek a hálózati eszköznek a neve a forgalomirányító (router), aminek egyszerűen fogalmazva az a feladata, hogy egy switch-re

csatlakoztatott számítógépcsoportot hozzákapcsoljon a világhálóhoz. Ezen hálózatok hálózata révén alakul ki az óriási világhálózat, azaz az internet.

Végül fontos megemlíteni, hogy a számítógépek hálózat(ok)ba kapcsolódásához és egymással történő kommunikációjához elengedhetetlen, hogy minden egyes gépben legyen hálózati kártya (network interface card), mely lehet vezetékes és vezeték nélküli is. Az önálló számítógépek világhálóba rendezésének ez is alapvető feltétele. A mai IOT (Internet of things) világban ezek az eszközök olyan tartozékok, amelyek egyaránt kötelezőek, alapvetőek és megkockáztatva a kifejezést, természetesek is.

A web működése

Miután „megtörtént" a világ számítógépeinek fizikai összekapcsolódása, következő lépésként arra vagyunk kíváncsiak, hogyan tudnak az adatok helyet cserélni, vándorolni a gépek között. Ilyen kérdésekre keressük a válaszokat: hogyan tudják a számítógépek azonosítani magukat, hogyan ismerik fel egymást, vagy a teljesség igénye nélkül, honnan tudja az én számítógépem egy begépelt karaktersorból, hogy milyen weboldalakat, információkat kívánok megjeleníteni a kijelzőmön.

Kiindulópontként illik tudnunk, hogy a World Wide Web (www, 1990, CERN, Tim Berners-Lee nevéhez fűződik a kifejlesztése) tulajdonképpen végtelen mennyiségű adat elhelyezésére szolgál, és a világhálót, az internetet használja fel az adatok elérésére. A fizikailag összekötött gépek között a kommunikáció alapvetően a HTTP protokoll segítségével, a címzés pedig az IP-címek segítségével történik.

Fontos megjegyezni, hogy az IP-címet nem szabad összekeverni a hálózati kártyánk egyedi címével (fizikai cím), ugyanis valójában ez utóbbinak számunkra nincs jelentősége. Az IP-címhez még szükség van az alhálózati maszkra is, hogy a számítógép el tudja dönteni, hogy a címzett géppel azonos hálózatban van-e, vagy a router felé kell küldenie az adatot, amely átjáróként fogja adott esetben továbbítani a megfelelő irányba a küldött információt.

Még mindig az IP-címnél maradva, felmerül a következő probléma, miszerint ez egy számsor, így rendkívül nehéz értelmeznünk. Másik oldalról közelítve pedig hétköznapi felhasználóként nem is találkozunk szinte az IP-címekkel, mert hiszen értelmes nevekkel ellátott webcímeket szoktunk a böngészőnkbe beírni, ha el szeretnénk érni egy honlapot. Kérdés: hogy jön össze az IP-cím és a webcím?

Ez a kérdés legegyszerűbben úgy magyarázható el, hogy van egy szolgáltató, amelyik felelős azért, hogy elvégezze a szükséges névfeloldást. A DNS (Domain Name System) egy olyan rendszer, amelynek a hálózat gépei kéréseket küldenek, és válaszok formájában megadja a kért IP-címet vagy, fordított esetben, a kért nevet. Úgy kell elképzelni, hogy a DNS egy hatalmas adatbázissal rendelkezik, sőt több DNS több adatbázissal, amelyek szintén kapcsolatban állnak egymással, és folyamatosan frissítve a változásokat, segítik egymást.

Az URL (Uniform Resource Locator) vagy az URI (Uniform Resource Identifier) tökéletesen egyedi mutatót biztosít a teljes interneten egy meghatározott erőforráshoz

(resource). Ez az erőforrás lehet egyszerű, mint például egy fájl vagy egy mappa, de hivatkozhat egy bonyolultabb objektumra is, mint például egy adatbázis lekérése vagy egy keresőmotor. Egy szabályos teljes URL elemei a protokoll (http://), a domain név vagy magyarul körzetnév (valami.hu), a portszám (:8080) és az elérési út (fomappa/almappa/index.html). Ennek elérésére szolgál tehát a számítógép IP-címe, ami a névszerverekben tárolt információk alapján egyértelműen megfeleltethető az egyes domain neveknek.

Két gép tehát a DNS szolgáltatás segítségével találja meg egymást IP-cím és név alapján.

Miután már tudjuk, hogy két gép hogyan „találja meg" egymást, azt is meg kell tudni határozni, hogy a gépeken futó programok között honnan hová irányul konkrétan az adat, hiszen egy gépen több program is futhat. Erre a célra szolgálnak a portok, amelyek olyan kapcsolódási pontok, amelyek pontosan azonosítják a processeket, és ezek alapján meghatározódnak a kiválasztott alkalmazások. Általában a portszám automatikusan hozzárendelődik a futó alkalmazáshoz és a klienshez is, amely számok a 0 és 65355 számok között vannak. A HTTP egy különleges alkalmazás, amely szabvány számára a 80-as port van fenntartva. A gyakorlatban, amikor böngészőnkbe beírjuk a webes kérésünket, a 80-as portot nem is szoktuk kiírni, mert ez olyan gyakran használt szabvány, hogy a böngészőnk tudja a http:// beírásából, hogy ezzel a 80-as portra kell küldenie a kérést.

A HTTP

A HTTP (Hyper Text Transfer Protocol) egy kérés–válasz alapú protokoll a kliens és a szerver között. Esetünkben kliens alatt értjük a webböngészőt, szerver alatt pedig a kliens számára weblapokat kiszolgáló webszervert. A kliens kezdeményez a szerver felé egy kérést, amire a szerver választ küld a HTTP protokollnak megfelelően.

A HTTP jellemzői:

- kérés–válasz protokoll: csak kérésre kapunk választ, önállóan, új információt a webszerver nem képes küldeni a kliens számára,
- szerver-pollozás: időzített, ismétléses kérésküldés a szerver számára,
- ajax: háttérben történő kommunikáció,
- állapotmentes: állapotok változásai nem jelennek meg sehol protokoll szinten,
- ASCII szöveges protokoll.

Egy HTTP-kérés felépítése a szerver felé a következő elemekből áll:

GET/ POST http://www.xxxxxxxx.xxx/ HTTP/1.1

*Accept: */**

Accept-Language: en-US, hu-HU;

User-Agent: xxxxxxxxxxxxx

Accept-Encoding: xxxxxxxxx

Connection: xxxxxxxxxx

Host: www.xxxxxxxxx.xxx

Pragma: no-cache

A GET, illetve POST metódusokkal tudatjuk, hogy milyen műveletet szeretnénk elvégezni. A GET-tel a letöltést kezdeményezzük (pl. képet, videót, weblapot), a POST-tal a feltöltést (pl. űrlapadatokat). Mindkettő kérés üzenet, amivel lefele és felfele is lehet adatokat mozgatni. GET esetén a címsorban lesz az adat, POST esetén a http-törzsben.

Ezt követi a kért erőforrás webcíme, majd a protokoll verziószáma a http://www.xxxxxxxx.xxx/ HTTP/1.1. A következő fejlécsorokkal további fontos információkat juttatunk el a szerverhez, például, hogy milyen típusú fájlt szeretnénk letölteni (Accept), a várt válasz nyelvét, vagy, tegye-e gyorsítótárba (cache) a kért címet stb.

A kérésre a szerver által küldött HTTP-válasz felépítése a következő:

HTTP/1.1 200 OK

Date: Xxx, 00 Xxx, 0000 00:00:00

Server: Apache

Last-Modified: Xxx, 00 Xxx 0000 00:00:00

ETag:xxxxxxxxxxxxxxxxx

Accept-Ranges: bytes

Content-Length:000

Connection: close

Content-Type: text/html; charset=UTF-8

<!DOCTYPE HTML PUBLIC ”-//W3C//DTD HTML 5 Transitional // EN”>

<HTML>

….

Az első sorban a kapott válasz tartalmazza a verziószámot és az angol nyelvű indoklást (OK).

A példában a 200-as érték egy úgynevezett státuszkód, ami azonosítja a hibakódokat, vagy éppen a sikerességet. Ha például ez 500-as értékű, akkor szerverhibára utal, ha 400-as, akkor a kliens oldalon történt hiba, és ha 404-es, akkor nem található a kért tartalom. Aztán következik a fejléc után egy üres sor, ami jelzi, hogy a válasz lényegi része, a törzs következik, ami egy HTML (HyperText Markup Language) nyelven írt szöveg. Ez tartalmazza a kért weboldalt, illetve amit le akartunk tölteni az internetről.

Ezzel eljutottunk arra a pontra, hogy megismerjük, hogyan tudjuk precízen megjeleníteni a kívánt dokumentumot a weben. A következő fejezet sorra veszi a HTML5 mint leíró nyelv fontosabb elemeit, melyek segítségével statikus weboldalakat tudunk létrehozni, és ezek alapul szolgálnak az azt követő, szerver oldali nyelvek megismeréséhez.

5. Leíró nyelvek, a HTML

A leíró nyelvek lényege, hogy úgy kell a megjelenítendő dokumentum szövegrészeit megadni, hogy minden böngésző egységesen értelmezni tudja. Standardizált kódot kell tehát előállítanunk, hogy azt képesek legyenek a kellő intelligenciával rendelkező böngészők értelmezni és megjeleníteni. Az informatika több évtizedes története folyamán a leíró nyelveknek szép fejlődéstörténetéről beszélhetünk már, de a történeti beszámolótól most eltekintek azért, hogy a gyakorlati megvalósításra fókuszálhassunk. A leíró nyelvek családjába több, különböző irányultságú nyelv tartozik, és ebből adódóan sokunkban keveredhetnek az információk arról, hogy melyik nyelvet is válasszuk weblapunk megalkotásához. Sőt, hogy egy kicsit bonyolítsam a helyzetet, elmondható, hogy az évek alatt a nyelvi fejlődések során több esetben egyik nyelvre alapul a másik, de olyanok is akadnak, amelyeknek semmi közük egymáshoz. A nyelvek fejlődését valójában mindig valamilyen probléma felbukkanása előzte meg, és ezeknek a problémáknak a megoldása, kiküszöbölése vezetett el egy-egy újabb verzióig. Jelentős mérföldkőnek számított, amikor különválasztották a weblap tartalmi és formai megjelenésének kérdését. Ekkor kerültek használatba a stíluslapok (CSS), amelyek a weblapok formai megjelenéséért felelősek. Másik nagy lépésnek tekinthető a mobil eszközökre való optimalizálás is, mely a 2011-ben kiadott HTML5-szabvány újításának tulajdonítható.

Ebben a fejezetben a HTML5 nyelv alapelemeit tekintjük át, amelyek ismerete nélkülözhetetlen a weblap elkészítéséhez és a gyakorlati feladatok megoldásához.

