

Կ. Ա. ԽԱՉԱՏՐՅԱՆ
Ա. Կ. ԽՈՒՐՇՈՒԴՅԱՆ

Ինտերնետ ծրագրավորման հիմունքներ

Ուսումնական ձեռնարկ

ԵՐԵՎԱՆ
2006

Մասնագետ խմբագիր տ.գ.թ., դոցենտ Վ.Ա.Սարգսյան
Գրախոսներ՝ տեխ. գ. դ., պրոֆեսոր Թ.Ա.Նալչաջյան
տեխ. գ.թ., դոցենտ Ռ.Վ.Արմենակյան

Խաչատրյան Կ. Ա., Խուրշուդյան Ա.Կ.

Ինտերնետ ծրագրավորման հիմունքներ: Ուսումնական ձեռնարկ: -
Եր.: «Տնտեսագետ», 2006.- 308 էջ:

Ներկայացվող աշխատանքը նվիրված է ինֆորմացիոն տեխնո-
լոգիաների զարգացման արդի փուլին համապատասխան ինտեր-
նետ ծրագրավորման նոր մեթոդների ուսումնասիրությանը:

Ձեռնարկում լուսաբանվում են Web կայքերի պատրաստման և
ավտոմատացման համար նախատեսված ծրագրավորման HTML,
Vbscript, Javascript լեզուների և ASP տեխնոլոգիայի աշխատանքի
սկզբունքներին:

Ձեռնարկը նախատեսված է տնտեսագիտական բուհերի ուսանող-
ների, ասպիրանտների, ինտերնետ ծրագրավորումը գործնա-
կանում կիրառող մասնագետների համար, ինչպես նաև կարող է
ուղեցույց լինել web դիզայն և ինտերնետ ծրագրավորում դասա-
վանդող դասախոսների համար:

ԳՄԴ 00.0

Խ 00000000000 2004
0000(00)-0000

ISBN 99999-111-1

Բովանդակություն

Առաջաբան	5
ԳԼՈՒԽ 1. WEB ԷՋԵՐԻ ՍՏԵՂԾՄԱՆ ՏԵԽՆՈԼՈԳԻԱՆԵՐԸ	6
§1.1. Համակարգչային ցանցերի փոխհամագործակցությունը	6
§1.2. Հասցեավորման համակարգը Ինտերնետում	9
§1.3. Ինտերնետում մատուցվող ծառայությունները	13
§1.4. Փաստաթղթաշրջանառության կազմակերպումը WWW-ում	17
§1.5. Web - էջերի ստեղծման տեխնոլոգիաները:	23
§1.6. Web – հանգույցի մշակման փուլերը:	28
ԳԼՈՒԽ 2. HTML ԳԾԱՆՇՄԱՆ ԼԵՁՈՒՆ	34
§2.1. HTML փաստաթղթերի կառուցվածքը և հիմնական տեգերը	34
§2.2. Տեքստի գծանշումը և կազմակերպումը HTML փաստաթղթերում	39
§2.3. Հիպերհղումների կազմակերպումը HTML-ում	50
§2.4. Գրաֆիկայի և մուլտիմեդիայի օգտագործումը HTML փաստաթղթերում	59
§2.5. Web-էջերի աղյուսակային ձևավորումը	70
§2.6. Ոճերի աղյուսակներ, հատուկ սիմվոլներ	84
§2.7. Սայթերի կառուցումը ֆրեյմերի միջոցով	100
§2.8. Meta որոշիչները	111
§2.9. Web-խմբագրիչները	113
Լաբորոտոր աշխատանքների առաջադրանքներ	117
ԳԼՈՒԽ 3. ԴԻՆԱՄԻԿ WEB-ԷՋԵՐԻ ՍՏԵՂԾՈՒՄ: ՍՑԵՆԱՐԱՅԻՆ ԼԵՁՈՒՆԵՐ	120
§3.1. HTML ֆորմաները	120
§3.2. Ծանոթություն JavaScript լեզվի հետ	130
§3.3. JavaScript լեզվի օբյեկտները	148
§3.4. Իրադարձությունների մշակումը Javascript լեզվում	162
§3.5. Բրաուզերի օբյեկտային մոդելը, օբյեկտները և հավաքածուները (կոլեկցիաները)	169
§3.6. HTML էջերի բովանդակության և տեսքի դեկլարումը	191
§3.7. Սովորական և մոդալ նոր պատուհանների ստեղծումը	203
§3.8. Ֆորմաների բովանդակության նախնական մշակումը	215
§3.9. ActiveX դեկլարման էլեմենտների օգտագործումը	222
Լաբորոտոր աշխատանքների առաջադրանքներ	239
ԳԼՈՒԽ 4. ԻՆՏԵՐԱԿՏԻՎ WEB-ԷՋԵՐ: ՍԵՐՎԵՐԱՅԻՆ	242

ԾՐԱԳՐԱՎՈՐՈՒՄ	
§4.1. Տվյալների ստացումը ֆորմաներից	242
§4.2. Ակտիվ սերվերային էջեր (Active Server Pages)	247
§4.3. ASP-ի Response և Request օբյեկտների կիրառությունը	256
§4.4. Session և Application օբյեկտների կիրառությունը	268
§4.5. Server օբյեկտը, աշխատանքը տվյալների բազների հետ	272
§4.6. XML լեզվի կիրառության բնագավառները	287
§4.7. PHP լեզվի կիրառության բնագավառները	
Առաջադրանքներ 3-րդ և 4-րդ գլուխների թեմաներով	297
ՀԱՎԵԼՎԱԾՆԵՐ	299
Հավելված 1. Ոճերի աղյուսակների հատկությունների տեղեկատու	299
Հավելված 2. Առավել հաճախ կիրառվող պրիմիտիվների ցուցակը	309

Առաջաբան

Մարդկության համար մոլորակն այսօր դարձել է չափազանց «փոքր» և մարդկային միտքը փորձում է նոր հորիզոններ գրավել: Ժամանակակից գիտությունը դասական ուղղությունների հետ մեկտեղ փորձում է ստեղծել քաղաքակրթության նոր պայմաններին համապատասխանող սիստեմոլոգիական ուղիներ: Ամենուրեք կատարվում է գլոբալիզացիայի պրոցեսը որին մեծապես նպաստում է նաև Ինտերնետը՝ համաշխարհային համակարգչային ցանցը:

Ինֆորմացիոն տեխնոլոգիաները վճռական ազդեցություն են գործում հասարակության տնտեսական և սոցիալական կառույցների վրա: Այդ տեխնոլոգիաների հիման վրա է ձևավորվում «ինֆորմացիոն» հասարակությունը, որում, որպես հետևանք, անհետանում է մասնագետների աշխատատեղերում գտնվելու ավանդական անհրաժեշտությունը:

Այդ առումով ինֆորմացիոն ռեսուրսների ձևավորման հիմնական միջոցներ են դառնում Web կայքերը: Դրանց աշխատանքը կազմակերպող Web տեխնոլոգիաները ապահովում են միևնույն գործառույթները ինչպես համաշխարհային՝ Internet, այնպես էլ առանձին կազմակերպությունների՝ Intranet լոկալ համակարգչային ցանցերում:

Ձեռնարկը ծանոթացնում է կայքերի ավտոմատացված մշակման ծրագրային միջոցներին՝ Web ծրագրավորման հնարքներին, ինչպես նաև Web խմբագրիչներին:

Հատուկ ուշադրություն է դարձվում HTML-ին և Javascript ու Vbscript սցենարային լեզուներին:

Ծրագրավորման տեխնոլոգիաների զարգացման արդի փուլում գիրքը արդիական և արժեքավոր է տնտեսագիտական համակարգերի ստեղծմամբ և դրանք Ինտերնետում տեղադրմամբ զբաղվող մասնագետների համար: Այն կարող է արժեքավոր ուղեցույց ծառայել ինչպես տնտեսագիտական բուհերի ուսանողների, այնպես էլ այն ասպիրանտների և դասախոսների համար:

Գործնական և մեթոդական առումով ձեռնարկը արժեքավոր է պարունակվող գործնական աշխատանքների կատարման համար անհրաժեշտ բազմաթիվ խնդիրների և ծրագրերի առկայությամբ:

ԳԼՈՒԽ 1. WEB ԷԶԵՐԻ ՍՏԵՂԾՄԱՆ ՏԵԽՆՈԼՈԳԻԱՆԵՐԸ

§1.1. Համակարգչային ցանցերի փոխհամագործակցությունը

Համաշխարային Ինտերնետ ցանցի նախահայրը կարելի է համարել ARPANet հաշվողական ցանցը, որը հիմնվել էր “Առաջադիմական հետազոտությունների գործակալություն” (Advanced Research Projects Agency) ամերիկյան կառավարական կազմակերպության կողմից 60-ական թվականների վերջում: Ինքստինքյա այդ էլեկտրոնային ցանցի ստեղծման փաստը կմնար աննկատ, կամ հետզհետե մոռացության կմատնվեր, եթե չլիներ մի քանի կոնցեպտուալ առանձնահատկություններ, որոնք դրված էին նախագծի հիմքում: Առաջին առանձնահատկությունն այն էր, որ ARPANet-ի բոլոր համակարգիչները հաղորդվում էին իրար հետ հավասար մակարդակներով, այսինքն չկար “գլխավոր համակարգիչ - ենթակա համակարգիչ” հասկացությունը՝ ցանցը ապակենտրոնացված էր: Երկրորդը՝ որպես ցանցային արձանագրություն ARPANet-ում ընդունվել էր միջցանցային IP (Internet Protocol) արձանագրությունը:

❖ **Ցանցային արձանագրություն է կոչվում համաձայնեցված և հաստատված ստանդարտ, որը պարունակում է երկու համակարգիչների միջև տեքստի, զծապատկերների (գրաֆիկայի) և այլ ինֆորմացիայի հաղորդման ու ընդունման կանոնները և, որը ծառայում է ցանցում գտնվող համակարգիչների աշխատանքը սինխրոնացնելու համար:**

Այդ արձանագրությունն օգտագործելու շնորհիվ ծրագրերն ու տվյալների փաթեթները ինքնուրույն էին “գտնում ճանապարհը” մեկ հանգուցից մյուսը, քանի որ դրանցից յուրաքանչյուրն ուներ սեփական հասցեն (այսպես կոչված IP-հասցեն՝ այն մենք կքննարկենք քիչ ուշ), որն իրենից ներկայացնում էր որոշակի թվային ծածկագիր: Դա նշանակում էր, որ ուղարկող համակարգիչից մինչև ստացողը փաթեթները կարող էին հասնել տարբեր ճանապարհներով և դա, անկասկած, թույլ էր տալիս մեծացնել այդպիսի համակարգի կայունությունը: Միջցանցային IP արձանագրությունը հանդիսացավ ունիվերսալ միջհիմքային ստանդարտ և թույլ տվեց միավորել մեկ ցանցում տարատեսակ համակարգիչներ, որոնք աշխատում էին տարբեր օպերացիոն համակարգերի ղեկավարման ներքո: Կարևոր էր միայն, որ այդ համակարգերը կարողանային սատարել IP արձանագրությանը:

Սակայն IP արձանագրությունը թույլ էր տալիս միայն հաղորդել տվյալները: Հաղորդման պրոցեսը ARPANet-ում ղեկավարելու համար 80-ական թվականների սկզբին վերջնականապես մշակվեց տվյալների փոխանակումը վերահսկող TCP (Transmission Control Protocol) արձանագրությունը, որը հիմնվում էր IP արձանագրության հնարավորությունների վրա: Միավորված TCP/IP արձանագրությունը դարձավ հիմնականը՝ փաթեթների փոխանակումը ցանցում իրականացնելու համար: Այդ ժամանակ էլ պարզ դարձավ, որ նշված արձանագրությունը կարող է օգտագործվել նաև տարբեր համակարգչային ցանցերի միացման համար: Դա թույլ էր տալիս կազմակերպել տվյալների փոխանակումը ազգային և, նույնիսկ, միջազգային մակարդակներով: Հենց այդպիսի “ցանցերի ցանցի” անվանման համար առաջին անգամ օգտագործվեց “Ինտերնետ” թերմինը: 80-ական թվականների վերջում ARPANet նախագիծն ավարտվեց, սակայն այդ ժամանակ Ինտերնետը արդեն հասանելի էր դարձել բազմաթիվ համալսարաններին և գիտական կազմակերպություններին: 90-ականների սկզբում տարբեր կորպորացիաներ ակտիվորեն սկսեցին օգտագործել Ինտերնետը “Էլեկտրոնային փոստի” միջոցով տվյալների փոխանակելու համար: Իսկ 1991 թ., երբ Ազգային Գիտական Ֆոնդը հանեց ցանցի կոմերցիոն օգտագործման վրա դրված արգելքը, Ինտերնետը հասանելի դարձավ բազմաթիվ անհատներին և կազմակերպություններին:

Ինֆորմացիայի հաղորդման պրոցեսի վերահսկումը պարզաբանելու նպատակով պատկերացնենք, որ ցանկանում ենք ուղարկել փոստով մի հաստ ամսագիր, չժախսելով լրացուցիչ գումար բանդերով ձևակերպելու համար: Ինչպե՞ս լուծել պրոբլեմը այն դեպքում, երբ փոստը չի ընդունում մեկ թերթիկից ավելին պարունակող նամակներ: Ելքը հետևյալն է՝ բաժանել ամսագիրը առանձին թերթիկների և ուղարկել յուրաքանչյուրն առանձին նամակով: Իսկ ստացողը ըստ էջերի համարների նորից կհավաքի առանձին թերթիկները մեկ ամսագրի:

Մոտավորապես նույն սկզբունքով էլ աշխատում է TCP արձանագրությունը: Այն մասնատում է ինֆորմացիան մի քանի մասերի, շնորհիվ յուրաքանչյուր մասին համար, ըստ որի հետազայում հնարավոր կլինի միավորել այն մյուսներին, կցում յուրաքանչյուր մասին “ծառայողական” ինֆորմացիա և տեղավորում առանձին “IP ծրարում”: Դրանից հետո ստացված “ծրարը” ուղարկվում է ցանցով հասցեատիրոջը (Ինտերնետը արդեն գիտի, ոնց մշակել IP ինֆորմացիան): Հասցեատուի համակարգչում TCP արձանագրությունը կատարում է հակառակ պրոցեսը՝ գտնում է ուղեկցող

ինֆորմացիան և ներկայացնում այն “հավաքված” (այսինքն՝ սկզբնական) տեսքով: Ինտերնետում տվյալների հաղորդման ժամանակակից սխեման ունի բազմաշերտ կառուցվածք, որն ընդգրկում է մի քանի մակարդակներ: Այդ կարուցվածքը կոչվում է OSI (Open Systems Interconnection՝ բաց համակարգերի փոխհամակցում): Այսինքն “ծրարի” փաթեթավորման պրոցեսը կատարվում է մի քանի մակարդակներում (հաղորդելիս վերից վար, իսկ ընդունելիս վարից վեր) և յուրաքանչյուր մակարդակում “ծրարին” ավելացվում է (ստացողի մոտ հակառակը՝ պակասեցվում է) որոշակի ծառայողական ինֆորմացիա:

Էլեկտրոնային ցանցերով հաղորդվող ինֆորմացիան շատ հաճախ կորչում է կամ տարբեր պատճառներով աղավաղվում կապի գծերում: TCP-ին ունի ներկառուցված ծրագրեր, որոնք վերահսկում են հաղորդվող ինֆորմացիայի ստույգությանը: Օրինակ, առավել տարածված վերահսկման մեթոդը կայանում է նրանում, որ յուրաքանչյուր IP փաթեթ ուղակելիս, համակարգիչը դրա վերնագրային (ծառայողական) մասում գրանցում է ստուգողական թվեր: Ստացող համակարգիչը համապատասխան ձևով հաշվարկում է այդ թիվը և համեմատում վերնագրում եղած թվի հետ: Անհամապատասխանության դեպքում TCP-ն փորձում է կրկնել հաղորդումը: Հարկ է նշել, որ ինֆորմացիոն փաթեթներ ուղարկելիս TCP-ն պահանջում է ստացման հավաստագրումը: Դա իրականացվում է հաղորդման ընթացքում հատուկ սպասումների (թայմ-աութների) կազմակերպման միջոցով: Եվ, նույնիսկ այն դեպքում, երբ հավաստագրումը դեռ չի ստացվել տվյալները շարունակվում են հաղորդվել: Առաջանում է հաղորդված, սակայն ստացումը չհավաստված տվյալների որոշակի ծավալ: Այլ կերպ ասած TCP-ն կազմակերպում է ինֆորմացիայի երկկողմանի փոխանակում, ինչը նպաստում է դրա հաղորդման ավելի մեծ արագագործությանը:

Իրականում հաղորդող համակարգիչից ինֆորմացիան անմիջապես չի հաղորդվում հասցեատեր համակարգիչին: Սկզբում տվյալները հաղորդվում են այն համակարգիչին, որը կապակցում է հաղորդողին Համաշխարհային ցանցի հետ, հետո հաջորդին և այդպես մինչև առաջին ցանցային հանգույց:

❖ **Ցանցային հանգույց է կոչվում համակարգիչը, որը միավորում է միևնույն ցանցային արձանագրությունը օգտագործող մի քանի լոկալ ցանցեր:**

Հանգույցում որոշվում է այն ուղղությունը, որը մոտավորապես համապատասխանում է հասցեատեր համակարգիչ գտնվելու վայրին, այսինքն որոշվում է ինֆորմացիոն փաթեթի երթուղին: Ամեն

մի հաջորդ հանգույցում նույնպես որոշվում է փաթեթի հետագա երթուղին և այդպես շարունակ, մինչև այն հասնում է վերջնական ստացողին: Որպեսզի փաթեթը “չնուրբի”, Ինտերնետի բոլոր հանգույցներում առկա են այսպես կոչված “երթուղային աղյուսակները” – տվյալների բազաներ, որոնք պարունակում են այս կամ այն փաթեթը ուղարկելու ուղղությունների որոշման կարգադրերը: Երթուղիները որոշող հանգույցների համակարգիչ սերվերները ստացել են երթուղավորողներ կամ ռուտերներ (router) անվանումը: Երթուղիները որոշող կանոնները նկարագրված են մի քանի արձանագրություններում՝ ICMP (Internet Control Message Protocol), RIP (Routing Internet Protocol) և OSPF (Open Shortest Path First):

Իսկ ինչպե՞ս է երթուղավորողը իմանում, որ ուղղությամբ է անհրաժեշտ ուղարկել ինֆորմացիոն փաթեթը: Իհարկե հենց հաղորդողից, քանի որ ինչպես և փոստով նմանակ ուղարկելիս, ինֆորմացիան ցանցով հաղորդելիս անհրաժեշտ է նշել, թե ու՞ր (կամ ու՞մ) է այն ուղարկվում:

§1.2. Հասցեավորման համակարգը Ինտերնետում

Ինչպես երկրագնդի յուրաքանչյուր բնակիչ ունի հասցե, ըստ որի անհրաժեշտության դեպքում նրան կարելի է գտնել, այնպես էլ Ինտերնետում աշխատող յուրաքանչյուր համակարգիչ ունի ունիկալ հասցե: Իհարկե, Ինտերնետի հասցեները տարբերվում են փոստայիններից: Օրինակ, եթե մենք գրենք “Պարոն Intel Pentium IV – ին, ք. Երևան, փ. Աբովյան, շ.15, բն.10”, ապա համակարգիչը լավագույն դեպքում այն կանտեսի: Սակայն օրինակ՝ 192.85.102.14 տեսքի գրանցումը համակարգին միանգամայն հասկանալի է, քանի որ այն համապատասխանում է TCP/IP արձանագրության ստանդարտին և կոչվում է **IP հասցե**:

IP հասցեն կազմված է չորս տասական մեկ բայտանոց իդենտիֆիկատորներից (**օկտետներից**), որոնք իրարից բաժանվում են կետերով: Ձախ օկտետը բնութագրում է այն լոկալ ենթացանցի տեսակը, որին անմիջապես միացված է համակարգիչը: TCP/IP ստանդարտը նախատեսում է հինգ տեսակի ենթացանցեր, որոնք բերված են աղյուսակ 1.1-ում:

A դասի հասցեները նշանակված են ընդհանուր օգտագործման խոշոր ցանցերի համար, որոնք թույլ են տալիս ստեղծել մեծ թվով հանգույցներ ընդգրկող համակարգեր: B-ն՝ միջին չափերի կորպորատիվ ցանցերի համար: C դասը օգտագործվում է ոչ մեծ ձեռնարկությունների լոկալ ցանցերում: Համակարգիչների առան-

ծին խմբերին դիմելու համար օգտագործվում են D դասի համարների սերիան, իսկ E դասը առայժմ (ինչպես մեզ հայտնի է) չի օգտագործվում:

Աղյուսակ 1.1.

Ենթացանցերի տեսակները

Ցանցի դասը	Չախ օկտետի համարները	Ենթացանցերի հնարավոր քանակը	Հանգույցների հնարավոր քանակը
A	1 – 126	126	16777214
B	128 – 191	16382	65534
C	192 – 223	2097150	254
D	224 – 239	-	2 – 28
E	240 - 247	-	2 - 27

IP հասցեի վերջին օկտետը ցույց է տալիս հոստի համարը տվյալ լոկալ ցանցում:

❖ Հոստ են անվանում Ինտերնետին միացած ցանկացած համակարգիչը, անկախ դրա նշանակումից:

Մյուս երկու օկտետները՝ ավելի ցածր մակարդակների ենթացանցերի համարներն են: Բացատրենք դա օրինակի վրա: Ենթադրենք, մենք ցանկանում ենք ուղարկել 196.65.28.14 հասցեով թարմ համակարգչային վիրուսների փաթեթ: Սկզբում այն կուղարկվի Ինտերնետի 196-րդ ցանցին (այն պատկանում է, ինչպես տեսնում ենք C դասին): Ընդունենք, որ այդ ցանցը պարունակում է 71 ենթացանցեր, սակայն փաթեթը ուղարկվում է դրանցից 65-րդի 28-րդ ավելի փոքր ենթացանցին և, եթե այդ ցանցին միացված են օրինակ, 30 համակարգիչներ, ապա դժվար չէ կռահել, որ փաթեթը կստանա դրանցից 14-րդ համար ունեցող համակարգիչը:

Սակայն աշխարհում դժվար թե գտնվի մի մարդ, որը կկարողանա անգիր անել թվերի այդ ահռելի հավաքածուները, իսկ քանի որ Ինտերնետը ստեղծող մասնագետները նույնպես մարդիկ են, ապա որոշ ժամանակ մտորելուց հետո նրանք ստեղծեցին հասցեավորման չափազանց օգտակար և հարմար եղանակ, որը կոչվում է “Անունների Դոմեյնային Համակարգ” – DNS (Domain Name System): Որպես հիմք DNS-ում ընդունված է տվորական փոստային հաղորդումների (օրինակ՝ նամակների) հասցեավորման սկզբունքը: Սկզբում “նամակը” ուղարկվում է առավել խոշոր ադմինիստրատիվ տարածք՝ երկիր: Քանի որ աշխարհում գոյություն չունեն միևնույն անուն ունեցող երկու երկրներ, ապա այդ փուլում նամակը չի կորչի: Դրանից հետո հաջորդաբար այն հասնում է քաղաք, փողոց, շենք և, վերջապես, բնակարան: Այսինքն եթե մենք նկարագրենք հասցեն օրինակ այսպես “բն. 1,

2.12, փ.Աբովյան, ք.Երևան, Հայաստանի Հանրապետություն” (Միացյալ Նահագնեություն, որտեղ ստեղծել են Ինտերնետը, հասցեն գրանցվում է հենց այդպիսի հաջորդականությամբ), ապա հասնելով սկզբում Հայաստան, ապա Երևան, Աբովյան փողոց, 12-րդ շենք և, վերջապես, 1-ին բնակարան՝ նամակը վերջապես կգտնի հասցեատիրոջը:

Գործնականում դոմեյնային հասցեն ոչնչով չի տարբերվում վերը նշվածից, օրինակ, **myhost.mydomain.am**: Վիրտուալ հասցեյի այդպիսի նշանակումը ընդունված է անվանել **URL** (Uniform Resource Locator), ինչը կարելի է թարգմանել որպես “ռեսուրսի տեղորոշման ունիվերսալ որոշիչ”: Ինչպես տեսնում ենք Համաշխարհային Ցանցի այս կամ այն ռեսուրսի հասցեն մասնատված է մի քանի բաղադրիչների, որոնք կոչվում են դոմեյններ:

❖ **Դոմեյնը Ինտերնետի որոշակի տրամաբանական մակարդակ է, այսինքն ցանցային ռեսուրսների մի խումբ, որն ունի սեփական անուն և, որը ղեկավարվում է սեփական ցանցային կայանով:**

DNS հասցեի հիմնական բաղադրիչը՝ այսպես կոչված “առաջին մակարդակի դոմեյնն է”, որը բնութագրում է որոշակի գլոբալ աշխարհագրական տարածք, օրինակ առանձին պետության տերիտորիան (հատկանշական է, որ Միացյալ Նահագնեների սեփական տերիտորիալ դոմեյնը՝ “**US**” սովորաբար չի նշվում, քանի որ սկզբնական շրջանում Ինտերնետը հանդիսանում էր ամերիկական ազգային ցանց): Երկրորդ մակարդակի դոմեյններին (դրանք կարող են լինել բանկերի, գիտական կազմակերպությունների, քաղաքային մունիցիպալ ծառայությունների լոկալ ցանցեր կամ առանձին սերվերներ, որոնք հատկացնում են օգտվողներին տարբեր տեսակի ծառայություններ) տրվում են կամայական անուններ, միայն թե վերջիններս չկրկնվեն ավագ դոմեյնի սահմաններում: Նույն սկզբունքով են տրվում անուններ երրորդ մակարդակի դոմեյններին: Չորրորդ մակարդակում որպես հոստ սովորաբար վերցվում է կամ լոկալ ցանցում համակարգչին տրված անունը (օրինակ, comp10) կամ որևէ այլ անուն (օրինակ՝ Sargis, Lusine և այլն)՝ ըստ համակարգչի տիրոջ ճաշակի:

Ինչպես ասվեց՝ Ինտերնետը միասնական գլոբալ կառուցվածք է, որը միավորում է 13 հազարից ավելի տարբեր լոկալ ցանցեր և, բացի այդ բազմաթիվ առանձին (անհատ) օգտվողներին: Առաջներում Ինտերնետում միավորված բոլոր ցանցերը տվյալներ հաղորդելու համար օգտագործում էին միայն IP արձանագրությունը, սակայն զարգացման ընթացքում նրան միացան նաև այն լոկալ համակարգերի օգտվողները, որոնք չէին օգտագործում IP:

Տարբեր արձանագրություններից օգտվող ցանցերը միավորելու համար ստեղծվեցին, այսպես կոչված, շյուզերը:

❖ **Շյուզը ծրագիր է, որի միջոցով կարելի է ինֆորմացիան փոխանակել երկու տվյալների հաղորդման տարբեր արձանագրություններ օգտագործող, համակարգերի միջև:**

Շյուզերի միջոցով ինֆորմացիան հաղորդելու համար մշակվեցին նաև միջանցիկ արձանագրությունները, որոնք ապահովում են տվյալների անարգել անցումը IP ցանցերից ոչ IP-երին և, հակառակը:

Ավարտելով ինտերնետի կարուցվածքի և աշխատանքի սկզբնունքների նկարագրությանը վերաբերվող թեման հարկ է նշել ևս մեկ ցանցային արձանագրություն՝ Ֆայլերի Փոխանակման Արձանագրությունը – **FTP** (File Transfer Protocol): Ինչպես հետևում է անվանումից այն նշանակված է ինտերնետի միջոցով ֆայլեր հաղորդելու համար: Հենց այդ արձանագրության վրա են հիմնված այսպես կոչված “դաունլոդի” և “ափլոդի” գործառնությունները:

❖ **Դաունլոդ է (Download) կոչվում հեռացված ցանցային համակարգչից (սերվերից) նրան հարցում կատարող համակարգչի (կլիենտի՝ այցելույի) վրա ֆայլերի պատճենահանման գործընթացը: Ափլոդը (Upload) դրան հակառակ պրոցեսն է՝ ֆայլերի բեռնումը հեռացված համակարգչի վրա:**

FTP արձանագրությունը թույլ է տալիս համակարգչից համակարգիչ փոխադրել ոչ միայն առանձին ֆայլեր, այլ և ամբողջական դիրեկտորիաներ, որոնք կարող են ընդգրկել ցանկացած խորությամբ ներդրված ենթադիրեկտորիաներ: Դա իրականացվում է տվյալ արձանագրության ներկառուցված ֆունկցիաները նկարագրող հրամանների համակարգին դիմելու միջոցով:

Սերվերի և հաճախորդի միջև կապի համար FTP արձանագրությունը օգտագործում է TCP-ի երկու տարբեր միացումներ: Այդ միացումները կոչվում են ղեկավարող (control connection) և տեղեկատվական (data connection): Վերջինս կոչվում է նաև տվյալների փոխանակման միավորում (data transfer connection): Միացումը կարող է գտնվել 2 վիճակներից մեկում՝

Ղեկավարող միացումը կապ է կայացնում հաճախորդի և FTP սերվերի միջև և մնում է կայացած սեանսի ամբողջ ընթացքում:

Ղեկավարող միացումը հսկվում է գործընթացների հատուկ խմբով՝ սերվերի արձանագրության ինտերպրետատորով (server Protocol interpreter կամ server PI):

Շահագործողական արձանագրության ինտերպրետատորը իրականացնում է տվյալների հաղորդման գործընթացի ղեկավարումը (user Data Transfer Process՝ user DTP): Ինֆորմացիոն միա-

ցունը իրականացվում է հաճախորդի և սերվերի միջև տվյալների հաղորդման ժամանակ և ինֆորմացիայի հաղորդումը վերջացնելուց հետո այն փակվում է, որից հետո ղեկավարումը մնում է բաց:

Ամեն դեպքում, երբ սկսում է հաղորդման նոր գործընթաց, բացվում է հաճախորդի նոր հանգույց: Տվյալների հանգույցը սերվերում միշտ ունի 20 համարը: FTP սեանսը գործարկվում է հրամանային տողով: Որպեսզի բացել գործընթացը և միանալ սերվերին, անհրաժեշտ է հրամանային տողում հավաքել՝

ftp: // IP հասցեն կամ դոմեյնային հասցեն:

§1.3. Ինտերնետում մատուցվող ծառայությունները

Ինտերնետի համար մշակված բոլոր ծրագրերը բաժանվում են երկու մեծ խմբերի՝ սերվերներ և կլիենտներ: Սերվեր-ծրագրերը հատկացնում են կլիենտ-ծրագրերին այս կամ այն տեսակի ռեսուրսներ և ապահովում դրանց հասանելիությունը: Երբ կլիենտին անհրաժեշտ է լինում որևէ ֆայլ կամ ընդհանրապես որևէ տեսակի տվյալներ, նա ձևավորում է հատուկ կլիենտական հարցում և ուղարկում սերվերին: Սերվերը մշակում է այդ հարցումը և ուղարկում **սերվերային պատասխան**, որը պարունակում է կամ հարցվող տվյալները, կամ (այն դեպքում երբ այդ տվյալները որևէ պատճառով հասանելի չեն) հաղորդակցություն սխալի մասին: Համակարգչային ճարտարապետության, այսինքն, հաշվողական համակարգի կառուցման այդպիսի սկզբունքը կոչվում է **“կլիենտ-սերվեր”** կամ երկօղականի: Հենց դրանով էլ այն տարբերվում է մեկօղականի համակարգերից, որոնցում ցանցում ընդգրկված բոլոր համակարգիչները միմյանց մեջ հավասար են և կարող են ինչպես բաժանել, այնպես էլ օգտագործել ցանցային ռեսուրսները: “Կլիենտ-սերվեր” սկզբունքի վրա են հիմնված Ինտերնետի համարյա բոլոր ծառայությունները:

Ինտերնետ գլոբալ ցանցի միջոցով օգտվողներին մատուցվում է ծառայությունների բազմատեսակ ընտրանի, որոնց շարքին են պատկանում մեզ քաջ հայտնի այնպիսի ծառայություններ, ինչպիսիք են՝

WWW (World Wide Web) - Համաշխարհային Ոստայն,

Electronic Mail կամ E-Mail - Էլեկտրոնային փոստային ծառայություն,

Usenet - Ինտերնետային քննարկումներ,

Telnet – համակարգիչների փոխհամագործակցում:

1.3.1. Համաշխարհային ոստայն (WWW)

Web-սերվերը՝ համակարգիչ է, որը միացված է Ինտերնետին և որն ունի սերվերային ծրագրային ապահովում (օրինակ՝ Windows 2000 server, MacOS, Unix և այլն): Այն մշտապես “նստած է” Ինտերնետում և սպասում է հարցումներ կլիենտական ծրագրերից: WWW ծառայությունում որպես կլիենտ հանդես են գալիս web-մեկնաբանները՝ բրաուզերները (browser) կամ համանման ծրագրերը, որոնք կարողանում են մշակել web-էջերը (օրինակ Internet Explorer, Netscape, Opera): Web-սերվերները մշակում են web-մեկնաբանների հարցումները և ուղարկում վերջիններին անհրաժեշտ ֆայլերը:

WWW-ում որպես սերվերների և կլիենտների միջև “երկխոսության” հիմք օգտագործվում են **HTTP** (HyperText Transfer Protocol) – բարձրամակարդակ արձանագրությունը, որը աշխատում է TCP/IP արձանագրության “վրայից”, այսինքն սկսում է աշխատել միայն այն բանից հետո, երբ հաստատվի կլիենտ-սերվեր միացումը TCP/IP արձանագրությունով: Այդ կապը սովորաբար իրագործվում է ստանդարտ՝ 80-րդ հանգույցի (պորտի) միջոցով: Միացումը կայանալուց հետո կլիենտը (web-մեկնաբանը) հարցում է ուղարկվում web-սերվերին HTTP հրամանի միջոցով: Հարցումը մշակվում և կլիենտին ուղարկվում է պատասխանը, որը պարունակում է պահանջվող ինֆորմացիան հատուկ կերպով գծանշված փաստաթղթի՝ HTML փաստաթղթի տեսքով: Փաստաթուղթը ստանալիս, ընդունող կողմը արդեն ինքն է որոշում թե ի՞նչ է պետք անել դրա հետ՝ ցուցադրել էկրանի վրա, պահապանել սկավառակի վրա, թե “կեցցե” բղավել:

HTTP արձանագրությամբ օգտագործվում է հասցեավորման հետևյալ տեսքը՝ **http://IP հասցեն կամ դոմեյնային հասցեն:**

1.3.2. Էլեկտրոնային փոստ (E-Mail)

Էլեկտրոնային փոստը ներկայումս մարդկանց համար հանդիսանում է միմյանց հետ հաղորդվելու առավել տարածված միջոցներից մեկը: Այն հնարավորություն է տալիս հաշված վարկյալների ընթացքում հաղորդել ամենուր, որտեղ կա Ինտերնետ, տեքստ, web-էջեր և, առհասարակ, կամայական ֆայլեր, որպես նամակների ներդիրներ:

Էլեկտրոնային փոստի հասցեները որոշ չափով տարբերվում են DNS համակարգում ընդունված հասցեներից: Յուրաքանչյուր օգտվողի փոստարկղի հասցեն ունի հետևյալ տեսքը՝

օտվողի_անուն@փոստային_սերվերի_հասցե:

Որպեսզի համակարգիչը կարողանա զատել սերվերի անունը, որին ուղարկվում է նամակը, օգտվողի անունից (ավելի շուտ նրա անձնական կատալոգից, որում սերվերը գրանցում է ստացված

նամակները) օգտագործվում է @ նշանը (շատ հաճախ այն անվանում են “շնիկ”): Այդ կատալոգում նամակը պահպանվում է այնքան ժամանակ, մինչև հասցեատերը չվերցնի այն կամ չվերջանա պահպանման ժամկետը:

1.3.3. Հեռակոնֆերանսներ (Usenet)

Usenet-ի հեռակոնֆերանսները իրենցից ներկայացնում են երկխոսական խմբեր, որոնք կազմակերպված են աստիճանական (հիերարխիկ) սկզբունքով: Վերին մակարդակում Usenet-ը բաժանված է յոթ հիմնական թեմատիկ խմբերի: Դրանցից յուրաքանչյուրը իր հերթին ընդգրկում է հարյուրավոր ենթախմբեր: Կազմավորված է ծառանման կառուցվածք, որը հիշեցնում է ֆայլային համակարգի կառուցվածքը: Որպես օրինակ հիմնական (վերին մակարդակի) թեմատիկ խմբերից կարելի է առանձնացնել հետևյալները՝ Comp, Sci, News, Soc, Talk:

Մասնակցելով հեռակոնֆերանսի ըստ որևէ թեմայի, բաժանորդը կարող է ուղարկել իրեն հետաքրքրող թեմայով հաղորդագրություն: Գոյություն ունի այս գործընթացի իրականացման երկու մեթոդ՝ պատասխանի ուղարկումը հոդվածի անմիջական հեղինակին էլեկտրոնային փոստի միջոցով, սեփական հաղորդագրության տրամադրումը հեռակոնֆերանսի բոլոր մասնակիցներին:

1.3.4. Համակարգիչների փոխհամագործակցումը (Telnet)

Telnet համակարգը (ծառայությունը) ստեղծվել է հեռացված համակարգիչներին ադմինիստրատորի մակարդակով հասանելիություն ստանալու նպատակով: Հիմնականում այն նշանակված է հեռացված սերվերների վրա տեղադրված սայթերը սպասարկելու համար:

Պարզեցված տեսքով Telnet-ի աշխատանքի մեխանիզմը հետևյալն է: Հեռացված սերվերային համակարգչի ադմինիստրատորը բացում է օգտվողի (սայթի տիրոջ) համար “ադմինիստրատորի հաշիվ” (root account), հատկացնելով տրամաբանական հաշվառման գրանցում (login) և գախտնաբառ (password) սերվերի հետ միացումը իրագործելու համար: Տեղադրելով անձնական համակարգչի վրա հատուկ՝ Telnet-կլիենտ ծրագրային ապահովումը և կապ հաստատելով հեռացված համակարգչի հետ, օգտվողը կարող է ղեկավարել հեռավոր համակարգչը ինչպես անձնականը (բնականաբար հեռացված համակարգչի ադմինիստրատորի կողմից թույլատրված շրջանակներում): Օրինակ՝ հնարավոր է փոփոխել, հեռացնել, ստեղծել ֆայլեր և դիրեկտորիաներ: Հիմնականում օգտվողը ստանում է անսահմանափակ հասանելիության հնարավորություն միայն անձնական “թղթապանակին”:

Telnet-ը կարող է մատուցել նաև ծառայությունների երկու տեսակներ՝

գրադարանային կատալոգներ,

հայտարարությունների էլեկտրոնային վահանակներ (BBS–Bulletin Board System):

Առաջին դեպքում օգտվողը հնարավորություն է ստանում դիտարկել այլ օգտվողների կատալոգների բովանդակությունը, սակայն նա չի կարող կատարել նրանցում որևէ փոփոխություն:

Գոյություն ունեն երկխոսական ծառայություններ, որոնք տրամադրում են հայտարարությունների էլեկտրոնային վահանակների դիտարկման իրավունք: Էլեկտրոնային վահանակների միջոցով կարելի է արտագրել տարբեր ֆայլեր, անցկացնել երկխոսություններ, մասնակցել տարբեր խաղերի: BBS-ը ունի նաև էլեկտրոնային փոստի սեփական համակարգ: Հայտարարությունների էլեկտրոնային վահանակների առավել հայտնի տեսակը՝ “Compu Serve” համակարգն է: Վերջինս հնարավորություն է տալիս Ինտերնետի բաժանորդներին օգտվել ցանցի ռեսուրսներից:

1.3.5. Փնտրող համակարգեր

Այն դեպքերում, երբ անհրաժեշտ է լինում ուսումնասիրել որոշակի թեմային վերաբերվող փաստաթղթեր, սակայն հայտնի չէ դրանց հասցեն Ինտերնետում, օգտվողները դիմում են այսպես կոչված “**փնտրող մեքենաների**” օգնության: “Փնտրող մեքենաները” հատուկ Web-սայթեր են, որոնք նշանակված են Ինտերնետում ինֆորմացիա փնտրելու համար:

Այդպիսի համակարգերի անհրաժեշտությունն առաջացավ Ինտերնետի զարգացման և ընդլայնման ընթացքում, երբ ինֆորմացիայի փնտրումը դարձավ լուրջ պրոբլեմ: 1995 թ-ին մի քանի ամերիկյան ուսանողներ միավորեցին իրենց օգտակար ինտերնետ-հղումների հավաքածուները տվյալների մեկ բազայում, ընդ որում բացի web-հասցեներից այն պարունակում էր դրանց թեմատիկայի նկարագրությունը և **բանալիական բառերի** ընտրանքները: Բանալիական բառերի օգնությամբ միարժեքորեն նկարագրվում էր յուրաքանչյուր էջի բովանդակությունը (օրինակ երաժշտական սայթի համար այդպիսիներն կարող են լինել “երաժշտություն”, “աուդիո”, “երաժիշտ” և այլն):

Դրանից հետո նրանք գրեցին հատուկ ծրագիր, որը իրականացնում էր հասցեի փնտրումը ըստ գրանցված բառի և վերադարձնում պատասխանը բոլոր գտնված հղումների ցուցակը պարունակող web-էջի տեսքով: Բացի այդ մշակվեց նաև ծրագիր, որը անընդհատ դիտարկում էր Ինտերնետը, գտնում նոր web-էջեր և գրանցում դրանք բազայում: Այդպիսի ծրագրերը այժմ անվանում

են “փնտրող ռոբոտներ” կամ կատակով՝ “սարդեր”: Այդպես առաջացավ առաջին “փնտրող մեքենան”, որը գոյություն ունի մինչև այժմ և որը կոչվում է “Yahoo!” (<http://www.yahoo.com>):

Այժմ գոյություն ունեն բազմաթիվ այդպիսի համակարգեր, որոնցից կարելի է նշել AltaVista, InfoSeek, Lycos, Google, Yandex, Rambler և այլն:

§1.4. Փաստաթղթաշրջանառության կազմակերպումը WWW-ում

Ներկայումս ինֆորմացիայի մեքենայական կրիչների վրա պահպանվում է միլիարդավոր մեքենագիր էջերի պարունակությամբ համարժեք ինֆորմացիա: Սակայն այդ ինֆորմացիայի մեծ մասը գրանցված է տարբեր տեսակի ֆայլերի տեսքով (այդ թվում նաև տվյալների բազաների) և վերջիններին դիմելու համար պահանջվում է կիրառել ծրագրեր, որոնց օգնությամբ դրանք ստեղծվել են: Այդպիսի բազմատեսակ ինֆորմացիան մեկ տեսակի փաստաթղթում միավորելու և մարդկային ընկալմանը հասանելի դարձնելու համար ստեղծվեցին հատուկ տեսակի էլեկտրոնային փաստաթղթեր՝ այսպես կոչված Web-փաստաթղթեր կամ Web-էջեր:

Ինտերնետում օգտագործվող փաստաթղթերը հիմնականում ստեղծված են **HTML** (Hyper Text Markup Language՝ Տեքստի Գծանշման Հիպեր Լեզու) ֆորմատում: Դա ստանդարտ կոդերի և համաձայնությունների հավաքածու է, որը նշանակված է web-էջերի ստեղծելու և կլիենտական համակարգչի (այսինքն՝ բրաուզերի) էկրանին դրանց արտացոլումը նախապատրաստելու համար: Այդպիսի փաստաթղթերը կազում են web-հանգուցների (սերվերների) հիմնական բովանդակությունը և թույլ են տալիս արտացոլել տեքստ, գրաֆիկա, մուլտիմեդիա, ինչպես նաև Ինտերնետի ռեսուրսների այլ բաղադրիչ մասերը: HTML-ի հիմնական ֆունկցիոնալ առանձնահատկություններից մեկը՝ հիպերհղումների կազմակերպման հնարավորությունն է, ինչի շնորհիվ էլ լեզուն ստացել է անվանումը:

❖ **Հիպերհղումը դա web-էջում պարունակվող որևէ օբյեկտի և այլ փաստաթղթի միջև դինամիկ կապի իրագործման ձևն է:**

Փաստաթղթում կարող է պարունակվել հղում այլ փաստաթղթերին, որոնք կապված են իմաստով, օրինակ տրված տեքստի խորացված հասկացությունը: Դիմումների հետ կարող են կապված լինել նկարներ, ձայնային գլխագրեր, վիդեո հատվածներ: Նկարները կամ նրանց մասերը իրենց հերթին կարող են ընդգրկել դիմումներ տեքստերին, նոր նկարներին կամ ձայներին: Փաստաթղթերը, որոնց հղում է կատարվում կարող են գտնվել

հեռացված համակարգիչների վրա: Դիմումների շղթայով կարելի է զգալիորեն հեռանալ ինֆորմացիայի սկզբնական աղբյուրից, սակայն կարելի է նաև հեշտորեն նրան վերադառնալ: Օրինակ՝ գեղարվեստական պատկերասրահի մասին հոդված կարդալիս անմիջապես կարելի է դիտել նկարները, իսկ ուսումնասիրելով երաժշտական գործիքները, լսել դրանց ձայնը:

Սակայն ստանդարտացման տեսակետից HTML-ի նույնիսկ վերջին չորրորդ տարբերակին բնորոշ է ազատության որոշակի աստիճան (HTML-ին զուգահեռ ստեղծված գծանշման այլ լեզուներից և ոչ մեկը համատեղելի չէր մի քանի պլատֆորմների և, նույնիսկ, ծրագրային փաթեթների հետ):

Էլեկտրոնային փաստաթղթերի գծանշումը ստանդարտացնելու և միջպլատֆորմային դարձնելու նպատակով 1986 թվականին Ստանդարտների Միջազգային Կազմակերպությունը (ISO) ստեղծեց SGML-ը (Standard Generalized Markup Language)՝ “գծանշման ընդհանրացված ստանդարտ լեզուն” (ավելի ստույգ այն պետք է անվանել մետալեզու): Դրա ստեղծումը հնարավոր դարձրեց գծանշման լեզուների ունիֆիկացումը, որը, իր հերթին, թույլ տվեց ապահովել վերջինների ճկունությունն ու օգտվողների հավելվածների և տարբեր օպերացիոն համակարգերի միջև ինֆորմացիայի փոխանակման հնարավորությունը:

SGML-ը կարելի է դիտարկել որպես լեզվի շաբլոն, որը կարելի է լրացնել օպերատորների և հատկանիշերի կոնկրետ արժեքներով: SGML-ի հիման վրա ստեղծված յուրաքանչյուր լեզվի ֆունկցիան կայանում է փաստաթղթի տարբեր էլեմենտների փոխադարձ կապերի որոշման մեջ, այսինքն SGML-ը նկարագրում է փաստաթղթի կառուցվածքը այլ ոչ թե արտաքին տեսքը:

Գծանշման լեզուն բաղկացած է տեքստի ապածակագրելու համար օգտագործվող հրամանների համախմբից: Այն պետք է որոշի՝ ո՞ր գծանշումն է հնարավոր, ո՞րն է անհրաժեշտ և ինչպե՞ս տարբերել այն փաստաթղթի մնացած տեքստից: SGML-ին այդ ամենը բնորոշ է, սակայն նրանում չեն որոշվում գծանշերի անվանումները և արժեքները:

2000 թվականին W3C կամ W3C կոնսորցիումը (որի մեջ մտնում են մասնավորապես Microsoft, Netscape, AOL և AT&T ֆիրմաները) ստեղծեց web-էջերի ստեղծման սկզբնաղբյուրներն նոր ստանդարտ՝ **XML** (Extensible Markup Language) – “Գծանշման Ընդլայնվող Լեզու”: Ըստ կառուցվածքի այն իրենից ներկայացնում է (ինչպես և SGML-ը) ոչ թե գծանշման լեզու, այլ մետալեզու, որը նշանակված է ավելի ցածր մակարդակի լեզուների նկարագրության համար:

Ինչպես նշվեց, web-փաստաթղթերը պահվում են Internet ցանցի web-սերվերներում: Դրանց հետ աշխատելու համար մշակված են բազմաթիվ կլիենտական ծրագրեր, որոնք կոչվում են **web-մեկնաբաններ կամ բրաուզերներ** (browsers): Բրաուզերները թույլ են տալիս կանչել հայտնի հասցեով տեղադրված անհրաժեշտ փաստաթղթերը, կուտակել դրանք, դասակարգել, միավորել, խմբագրել, տպել: Գոյություն ունեն բրաուզերների մի քանի դասեր, որոնք տարբերվում են իրագործվող հնարավորությունների շրջանակներով: Հիմնական երկուսը՝ դասերն են, որոնք սատարում են կամ ոչ web-էջերի գրաֆիկական էլեմենտները: Ժամանակակից բրաուզերնի մեծամասնությունը պատկանում է առաջին դասին: Այդպիսի ծրագրերից առավել ճանաչված են Microsoft Internet Explorer-ը (IE), Netscape Navigator-ը, Opera-ն և այլն: Երկրորդներից կարելի է նշել Unix-համատեղելի Lynx բրաուզերը:

Ներկայումս առավել տարածված են Internet Explorer և Netscape Navigator բրաուզերները, որոնք մշակված են Windows օպերացիոն համակարգի ղեկավարման ներքո աշխատելու համար: Հարկ է նշել, որ տարբեր բրաուզերներում ներկառուցված ինտերպրետատորները միանման չեն աշխատում, ինչի պատճառով միևնույն HTML փաստաթուղթը կարող է արտապատկերվել դրանցում տարբեր տեսքով: Այնուամենայնիվ որոշակի վիճակագրական մշակումներ կատարելուց հետո հնարավոր է դառնում վերացնել կոդի մշակման ալգորիթմների բոլոր էական տարբերությունները:

W3C կոնսորցիումի խնդիրներից մեկը HTML լեզվի սպեցիֆիկացիաների մշակումն է և, քանի որ տեխնոլոգիաները անընդհատ զարգանում են, աշխատանքը նոր ստանդարտների մշակման բնագավառում նույնպես տարվում է անընդհատ: Նոր ստանդարտներին համապատասխան ֆիրմաները պետք է թարմավենն արտադրվող բրաուզերների վարկածները: Հակառակ դեպքում այդ ֆիրմաների ծրագրային արտադրանքը չի արտապատկերի նոր ստանդարտների համապատասխան կազմված փաստաթղթերը, իսկ դա նշանակում է շուկայի կորուստ:

HTML լեզվի վերջին վարկածը (4.01) մշակվել է 1999 թվականին: Դրանից հետո հիմնական ուշադրությունը հատկացվեց լեզվի այն բաղադրամասերի ստեղծմանը, որոնք պետք է համատեղելի լինեն XML-ի ստանդարտի հետ: Այդպիսի ձևափոխության արդյունքում կառուցվեց նոր՝ **XHTML** (Extensible Hyper Text Markup Language) ստանդարտը: Այն քչով է տարբերվում իր նախորդից HTML4.01-ից և, ավելի շուտ, իրենից ներկայացնում է հին ստանդարտների ավելի խիստ ձևակերպված տարբերակը:

WWW-ի համընդհանուր զարգացումը և կիրառումը առաջացրեց տարաբնույթ ինֆորմացիայի մեծածավալ քանակների web-սերվերներում արագ և որակով տեղադրման, ինչպես նաև սերվերներից ինֆորմացիայի պահանջագրման անհրաժեշտությունը: Գծանշման լեզուների և բրաուզերների ստեղծումը համդիսացավ հարցի լուծման միայն մեկ մասը, քանի որ միայն դրանց առկայությունը հնարավորություն չի տալիս կազմակերպել աշխատանքի, այսպես կոչված, ինտերակտիվ ռեժիմը, երբ սերվերը մշակում է կլիենտից ստացված հայտը և, կախված նրանում պարունակվող պահանջներից, մշակում այն ու ուղարկում դինամիկ կերպով ստեղծված պատասխանը (web-էջը):

Այդ ուղղությամբ WWW-ի հնարավորությունները ընդլայնելու համար օգտագործվում են հատուկ՝ **սերվերային ծրագրեր**, որոնք սերվերի վրա մշակում են հաճախորդի կողմից ուղարկված տվյալները և վերադարձնում պատասխանը պատրաստի web-էջի տեսքով:

Իսկ ինպե՞ս է մշակում սերվերը օգտվողներից ստացված տվյալները: Բանն այն է, որ սերվերը ունակ չէ դրանք մշակել: Սերվերի խնդիրը հետևյալն է՝

ընդունել բրաուզերից տարբեր ֆայլերի հարցումները (web-էջերի, ռճերի աղյուսակների, գրաֆիկական պատկերների, արքիվների, ֆիլմերի, երաժշտության և այլն), փնտրել անհրաժեշտ տվյալները սեփական (որոշ դեպքերում FTP-սերվերի) կոշտ մագնիսական սկավառակի վրա, ուղարկել գտնված ֆայլերը կամ ձևավորված պատասխանը հարցում կատարող բրաուզերին:

Սերվերային ծրագրերը աշխատում են սերվերի հետ համատեղ նույն սերվերային համակարգչի վրա: Դրանք չունեն օգտվողի ինտերֆեյս և “հաղորդվում են” միայն սերվերի հետ՝ ընդունում են նրանից օգտվողից ստացած տվյալները և վերադարձնում սերվերին մշակման արդյունքները: Դրանով էլ նրանք տարբերվում կլիենտական ծրագրերից (օրինակ, բրաուզերներից), որոնք աշխատում են անմիջապես օգտվողի հետ: Առավել կարևորն այն է, որ սերվերային ծրագրի կողմից սերվերին վերադարձրած արդյունքը – սովորական HTML կոդ է, այսինքն փաստորեն web էջ, որը ձևավորված է օգտվողից ստացած տվյալների հիման վրա: Ի տարբերություն web-դիզայների ստեղծած և սերվերի վրա ֆայլերի տեսքով պահպանված **ստատիկ** էջերից այդպիսի էջը կոչվում է **դինամիկ**: Հենց այդ՝ դինամիկ էջն է ուղարկվում կլիենտին ի պատասխան նրանից ստացված տվյալների:

Սերվերային ծրագրերը բաժանվում են հետևյալ չորս խմբերի:

- CGI ծրագրեր: Դրանք կատարվող ծրագրեր են, որոնք աշխատում են CGI (Common Gateway Interface) Փոխանակման Ընդհանուր Ինտերֆեյսի միջոցով:
- Web-սերվերի ընդլայնումներ՝ ISAPI, NSAPI, Apache ֆորմատի ծրագրեր, որոնք ներկառուցվում են հենց web-սերվերի մեջ, դարձնելով դրա բաղկացուցիչ մասը:
- Ակտիվ սերվերային էջեր (JSP և ASP):
- Սերվերային սցենարներ, որոնք ստեղծվում են որևէ մեկնաբանվող լեզվով, օրինակ Perl, Python, JavaScript, VbScript և այլն:

CGI ծրագրերը՝ սովորական կատարվող (.exe) ֆայլեր են, ծրագրավորման որևէ լեզվով կազմած և կոմպիլացված: Դրանք աշխատում են հենց սերվերային համակարգչի օպերացիոն համակարգի ղեկավարությամբ և գործարկվում են այն ժամանակ, երբ անհրաժեշտություն է առաջանում մշակել օգտվողների տվյալները: Ընդ որում, եթե սերվերը ստացել է մի քանի հարցում, ապա սկսում են աշխատել CGI ծրագրի համապատասխան քանակով պատճեններ: Այդ տեսակի ծրագրերի արժանիքներից կարելի է նշել ստեղծման և կարգավորման համեմատական հեշտությունը: Բացի այդ, քանի որ դրանք աշխատում են սերվերից անկախ, ապա ծրագրի խաբանման դեպքում կոպիտ քրեական միայն իրեն աշխատանքը իսկ սերվերը կշարունակի աշխատել: Թերություններից նշենք սիստեմային ռեսուրսների մեծ ծախսը՝ միաժամանակ աշխատող պատճենների մեծ քանակի դեպքում սերվերը կարող է «կախվել»:

Web-սերվերի ընդլայնումները՝ սովորական dll (dynamic linked library) գրադարաններ են (նույնպես կոմպիլացվող), որոնց միջոցով իրագործվում է սերվերային ծրագրերի տրամաբանությունը: Դրանք ներկառուցվում են սերվերի ծրագրին և աշխատում որպես վերջինի անբաժանելի մաս: Քանի որ dll գրադարանները աշխատում են միայն Windows միջավայրում, մյուս օպերացիոն համակարգերի համար ստեղծվել են այլ ընդլայնումներ, օրինակ Apache սերվերի ընդլայնումների ֆորմատը կոչվում է Apache: Microsoft ֆիրմայի Internet Information Server (IIS) և Netscape ֆիրմայի Netscape Web Server (NWS) սերվերների ընդլայնումների ֆորմատները համապատասխանաբար՝ ISAPI և NSAPI են: Ընդլայնումների գլխավոր արժանիքը, սիստեմային ռեսուրսների խնայողաբար օգտագործումն է, քանի որ անկախ օգտվողի տվյալների համախմբերի քանակից աշխատեցվում է ընդլայնման ընդամենը մեկ նմուշ: Սակայն դրանց ստեղծումը բավականին դժվար է, պահանջում է ծրագրավորողից մեծ վարպետություն և, բացի այդ

դրանք այնքան էլ անվտանգ չեն ինչպես CGI ծրագրերը, քանի որ աշխատում են որպես սերվերի մի մաս և ցանկացած ծրագրային սխալ կարող է “կախել” սերվերը:

Ակտիվ Սերվերային էջերը սովորական web-էջեր են, որոնք ընդգրկում են նաև սերվերի կամ սերվերային ընդլայնումների կողմից կատարվող հատուկ սերվերային սցենարներ: Մասնավորապես ASP (Active Server Pages), որոնք սատարվում են Microsoft IIS սերվերի կողմից և JSP (Java Server Pages)՝ նաև մի շարք այլ սերվերների կողմից աշխատում են հենց այդ սկզբունքով: ASP էջերը կազմվում են JavaScript և VbScript լեզուներով, իսկ JSP-ն JavaScript-ով: Ակտիվ Սերվերային էջերի արժանիքներն են՝ կառուցման և կարգավորման հեշտությունն ու արագությունը, իսկ թերությունը՝ սիստեմային ռեսուրսներին ներկայացվող բարձր պահանջները և կատարման հարաբերական դանդաղությունը, քանի որ ի տարբերություն արագագործ կոմպիլացված ծրագրերի, որոնք արդեն վերածված են պրոցեսորի մեքենայական կոդերի, ինտերպրետացվող լեզուներով կազմված ծրագրերի յուրաքանչյուր կարգադիր (ինստրուկցիան) ընթերցվում է, վերածվում մեքենայական կոդի և մշակվում միայն կատարման ընթացքում:

Սերվերային սցենարները, ինչպես և ակտիվ սերվերային էջերը ինտերպրետացվող են, սակայն դրանք ընդգրկում են միայն “մաքուր” ծրագրային կոդ: Սովորաբար սցենարները գրվում են հատուկ՝ տեքստի մշակման համար նշանակված ծրագրավորման Perl լեզվով: Գործնականում դրանք կարելի է գրել ցանկացած լեզվով, որն ունի ինտերպրետատոր, օրինակ՝ Python, JavaScript և այլն: Սցենարների արժանիքները և առավելությունները նույնն են, ինչ և ակտիվ սերվերային էջերինը, սակայն սցենարները խլում են չափազանց, նույնիսկ CGI ծրագրերից էլ շատ սիստեմային ռեսուրսներ, քանի որ օգտվողների տվյալների յուրաքանչյուր հավաքածույի մշակման համար գործարկվում է ինտերպրետատորի առանձին պատճեն, որն իր հերթին պահանջում է ռեսուրսների մեծ ծախս:

Սերվերների թեմատիկ բովանդակությունը կարող է տատանվել լայն սահմաններում: Այն կախված է ստեղծման նպատակներից, հնարավորություններից և մի շարք այլ հանգամանքներից: Ցանկացած դեպքում լիարժեք սերվերը պետք է իրենից ներկայացնի ինչպես ընդունված է ասել՝ “ինֆորմացիոն պորտալ”, այսինքն բավականաչափ մեծ վիրտուալ տարածություն, որը բաղկացած է փոքրաչափ թեմատիկ ստորաբաժանումներից կամ ինքնուրույն նախագծերից:

§1.5. Web - էջերի ստեղծման տեխնոլոգիաները:

Web ծառայությանների հիմքում ընկած են **web-կայքերը** (web-site, հետագա շարադրության ընթացքում կանվանենք պարզապես կայքեր)՝ էլեկտրոնային փաստաթղթերի հավաքածուները, որոնք պարունակելով հրապարակվող ինֆորմացիա և գետեղվելով web-server-ներում, մատչելի են դառնում Ինտերնետի օգտվողների համար: Ի տարբերություն սերվերների կայքերը չունեն առանձնացված սերվերային ծրագրեր: Որպես կանոն յուրաքանչյուր կայք հանդիսանում է սերվերի ինտեգրված մի մաս, կատալոգ, չնայած կայքերի մեծամասնությունն ունի սեփական դոմեյնային անուն: Սայթը տարբերվում է սերվերից ևս մեկ պարագայում՝ ինֆորմացիոն բովանդակությամբ: Սայթը (անգլերեն site – հողամաս, տեղամաս) սերվերի մի տեղամասն է, այսինքն բաժինը, որը լիովին նվիրված է որևէ մեկ թեմայի: Իհարկե, գործնականում բոլոր կայքերը ընդգրկում են բազմաթիվ ենթաբաժիններ, որոնք իրենց հերթին կարող են բաժանվել ավելի փոքր բաղկացուցիչ մասերի, սակայն բոլոր դեպքերում դրանց միավորում է որոշակի իմաստային ուղղվածությունը, լրացման ընդհանուր եղանակը:

Ունիվերսալ “բաղադրատոմսեր” կայքի կառուցվածքի վերաբերյալ գոյություն չունեն՝ ամեն ինչ կախված է հեղինակի նպատակներից և դրանք իրականացնելու համար ընտրված մեթոդներից: Որոշակի “ստանդարտ” ձևավորվել է միայն պաշտոնական առևտրային կայքերի վերաբերյալ: Այստեղ պարտադիր է համարվում “Ձեռնարկության մասին” (բերված անունները պայմանական են) էջը, որում տրվում են տեղեկություններ դրա ստեղծման պատմության, գործունեության ուղղվածության, զարգացման ուղիների, նախագծերի վերաբերյալ: “Արտադրատեսակներ/ծառայություններ” էջում նկարագրվում է, թե ինչով է զբաղվում ձեռնարկությունը, տրվում ապրանքների և ծառայությունների ցանկը, առաջարկվում ձեռք բերել կամ պատվիրել որևէ ապրանք կամ ծառայություն: “Աշխատատեղեր” էջում բերվում է թափուր պաշտոնների (պահանջվող մասնագետների) ցուցակը և աշխատանքի ընդունման պայմանները:

Համաաշխարհային Ցանցում գետեղված կամայական ռեսուրս, լինի դա հզոր ինֆորմացիոն պորտալ, որն օրական այցելում են տասնյակ հազարավոր հաճախորդներ, կամ համեստ անհատական էջ (այսպես կոչված **homepage**), որի ամսվա հաճախորդների թիվը սահմանափակվում է երկուսով (հաշված հեղինակին), դա նախ և առաջ արվեստի գործ է, ինժեներադիզայներական խնդիրների լուծման բարդ համալիր: Ցանկացած դեպքում դրանց ստեղծման

պրոցեսում կիրառվում են միանման հնարքներ և տեխնիկական լուծումներ:

Ակնհայտ է, որ յուրաքանչյուր տեխնոլոգիան, յուրաքանչյուր ստեղծագործական պրոցեսը կամայական բնագավառում ենթարկվում են որոշակի օրենքների ու կանոնների և դրանց չհետևելը կարող է հանգեցնել բազմաթիվ անախորժությունների: Սայթը, որպես ինժեներագեղարվեստական լուծումների համալիր նույնպես ակնկալում է մի շարք չափանիշներ, որոնց անհրաժեշտ է հետևել, որպեսզի ստեղծված ռեսուրսն ունենա մասնագիտական տեսք:

Գլխավոր չափանիշը՝ հարմարավետությունն է վերջնական օգտվողի, այսինքն կայքի ապագա այցելուների համար: Քանի որ նրանք քանակը բավականաչափ մեծ է, ընդ որում նրանք օգտագործում են ծրագրային ապահովման և ապարատային միջոցների լայն ամպլանացանկ, ապա անհրաժեշտություն է առաջանում մշակել web-դիզայնի և մշակման ալգորիթմների որոշակի ստանդարտ, որը կբավարարի հնարավոր այցելուների պահանջները, թույլ կտա մաքսիմալ հարմարավետությամբ ընկալել կայքի բովանդակությունը: Շատ դեպքերում դրան կարելի է հասնել հետևելով մի քանի պարզ կանոնների:

- Անհրաժեշտ է ճիշտ կազմակերպել կայքը, այսինքն ապահովել օգտվողին անհրաժեշտ ինֆորմացիայի դյուրին փնտրումը:
- Սայթի ճիշտ կազմակերպումը անբաժանելի է կայքի այսպես կոչված “նավիգացիայի” (էջերով և բաժիններով “երթևեկելու”) հետ, այսինքն անհրաժեշտ է կազմակերպել հասկանալի հղումներ կայքի բոլոր էջերին: Ընդ որում ցանկալի է, որ հղումները միշտ տեղաբաշխված լինեն էկրանի միևնույն մասում:
- Հարկ չէ ստիպել օգտվողներին “նավիգացիան” իրականացնել մուլտիմեդիա տեխնոլոգիաների միջոցով, քանի որ շատ դեպքերում այցելույին կարող է պարզապես չհերիքել համբերություն մուլտիմեդիայի բոլոր էլեմենտների բեռնմանը սպասելուն:
- Եվ, վերջապես, պետք է հաշվի առնել այցելուների համակարգչային կրթվածության մակարդակը: Օրինակ՝ եթե կայքը նվիրված է համակարգչային խաղերին, ապա կարելի է օգտագործել տարբեր զանգակների, ձայնային էֆեկտների, սուլիչների և հղումների հարուստ զինանոց: Հակառակը՝ եթե այն նվիրված է օրինակ սեղանի լուրջ խաղերին (նարդի, դոմինո և այլն) այդպիսի մոտեցումը արդարացված չի լինի, քանի որ “մտավոր” խաղերի սիրահարները չեն սիրում ավելորդ աղմուկ:

Ինտերնետի յուրաքանչյուր ռեսուրս ընդգրկում է միմյանց հետ հիպերհղումներով կապված մի շարք թեմատիկ բաժիններ: Որպես

կանոն, կայքի բոլոր բաժիններին կատարվող հղումները և դրանց բովանդակության կրճատ անոմսները բերվում են առաջին՝ մուտքային էջում, որին սովորաբար շնորհվում է **“index.html”** անունը:

Եթե թեմատիկ բաժինները իրենց հերթին պարունակում են ենթաբաժիններ, ապա բաժիններից յուրաքանչյուրը նույնպես պետք է ունենա մուտքային **“index.html”** էջ: Այդպես խորհուրդ է տրվում անվանել կայքի բոլոր մուտքային էջերը: Հակառակ դեպքում, եթե դիմենք որևէ ենթաբաժնին կրճատ հասցեով՝ առանց համապատասխան էջի անունը նշելու, էկրանին կստանանք այդ ենթաբաժնի (թղթապանակի) ֆայլերի ցուցակը, այլ ոչ թե ինքը էջը: Օրինակ՝ ենթադրենք, որ կայքում մենք ունենք ֆոտոնկարների ենթաբաժին (photos թղթապանակ), որի սկզբնական էջը անվանել ենք ոչ թե index.html, այլ photo.html: Այժմ եթե ենթաբաժնին անցնելու համար հիպերհղումը գրանցենք այսպես՝

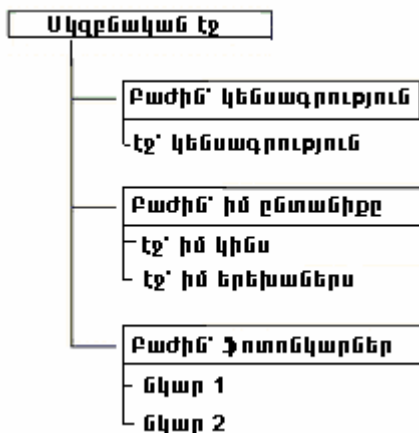
<http://www.oursite.am/photos/>,

այլ ոչ թե

<http://www.oursite.am/photos/photo.html>,

ապա արդյունքում բրաուզերը կարտապատկերի photos թղթապանակում պարունակվող ֆայլերի և թղթապանակների ցուցակը:

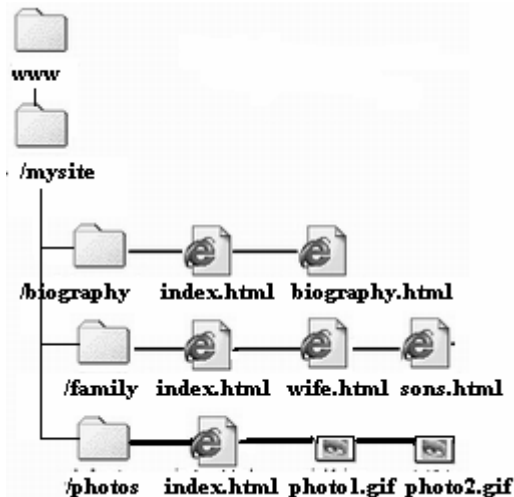
Թեմատիկ բաժինների, դրանցից յուրաքանչյուրին պատկանող փաստաթղթերի և դրանց միջև նախապես նախագծված բոլոր հիպերկապերի մանրամասն ցանկը կոչվում է կայքի տրամաբանական կառուցվածք: Պատկեր 1.5.1-ում ցուցադրված է կայքի տրամաբանական կառուցվածքի օրինակ:



Պատկեր 1.5.1. Կայքի տրամաբանական կառուցվածքի օրինակ

Սայթի ֆիզիկական կառուցվածքը նախատեսում է ֆիզիկական ֆայլերի տեղաբաշխումը ըստ հիմնական (այսպես կոչված արմատային) դիրեկտորիայի ենթադիրեկտորիաների: Ֆիզիկական և տրամաբանական կառուցվածքները կարող են և չհամընկնել, քանի որ ռեսուրսի ֆիզիկական կառուցվածքը մշակվում է ֆայլերի տեղաբաշխման հարմարությունից ելնելով: Սակայն տրամաբանական կառուցվածքի բաժինների հաջորդականության պահպանումը նաև ֆիզիկական կառուցվածքը նախագծելիս թույլ է տալիս բավականաչափ հեշտացնել կայքի հետագա լրացումները և փոփոխությունները: Պատկեր 1.5.2-ում ցուցադրված է պատկեր 1.5.1-ում բերված կայքի տրամաբանական կառուցվածքին համապատասխանող ֆիզիկական կառուցվածքը:

Խորհուրդ է տրվում կայքի նախագծի էլեմենտներ հանդիսացող բոլոր գրաֆիկական պատկերները պահպանել առանձին թղթապանակում (անվանելով այն, օրինակ, “**images**”), որը պետք է տեղադրվի կայքի արմատային դիրեկտորիայում: Դա թույլ կտա փոփոխություններ կատարել մյուս թեմատիկ բաժինների HTML փաստաթղթերում առանց գրաֆիկայի տեղափոխության, օգտագործել միևնույն գրաֆիկական ֆայլերը կայքի բոլոր բաժիններում և, անհրաժեշտության դեպքում, նույնիսկ հեռացնել որոշ դիրեկտորիաներ ամբողջովին:



Պատկեր 1.5.2. Կայքի ֆիզիկական կառուցվածքը

Որպեսզի կայքի բոլոր հիպերհղումներն աշխատեն կորեկտ և բոլոր փաստաթղթերը բացվեն ճիշտ կերպով, անհրաժեշտ է ֆիզիկական կառուցվածքի ստեղծման ընթացքում ղեկավարվել մի քանի պարզ կանոններով:

- Դիրեկտորիաների անունները, HTML և այլ ֆայլերի անունները և ընդլայնումները գրանցվում են միայն լատինական այբուբենի փոքրատառերով:
- Պետք է աշխատել, որ նիշերի քանակը անուններում չգերազանցի ութից:
- Ցանկալի է, որ ֆայլերի և դիրեկտորիաների անունները լինեն “իմաստալից”, քանի որ հետագայում web-էջերի նշանակումը և բովանդակությունը կարող են մոռացվել, եթե նրանք ունենան, օրինակ, հետևյալ տեսքի անուններ՝ 1.html, 2.html և այլն:

Ասվածից ակնհայտ է դառնում, որ կայքի այցելուները կարող են տեսնել միայն դրա տրամաբանական կառուցվածքը, ընդ որում, ընդամենը նավիգացիայի էլեմենտների միջոցով ներկայացված տեսքով: Այստեղից կարելի է եզրակացնել, որ նավիգացիայի համակարգի կառուցվածքը պետք է, եթե ոչ լիովին, ապա մեծագույն չափով համապատասխանի մշակված տրամաբանական կառուցվածքին:

§1.6. Web – հանգույցի մշակման փուլերը

Ծրագրային արտադրանքի նախագծման մեթոդները բազմազան են: Դրանք կարելի է դասակարգել ըստ տարբեր հայտանիշների, որոնցից կարևորագույններն են համարվում՝

- նախագծային աշխատանքների մեթոնայացման աստիճանը,
- մշակման պրոցեսի ընդունված տարբերակը:

Ըստ նախագծային աշխատանքների ավտոմատացման աստիճանի կարելի է առանձնացնել՝

- նախագծման ավանդական մեթոդներ,
- նախագծման ավտոմատացված մեթոդներ:

Ավանդական մեթոդներով նախագծումը նախընտրելի է ոչ շատ աշխատատար, պարզ կառուցվածք ունեցող ծրագրային արտադրանքի համար, որը մեծամասամբ ունենում է կիրառական բնույթ:

Ավտոմատացված նախագծման անհրաժեշտությունն առաջացավ այն ժամանակ, երբ պահանջվեց նվազեցնել նախագծային աշխատանքների վրա կատարվող ծախսերը և կատարման ժամկետները:

Ծրագրերի նախագծումը կարող է հիմնվել տարբեր մոտեցումների վրա, որոնցից առավել տարածվածներից են՝

- ծրագրային արտադրանքի կառուցվածքային նախագծումը,
- առարկայական տիրույթի և նրա լրացումների ինֆորմացիոն մոդելավորումը,
- ծրագրային արտադրանքի օբյեկտակոդմնորոշված նախագծումը:

Կառուցվածքային ծրագրավորման հիմքում ընկած է հաջորդական դեկոմպոզիցիայի և նրա առանձին բաղկացուցիչների նպատակաուղղված կառուցվածքավորումը: Կառուցվածքային նախագծման տիպիկ մեթոդներն են՝

- ծրագրային կոդավորումը և տեստավորումը,
- մոդուլային ծրագրավորումը,
- կառուցվածքային ծրագրավորումը:

Կախված ստեղծվող օբյեկտի կառուցվածքից տարբերվում են ֆունկցիոնալ – կոդմնորոշված մեթոդներ, երբ խնդիրը կամ պրոբլեմը բաժանվում է առանձին, որոշակի ֆունկցիոններ կատարող պարզ բաղադրամասերի, տվյալների կառուցման մեթոդներ:

Ֆունկցիոնալ - կոդմնորոշված մեթոդների կիրառության դեպքում առաջին հերթին հաշվի են առնվում տվյալների մշակման ֆունկցիաները, որոնց համապատասխան որոշվում է ծրագրի առանձին բաղկացուցիչ մասերի կազմը և տրամաբանությունը:

Կառուցվածքային մոտեցման հիմնական շեշտը տվյալների մշակման գործընթացների մոդելավորման բաժանումն է: Տվյալների ինֆորմացիոն մոդելը և կառուցվածքը առավել մեծ նշանակություն ունեն առարկայական տիրույթի ինֆորմացիոն մոդելավորման համար: Այս սկզբունքն ընկած է տվյալների բազաների ղեկավարման համակարգերի հիմքում:

Սկզբնական փուլում կառուցվում են տվյալների ներկայացման տարբեր մակարդակների ինֆորմացիոն մոդելները՝ ինֆորմացիոն-տրամաբանական մոդելը, որը կախված չէ տվյալների մշակման և պահպանման ծրագրային իրականացումից և արտացոլում է տվյալների ինտեգրված կառուցվածքը, դատալոգիական մոդելը (տվյալների մշակման և պահպանման միջավայրի բնութագրիչը):

Ծրագրային արտադրանքի մշակման ավանդական մոտեցումները միշտ ընդգծել են տարբերությունը տվյալների և նրանց մշակման գործընթացների միջև: Ինֆորմացիոն մոդելավորման տեխնոլոգիաները սկզբից ընտրում են տվյալներն ըստ բնութագրիչների, որից հետո բնութագրում են այդ տվյալների գործընթացները:

Կառուցվածքային տեխնոլոգիաները կոդմնորոշված են տվյալների մշակման և նրանց միջև ինֆորմացիոն հոսքերի կապակցող գործընթացները ճշտելուն:

Մշակման գործընթացը կարելի է բաժանել հետևյալ փուլերի՝

1. Տեխնիկական հանձնարարագրերի կազմում, որը պահանջում է՝

- մշակվող ծրագրի պլատֆորմի որոշումը
- ծրագրի ցանցային տարբերակի գնահատումը,
- ծրագրերի մշակման անհրաժեշտության որոշումը՝ այլ ծրագրային պլատֆորմի տանելու համար,
- տվյալների բազաների ղեկավարման համակարգերի ղեկավարմամբ բազաների հետ աշխատանքի հիմնավորումը:

Այս փուլում որոշվում են խնդրի լուծման մեթոդները, մշակվում են համալիր խնդիրների լուծման ընդհանրացված ալգորիթմները, դրանց ֆունկցիոնալ կառուցվածքը կամ օբյեկտային կազմը, ինֆորմացիայի մշակմանը ներկայացվող տեխնիկական պահանջները և շահագործողի ինտերֆեյսը:

2. Տեխնիկական նախագիծ: Այս փուլում՝

- ելնելով ծրագրային արտադրանքի նախագծման ընդունված մոտեցումից մշակվում է ալգորիթմը կամ ճշգրտվում է օբյեկտների կազմը, նրանց հատկությունները, մշակման մեթոդները և այլն,
- որոշվում է ընդհանուր համակարգային ծրագրային ապահովումը՝ ներառելով բազային միջոցները (օպերացիոն համակարգը, տվյալների բազաների ղեկավարման համակարգերի մոդելը, էլեկտրոնային աղյուսակները),
- մշակվում է ծրագրային արտադրանքի ներքին կառուցվածքը՝ առանձին ծրագրային մոդուլների տեսքով,
- իրականացվում է ծրագրային մոդուլների գործիքային միջոցների ընտրությունը:

3. Աշխատանքային փաստաթղթեր (աշխատանքային նախագիծ):

Այս փուլում իրականացվում է ծրագրային ապահովման բազային միջոցների ադապտացիան: Իրականացվում է ծրագրային մոդուլների մշակումը կամ օբյեկտների մշակման մեթոդները՝ ծրագրավորումը կամ ծրագրային կոդի ստեղծումը: Կատարվում է ծրագրային արտադրանքի ինքնուրույն և համալիր հղկումը, բազային ծրագրային միջոցների և մոդուլների աշխատունակության փորձարկումը: Համալիր հղկման համար նախատեսվում է ստուգողական օրինակ, որը ստուգում է ծրագրային արտադրանքի համապատասխանությունը:

Այս փուլի հիմնական աշխատանքային արդյունքը ծրագրային արտադրանքի շահագործման փաստաթղթերի ստեղծումն է (շահագործման նկարագրությունը, ցուցումներ շահագործողին և ծրագրի ինստալյացիայի յուրահատկությունները):

4. Մուտքագրումը և աշխատանքը:

Պատրաստի ծրագրային արտադրանքը անցնում է փորձնական շահագործում, որից հետո միայն ներկայացվում է զանգվածային շահագործման: Web հանգույցի մշակման պլանի նախապատրաստումը ընդգրկում է հետևյալ փուլերը:

Նախագծի բյուջեի որոշումը: Գլխավոր հարցը, որը պետք է լուսաբանվի պլանի առաջին տողերում, նախագծի իրականացման արժեքն է: Հաճախ տեխնիկական առաջադրանքում պատվիրատուն սահմանափակում է դնում նախագծի բյուջեի վրա: Այդպիսի դեպքերում պետք է, ելնելով հատկացվող միջոցների ծավալից, բաշխել ուժերն այնպես, որպեսզի հնարավոր լինի դուրս չգալ բյուջեի շրջանակներից: Եթե այդպիսի սահմանափակում չկա, ապա ծախսերը գնահատվում են ելնելով պլանի ծրագրի և աշխատանքների ծավալից: Ծախսերի նախահաշիվ կազմելու ժամանակ միշտ պետք է ավելացնել ընդհանուր գումարի մոտավորապես 20-25 տոկոսը՝ չնախատեսված ծախսերի համար: Եվ այն դեպքում, երբ կատարված ծախսերը նախատեսվածից քիչ գումար կազմեն, հնարավոր կլինի հաճելի անակնկալ մատուցել պատվիրատուին: Իսկ լրացուցիչ ներդրումներ կատարելու անհրաժեշտության դեպքում կատարողն ինքը շնորհակալ կլինի այդպիսի խնայողության համար: Ուստի պլանում պետք է հստակ նկարագրվեն նախագծի վրա բոլոր նախատեսվող ծախսերը և ենթադրվող շահույթի չափը:

Հանգույցի բովանդակության հիմնական և լրացուցիչ ֆունկցիաների առանձնացումը: Կազմվում է հանգույցի բովանդակության և ֆունկցիաների սխեմատիկ նկարագրությունը՝ կատարողի պատկերացմամբ: Այն պետք է հենվի տեխնիկական առաջադրանքի և պատվիրատուի հետ նախագծի քննարկման արդյունքների վրա: Օրինակ եթե ինտերակտիվ ժամանակային սանդղակը կարող է լինել web-հանգույցի հիմնալի լրացում, ապա պետք է առանձնացրել այն որպես լրացուցիչ ֆունկցիա և նկարագրել թե ինչպես այն պետք է գործի:

Ինտերֆեյսի կազմակերպումը: Հանգույցի բովանդակության և ֆունկցիաների փորձնական կազմումը հենց այն փուլն է, որում պետք է մտածել նրա ինտերֆեյսի կազմակերպման մասին: Հանգույցի հիմնական բաժինների և ենթաբաժինների ցուցակի պատրաստումով արդեն կարելի է պատկերացնել թե ինչպես պետք է գործի նրա ինտերֆեյսը՝ անհրաժեշտ է արդյոք դա ժամանակակից նավարկման համակարգ պատրաստված Flash-ով թե լինելու է ստանդարտ համակարգ:

Իյուստրացիաների պատրաստումը: Պատվիրատուն և, առհասարակ ցանկացած մարդ, սովորաբար առավել լավ է ընկալում

գրաֆիկական պատկերազարդ ինֆորմացիան: Այդ պատճառով ամենուրեք, ուր հնարավոր է, անհրաժեշտ է օգտվել գրաֆիկորեն ձևավորված դիագրամաներից և մրցակից-հանգույցների էջերի պատկերներից:

Աշխատանքային գրաֆիկի կազմումը: Անհրաժեշտ է պլանում ընդգրկել մշակման գրաֆիկի հսկիչ կետերը, այսինքն այն փուլերը որոնցում նախագիծը ստուգվում է պատվիրատուի կողմից: Պետք է թվարկվեն նաև այն խնդիրները, որոնց համար պատասխանատու է պատվիրատուն և նշվեն նրանց կատարման ժամկետները: Հսկիչ կետերում պատվիրատուն ընդունում է նախագծի ավարտում մասը և դրանից հետո նա իրավունք չունի պահանջել փոփոխություններ մտցնել առանց լրացուցիչ վճարման:

Կարևոր է նաև հենց սկզբից պարզել նախագծի մշակման ավարտի ցանկալի ժամկետը: Շատ դեպքերում պատվիրատուն ինքն է որոշում ժամկետները, որոնց վրա հենվելով կարելի է պլանավորել աշխատանքները: Իհարկե ժամկետները պետք է որոշված լինեն առողջամիտ: Օրինակ՝ եթե նախագիծը պետք է պատրաստ լինի մեկ ամսվա ընթացքում, ապա նույնիսկ մեծագույն ցանկության դեպքում այդ ժամկետում հնարավոր չէ մշակել սոլիդ web հանգույց:

Այն փուլերը, որոնց համար պատասխանատու է պատվիրատուն կարևոր դեր են խաղում դիզայների պլանում: Հսկիչ կետերը շատ կարևոր են, քանի որ դրանք թույլ են տալիս ավարտել մշակման մեկ փուլը, հաստատել որ պատվիրատուն ընդունել է մինչ այդ կետը կատարված բոլոր աշխատանքները և անցնել հաջորդ փուլին:

Հիմնականում պատվիրատուի պատասխանատվությունը կայանում է նրանում, որ նա պետք է տրամադրի մշակողին հանգույցի էջերի բովանդակությունը լրացնելու համար անհրաժեշտ տեքստերը, գրաֆիկական պատկերները և մուլտիմեդիայի ֆայլերը: Չափազանց կարևոր է պլանում գրանցել դրանց հատկացման ժամկետները, քանի որ մշակման ժամկետները պատվիրատուի մեղքով խախտվելու դեպքում այդ գրանցումները արդարացման հիմք կհանդիսանան մշակողի համար: Մշակողի և պատվիրատուի պարտավորությունների ճիշտ սահմանազատումը կօգնի հետագայում խուսափել տհաճ իրավիճակներից:

Շուկայի հետազոտումը և հանգույցի մրցունակության որոշումը: Որոշ նախագծերի համար այդ փուլը կարող է ավելորդ լինել: Սակայն առևտրային կառույցների (օրինակ՝ e-խանութ) հանգույցների նախագծման դեպքերում, օգտակար է այցելել մնամատեսակ

հանգույցներ և հանդգվել՝ արդյո՞ք առաջարկվող դիզայնը մրցունակ է շուկայում:

Դիզայների ծառայությունների գնահատումը: Միշտ դժվար է որոշել սեփական աշխատանքի արժեքը: Կարելի է օգտագործել աշխատանքի վարձատրության տարբեր համակարգեր, օրինակ ժամավարձով որի չափը որոշվում է տարվա ընթացքում վերլուծելով ծախսերը, եկամտի չափը կամ ամսական աշխատավարձը: Բայց ամենագլխավորը չկորցնել չափը և իրատեսորեն նայել իրերին: Օրինակ ազատ դիզայների ժամավարձի որոշման համար անհրաժեշտ է հետևյալը.

- Այլ դիզայներների ծառայությունների վճարման չափի որոշումը: Պետք է իմանալ այլ դիզայներների ծառայությունների չափը և որոշել դիզայների ծառայության արժեքի իրական մակարդակը:
- Պետք է օբյեկտիվորեն գնահատել սեփական վարպետության և փորձի մակարդակը: Փորձառու web-դիզայները, որը մեկ տարի չէ որ աշխատում է բիզնեսում և ունի մի շարք հզոր նախագծերի մշակման փորձ, հստակ գիտի թե ինչի համար են իրեն վճարում և հավակնում է վճարման համապատասխան մակարդակի: Համապատասխան փորձ չունեցող սկսնակ web-դիզայները կարող է հավակնել միայն վճարման միջին մակարդակի:
- Սեփական ծախսերի գնահատումը: Անհրաժեշտ է հաշվարկել թե ի՞նչ կարժենա ամենամսյա աշխատանքը, ինչքա՞ն կպահանջվի աշխատավարձի, գովազդի և հարկերի վճարման համար:
- Անհրաժեշտ է մտածել թե ի՞նչ է պետք ձեռք բերել աշխատանքի համար՝ էլեկտրականություն, օֆիսային պիտույքներ, համակարգիչներ, ծրագրային ապահովում, ինտերնետին դիմելու հնարավորություն, համապատասխան թեմատիկայով գրականություն և այլն: Կազմվում է բյուջեն ամսվա կտրվածքով և որոշվում, թե ինչքա՞ն է պետք վաստակել ծախսերը փակելու համար:

Նախագծի ընդհանուր արժեքի հաշվարկների մեջ առավել դժվար է աշխատանքի ծավալի և նախագծի վրա ժամանակի ծախսերի իրական գնահատումը:

ԳԼՈՒԽ 2. HTML ԳԾԱՆՇՄԱՆ ԼԵԶՈՒՆ

HTML լեզուն ստեղծվել է գիտական և ուսումնական ինֆորմացիայի փոխանակման նպատակներով այն տարիներին, երբ Ինտերնետից օգտվում էին միայն գիտական և կառավարական հիմնարկությունները: Ժամանակի ընթացքում, երբ լեզվի կիրառության բնագավառը ընդլայնվեց՝ կրթություն, զվարճանքներ, կոմերցիա և այլն, այսինքն երբ առաջացավ վիզուալ լավ ձևավորված դյուրին ընկալվող web-էջերի անհրաժեշտությանը, պարզ դարձավ, որ լեզվի ընթացիկ վարկածը չի համապատասխանում աճող պահանջներին: Այդ պատճառով web-մեկնաբաններ (հատկապես Internet Explorer և Netscape) արտադրող ֆիրմաները սկսեցին ավելացնել սեփական ոչ ստանդարտ հրամաններ, որոնք հնարավորություն տվեցին բրաուզերներին արտապատկերել պատկերազարդ, վառ, գրավիչ էջեր:

Սակայն շատ շուտով web-դիզայներների պահանջների և web-մեկնաբանների հնարավորությունների միջև առաջացան համատեղելիության պրոբլեմներ և, ինչպես արդեն նշել էինք, 2000 թվականին W3C կոնսորցիումը ստեղծեց XML ստանդարտը, որը հիմք հանդիսացավ գծանշման լեզվի մշակվող բոլոր տարբերակների համար: Որպես անցումային ստեղծվեց XHTML գծանշման լեզուն, որի ընթացիկ վարկածը քչով է տարբերվում HTML4.01-ից (HTML-ի վերջին վարկածից): Այն պարզապես նման է հին ստանդարտների ավելի խստապահանջ տարբերակի և պահանջում է web-դիզայներներից ծրագրային կոդը ավելի “կոկիկ” ձևակերպում:

Կարող է թվալ, որ բոլոր դեպքերում անհրաժեշտ է կիրառել նորագույն ստանդարտը, սակայն իրականում նույնիսկ ստանդարտի ներքո գոյություն ունեն երկու տարբերակներ՝ խիստ և անցումային: Բանն այն է, որ ոչ բոլոր բրաուզերներն են կարողանում արտապատկերել էջերը նոր ֆորմատում: Իհարկե օգտվողների մեծամասնությունը “թարմացնում է” ծրագրային և ապարատային ապահովումները, սակայն դեռ գոյություն ունեն բավականաչափ քանակության հին մեքենաներ, որոնց վրա տեղադրված է հին ծրագրային ապահովում և դրանք “անտեղյակ են” XHTML-g:

§2.1. HTML փաստաթղթերի կառուցվածքը և հիմնական տեգերը

Ինչպես և ցանկացած այլ ծրագրավորման լեզու HTML-ը նախատեսում է ծրագրի, մեր դեպքում HTML-փաստաթղթի, որոշակի ստանդարտացված կառուցվածք: HTML-փաստաթղթում այդպիսի կառուցվածքը նկարագրում է ոչ թե հրամանների, այլ անմիջա-

կանորեն ծրագրային կոդ պարունակող պարտադիր բլոկների դասավորության հաջորդականությունը: Ի տարբերություն ծրագրավորման այլ լեզուների HTML-ի դիրեկտիվները չեն անվանում “հրամաններ”, “պրոցեդուրաներ” կամ “օպերատորներ”: Դրանք ունեն սեփական անվանում՝ “**տեգ**”-եր (անգլերեն “tag” - գծանիշ): Տեգ-ը իրենից ներկայացնում է անկյունային փակագծերի մեջ վերցված որոշակի բանալիական բառ՝ դեկլարատոր: Տեգի գրանցման եղանակը ընդհանուր դեպքում կարելի է ներկայացնել հետևյալ տեսքով՝

<տեգ>:

Բոլոր այն օբյեկտները, որոնք վերցված չեն անկյունային փակագծերի մեջ, ինտերպրետատորը (ծրագրի վերծանիչը) հասկանում է որպես տեքստային էլեմենտ և արտապատկերում համակարգչի էկրանին այնպես “ինչպես որ կան”:

❖ Այսպիսով՝ տեգը դա HTML-ի որոշակի հրաման է, որը թելադրում է բրաուզերի ինտերպրետատորին, թե ինչպես այն պետք է մշակի յուրաքանչյուր կոնկրետ դիրեկտիվին համապատասխանող արժեքը:

Այդ դիրեկտիվները կոչվում են տեգի ատրիբուտներ (բնութագրիչներ): Յուրաքանչյուր ատրիբուտ ունի անուն և արժեք: Օրինակ, քանի որ յուրաքանչյուր աղյուսակ ունի որոշակի լայնություն (width) ապա աղյուսակի ստեղծման համար նախատեսված <table> տեգում կարելի է նախատեսել համապատասխան ատրիբուտը՝

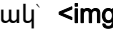
<table width="100">: Այստեղ width="100" արտահայտությունը <table> տեգի ատրիբուտն է, որի անունն է՝ **width**, իսկ արժեքը հավասար է 100 պիքսելի: Այդպիսի գրանցումը նշանակում է, որ էկրանին պետք է արտապատկերվի 100 պիքսել լայնություն ունեցող աղյուսակ: Համաձայն XML ստանդարտի բոլոր տեգերի և ատրիբուտների անունները պետք է գրանցվեն փոքրատառերով, իսկ ատրիբուտների արժեքները պետք է վերցվեն զույգ չափերտների մեջ: Հետագա շարադրությունում մենք կդեկավարվենք այդ կանոններով:

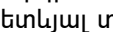
HTML-ն ունի ևս մեկ առանձնահատկություն, որը տարբերում է նրան ծրագրավորման այլ լեզուներից՝ գործնականում լեզվի բոլոր տեգերը, որոշ բացառությամբ, “զույգ են”, այսինքն բաղկացած են “բացող” և “փակող” մասերից (դրանք կոչվում են “տեգ-կոնտեյներներ”): Փակող տեգը տարբերվում է բացողից “ / ” սիմվոլի առկայությամբ: Այն ամենը ինչ գտնվում է այդպիսի տեգերի միջև մշակվում է ըստ տվյալ տեգին բրաուզերում շնորհված ալգորիթմի: HTML-ի հրամանային տողը ընդհանուր տեսքով կարելի է ներկայացնել հետևյալ կերպով՝

<տեգ>մշակվող (արտապատկերվող) արժեք</տեգ>:

Լեզվի այդպիսի հատկությունը թույլ է տալիս կիրառել մեկ տեգը մյուսի մեջ ներդնելու սկզբունքը, այսինքն այդ դեպքում ներքին (ներդրված) տեգի մշակման արդյունքը արտաքին տեգի համար դառնում է մշակվող արժեք: Բերենք տեգերի ներդրման պարզագույն օրինակ՝

<տեգ1><տեգ2>մշակվող արժեք</տեգ2></տեգ1>:

“Եզակի” տեգերը, ի տարբերություն “գույգերի”, չունեն փակող բաղադրիչ: Դրանք հիմնականում օգտագործվում են էջում գրաֆիկական էլեմենտներ (նկարներ, բաներներ), երաժշտություն և որոշ ղեկավարման էլեմենտներ (կոճակներ, տեքստային դաշտեր) ներդնելու համար: Օրինակ՝  տեգի օգնությամբ էջում ներդրվում է պատկեր (image), որի “սկզբնաղբյուրը” (src՝ անգլերեն source - սկզբնաղբյուր) **MyImage** անունով և gif ընդլայնումով պատկերը նկարագրող ֆայլն է:

Համաձայն նոր՝ XML ստանդարտի նույնիսկ այդպիսի տեգերը պետք է ունենան “փակելու” փաստը ազդարարող էլեմենտ: Այդ նպատակով կենտ տեգում՝ փակող անկյունային փակագծից առաջ դրվում է “ / ” սիմվոլը: Իսկ որպեսզի հին web-մեկնաբանները անտեսեն այդ սիմվոլը դրա արջև պարտադիր կերպով դրվում է “պրոբել”: Պատկերի ներդրման հրամանային տողը այդ կանոնի համաձայն կընդունի հետևյալ տեսքը՝ :

- **XHTML-ի կոդի հետ աշխատելիս անհրաժեշտ է ղեկավարվել մի պարզ օրենքով՝ եթե ծրագրի տեքստում հանդիպում է բացող տեգ, ապա պարտադիր կերպով պետք է առկա լինի նաև դրան փակող տեգը (կամ փակող սիմվոլը):**

Գծանշման լեզվի նախկին վարկածները կախված չէին տառերի գրանցման ռեգիստրից, այսինքն տեգերի և ատրիբուտների անունները կարելի էր գրանցել ինպես մեծատառերով այնպես էլ փոքրատառերով (իհարկե՝ դա չի վերաբերվում ատրիբուտների արժեքներին): XML ստանդարտում դա այդպես չէ՝ բոլոր տեգերը և ատրիբուտները պետք է գրանցվեն փոքրատառերով (ստորին ռեգիստրում): Բացի այդ չեն թույլատրվում դատարկ (արժեք չունեցող) ատրիբուտներ: Օրինակ նախկինում checked ատրիբուտը օգտագործվում էր առանց արժեքի, իսկ ըստ նոր ստանդարտի այն պետք է գրանցվի հետևյալ տեսքով՝ checked = “checked”:

Ինչպես և կամայական փաստաթուղթ, HTML փաստաթուղթը բաղկացած է երկու հիմնական մասերից՝

- վերնագրային, “գլխային” մաս (<head>...</head> տեգերի միջև), որը ընդգրկում է սպառիչ ինֆորմացիա փաստաթղթի վերաբերյալ (այդ թվում փաստաթղթի արտաքին անունը՝ **title**, որն

արտապատկերվում է բրաուզերի վերնագրային տողում) և, որոշ դեպքերում, տրանսլատորի հատուկ դիրեկտիվները՝ **<meta>** տեգերը, որոնք հուշում են բրաուզերում ներկառուցված HTML վերձանիչին այն կանոնները, ըստ որոնց պետք է մշակվի web-էջը կազմավորող կոդը;

- փաստաթղթի “մարմին” (**<body>...</body>** տեգերի միջև), որտեղ պարունակվում է այն ամենը, ինչը մենք ցանկանում ենք արտապատկերել էկրանին՝ տեքստը, իլյուստրացիաները, նավիգացիայի և ղեկավարող (commons) էլեմենտները և այլն:

Իսկ որպեսզի ամբողջ աշխարհը հասկանա, որ գործ ունի HTML փաստաթղթի հետ անհրաժեշտ է **<head>** և **<body>** մասերը եզրափակել համապատասխան էլեմենտի ներքո: Այդպիսին է հանդիսանում **<html>...</html>** տեգը:

Այսպիսով, էջի կոդը կարելի է ներկայացնել հետևյալ տեսքով՝

```
<html>
<head>
<title>Փաստաթղթի անունը</title>
</head>
<body>
</body>
</html>
```

HTML-ով գրված կամայական web-էջ իրենից ներկայացնում է սովորական տեքստային ֆայլ: Դա նշանակում է, որ web-էջերի ստեղծման համար կարելի է օգտագործել ցանկացած տեքստային խմբագրիչ, մասնավորապես Notepad-ը, կամ, նույնիսկ, տեքստային պրոցեսորներ (օրինակ՝ Word-ը): Ձևակերպելով համապատասխան կոդը խմբագրիչում և պահպանելով այն հիշողության մեջ “.htm” (հին օպերացիոն համակարգերի համար) կամ “.html” (նորերի) ընդլայնումով, կստանանք HTML փաստաթուղթ:

XML (ներկա փուլում՝ XHTML) ստանդարտը ավելի ու ավելի ամուր դիրքեր է գրավում համաշխարհային ցանցում: Բացի այդ ընդլայնվում է նաև web-ծառայություններից օգտվելու համար կիրառվող սարքավորումների և հավելվածների տեսականին (օրինակ՝ Ինտերնետին կարելի է միանալ բջջային հեռախոսների միջոցով): Այդ ամենը բերում է փաստաթղթի տեսակի խիստ բնորոշման անհրաժեշտությանը: Ոչ այնքան հեռու ապագայում ցանկացած web-փաստաթուղթ պարտադիր կերպով պետք է ունենա այսպես կոչված **DTD**-ն (Document Type Definition - փաստաթղթի տեսակի սահմանում) ըստ որի ընդունող սարքավորումը կհասկանա, թե ինչպե՞ս այն վերձանել:

Որպես գաղտնիք, ասենք, որ ժամանակակից բրաուզերների մեծամասնությունը առանց դրա էլ հիանալի կերպով կարող է արտապատկերել web-էջը, սակայն ժամանակի ընթացքում փաստաթղթի տեսակի որոշումը կունենա ավելի ու ավելի կարևոր նշանակություն:

HTML4.01 վարկածի համար DTD-ն ունի հետևյալ տեսքը՝
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">,
իսկ XHTML1.0-ի անցումային (transitional) վարկածի համար հետևյալը՝
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">:

Այդ սահմանումը գրանցվում է անմիջապես փաստաթուղթը բացող <html> տեգի առջև, ընդ որում նշված վարկածների համար այդ տեգը չունի ատրիբուտներ: XHTML-ի խիստ (strict) վարկածը կիրառելիս (այսինքն XML ստանդարտին հետևելիս) անհրաժեշտ է <html> բացող տեգը փոխարինել հետևյալ սահմանումով՝

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">:

Ինչպես տեսնում ենք նոր ստանդարտում <html> տեգը արդեն ունի մեկ ատրիբուտ՝ *xmlns*, որի միջոցով փաստաթղթին միացվում է XML անունների բազմությունը:

Քանի որ առայժմ առավել հաճախ կիրառվում է HTML4.01 ստանդարտը (դեռ ոչ բոլոր օգտվողներն ունեն նոր ստանդարտը վերծանող բրաուզերներ), ապա կարելի է web-էջեր ստեղծելիս օգտվել առաջին սահմանումով:

Հաճախ անհրաժեշտ է ծրագրի ստեղծման ընթացքում գրանցել որոշակի մեկնաբանություններ, որոնք հետագայում կօգնեն խմբագրման կամ փոփոխությունների կատարելու դեպքում ավելի հեշտ հասկանալ, թե ի՞նչ է կատարվում տվյալ ծրագրային բլոկում, փաստաթղթի ո՞ր մասն է այն նկարագրում և, առհասարակ ի՞նչ է պետք դրա հետ անել: Այդ նպատակին ծառայող **comment** էլեմենտը որոշ չափով տարբերվում է մյուս տեգերից, քանի որ դրա բացող՝ **<!--** և փակող՝ **-->** մասերը պարունակում են միայն միակողմանի անկյունային փակագծեր: Այն ամենը ինչ գտնվում է այդպիսի նշանների միջև անտեսվում է բրաուզերի կողմից:

Իմի բերելով ասվածը ստեղծենք Notepad խմբագրիչում HTML փաստաթղթի շաբլոն և պահպանենք այն որպես սովորական տեքստային ֆայլ (անվանելով օրինակ՝ *template.txt*),

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01

```

Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Փաստաթղթի անունը</title></head>
<body></body>
</html>

```

Ամեն անգամ, երբ անհրաժեշտ լինի ստեղծել նոր web-փաստաթուղթ, մենք կարող ենք բացել այդ ֆայլը, խմբագրել այն և, ընտրելով File ցանկի Save As կետը, շնորհիվ ֆայլին ընտրած անունը և “.html” ընդլայնումով պահպանել համապատասխան թղթապանակում (կամ դիրեկտորիայում):

§2.2. Տեքստի գծանշումը և կազմակերպումը HTML փաստաթղթերում

Տեքստային խմբագրիչներում տեքստի գծանշումը կարելի է իրականացնել ստեղծաշարի միջոցով: Օրինակ՝ նոր տողին անցնելու համար օգտագործվում է “Enter”, բառերի միջև հավասար հեռավորություն պահպանելու համար “Tab” (տաբուլյացիայի), իսկ տարբեր հեռավորություններ՝ պրոբելի ստեղծները: Սակայն բրաուզերները անտեսում են ստեղծաշարի միջոցով կատարված գծանշումը, վերծանելով և տողափոխությունը, և տաբուլյացիան, և պրոբելները (անկախ քանակից) որպես մեկ պրոբել: Այդ պատճառով հիմնական տեքստը web-էջում գծանշելու նպատակով օգտագործվում են հատուկ տեգեր, որոնք կարելի է բաժանել երկու հիմնական խմբերի՝ տեքստի ֆիզիկական ոճի գծանշման տեգեր, տեքստի տրամաբանական ոճի գծանշման տեգեր:

2.2.1. Տեքստի ֆիզիկական ոճի գծանշում

Տեքստի ֆիզիկական ոճի գծանշման որոշ տեգերը և նրանց նշանակումը բերված են աղյուսակ 2.2.1-ում:

Տեքստի ֆիզիկական ոճի գծանշման եղանակը շատ պարզ է՝ ընտրվում է տեքստի այն մասը, որը մենք ցանկանում ենք այս կամ այն ձևով առանձնացնել և վերցվում համապատասխան տեգի մեջ: Հետևյալ ծրագրային կոդում ցուցադրված է ֆիզիկական ոճի գծանշման տեգերի կիրառության եղանակը: Ծրագրի արտապատկերման արդյունքը բերված է պատկեր 2.2.1-ում:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Phisical style formatting example</title>
</head>
<body>

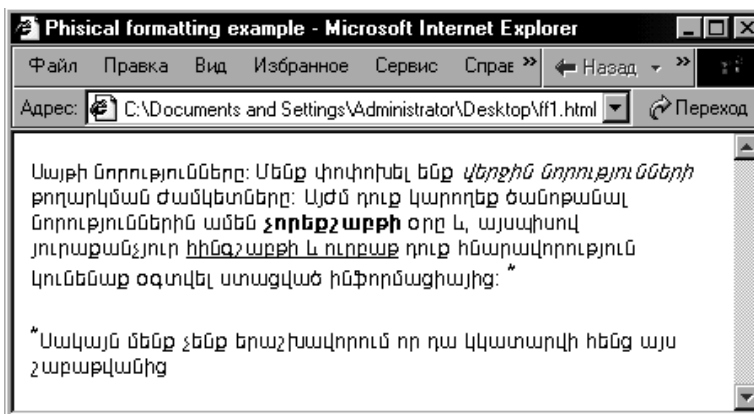
```

<p><tt>Սայթի նորությունները: </tt> Մենք փոփոխել ենք <i>վերջին նորությունների</i> թողարկման ժամկետները: Այժմ դուք կարողեք ծանոթանալ նորություններին ամեն չորեքշաբթի օր և, այսպիսով յուրաքանչյուր <u>ս>հինգ-շաբթի և ուրբաթ</u> դուք հնարավորություն կունենաք օգտվել ստացված ինֆորմացիայից: ^{*}</p>
<p>^{*}Սակայն մենք չենք երաշխավորում որ դա կկատարվի հենց այս շաբաթվանից</p>
</body></html>

Աղյուսակ 2.2.1.

Ֆիզիկական ոճի գծանշման տեղերը

Տեղը	Նշանակումը
, 	Թավ
<i>, </i>	Շեղ (իտալիկ)
<tt>, </tt>	Հավասարալայն
<u>, </u>	Ընդգծված
<big>, </big>	Մեկ միավորով մեծացված չափի
<small>, </small>	Մեկ միավորով փոքրացված չափի
_,	Ստորին ինդեքս
[,]	Վերին ինդեքս
<acronym>, </acronym>	Ակրոնիմ



Պատկեր 2.2.1. Ֆիզիկական ոճի գծանշման տեղերի օրինակ

2.2.2. Տեքստի տրամաբանական ոճի գծանշում

Բերված ծրագրային կոդում բացի ֆիզիկական գծանշման տեգերից օգտագործված է <p>...</p> տեգը, որը ծառայում է հիմնական տեքստը պարբերությունների բաժանելու համար: Այն փաստորեն փոխարինում է Enter ստեղծը, որի միջոցով իրականացվում է անցում նոր տողին: Տեքստի այն մասերը, որոնք եզրափակված են տեգում կազմում են առանձին պարբերություններ: Տեգը չի որոշում տառաշարի չափսերը և տեքստի գրանցման խորությունը (յուրաքանչյուր բրաուզեր այդ պարամետրերը վերծանում է յուրովի): Դա նշանակում է, որ <p> տեգը հանդիսանում է տեքստի տրամաբանական ոճի գծանշման էլեմենտ և նշանակված է տեքստը ավելի կազմակերպված տեսքի բերելու՝ տրամաբանական բլոկների (պարբերությունների) բաժանելու համար:

Նույն նպատակներին են ծառայում տրամաբանական ոճերը որոշող տեգերը: Տրամաբանական է համարվում այն ոճը, որի կոնկրետ պարամետրերը որոշում է բրաուզերը: Հիմնականում բոլոր բրաուզերները տրամաբանական ոճերի գծանշման տեգերը ընկալում են միանման, սակայն կարող են լինել բացառություններ: Բոլոր այդ տեգերը կոնտեյներներային են, այսինքն ունեն բացող և փակող մասեր: Տրամաբանական ոճերի գծանշման հիմնական տեգերի ցուցակը բերված է աղյուսակ 2.2.2-ում:

Տրամաբանական ոճերի գծանշման տեգերի օգտագործման օրինակը բերված է հետևյալ ծրագրային կոդում, իսկ դրա արտապատկերման արդյունքը՝ պատկեր 2.2.2-ում:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  
<html><head><title>Logical formatting example</title>  
</head>  
<body>  
<p><em>Բոլոր օգտվողները</em> պետք է հիշեն այն մասին, որ  
անհրաժեշտ է կորեկտ ձևով ավարտել աշխատանքը <strong>տուն  
զնալուց առաջ:</strong> Սեանսը <dfn>ավարտելու համար</dfn>  
անհրաժեշտ է հավաքել հրամանային տողում <kbd>logout</kbd>  
կամ <kbd>exit</kbd>: Երբ տեսնենք էկրանին այդ հրամանի  
կատարման արդյունքը՝ <samp>Thank you.Goodbye.</samp>,  
կարողեք հանգիստ անջատել համակարգիչը</p>  
</body>  
</html>
```

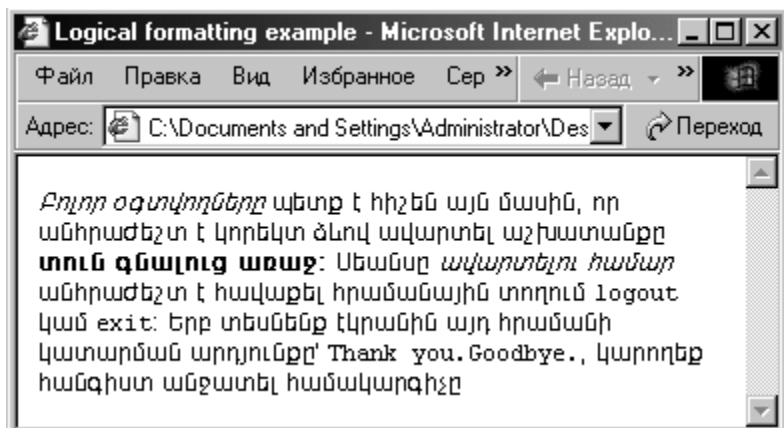
Հաճախ պարբերության ներքո անհրաժեշտ է լինում գծանշել տողի ավարտը, օրինակ, այն դեպքերում, երբ անհրաժեշտ է գրանցել որևէ հասցե: Քանի որ բրաուզերը չի հասկանում ստեղծա-

շարի միջոցով կատարած գծանշումը, ապա անհրաժեշտ է դրա համար դիմել գծանշման այլ եղանակներին:

Աղյուսակ 2.2.2.