5.1. Alapok

Statikus weblapok szerkesztésének megtanulásához egy egyszerű szövegszerkesztőre (editorra) van szükség, valamint a legismertebb böngészőkre, melyek birtokában internetelérés nélkül is történhet a weblap fejlesztése. Ilyen egyszerű szövegszerkesztő lehet a Windows Jegyzettömbje, vagy akár a Notepad++, és a napjainkban leggyakrabban használt böngészők pedig a következők:

Desktop/laptop (x86 processzoros) eszközökön:

- *Microsoft Windows és Linux operációs rendszerek alatt Firefox, Chrome és Opera;*
- *Apple (Mac) OS X operációs rendszerek alatt Safari, Chrome, Firefox, Opera.*

Mobil (ARM processzoros) eszközökön:

- *Microsoft Windows Phone Internet Explorer Mobile for WP,*
- *Android alatt Chrome for Android, Firefox for Android, Opera Mobile,*
- *Apple iOS alatt iOS Safari, Chrome for iOS, Atomic és Dolphin.*

Ha ezek megvannak és készen állunk a munkára, első lépésként célszerű létrehozni egy külön mappát a számítógépünkön, amelybe érdemes összegyűjteni a weblap alkotóelemeit, a különböző megjelenítendő szöveges és multimediális állományokat. A gyűjtőmappa neve bármi lehet, a webhely kezdő HTML-fájljának neve mindig index. A többi oldal fájlneve lehetőleg a tartalmukra utaló, ékezet nélküli kisbetűs szó legyen, és valamennyi fájlnév kiterjesztése .html. A böngészők, attól függően, hogy mi van beállítva a webszerveren, de leggyakrabban az index.html nevű oldalt fogják a webhely kezdő vagy címlapjának tekinteni. A stíluslapok nevei .css kiterjesztésű, ékezet nélküli kisbetűs szavak.

Tanulás során először érdemes a tartalmi (HTML) kódolást elsajátítani, majd azt követve a formázási (CSS) kódolást. Ha valaki a grafikus tartalomnak kiemelt jelentőséget tulajdonít, akkor ezek után célszerű megtanulni a grafikai elemek kódolásának (SVG, Scalable Vector Graphics) használatát. Jelen jegyzetben a tartalmi kódolásra helyezzük a hangsúlyt, formai megközelítésekkel csak említés szintjén foglalkozunk.

5.2. A HTML oldal felépítése

A HTML jelölő nyelv alap nyelvtani szabálya, hogy a tartalomba úgynevezett tageket (ejtsd: tegeket, magyarul címkéket) helyezünk el. Ezek határozzák meg, hogy a böngésző hogyan értelmezi a kódot. A címkék megnevezése kifejezi (angolul) a tartalmukat is. Az angol szó vagy rövidítés, mindig „<" és „>" („kisebb mint", ill. „nagyobb mint") jelek között helyezkedik el. Kisbetű/nagybetű-érzéketlenek, de csupa kisbetűt célszerű használni (az XHTML-es hagyomány miatt). A tagek többsége párban, kezdő és záró tagként szerepel, csupán annyi közöttük a különbség, hogy ami kezdő címkeként <...>, az záró címkeként </...>, azaz egy „per" vagy „törtvonal" (slash) karakterrel bővül. Léteznek páratlan tagek is, ami annyit jelent, hogy a nyelvtani elemhez nem tartozik záró tag. Továbbá fontos tudni, hogy az elemeknek lehetnek jellemzőik és azoknak értékeik. Tehát általánosan így néz ki egy HTML-séma:

<tag1 jell1="érték1" jell2="érték2">

tartalom… <tag0> … tartalom

</tag1>

<tag2 jell3="érték3" jell4="érték4">

tartalom

</tag2>

stb.

Fontos: A tagek egymásba ágyazhatók.

A kódolás áttekinthetőbb, ha struktúráját tekintve az új elemeket mindig új sorokba kezdjük. Érdemes törekedni arra, hogy a tagek hierarchiája a kódban is megjelenjen.

Egy weboldal alapvetően három fő részből áll: dokumentum típusának meghatározása (document type definition), fej (head), törzs (body).

Az alábbi példában látható, hogy a !doctype segítségével adjuk meg a böngésző számára, hogy milyen módon értelmezze a weboldal nyelvtanát. Ez azt jelenti, hogy a böngészőnk nem magától értelmezi a kódot, hanem egy standard szabály szerint:

<!doctype html>

<html lang="hu">

fej

törzs

</html>.

A böngésző számára a <html> tagnél kezdődik, és a </html> záró tagnél van vége a megjelenítendő weboldalnak. A lang jellemzővel javasolt megadni a weboldal nyelvét, ami az említett példánkban a magyar („hu").

A fej (head) HTML-kód értelmezését nézzük meg példán keresztül:

<head> – meta adatok megadásának kezdete

<meta charset="utf-8"> – a megadott szabvány kódkészlet használata

<title>.....</title> – a böngésző címsorában megjelenő cím megadása

<link rel="stylesheet" href=".....css"> – külső stíluslapok megadása

<style>.....</style> – CSS formázás

</head> – meta adatok vége.

Fontos: A meta adatok megadásának sorrendje kötött.

Végül a weblap harmadik, törzs (body) része az, amit a böngészőnk megjelenít. Megadása a következőképpen történik:

<body>

.....

tartalom …….

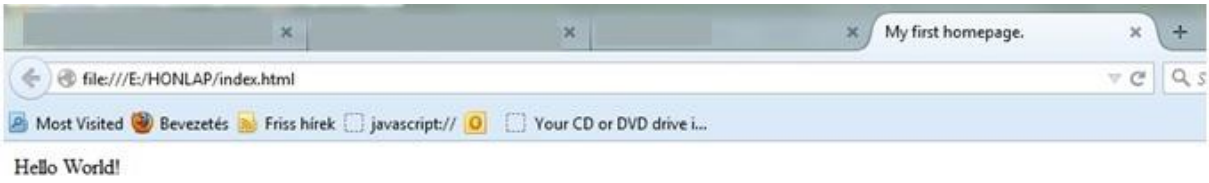
.....

</body>

Nézzük meg az első komplett példát, ami már egy weblapot prezentál!

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>My first homepage.</title>
        <link rel="stylesheet" href="….css">
        <style></style>
    </head>
    <body>
        Hello World!
    </body>
</html>
```

Megjelenése a böngészőben:



1. ábra „Hello World!" kiíratása

5.3. Szöveges tartalom

A szöveges tartalom szerkesztése kapcsán ötféle alap taget célszerű említeni. Ezek a következők:

<*h1*>.....</*h1*> <*h6*>.....</*h6*>, amely címsort definiál,

<*p*></*p*>, ami bekezdést definiál,

<*br*>, sortörést hoz létre,

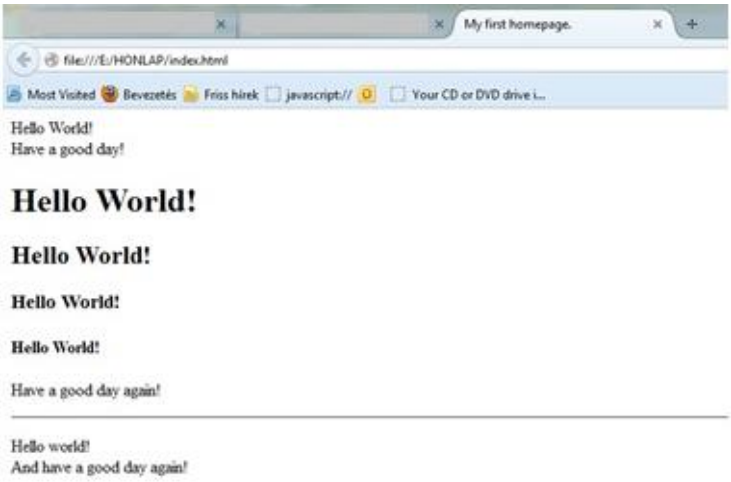
<*hr*>, vízszintes elválasztó vonalat szúr be, és végül a

<*!----*>, megjegyzés beszúrása vagy kódrészlet ideiglenes kiiktatása.

Mindezeket felhasználva az alábbi kódban,

```
<body>
    Hello World! <br> Have a good day!
    <h1> Hello World! </h1>
    <h2> Hello World! </h2>
    <h3> Hello World! </h3>
    <h4> Hello World! </h4>
    <p> Have a good day again! </p>
    <hr>
    <p> Hello world! <br> And have a good day again! </p>
    <!--This is an important note.-->
</body>
```

a következő eredményt kapjuk a képernyőn:



2. ábra Szöveges tartalom megjelenítése

5.4. Listák létrehozása

Felsorolások, listák létrehozásához konkrét tageket használhatunk weblapunk fejlesztése során. Három fajta listát különböztetünk meg: számozott vagy rendezett lista, számozatlan lista és a meghatározás vagy definíciós lista. Mind a három típust használhatjuk külön-külön

és egymásba ágyazva is. Egymásba ágyazás esetén érdemes a kódot tagolva, tabulátorokat használva szerkeszteni.

Számozott listák használatánál lehetőség van a lista kezdőértékének megadására, és a csökkenő sor beállítására is.

A meghatározás lista annyiban tér el az számozott listától, hogy a felsorolásokban általunk megadott szavakat, fogalmakat, kifejezéseket kapcsol egymáshoz. Ennek a listatípusnak egyik tipikus felhasználása a gyakran ismételt kérdések (GYIK) webes menüpontban fordul elő különböző weboldalakon.

A felsorolásoknál, listák létrehozásához használható tagek a következők:

A felsorolásoknál, listák létrehozásához használható tagek a következők:

<*ol*> a számozott, rendezett listához,

<*ul*> a számozatlan felsoroláshoz,

<*li*> a listaelemet definiálja,

<*dl*> a meghatározási listához,

<*dt*> a meghatározandó fogalom/kifejezés adható meg,

<*dd*> a társítást határozza meg.

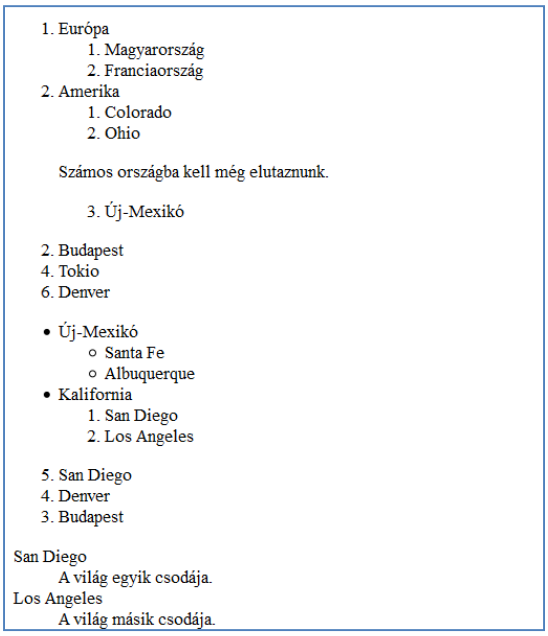
Fontos: Minden listaelem új sorban kezdődik, és a következő elem automatikusan az azt következő sorba kerül.

Nézzük meg, hogy a következő példakód mit eredményez!