Տրամաբանական ոճի գծանշման տեղերը

Տեղը	Նշանակումը
, 	Առանձնացում
, 	Խիստ առանձնացում
<cite>, </cite>	Մեջբերում կամ հղում արտաքին սկզբնաղբյուրին
<dfn>, </dfn>	Ծրագրի սկզբնական կողը
<samp>, </samp>	Ծրագրի աշխատաքի օրինակ (հաճախ արտապատկերվում է նախորդ տեղին նման)
<kbd>, </kbd>	Ստեղծարարից մուտքագրված տեքստ
<var>, </var>	Փոփոխական կամ արժեք
<q>, </q>	Ցիտվող տեքստ
<acronym>, </acronym>	Ակրոնիմ



Պատկեր 2.2.2. Տրամաբանական ոճի գծանշման տեղերի կիրառության օրինակ

Դիցուկ մենք ցանկանում ենք գրանցել հասցեն հետևյալ ձևով՝
 Հովհաննեսյան Հովհաննես
 375001, ք.Երևան,
 փ.Արսլան, շ.14, բն. 14:

Եթե պարզապես հավաքենք այդ տեքստը պարբերության մեջ տողափոխության համար օգտագործելով Enter ստեղծել էկրանին այն կարտապատկերվի մեկ տողով: Եթե փորցենք յուրաքանչյուր տողը ներկայացնել որպես պարբերություն՝

<p>Հովհաննեսյան Հովհաննես </p>

<p>375001, ք.Երևան, </p>

<p>փ.Արծվյան, շ.14, բն. 14</p>,

արդյունքում նույնպես չենք ստանա արտապատկերման ցանկալի տեսքը: Այդպիսի դեպքերի համար գոյություն ունի հատուկ՝ եզակի
 (break row) տեգը, որը տեղադրվում այն դիրքում, որտեղ մենք ցանկանում ենք կատարել տողափոխությունը: Հետևյալ ծրագրային կոդի միջոցով մենք կարող ենք հասնել ցանկալի արդյունքի՝

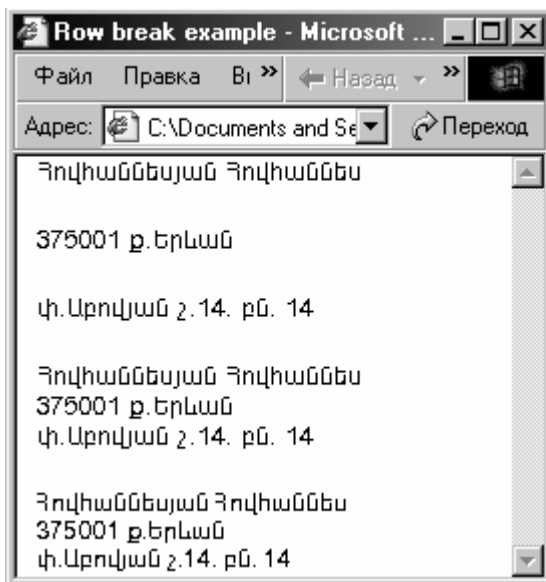
<p>Հովհաննեսյան Հովհաննես

375001, ք.Երևան,

փ.Արծվյան, շ.14, բն. 14

</p>:

Բերված վերջին երկու եղանակների կիրառության արդյունքը ցույց է տրված պատկեր 2.2.3-ում:



Պատկեր 2.2.3.
 և <pre> տեգերի կիրառության օրինակ

Այդպիսի դժվարությունները կարելի է շրջանցել օգտագործելով պարբերության ստեղծման մի այլ տեղ՝ <pre></pre> (preformatted նախագծանշված բառից): Այդ տեղի ներսում գրանցված տեքստը արտապատկերվում է նույն տեսքով, ինչպես հավաքվել է տեքստային խմբագրիչում: Օրինակ՝

<pre>

Հովհաննեսյան Հովհաննես

375001, ք.Երևան,

փ.Արմյան, շ.14, բն. 14

</pre>

պարբերությունը կարտապատկերվի նույն տեսքով, սակայն հավասարալայն տառաշարով (տես՝ պատկեր 2.2.3.):

2.2.3. Տեքստի կազմակերպումը (վերնագրեր, հորիզոնական գծեր)

Հիմնական պարբերությունները էջում տեղադրելուց հետո, տեքստի ընկալումը բարելավելու նպատակով անհրաժեշտ է բերել այն ավելի կազմակերպված տեսքի: Դա կատարվում է երկու տարբեր տեղերի օգնությամբ՝ տարբեր մակարդակների վերնագրերի և հորիզոնական գծերի, որոնք թույլ են տալիս կատարել տեքստի տեսողական բաժանումը: Վերնագրերը, հանդիսանալով կոնտեյներներ ի տարբերություն մյուս տեղերի ինքնաառանձնացվում են (ինչպես և <p> տեղը), այսինքն վերնագրի փակող տեղը տեսնելիս բրաուզերը կատարում է պարտադիր կրկնակի տողադարձ: Տեղադրենք հետևյալ ծրագրային կոդը՝

<h1>Առաջին մակարդակի վերնագրերը ամենագլխավոր
վերնագրերն են</h1>

<h2>Երկրորդ մակարդակի վերնագրերն էլ ոչինչ, կարելի է
օգտագործել որպես ենթավերնագրեր</h2>

<h3>Երրորդ մակարդակի վերնագրերը երկրորդից փոքր են
կարելի է օգտագործել, որպես երկրորդական
ենթավերնագրեր</h3>

<h4>Չորրորդ մակարդակի վերնագրերը արտապատկերվում են
համարյա ինչպես սովորական տեքստ, բայց որոշ չափով
առանձնացվում են</h4>

<h5>Հինգերորդ մակարդակի վերնագիրը սովորական տեքստից էլ
փոքր է, բայց նույնպես առանձնացվում է </h5>

<h6>Սա վեցերորդ մակարդակի վերնագիր է</h6>

մեր շաբլոնի <body> և </body> տեղերի միջև և հիշենք ֆայլը
“.html" ընդլայնումով: Բացելով ստացված ֆայլը բրաուզերում ար-
դյունքում կտեսնեն այն, ինչ ցուցադրված է պատկեր 2.2.4-ում:

Առաջին մակարդակի վերնագրերը ամենագլխավոր վերնագրերն են

**Երկրորդ մակարդակի վերնագրերն էլ
ոչինչ, կարելի է օգտագործել որպես
ենթավերնագրեր**

**Երրորդ մակարդակի վերնագրերը երկրորդից
փոքր են, կարելի է օգտագործել որպես
երկրորդական ենթավերնագրեր**

**Չորրորդ մակարդակի վերնագրերը արտապատկերվում
են հաճախ ինչպես սովորական տեքստ, բայց որոշ
չափով առանձնացվում են**

**Հինգերորդ մակարդակի վերնագիրը սովորական
տեքստից էլ փոքր է, բայց նույնպես առանձնացվում է**

Սա վեցերորդ մակարդակի վերնագիր է

Պատկեր 2.2.4. Վերնագրերի կիրառության օրինակ

Ընդհանուր դեպքում խորհուրդ չի տրվում բաց տողներ վերնագրերի մակարդակները, օրինակ՝ <h3> մակարդակի վերնագիրը չի կարող հաջորդել <h1>-ին, եթե նրանց միջև չկա <h2> մակարդակի վերնագիր: Սակայն հաճախ վերնագրերը օգտագործվում են պարզապես տառաշարի չափերը ինտերպրետատորին թելադրելու նպատակով, ինչը սխալ է, քանի որ տարբեր բրաուզերները արտապատկերում են դրանք տարբեր ձևով: Տառաշարի չափերը փոխելու համար ավելի հարմար է ոճերի աղյուսակների օգտագործումը (ոճերի աղյուսակները մենք կուսումնասիրենք ավելի ուշ):

Տեքստի ընդհանուր տեսքը բարելավելու համար հարմար է նաև <hr /> (hr - horizontal rule) տեղի օգտագործումը: Դրա միջոցով պարբերությունների (կամ ցանկացած այլ էլեմենտների) միջև տե-

ղաղրվում է հորիզոնական գիծ, որը կարելի է տարածել էկրանի ամբողջ լայնությամբ: Տեգը եզակի է՝ այսինքն չի պահանջում փակող էլեմենտ: <hr /> տեգը կարող է ունենալ որոշակի ատրիբուտներ, որոնց միջոցով փոփոխվում են հորիզոնական գծի տեսքը և չափերը, սակայն, եթե մենք ցանկանում ենք ղեկավարվել խիստ ստանդարտներով, այդ նպատակները իրականացնելու համար ատրիբուտների փոխարեն անհրաժեշտ է օգտագործել ոճի աղյուսակներ:

Կարող է թվալ, որ տրամաբանական ոճավորման էլեմենտները ավելցուկային են, սակայն դա այդպես չէ, քանի որ ոչ բոլոր բրաուզերներն են կարողանում արտապատկերել ֆիզիկական ոճավորման տեգերը: Եթե, օրինակ, բջջային հեռախոսի ներկառուցված բրաուզերը չի կարող արտապատկերել թավ տառաշարը, ապա այն պարզապես կանտեսի տեգը: Իսկ եթե օգտագործված է տրամաբանական ոճավորման էլեմենտ, օրինակ՝ , ապա այն կաշխատի որևէ կերպ առանձնացնել այդ տեգում եզրափակված տեքստը՝ ընդգծի կամ փոփոխի պայծառությունը: Իհարկե, եթե հեղինակը համոզված է, որ էջի բոլոր այցելուներն ունեն գրաֆիկական բրաուզերներ, ապա այդ դեպքում տրամաբանական ոճավորումը կարող է չպահանջվել:

2.2.4. Ցուցակներ

Ցուցակների էլեմենտները հանդիսանում են կոնտեյներներ, որոնք կարող են ընդգրկել նաև տեքստի գծանշման այլ էլեմենտներ: Ցուցակում միշտ պետք է պարունակվեն երկու տեսակի էլեմենտներ, որոնցից առաջինը սահմանում է ցուցակի տեսակը, իսկ երկրորդը՝ ցուցակի կոնկրետ կետը, որն իրենից կարող է ներկայացնել բառ, նախադասություն, պարբերություն, պատկեր կամ մի որևէ այլ HTML էլեմենտ: Հիմնականում ցուցակների գրանցման համար օգտագործվում է հետևյալ ֆորմատը՝

<ցուցակի տեսակը>

առաջին կետ

երկրորդ կետ

...

վերջին կետ

</ցուցակի տեսակը>

Յուրաքանչյուր (list item) էլեմենտի պարունակությունը դա ցուցակի կետ է, որը միշտ գրանցվում է նոր տողում: Իսկ թե ինչից է սկսվում այդ տողը, կախված է ցուցակի տեսակից: HTML լեզվում օգտագործվում են երեք տեսակների ցուցակներ՝

- Կարգավորված կամ համարակալված ցուցակներ:
- Չկարգավորված կամ պիտակավորված ցուցակներ:

– Սահմանումների ցուցակներ:

Համարակալված (կարգավորված) ցուցակը կազմակերպվում է **** (ordered list) տեգի միջոցով, իսկ պիտակավորվածը՝ **** (unordered list): Եվ համարակալման, և պիտակավորման եղանակները կարելի է սահմանել այդ տեգերի **“type=”տեսակ**”

Մեկ անգամ էլ նշենք, որ համաձայն նոր ստանդարտի հետագայում կպահպանվեն միայն հղումային ատրիբուտները (հասցեները): Համապատասխան հատկությունները յուրաքանչյուր տեգին պետք է հնարավորին չափ շնորհվեն ռժերի աղյուսակների միջոցով:

Բերենք համարակալված և պիտակավորված ցուցակների օրինակներ՝

```
<ol>
<li>առաջին կետ</li>
<li>երկրորդ կետ</li>
<li>երրորդ կետ</li>
</ol>
<ul>
<li>առաջին կետ</li>
<li>երկրորդ կետ</li>
<li>երրորդ կետ</li>
</ul>
```

Համարակալումը կարելի է կատարել՝

- արաբական թվերով՝ type=“1”,
- լատինական այբուբենի մեծատառերով՝ type=“A”,
- լատինական այբուբենի փոքրատառերով՝ type=“a”,
- հռոմեական փոքրատառերով՝ type=“i”,
- հռոմեական մեծատառերով՝ type=“I”:

Եթե համարակալումը անհրաժեշտ է սկսել որևէ կոնկրետ թվից, ապա type ատրիբուտի փոխարեն գրանցվում է start ատրիբուտը համապատասխան արժեքով (դա վերաբերվում է միայն արաբական թվերին): Օրինակ, եթե պետք է համարակալումը սկսել 15-ից, ապա ցուցակը կընդունի հետևյալ տեսքը՝

```
<ol start=“15”>
<li>առաջին կետ</li>
<li>երկրորդ կետ</li>
<li>երրորդ կետ</li>
</ol>
```

Պիտակավորումը, ըստ լրելայն կատարվում է ներկված (լիքը) շրջանակներով (**type=“disc”**): Բացի այդ կարելի է կազմել ցուցակ-

ներ քառակուսիներով (**type="square"**) և դատարկ շրջանակներով (**type="circle"**):

Սահմանումների ցուցակը ընդգրկում է երկու մակարդակներ՝
թերմիններ և սահմանումներ: Այն բաղկացած է **<dl>** (definition list)
զլխավոր կոնտեյներից, **<dt>**(definition term) թերմիններից և սահ-
մանումներից **<dd>** (data definition): Սահմանումների ցուցակի կա-
ռուցվածքը կարելի է ներկայացնել հետևյալ տեսքով՝

```
<dl>
<dt>առաջին թերմին</dt>
<dd>առաջին թերմինի սահմանումը</dd>
<dt>երկրորդ թերմին</dt>
<dd>երկրորդ թերմինի սահմանումը</dd>
...
<dt>ցուցակի վերջին թերմինը</dt>
<dd>վերջին թերմինի սահմանումը </dd>
</dl>
```

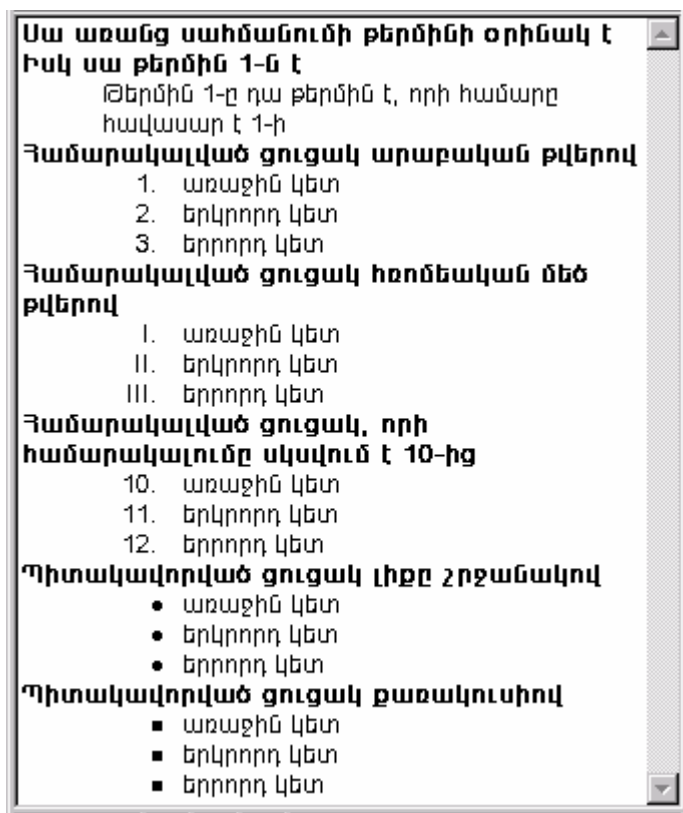
Գրաֆիկական բրաուզերների մեծամասնությունը տեղաբաշ-
խում է թերմինի սահմանումը քիչ խորքից: Պարտադիր չէ օգտա-
գործել սահմանումների ցուցակի բոլոր էլեմենտները: Օրինակ
կարելի սահմանափակվել միայն **<dl>** և **<dt>** էլեմենտներով: Այդ
դեպքում ցուցակը նման է առանց պիտակների պիտակավորված
ցուցակին: Ստորև բերված է տարբեր ցուցակների կառուցման
օրինակները պարունակող HTML կոդը: Տեղադրենք այն շաբլոնի
<body> տեգում և հիշեցնենք **“.html"** ընդլայնումով: Ծրագրի
աշխատանքի արդյունքը բերված է պատկեր 2.2.5-ում:

```
<dl>
<dt><b>Սա առանց սահմանումի թերմինի օրինակ է</b></dt>
<dt><b>Իսկ սա թերմին 1-ն է</b></dt>
<dd>Թերմին 1-ը դա թերմին է, որի համարը հավասար է 1-ի</dd>
<dt><b>Համարակալված ցուցակ արաբական թվերով</b></dt>
<dd><ol><li>առաջին կետ</li>
<li>երկրորդ կետ</li>
<li>երրորդ կետ</li></ol></dd>
<dt><b>Համարակալված ցուցակ հռոմեական մեծ թվերով</b>
</dt>
<dd><ol type="I"><li>առաջին կետ</li>
<li>երկրորդ կետ</li>
<li>երրորդ կետ</li></ol></dd>
<dt><b>Համարակալված ցուցակ, որի համարակալումը սկսվում է
10-ից</b></dt>
<dd><ol start="10"><li>առաջին կետ</li>
```

```

<li>երկրորդ կետ</li>
<li>երրորդ կետ</li></ol></dd>
<dt><b>Պիտակավորված ցուցակ լիքը շրջանակով</b></dt>
<dd><ul><li>առաջին կետ</li>
<li>երկրորդ կետ</li>
<li>երրորդ կետ</li></ul></dd>
<dt><b>Պիտակավորված ցուցակ քառակուսիով</b></dt>
<dd><ul type="square"><li>առաջին կետ</li>
<li>երկրորդ կետ</li>
<li>երրորդ կետ</li></ul></dd>
</dl>

```



Պատկեր 2.2.5. Ցուցակների կիրառության օրինակներ

Ինչպես տեսնում ենք բերված կոդից կոնտեյներային տեգերի ներսում կարելի է կիրառել HTML ռճային գծանշումը, այսինքն ներդնել գծանշման այլ տեգեր:

§2.3. Հիպերհղումների կազմակերպումը HTML-ում

2.3.1. Հիպերհղումների աշխատանքի սկզբնունքը: Բազարձակ և հարաբերական հասցեավորում

Հիպերհղումները հնարավորություն են տալիս այցելուներին երթևեկել web-կայքերով: Օգտվելով հիպերհղումից օգտվողը սովորաբար կանչում է որևէ URL հասցեով գտնվող ֆայլ, որը գործարկվում է բրաուզերի կամ օժանդակ հավելվածի միջոցով: Դա կարող է լինել նոր web-էջ, կամ Ինտերնետի որևէ այլ ռեսուրս:

Հիպերհղումներ ստեղծելու համար օգտագործվում է հատուկ տեգ, որը կոչվում է «**խարիսխ**» (անգլերեն՝ anchor)՝ **<a>...**: Տեգն ունի պարտադիր ատրիբուտ՝ **href**: Ատրիբուտի արժեքը՝ URL հասցե է, որով հարկավոր է անցնել ըստ տվյալ հղումի:

Ինչպես նշվել էր առաջին գլխում URL հասցեն բաղկացած է երկու մասերից՝ առաջինում գրանցվում է ինֆորմացիայի փոխանակման արձանագրությունը, իսկ երկրորդում՝ նպատակային ռեսուրսին հասնելու ուղին: Ընդ որում դա կարող է լինել ֆայլ, կատալոգ կամ համակարգիչ: Օրինակ՝

<http://www.fakecorp.com/products/index.html>

հասցեն ցուցանշում է index.html փաստաթղթին, իսկ՝

<ftp://ftp.netscape.com>

հասցեն թելադրում է բրաուզերին, որ անհրաժեշտ է օգտագործել FTP արձանագրությունը [ftp.netscape.com](ftp://ftp.netscape.com) դոմեյնային անունով համակարգչի հետ կապ հաստատելու համար:

Եթե հղումային հասցեն ընդգրկում է վերը նշված երկու մասերը լիովին, ապա այն կոչվում է **բացարձակ հասցե**:

Նշենք մի կարևոր հանգամանք: Այն դեպքերում երբ հիպերհղումում գրանցված հասցեն տարբերվում է ընթացիկ հասցեից (այսինքն այն փաստաթղթի հասցեից, որը տվյալ պահին արտապատկերվում է բրաուզերում) ընդամենը ֆայլի անունով, ապա պարտադիր չէ գրանցել հղումային հասցեն լիովին՝ ավելի հարմար է օգտագործել, այսպես կոչված, «**հարաբերական հասցեն**»: Ասվածը պարզաբանելու համար դիտարկենք երկու հասցեներ՝

<http://www.mysite.am/index.html>,

<http://www.mysite.am/resume.html>

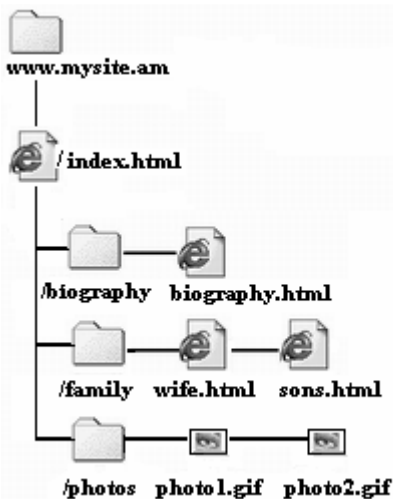
Երկուսն էլ բացարձակ հասցեներ են: Դրանց օգնությամբ կարելի է այցելել նշված էջերը Ինտերնետի ցանկացած տեղից: Սա-

կայն պատկերացնենք, որ index.html էջում ստեղծված է հղում resume.html էջին: Արդյո՞ք այդ դեպքում պարտադիր է նորից գրանցել այդ երկար հասցեն: Պարզվում է, որ ոչ՝ կարելի է օգտագործել հարաբերական հասցեավորումը, քանի որ բերված երկու հասցեների հիմնական մասը միևնույնն է:

Երբ արտապատկերվում է որևէ ֆայլ, ապա դրան պարունակող կատալոգը գրանցվում է բրաուզերի հիշողության մեջ: Եվ հայտնաբերելով որևէ հղումային հարաբերական հասցե, բրաուզերը պարզապես ավելացնում է այն ընթացիկ կատալոգին և ստանում բացարձակ հղումային հասցեն:

Մեր օրինակում երկու ֆայլերն էլ գտնվում են միևնույն՝ “http://www.mysite.am/” կատալոգում: Եթե տվյալ պահին արտապատկերվում է index.html ֆայլը, բրաուզերի հիշողությունում, որպես ընթացիկ, գրանցվել է “http://www.mysite.am/” կատալոգը և, եթե հղումային հասցեն հետևյալն է՝ “resume.html”, ապա բրաուզերը կգումարի `http://www.mysite.am/` և `resume.html` արժեքները, ստանալով բացարձակ՝ “http://www.mysite.am/resume.html” հասցեն, որով և կանցնի resume.html ֆայլին:

Հարաբերական հասցեավորումը կարելի է օգտագործել նաև կայքի այլ կատալոգներում և ենթակատալոգներում գտնվող փաստաթղթերին և ֆայլերին դիմելու համար: Պարզաբանելու համար օգտվենք պայմանական կայքից (տես՝ պատկեր 2.3.1):



Պատկեր 2.3.1. Պայմանական կայքի կառուցվածքը

Ենթադրենք մեզ անհրաժեշտ է `index.html` էջից անցնել `sons.html` էջը, որը գտնվում է `"http://www.mysite.am/"` (այսինքն ընթացիկ) կատալոգի `family` ենթակատալոգում: Այդ դեպքում հղումային հարաբերական հասցեն կլինի հետևյալը՝ `"family/sons.html"`: Բրաուզերը կգումարի ընթացիկ `"http://www.mysite.am/"` կատալոգին հարաբերական հասցեն և արդյունքում կանցնի բացարձակ՝ `http://www.mysite.am/family/sons.html` հասցեով էջին:

Այժմ պատկերացնենք, որ արտապատկերված է `sons.html` ֆայլը: Այդ դեպքում բրաուզերի հիշողության մեջ որպես ընթացիկ գրանցված է `"http://www.mysite.am/family/"` կատալոգը:

Եթե անհրաժեշտ է այդ էջից հղում կատարել գլխավոր՝ `index.html` էջին, որը գտնվում է կայքի, այսպես կոչված, արմատային (`root`)՝ `"http://www.mysite.am/"` կատալոգում, ապա հղումային հարաբերական հասցեն գրանցվում է հետևյալ տեսքով՝ `"../index.html"`: Իսկ եթե `sons.html` էջից պետք է անցնել արմատային կատալոգի մի այլ ենթակատալոգում գտնվող էջի, օրինակ՝ `biography.html`, ապա հարաբերական հասցեն գրանցվում է հետևյալ կերպով՝ `"../biography/biography.html"`:

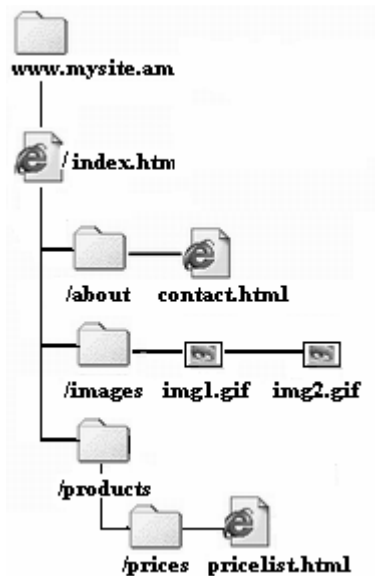
Երկու դեպքում էլ, հարաբերական հասցեի սկզբում դրված երկու կետերը թելադրում են բրաուզերին, որ պետք է մեկ մակարդակով վեր բարձրանալ կատալոգների հիերարխիայում: Իսկ ինչպե՞ս վարվել այն դեպքերում, երբ կայքն ունի բարդ կառուցվածք, այսինքն արմատային կատալոգի ենթակատալոգները իրենց հերթին ունեն ներդրված ենթակատալոգներ և այլն: Քննարկենք դա օրինակի վրա:

Դիցուկ մենք ստեղծել ենք `"http://www.mysite.am/"` արմատային կատալոգով կայք (տես՝ պատկեր 2.3.2.):

Արմատային կատալոգում ստեղծել ենք `images`, `about` և `products` ենթակատալոգները:

`products` կատալոգում ստեղծել ենք `prices` ենթակատալոգը և վերջինում՝ `pricelist.html` էջը: Այդ էջի բացարձակ հասցեն կլինի հետևյալը՝ `"http://www.mysite.am/products/prices/pricelist.html"`:

Այժմ պատկերացնենք, որ `pricelist.html` էջից մենք ցանկանում ենք տեղափոխվել `about` կատալոգի `contact.html` էջը: Պատկեր 2.2.7-ից երևում է, որ `about` կատալոգը ներդրված է անմիջապես արմատային կատալոգում (առաջին մակարդակի ներդրվածություն), իսկ `prices` կատալոգը գտնվում է կայքի հիերարխիայի երկրորդ մակարդակում: Այսինքն, որպեսզի իրականացվի անցումը պահանջվող էջին, բրաուզերը պետք է `pricelist.html` էջից բարձրանա երկու մակարդակով (մինչև արմատային կատալոգ) և այնտեղից նոր անցնի `about` կատալոգին:



Պատկեր 2.3.2. Բազմամակարդակ կառուցվածք ունեցող կայքի օրինակ

Կարելի է, իհարկե, տվյալ անցումը ապահովելու համար հարաբերական հասցեն գրանցել, օգտագործելով զույգ կետերի սկզբնունքը՝ “../..../about/contact.html” (ամեն մի “../” գրանցումը նշանակում է վերելք հիերարխիայի մեկ մակարդակով): Սակայն հասցեի գրանցման այդ եղանակի կիրառությունը այնքան էլ չի տարբերվում հասցեի բացարձակ արժեքի գրանցման եղանակից և, բացի այդ մեծանում է վրիպակ թույլ տալու հավանականությունը (օրինակ՝ եթե հանկարծ շփոթենք կետերի քանակը):

Այդպիսի իրավիճակներից խուսափելու և հասցեավորումը հստակեցնելու համար HTML-ում գոյություն ունի հատուկ՝ **<base />** եզակի տեգը “href” ատրիբուտով, որի միջոցով յուրաքանչյուր էջի համար կարելի է սահմանել կայքի արմատային կատալոգի բազային՝ URL հասցեն: Եթե այժմ էջի վերնագրային մասում գրանցենք հետևյալ՝ **<head><base href="http://www.mysite.am/" /></head>** կոդը, ապա բոլոր հասցեները էջում կարելի է ստեղծել հարաբերելով տեգի href ատրիբուտում նշված հիմքին, այլ ոչ թե էջի ընթացիկ տեղաբաշխմանը: Էջերի հղումային բացարձակ հասցեները ստեղծելիս բրաուզերը պարզապես կգումարվի հարաբերական հասցեն

“բազային” հասցեին և կստանա բացարձակ հասցեն: Օրինակ, եթե `pricelist.html` էջից պետք է անցնել `contact.html` էջին, ապա բավական է գրել հարաբերական հասցեն հետևյալ՝ `about/contact.html` տեսքով: Գումարելով այն բազայինին բրաուզերը կստանա՝ `http://www.mysite.am/about/contact.html`:

Նշենք, որ `<base />` էլեմենտը վերաբերվում է էջի բոլոր հարաբերական հասցեներին, սակայն ոչ մի կերպ չի ազդում բացարձակ հասցեների վրա:

2.3.2. Հիպերդոմենների և ներքին հղումների ստեղծումը web-էջում:

Հիպերհղումների մեծ մասը ստեղծվում է, ինչպես նշվեց 2.3.1. կետում, `<a>`, `` տեգի միջոցով, որի “`href`” ատրիբուտում գրանցվում է հղումի բացարձակ կամ հարաբերական հասցեն: Տեգի ներսում գրանցվում է տեքստ, որը նկարագրում է տվյալ հղումը: Օրինակ՝ `` Իմ կենսագրությունը``, կամ, եթե մենք օգտագործում ենք հարաբերական հասցե՝ ``Իմ կենսագրությունը``:

Հարաբերական հասցեների օգտագործումն ունի շատ կարևոր առանձնահատկություն՝ դրանք չեն փոփոխվում նույնիսկ այն դեպքում, երբ կայքը վերատեղաբաշխվում է: Օրինակ, եթե կայքը տեղադրվի այլ սերվերի վրա, ապա հարաբերական հասցեները կշարունակեն “աշխատել”: Անհրաժեշտ կլինի փոփոխել միայն բացարձակ հասցեները և (եթե այն առկա է) `<base />` էլեմենտի `href` ատրիբուտի արժեքը:

`<a>` տեգի մեջ չի կարելի ներդնել մի այլ `<a>` տեգ, այսինքն չի կարելի գրանցել հղումը, օրինակ, հետևյալ տեսքով՝
``Պատվիրեք առաջին
``կամ երկրորդ`` տեսակի ապրանքները մեր խանութում``

Դրա փոխարեն պետք է գրանցել հետևյալը՝
Պատվիրեք ``առաջին`` կամ
``երկրորդ`` տեսակի ապրանքները մեր խանութում

`<a>...` տեգը ունի ևս մեկ՝ **target** (հայերեն՝ նշանակետ) ատրիբուտը: Մանրամասն այդ ատրիբուտը կքննարկվի ավելի ուշ, շրջանակների միջոցով web-էջերի կազմակերպմանը նվիրված թեմայում: Սակայն այն ունի շատ հետաքրքիր կիրառություն: Այն դեպքերում, երբ ցանկալի է կամ անհրաժեշտ կանչվող փաստաթուղթը բացել նոր պատուհանում, կարելի ատրիբուտին շնորհել “**_blank**” արժեքը: Օրինակ եթե հղումը գրանցվի հետևյալ տեսքով՝

``Ապրանքների գնացուցակը``, ապա `pricelist.html` փաստաթուղթը կարտապատկերվի բրաուզերի նոր պատուհանում (ընդ որում ընթացիկ պատուհանը չի փակվի):

Եթե HTML փաստաթուղթը ծավալուն է, հատուկ միջոցներ չձեռնարկելու դեպքում մեզ հետաքրքրող մասը գտնելու համար անհրաժեշտ է լինում պտտաստեղնի (`scrollbar`-ի) միջոցով պտտել էջը վեր ու վար: `<a>` տեգը թույլ է տալիս ստեղծել հղումներ, որոնք հնարավորություն են ընձեռում անցնել ընթացիկ էջի տարբեր մասերին այսպես կոչված “ներքին հղումների” միջոցով: Պատկերացնենք, որ մենք կազմել ենք ծավալուն HTML փաստաթուղթ, որը ընդգրկում է մեծ թվով բաժիններ (օրինակ՝ էլեկտրոնային ձեռնարկ) և ցանկանում ենք առանձին բաժիններին անմիջականորեն անցնելու նպատակով ստեղծել վերը նշված տեսակի հղումներ:

Այդ խնդիրը լուծելու համար, առաջին հերթին, յուրաքանչյուր բաժնի սկզբում գրանցվում է հատուկ՝ **name** ատրիբուտ ունեցող խարիսխ (`<a>` տեգ): Օրինակ, առաջին բաժնի սկզբում կգրենք՝

`Բաժին 1`:

Նշենք, որ համաձայն XML ստանդարտի `name` ատրիբուտը փոխարինվելու է `id` (իդենտիֆիկատոր) ատրիբուտով: Իսկ `id` ատրիբուտը, իր հերթին, չի սատարվում որոշ հին բրաուզերների կողմից: Այդ պատճառով մոտակա ապագայում համատեղելիությունը ապահովելու նպատակով առավել հուսալի կլինի օգտագործել երկու ատրիբուտներն էլ միաժամանակ: Հաշվի առնելով այդ փաստը գրանցենք խարիսխը հետևյալ տեսքով՝

`Բաժին 1`:

- Ինչպես `name`, այնպես էլ `id` ատրիբուտների արժեքները կարող են պարունակել տառեր և թվանշաններ, սակայն համաձայն նոր ստանդարտի ցանկացած դեպքում դրանց առաջին սիմվոլը պետք է լինի տառ:

Բոլոր բաժինները անվանակոչելուց հետո ստեղծվում են համապատասխան հղումները, որոնք սովորաբար տեղադրվում են էջի սկզբնամասում՝ ցանկի տեսքով: Այդ նպատակով URL հասցեի փոխարեն `href` ատրիբուտին որպես արժեք շնորհվում է համապատասխան բաժնի իդենտիֆիկատորի (կամ անունի) արժեքը, որի արջև գրանցվում է “#” սիմվոլը՝

`Բաժին 1`:

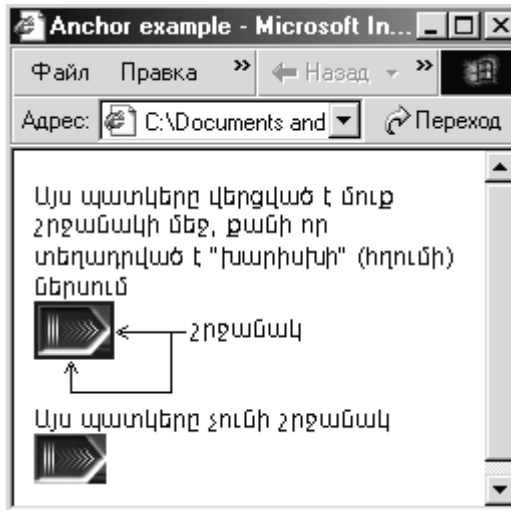
Անվանակոչված բաժիններին կարելի է դիմել նաև այլ էջերից՝ պարզապես բաժնի անունը գրանվում է որպես էջը որոշող URL հասցեի մի մաս: Օրինակ՝

`Բաժին 1`:

Եթե ցանկանում եմք հղման մեջ նկարագրող տեքստի փոխարեն (կամ մեկտեղ) տեղադրել պատկեր, ապա `<a>` տեգում պարզապես ներդրվում է `` տեգը, որի `src` ատրիբուտի արժեքը՝ պատկերը պարունակող ֆայլի URL հասցեն է: Օրինակ՝

``

Պատկեր-հղումը տարբերվում է սովորական պատկերից նրանով, որ վերցվում է շրջանակների մեջ (տես՝ պատկեր 2.3.3.):



Պատկեր 2.3.3. Հղումային և սովորական պատկերները

Պատկեր-հղումի շրջանակը կարելի է կարգավորել `` տեգի `border="արժեք"` ատրիբուտի միջոցով: Օրինակ, եթե ցանկալի է, որ պատկերն ունենա շրջանակներ, ապա պարզապես կարելի է գրել՝ ``:

2.3.2. Հատուկ հիպերհղումներ:

URL-ը ճկուն և ունիվերսալ միջոց է՝ գործնականում այն հնարավոր է օգտագործել ցանցում առկա բոլոր ռեսուրսներին հղումներ ստեղծելու համար: Դա հնարավորություն է ընձեռնում տարատեսակ ինֆորմացիան կապակցված տեսքով համատեղել մեկ էջում:

ճկունությունը բրաուզերների գլխավոր առավելություններից մեկն է: Բրաուզերների միջոցով կարելի անել այն ամենը ինչ հնարավոր է ցանցում և, նույնիսկ ավելին՝ հնարավոր է ստանալ հասա-

նելիությունն այն ինֆորմացիային կամ ծրագրերին, որոնք տեխնոլոգիապես չեն հանդիսանում Ինտերնետի ռեսուրսներ: Եթե բրաուզերը ունակ չէ ինքնուրույն մշակել Ինտերնետի այս կամ այն ծառայությանը վերաբերվող ֆայլը, ապա ավտոմատ կերպով գործարկում է համապատասխան օժանդակ հավելվածը, օրինակ՝ էլեկտրոնային փոստի խմբագրիչը կամ մուլտիմեդիայի ծայնարկիչը: Որպես արդյունք հնարավոր է դառնում հղումներ ստեղծել Ինտերնետի բոլոր ծառայություններին: Ընդ որում հղումների քերականությունը մնում է նույնը:

Հղումը էլեկտրոնային փոստին կատարվում է դյուրին եղանակով, պարզապես պետք է իմանալ ստույգ հասցեն: Էլեկտրոնային փոստի հասցեն բաղկացած է չորս մասերից՝ օգտվողի անունից, շնիկ (@) մշանից, համակարգչի անունից (այն կարող է նաև բացակայել) և դոմեյնից: Բերենք հասցեների օրինակներ:

1. shmavon@mac-upgrade.com: Այստեղ՝ shmavon-ը օգտվողի անունն է, իսկ mac-upgrade.com-ը դոմեյն է: Համակարգչի անունը բացակայում է:

2. vazgen@mail.fakecorp.com: Այս հասցեում արդեն համակարգչի անունը առկա է՝ mail:

Հասցեներին ավելացնելով արձանագրության անունը՝ “mailto:” և տեղադրելով որպես <a> տեգի href ատրիբուտի արժեք՝ կստանանք հղումը, օրինակ՝

Ուղարկեք ձեր մանակները այս հասցեով:

Հղումները FTP կայքերին օգտագործվում են համակարգիչների միջև ֆայլերի փոխանակություն կազմակերպելու նպատակով: Հեռացված համակարգչի հետ սեսիա կազմակերպելուց հետո FTP-ի օգտվողները կարող են ստանալ անհրաժեշտ ֆայլերը: Օրինակ՝

Այցելեք Microsoft ֆիրմայի FTP կայքը:

Եթե օգտվողին արաջարկվում է “քաշել” (այսինքն ներմուծել և գրանցել իրեն պատկանող համակարգչի կոշտ սկավառակի վրա) որոշակի ֆայլ, ապա ցանկալի է մշել այդ ֆայլի լրիվ հասցեն: Օրինակ եթե օգտվողին արաջարկվում է ստանալ program.zip ֆայլը, որը գտնվում է downloads ենթակատալոգում, ապա հասցեն պետք է գրանցվի հետևյալ կերպով՝

Այս ֆայլը՝ zip ֆորմատով ծրագիր է :

Այսպիսի հղումը հրամայում է բրաուզերին միանալ FTP սերվերին և անմիջապես սկսել ներմուծել անհրաժեշտ ֆայլը:

Երբ անհրաժեշտ է ստեղծել կայք օգտվողներին բազմազան ֆայլեր հատկացնող կոմպանիայի համար, ապա բոլոր այդ ֆայլերը կարելի է զետեղել FTP սերվերի որևէ կատալոգում, իսկ կայքում ստեղծել միայն հղում այդ սերվերին: Դա թույլ կտա խուսափել տարբեր ֆայլերին բազմաթիվ հղումներ կազմակերպելուց: Բացի այդ FTP սերվերը օգտագործում է ֆայլերի փոխանցման առավել կատարյալ մեթոդներ: Իհարկե դա չի նշանակում, որ յուրաքանչյուր փոկրիկ կայք պետք է ունենա FTP սերվերում զետեղված ֆայլեր՝ HTTP արձանագրությունը նույնպես ունակ է հաղորդել երկուսական տվյալներ, ընդ որում, առավել հաջող է այն “հաղթահարում” “.exe”, “.zip” և “.sit” (“Stuffer” արքիվատորի միջոցով ստեղծված) ընդլայնումներով ֆայլերի հաղորդումը: Եթե ֆայլին կատարվում է սովորական HTTP հղում, օրինակ՝ <http://www.microsoft.com/downloads/program.zip>”>Ստանանք ֆայլը zip ֆորմատով, ապա բրաուզերը սկսում է ֆայլի ընդունան գործընթացը և արաջարկում պահպանել կոշտ սկավառակի վրա, հարցնելով թե կոնկրետ որտե՞ղ է ցանկանում օգտվողը այն պահպանել:

Gopher սերվերների կայքերին հղումը կառուցվում է նույն սկզբնունքով, պարզապես հեռացված համակարգչի անունից առաջ գրանցվում է համապատասխան արձանագրության անունը, օրինակ՝ <gopher://marvel.loc.gov/>”>Ամերիկյան կոնգրեսի գրադարանը:

Usenet-ին կատարվող հղումների ֆորմատը որոշ չափով տարբերվում է սովորական հիպերհղումներից: Այս դեպքում կոնֆերանսի անվան դիմաց պետք է գրանցել “news:” արտահայտությունը: Հղումը Usenet-ին կատարվում է հետևյալ եղանակով՝ <news:sci>”>Այսինչ պորբլեմին նվիրված Usenet-ի կոնֆերանս: Բերված հղումը ուղղված է Usenet-ի Sci (գիտական կոնֆերանսներ) թեմատիկ խմբին: Այդպիսի հղում հանդիպելիս բրաուզերը կատարում է երկու գործողություններից մեկը՝ կամ փորձում է ինքնուրույն միանալ նորությունների սերվերին և գտնել հաղորդակցությունների համապատասխան խումբը, կամ հրամանը փոխանցում է Usenet կոնֆերանսների հետ աշխատող ծրագրին:

Telnet սերվերին կատարած հղումը թույլ է տալիս օգտվողին անմիջական կապ հաստատել այն համակարգչի հետ, որում տեղադրված է հեռացված հասանելիության սպասարկման ծրագրային ապահովումը: Իրականում ոչ մի բրաուզեր չի սատարում Telnet տեխնոլոգիան և առավել հավանական է, որ օգտվողը պետք է աշխատի որևէ օժանդակ հավելվածի միջոցով, որը կբեռնվի

հղումային հասցեով անցնելու դեպքում: Հղումը Telnet սերվերին ունի հետևյալ քերականությունը՝
Կապ հաստատել Telnet-սերվերի հետ:

§2.4. Գրաֆիկայի և մուլտիմեդիայի օգտագործումը HTML փաստաթղթերում

2.4.1. Գրաֆիկայի օգտագործումը web-էջերում:

Դժվար է այժմ հանդիպել որևէ web-մախազիժ, որը չպարունակի պատկերազարդ էջեր (եթե կան, ապա հիմնականում դրանք կան գրական տեքստեր են կամ մասնագիտական տեղեկատուներ): Web-էջերի ճնշող մեծամասնությանը հատուկ է գրաֆիկական պատկերների առկայությունը, որը կամ բարելավում է արտաքին տեսքը, դյուրին է դարձնում կայքի օգտագործումը կամ այցելումներին մատուցում է որոշակի ինֆորմացիա: Ըստ ֆունկցիոնալ նշանակման գրաֆիկական բաղադրիչները կարելի է պայմանականորեն բաժանել երեք ծավալում խմբերի.

պատկերազարդ (իլյուստրատիվ) գրաֆիկա, օրինակ՝ տեքստը լրացնող ֆոտոնկարներ, բացատրական նկարներ, գծագրեր և սխեմաներ,

ֆունկցիոնալ գրաֆիկա՝ կայքի ղեկավարման էլեմենտները ձևավորելու համար: Օրինակ՝ նավիգացիայի կոճակներ, հաշվիչներ և ինտերակտիվ ձևերի (ֆորմաների) էլեմենտներ, դեկորատիվ գրաֆիկա, օրինակ՝ գրաֆիկական ֆայլերի տեսքով մշակված վերնագրեր և ֆոնային պատկերներ:

Համաձայն վիճակագրական տվյալների ինտերնետի օգտվողների մեծամասնությունը կապը ցանցի հետ իրագործում է մոդեմների միջոցով՝ կոմուտացվող հեռախոսային կաճալներով: Քանի որ դա կապի բավականին դանդաղագործ եղանակ է, ապա ցանկալի է հնարավորին չափ կրճատել գրաֆիկական պատկերների բեռնման տևողությունը կլիենտական բրաուզերներում: Հենց այդ փաստը պատճառ հանդիսացավ gif և jpeg տեսակների (ավելի շուտ՝ ընդլայնումների) գրաֆիկական ֆայլերի առավել լայն կիրառության համար: Այդպիսի ֆայլերին բնորոշ են պատկերի սեղմման ալգորիթմներ, որոնք թույլ են տալիս զգալիորեն կրճատել նպատակային ֆայլի ծավալը (հիարկե, որակի որոշակի կորուստներով):

jpeg (Joint Photographic Experts Group) ֆորմատը սովորաբար օգտագործվում է ֆոտոնկարներ կամ այլ մեծ թվով գույներ

պարունակող բազմերանգ պատկերներ հաղորդելու համար (թվային ֆոտոխցիկով կամ սկաներով ստացած):

gif (Graphics Interchange Format) ֆորմատը հիմնականում օգտագործվում է այսպես կոչված բիզնես-գրաֆիկայի արտապատկերման համար, օրինակ դիագրամների, լոգոտիպների, ղեկավարման կոճակների և ձևավորման այլ էլեմենտների: Այդպիսի ֆայլերը առավել հաճախ ստեղծվում են հենց համակարգչի միջոցով, լավ են սեղմվում և, հետևաբար ավելի արագ են հաղորդվում ցանցով: Բացի այդ gif (մասնավորապես GIF89a) ֆորմատն ունի նաև երկու ունիկալ հնարավորություններ, որոնք լայնորեն կիրառվում են Ինտերնետում:

Առաջինը կոչվում է **“transparency”** (թափանցիկություն): Այն թույլ է տալիս թափանցիկ ֆոն ստեղծել պատկերի համար: Կիրառվում է, օրինակ, այն դեպքերում, երբ բարդ ֆոնային պատկեր ունեցող էջում անհրաժեշտ է տեղադրել ոչ ճիշտ երկրաչափական ձև ունեցող պատկերներ և հնարավոր չէ կորեկտ համատեղել այդ պատկերների առանձին մասերը:

Ֆորմատի երկրորդ օգտակար առանձնահատկությունն այն է, որ հնարավորություն է ընձեռնվում մեկ ֆիզիկական անուն ունեցող ֆայլում պահպանել մի քանի տարբեր պատկերներ՝ հաջորդաբար ցուցադրելու հնարավորությամբ: Հենց այդ սկզբունքի վրա է հիմնված Ինտերնետում լայն տարածում գտած gif-անիմացիան: Առավել հաճախ անիմացված գրաֆիկան հանդիպում է web-կայքերում զետեղված գովազդում՝ **բաներների** տեսքով:

Բաները՝ մեկ ֆայլում պահպանված և որոշակի ժամանակահատվածում իրար հաջորդող առանձին կադրերի բազմություն է: Կադրերի ադպիսի հաջորդման հաշվին ստեղծվում է անիմացիայի, (ավելի շուտ մուլտիպլիկացիայի) էֆեկտ:

Web-էջում պատկեր տեղադրելը բարդ չէ, սակայն բարդ է այն տեղադրելիս բավարարել միաժամանակ մի քանի իրար հակասող պահանջները՝ պատկերը պետք է լինի հետաքրքրաշարժ, տվյալ ենթատեքստին օգտակար, գրավիչ և, միաժամանակ ոչ այնքան խոշոր (ֆայլի ծավալի տեսակետից): Գրաֆիկան տեղադրելիս անհրաժեշտ է ղեկավարվել մի քանի պարզ կանոններով՝

- Պատկերները և ֆոտոնկարները պետք է անմիջականորեն վերաբերվեն էջում զետեղված ինֆորմացիային: Հարկ չէ օգտագործել դրանք միայն էջի ծավալը մեծացնելու համար: Օգտվողների մեծամասնությունը Ինտերնետում փնտրում է ինֆորմացիա կամ ձգտում է օգտվել այս կամ այն ծառայությունից այլ ոչ թե ցանկանում է տեսնել հեղինակի սիրելի շնիկի նկարը:

- Գրաֆիկան պետք է բեռնվի հնարավորին չափ արագ, իսկ դա նշանակում է, որ ֆայլերը պետք է ունենան խելամիտ չափսեր: Ոչ բոլոր այցելուները կշարունակեն “մնալ” կայքում, եթե ամեն մի պատկերը բեռնվի մի քանի րոպեի ընթացքում:
- Հարկ է համոզվել, որ պատկերը ընտրված է ճիշտ ֆորմատում: Գոյություն ունեն պատկերների մշակման բազմաթիվ թվային միջոցներ, օգտվելով որոնցից կարելի է հասցնել պատկերը բավարար որոկի և զգալիորեն նվազեցնել ֆայլերի ծավալը:

Պատկերները էջում ներմուծվում են եզակի **** տեգի միջոցով: Տեգի ընդհանուր տեսքը հետևյալն է՝

****,

Ինտերնետի օգտվողների զգալի մասը չունի որևէ տեսակի գրաֆիկական բրաուզեր, կամ չի դիտարկում գրաֆիկական ֆայլերը դանդաղ միացման պատճառով: Պետք է հիշել նաև տարբեր տեսակի այլ սարքավորումներից (օրինակ՝ բջջային հեռախոսներ, զրպանի համակարգիչներ, ծայնային բրաուզերներ կույրերի համար) օգտվող այցելուների մասին: Այդպիսի դեպքերի համար HTML-ում գոյություն ունի հատուկ ատրիբուտ՝ **alt** (alternative text՝ այլընտրանքային տեքստ), որի արժեքը գրաֆիկական պատկերը փոխարինող որոշակի տեքստ է: Հաճախ այդ տեքստը արտապատկերվում է շրջանակներում, որոնք տեղադրվում են էկրանի այն մասում, որտեղ պետք է ցուցադրվեն պատկերը: Օրինակ՝

****:

Իհարկե տեքստը պետք է լինի հակիրճ, սակայն ինֆորմատիվ: Խորհուրդ չի տրվում գրանցել ոչ իմաստալից հաղորդակցություններ՝ օրինակ “Այստեղ պատկերված է իմ սիրելի զոքանչը”: Եթե անհրաժեշտ է ավելի մանրամասն նկարագրել բացակայող պատկերը, ապա կարելի է օգտվել մի այլ ատրիբուտից՝ **longdesc** (long description): Ատրիբուտի միջոցով հղում է կատարվում տվյալ պատկերի մանրամասն նկարագրությունը պարունակող ֆայլին՝

****:

Բրաուզերները վերծանում են պատկերները որպես սովորական տառեր և եթե չի կատարվել որոշակի լրացուցիչ կարգավորում, ապա էկրանին արտապատկերված տեքստի և պատկերների փոխտեղաբաշխումը ընդհանուր դեպքում կարող է չհամապատասխանել սպասվող արդյունքին: Այդ նպատակին է ծառայում հատուկ **align** ատրիբուտը, որն էլ հենց պատասխանատու է տեքս-

տի և պատկերների փոխադարձ դասավորության համար: Ատրիբուտն ունի կիրառության երկու եղանակներ:

Առաջին դեպքում ատրիբուտի միջոցով որոշվում է, թե ինչպես պետք է հավասարեցվի պատկերի հետ մեկ տողում գտնվող տեքստը փաստաթղթի կողմնային լուսանցքների հանդեպ: Աղյուսակ 2.4.1-ում բերված են align ատրիբուտի ստանդարտ արժեքները:

Աղյուսակ 2.4.1.

align ատրիբուտի ստանդարտ արժեքները

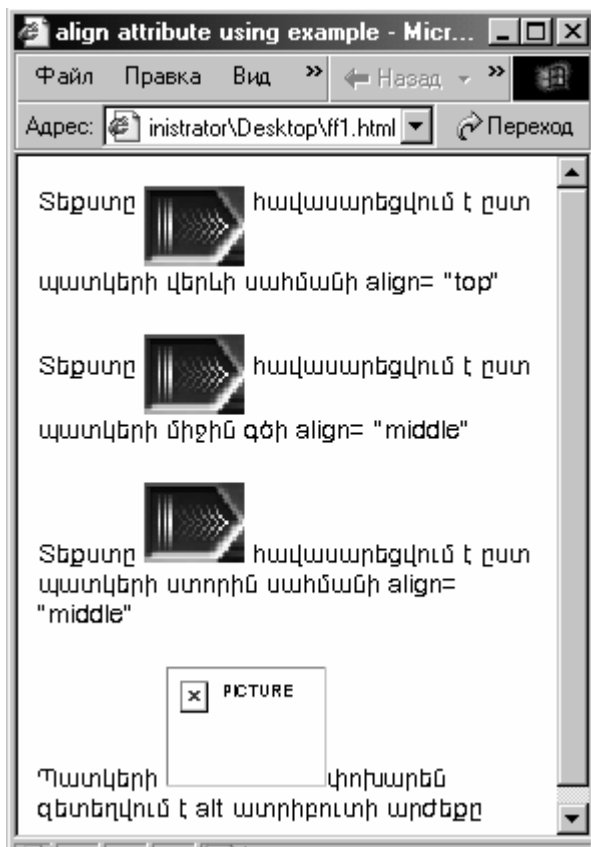
Արժեքը	Ազդեցությունը
top	Տեքստը հավասարեցվում է ըստ պատկերի վերին սահմանի
middle	Տեքստը հավասարեցվում է ըստ պատկերի միջին գծի
bottom	Տեքստը հավասարեցվում է ըստ պատկերի ստորին սահմանի

Ըստ լրեյայն, այսինքն եթե align ատրիբուտը նշված չէ, տեքստը հավասարեցվում է ըստ պատկերի ստորին սահմանի (համապատասխանում է align="bottom" արժեքին): Ատրիբուտի տարբեր արժեքների ազդեցությունը տեքստի և պատկերի փոխադարձ դասավորության վրա ուսումնասիրելու նպատակով կազմենք հետևյալ HTML ֆայլը՝

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>align attribute using example</title></head>
<body>
<p>Տեքստը  հավասարեցվում
է ըստ պատկերի վերևի սահմանի align= "top"</p>
<p>Տեքստը 
հավասարեցվում է ըստ պատկերի միջին գծի align= "middle"</p>
<p>Տեքստը 
հավասարեցվում է ըստ պատկերի ստորին սահմանի align=
"middle"</p>
<p>Պատկերի <img src="" align="bottom" alt="PICTURE" /> փոխա-
րեն զետեղվում է alt ատրիբուտի արժեքը</p>
</body></html>
```

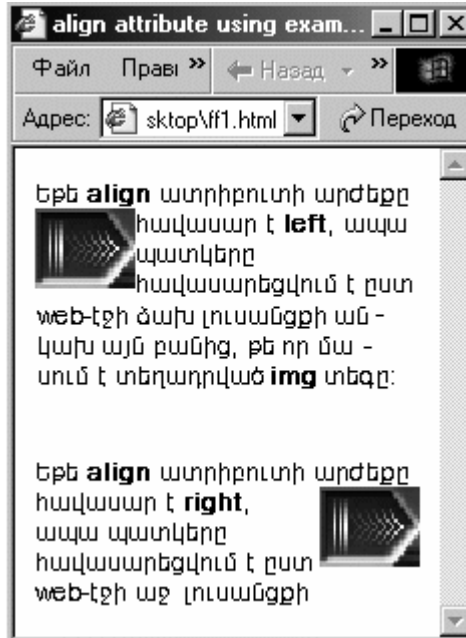
Ֆայլի արտապատկերման արդյունքը ցուցադրված է պատկեր 2.4.1-ում: "alt" ատրիբուտի աշխատանքը ցուցադրելու նպատակով չորրորդ <p> պարբերության մեջ միտումնավոր չի նշված

տեգի “src” ատրիբուտի արժեքը (պատկերը պարունակող ֆայլի հասցեն):



Պատկեր 2.4.1. Տեքստի և պատկերի փոխադարձ դասավորվածությունը

“align” ատրիբուտը կարող է ընդունել ևս երկու արժեքներ **“right”** (աջ) և **“left”** (ձախ), որոնք պատկերը դարձնում են, այսպես կոչված **“լողացող”**: Այդ արժեքների դեպքում պատկերը տեղադրվում է ոչ թե այն դիրքում, որտեղ տեղադրված է **“”** տեգը, այլ տեղաբաշխվում է web-էջի աջ (align=“right”) կամ ձախ (align=“left”) լուսանցքին հավասար, իսկ տեքստը շրջափակում է դրան: Պատկեր 2.4.2-ում ցուցադրված է պատկերի և տեքստի փոխադարձ դիրքավորումը “align” ատրիբուտի այդ արժեքների համար:



Պատկեր 2.4.2. align ատրիբուտի right և left արժեքների կիրառության օրինակները

HTML կոդի ֆրագմենտը հետևյալն է (հիշենք, որ այդ կոդը գրանցվում է մեր շաբլոնի <body> տեգի ներսում և File ->Save Us հրամանների միջոցով հիշվում է, որպես ".html" ընդլայնունով ֆայլ, որի անունը կարելի ընտրել ըստ ճաշակի)

<p>Եթե align ատրիբուտի արժեքը հավասար է left, ապա պատկերը հավասարեցվում է ըստ web-էջի ձախ լուսանցքի անկախ այն բանից, թե տեքստի որ մասում է տեղադրված img տեգը: </p>

<p>Եթե align ատրիբուտի արժեքը հավասար է right, ապա պատկերը հավասարեցվում է ըստ web-էջի աջ լուսանցքի</p>

 տեգի width (երկարություն) և height (բարձրություն) ատրիբուտների միջոցով կարելի է փոփոխել պատկերների իրական չափսերը: Օրինակ՝ կարելի է 300x200 պիքսել չափսեր ունեցող պատկերը ցուցադրել էկրանում 100x75 չափսերով (հիարկե՝

պատկերի ֆայլի ծավալը դրանից չի փոխվում): Սակայն դրանց օգտագործման հիմնական նպատակը՝ web-էջի բեռնման պրոցեսի արագացումն է: Քանի որ պատկերի չափսերը բրաուզերին հայտնի են դարձնում նախապես, ապա այն կարողանում է պատկերի համար տեղ հատկացնել մինչև բեռնման ավարտը, իսկ դա նշանակում է, որ սկզբում կբեռնվի տեքստը և այցելուն՝ կարողանալով անհրաժեշտ ինֆորմացիան, կարող է հղումներով բացել իրեն հետաքրքրող էջը, չսպասելով պատկերի բեռնման ավարտին:

2.4.2. Մուլտիմեդիայի օգտագործումը web-էջերում:

“Մուլտիմեդիա” նշանակում է ինֆորմացիայի համակարգչային այնպիսի պատկերացում, որը բաղկացած է տվյալների մեկից ավելիոն տեսակներից: Օրինակ՝ տեքստ և ձայն կամ վիդեո և ձայն: Սակայն հաճախ այդ թերմինը օգտագործում են, որպեսզի անվանեն մի ինչ որ հասկացություն, որն ավելի նշանակալի է տեքստից կամ գրաֆիկայից:

Մուլտիմեդիայի էլեմենտների մեծամասնությունը հիմնված է ժամանակից կախված այս կամ այն նյութերից, լինի դա ձայն, վիդեո թե անիմացիա: Մուլտիմեդիան դիտելու (կամ լսելու) պրոցեսը կարելի ընդհատել կամ վիդեոշարքը և ձայնը “հետ պտտել” սովորական վիդեո- կամ ձայնահոլովակների պես: Համակարգչային տեխնոլոգիաները, իհարկե, ավելի բարդ են: Օրինակ՝ վիդեոշարքը ուղեկցվում է համաչափված (սինխրոնացված) ձայն-ուղիով, սակայն սկզբունքը նույնն է՝ ժամանակի ընթացքում փոփոխվող պատկերներ: Նույնը կարելի ասել ձայնային ֆայլերի վերաբերյալ, որոնք իրենցից ներկայացնում են բարձր հաճախականությամբ ձայնարկվող հազարավոր “հնչյունակտորներ” (սամփլներ՝ անգլերեն sample բառից), ընդ որում հնչողության որակը կախված է վերարտադրության հաճախականությունից և սամփլների կառուցվածքից:

➤ Համակարգչային ֆորմատի բերված վիդեո- կամ ձայնագրությունները կոչվում են թվային:

Նախկան մուլտիմեդիայի էլեմենտները web-էջում օգտագործելը հարկավոր է պարզաբանել երեք հիմնական հարցեր՝

- Անհրաժեշտ է արդյոք դրանք օգտագործել:
- Ի՞նչ տիպի ֆայլեր օգտագործել, այցելուների ուշադրությանը արժանանալու համար:
- Ինչպիսի՞ եղանակ ընտրել մուլտիմեդիայի վերարտադրության համար: Այսինքն՝ օգտագործել օժանդակ հավելվածներ, ներդնել մուլտիմեդիայի էլեմենտները էջում, թե՞ թողնել այդ ամենը բրաուզերի “խղճին”:

Ինտերնետում օգտագործվում են 20-ից ավելին տեսակի մուլտիմեդիա ֆայլեր, որոնցից մի քանիսը (ընդլայնումներով) բերված են աղյուսակ 2.4.2-ում: Նշենք, որ QuickTime և RealMedia ֆորմատները իրենցից ներկայացնում են տվյալների հոսքեր: Դա նշանակում է, որ վիդեոն կամ ձայնը սկսում է ցուցադրվել (կամ ձայնարկվել) օգտվողի համակարգչում ֆայլի բեռնման ընթացքում, այլ ոչ թե այն ժամանակ երբ ֆայլը բեռնվել է լիովին: Ընդ որում այդ ֆորմատները կարող են օգտագործվել նաև զուտ որպես ձայնային:

Բրաուզերների մեծամասնությունը սատարում է սահմանափակ թվով ֆորմատներ: Ընդհանուր դեպքում սովորական բրաուզերը կարող է արտապատկերել HTML փաստաթղթեր, սովորական տեքստ և առավել տարածված գրաֆիկական ֆորմատների ֆայլեր: Որոշ բրաուզերներ կարողանում են հասանելիություն ստանալ հեռացված ինտերնետ-սերվերներում գրանցված ինֆորմացիային և, ներմուծել ու արտապատկերել այն օգտագործելով FTP կամ Usenet արձանագրությունները: Գոյություն ունեն բրաուզերներ, որոնք կարող են ձայնարկել WAV ֆորմատի ձայնային ֆայլեր: Սակայն ընդհանուր դեպքում մուլտիմեդիան ընկալելու հնարավորությունները բրաուզերներում ներկառուցված չեն: Մուլտիմեդիայի վերծանման գործառնությունները կատարվում են օժանդակ հավելվածների (**accessories**) կամ հատուկ գրադարանների (այսպես կոչված պլագինների՝ **plugins**) միջոցով:

Աղյուսակ 2.4.2.

Մուլտիմեդիա ֆայլերի ֆորմատները

Ֆայլի ֆորմատը	Ֆայլի տեսակը	Ընդլայնումը
Windows sound	Թվային աուդիո	.wav
MPEG/MP3 audio	Թվային աուդիո	.mpg / mp3
MIDI audio	Չայնի դեկավարման հրամաններ	.mid, .midi
RealMedia	Աուդիո/վիդեո հոսք	.ra, .rm, .ram
MPEG video	Թվային վիդեո	.mpg, .mpeg
QuickTime	Թվային վիդեո	.mov, .qt
Microsoft Media	Թվային վիդեո	.avi,
Macromedia Shockwave Flash	Անիմացիա	.swf

Օժանդակ հավելվածը ծրագիր է, որը սկսում է աշխատել ավտոմատ կերպով, երբ անհրաժեշտ է վերծանել իրեն հետ կապված այս կամ այն տեսակի ֆայլ: Այդ դեպքում ընդունված է ասել, որ

մուլտիմեդիա ֆայլը կապակցվում է web-էջի հետ “**հիպերմեդիա հղումով**”, այլ ոչ թե “հիպերհղումով”:

Պլագինները՝ ծրագրային ոչ մեծ ֆայլեր են, որոնք սովորաբար տեղադրվում են կոշտ սկավառակի վրայի հատուկ ենթաանվանացանկում (ենթակատալոգում), ընդ որում դրանք ակտիվանում են բրաուզերի բեռնման ժամանակ: Պլագինները անմիջականորեն համագործակցում են բրաուզերի հետ մուլտիմեդիան դիտելու կամ/և լսելու համար, ընդ որում ամեն ինչ կատարվում է միևնույն պատուհանում: Մուլտիմեդիայի էլեմենտը ընկալվում է որպես էջի մի մաս և այդ պատճառով խոսքը արդեն գնում է ոչ թե կապակցման, այլ ներդրման մասին:

Ներդրվող ֆորմատներից կարելի է նշել QuickTime, RealMedia, Windows Media և Macxromedia Shockwave Flash-ը: Իսկ կապակցվողներից, օրինակ՝ MP3 ֆորմատի ֆայլերը, որոնք գործարկելու համար օգտագործվում է, օրինակ Windows Media Player օժանդակ հավելվածը:


Սայթի ձևավորման (դիզայնի) տեսակետից կապակցման կամ ներդրման հարցը ունի այլ կտրվածք: Աանհրաժեշտ է որոշել թե ինչպիսի՞ տեսքով պետք է ներկայացնել մուլտիմեդիայի էլեմենտը՝ որպես էջի մաս, թե առանձին ֆայլ: Եթե համապատասխան պլագինը առկա է և ֆայլը հետագայում չի նախատեսվում օգտագործել, ապա ավելի հարմար է ներդնել այն էջում: Իսկ եթե ֆայլը պետք է հետագայում օգտագործվի նաև մյուս էջերում, ապա իհարկե նախընտրելի է այն կապակցել և գործարկելու համար ամեն անգամ օգտագործել համապատասխան օժանդակ հավելվածը:

Այժմ քննարկենք web-էջերի և մուլտիմեդիայի էլեմենտների համատեղման երկու մոտեցումների՝ ներդրման և կապակցման կիրառությունը:

Մուլտիմեդիայի էլեմենտների կապակցման համար, ինչպես ասվեց, օգտագործվում են հիպերմեդիա հղումները, որոնք քչով են տարբերվում սովորական հիպերհղումներիցից: Սակայն գոյություն ունի մեկ էական տարբերություն՝ հղումը կատարվում է ոչ թե փաստաթղթին, որը պետք է արտապատկերվի բրաուզերի պատուհանում, այլ որոշակի ֆայլին: Բրաուզերը պարզում է ֆայլի տեսակը և բեռնում է համապատասխան օժանդակ հավելվածը, որն էլ ցուցադրում է մուլտիմեդիա էլեմենտը համապատասխան տեսքով: Բերենք MP3 ֆորմատի ֆայլին հիպերմեդիա հղումի օրինակ՝

Սիրելի առաջնորդի ամանօրյա շնորհավորանքները (1.2 Mb):

Երբ օգտվողը կտտացնում է հղումին, ֆայլը բեռնվում է համակարգչի կոշտ սկավառակի վրա: Եթե բրաուզերը չի կարողանում գտնել համապատասխան ֆորմատը սպասարկող օժանդակ հավելված, ապա կարելի է պարզապես պահպանել այդ ֆայլը հիշողությունում հետագա օգտագործման նպատակով:

Մուլտիմեդիայի էլեմենտների ներդրումը էջում նման է  տեգի օգտագործմանը: Ընդհանուր դեպքում ներդրումը նշանակում է, որ էջում տեղ է հատկացվում միացվող էլեմենտի համար՝ հետագայում պլագինի միջոցով այն մշակելու նպատակով:

Ինչ ինչպես է իրականացվում այդ “կախարդական” ներդրումը: Մինչ այժմ դրա համար օգտագործվում էր լայն կիրառություն ստացած **<embed>**, **</embed>** տեգ-կոնտեյները: Տեգի պարտադիր ատրիբուտն է գործարկվող մեդիա ֆայլի URL հասցեն: Օրինակ՝ **<embed src="mymovie.mov"></embed>**:

Միանգամայն ակնհայտ է, որ ցուցադրվող մեդիայի (ֆիլմեր, կլիպեր, ֆլաշ-ֆայլեր և այլն) համար պետք է նաև նախատեսել ցուցադրման համար նախատեսված պատուհանի չափսերը՝ լայնությունը (width) և բարձրությունը (height): Չափազանց օգտակար ատրիբուտ է նաև **pluginspace** ատրիբուտը, որը օգնում է բրաուզերին գտնել ցանցում (իհարկե, եթե օգտվողը միացած է վերջինին) անհրաժեշտ պլագինը, եթե այն բացակայում է օգտվողի համակարգչում: Բացի այդ շատ դեպքերում **<embed>** տեգում ընդգրկվում է դրա անունը բնորոշող ատրիբուտը (name), որը օգտակար է սցենարներում օգտագործելու դեպքում: Օրինակ՝ **<embed name="movie1" src="movie1.mov" width="100" height="60" pluginspace="http://www.apple.com/quicktime/download/"></embed>**

Առավել հետաքրքիր է href ատրիբուտի կիրառությունը: Այն թույլ է տալիս բեռնել և ցուցադրել սկզբում ոչ մեծ պատուհանում միայն փոքրածավալ, հաճախ միջին որակի ցուցադրական ֆայլը (իհարկե այն պետք է նախապես պատրաստվի): Այն դեպքում, երբ օգտվողը ցանկանում է դիտարկել ֆայլը լիովին և նորմալ որակով, ապա կարող է օգտվել ներդրված վիդեոյի պատուհանից ինչպես հիպերհղումից և անցնել href ատրիբուտում գրանցված հասցեով գտնվող ֆայլի դիտարկմանը: Եթե օրինակ, **<embed>** տեգը գրանցվի հետևյալ կերպ՝ **<embed src="fastmovie.mov" autoplay="true" autoplay="true" controller="false" href="fullmovie.mov"></embed>**, ապա սկզբում կբեռնվի և անմիջապես (քանի որ **autoplay="true"**) կցուցադրվի **fastmovie.mov** ցուցադրական ֆայլը, որի պատուհանը կարտապատկերվի հիպերհղումի տեսքով: Այժմ նորմալ՝ **fullmovie.mov** ֆայլը կցուցադրվի միայն այն դեպքում, երբ այցելուն մկնիկով կտտացնի այդ հիպերհղումին:

Սակայն պլագիները հետ աշխատելու մեխանիզմը տեղ չի գտել XML ստանդարտում (և, նույնիսկ XHTML-ի խիստ տարբերակում): Եվ եթե մեդիայի ներդրման համար փատաթղթում օգտագործվում է միայն `<embed>` տեգը, անհրաժեշտ է պարտադիր կերպով նշել դրա DTD-ն (զոնե որպես XHTML-ի անցումային վարկած): Internet Explorer 5.5 և դրան հաջորդող վարկածները չեն սատարում `<embed>`-ը, ստիպելով հեղինակներին կիրառել մի այլ՝ ստանդարտին համապատասխանող `<object>...<object>` տեգը կամ, նույնիսկ, Microsoft ֆիրմայի սեփական **ActiveX** տեխնոլոգիան:

Այդ պրոբլեմը լուծելու համար Apple ֆիրման արաջարկել է երկու տեգերի համատեղ կիրառությունը՝ `<embed>` տեգը գրանցվում է `<object>` տեգի ներսում և, հին բրաուզերները, անտեսելով `<object>`-ը կօգտվեն `<embed>`-ից, իսկ նորերը կանտեսեն `<embed>`-ը և կօգտվեն `<object>` տեգից: Բերենք QuickTime ֆորմատի վիդեոֆայլի ներդրման օրինակ:

```
<object
classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
width="300" height="220"
codebase=""http://www.apple.com/qtactivex/qtplugin.cab">
<param name="src" value="moviel.mov" />
<param name="autoplay" value="true" />
<param name="controller" value="false" />
<embed src="movie.mov" width="320"
height="240" autoplay="true" controller="false"
pluginspage="http://www.apple.com/quicktime/download/">
</embed>
</object>
```

Ինչպես տեսնում ենք՝ `<object>` տեգը նույնպես ունի ատրիբուտներ, որոնցից width և height ատրիբուտների նշանակումը մեզ արդեն հայտնի է:

classid (class identifier) ատրիբուտի արժեքը՝ դա պլագինի կամ ActiveX օբյեկտի ընդունված իդենտիֆիկատորն է:

codebase ատրիբուտը թույլ է տալիս ավտոմատ կերպով լրացնել ծրագրային էլեմենտների պակասը ցանցից այն դեպքում, երբ վերջինները բացակայում են օգտվողի համակարգչում:

Մուլտիմեդիայի ծայնարկումը կամ դիտարկումը կարգավորող բոլոր մյուս ատրիբուտներին արժեքները շնորհվում են միջնորդված եղանակով՝ հատուկ ներդրված եզակի `<param />` տեգերի միջոցով (համեմատության համար՝ `<embed>` տեգում դա կատարվում է անմիջապես տեգը բացող բաղադրիչի ներքո): Յու-

րաքանցյուր այդպիսի տեգի name ատրիբուտում նշվում է կարգավորող էլեմենտի անունը, իսկ value-ում այդ էլեմենտի կարգավորվող վիճակի արժեքը:

Ինչպես տեսնում ենք, բերված օրինակում հին բրաուզերները “չնեղացնելու” և Apple ֆիրմայի խորհուրդին հետևելու նպատակով <object> տեգում ներդրված է նաև <embed> տեգը:

§2.5. Web-էջերի աղյուսակային ձևավորումը

2.5.1. Աղյուսակների կառուցման հիմունքները

Մեզ արդեն հայտնի են (տես՝ §2.2.) web-էջերում տեքստի տրամաբանական կազմակերպման այնպիսի միջոցներ, ինչպիսիք են, օրինակ, պարբերությունները և ցուցակները: Տրամաբանական կազմակերպման մի այլ՝ ավելի հզոր միջոցներ են աղյուսակները, որոնք օգտագործվում են էջերը սյունակների և տողերի բաժանելու նպատակով: Աղյուսակների օգնությամբ կարելի է ավելի հարմար և ակնառու կերպով տեղաբաշխել տվյալները, տեքստը, հղումները և, նույնիսկ, պատկերները:

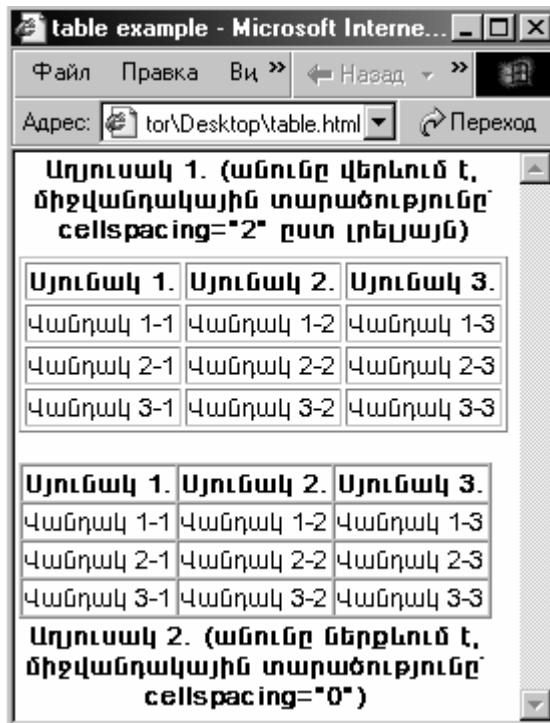
Ցանկացած աղյուսակ HTML-ում պառփակվում է <table>... </table> զույգ տեգի ներքո: Տեգի բացող և փակող էլեմենտների միջև տեղաբաշխվում են աղյուսակի տողերը, որոնք սահմանվում են <tr>...</tr> (table row - աղյուսակի տող) կոնտեյներների միջոցով: Յուրաքանչյուր տողում ընդգրկվում է այնքան վանդակ, ինչքան սյունակ է նախատեսված տվյալ աղյուսակում: Վանդակները աղյուսակի փոքրագույն կազմավորող միավորներն են, որոնց մեջ էլ գրանցվում են բոլոր տվյալները: Վանդակները որոշվում են <td>...</td> (table division) տեգերի միջոցով:

Աղյուսակը կարող է ունենալ անվանում, որը գրանցվում է <caption>... </caption> տեգում <table> բացող տեգից անմիջապես հետո: Բրաուզերում այն արտապատկերվում է աղյուսակից վեր: Ցանկության դեպքում անվանումը կարելի է գրանցել աղյուսակի տակ, շնորհիվ տեգի align ատրիբուտին bottom արժեքը

Յուրաքանչյուր սյունակին կարելի է շնորհել վերնագիր՝ գծանշելով առաջին տողի վանդակները <th>... </th> (table header) տեգի միջոցով: Այն տարբերվում է <td>-ից միայն նրանով, որ տեքստը գրանցվում է ավելի թավ տառաճիշերով (տես՝ պատկեր 2.5.1.):

Աղյուսակ ստեղծելիս անհրաժեշտ է հիշել, որ բոլոր տողերի սյունակների քանակը պետք է համապատասխանի առաջին տողի սյունակների քանակին: Օրինակ, եթե առաջին տողում գրանցված է երեք <th> կամ <td> (երեք վանդակ), ապա մյուս բոլոր տողերում վանդակների քանակը պետք է հավասար լինի երեքի: Իհարկե,

կան նաև միջոցներ, որոնք թույլ են տալիս անհրաժեշտության դեպքում արհեստականորեն միավորել վանդակները՝ դրանք մենք կքննարկենք քիչ ուշ:



Պատկեր 2.5.1. Աղյուսակների օրինակներ

<table> տեգն ունի բազմաթիվ ատրիբուտներ, որոնցից առավել հաճախ օգտագործվողը border ատրիբուտն է, որի միջոցով սահմանվում է աղյուսակի արտաքին շրջանակի և բոլոր ներքին վանդակները բաժանող, գծերի հաստությունը (ըստ լրեցյալն այն հավասար է 0-ի): Օրինակ, եթե գրանցենք border = "1", ապա բոլոր նշված գծերի հաստությունը կկազմի 1 պիքսել: Կարևոր նշանակություն ունի նաև cellpadding ատրիբուտը, որի միջոցով սահմանվում է տարածությունը վանդակների միջև (ըստ լրեցյալն այն հավասար է երկուսի): Եթե ատրիբուտի արժեքը զրո չէ, ապա ստեղծվում է այնպիսի տպավորություն, որ յուրաքանչյուր վանդակ վերցված է առանձին շրջանակի մեջ: Պատկերում բերված են վերը նշված

ատրիբուտների տարբեր արժեքներ ունեցող երկու աղյուսակների օրինակներ: Համապատասխան ծրագրային կոդը հետևյալն է՝

```
<table border="1"><caption><b>Այս աղյուսակի միջվանդակային տարածությունը 2 է (ըստ լրելայն)</b></caption>
<tr><th>Սյունակ 1.</th><th>Սյունակ 2.</th> <th> Սյունակ 3. </th>
</tr>
<tr><td>Վանդակ 1-1</td><td> Վանդակ 1-2</td><td> Վանդակ 1-3</td></tr>
<tr><td> Վանդակ 2-1</td><td> Վանդակ 2-2</td><td> Վանդակ 2-3</td></tr>
<tr><td> Վանդակ 3-1</td><td> Վանդակ 3-2</td><td> Վանդակ 3-3</td></tr>
</table>
```

```
<table border="1" cellspacing="0">
<caption><b>Իսկ այս աղյուսակի միջվանդակային տարածությունը զրո է</b></caption>
<tr><th>Սյունակ 1.</th><th> Սյունակ 2.</th> <th> Սյունակ 3.</th>
</tr>
<tr><td>Վանդակ 1-1</td><td> Վանդակ 1-2</td><td> Վանդակ 1-3</td></tr>
<tr><td> Վանդակ 2-1</td><td> Վանդակ 2-2</td><td> Վանդակ 2-3</td></tr>
<tr><td> Վանդակ 3-1</td><td> Վանդակ 3-2</td><td> Վանդակ 3-3</td></tr>
</table>
```

Տողերի պարունակությունը հավասարեցնելու համար կարելի է օգտագործել երկու ատրիբուտներ՝ align (հորիզոնական) և valign (ուղղահայաց): Եթե այդ ատրիբուտներին արժեքները չնորիվեն <tr> տեգում, ապա հավասարեցումը կվերաբերվի տվյալ տողի բոլոր վանդակներին:

Հորիզոնական հավասարեցումը կարելի է կատարել տողի վանդակների ձախ (align="left", ըստ լրելայն) և աջ (align="right") սահմաններով, ինչպես նաև կենտրոնով (align="center"):

Ուղղահայաց հավասարեցումը կատարվում է ըստ տողի վանդակների վերին (valign="top"), ստորին (valign="bottom") սահմանների և միջին գծով (valign="middle", ըստ լրելայն): Օրինակ, եթե պետք է տողի վանդակների պարունակությունը հավասարեցնել դրանց աջ սահմանով և միջին գծով, ապա կարելի է գրանցել՝

```
<tr align="right" valign="middle">տողի վանդակները</tr>:
```

Նույն ատրիբուտները կարելի է կիրառել նաև առանձին տողերի և վանդակների պարունակությունը հավասարեցնելու համար՝

պարզապես արժեքը ատրիբուտին շնորհվում է տողը կամ վանդակը գծանշող <tr> կամ <td> տեղերում:

Չորիզոնական (align) և ողղահայաց (valign) հավասարեցում:

Տող 1 - Ըստ լրեյան համապատասխանում է align="left", valign="middle"

Տող 2 - align="right", valign="top"

Տող 3 - align="center" valign="bottom"

Տող 4 - տողը հավասարեցված է աջով՝ align="right", իսկ առաջին վանդակը կենտրոնով՝ align="center"

Տող 5 - տողը հավասարեցված է կենտրոնով և ստորին սահմանով՝ align="center" valign="bottom", իսկ երկրորդ վանդակը աջով և միջին գծով՝ align="right" valign="middle"

Տող 6 - Առաջին վանդակը՝ տողի վերնագրային վանդակն է՝ scope="row"

Սյունակ 1	Սյունակ 2	Սյունակ 3
1-1	1-2	1-3
2-1	2-2	2-3
3-1	3-2	3-3
4-1	4-2	4-3
5-1	5-2	5-3
Տող 6	6-2	6-3

Պատկեր 2.5.2. Տողերի և վանդակների պարունակության հավասարեցման օրինակներ

Վերնագրային <th> տեղը ունի ևս մեկ հետաքրքիր ատրիբուտ՝ **scope**, որի միջոցով կարելի է սահմանել, թե ինչի՞ համար է վերնագիր ծառայելու տվյալ վանդակը՝ տողի թե՞ սյունակի: Ըստ լրեյայն ատրիբուտի արժեքն է scope="col", ինչը նշանակում է սյունակի վերնագիր: Եթե ատրիբուտին շնորհվի scope="row" արժեքը, ապա տվյալ վանդակը վերնագրային կլինի տողի համար:

Պատկեր 2.5.2-ում բերված են աղյուսակի տողերի և վանդակների պարունակության հավասարեցման տարբեր օրինակներ:

Հաճախ անհրաժեշտություն է առաջանում միավորել մեկ կամ մի քանի վանդակներ մեկում: Պատճառները կարող են լինել տարբեր, օրինակ՝ բոլոր վանդակները պարունակում են միևնույն ինֆորմացիան կամ անհրաժեշտ է մի քանի սյունակների համար ստեղծել միացյալ վերնագիր և այլն: Անկախ պատճառներից գոյություն ունեն երկու ատրիբուտներ այդ նպատակը իրագործելու համար՝ **colspan** և **rowspan**: Առաջինը թույլ է տալիս մեկում ընդգրկել երկու կամ ավելի հորիզոնական, իսկ երկրորդը՝ ուղղահայաց վանդակներ: Օրինակ, եթե գրանցենք `colspan="3"`, ապա մեկ սյունակում կընդգրկվեն երեք հարևան վանդակներ: Բերենք օրինակ՝

```
<html><head><title>Colspan and rowspan attributes example</title>
</head><body>
<table border="1">
<caption>Համակարգիչների բնութագրերը</caption>
<tr><th>&nbsp;</th><th>Մոդել 100</th>
<th>Մոդել 200</th></tr>
<tr><th>Պրոցեսոր</th><td>G9 – 1.6</td>
<td>G9 – 1.7</td></tr>
<tr><th>HDD</th><td>78 Gb</td><td>90 Gb</td></tr>
<tr><th>Վիդեոքարտ</th>
<td colspan="2" align="center">Rageos 428</td></tr>
<tr><th>Ա/Վ ելք</th><td rowspan="2" align="center"
valign="middle">Չկա</td>
<td>Կա</td></tr>
<tr><th>CD պորտ</th><td>Օպտիոնալ</td></tr>
</table>
</body></html>
```

Ինչպես տեսնում ենք աղյուսակի չորրորդ տողում գրանցված է ընդամենը երկու վանդակ, ընդ որում երկրորդ վանդակի `colspan` ատրիբուտին շնորհիվ է 2 արժեքը: Դա նշանակում է, որ այդ վանդակը պետք է զբաղեցնի երկու վանդակի տարածք (մեր օրինակում՝ մյուս տողերի երկրորդ և երրորդ վանդակների չափով): Հինգերորդ տողի երկրորդ վանդակի `rowspan="2"` նշանակում է, որ այդ վանդակը կընդլայնվի մաս հաջորդ տողի երկրորդ վանդակի չափով: Եվ հենց այդ պատճառով հաջորդ (6-րդ) տողում նույնպես նախատեսված են ընդամենը երկու վանդակներ: Ծրագրի աշխատանքի արդյունքը ցուցադրված է պատկեր 2.5.3-ում:

Colspan and rowspan attribute...

Файл Правка >> << Назад >>

Адрес: or\Desktop\vf1.html Переход

Համակարգիչների բնութագրերը

	Մոդել 100	Մոդել 200
Պրոցեսոր	G9 – 1.6	G9 – 1.7
HDD	78 Gb	90 Gb
Վիդեոքարտ	Rageos 428	
Ա/Վ ելք	Չկա	Կա
CD պորտ		Օպտիոնալ

Պատկեր 2.5.3. Աղյուսակի վանդակների միավորման օրինակներ

2.5.2. Web-էջերի աղյուսակային ձևավորումը

Աղյուսակային գծանշման տեղերը թույլ են տալիս տեղաբաշխել web-փաստաթղթերում ոչ միայն սովորական տեքստային ինֆորմացիա: Աղյուսակները հզոր զենք են հանդիսանում web-էջերի ֆորմատավորման գործընթացում, քանի որ դրանց կիրառությունը հնարավորություն է ընձեռնում կարգավորել էջի արտաքին տեսքը, առավել ակնառու և արդյունավետ եղանակով տեղաբաշխելով պարբերությունները, հիպերհղումները և գրաֆիկան: Այդ ամենը իրագործվում է աղյուսակների վանդակները էջի համապատասխան մասերում տեղադրելու, դրանց չափսերը մեծացնելու կամ փոքրացնելու, միջվանդակային և ներվանդակային տարածությունները փոփոխելու միջոցով: Բացի այդ HTML-ում հնարավոր է նաև միմյանց մեջ ներդնել մի քանի աղյուսակներ:

Ընդհանուր դեպքում աղյուսակային գծանշումը թույլ է տալիս էջը պարզապես բաժանել այնպիսի տրամաբանական մասերի, որոնցում հետագայում պետք է տեղադրվի այն ամենը, ինչ մատուցվելու է այցելուներին: Այդպիսի տրամաբանական բաժանումը լայնորեն կիրառվում է օրինակ՝ թերթերում և ամսագրերում, որոնցում յուրաքանչյուր էջ տեսողականորեն բաժանվում է մի քանի բաղադրյալ մասերի՝ դա ապահովում է նյութերի առավել դյուրին ընկալումը ընթերցողների կողմից:

Աղյուսակի լայնությունը սահմանվում է **width** ատրիբուտի միջոցով, ընդ որում դա կարելի է անել երկու եղանակներով՝ բացարձակ միավորներով (պիքսելներով), հարաբերական եղանակով՝ տոկոսներով:

Եթե, օրինակ, ցանկանում ենք արտապատկերել աղյուսակ, որի լայնությունը հավասար լինի 150 պիքսելի, ապա պետք է գրանցենք՝ **<table width="150">**: Իսկ եթե նախատեսել ենք, որ աղյուսակը պետք է զբաղեցնի էկրանի լայնության 80 տոկոսը, ապա գրանցումը ստանում է հետևյալ տեսքը՝ **<table width="80%">**:

Կարելի է, իհարկե, սահմանել մեկ (օրինակ՝ առաջին) տողի առանձին վանդակների լայնությունները և ստանալ աղյուսակի ընդհանուր լայնությունը որպես գումարային, սակայն համաձայն նոր ստանդարտի width ատրիբուտը չի սատարվելու <td> տեգի նկատմամբ: Գոյություն ունի նաև այդ հարցի լուծման ևս մեկ եղանակ՝ կարելի է յուրաքանչյուր վանդակում ներդնել առանձին աղյուսակ և սահմանել վերջինիս լայնությունը:

- Նշենք, որ աղյուսակների և վանդակների լայնությունները հաշվելիս անհրաժեշտ է հաշվի առնել նաև շրջանակների (border) լայնությունը, միջվանդակային տարածությունը (cellspacing) և ներվանդակային լուսանցքների (cellpadding) լայնությունը:

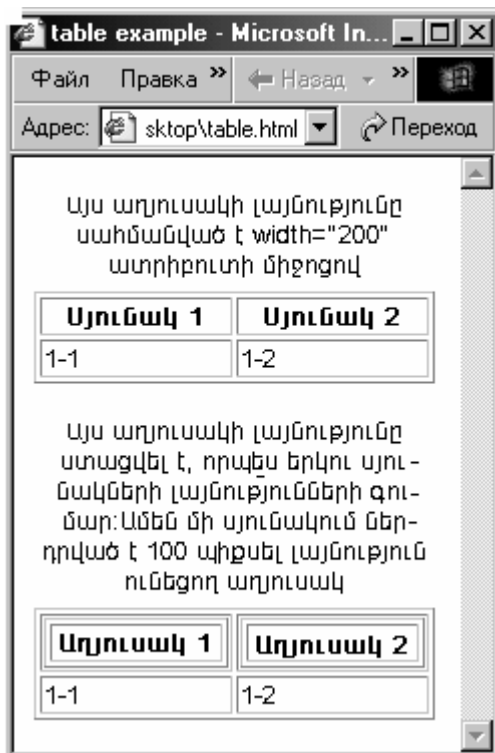
Բերենք աղյուսակների լայնության սահմանման երկու օրինակ: Առաջին դեպքում աղյուսակի լայնությունը որոշվում width ատրիբուտի միջոցով, իսկ երկրորդում՝ առաջին տողի յուրաքանչյուր վանդակում ներդրված է մի այլ, համապատասխան լայնություն ունեցող աղյուսակ:

Առաջին աղյուսակի կոդը հետևյալն է՝

```
<table border="1" width="200">
<tr><th>Սյունակ 1</th><th>Սյունակ 2</th></tr>
<tr><td>1-1</td><td>1-2</td></tr>
</table>,
երկրորդին՝
<table border="1">
<tr><th><table border="1" width="92">
<tr><th>Սյունակ 1</th></tr>
</table>
</th><th>
<table border="1" width="92">
<tr><th>Սյունակ 2</th></tr>
</table>
</th></tr>
<tr><td>1-1</td><td>1-2</td>
```

</table>

Ներդրված աղյուսակների լայնությունը (92 պիքսել) ընտրված է այնպես, որ հաշվի առնվեն նաև արտաքին աղյուսակի շրջանակները, միջվանդակային տարածությունը և ներվանդակային լուսանցքների լայնությունը: Երկու աղյուսակներն էլ արտապատկերվում են հավասար լայնությամբ (տես՝ պատկեր 2.5.4.):



Պատկեր 2.5.4. Աղյուսակների լայնության սահմանման օրինակներ

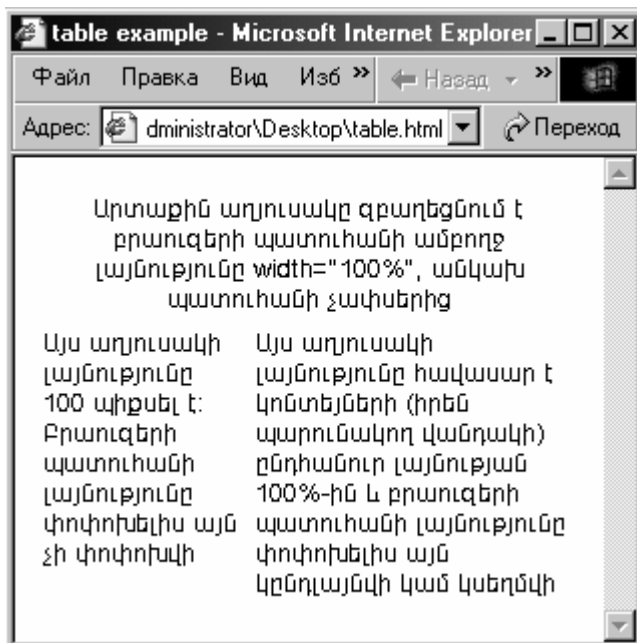
Որոշ դեպքերում նպատակահարմար է տեքստը ներկայացնել մի քանի իրարից որոշակի հեռավորության վրա գտնվող տարբեր կամ հավասար լայնության սյունակների տեսքով (տեքստի այդպիսի դասավորություն կարելի է տեսնել, օրինակ, թերթերում): Այդպիսի դեպքերում կարելի է օգտագործել “անտեսանելի” աղյուսակներ, որոնք բաժանում են տեքստը ըստ սյունակների և, անհրաժեշտության դեպքում հավասարեցնում են ըստ տողերի:

Իհարկե պետք է բավականին “քրտնել”, որպեսզի տեքստը արտապատկերվի կորեկտ, սակայն դա կփոխհատուցվի նրանով, որ այդպիսի ներկայացումը կախված չի լինի բրաուզերների վարկածներից և օգտվողի էկրանի “բացվածքից”:

Բերենք պարզագույն օրինակ, ընդ որում սյունակների տարբեր լայնությունները ապահովելու համար օգտվենք նաև ներդրված աղյուսակներից: Օրինակում արտաքին աղյուսակն ունի երկու սյունակ մեկ տողում և լայնությունը սահմանված է հավասար 100%-ի: Դա նշանակում է, որ այն կգբաղեցնի իրեն պարունակող կոնտեյների (տվյալ դեպքում դա փաստաթուղթի <body> տեգն է) ամբողջ լայնությունը, անկախ բրաուզերի պատուհանի լայնությունից: Աղյուսակի վանդակներում ներդրված են աղյուսակներ, ընդ որում ձախի համար լայնությունը սահմանված է բացարձակ եղանակով` width="100", իսկ աջինը իրեն կոնտեյների (այսինքն պարունակող վանդակի) լայնության տոկոսներով` տվյալ դեպքում` 100%: Ստորև բերված է ծրագրային կոդի մասը, որի աշխատանքի արդյունքը ցուցադրված է պատկեր 2.5.5-ում:

```
<table border="0" width="100%" cellspacing="0" cellpadding="3">
<caption>Արտաքին աղյուսակը զբաղեցնում է բրաուզերի
պատուհանի ամբողջ լայնությունը width="100%", անկախ
պատուհանի չափսերից </caption><br />
<tr valign="top">
<td><table border="0" width="100%" cellspacing="0" cellpadding="1">
<tr><td>Այս աղյուսակի լայնությունը 100 պիքսել է: Բրաուզերի
պատուհանի լայնությունը փոփոխելիս այն չի փոփոխվի</td>
</tr></table>
</td> <td>
<table border="0" width="100%" cellspacing="0" cellpadding="1"
valign="top">
<tr><td>Այս աղյուսակի լայնությունը հավասար է կոնտեյների
(իրեն պարունակող վանդակի) ընդհանուր լայնության 100%-
ին և բրաուզերի պատուհանի լայնությունը փոփոխելիս այն
կընդլայնվի կամ կսեղմվի</td>
</tr>
</table>
</td> </tr></table>
```

Հետևելով նոր ստանդարտին, W3C կոնսորցիումը ընդլայնել է աղյուսակների ֆունկցիոնալ հնարավորությունները, ավելացնելով մի քանի նոր տեգեր: Դրանք են` “<thead> </thead>”, “<tbody> </tbody>”, “<tfoot> </tfoot>”, “<colgroup> </colgroup>”, “<col />” տեգերը:



Պատկեր 2.5.5. Տեքստի սյունակային դասավորության օրինակ

Առաջին երեքը թույլ են տալիս բաժանել աղյուսակի տողերը երեք անկախ տրամաբանական խմբերի՝ վերնագրային մաս (<thead>), հիմնական մաս կամ աղյուսակի “մարմին” (<tbody>) և ստորին մաս (<tfoot>)՝ կոլոնտիտուլ: Այդպիսի բաժանման հիմնական իմաստը կայանում է նրանում, որ նշված երեք տրամաբանական մասերը ստեղծելուց հետո հնարավոր է դառնում “պտտել” աղյուսակի մարմինը, անշարժ թողնելով վերին և ստորին մասերը: Դա հատկապես հարմար է բազմաթիվ տողեր ունեցող աղյուսակների համար, երբ հնարավոր չէ դրանք լիովին արտապատկերել էկրանին: Առայժմ այդ ֆունկցիան չի սատարվում ոչ մի բրաուզերի կողմից, սակայն հույս փայփայենք, որ այն շատ շուտով կսկսի աշխատել:

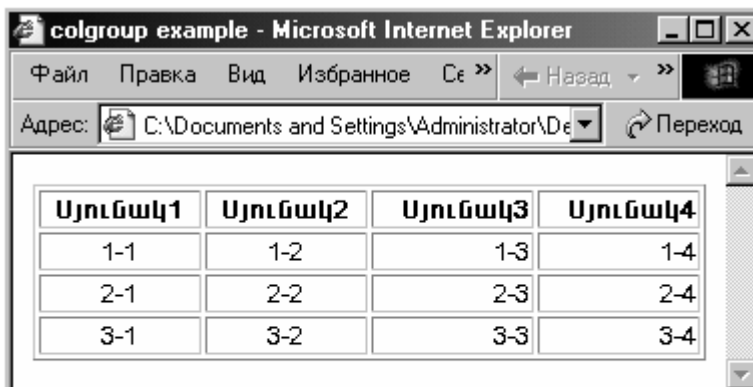
Ինչքան էլ դա կարող է տարրորինակ թվալ, այն դեպքում երբ մենք որոշում ենք ընդգրկել աղյուսակում վերին և ստորին մասերը՝ <tfoot>-ը պետք է սահմանել նախկան <tbody>-ի սահմանումը, օրինակ՝

```
<table border="1">
```



```
|  |  |  |  |
| --- | --- | --- | --- |
| 3-1 | 3-2 | 3-3 | 3-4 |

```



Աղյուսակի առաջին երկու սյունակներում հավասարեցումը կատարված է ըստ վանդակների կենտրոնի և միջին գծի (`align="center" valign="middle"`), իսկ մյուս երկուսում՝ ըստ աջ եզրագծի և նույախ միջին գծի (`align="center" valign="middle"`):

Պատկեր 2.5.6. Աղյուսակի սյունակների խմբավորման օրինակ

Երկրորդ օրինակում կառուցվում է աղյուսակ որն ունի երկու՝ 3 և 2 սյունակներ ընդգրկող խմբերի բաժաված 5 սյունակներ: Սակայն առաջին խմբի առաջին սյունակի լայնությունը տարբերվում է մյուս երկուսինից, իսկ երկրորդ խմբի սյունակները տարբեր են ըստ հավասարեցման բնութագրիչի: Այդ տարբերությունները գրանցելու համար օգտագործվում են `<col>` տեղը `<colgroup>` տեղերի ներքո (տես նաև պատկեր 2.5.7)՝

```
<colgroup span="3" align="center">
```

```
<col width="70" />
```

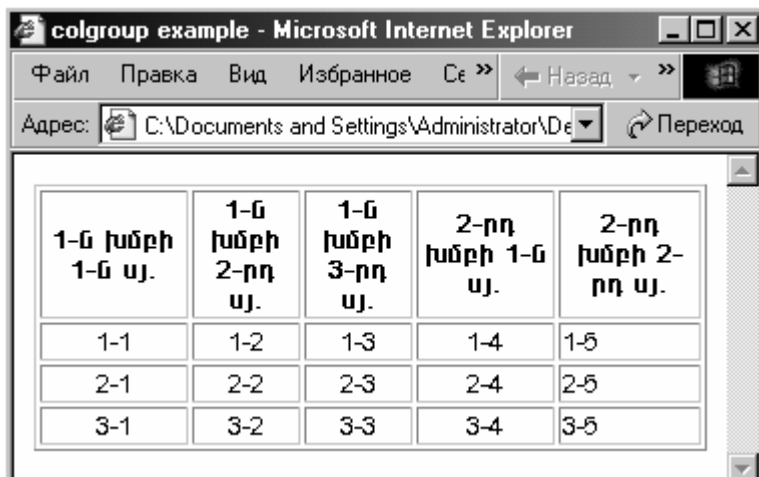
```
<col span="2" width="50" />
```

```
</colgroup>
```

```
<colgroup span="2">
```

```
<col align="center" /></colgroup>
```

`<col>` տեղն ունի նույն ատրիբուտները, ինչ որ `<colgroup>` տեղը: Միակ տարբերությունն այն է, որ `<col>`-ը որոշում է ոչ թե սյունակների խումբ, այլ միայն մեկ սյունակ:



Պատկեր 2.5.7. խմբերի ներքո տարբեր բնութագրեր ունեցող սյունակներով աղյուսակի օրինակ

Նշված տեգերի օգնությամբ կարելի ձևավորել ոչ միայն տվյալներ պարունակող աղյուսակները, այլ և այնպիսի աղյուսակներ, որոնք օգտագործվում են ամբողջ էջերի տրամաբանական բաժանման համար:

Աղյուսակը սահմանող `<table>` տեգն ունի երկու ատրիբուտներ, որոնք օգտագործվում են աղյուսակի արտաքին և ներքին շրջանակները արտապատկերելու եղանակները սահմանելու համար:

Աղյուսակի արտաքին շրջանակները գծանշելու եղանակները սահմանվում են **frame** ատրիբուտի միջոցով: Ըստ լրելյայն (իհարկե այն դեպքում, երբ **border** ատրիբուտին չնորհված է զրոից մեծ արժեք) գծանշվում են բոլոր չորս արտաքին շրջանակները: Այն կարող է ընդունել հետևյալ արժեքները՝

- void – շրջանակներ չկան,
- above - միայն վերին շրջանակը,
- below - միայն ստորին շրջանակը,
- hsidеs - միայն վերին և ստորին շրջանակները,
- vsides – միայն աջ և ձախ շրջանակները,
- lhs - միայն ձախ շրջանակը,
- rhs - միայն աջ շրջանակը,
- box - բոլոր չորս շրջանակները:

Աղյուսակի ներքին շրջանակները գծանշելու եղանակները տրվում են **rules** ատրիբուտի միջոցով: Այն կարող է ընդունել հետևյալ արժեքները՝

- none – ներքին գծերը բացակայում են,
- groups - գծերով առանձնացվում են միայն տողերի (<thead>, <tfoot>, <tbody> տեգերով սահմանված) կամ սյունակների (<colgroup> և <col> տեգերով սահմանված) խմբերը,
- rows - առանձնացվում են միայն տողերը,
- cols - առանձնացվում են միայն սյունակները,
- all - բոլոր գծերը կան:

Որպես օրինակ, ձևափոխենք պատկեր 2.5.7-ում ցուցադրված աղյուսակը այնպես, որ դրանում գծանշվեն միայն վերին և ստորին արտաքին շրջանակները (frames="hsides") և խմբերը բաժանող գծերը (rules="groups"): Ստորև բերված է համապատասխան ծրագրային կոդի ֆրագմենտը (տես՝ պատկեր 2.5.8):

1-ն խմբի 1-ն սյ.	1-ն խմբի 2-րդ սյ.	1-ն խմբի 3-րդ սյ.	2-րդ խմբի 1-ն սյ.	2-րդ խմբի 2-րդ սյ.
1-1	1-2	1-3	1-4	1-5
2-1	2-2	2-3	2-4	2-5
3-1	3-2	3-3	3-4	3-5

Պատկեր 2.5.8. Աղյուսակի արտաքին և ներքին շրջանակների գծանշման frame և rules ատրիբուտների կիրառության օրինակ

```
<table border="1" frame="hsides" rules="groups">
<colgroup span="3" align="center">
<col width="70" />
<col span="2" width="50" />
</colgroup>
<colgroup span="2">
```