```
<body>
    <ol>
        <li>Európa
            <ol>
                <li>Magyarország</li>
                <li>Franciaország</li>
            </ol>
        </li>
        <li>Amerika
            <ol>
                <li>Colorado</li>
                <li>Ohio</li>
            </ol>
                <p>Számos országba kell még elutaznunk.</p>
            <ol start="3">
                <li>Új-Mexikó</li>
```

```
                </ol>
            </li>
        </ol>
                <ol>
                    <li value="2">Budapest</li>
                    <li value="4">Tokio</li>
                    <li value="6">Denver</li>
                </ol>
        <ul>
            <li>Új-Mexikó
                <ul>
                    <li>Santa Fe</li>
                    <li>Albuquerque</li>
                </ul>
            </li>
            <li>Kalifornia
                <ol>
                    <li>San Diego</li>
                    <li>Los Angeles</li>
                </ol>
            </li>
        </ul>
        <ol reversed start="5">
                    <li>San Diego</li>
                    <li>Denver</li>
                    <li>Budapest</li>
        </ol>
        <dl>
            <dt>San Diego</dt>
                <dd> A világ egyik csodája.</dd>
            <dt>Los Angeles</dt>
                <dd> A világ másik csodája.</dd>
        </dl>
</body>
```

Ez a kódrészlet tartalmazza a lehetséges listaszerkezeteket, példát mutatva a háromféle listára és azok egybeágyazására. Futtatva a kódot, a böngésző a következő eredményt jeleníti meg:



3. ábra Listaszerkezetek

5.5. Táblázatok készítése

Ebben az alfejezetben a táblázatok szerkesztéséről lesz szó. Szükséges megjegyezni, hogy a táblázatokhoz kapcsolódóan a teljesség igénye nélkül most csak egy egyszerű táblázat létrehozására hagyatkozunk. Ide tartozóan fontos tudni, hogy a táblázat celláknak sorokban és oszlopokban rendezett halmaza, mely cellák tartalmazzák az ábrázolandó adatokat. Ha HTML-ben szeretnénk táblázatba foglalni adatainkat, fontos megjegyezni, hogy minden oszlopban ugyanannyi cellának kell lennie, így üres cellát is létre kell hozni adott esetben. Egy oszlopon belül valamennyi cellának ugyanolyan szélességűnek kell lennie, és a táblázat címét, fejlécét cellák egyesítésével tudjuk megszerkeszteni. Táblázatok létrehozásához az alábbi tagek állnak rendelkezésünkre:

<table>.....</table> táblázatot nyitó és záró tag,
<tr>.....</tr> vízszintes sor tag,
<td>.....</td> cellákban lévő adatokhoz tag,
<th>.....</th> sorok és oszlopok fejléceinek létrehozásához szükséges tag,
<caption>.....</caption> cím létrehozásához szükséges tag.

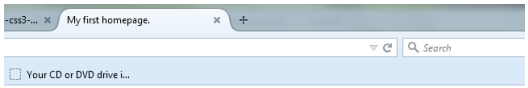
Példánkban az amerikai nagyvárosok lakosságáról szóló adatokat ábrázoljuk. Kódrészletünk a következő:

```
<table border="1" align="center" cellpadding="2" cellspacing="0">
<caption><b>Egyesült Államok nagyobb városainak lakossága</b></caption>
<thead>
    <tr>
        <th>Helyezet</th>
        <th>Város</th>
```

```
                <th colspan="2" align="left">Lakosság</th>
        </tr>
</thead>
<tfoot>
        <tr>
                <td colspan="4" align="center">
                <strong>A legfrissebb adatok alapján</strong></td>
        </tr>
</tfoot>
<tbody>
<colgroup align="center" span="2" valign="middle" />
        <tr>
                <td><i>1.</i>
                </td>
                <td>New York (New York)</td>
                <td>8.244.910</td>>
        </tr>
        <tr>
                <td><i>2.</i></td>
                <td>Los Angeles (Kalifornia)</td>
                <td>3.819.702</td>
        </tr>
        <tr>
                <td><i>3.</i></td>
                <td>Chicago (Illinois)</td>
                <td>2.707.120</td>
        </tr>
        <tr>
                <td><i>4.</i></td>
                <td>Houston (Texas)</td>
                <td>2.145.146</td>
        </tr>
</tbody>
</table>
```

A táblázat formázásához felhasználtuk a border, az *align,* a *cellpadding* és *cellspacing* jellemzőket, de ha további formázási műveleteket szeretnénk végrehajtani, azt mindenképpen CSS-ben javasolt elkészíteni a későbbiek folyamán, mert az szebb és modernebb megoldást kínál. Általában nagyobb méretű táblázatoknál az áttekinthetőséget segítik a példában is felhasznált *thead, tbody* és *tfoot* tagek. Több részre bontott táblázatnál a darabok így külön is kezelhetők, formázhatók, és nyomtatáskor a táblázat fej- és lábléce minden oldal tetején és alján is megismétlődik. A *col, colgroup, colspan* tagekkel pedig csoportokat képezhetünk és a csoportokhoz műveleteket rendelhetünk.

Jelen esetben a következő eredményt kapjuk a kódunk futtatásakor:



Egyesült Államok nagyobb városainak lakossága

Helyezet	Város	Lakosság
1.	New York (New York)	8.244.910
2.	Los Angeles (Kalifornia)	3.819.702
3.	Chicago (Illinois)	2.707.120
4.	Houston (Texas)	2.145.146
A legfrissebb adatok alapján		

4. ábra Táblázat megjelenése HTML-ben

Táblázatok kódolásakor CSS-sel, különböző tulajdonságokat rendelve a tagekhez, változatosabb formázási lehetőségeket használhatunk ki. Példaként említve azt az esetet, ha különböző színű hátteret szeretnénk adni különböző oszlopoknak.

Fontos: Ha az alapértelmezettől eltérő formázást szeretnénk alkalmazni, mindenképp alkalmazzunk CSS megoldásokat!

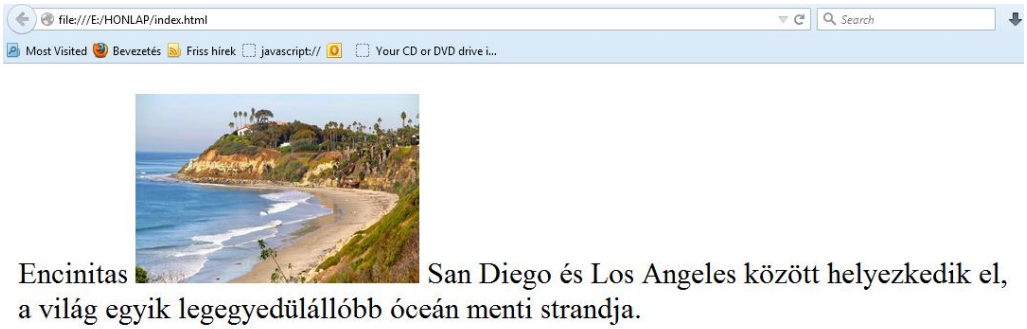
5.6. Képek használata

Képek elhelyezése a weblapra az *img* taggel történik. Kötelezően meg kell adni hozzá a forrás fájl elérési útvonalát az *src* jellemző segítségével. Ha a beszúrandó kép a létrehozott mappában van, ahol az összes, weblapunkkal kapcsolatos fájl is, akkor elég megadni a képfájl nevét és kiterjesztését, különben a teljes elérési útvonalat szükséges megadni. A böngészők által leggyakrabban és legkönnyebben értelmezhető képformátumok a JPEG, GIF és PNG. Az *img* tagen belül megadhatunk további jellemzőket is a képre vonatkozóan, amelyek segítik a formázást, de ha komolyabb, összetettebb formázási műveleteket szeretnénk elvégezni, akkor azt már CSS-ben ajánlott továbbfejleszteni.

Nézzük és értelmezzük a következő kódrészletet:

img – a kép megjelenítésére szolgáló tag
src="encinitas.jpg" – az „encinitas.jpg” nevű képet illesztjük a honlapra (az objektum elérési útvonala itt is beállítható), ami
width="150px" – 150 pixel széles legyen és
height="100px" – 100 pixel magas.
alt="Encinitas beach" – Képbetöltési hiba esetén ezt a feliratot szeretnénk látni, és
title="Encinitas" – a beszúrt képre állva az egérrel, ezt a képcímet fogjuk látni.

Végül a képernyőn megjelenő eredmény az adott kódrészletet felhasználva:



5. ábra Kép beszúrása

5.7. Multimédia (video, hang)

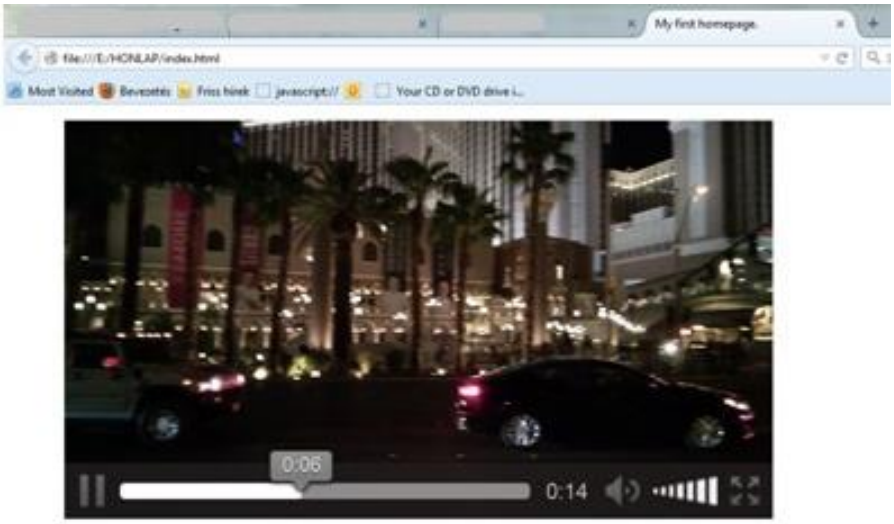
Képek weblapra történő beillesztése után nézzük meg a video és a hang multimédiás elemek beépülési lehetőségeit!

Videónk és hangfájlunk honlapra történő beillesztését HTML5-ben a video és audio tagekkel valósíthatjuk meg. A mozgókép- és hangfájlok esetében azok tartalmát, úgynevezett codec kódolja és dekódolja (tömöríti). Különböző böngészők különbözőképpen támogatják ezeket a rendszereket, sőt a mobil és asztali eszközök rendszerei között is nagy különbségek vannak. Emiatt a fejlesztésnél nagyon oda kell figyelni az optimalizálásra. A következő kódrészlettel .mp4 kiterjesztésű fájlt építünk be a honlapba:

<video width="400" height="200" controls preload=”metadata”>
<source src="peldavideo.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>

A *width* jellemző a szélesség, a height jellemző a magasságot jelöli a kódrészletben. A contorls preload jellemzővel automatikus vezérlőpanelt állítunk be. A *source src*-ben megadjuk a videofájl nevét, elérését, kiterjesztésével együtt. Kiterjesztések közül az *.mp4* a legelterjedtebb, ezen kívül az *.asf, .wmv, .avi* (Microsoft) vagy *.mov* (Apple) kiterjesztések is gyakoriak.

A fenti kódrészletet felhasználva, a következő honlaperedményt kapjuk Firefox böngészőben történő futtatás esetén:



6. ábra Multimédia tartalom beillesztése

Az audio tag a video tag analógiájára használható.

5.8. Hivatkozások

Hivatkozások (linkek) alatt értjük azokat a szavakat, szövegrészleteket, esetleg képeket, objektumokat a honlapon, amelyekre kattintva másik honlapra vagy szövegrészre jutunk el. Tulajdonképpen a hivatkozások segítségével járkálhatunk a világhálón egyfajta, sajátos logikát követve. Alapvetően kétféle hivatkozást különböztetünk meg; belső és külső hivatkozásokat. Hivatkozhatunk az aktuális honlapon belül maradva, azaz egy adott szövegrészre kattintva ugorhatunk egy következő bekezdésre, képre vagy akármilyen objektumra. Külső hivatkozásról akkor beszélünk, ha egy másik weboldalra szeretnénk átugrani. Az a (a = anchor = horgony) tag az aktuális oldalon belül másik helyre, más webhelyekre mutató hivatkozásokat rögzít („horgonyoz") a weblapra. Kötelező jellemzője a href, melynek értéke a hivatkozás célpontja. Általános alakja:

......

Oldalon belüli hivatkozási célpontjaként kötelező jellemzője egy id azonosító, melynek értéke az azonosító neve. Általános alakja:

>.

Példák hivatkozás kódolására:

```
<p>San Diegoról itt minden fontos információ megtalálható:
        <a href=http://www.sandiego.com/>San Diego</a>
</p>
<p>Kedvenc amerikai receptek:</p>
        <ul><li><a href="palacsinta.html">Áfonyás amerikai palacsinta juharsziruppal
           </a></li>
              <li><a href="cookie.html">Áfonyás cookie csokoládé darabokkal
                   </a></li>
```


Eredménye a képernyőn:

7. ábra Hivatkozások létrehozása

5.9. Űrlapok készítése

Bizonyára mindenki látott, használt, kitöltött már hagyományos űrlapot. Ebben az esetben evidensnek mondhatjuk, hogy az űrlapokkal történő információkezelés kétirányú, vagyis a kitöltő és a webhely között kommunikáció jön létre. Mivel egy űrlap elsősorban nem az adatok megjelenítésére szolgál, hanem azok bekérésére, felmerül a kérdés, hogy hová jutnak a felhasználó által bevitt adatok. Nos, egyszerűen fogalmazva, a bevitt adatokat a böngésző juttatja el egy szerverre. Arra a kérdésre válaszolni, hogy az adott szerveren mi történik, egy későbbi fejezetben térünk ki, hiszen az eljuttatott adatok feldolgozása a webszerveren futó alkalmazás feladata lesz. Ebben a fejezetben néhány fontosabb, egy űrlap kialakításához szükséges elem bemutatása következik.

A következő táblázat összefoglalja a HTML5-ben használatos fontosabb tageket, amelyekkel űrlap szerkeszthető:

1. táblázat HTML tagek

Tag neve	Funkciója
<form>	definiálja az űrlapot
<input>	adatbeviteli mező létrehozása
<label>	felirat készítése az adatbeviteli mezőhöz
<fieldset>	elemek, mezők csoportosítása
<legend>	mezőcsoportok elnevezése
<select>	választható elemek definiálása
<option>	értékek megadása
<textarea>	többsoros szövegmező létrehozása
<button>	tetszőleges nyomógomb létrehozása
<datalist>	legördülő, választó lista létrehozása
<optgroup>	listaelemek csoportosítása

Forrás: Saját szerkesztés

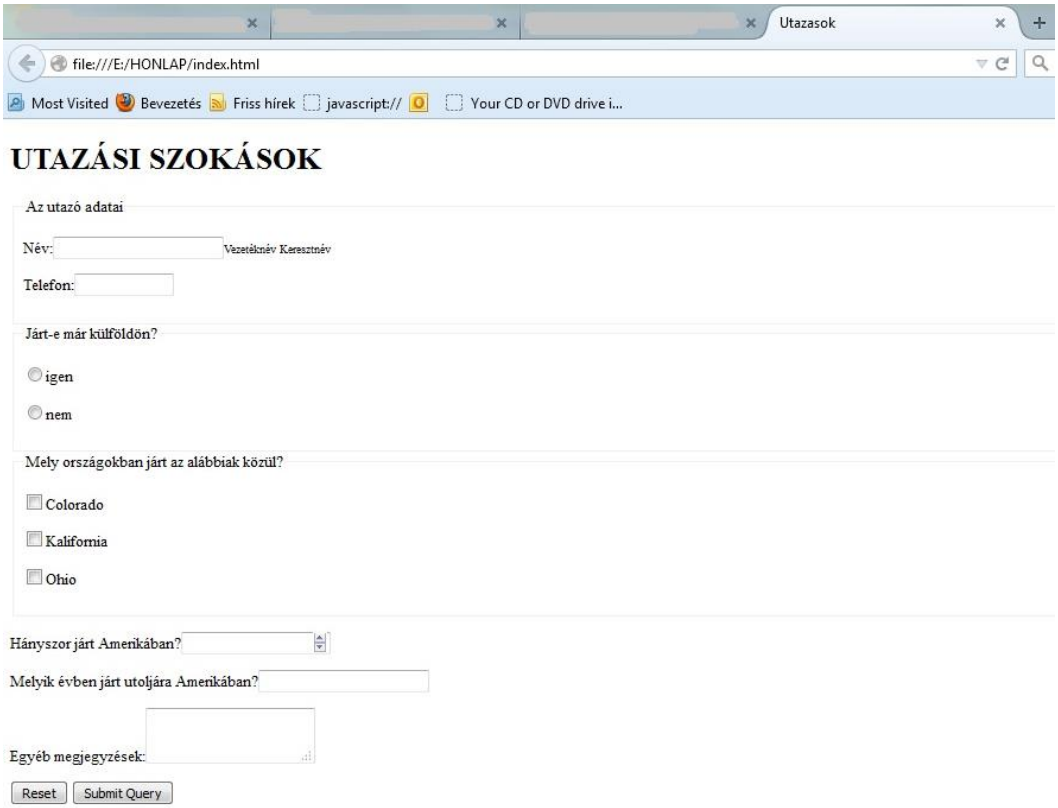
A következő példakódban értelmezzük a különböző tageket és jellemzőiket.

```
<!doctype html>
        <html lang="hu">
        <head>
                <meta charset="utf-8">
                <title>Utazasok</title>
                <style></style>
        </head>
<body>
        <form name="utazas"><h1>UTAZÁSI SZOKÁSOK</h1>
        <fieldset><legend>Az utazó adatai</legend>
                <p><label>Név:<input type="text" name="nev"><small>Vezetéknév
                Keresztnév</small></label></p>
                <p><label>Telefon:<input type="tel" name="telefon"
                size="10"></label></p>
        </fieldset>
        <fieldset><legend>Járt-e már külföldön?</legend>
                <p><label><input type="radio" name="eldontes"
        value="igen">igen</label></p>
                <p><label><input type="radio" name="eldontes"
        value="nem">nem</label></p>
        </fieldset>
        <fieldset><legend>Mely országokban járt az alábbiak közül?</legend>
                <p><label><input type="checkbox" name="orsz1"
        value="col">Colorado</label></p>
                <p><label><input type="checkbox" name="orsz2"
        value="kalif">Kalifornia</label></p>
                <p><label><input type="checkbox" name="orsz3"
        value="ohi">Ohio</label></p>
        </fieldset>
                <p><label>Hányszor járt Amerikában?<input type="number"
                name="amerika"></label></p>
                <p><label>Melyik évben járt utoljára Amerikában?<input type="time"
                name="ev"></label></p>
                <p><label>Egyéb megjegyzések:<textarea
        name="egyeb"></textarea></label></p>
                <input type="reset" name="visszaallitas">
                <input type="submit" name="kuldes">
        </form>
</body>
</html>
```

Egy egyszerű kérdőívet kódoltunk, melyben utazási szokásokat mérünk fel, és stíluslap nélkül, alaptulajdonságokat alkalmaztunk. Miután a fejlécben beállítottuk a *title* címet, a weblap törzsében először definiáltuk a kérdőívünket a *form name* taggel és

jellemzőjével. A fieldset segítségével három elhatárolt kérdéscsoportot hoztunk létre, sorra megkérdezve az Az *utazó adatai-t, Járt-e már külföldön?* és *Mely országokban járt az alábbiak közül?*. Mindezeknek a kérdéscsoportnak az elnevezéseit a *legend* taggel írattuk ki a felhasználó számára. Ezek után jönnek az adatbekérések, amelyekhez az *input* taget használtuk fel. Az adatbeviteli mezőkhöz a *label* taggel írattuk ki az elnevezéseket (pl. név, telefon), majd a bekért adattípus (*type*), a név (*name*) és a karakterhossz (*size*) értékeinek a megadása következett. A második kérdéscsoportban eldöntendő kérdésre kérjük a választ, melyhez legcélszerűbb a rádiógombot, vagy más szóval, választógombot használni. A harmadik kérdéscsoportban szintén választógombot (*checkbox*) használunk, ami annyiban tér el az előzőtől, hogy ebben az esetben megengedjük többféle válasz adását is az eldöntés (pl. igen/nem) helyett. A *Hányszor járt Amerikában?* kérdésre egy számot várunk válaszul, melyet beállító nyilak segítségével tud a kitöltő beírni, így a beviteli mező típusát szám típusra, azaz *number*re állítottuk. Ennek analógiájára használtuk a *year* típust, amikor is egy évszámra vagyunk kíváncsiak. Végül a *textarea* taggel egy hosszabb szövegmezőt definiáltunk, ahová a kitöltő 20 karakternél hosszabb szöveget is láthatóan be tud gépelni. A *reset* és *submit* beviteli értékekkel zárhatja a kitöltő a munkáját annak megfelelően, törli-e a bevitt adatokat, vagy elküldi feldolgozásra.

Így jelenik meg az általunk elkészített kérdőív a képernyőn Firefox böngésző alatt:



8. ábra Ürlapok készítése

5.10. Validálás

A HTML5 leíró nyelvvel kapcsolatos fejezet végéhez értünk. Mielőtt továbblépnénk és megismernénk, hogyan lehet dinamikussá tenni az elkészített statikus honlapunkat, érdemes ellenőrizni, hogy az általunk elkészített HTML-kód helyes-e. Ez azért is fontos, mert a leíró nyelvek fejlődése során sokszor adódhatnak a szabványból kivett, illetve betett részek, amelyekkel nekünk mint hétköznapi felhasználóknak, nagyon résen kell lennünk, ha haladni akarunk a korral és mindig a legmodernebb, legfejlettebb technológiákat akarjuk használni munkánk során.

Jól jöhet továbbá egy olyan logikai, szerkezeti ellenőrzés, amely kiszűri az általunk elkövetett hibákat (ilyen hiba lehet például a tagek hibás egymásba ágyazása), hogy azokat nyilvánosságra hozatal előtt kijavíthassuk.

Azt a folyamatot, melynek során belátjuk, hogy kódunk megfelel az érvényben lévő szabványnak, validálásnak nevezzük. Ehhez a W3C konzorcium létrehozott ingyenes és online használható szoftvereket, melyekkel könnyen ellenőrizhetjük fájljainkat. A W3C konzorcium hivatalos validátor eszközét a http://validator.w3.org/ címen találjuk, melyet háromféle módon is használhatunk. Megadhatjuk az általunk elkészített honlap URL-jét, feltölthetjük az állományainkat, de akár vágólap segítségével is beilleszthetjük a kódot ellenőrzésre.

Fontos: Mindig végezzük el a validálást, mielőtt aktiváljuk az általunk létrehozott munkát!

6. Bevezetés a JavaScript programozási nyelvbe

Az előző fejezetben betekintettünk a statikus weboldalak elkészítésébe. Mint a címből kiderül, ennek a fejezetnek is az alapozó tudás megszerzés a célja. Olyan áttekintést szeretnék nyújtani, amelynek segítségével a tanulók webprogramozási készségüket tudják elmélyíteni, és egyfajta inspirációt alakíthatnak ki magukban további programozási munkákhoz. Miután az eddigiek alapján statikus honlapot már létre tudunk hozni, most nézzük, mi a helyzet a honlapok dinamikusságával. Dinamikus alatt értendő, hogy a honlap interaktív is egyben. A honlap célja nemcsak a megjelenítés, hanem felhasználói oldalról is fogad adatokat, információkat, amelyeket képes feldolgozni és reagálni is tud azokra.

Erre a feladatra alkalmas a HTML-hez hasonlóan kliens oldali programozási nyelv, a JavaScript. Ez egy olyan parancsnyelv, amelyet beépítve a HTML-kódba, interaktívvá tehetjük honlapunkat. Ebben a fejezetben az elméleti ismereteken túl megpróbálok néhány gyakorlati példát is bemutatni, amelyek segítségével megszerezhetik a hallgatók az ide kapcsolódó alaptudást.

6.1. A nyelv szerepe

Az első legfontosabb dolog, amit a JavaScripttel kapcsolatosan tudnunk kell, hogy az ezen a nyelven írt programot hogyan tudjuk futtatni. Erre a kérdésre egyszerű a válasz. Olyan böngészőre van szükség a futtatáshoz, amely képes kezelni a JavaScriptet. A ma használatban lévő böngészőkre bátran állítható ez a kezelési képesség, így elég azt mondanunk, hogy egy internetböngészőre van szükség ahhoz, hogy futtatni tudjuk a JavaScriptben megírt kódunkat.

Ezek után fontos tudni, hogy a JavaScript-kód HTML-oldalba épül be, ahogy azt a következő példában láthatjuk:

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript1.1">
    document.write("Ez egy JavaScriptben írt mondat.”)
</SCRIPT>
</BODY>
</HTML>
```

A JavaScript document.write() parancsával írattuk ki az első alapmondatunkat. Látható, hogy a HTML <SCRIPT> elemét használtuk arra, hogy JavaScript-utasításokat ágyazzunk a HTML-dokumentumunkba. Minden, amit a <SCRIPT> és a </SCRIPT> elemek

közé teszünk, JavaScript-utasításokként értékelődnek ki. A LANGUAGE attribútum segítségével a JavaScript verziószámát határozhatjuk meg.

Fontos: A JavaScript nem védhető meg a „kíváncsiskodó szemek” elől, hiszen a HTML-dokumentumok forrása megtekinthető a legtöbb böngésző segítségével, a JavaScript-utasítások pedig ennek közvetlen részét képezik.

A programoknak gyakran kell reagálniuk olyan eseményekre, mint egy billentyű lenyomása, vagy például az egér mozgatása. Az események és eseménykezelők a JavaScript programozásban is nagyon fontosak. Az ilyen események kezeléséhez különféle függvények, eseménykezelők léteznek. Nézzük meg a következőkben egyszerű példákon keresztül, hogy mit is értünk esemény alatt honlapok létrehozásánál.

6.2. Események

A következő példában azt akarjuk elérni, hogy a JavaScript programunk reagáljon valamilyen eseményre, pontosabban létrehozunk egy gombot, amire kattintva egy ablak jelenik meg a képernyőn. Ez valójában azt jelenti, hogy az ablaknak egy kattintás-esemény (click-event) hatására kellene megjelenni. Az itt használt eseménykezelő neve: *onClick*.

```
<form>
<input type="button" value="Kattints rám!" onClick="alert('Szia!')">
</form>
```

Egy gombot (*button*) hoztunk létre, amelyre kattintva megjelenik egy ablak a *Szia* felirattal. Az újdonság HTML-hez képest az *onClick="alert('Szia!')"* rész, ami az *<INPUT> HTML* elem belsejében található, és ez gyakorlatilag a JavaScript elem. Ez mondja meg, hogy mi történjen akkor, ha a felhasználó megnyomja a gombot. Ha ez bekövetkezik, akkor valójában egy *onClick* esemény generálódik, és ez végrehajtja az *alert('Szia!)* JavaScript-kódot. Ez egy függvény, amely létrehoz egy üzenetablakot, melyben a zárójelen belül aposztrófok közé írt szöveget jeleníti meg.

JavaScriptben mindkét idézőjelet lehet használni, csak arra kell ügyelni, hogy ez a számítógép számára is egyértelmű legyen. Az alábbi idézőjel-használat is helyes lett volna: *onClick='alert("Szia!")'.*

Az eseménykezelés általános formája a *<HTML_elem eseménykezelő_neve="JavaScript_utasítás">*, amely szintaxis még sok más eseménykezelőre is jellemző. Ezek közül a következő táblázat foglalja össze a fontosabb eseményeket:

2. táblázat Események JavaScriptben

Események	*Mikor lehet alkalmazni?*	*Az eseménykezelő neve*
abort	A felhasználó megszakítja a kép betöltését	onAbort
blur	Az egérrel az aktuális mezőn kívülre kattintunk	onBlur

change	Megváltoztatunk egy űrlapbeli értéket	onChange
click	Az űrlap valamely elemére, vagy egy csatolásra (link) kattintunk	onClick
error	Ha kép vagy dokumentum betöltésekor hiba lép fel	onError
focus	Kijelöljük az űrlap egy elemét	onFocus
load	A böngésző új lapot tölt be	onLoad
mouseout	Az egérmutató elhagyja a linket	onMouseOut
mouseover	Az egérmutató valamely link felett tartózkodik	onMouseOver
reset	Űrlap törlésekor	onReset
select	Új mezőt jelölünk ki	onSelect
submit	Űrlap elküldésekor	onSubmit
unload	Ha másik HTML-oldalra lépünk	onLoad

Forrás: Saját szerkesztés

6.3. Változók

A változó olyan tároló, amelybe adat(ok)at helyezünk, melyeknek meg tudjuk változtatni, le tudjuk kérdezni az értékét, és nevével tudunk hivatkozni rá. A változó nevek kis-nagybetű-érzékenyek, és betűvel vagy aláhúzás (_) karakterrel kezdődnek. A változó néven belüli karakterei számok is lehetnek.
Fontos: Törekedjünk a rövid, de kifejező nevek használatára, pl. nev.

Ahhoz, hogy változókat tudjunk használni, deklarálnunk kell azokat a var kulcsszó segítségével, amelyet megtehetünk kezdőérték-adással és nélküle is.
Pl.

```
var a;
var nev="San Diego";
```

Két fontos definíciót kell itt megjegyezni; a lokális és globális változók definícióit. Lokális változó esetén annak hatóköre csak azon az adott függvényen belül esedékes, ahol deklaráltuk, a függvényen kívül megszűnik a funkciója. A globális változót a függvényeken kívül definiáljuk, így azok az egész program során élnek és működésük csak a program vagy a script lefutása után szűnik meg.

Léteznek konstans változók is, amelyekről azt kell tudni, hogy értékük fix. pl. pi=3.14.

A változók típusát a JavaScriptben nem kell megadnunk, a nyelv értelmezi az első értékadásból.

A változók csoportjába tartoznak a literálok is, amelyeket értékadáshoz használunk. Ezek lehetnek számok, tömbök, sztringek és objektumok. Például:

```
Számok:
    var   a=0;
          b=74;
          pi=3.14;
          no=true;
Tömb:
    var   szam=[4,6,8,..,12]
Sztring
    var   szoveg="Los Angeles"
```

Objektum

var hely={orszag:"Kalifornia", varos:"Los Angeles", minosites:10}.

6.4. Kifejezések és operátorok

Miután változókkal próbálunk dolgozni, ehhez tudnunk kell, milyen műveleteket, vezérléseket tudunk végezni velük. Erre jók az operátorok, amelyek segítségével kifejezéseket fogalmazunk meg JavaScript nyelven, és ez által műveleteket végzünk velük, és eredményértékekkel állhatunk elő. Összesen hatféle operátorról beszélhetünk: aritmetikai, értékadó, összehasonlító, logikai, sztring és feltételes operátorokról. A következő táblában láthatók az értelmezéseik:

3. táblázat Kifejezések és operátorok

operátor	*név*	*példa*	*eredmény*
Aritmetikai operátorok			
+	összeadás	a=2, b=3 c=a+b	c=5
-	kivonás	a=5, b=2 c=a-b	c=3
*	szorzás	a=4, b=3 c=a*b	c=12
/	osztás	a=5, b=2 c=a/b	c=2.5
%	maradékképzés	a=5, b=2 c=a%2	c=1
++	növelés 1-gyel	a=3, a++	a=4
--	csökkentés 1-gyel	a=3, a--	a=2
Értékadó operátorok (a=5, b=2 esetén)			
=	a legyen egyenlő b-vel	a=b	a=2
+=	a növelése b-vel	a+=b	a=7
-=	a csökkentése b-vel	a-=b	a=3
*=	a szorzás b-vel	a*=b	a=10
/=	a osztása b-vel	a/=b	a=2.5
%=	a maradékképzése b-vel	a%=b	a=1
Összehasonlító operátorok			
==	egyenlő	4==3	false
===	egyenlő típus és érték	a=4, b="3" a==b, a===b,	 true false
!=	nem egyenlő	4!=3	true
>	nagyobb, mint	4>3	true
<	kisebb, mint	4<3	false
>=	nagyobb vagy egyenlő	4>=3	true
<=	kisebb vagy egyenlő	4<=3	false
Logikai operátorok			
&&	és	a=4, b=3 5>a && a>b	 true
||	vagy	a=6 || b=4	false
!	nem	!(a==b)	true
Sztring operátor			
+	s1="Los" s2=" Angeles"	s3=s1+s2	s3="Los Angeles"

Feltételes operátor		
(feltétel) ? kifejezeshaigaz : kifejezeshahamis	var a=kor var b=(kor<70?”fiatal”:”idős”)	ha a=40 b=”fiatal” ha a=80 b=”idős”

Forrás: Saját szerkesztés

6.5. Vezérlési szerkezetek

Programozásban vezérlési szerkezet alatt feltételhez kötött, akár többszörösen elvégzett műveletsorozatot értünk. Mint a legtöbb programozási nyelvben, JavaScriptben is alapvetően két csoportba sorolhatók ezek a szerkezetek: az elágazások és ciklusok körébe. Az elágazások használata akkor indokolt, ha különböző esetek vannak és meg kell őket különböztetni, ciklust pedig akkor használunk, ha egy kódot vagy kódsorozatot többször végre akarunk hajtani. Vegyük sorba a lehetőségeket! Háromféle elágazásra, kétféle ciklusra vizsgáljuk meg a példákat!

6.5.1. Elágazások

if utasítás

```
if (feltétel) {
utasítás
}
```

példa:

```
var kor=40
if (kor<70) {
document.write("Fiatal")
}
```

if-else utasítás

```
if (feltétel) {utasítás ha igaz}
else {utasítás ha hamis}
```

példa:

```
var kor=40
if (kor<70) {
document.write("Fiatal")
}
else {
document.write("Idős")
}
```

switch utasítás

```
switch(n)
        {
        case 1:
```

```
        utasítás 1
    break
    case 2:
        utasítás 2
    break
    default:
        utasítás
    }
```

példa:

```
switch(navigator.language)
    {
    case 'en':
        nyelv='angol'
    break
    case 'hu':
        nyelv='magyar'
    break
    case 'de':
        nyelv='német'
    break
    default:
    nyelv='ismeretlen'
    }
 document.write('Ön ',nyelv,' nyelvű böngészőt használ')
```

6.5.2. Ciklusok

for ciklus

A for ciklust akkor alkalmazzuk, ha a ciklus elkezdése előtt már tudjuk, hogy hányszor kell majd a ciklusnak lefutnia. Általános szintaxis:

```
for (változó=kezdőérték; változó <= végérték; változó++) {
ciklusmag
}
```

példa:

```
for(i = 1; i <= 20; i++)  {
var x = i * i;
document.write( i + "négyzete = " + x + <br>);
}
```

for–in ciklus

A for ciklust gyakran használjuk arra, ha például a tömb minden elemével akarunk csinálni valamit. Ilyen esetekben jó a for–in ciklus. Szintaxis:

```
for (változó in objektum) {
```

ciklusmag
}

példa:

```
var cities = new Array()
var x
cities[0] = "Los Angeles"
cities[1] = "Denver"
cities[2] = "New York"
        for (x in cities) {
                document.write(cities[x] + "<br>")
                }
```

while és do while ciklusok

Ezeket az elöl- és hátultesztelő ciklusokat akkor alkalmazzuk, 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.

while (feltétel) {
cikusmag
}
A do while ciklus annyiban különbözik, hogy mindenképpen lefut egyszer a ciklusmag, és csak utána értékelődik ki a feltétel.

do {
cikusmag
} while (feltétel);

példa:

1.
```
var x,i
while (i<=12){
x=i*i;
document.write( i  + " négyzete " + x + <br>);
++i;
}
```
2.
```
var x = 12;
do {
document.write("<br>x * x = " + (x * x));
x = x + 1;
}
while(x < 10);
```

6.6. Függvények

A JavaScript fontos nyelvi eleme a függvény (function). A nyelv sok függvényt tartalmaz, például az előzőekben használt alert() függvény is ilyen. Függvényeknek adhatunk át paramétereket, amelyekkel dolgozni fognak (az alert() függvény paramétere az idézőjelek között átadott szöveg volt, amit a függvény egy üzenetablakban jelenített meg). Mi magunk is írhatunk függvényeket, hogy a nagyobb, összetettebb feladatokat kisebb, jobban kezelhető részekre bontsuk. Ezek a függvények szintén átvehetnek paramétereket, sőt értéket is adhatnak vissza, csakúgy, mint más programozási nyelvekben.