```

<col align="center" />
</colgroup>
<thead><tr>
<th>1-ն խմբի 1-ն սյ.</th><th>1-ն խմբի 2-րդ սյ.</th><th>1-ն խմբի
3-րդ սյ.</th><th>2-րդ խմբի 1-ն սյ.</th><th>2-րդ խմբի 2-րդ սյ.</th>
</tr>
</thead>
<tbody>
<tr><td>1-1</td><td>1-2</td><td>1-3</td><td>1-4</td><td>1-
5</td></tr>
<tr><td>2-1</td><td>2-2</td><td>2-3</td><td>2-4</td><td>2-
5</td></tr>
<tr><td>3-1</td><td>3-2</td><td>3-3</td><td>3-4</td><td>3-
5</td></tr>
</tbody>
</table>:

```

§2.6. Ոճերի աղյուսակներ, հատուկ սիմվոլներ

2.6.1. Ոճերի աղյուսակների ընդհանուր հասկացությունը

Ինչպես վերը նշվեց HTML-ը փաստաթղթերի գծանշման ծրագրային անկախ լեզու է: HTML-ով գրված փաստաթղթերը պետք է առանց աղավաղվելու արտապատկերվեն բոլոր տեսակի բրաուզերներում՝ ինչպես լիարժեք գրաֆիկական, այնպես էլ, օրինակ, բջջային հեռախոսներում ներկառուցված: Այդ ամենը հնարավոր է դառնում web-էջերի ստեղծման երկու տարբեր կոնցեպցիաների՝ բովանդակության և ձևի (վիզուալ ձևավորման) տարազատման շնորհիվ:

Ինտերնետի բուռն զարգացման տարիներին, երբ այն դարձավ ինֆորմացիայի փոխանակման առևտրային ցանց՝ դիզայներները սկսեցին հեղեղել Ինտերնետի ոլորտը դիզայներական մշակումներով, դառնալով web-դիզայներներ: Իսկ դա նշանակում էր, որ բրաուզերների խոշոր արտադրողները պարտավոր էին զինել դրանց համապատասխան գործիքներով: Այդպես առաջացան նոր էլեմենտներ (տեգեր), որոնք ապահովեցին web-էջերի ավելի ճկուն ձևավորման հնարավորությունները: Սակայն որոշ էլեմենտների ստեղծումը բերեց նաև բացարձակապես արտաքին տեսքին կողմնորոշված և մինիմալ ինֆորմացիա պարունակող էջերի առաջացմանը: Այդպիսի էլեմենտներից էր, օրինակ, **** տեգը, որը առհասարակ չեն ճանաչում ոչ գրաֆիկական բրաուզերները և ճանաչում են ոչ բոլոր գրաֆիկականները: Օրինակ, եթե փորձենք

վերնագիր ձևակերպելու նպատակով տառերը ֆորմատավորել հետևյալ կերպով՝

Բարի գալուստ,

ապա տեզը չճանաչող բրաուզերները ոչ միայն այն անտեսում են, այլ և առհասարակ ոչ մի կերպ չեն առանձնացնում տեզում գրանցված տեքստը: Իսկ եթե այդ նույն տեքստը գրանցվի մեզ ծանոթ վերնագրային՝ <hi> տեգերում, ապա նույնիսկ ոչ գրաֆիկական բրաուզերները կաշխատեն այն որևէ կերպով առանձնացնել որպես վերնագիր:

Այդպիսի պրոբլեմների լուծման նպատակով ստեղծվեցին **ոճերի աղյուսակները** և, արդյունքում գրաֆիկական բրաուզերների տերերը կարողացան տեսնել գեղեցիկ տառաշարեր (շրիֆտեր), իսկ ոչ գրաֆիկականներինը՝ տեքստի որոշակի, համեմայն դեպս իմաստային (տրամաբանական), առանձնացում:

Ոճերի աղյուսակների կիրառության հիմնական սկզբունքը հետևյալն է՝ web-էջերի ստեղծման աշխատանքների ինֆորմատիվ և ձևավորման մասերի զատում: Դրանց կիրառությունը չափազանց օգտակար է և կարևոր:

Նախ, ոճերի աղյուսակները կարող են գոյություն ունենալ անկախ HTML փաստաթղթերից: **style** ատրիբուտը, համաձայն XML ստանդարտի, մնալու է միակ ատրիբուտը տեգերի ճնշող մեծամասնության ոճերի ձևավորման համար, սակայն այն չի հանդիսանալու այդ նպատակին հասնելու միակ միջոցը: Հնարավոր է ստեղծել մի ամբողջ կայք, չնշելով և ոչ մի էջում որևէ տեգի ոճը սահմանող ոչ մի ատրիբուտ: Բոլոր տեգերը կարտապատկերեն ինֆորմացիան ոճերի աղյուսակներին համապատասխանող գույնով, շրիֆտով, ֆոնով և այլն: Հնարավոր է սահմանել նույնիսկ ինֆորմացիայի երաժշտական համաչափության և ձայնային այլ բնութագրիչների արժեքները: Այսպիսով, ոճերի աղյուսակները թույլ են տալիս բրաուզերներին տալ այնպիսի ցուցումներ, որոնց միջոցով կղեկավարվի ամբողջ կայքը:

Ոճային էլեմենտների առանձնացումը կազմակերպականներից ունի ևս մեկ առավելություն՝ HTML կոդի առավել դյուրին ընկալում: Պատկերացնենք պարբերությունում գրանցված տեքստի ոճի այսպիսի սահմանում՝

<p>Շնորհակալ ենք մեր կայքը այցելելու համար

</p>:

Բերված կոդում իհարկե այնքան էլ դժվար չէ տարբերել ինֆորմացիոն բովանդակությունը (էջում արտապատկերվող

տեքստը) ոճային սահմանումից՝ տառաշարի պահանջվող “ընտանիքը” (font face), և տառերի չափսերը (10 pt) և գույնը (կանաչ): Սակայն, երբ փաստաթղթում այդպիսի պարբերությունների քանակը մեծաթիվ է՝ կողը կազմելու ընթացքում կամ խմբագրելու անհրաժեշտության դեպքում մեծ ուշադրություն և ջանքեր կպահանջվեն սխալներ թույլ չտալու համար: Իսկ եթե ոճերի աղյուսակում (քիչ ուշ մենք կքննարկենք, թե ինչպես է դա արվում) գրանցենք՝

<style>

p {font-family:Arial Armenian;font-size:10pt;color:green}

</style>,

ապա այն դեպքում, երբ էջի բոլոր պարբերություններում պահանջվում է տեքստի հենց այդպիսի ոճավորում, բավական է գրանցել միայն պարբերությունների <p> տեգերը և դրանց տեքստային բովանդակությունը: Օրինակ շնորհակալական տեքստի արտապատկերման կողը կընդունի հետևյալ տեսքը՝

<p>Շնորհակալ ենք մեր կայքը այցելելու համար</p>

Սակայն վերջապես ի՞նչ են իրենցից ներկայացնում ոճերի աղյուսակները: Այն ամենը, ինչ արդեն մենք իմացանք HTML-ի մասին բավական է, որպեսզի հասկանանք, որ տեքստի գծանշման լեզվի էլեմենտների նկարագրությունը չափազանց աշխատատար և մեծ ուշադրություն պահանջող գործընթաց է: Ենթադրենք մշակվող web-էջը պարունակում է տեքստի տասը մեծ պարբերություններ, որոնք անհրաժեշտ է տեղաբաշխել բրաուզերի պատուհանի ամբողջ լայնքով և քսան պատկերներ, որոնք պետք է հավասարեցնել էկրանի կենտրոնով: Դա նշանակում է, որ մենք քսան անգամ պետք է փաստաթղթում գրանցենք <p align="justify"></p> տեգերը և քսան անգամ՝ <center></center> կամ տեգերը: Իսկ եթե անհրաժեշտ է ներկայացնել բարդ աղյուսա՞կ, որն ունի մի քանի տասնյակ վանդակներ: Պատկերացնում էք քանի՞ անգամ է պետք գրանցել, օրինակ, height (բարձրություն) ատրիբուտը, որպեսզի արտապատկերվեն հավասար բարձրություն ունեցող վանդակներ բոլոր բրաուզերների էկրաններին: Էլ չասենք, որ ոճավորումը չի սահմանափակվում միայն տեքստն ու նկարները հավասարեցնելով և աղյուսակի վանդակների բարձրությունը որոշելով: Իսկ եթե այդպիսի էջերի քանակը կայքում նույապես մի քանի տասնյակ կամ, նույնիսկ, հարյուր է: Տեգերի քանակի ավելացման հետ առաջանում է ևս մեկ պրոբլեմ՝ աճում է արդյունքային ֆայլի ծավալը և, որպես արդյունք, նվազում է դրա բեռնման ժամանակը:

Ոճերի աղյուսակներում օգտագործվում է web-էջերի էլեմենտների ատրիբուտների նկարագրման մի փոքր այլ ալգորիթմ: Մեկ անգամ գրանցելով յուրաքանչյուր տեգի ոճի հատկությունները “**.css**” ընդլայնումով (օրինակ՝ **style.css**) տեքստային ֆայլում և որևէ եղանակով միավորելով այն HTML փաստաթղթերին (ընդ որում անսահմանափակ քանակով ֆայլերին), կարելի է ստիպել բրաուզերին կարողալ բոլոր էլեմենտների հատկությունները արդեն այդ ֆայլից: Գոյություն ունի ոճերի աղյուսակների ևս մեկ անվիժելի առավելություն՝ այն դեպքերում, երբ անհրաժեշտ է կայքի բոլոր էջերում փոխել որևէ էլեմենտի հատկությունները բավական է փոփոխել այդ էլեմենտի ձևավորման ոճը ընդամենը մեկ փաստաթղթի մեկ տողում:

Հատկությունների և դրանց արժեքների գրանցման քերականությունը կարելի է ներկայացնել հետևյալ ընդհանուր տեսքով՝ **հատկություն:արժեք** - երբ հատկությունը մեկն է և՝ **հատկություն1:արժեք1;...;հատկությունN: արժեքN** - երբ հատկությունները մի քանիսն են (հատկությունները ցուցակում առանձնացվում են կետ ստորակետերով):

Այն դեպքերում, երբ անկախ ընդհանուր ոճավորումից որևէ մեկ կամ երկու տեգեր անհրաժեշտ է ներկայացնել այլ ոճով, կարելի այդ տեգերում օգտագործել **style =”հատկություն:արժեք”** ատրիբուտը: Օրինակ, երբ անհրաժեշտ է, որպեսզի որոշակի պարբերությունում տեքստը գրանցվի կանաչ գույնի 10 պիքսել մեծության հայկական տառաշարով, ապա այդ պարբերությունը գծանշող <p> տեգը կգրանցվի հետևյալ տեսքով՝

<p style=“font-family:Arial Armenian,Arial LatArm,Times Armenian;font-size:10px;color:green”>Տեքստ</p>

Ինչպես տեսնում ենք բերված օրինակում **font-family**՝ տառաշարի տեսակը բնութագրող հատկությամբ շնորհիված են ստորակետերով բաժանված մի քանի արժեքներ: Այդպիսի դեպքերում բրաուզերը կլինետական համակարգչում փնտրում է ցուցակում գրանցված առաջին տառաշարը, եթե այն բացակայում է՝ երկրորդը և այլն, մինչև գտնվի թվարկված տառաշարերից որևէ մեկը:

➤ Հարկավոր է font-family հատկությունը օգտագործելիս լինել շափազանց ուշադիր: Կարող է պատահել, որ ցուցակում նշված և ոչ մի տառաշար առկա չլինի օգտվողի համակարգչում: Այդպիսի դեպքերից խուսափելու համար կարելի է պահանջվող տառաշարի ֆայլը տեղադրել սերվերի վրա TTF (True Type Font) ֆորմատում և որպես հատկության արժեք նշել դրա URL հասցեն: Օրինակ՝ **<p style=“font-family:url(ARIALAM.ttf)”>**:

2.6.2. Ոճերի աղյուսակների ներդրումը web-էջում

Ոճերի աղյուսակների օգտագործման եղանակներից մեկը՝ դրանց անմիջական ներդրումն է web-էջում, ընդ որում, կախված նպատակներից, դա կարելի է իրագործել երկու հնարավոր տարբերակներով:

Առաջինը, ինչպես վերը նշվեց՝ ոճի շնորհիվն է անմիջապես տեղին style ատրիբուտի միջոցով: Ոճավորման այդպիսի եղանակը կիրառվում է միայն այն դեպքերում, երբ անհրաժեշտ է փոփոխել էջի կոնկրետ մասում տեղաբաժխված մեկ-երկու տեղերի հատկությունները:

Երբ անհրաժեշտ է ստեղծել ոճի ավելի լուրջ միանմանություն, ապա կարող ենք օգտագործել <style></style> տեգը: Հենց այդ տեղի միջոցով է web-էջում ներդրվում ոճերի աղյուսակը: Տեգը տեղադրվում է փաստաթղթի <head></head> սեկցիայում: Գրանցման ֆորմատը հետևյալն է՝

```
<style type="text/css">
```

```
էլեմենտ1 {հատկություն:արժեք;...;հատկություն:արժեք}
```

```
էլեմենտ2 {հատկություն:արժեք;...;հատկություն:արժեք}
```

```
...
```

```
էլեմենտN {հատկություն:արժեք;...;հատկություն:արժեք}
```

```
</style>:
```

Աղյուսակում թվարկված էլեմենտները CSS-ի թերմիններով կոչվում են **սելեկտորներ**, այն ամենը ինչ գրանցված է ձևավոր փակագծերում **սահմանումներ**: Սելեկտորը իրեն սահմանումով կազմում է **կանոն**: <style> տեգում կարող է գրանցվել մի շարք կանոններ (դրանց գրանցումը նման է աղյուսակային գրանցմանը և հենց այդտեղից էլ ստացվել է անվանումը՝ ոճերի աղյուսակներ): Յուրաքանչյուր սահմանում կարող է ընդգրկել մի քանի կանոններ, որոնք այդ դեպքում առանձնացվում են կետ ստորակետերով:

Այժմ եթե մենք ցանկանում ենք, որ բոլոր պարբերություններում պարունակվող տեքստը գրանցվի թավ (bold), շեղ (italic), 12 պիքսել չափսի կարմիր գույնի (#ff0000) հայկական տառաշարով, ապա կարող ենք գրանցել՝

```
<style type="text/css">
```

```
p {font-family:Arial Armenian;font-size:12px;font_weight:bold;
```

```
font-style:italic;color:#ff0000}
```

```
</style>:
```

Միևնույն սահմանումը կարելի է վերագրել միաժամանակ մի քանի սելեկտորների, որոնք այդ դեպքում իրարից բաժանվում են ստորակետերով: Օրինակ՝

```
<style type="text/css">
```

p,h1,ul,ol {font-family:Arial Armenian}

</style>:

Համաձայն բերված կանոնի էջի բոլոր պարբերություններում, առաջին կարգի վերնագրերում, համարակալված և պիտակավորված ցուցակներում տեքստը կգրանցվի հայկական “Arial Armenian” ընտանիքի տառաշարով:

Ընդհանրապես ոճերի աղյուսակները կոչվում են Cascading Style Sheets` Կասկադային Ոճերի Թերթեր (աղյուսակներ): Կասկադային, քանի որ դրանց հատուկ է ժառանգելու հատկությունը: Եթե, օրինակ <table> տեղին շնորհիված է որևէ ոճ, ապա դրանք մտնող բոլոր էլեմենտները (տողերը և վանդակները) կժառանգեն այդ նույն ոճը:

2.6.3. Ոճերի դասեր, ոճավորման հատուկ տեղեր` <div> և

Ոճերի աղյուսակները հնարավորություն են ընձեռնում ոչ միայն սահմանել առանձին էլեմենտների հատկությունները, այլ և ստեղծել ամբողջ **դասեր**, որոնք թույլ են տալիս տարբերակել էլեմենտների բնութագրերը: Օրինակ, եթե անհրաժեշտ է առաջին կարգի վերնագրերը որոշ տեղերում գրանցել կարմիր գույնի տառերով, ապա կարող ենք ստեղծել դաս`

<style type="text/css">

h1.krivoy {font-style:italic;color:red}

</style>:

Տվյալ դեպքում h1 սելեկտորի փոխարեն օգտագործված է h1.krivoy սելեկտորը: Աղյուսակում գրանցված ոճը կկիրառվի h1-ին միայն այն դեպքերում, երբ տեղում մշվի class ատրիբուտը, որին որպես արժեք շնորհվում է համապատասխան դասի անունը: Օրինակ, եթե h1 տեղը գրանցվի հետևյալ տեսքով` <h1>Բառև Ձեզ</h1>, ապա տեղում գրանցված տեքստը դուրս կբերվի էկրանին սովորական տառաշարով, սև գույնի: Իսկ գրանվման այս տարբերակում <h1 class="krivoy">Բարև Ձեզ</h1>, տեքստը կարտապատկերվի կարմիր գույնի իտալիկ (շեղ) տառաշարով:

Այդ օրինակը հասկանալու համար կազմենք համապատասխան ծրագրային կոդը, որի արտապատկերումը բրաուզերում բերված է պատկեր 2.6.1-ում:

```
<html><head><title>CSS class example</title>
```

```
<style type="text/css">
```

```
h1.krivoy {font-style:italic;color:red}
```

```
</style >
```

```
</head>
```

```
<body>
```

```
<h1>Բարև Ձեզ</h1>
```



```
<h1 class="krivoy">Բարև Ձեզ</h1>
</body></html>
```

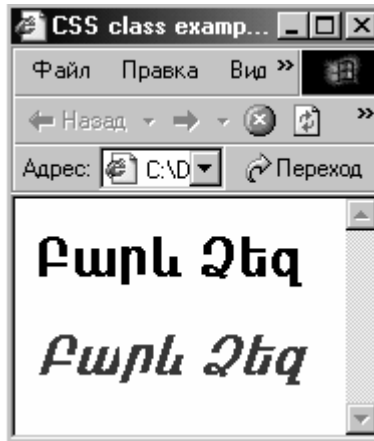
Կարելի է մեկ էլեմենտի համար ստեղծել ցանկացած քանակության դասեր և, փոխելով class ատրիբուտի արժեքը փաստաթղթի մարմնում, փոփոխել այդ էլեմենտի տեսքը անհրաժեշտ ոճին համապատասխան:

Ոճերի աղյուսակների օգտագործումը թույլ է տալիս նաև ստեղծել, այսպես կոչված **ունիվերսալ դասեր**, որոնք կարելի է կապակցել կամայական տեգի հետ: Ունիվերսալ դասի սելեկտորը (անունը) գրանցվում է կետից հետո: Օրինակ եթե ստեղծում ենք դաս, որն ուզում ենք անվանել "visshiy", ապա դրա դասի նկարագրությունը կլինի հետևյալը՝

```
<style type="text/css">
```

```
.visshiy {font-family:Arial Armenian;font-size:14px}
```

```
</style >:
```



Պատկեր 2.6.1. Դասերի կիրառության օրինակ

Ունիվերսալ դասի միջոցով կարելի է ստեղծված ոճը շնորհիվ էջի կամայական էլեմենտին, օրինակ՝

```
<p class=" visshiy">...</p>
```

```
<ul class=" visshiy">...</ul>
```

 և այլն:

Դասի գաղափարի օգտագործումը թույլ տվեց ստեղծել web-էջերին ինտերակտիվ տեսք տալու շատ տարածված միջոց: Այն հիմնված է, այսպես կոչված, **պսևդոդասերի** կիրառության վրա, ինչը թույլ է տալիս փոփոխել հղումների և մանատիպ օբյեկտների (հիմնականում ֆորմաների էլեմենտների) ոճը, կախված որոշակի դինամիկ կարգավիճակից:

Պսևդոդասը ստեղծվում է հետևյալ եղանակով՝ տեգի դեսկրիպտորից (տեգի անունը սահմանող բանալիական բառից) հետո դրվում է զույգակետ (:) և գրանցվում է պսևդոդասի անունը: Աղյուսակ 2.6.1-ում բերված են հղումների և նման օբյեկտների (հիմնականում ֆորմաների էլեմենտների) համար նախատեսված պսևդոդասերը:

Աղյուսակ 2.6.1.

Պսևդոդասերի օրինակներ

Հատկությունը	Նշանակումը
:link	Դեռ շայցելված հղումի տեսքը:
:visited	Արդեն այցելված հղումի տեսքը:
:hover	Հղումի կամ օբյեկտի տեսքը այն պահին, երբ դրա վրա է գտնվում մարկերը (մկնիկի նշիչը) առանց որևէ ստեղծ սեղմելու:
:focus	Հղումի կամ օբյեկտի տեսքը, երբ այն առանձնացված է կամ ստեղծաշարային մուտքի ընթացքում:
:active	Հղումի կամ օբյեկտի տեսքը այն պահին, երբ դրա վրա է գտնվում մարկերը և միաժամանակ սեղմած վիճակում է մկնիկի ձախ (կամ Enter) ստեղծը:
:first-letter	Սահմանում է էլեմենտի առաջին տառի ոճը:
:first-line	Սահմանում է էլեմենտի առաջին տողի ոճը:

Այն դեպքերում, երբ անհրաժեշտ է, որ սահմանվող ոճը ավտոմատ կերպով վերագրվի համապատասխան դիմամիկ վիճակում գտնվող օբյեկտին, ապա կանոնը ձևակերպվում է հետևյալ տեսքով՝

սեւեկտոր:վիճակ {կանոններ}:

Օրինակ, եթե պսևդոդասը ստեղծենք այսպիսի եղանակով՝

```
<style type="text/css">
```

```
a:hover {color:blue;background:yellow}
```

```
</style >
```

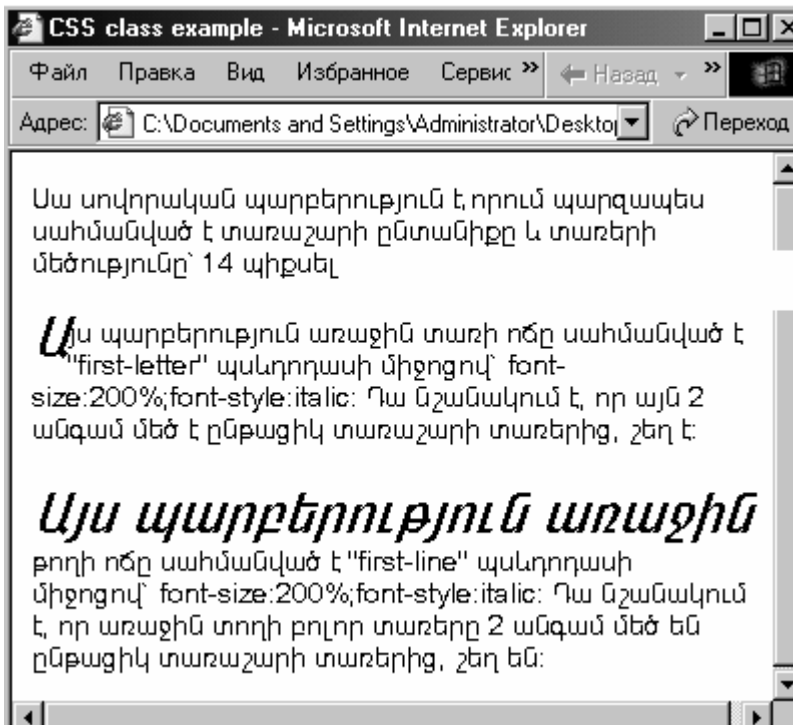
ապա մկնիկի նշիչը web-էջի կամայական հղումի վրա պահելիս (առանց որևէ ստեղծ սեղմելու) տառերը կդառնան կապույտ, իսկ ֆոնը՝ դեղին:

Եթե չի պահանջվում, որպեսզի տեղը, որի համար ստեղծվում է պսևդոդասը, փոփոխի ոճը բոլոր դեպքերում, ապա կիրառվում է պսևդոդասի ստեղծման միջնորդված եղանակը՝ ենթադասի միջոցով՝ **սեւեկտոր.ենթադաս:վիճակ {ոճ}:** Օրինակ՝

```
<style type="text/css">a.taphtaza:hover {color:blue;background:yellow}</style >
```

Այս դեպքում ոճը կվերագրվի փաստաթղթի միայն այնպիսի <a> տեգերին, որոնց համար նշված է class="taphtaza" ատրիբուտը: Աղյուսակ 2.6.1-ում բերված վերջին երկու պսևողդասերը օգտագործվում են հիմնականում տեքստը ձևավորելու նպատակով:

Բերենք պսևողդասերի կիրառության օրինակ (պատկեր 2.6.2):



Պատկեր 2.6.2. first-letter և first-line պսևողդասերի կիրառության օրինակ

Պատկեր 2.6.2-ում արտապատկերման ծրագրային կոդը հետևյալն է՝

```
<html><head><title>CSS class example</title>
<style type="text/css">
p {font-family:"Arial Armenian";font-size:14px}
p.letter:first-letter {font-size:200%;font-style:italic;float:left}
p.line:first-line {font-size:200%;font-style:italic;float:left}
</style>
```

```

</head>
<body>
<p>Սա սովորական պարբերություն է որում պարզապես սահման-
ված է տառաչարի ընտանիքը և տառերի մեծությունը՝ 14 պիքսել:
</p>
<p class="letter">Այս պարբերությունն առաջին տառի ոճը սահման-
ված է "first-letter" պսևդոդասի միջոցով՝ font-size:200%;font-style:
italic: Դա նշանակում է, որ այն 2 անգամ մեծ է ընթացիկ տառա-
չարի տառերից, շեղ է:
</p>
<p class="line">Այս պարբերությունն առաջին թողի ոճը սահմանված
է "first-line" պսևդոդասի միջոցով՝ font-size:200%;font-style:italic: Դա
նշանակում է, որ առաջին տողի բոլոր տառերը 2 անգամ մեծ են
ընթացիկ տառաչարի տառերից, շեղ են:
</p>
</body></html>

```

Ոճերի աղյուսակների հետ աշխատելիս շատ հաճախ օգտագործվում են երկու հատուկ կոնտեյներներ՝ **...** և **<div>...</div>**: span (հայերեն կարելի է թարգմանել՝ մատնաչափ) և div (division՝ բաժանում) էլեմենտների միջոցով ոճերի աղյուսակները կարելի է կիրառել փաստաթղթի կամայական մասին: Տարբերությունը կայանաում է նրանում, որ span կոնտեյները ներտողային է, այսինքն դրա մեջ չի կարելի տեղադրել պատկերներ, մուլտիմեդիա և առհասարակ էջի մեծածավալ բաղադրյալներ:

div-ը օգտագործվում է web-էջը սեկցիաների բաժանելու համար և, քանի որ կոնտեյներ է, ապա դրան կիրառելի է կամայական ֆորմատավորում: Այն կարելի է դիտարկել որպես սպառողական էլեմենտ և համադրել <body> և <head> տեգերի հետ: Տեգը պատկանում է բլոկային էլեմենտների դասին: Դա նշանակում է, որ բրաուզերը թողնում է նրա շուրջը ազատ տարածք (օրինակ՝ ինչպես <p> կամ <blockquote> տեգերի շուրջ):

2.6.4. Ոճերի աղյուսակների կապակցումը web-էջերին

Ինչպես վերը նշվեց <style> կոնտեյների միջոցով ոճերի աղյուսակները ներդրվում են HTML փաստաթղթում (<head> սեկցիայում) և աղյուսակի ազդեցությունը տարածվում է միայն այն փաստաթղթի վրա, որում այն հերդրված է: Սակայն բազմաթիվ էջերից բաղկացած կայքեր մշակելիս այդպիսի մոտեցումը, առնվազն անարդյունավետ է (մանավանդ երբ կայքը ցանկալի է ձևավորել միանման ոճով), քանի որ յուրաքանչյուր էջում <style> կոնտեյներում գրանցել միևնույն աղյուսակը բավականին աշխատատար է և ոչ խելամիտ: Ավելի հարմար կլինի կազմել մեկ՝ ունիվերսալ

աղյուսակ և այն կապակցել բոլոր անհրաժեշտ էջերի հետ: Կապակցումը ունի երկու ակնհայտ առավելություններ:

Առաջին պետք չէ յուրաքանչյուր էջի սկզբում գրանցել `<style>` տեգը:

Երկրորդ՝ եթե ամբողջ կայքի ոճերի կանոնները պահպանվում են մեկ աղյուսակում, ապա զգալիորեն պարզեցվում է ոճերի խմբագրման գործընթացը: Բացի այդ, եթե կայքի ձևավորումը իրականացնում են մի քանի դիզայներներ, ապա դյուրին կլինի նաև նրանց համագործակցությունը:

Web-էջը ոճերի աղյուսակի հետ կապակցելու համար նախ պետք է ստեղծել առանձին փաստաթուղթ, որում կնկարագրվեն միայն ոճերը: Չի կարելի գրանցել ոչինչ, ինչ կարող է արտապատկերվել բրաուզերում: Այդ փաստաթղթում կարելի է օգտագործել մեկնաբանություններ գրանցելու հատուկ տեգեր՝ `/*...*/`: Ստեղծված ֆայլը պահպանվում է հետագա օգտագործման համար ինչպես վերը նշվել է, `.css` ընդլայնումով (օրինակ՝ `styles.css`): Բերենք այդպիսի փաստաթղթի փոքրիկ օրինակ՝

`/* Վերնագրային տեգերի ոճերի նկարագրությունը */`

`h1 {font-family:Arial Armenian;font-size:24pt;font-weight:800}`

`h2 {font-family:Arial Armenian;font-size:20pt;font-weight:bold}`

`/* Վերնագրային տեգերի ոճերի նկարագրության ավարտը */:`

Ոճերի աղյուսակները պարունակող ֆայլը (և առհասարակ կամայական ֆայլը) էջի հետ կապակցելու համար օգտագործվում է հատուկ `<link />` տեգը, որն ունի երկու հիմնական ատրիբուտ՝ `rel` (relation), որը ցույց է տալիս թե ինչ տեսակի ֆայլի է կապակցվում էջը և `href`, որի միջոցով թելադրվում է այդ ֆայլի URL հասցեն: Տեգը տեղադրվում է փաստաթղթի վերնագրային մասում: Օրինակ՝

`<head><title>Փաստաթղթի վերնագիրը</title>`

`<link rel="stylesheet" href="styles/styles.css" />`

`</head>`:

URL հասցեն հասցեն կարող է լինել հարաբերական կամ բացարձակ (դա քննարկվել է §1.3-ում): Ստեղծված ոճերի աղյուսակները պարունակող ֆայլը կարելի է նաև ներդնել յուրաքանչյուր էջում հատուկ՝ **import** դիրեկտիվի միջոցով: Այն գրանցվում է `<style>` տեգի ներքո և պարունակում `“css”` ֆայլի URL հասցեն: Օրինակ՝

`<style>`

@import url(ոճերի աղյուսակի ֆայլի URL հասցեն)

`</style>`

2.6.5. Բովանդակի էլեմենտների տեղաբաշխման և տեսքի ոճերը

HTML-ի այն էլեմենտները, որոնց շուրջը բրաուզերները ըստ լրելայն թողնում են դատարկ դաշտեր (տարածություն) կոչվում են բլոկային էլեմենտներ: Այդ անվանումը բացատրվում է նրանով, որ նրանք ստեղծում են տեքստի կամ այլ էլեմենտների բլոկներ և կարող են դիտարկվել որպես օբյեկտներ: Այդ ամենը թույլ է տալիս փոփոխել դրանց տեղաբաշխումը և արտաքին տեսքը, սահմանել դաշտեր, խորություն, ավելացնել կամ պակասեցնել սահմաններ, շրջանակներ և այլն: Ոճերի աղյուսակների միջոցով այդ հատկությունները կարելի է վերագրել բոլոր բլոկային մակարդակի էլեմենտներին: Աղյուսակ 2.6.2-ում թվարկված են այն հատկությունները, որոնց միջոցով կարելի է կարգավորել բլոկների արտաքին տեսքը:

Աղյուսակ 2.6.2.

Բլոկային էլեմենտների հատկությունները

Հատկությունը	Արժեքները	Օրինակներ
margin	Երկարություն կամ %	2px, 5%, 10%
padding	Երկարություն կամ %	2px, 5%, 10%
border	Լայնություն/ոճ/գույն	medium dashed red
width	Երկարություն կամ %	100px, 5in, 50%
height	Երկարություն կամ %	50px, 100%
clear	Ուղղություն	left, right, none
float	Ուղղություն	left, right, none, both

margin և **padding** հատկությունները սահմանում են համապատասխանաբար՝ արտաքին և ներքին լուսանցքները, այսինքն լրացուցիչ պորեթներ բլոկի սահմաններից դուրս և, սահմանների ու տեքստի միջև: Երկու հատկություններն էլ ունեն «կողմնորոշված» ենթատեսակներ: Օրինակ՝ «div {margin-left: 5px}», «table {margin-right: 3px}», «p {padding-top: 3px}», «table {padding-bottom: 10px}»: Ավելի մանրամասն այդ և, արհասարակ ոճերի աղյուսակներում օգտագործվող բոլոր հատկությունները և դրանց հնարավոր արժեքները նկարագրված են Հավելված 1-ում:

- Բլոկային էլեմենտների որոշ հատկությունները (հիմնականում՝ շրջանակների և լուսանցքների բնութագրման հետ կապված) ունեն, այսպես կոչված «կողմնորոշված» կամ «ուղղված» տարատեսակներ: Օրինակ՝ ներքին լուսանցքները (padding) կարելի է բնութագրել ինչպես ամբողջովին միաժամանակ չորս կողմինն էլ, այնպես էլ առանձին՝ վերին, ստորին, ձախակողմյան և աջակողմյան: Այդպիսի դեպքերում համապատասխան

հատկության անվանումը ոճերի աղյուսակում գրանցվում է բաղադրայալ տեսքով: Այն կառուցվում է հետևյալ սկզբունքով՝ հիմնական բնութագրվող պարամետրի անվանմանը գումարվում է նրա “ուղղությունը” և, եթե անհրաժեշտ է, համապատասխան բնութագրիչը, որին պետք է շնորհվի արժեք: Դրանք առանձնացվում են գծիկներով: Քերականությունը հետևյալն է՝ **հատկություն-ուղղություն:բնութագրիչ**: Օրինակ՝
էլեմենտի “ծախ ներքին լուսանցք” հատկությունը կգրանցվի՝
padding-left, “վերին արտաքին լուսանցք”՝ margin-top, border-left-color և այլն:

Շրջանակների հաստությունը սահմանվում է border հատկությունով: Արժեքները կարող են լինել ինչպես թվային, այնպես էլ անվանումներ՝ thin, medium, thick: Օրինակ՝ “table {border:thin}” կամ “table {border:1pt}”: Երբ անհրաժեշտ է սահմանել որևէ հատկություն էլեմենտի բոլոր շրջանակների համար, ապա border բառից հետո գրանցվում է գծիկ և համապատասխան հատկության անվանումը, օրինակ՝

border-color – շրջանակի գույնը (արժեքները շնորհվում են ինչպես և color հատկությանը)

border-style – սահմանում է շրջանակի գծագրման ոճը՝ ընդհատ, անընդհատ և այլն,

border-width – սահմանում է շրջանակի հաստությունը

Այդ հատկությունը նույնպես ունի կողմնորոշված տարատեսակներ՝ border-top, border-left, border-right, border-bottom, որոնց միջոցով կարելի է սահմանել համապատասխանաբար՝ վերին, ձախ, աջ և ստորին շրջանակների բնութագրիչների արժեքները: Օրինակ՝

border-left-color:#00ff00 – ձախ-շրջանակի-գույնը:կանաչ,

border-right-style:dash – աջ-շրջանակի-ոճը:ընդհատ,

border-bottom-width:3px – ստորին-շրջանակի- հաստությունը:10px:

էլեմենտների լայնությունը և բարձրությունը սահմանելու համար օգտագործվում են width և height հատկությունները, որոնք կարող են ընդունել թվային կամ տոկոսային արժեքներ: Օրինակ՝
div {width:300px} կամ table {width:50%;height:70%}:

float հատկությունը թույլ է տալիս ստեղծել “լողացող” բլոկներ, որոնք շրջափակվում են տեքստով կամ այլ էլեմենտներով: Այն հիշեցնում է img տեգի align ատրիբուտը: Տարբերությունը նրանում է, որ float հատկությունը կարելի է վերագրել կամայական բլոկային էլեմենտի: Արժեքներն են՝ none, left, right: Օրինակ, եթե գրանցվի “float:left”, ապա էլեմենտը կսեղմվի ձախ լուսանցքին, իսկ մյուս էլեմենտները կշրջանցեն դրան աջից:

Էլեմենտների **clear** հատկության միջոցով որոշվում է՝ կարո՞ղ են արդյոք մյուս էլեմենտները շրջափակել տվյալ էլեմենտը որևէ կողմից: Օրինակ, “clear:left” արժեքի դեպքում տվյալ բլոկը տեղաշարժվում է ձախով հավասարեցված “լողացող” էլեմենտների տակ և, միաժամանակ, արգելվում է դրա շրջանցումը աջից:

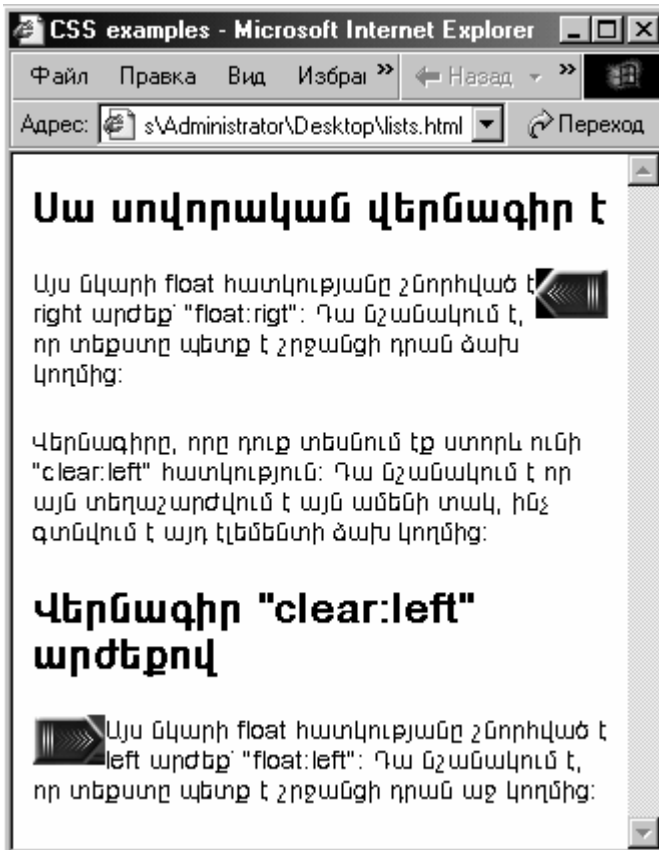
Վերջին երկու հատկությունների կիրառությունը ավելի լավ պատկերացնելու համար կազմենք հետևյալ HTML փաստաթուղթը (տես նաև պատկեր 2.6.3-ը):

```
<html>
<head>
<title>CSS examples
</title>
<style type="text/css">
img.right {float:right}
img.left {float:left}
h2.no_wrap {clear:left}
</style>
</head>
<body>
<h2 class="wrap">Սա սովորական վերնագիր է</h2>
<p>
Այս նկարի float հատկությանը շնորհիված է right արժեք՝ "float:right":
Դա նշանակում է, որ տեքստը պետք է շրջանցի դրան ձախ կողմից:
</p>
<p>Վերնագիրը, որը դուք տեսնում էք ստորև ունի "clear:left"
հատկություն: Դա նշանակում է որ այն տեղաշարժվում է այն
ամենի տակ, ինչ գտնվում է այդ էլեմենտի ձախ կողմից:
</p>
<h2 class="no_wrap"> Վերնագիր "clear:left" արժեքով</h2>
<p>
Այս նկարի float հատկությանը շնորհիված է left արժեք՝ "float:left":
Դա նշանակում է, որ տեքստը պետք է շրջանցի դրան աջ կողմից:
</p>
</body>
</html>
```

2.6.6. Հատուկ սիմվոլներ (պրիմիտիվներ)

Շատ դեպքերում անհրաժեշտություն է առաջանում բրաուզերի էկրանին արտապատկերել որոշ նշաններ, որոնք կամ բացակայում են ստեղծաչարի վրա, կամ կարող են վերծանվել բրաուզերի կողմից որպես գծանշում (տեգ), օրինակ <, >, @ և այլն: Այդ խնդիրը լուծելու նպատակով ստեղծվել են հատուկ նշաններ՝ **պրիմիտիվ-**

Ոճեր, որոնք լինում են երկու տեսակների՝ տառանիշային և թվանիշային:



Պատկեր 2.6.3. float և clear հատկությունների կիրառության օրինակ

Երկու տեսակի նշաններն էլ սկսվում են միևնույն՝ & (ամպերսանդ) նշանից և ավարտվում “;”, որը ազդարարում է պրիմիտիվի ավարտը: Տառանիշային պրիմիտիվներում այդ երկու նշանի միջև գրանցվում են տառեր, իսկ թվանիշայիններում &-ից հետո գրանցվում է “#” նշանը և, ապա թվային կոդը: Որոշ նշաններ կարելի է արտահայտել ինչպես տառանիշային, այնպես էլ թվանիշային պրիմիտիվներով: Օրինակ՝ ուղիղ չափերտների տառա-

նիշային պրիմիտիվը գրանցվում է հետևյալ տեսքով՝ “"”, իսկ թվանիշայինը՝ “"”:

HTML-ում օգտագործվող նշանների բազմությունը կոչվում է ISO-Latin-1: Այն հաստատված է Ստանդարտների Միջազգային Կազմակերպության (International Organization for Standardization՝ ISO) կողմից: Բոլոր հաստատված ստանդարտներն ունեն համարներ: ISO-Latin-1-ը հայտնի է նաև, որպես ISO 8859-1:

Ասվածը պարզաբանելու համար կազմենք մի օրինակ: Ենթադրենք անհրաժեշտ է արտապատկերել բրաուզերի էկրանին հետևյալ տեքստը՝

Տեքստը թավ տառաշարով արտապատկերելու համար օգտագործվում են **** բացող և **** փակող տեգերը:

Եթե փաստաթուղթը կազմելիս նախադասությունը գրանցենք վերը բերված տեսքով, ապա բրաուզերը **** և ****-ն կվերծանի որպես գծանշման տեգեր և բացող և պարզապես կարտապատկերի “բացող և” բառերը թավ տառաշարով: Սակայն մեր նպատակն այն է, որ տեքստը արտապատկերվի նույն տեսքով ինչ գրանցված է: Հենց դրա համար էլ օգտագործվում են պրիմիտիվները: Այժմ եթե նույն նախադասությունը HTML փաստաթղթում գրանցվի հետևյալ կերպով՝

Տեքստը թավ տառաշարով արտապատկերելու համար օգտագործվում են ****; բացող և ****; փակող տեգերը,

ապա արտապատկերումը կհամապատասխանի մեր ցանկացածին:

Օրինակը պարզաբանելու համար կազմենք հետևյալ HTML փաստաթուղթը (տես նաև պատկեր 2.6.4.):