Példa két függvényre:

```
<SCRIPT LANGUAGE="JavaScript">
function osszead1(a,b){
        var osszesen=a+y;
        document.write("<H2>"+a+"+"+b+"="+osszesen+"</H2><BR>";
    }
    function osszead2(a,b){
        return a+b;
    }
    osszead1(15,40);
    document.write("<H2>11+4="+osszead2(11,4)+"<H2>");
</SCRIPT>
```

A fenti példában két függvényt hoztunk létre *osszead1* és *osszead2* néven. A JavaScriptben írt függvények elé be kell írni a *function* kulcsszót, viszont ellentétben pl. a C++ vagy a Java függvényeivel, nem kell a visszatérési értéknek, valamint a paramétereknek a típusát megadni. Továbbá tudnunk kell, hogy a JavaScript a kis- és nagybetűk között különbséget tesz.

Az *osszead1* függvény példáján keresztül annak működéséről azt kell tudni, hogy a *function* kulcsszót közvetlenül a függvény neve követi, majd zárójelekben a paramétereket adjuk meg. Ennek a függvénynek két paramétere van, a és b (amelyek típusát nem adtuk meg). A függvény törzse kapcsos zárójelek között van, és ide helyezzük el a függvényhez tartozó utasításokat.

A függvényben három változót is használunk (*a, b és osszesen*). A JavaScript programok változókban tárolják az információkat, adatokat. A példánál maradva, *a* és *b* tárolja el az összeadandó számok értékeit, és az *osszesen* nevű változó tárolja el az összeget.

Változót a *var* kulcsszóval, majd a változó nevének megadásával deklarálhatunk. Deklaráláskor az értékadó operátor (=) segítségével kezdőértéket is rendelhetünk a változókhoz. A példában, ha *a* értéke 15, *b* értéke 40, akkor az *osszesen* változó értéke *a* és *b* összege, azaz 55 lesz, és a kiírandó karakterlánc a következő:

15+40 = 55.

A függvényeket a nevükkel hívjuk meg, és híváskor adjuk át a paraméterek aktuális értékét. Az *osszead2* függvény egy lényeges dologban különbözik az *osszead1* függvénytől. Rendelkezik visszatérési értékkel (*return a+b*). Az *osszead2* függvényt a 11 és 4 paraméterekkel hívtuk meg, amely kiszámolja azok összegét és visszatér ezzel az értékkel. Ez az érték behelyettesítődik a hívás helyére, így a *document.write* függvény argumentuma a következő:

11+4 = 15.

6.7. Nyelvi elemek

Az eddigi példákban mint függvényt emlegettük a *document.xxxx* (pl. *document.write*) elemeket. A JavaScript lehetőséget kínál arra, hogy a böngészőnk bármely objektumát kezeljük és saját elképzeléseink szerint programozzuk. A *document* objektumot nagyon gyakran használjuk, így ennek megértésére, megismerésére szeretnék kitérni. Ezen objektum alatt a böngészőnk éppen aktuális oldalát értjük, és ha ezt hívjuk meg, akkor az aktuális oldalon végzi el a kódban szereplő utasításokat. Az aktuális oldalon szerepelhetnek linkek, képek, szövegek, táblák stb., melyek jellemzőinek beállítása történhet a következő utasítás kiadásával:
document.mezonev=#xxxxxx. A *mezonev* nem más, mint az oldalon lévő bármelyik objektum, melynek értéket adva (pl. színkódot) állíthatjuk be a tulajdonságát.
Például a

```
function ocean(){
 document.bgColor="#4F5FBF";
}
```

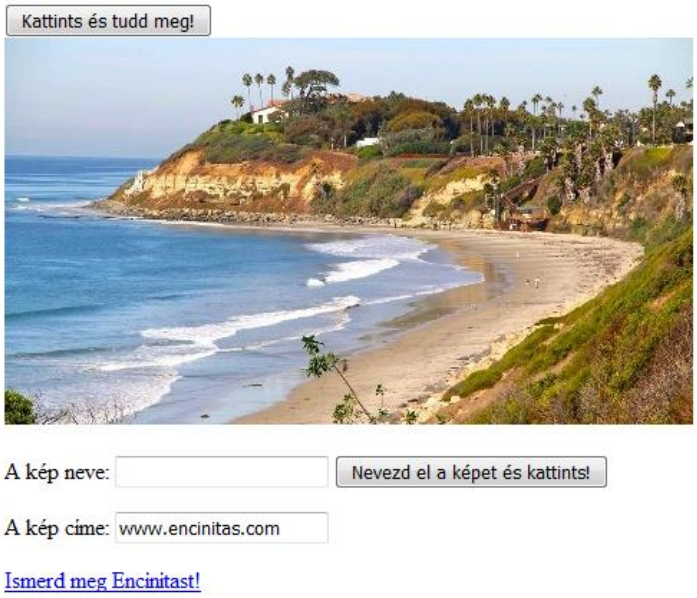
függvénnyel az oldal hátterét állítjuk be kék színűre. A *bgColor* jelenti a háttérszín-mezőt, a *#4F5FBF* a beállítandó RGB szín kódját. Léteznek más, gyakran használatos színbeállításokkal kapcsolatos mezők is, mint pl. *linkColor* (a linkek színének beállításához), *fgColor* (előtér színének beállításához) vagy akár a *vlinkColor* (a már meglátogatott link színének beállításához).
A színeken kívül természetesen vannak más fontos nyelvi elemek is, amelyeket a JavaScripttel kezelünk. Egy adott honlap minden mezőjére, objektumára hivatkozhatunk, melynek általános formája a következő:
képre hivatkozás: *document.images[..]*
szövegmezőre, űrlapgombra: *document.forms[..].elements[..]*
linkre: *document.links[..].*
A szögletes zárójelen belül *[..]* a lapon előforduló elem sorszámát kell megadni, amelyben az elsőt a 0. jelöli.

Példa az eddigiekre:

```
<!doctype html>
<html lang="hu">
```

```
	<head>
		<meta charset="utf-8">
<SCRIPT LANGUAGE="JavaScript">
	function Elnevez(){
	var name = document.forms[0].elements[0].value;
	alert(" A kép neve:"+name);
	}
</SCRIPT>
	</head>
<body bgColor=#FFFFFF>
	<input type="button" value="Kattints és tudd meg!" name="gomb"
onClick="alert('hogy San Diego híres strandja Encinitas!')"><br>
	<img src="encinitas1.jpg" name="encinitas1"><p>
	<form name="objektumpelda">
		A kép neve:
		<input TYPE="text" name="nev">
		<input TYPE="button" value="Nevezd el a képet és kattints!"
onClick="Elnevez()"><p>
		A kép címe:
		<input type="text" name="cim" value="www.encinitas.com"><br>
	</form><p>
	<a href="http://www.encinitas.com">Ismerd meg Encinitast! </a>
</body>
</html>
```

Megjelenése a böngészőben:



9. ábra Nyelvi elemek

A fenti példában tehát jól látható, hogy az *Elnevez()* függvényben a var *name = document.forms[0].elements[0].value;*hivatkozással adjuk meg a kép nevét és az *alert(" A kép neve:"+name);* függvénnyel íratjuk ki egy üzenetablakba.

Joggal merül fel viszont egy következő probléma, miszerint, ha komolyabb, összetettebb honlapot készítünk, hogyan tudjuk precízen követni az egyes objektumok sorszámait, melyeket azok hivatkozásánál kell megadnunk a szögletes zárójelekben *([..]).* Erre az a megoldás, hogy az egyes objektumoknak nevet kell adnunk és az objektumok neveivel tudunk hivatkozni kódon belül. Így, ha az űrlapunknak az objektumpelda nevet, az első szövegmezőnek a *nev* nevet adjuk, akkor az objektumra *document.forms[0].elements[0]* helyett használhatjuk a *document.objektumpelda.nev* hivatkozást is.

6.8. Keretek és ablakok

A keretek és ablakok létrehozása, kezelése HTML-ben is megvalósítható, de JavaScriptben elmondható, hogy kibővültek ezeknek a lehetőségei is. A keretek gyakorlatilag felosztják a böngésző ablakát, és mindegyik rendelkezik olyan tulajdonságokkal, miszerint meg tud jeleníteni egy önálló HTML-dokumentumot, rendelkezhet névvel, amelynek segítségével hivatkozhatunk rá, továbbá automatikusan méretezi önmagát az ablak méretének változása során, valamint engedélyezhetjük vagy megtilthatjuk, hogy a felhasználó saját maga változtathassa a keretek méretét.

Kereteket a <*FRAMESET*> HTML-elemmel készíthetünk, amelyen belül a ROWS attribútumot használjuk *(<FRAMESET ROWS="50%">*). Ez azt jelenti, hogy a kereteink sorokban fognak elhelyezkedni, és a ROWS attribútumot követően százalékos arányt kell megadni. Ha például "50%"-ot használunk, az első keret a böngésző ablakának felét fogja elfoglalni, míg a másik keret a fennmaradó részt. Ha nem százalékosan, hanem pixelekben szeretnénk megadni a keretek méretét, akkor ezt is megtehetjük a százalékjel elhagyásával. Sorok helyett oszlopokba is rendezhetjük a kereteket, ekkor a ROWS attribútum helyett COLS attribútumot használjunk, vagy akár a kettőt kombinálhatjuk is.

JavaScript esetében úgy kell elképzelni, hogy a keretek egyfajta logikus hierarchiába rendeződnek. A böngésző ablaka van a hierarchia legmagasabb fokán, és mint egy gráfban, a szülő (parent) szerepet tölti be. Ha ezt az ablakot bontjuk kétfelé, akkor a két létrejött keretet tekinthetjük a gyerekeknek, amelyek a szülővel hozzáférnek egymáshoz.

Ha a szülő ablakhoz tartozó HTML-kódban, amely a keretet létrehozta, van egy JavaScript-kódunk, amely a keretekben szeretne bizonyos műveleteket elvégezni, például megjeleníteni valamit, akkor a keret nevének felhasználásával férhetünk hozzá a kerethez.

Példa.

A következő HTML-kód esetén

```
<html>
<frameset rows="50%,*">
        <frame src="oldal1.htm" name="keret1">
        <frmae src="oldal2.htm" name="keret2">
```

```
</frameset>
</html>
```

a *keret2* oldalra a következő utasítással tudunk kiíratni bármit:

```
keret2.document.write("San Diego és Los Angeles csodás városok!");
```

A szülő ablakhoz parent névvel férhetünk hozzá egy gyerek ablakból (keretből). Új dokumentum betöltéséhez a szülő ablak *location.href* mezőjéhez kell új értéket rendelni. Ezt a következő utasítással tehetjük meg:

```
parent.location.href = "http://...";
```

Valamint, ha egyik keretből szeretnénk megjelentetni valamit a másik keretben, és azt tudjuk, hogy a két gyerek ablak közvetlenül nem fér egymáshoz, így a szülő ablakon keresztül érhetik el egymást a következő utasítással:

```
parent.keret2.document.write("A másik keretbe is kiírathatjuk, hogy San Diego és Los Angeles csodás városok! ");
```

Arról, hogy mennyire számít modernnek a keretek használata, igencsak megoszlanak a vélemények. Az ablakok használata viszont egyértelmű, követve a trendet, mert funkciója klasszikus eszközzé tette, így fejlesztőként, ha JavaScriptből szeretnénk új ablakokat kezelni, létrehozni, „a szükség divatot bont” alapján megoldható a kérdés. A JavaScript segítségével megnyithatunk új böngésző ablakokat, amelyekbe betölthetünk új HTML-oldalakat, de akár mi magunk is létrehozhatunk dokumentumokat az új ablakban.
A következő példa jól mutatja a technikai megvalósítását egy új ablak megnyitásának.

```
<html>
<head>
<script language="JavaScript">
function ujablak(){
        ablak = open("pelda.htm");
}
</script>
</head>
<body>
<form>
<input type="button" value="Új ablak nyitása" onClick="ujablak()">
</form>
</body>
</html>
```

JavaScriptben lehetőségünk van arra is, hogy az új, megnyitandó ablakot szabályozzuk. Például megadhatjuk az új ablak pontos méreteit. Ennek megvalósításához egy példa függvény így néz ki:

```
function ujablak2(){
        ablak = open("pelda.htm", "uj_ablak",
                "width=300,height=200,status=no,menubar=no");
}
```

JavaScript segítségével be is zárhatjuk az ablakunkat. Ehhez a close() függvényt kell használnunk.

```
<html>
<head>
<script language="JavaScript">
        function bezar(){
                close();
        }
</script>
</head>
        <body>
                <form>
                <input type="button" value="Zárj be!" onClick="bezar()">
                </form>
        </body>
</html>
```

6.9. A JavaScript objektumai

Mivel a JavaScript objektumorientált nyelv, nem hagyhatjuk figyelmen kívül az objektumokat még akkor sem, ha csak kezdő programozók vagyunk. Van néhány fontos dolog, amit már a kezdeteknél érdemes megtanulni, és fokozatosan elmélyülni a részletekben. Egy JavaScript-kód futtatásakor a böngésző sok beépített objektumot bocsát rendelkezésre, melyek segítségével honlapunk tulajdonságait és jellemzőit vagy beállíthatjuk vagy módosíthatjuk azokat.

A JavaScript több, előre definiált objektumot tartalmaz. Ezek az Array (tömb), a Boolean, Date (dátum), Function (függvény), Math (matematikai), Number (szám) és a String (karakterlánc). Minden objektumra jellemző, hogy tulajdonságaik és metódusaik vannak, valamint, hogy a következőkkel rendelkeznek:

- a *name* a HTML-ben megadott neve az objektumnak és
- a toString([..]) szöveggé konvertálja az objektumot. A szögletes zárójelbe számrendszer alapszámát kell írni, ahol szükséges.

Objektum létrehozása történhet többféleképpen is. Egy példa objektum inicializására:

var orszag={fovaros: ”Los Angeles”, nepesseg:3819702}

Egy objektum tulajdonságokból áll. Egy tulajdonságot a neve határoz meg, és tartozik hozzá egy érték, illetve attribútumok. Az értéke lehet egy objektum, primitív érték vagy metódus (függvény objektum). Az attribútumok belső célokat szolgálnak, közvetlenül nem érhetjük el, nem állíthatjuk őket; négyféle van belőlük:
ReadOnly: a tulajdonság csak olvasható, bármilyen írási kísérlet hatás nélkül marad.
DontEnum: a tulajdonság nem felsorolható, nem jelenik meg például egy for–in ciklusban.
DontDelete: a tulajdonság nem törölhető, bármilyen törlési kísérlet hatás nélkül marad.
Internal: belső célra használt tulajdonság.

Objektumokat egyszerű függvények segítségével hozhatunk létre, melyek a *new* operátorral együtt használva konstruktorként működnek, a bennük lefutó kód hozza létre az objektum tulajdonságait (illetve azok egy részét). A függvényen belül ilyenkor a *this* változó testesíti meg a létrehozandó objektumot, mint azt az alábbi példa mutatja:

```
function tipus(tul) {
this.tulajdonsag = tul;
}
var objektum = new tipus(5);
alert(objektum.tulajdonsag);
```

Az előbb felsorolt, előre definiált objektumok közül a következőkben néhány fontosabbra hozunk példát, amelyeket érdemes áttekinteni.

6.9.1. Az Array (tömb) típus

A tömbök rendkívül fontosak a programozásban. Gyakran van szükségünk nagy mennyiségű adat kényelmes tárolására úgy, hogy bármelyikhez könnyen hozzáférjünk. A tömbök sok változóból felépülő összetett adattípusok. Az egyes adatokhoz egy név (a tömb neve) és egy szám segítségével férhetünk hozzá.

Létrehozása: *tomb = new Array()*

A *tomb* nevű új tömbünkhöz rögtön értékeket is rendelhetünk:

```
tomb[0] = "San Diego";
tomb[1] = "2015";
tomb[2] = "július";
```

A létrehozás egy másik lehetséges módja:

```
Orszagok = new Array("Kalifornia","Colorado","Új-Mexikó");
```

Az Array objektum legfontosabb metódusai a következők:
a *join()* metódus összefűzi a tömb elemeit egyetlen sztringgé *(Orszagok.join()=>”Kalifornia,Colorado,Új-Mexikó”),*

a *reverse()* megfordítja (transzponálja) a tömb elemeit, az utolsóból lesz az első, az elsőből az utolsó (*Orszagok.reverse()=> Orszagok[0] az Új-Mexikó lesz, Orszagok[1] a Colorado, míg Orszagok[2] a Kalifornia,*
a *sort()* rendezi a tömb elemeit (*Orszagok.sort()=>Orszagok[0] a Colorado, Orszagok[1] a Kalifornia, míg Orszagok[2] az Új-Mexikó lesz*).

A következő példában gomb lenyomására kiíratjuk az *Orszagok* tömb értékeit:

```
<html>
<head>
<script language="JavaScript">
function tombkiir(){
        var kiir = '';
        Orszagok = new Array('Kalifornia','Colorado', 'Új-Mexikó');
        kiir = "A tömb elemei:\n"
                for (var i=0; i<3; i++){
                kiir += i + ". " + Orszagok[i] + "\n";
                }
alert(kiir);
}
</script>
</head>
<body>
<form>
        <input type="button" value="A tömb elemeinek kiíratása" onClick="tombkiir()">
</form>
</body>
</html>
```

6.9.2. A Date (dátum) objektum

A *Date* objektummal idő- és dátumértékek kezelésére alkalmas programokat írhatunk. Az alábbi módon hozhatjuk létre a *Date* objektum új példányait:
dátum_objektum_neve = new Date([paraméterek]);
A paraméter a következő variációk bármelyike lehet:
Nem adunk meg paraméterként semmit. Ekkor az aktuális dátum- és időértékeket használjuk fel. Például: *ma = new Date();*
A paraméter egy *sztring*, ami egy dátumot reprezentál a következő formában: *"Hónap Nap, Év óra:perc:másodperc"*. Például: *nyarinap_2015 = new Date("August 08, 2015 12:30:30");*
Ha elhagyjuk az óra:perc:másodperc részt, akkor ez automatikusan a 0 értéket veszi fel.
Számokkal adjuk meg a dátumot. Például: *2015nyarinap = new Date(2015, 7, 8, 12, 30, 30);*

Fontos: A hónapok számozása nem 1-től indul, hanem 0-tól, azaz januárnak a 0, februárnak az 1 stb. felel meg.

Egy példa a dátum objektum kezelésére (egy óra és aktuális dátum kiírása):

```
<html>
<head>
<script language="JavaScript">
var idoStr, datumStr;

function ido(){
most= new Date();
ora= most.getHours();
perc= most.getMinutes();
mp= most.getSeconds();

ev= most.getYear();
honap= most.getMonth()+1;
nap= most.getDate();

idoStr= ora;
idoStr += ((perc < 10) ? ":0" : ":") + perc;
idoStr += ((mp < 10 ) ? ":0" : ":") + mp;

datumStr = ((ev < 2000) ? "19" : "") + ev;
datumStr += ((honap < 10) ? "-0" : "-") + honap;
datumStr += ((nap < 10) ? "-0" : "-") + nap;

document.digiora.ido.value = idoStr;
document.digiora.datum.value = datumStr;

setTimeout("ido()",1000);
}
</script>
</head>
<body onLoad="ido()">
<form name="digiora">
Idő: <input type="text" name="ido" size="8" value=""><br>
Dátum: <input type="text" name="datum" size="10" value=""><br>
</form>
</body>
</html>
```

6.9.3. A Math (matematikai) objektum

A matematikai objektum matematikai konstansokat és függvényeket tartalmaz. Többek között ide sorolhatók a trigonometrikus, az exponenciális és logaritmikus függvények.
Példa matematikai objektum használatára:

```
with (Math){
```

```
 kor_terulet = PI * r * r;
 x = r * sin(beta);
 c = sqrt(a*a + b*b);
}
```

Megjegyzés: sqrt() – négyzetgyök függvény

6.9.4. String objektum

A String egy karakterlánc. Ilyet legegyszerűbben idézőjeles szöveg megadásával hozhatunk létre. Általános szintaxis:

String_objektum_neve = new String(„sztring”);

Nézzünk egy konkrét példát:

szoveg = new String("Amerika");
JavaScript

Erre alkalmazva az egyes metódusokat, a következő visszatérési értékek az érvényesek:

szoveg.length = 7
szoveg.substring(3,5) = ri
szoveg.toUpperCase() = AMERIKA
szoveg.toLowerCase() = amerika
szoveg.charAt(0) = A
szoveg.charAt(4) = I
szoveg.indexOf('k') = 5
szoveg.indexOf('s') = -1
szoveg.lastIndexOf('A') = 0
szoveg.lastIndexOf('s') = -1

A HTML típusú metódusok a következőképpen alakulnak:

szoveg.link('http://www.amerika.com') = *Amerika*
szoveg.big() = Amerika
szoveg.blink() = Amerika
szoveg.bold() = **Amerika**
szoveg.fixed() = Amerika
szoveg.italics() = *Amerika*
szoveg.small() = Amerika
szoveg.strike() = ~~Amerika~~
szoveg.sub() = Amerika
szoveg.sup() = Amerika

6.10. Űrlapok

Webprogramozással kapcsolatos tanulmányaink során a HTML-ről szóló fejezetben egyszer már készítettünk űrlapot. Konkrétan egy utazási szokásokról szóló egyszerű kérdőívet szerkesztettünk, amelyet HTML-tagekkel építettünk fel. Az űrlapokat szokás szerint kitöltés után elküldjük egy szerveren lévő programnak, az adatok további feldolgozása céljából. Ezzel kapcsolatosan olyan kérdés merülhet fel, hogy ha egyszer HTML-ben el tudtuk már készíteni az űrlapot, akkor mi dolgunk lehet még JavaScriptben? Alapesetben semmi. Viszont, ha továbbgondoljuk az űrlapkészítés hatékonyságát, felléphet olyan eset, hogy a kapott adatok nem a megfelelőek számunkra.

Mert például mi történik akkor, ha egy tudományos felmérést szeretnénk végezni, és ehhez egy webes kérdőívet kérnek kitölteni a vizsgált közönséggel. Hogyan tudok magabiztos nyugalommal bízni a kitöltőkben, hogy tényleg releváns adatokat szolgáltatnak, és nem valótlanokat, hibásakat. Esetleg nem hagynak-e üresen kitöltendő mezőket? Ilyen és ehhez hasonló kérdésekkel nézhetünk szembe, de a JavaScript segítségével ki is iktathatjuk ezeket a nemkívánatos eseteket. Ezzel növelvni tudjuk az interaktivitást és az űrlap kitöltésének hatékonyságát.

Űrlapok készítésénél célszerű már a kitöltés fázisában ellenőrizni, hogy megfelelőek-e a bevitt adatok, és ha probléma merül fel, még akkor lehet korrigálni egy figyelmeztető üzenettel, vagy meg sem engedjük a szintaktikailag helytelen adatok begépelését. Ráadásul, ha hibás a kitöltött űrlap, egy ellenőrzéssel meggátolhatjuk, hogy a hibás űrlapot elküldjék, ezzel feleslegesen növelve az internetes forgalmat.

Általában két helyen szokás az ellenőrzést elvégezni. Minden egyes űrlapelemnél, amint a felhasználó kitölti azt, az *onChange*, illetve elküldéskor az *onSubmit* eseménykezelő segítségével. A következőkben több példát is látunk, amelyekben valamilyen adat ellenőrzését végezzük el.

1. példánkban azt ellenőriztetjük egy JavaScript-függvénnyel, hogy a beviteli mezőbe vittek-e be adatot. Ha üres az ellenőrző gomb lenyomásakor, akkor egy figyelmeztető üzenetablak jelenik meg.

```
<html>
<head>
<script language="JavaScript">
function uresCheck(mezo){
  if (mezo.value != "") return true;
  else{
    alert("Üres mező!");
    return false;
  }
}
</script>
</head>
<body>
<form ures="form_ures">
 <input type="text" name="ures" size="20">
```

```
 <input type="button" value="Ellenőrizd" onClick="uresCheck(form_ures.ures)">
</form>
</body>
</html>
```

A példakód kiemelt sorai jelentik a beviteli mező ellenőrzési fázisának a lényegét. A program törzsében hívjuk meg gomb lenyomásának hatására az *uresCheck(mezo)* függvényt. A függvény argumentuma a *mezo*, melynek értékét (value) egy elágazás segítségével vizsgáljuk meg, hogy üres-e vagy sem. Ha nem üres (*!=""*), akkor igaz értékkel tér vissza a függvényünk, de ha üres, akkor megkapjuk az *„Üres mező!"* üzenetet.

2. példánkban egy másik tipikus hibalehetőséget zárunk ki. Az e-mail címet ellenőrizzük le, hogy formailag helyes címet adott-e meg a kitöltőnk. Részleteiben azt tudjuk ellenőrizni, hogy a megadott e-mail címben szerepel-e az @ jel, illetve előtte és utána szerepelnek-e karakterek. Az erre vonatkozó függvény a következőképpen nézhet ki:

```
function emailCheck(mezo){
  szoveg = mezo.value;
  if (!(szoveg.indexOf('@')>0 && szoveg.indexOf('@')<szoveg.length-1)){
    alert("Rossz e-mail cím!");
    return false;
  }
  else{
    alert("Formailag helyes e-mail cím!");
     return true;
  }
}
```

Paraméternek ez a függvény is a vizsgált űrlapmezőt várja, mely értékét beteszi a *szoveg* nevű változóba. A *szoveg.indexOf('@')* függvénnyel lekérdezzük az *'@'* helyét. Ha ez nagyobb, mint 0 (azaz nem az első karakter) és kisebb, mint *szoveg.length-1* (azaz nem az utolsó karakter), akkor egy formailag helyes e-mail címet írtunk be. Ezt még negáljuk, hiszen elsősorban arra vagyunk kíváncsiak, hogy mikor nem jó a cím.

3. példánkban a mezőre fókuszálást mutatjuk be, vagyis azt, amikor például a kitöltő számára meg akarjuk könnyíteni a kitöltési munkákat és kiemelve az aktuálisan kitöltendő mezőre fókuszálunk, jelezve a kitöltőnek, hogy mi az aktuálisan kitöltendő mező. Ezt a módszert gyakran akkor használják, amikor ellenőrzés után hibásnak véli a program a begépelt adatot és ráállítja a mutatót a mezőre, kérve a felhasználót az újbóli kitöltésre.

```
<html>
<head>
<script lanuage="JavaScript">
var aktualis = 0;
function fokusz_allitas(){
  if (aktualis == 0){
```

```
    document.select_form.text1.focus();
    document.select_form.text1.select();
  }
  else{
    document.select_form.text2.focus();
    document.select_form.text2.select();
  }
  aktualis = 1 - aktualis;
}
</script>
</head>
<body>
<form name="select_form">
<input type="text" name="text1" value="Az első mező" size="15"><br>
<input type="text" name="text2" value="A második mező" size="15"><br><br>
<input type="button" value="A fókusz állítása" onClick="fokusz_allitas()">
</form>
</body>
</html>
```

A *fokusz_allitas()* függvény végzi a fókuszálást, illetve a mező tartalmának kijelölését. Ezt az *aktualis* nevű változó alapján teszi. Ha ez 0, akkor az első mezőt, ha 1, akkor a második mezőt teszi aktuálissá.

Végül a fejezet zárásaként nézzük meg, hogyan lehet a kitöltött űrlapot elküldeni levélben.

```
<form method="post" action="mailto:e-mail@cim" enctype="text/plain">
...
Itt az űrlap mezőinek definiálása következik
...
</form>
```

Ezzel a HTML-kódrészlettel tudjuk megoldani az űrlapok levélbe történő elküldését. Könnyen értelmezhető, hogy az action jellemzőbe adjuk meg az e-mail címet, ahová szeretnénk kapni eredeti formában (ezt az *enctype* metódus biztosítja) a kitöltött űrlapokat.

Természetesen ezzel nem merítettük ki a lehetőségek tárházát a JavaScript tekintetében. Célunkat viszont teljesítettük, és a legfontosabb alappilléreket áttekintettük, illusztrálva azokat konkrét példákkal, kódrészletekkel. Ahhoz, hogy mélyebb, részletesebb ismeretekkel és rutinnal rendelkezzünk, szükségesnek tartom más irodalmak felkutatását, és nem utolsósorban feladatok gyakorlását, megoldását. Ezzel a fejezettel a kliens oldali webprogramozási ismereteket alapoztuk meg, és a következő fejezetben megnézzük, hogy mi történik a szerver oldalon.

7. Összefoglalás

A gyakorló feladatok elkészítésével tananyagunk végéhez értünk. Ez azonban nem azt jelenti, hogy a tanulnivalók végét is elértük. A webprogramozás misztikus világáról bátran állíthatjuk, hogy a „végtelenhez konvergál”, így az ezzel kapcsolatos ismeretszerzési tanulmányainknak nem érhetünk sosem a végére. Napjainkban elképesztő fejlesztési verseny van a szoftverpiacon, és ha valaki ebbe belecsöppen, nem sok eséllyel tud kilépni a fejlődés folyamatából. Azt gondolom, hogy ez így van jól.

Amikor elkezdtem dolgozni ezen a tananyagon, magam sem gondoltam, hogy ennyire „hiányos” munkát fogok kiadni a kezemből. Ezeket a hiányosságokat azonban nem lehet negatívumként jegyezni, mert a cél egyrészt nem is az volt, hogy egy teljes, webprogramozással kapcsolatos, minden ismeretet tartalmazó egyetemi jegyzet szülessen. Meggyőződésem, hogy ilyen munkát képtelenség lenne elkészíteni. Másrészt pedig a jegyzet elkészítésével arra vállalkoztam, hogy bevezessem a hallgatókat a webprogramozás világába, mellyel egyfajta inspirációt keltsek a tanulókban továbbfejlődési szándékukhoz.

Tartalmi összefoglalás címén elmondható, hogy jelen jegyzet tartalmazza mindazt az ismeretanyagot, amelyet egy szemeszter alatt el kell sajátítaniuk azoknak a hallgatóknak, akik felveszik a webprogramozás kurzust. A könyv első részében a szükséges alapvető ismereteket gyűjtöttem össze a web működésével kapcsolatosan. A számítógépek fizikai összekapcsolódásától kezdve eljutottunk a statikus és dinamikus honlapok kialakításáig. Először a HTML-lel mint leíró nyelvvel kapcsolatos tudnivalókat szedtük rendbe. Kliens oldali programozást tanultunk először a statikus lapok kialakításához, majd ezt követően még mindig kliens oldalról, de már a dinamikusságot próbáltuk ki a JavaScript segítségével. JavaScripttel interaktívvá tehetjük honlapjainkat, ami napjainkban marketingszempontból is fontos.

A jegyzet következő, lényegi része a szerver oldali programozásba vezette be az olvasót. A PHP programozási nyelvvel kezdődött meg az ismerkedés. Megtanultuk, hogyan tudjuk használni a szervert, és kapcsolatot létrehozni vele.

Telepítettük az XAMPP programcsomagot, majd el is készítettük első PHP-programunkat. Lépésről lépésre vettük át az ehhez a nyelvhez kapcsolódó alapismereteket, mint például a literálok szerepét, a függvények használatát, a MySQL adatbázisok használatát, vagy akár a teljesség igénye nélkül, az objektumorientálás jelentőségét.

A jegyzetet záró fejezet a gyakorló feladatokat tartalmazza. HTML-ben, JavaScriptben és PHP-ban megoldandó feladatokat gyűjtöttem össze gyakorláshoz. Szívből ajánlom, önállóan dolgozzanak főleg azok, akik még sosem készítettek ezelőtt webes alkalmazásokat. A feladatokat nem csupán a leírásukkal, hanem megoldási javaslatokkal is alátámasztottam az ellenőrizhetőség megkönnyítése miatt.

Szinte biztos vagyok abban, hogy az alapok megismerése után mindenki úgy érzi, hogy bővült az elsajátítandó ismeretek halmaza, annak ellenére, hogy megtanulta a könyvben szereplő sok ismeretanyagot.

Bízom benne, hogy ezzel nem ment el senkinek a kedve a webprogramozástól, hanem inkább, aki kicsit félve is kezdett bele a tanulásba, további webes kurzus felé

kacsingat. Innentől kezdve akár saját kreativitásunkat is elővehetjük és a fejlesztési lehetőségek végtelen tárházába merülhetünk akár hobbi, akár hivatásunk szemszögéből is.

Végül bízom benne, hogy ezzel a jegyzettel érdemben hozzájárultam a hazai felsőfokú képzésfejlesztéshez, és hatékonyan bővítettem a rendelkezésre álló tanulási segédeszköz-repertoárt az informatikai képzési területen.

Sok sikert kívánok! Dr. Nagy Enikő

8. IRODALOMJEGYZÉK

Balássy György: *Honlapépítés a XXI. században, Kezdőknek és haladóknak*, Jedlik Oktatási Stúdió Kft. Budapest 2010

Covic Zlatko: *Internet technológiák*, Szabadkai Műszaki Főiskola, Szabadka 2005

Dr. Medzihradszky Dénes: *A webprogramozás alapjai*, Főiskolai jegyzet 2007

Kupás Péter: *Weblapszerkesztés*, Oktatási segédlet, Károly Róbert Főiskola, Gyöngyös 2003

Matt Zandstra: *Tanuljuk meg a PHP4 használatát 24 óra alatt*, Kiskapu Kft. Budapest 2001

Mester Gyula: *Web programozás*, Szabadkai Műszaki Főiskola, Szabadka 2004

Nagy Gusztáv: *Web programozás alapismeretek*, Ad Librum Kiadó Budapest 2011

Paczona Zoltán: *HTML technikák a gyakorlatban*, Professzionális sorozat, Computer Panoráma Kiadó, Budapest 2001

Szabó Bálint: *Webprogramozás I.* Médiainformatikai kiadványok, Eszterházy Károly Főiskola, Eger 2013

http://www.w3schools.com
http://www.w3c.org
http://developer.mozilla.org/en/docs/JavaScript
http://www.apache.org/
http://www.mozilla.com/en-US/firefox/
http://weblabor.hu/