```
<html>
```

```
<head><title>Primitives using example</title>
```

```
</head>
```

```
<body>
```

```
<p>Տեքստը թավ տառաշարով արտապատկերելու համար  
օգտագործվում են <b> բացող և </b> փակող տեգերը</p>
```

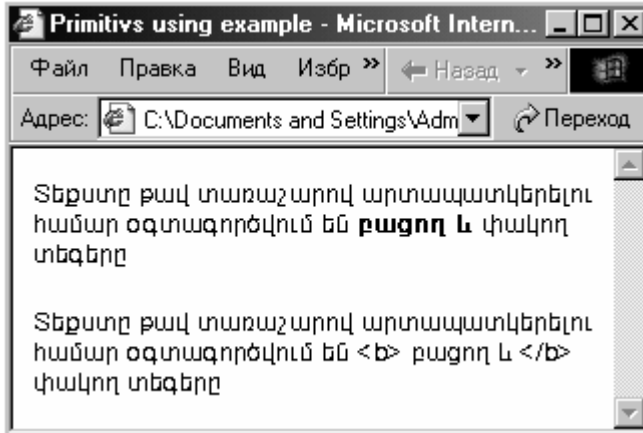
```
<p>Տեքստը թավ տառաշարով արտապատկերելու համար  
օգտագործվում են &lt;b>; բացող և &lt;/b>; փակող տեգերը</p>  
</body></html>
```

Որոշ առավել հաճախ օգտագործվող պրիմիտիվների ցուցակը բերված է հավելված 2-ում:

§2.7. Սայթերի կառուցումը ֆրեյմերի (շրջանակների) միջոցով

Երկար ժամանակ web-կայքերի հեղինակները ջանում էին կազմակերպել կայքի էջերով “երթևեկելու” միջոց, որը կատարված

լինել միատեսակ ոճով ամբողջ կայքի և, նույնիսկ ամբողջ պորտալի համար: Ստանդարտ գործիքներ օգտագործելիս անհրաժեշտություն էր առաջանում կրկնել հղումային մեկուկերը յուրաքանչյուր էջում: Սակայն HTML Frames սպեցիֆիկացիայի զարգացումը հնարավորություն ընձեռնեց բաժանել բրաուզերի պատուհանը մի քանի բաղադրյալ պատուհանների, արտապատկերելով դրանցում իրարից լիովին անկախ էջեր:



Պատկեր 2.6.4. Պրիմիտիվների կիրառության օրինակ

Ֆրեյմերի օգտագործման համար <body> տեգը փոխարինվում է <frameset>,</frameset> կոնտեյներով, որը կարող է պարունակել մի քանի առանձին <frame /> էլեմենտներ, որոնցից յուրաքանչյուրում արտապատկերվում են սեփական URL հասցե ունեցող փաստաթղթեր: Անհրաժեշտ է նաև նշել փաստաթղթի տեսակի սահմանումը: Համաձայն նոր ստանդարտի այն կլինի հետևյալը՝
 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset/EN"
 "http://www.w3.org/TR/xhtml1/DTD/frameset.dtd">
 <html xmlns="http://www.w3.org/1999/xhtml" >:

Ֆրեյմերի ներսում առանձին էջեր կարող են պարունակել հիպերհղումներ, ընդ որում նշվում է, թե որ ֆրեյմերում պետք է արտապատկերվեն այդ հասցեներով գտնվող փաստաթղթերը: Հիպերհղումներ պարունակող էջը կարելի է կազմակերպել այնպես, որ հղումների ցուցակը արտապատկերվի առանձին պատուհանում, իսկ հղումային էջերը միշտ արտապատկերվեն միևնույն նպատակային պատուհանում (ֆրեյմում), որը կարելի է անվանել

“դիտարկման գլխավոր պատուհան”։ Հաճախ ֆրեյմերն ունենում են սեփական պատաստեղն կամ մկնիկի միջոցով չափսերի փոփոխության հնարավորություն։

`<frameset>` տեգն ունի երկու հիմնական ատրիբուտներ՝ **cols** և **rows**, որոնք թույլ են տալիս բաժանել բրաուզերի պատուհանը ցանկացած քանակության տողերի և սյունակների (ֆրեյմերի)։ Օրինակ՝ `<frameset cols="30%,70%"></frameset>` գրանցումը նշանակում է, որ բրաուզերի պատուհանը բաժանվելու է երկու սյունակների (ֆրեյմերի), որոնցից ձախը պետք է զբաղեցնի պատուհանի լայնության 30%-ը, իսկ աջը՝ 70%-ը։ Իսկ եթե տեգը գրանցենք հետևյալ տեսքով՝ `<frameset rows="20%,50%,30%"></frameset>` ապա բաժանումը կկատարվի ըստ տողերի (3 տող), որոնք կզբաղեցնեն բրաուզերի պատուհանի համապատասխանաբար 20, 50 և 30 տոկոսը։

Ֆրեյմերի լայնությունը կամ բարձրությունը կարելի է սահմանել նաև պիքսելներով։ Օրինակ՝ `<frameset rows="10,100,300,200"></frameset>` գրանցումը նշանակում է, որ սահմանվում են չորս, համապատասխանաբար՝ 10, 100, 300 և 200 պիքսել բարձրություններ ունեցող հորիզոնական ֆրեյմեր (բաժանում ըստ տողերի)։

Ընդհանուր առմամբ հարմար է լինում նշել տոկոսներով կամ պիքսելներով միայն ծառայողական (օրինակ՝ նավիգացիայի) և (եթե կա) ձևավորման նպատակներով ստեղծվող ֆրեյմերի չափսերը, թողնելով մնացած տարածությունը “գլխավոր” պատուհանին։ Դա իրականացվում է հատուկ՝ “*” (աստղանիշ) արժեք շնորհելու միջոցով։ Օրինակ, եթե ֆրեյմերը սահմանվեն հետևյալ տեսքով՝ `<frameset cols="30%,*"></frameset>`, ապա դա կնշանակի, որ ձախ ուղղահայաց ֆրեյմը կզբաղեցնի պատուհանի լայնքի 30 տոկոսը, իսկ աջը՝ մնացած տարածությունը, անկախ էկրանի բացվածքից։

Սակայն `<frameset>` կոնտեյները ինքստինքյա որոշում է միայն բաժանման եղանակը և, որպեսզի հնարավոր լինի որևէ բան տեսնել էկրանին՝ անհրաժեշտ է նկարագրել յուրաքանչյուր առանձին ֆրեյմը։ Դա կատարվում է `<frame />` տեգի օգնությամբ, որի `src` ատրիբուտում նշվում է այն էջի հասցեն, որը պետք է արտապատկերվի տվյալ ֆրեյմում, ընդ որում բոլոր արտապատկերվող էջերը պետք է լինեն նորմալ, լիարժեք HTML փաստաթղթեր։ Հասցեն կարող է լինել ինչպես բացարձակ այնպես էլ հարաբերական (մանրամասն հասցեավորման հարցերը քննարկվել են §2.3-ում)։

`<frame />` տեգերի քանակը պետք է համապատասխանի `<frameset>` տեգում սահմանված ֆրեյմերի քանակին։ Օրինակ, եթե

կառուցվում է երկու սյունակից կազմված ֆրեյմերի խումբ, ապա անհրաժեշտ է գրանցել երկու <frame /> էլեմենտներ՝

```
<frameset cols="30%,*">
```

```
<frame src="հասցե" />
```

```
<frame src="հասցե" />
```

```
</frameset>:
```

Ասվածը պարզաբանելու համար կազմենք երեք փաստաթղթեր:

Առաջին փաստաթղթում՝ անվանենք այն "frames.html", սահմանենք ֆրեյմերի բազմությունը՝

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
<head>
```

```
<title>Frames</title>
```

```
</head>
```

```
<frameset cols="40%,*">
```

```
<frame src="left.html" />
```

```
<frame src="right.html" />
```

```
</frameset>
```

```
</html>
```

Սահմանվում են երկու սյունակներ (cols), որոնցից առաջինը (ծախը) կբաղեցնի բրաուզերի պատուհանի լայնքի 40 տոկոսը, իսկ մյուսը մնացած մասը: Չախ պատուհանում կարտապատկերվի "left.html" անունով փաստաթուղթը, իսկ աջում՝ "right.html"-ը: Քանի որ հասցեները հարաբերական են, ապա պարզ է, որ բոլոր երեք փաստաթղթերը պետք է գտնվեն միևնույն թղթապանակում: Ստորև բերված են "left.html" և "right.html" ֆայլերի կոդերը: Հիմնական ֆայլի արտապատկերումը ցուցադրված է պատկեր 2.7.1-ում:

left.html ֆայլի կոդը հետևյալն է՝

```
<html>
```

```
<head><title>Left frame</title>
```

```
</head>
```

```
<body>
```

Սա ծախ ֆրեյմում արտապատկերվող էջն է: Ջբաղեցնում է բրաուզերի պատուհանի 40%-ը: Դա նշված է <frame> տեգի cols ատրիբուտում <frame cols="40%,*">

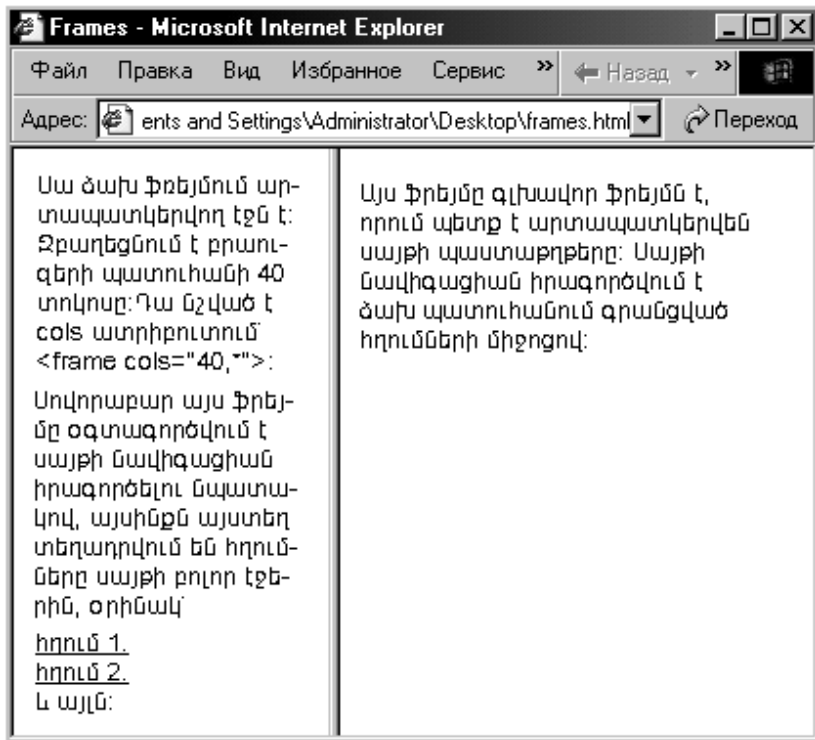
Սովորաբար այս ֆրեյմը օգտագործվում կայքի նավիգացիան իրագործելու նպատակով, այսինքն այստեղ լինում են հղումները կայքի բոլոր էջերին: Օրինակ


```
<u>հղում 1.</u><br />
```

```
<u>հղում 2.</u><br />
```

և այլն:

```
</body></html>
```



Պատկեր 2.7.1. Ֆրեյմային կառուցվածքի օրինակ

right.html ֆայլի կոդը հետևյալն է՝

```
<head><title>Right frame src</title>
```

```
</head>
```

```
<body>
```

```
<p>Այս ֆրեյմը գլխավոր ֆրեյմն է, որում պետք է արտապատկերվեն կայքի պաստաթղթերը: Սայթի նավիգացիան իրագործվում է ձախ պատուհանում գրանցված հղումների միջոցով:
```

```
</p>
```

```
</body></html>
```

Այն դեպքերում, երբ անհրաժեշտ է կազմակերպել ֆրեյմերի ավելի բարդ կառուցվածք սկզբից `<frameset>` տեգի միջոցով սահմանվում են անհրաժեշտ քանակության տողերը և, դրանից հետո, յուրաքանչյուր տողում նույնպես `<frameset>` տեգի միջոցով ներ-

դրվում են (եթե տողում պետք է տեղադրվի մեկից ավելի ֆրեյմեր) ուղղահայաց ֆրեյմերը տեղադրվող սյունակների քանակը: Օրինակ, անհրաժեշտ է էկրանի վերին մասում տեղադրել ֆրեյմ, որում կարտապատկերվի որևէ գովազդային պատկեր, իսկ ստորև՝ երկու ուղղահայաց ֆրեյմեր: Դա կարելի է անել հետևյալ կերպով՝

```
<frameset rows="100,*">
<frame src="banner.html" />
<frameset cols="50%,*">
<frame src="left.html" />
<frame src="right.html" />
</frameset></frameset>
```

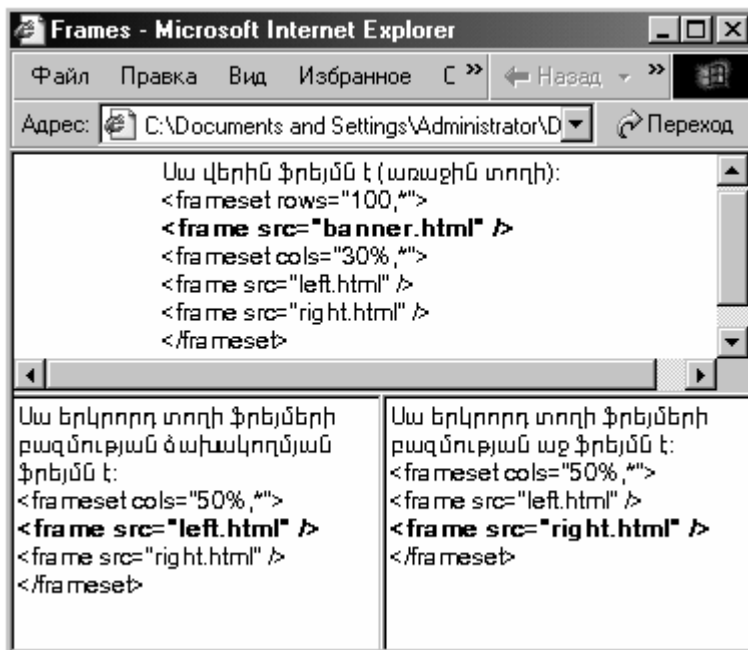
Համեմատ նախորդ օրինակին այս դեպքում ավելանում է ևս մեկ փաստաթուղթ (անվանեք այն “banner.html”): Դրա ծրագրային կոդը հետևյալն է՝

```
<html>
<head><title>Bunner</title>
</head>
<body style="font-size:9pt" topmargin="2" leftmargin="2"
rightmargin="0">
<table>
<tr><td width="20%">&nbsp;</td><td>
Սա վերին ֆրեյմն է (առաջին տողի):<br />
<frameset rows="100,*"><br />
<b><frame src="banner.html" /></b><br />
<frameset cols="30%,*"><br />
<frame src="left.html" /><br />
<frame src="right.html" /><br />
</frameset><br />
</frameset>
</td>
<td width="20%">&nbsp;</td>
</tr>
</table>
</body>
</html>
```

Ինչպես կարելի է նկատել այն տեքստը, որը արտապատկերվել է էկրանին (տես՝ պատկեր 2.7.2.) տեղադրված է աղյուսակի միջին վանդակում, իսկ առաջին և երրորդ վանդակների լայնությունները սահմանված են աղյուսակի լայնության 20 տոկոսի չափով: Դրա հաշվին տեքստը արտապատկերվում է մոտավորապես էկրանի միջին մասում (հորիզոնական):

Մյուս երկու փաստաթղթերը, որոնք պետք է արտապատկերվեն երկրորդ տողի ձախ (left.html) և աջ (right.html) սյունակներում առաջարկում ենք կազմել ինքնուրույն:

Նշենք, որ <frameset> տեգը ունի մի շարք ատրիբուտներ, որոնց միջոցով կարելի է կարգավորել շրջանակների հաստությունը (border), գույնը (bordercolor), տարածությունը ընդգրկվող ֆրեյմերի միջև (framespacing) և այլն: Նույն ատրիբուտները (բացի framespacing ատրիբուտից) բնորոշ են նաև առանձին ֆրեյմը սահմանող <frame /> տեգին:



Պատկեր 2.7.2. Ներդրված ֆրեյմերի բազմությունների օրինակ

Սովորաբար կայքերը բաղկացած են մեծ թվով բազմապիսի web-փաստաթղթերից, անցումը որոնց իրականացվում է հղումների միջոցով: Այն դեպքերում երբ բրաուզերի պատուհանը միակն է (սովորական փաստաթղթերի համար) նոր բեռնվող էջը սովորաբար բացվում է նույն պատուհանում նախորդի փոխարեն: Անհրաժեշտության դեպքում հնարավոր է բացել փաստաթղթերը առանձին պատուհանում, պահպանելով միաժամանակ հիմնական

նը: Դա կատարվում է <a> տեգի **target** ատրիբուտի օգնությամբ, շնորհիվ դրան `"_blank"` արժեքը (`target="_blank"`):

Սայթի ֆրեյմային կառուցվածքի դեպքում **target** ատրիբուտը ստանձնում է կարևորագույն դեր: Շնորհիվ առանձին ֆրեյմերին ունիկալ անուններ և գրանցելով դրանք հղումներում, որպես ատրիբուտի արժեք, կարող ենք բացել հղումային փաստաթղթերը պահանջվող ֆրեյմերում: Անունը շնորհվում է ֆրեյմին `name` ատրիբուտի միջոցով, օրինակ՝

```
<frame src="right.html" name="viewer" />
```

Հիշենք, որ `src` ատրիբուտում գրանցվում է այն փաստաթղթի հասցեն, որը պետք է արտապատկերվի ֆրեյմում առաջին անգամ: Պատկերացնենք այժմ, որ վերջին օրինակում (տես՝ պատկեր 2.7.2.) բերված ֆրեյմերի կառուցվածքում բոլոր նոր բացվող փաստաթղթերը մենք ցանկանում ենք արտապատկերել ստորին աջ ֆրեյմում: Նպատակը իրականացնելու համար նախ `frames.html` ֆայլում շնորհենք այդ ֆրեյմին անուն՝ օրինակ, `"viewer"`՝

```
<frameset rows="100,*">
```

```
<frame src="banner.html" />
```

```
<frameset cols="50%,*">
```

```
<frame src="left.html" />
```

```
<frame src="right.html" name="viewer" />
```

```
</frameset>
```

```
</frameset>
```

Այժմ ստեղծելով հղում այն փաստաթղթին, որը պետք է արտապատկերվի `"viewer"` անունով ֆրեյմում՝ բավական է <a> տեգում նշել որպես **target** ատրիբուտի արժեք `"viewer"` անունը: Ընդ որում կարևոր է, թե որ ֆրեյմում է արտապատկերվում հղումը պարունակող էջը (սովորաբար հղումների նավիգացիայի ցուցակը տեղադրվում է ձախ ֆրեյմում): Մեր դեպքում՝ եթե `left.html` փաստաթղթում ստեղծենք հղում որևէ այլ փաստաթղթին, օրինակ `"newpage.html"`, ապա այն պետք է գրանցվի հետևյալ տեսքով՝

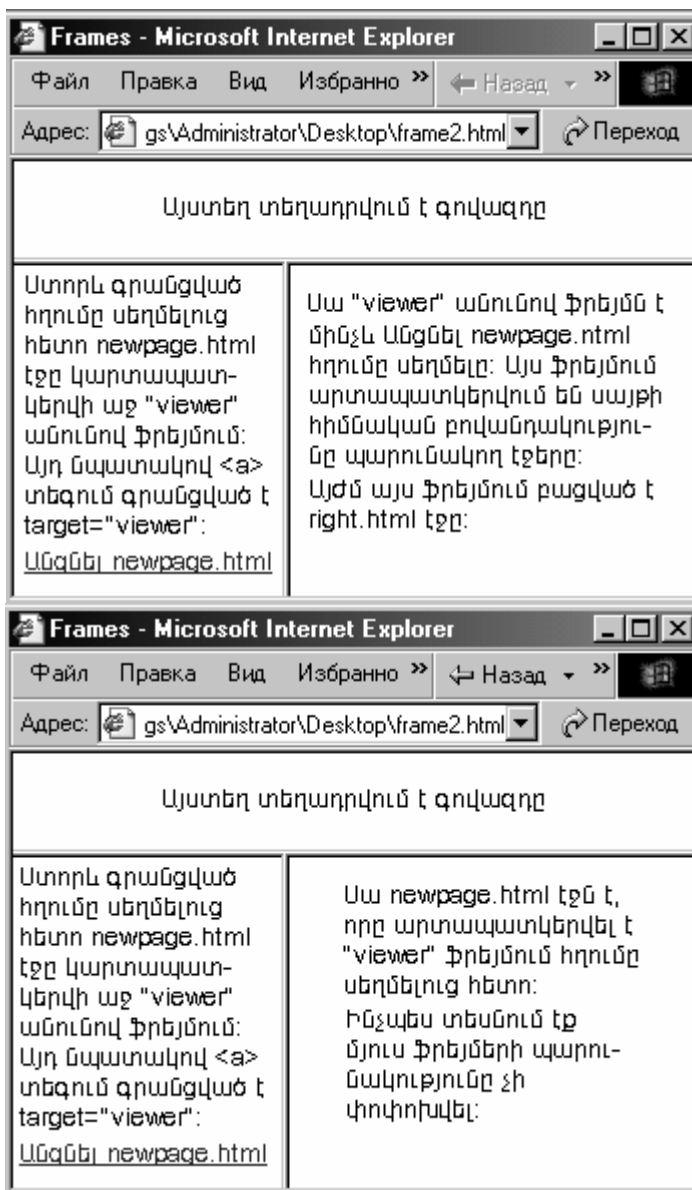
```
<a href="newpage.html" target="viewer">
```

```
Անցնել newpage.html
```

```
</a>:
```

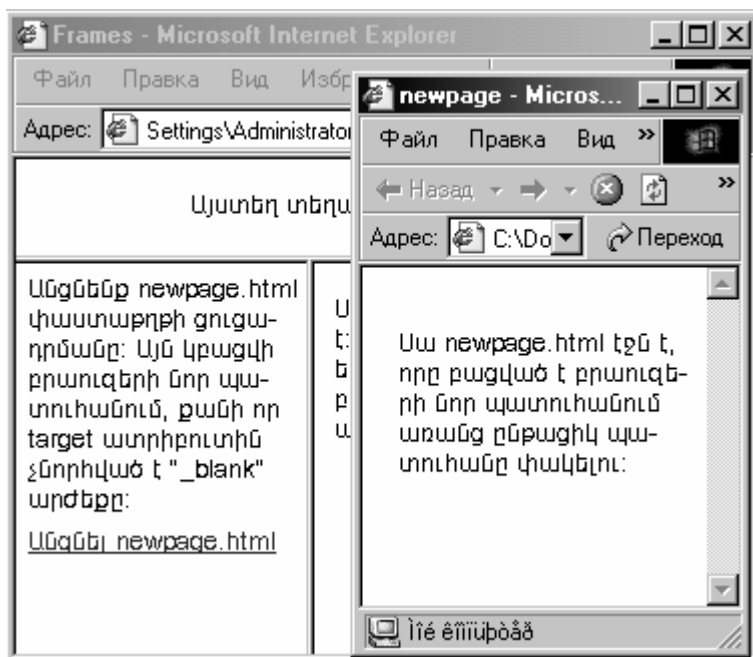
Հղումին սեղմելուց հետո բրաուզերի ստորին աջ ֆրեյմում (`viewer`-ում) կբացվի `newpage.html` էջը (տես՝ պատկեր 2.7.3): `left.html`, `right.html` և `newpage.html` ֆայլերի ստեղծումը թողնում ենք ինքնուրույն աշխատանքի համար:

Թվարկենք `"target"` ատրիբուտի արժեքները, որոնք սկսվում են ընդգծման նշանից (`"_"`) և թույլ են տալիս սահմանել նպատակային օբյեկտները, չդիմելով ֆրեյմերին դրանց անուններով՝



Պատկեր 2.7.3. target ատրիբուտի օգտագործման օրինակ

- **_self** – փաստաթուղթը արտապատկերվում է նույն պատուհանում, որում տեղադրված է հղումը:
 - **_parent** - փաստաթուղթը արտապատկերվում է այն պատուհանում, որը հանդիսանում է ծնողական տվյալ պատուհանի նկատմամբ:
 - **_top** - փորձ է կատարվում արտապատկերել կանչվող փաստաթուղթը բրաուզերի ընթացիկ պատուհանում, հեռացնելով ֆրեյմերի համախումբը:
 - **_blank** - փաստաթուղթը արտապատկերվում է բրաուզերի նոր պատուհանում, առանց ընթացիկը փակելու:
- Պատկեր 2.7.4-ում ցուցադրված է “_blank” արժեքի կիրառության օրինակը: Ծրագրային կոդը նույնն է, ինչ որ նախորդ



Պատկեր 2.7.4. target ատրիբուտի _blank արժեքի օգտագործման օրինակ

օրինակում, միայն left.html փաստաթղթում <a> տեգի target ատրիբուտին շնորհիվ է “_blank” արժեքը:

Ֆրեյմերի ստեղծման ընփառցում առավել կարեւոր ատրիբուտներն են src և, եթե կան հղումներ այլ փաստաթղթերին, target

ատրիբուտները: Բացի այդ երկուսից գոյություն ունեն ևս մի քանի ոչ պարտադիր ատրիբուտներ, որոնք թույլ են տալիս ավելի ստույգ կարգավորել ֆրեյմերը բրաուզերի պատուհանում: Դրանք են՝

- `noresize="noresize"` - արգելում է ֆրեյմի չափսերի փոփոխությունը:
- `frameborder` – կարող է ընդունել ընդամենը երկու արժեքներ՝ 1 և 0: 1 – նշանակում է որ ֆրեյմը ունի շրջանակ, 0՝ հակառակը:
- `scrolling` - արժեքները՝
`yes` – նշանակում է ֆրեյմի պտտաժապավենի մշտական առկայությունը,
`no` - արգելում է պտտաժապավենի ստեղծումը,
`auto` – պտտաժապավենը ստեղծվում է ըստ անհրաժեշտության՝ այն դեպքում երբ բովանդակությունը լիովին չի տեղավորվում ֆրեյմում:
- `marginwidth` և `marginheight` – սահմանում են հորիզոնական և ուղղահայաց արտաքին լուսանցքները պիքսելներով:
- `longdesc` – օգտագործվում է ֆրեյմի լրիվ URL հասցեն:

Բացի վերը թվարկված ատրիբուտներից `<frameset>` կոնտեյներն ունի **framespacing** ատրիբուտը, որի միջոցով սահմանվում է հեռավորությունը (պիքսելներով) առանձին ֆրեյմերի միջև:

XHTML-ի անցումային տարբերակում դեռ օգտագործվում է նաև ֆրեյմերի յուրահատուկ տեսակը՝ այսպես կոչված **“լողացող ֆրեյմը”** – **iframe** (inline floating frame): Այն թույլ է տալիս ստեղծել `<frameset>` կոնտեյներից անկախ ներտողային ֆրեյմեր սովորական HTML փաստաթղթի `<body>` տեգի ներքո և նշանակված է այլ էջեր արտապատկերելու համար, ինչպես և `<frame />` տեգը: Լողացող անվանումը առաջացել է տեգի `align` ատրիբուտի շնորհիվ՝ դրա միջոցով (`align="right"` կամ `align="left"`) կարելի է `iframe`-ը տեղադրել տեքստի աջից կամ ձախից, ընդ որում տեքստը դրան կշրջանցի: Գրանցման քերականությունը հետևյալն է՝

```
<iframe src="iframe.html" frameborder="1" scrolling="yes" width="300" height="200">
```

Եթե դուք տեսնում եք այս տեքստը, դա նշանակում է, որ ձեր բրաուզերը չի սատարում `iframe` տեգը

`</iframe>`:

Ինչպես տեսնում ենք բերված օրինակից `<iframe>`-ը կոնտեյներ տեգ է, այսինքն ունի բացող և փակող մասեր: Այն դեպքերում, երբ այն չի սատարվում օգտվողի բրաուզերի կողմից, `src` ատրիբուտում նշված ֆայլի փախարեն փաստաթղթում արտապատկերվում է տեգի ներսում գրանցված տեքստը: Բացի `<frame />` տեգին բնորոշ ատրիբուտներից `<iframe>`-ը ունի նաև իրեն հատուկ՝ `width` և

height ատրիբուտները, որոնց միջոցով սահմանվում են ֆրեյմի լայնությունն ու բարձրությունը պիքսելներով:

Կազմենք փաստաթուղթ, որը կցուցադրի <iframe> տեգի կիրառության օրինակը՝ անվանենք այն `iframepage.html`: Լողացող ֆրեյմում արտապատկերվող փաստաթուղթը անվանենք `iframe.html`:

```
<html>
```

```
<head>
```

```
<title>
```

```
iframe example
```

```
</title>
```

```
</head>
```

```
<body>
```

Որոշ դեպքերում հարմար է փաստաթղթի հերքոց ցուցադրել այլ փաստաթղթերի բովանդակությունը: Դա կարելի է իրագործել <iframe>...</iframe> տեգի միջոցով տեղադրելով այն տեքստի ցանկացած մասում:

```
<iframe src="iframe.html" border="1" scrolling="yes" width="200" height="100" align="left">
```

Եթե դուք տեսնում էք այս տեքստը, դա նշանակում է, որ ձեր բրաուզերը չի սատարում է `iframe` տեգը:

Որպեսզի կարողանաք տեսնել այս ֆրեյմի բովանդակությունը

```
<a href="iframe.html">սեղմեք այստեղ</a>
```

```
</iframe>:
```

Տեգը կարող է լինի նաև "լողացող", եթե

```
<b><i>align</i></b>
```

ատրիբուտին շնորհիվ

```
<b><i>align</i></b>
```

կամ <i>align</i>

արժեքը (տվյալ դեպքում `align="left"`): Ի տարբերություն ֆրեյմային կառուցվածք ունեցող փաստաթղթերի այս դեպքում առկա է նաև

```
<b>&lt;body>&lt;/b>
```

տեգը:

```
</body>
```

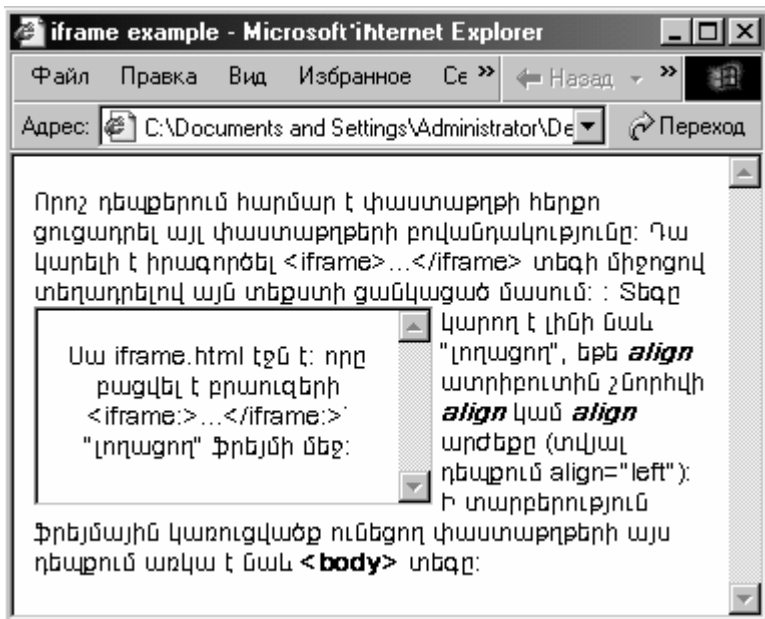
```
</html>
```

Ֆայլի արտապատկերումը ցուցադրված է պատկեր 2.7.5-ում:

Ինչպես նշեցինք <iframe> տեգը չի ընդգրկվել XML ստանդարտում և հետագայում այն կփոխարինվի <object> տեգով:

META-որոշիչները HTML փաստաթղթերի վերնագրային էլեմենտներ են, որոնք ունեն զուտ ծառայողական նշանակություն: Դրանք չեն ազդում բրաուզերում web-էջերի արտապատկերման վրա և ոչ մի կերպ չեն փոփոխում փաստաթղթի բովանդակու-

թյունը: Ընդհանուր առմամբ meta-որոշիչները նկարագրում են փաստաթղթի հատկությունները և շատ հաճախ անտեսվում են դիզայներների կողմից: Սակայն, ինչպես քաջ հայտնի է՝ ոչ մի ծրագրավորման լեզվում ոչ մի հրաման չի ստեղծվում “հենց այնպես”: Դրանք բոլորն էլ ունեն որոշակի ֆունկցիոնալ նշանակում:



Պատկեր 2.7.5. “Լողացող” ֆրեյմի կիրառության օրինակ

§2.8. META - որոշիչները

meta-որոշիչները գրանցվում են <head>...</head> սեկցիայում անմիջապես փաստաթղթի <title> վերնագրից հետո: Տարբերում են որոշիչների երկու խմբեր, որոնք ունեն տարբեր ատրիբուտներ և, համապատասխանաբար՝ տարբեր նշանակում:

Առաջին խմբին են պատկանում այն որոշիչները, որոնք ունեն **name** ատրիբուտը և օգտագործվում են բրաուզերին որոշ քողարկված ինֆորմացիա հաղորդելու համար: Գրանցման քերականությունը հետևյալն է՝ <meta name="արժեք1" content="արժեք2" />: Այդ խմբի առավել հաճախ կիրառվող որոշիչներն են՝ <meta name="description" content="կայքի հակիրճ նկարագրությունը" /> և

<meta name="keywords" content="անհրաժեշտ բանալիական բառերի թվարկումը ստորակետներով բաժանած" />:

Առաջին որոշիչը օգտագործվում է փնտրող սերվերների կողմից (դրանց մասին մենք խոսել ենք առաջին գլխում) web-կատալոգներում կայքը նկարագրելու համար: Սերվերում դիմամիկ կերպով ստեղծվող հաշվետվությունում ընդգրկվում է կամ այդ որոշիչում գրանցված տվյալները կամ այն ինֆորմացիան, որն առաջինն է հանդիպում HTML փաստաթղթի տեքստում: Դա նշանակում է, որ լավ ձևակերպված meta որոշիչի առկայությունը բազմակի անգամ ավելացնում է հնարավորությունը, որ ռեսուրսը (կայքը) կընդգրկվի օգտվողի պահանջով գտնված կայքերի առաջին տասնյակում:

Երկրորդ որոշիչը content ատրիբուտում ընդգրկում է այն բանալիական բառերը, որոնք փնտրող մեքենան կկապակցի կայքի հետ և եթե օգտվողը մուտքագրի այդ բառերից որևէ մեկը փնտրման դաշտում, ապա շատ հավանական է, որ հղումը կայքին կհայտնվի փնտրման արդյունքներում: Փնտրող մեքենաների կողմից meta որոշիչների մշակման մեխանիզմը հասկանալու համար բերենք պարզագույն օրինակ: Ենթադրենք html փաստաթղթում գրանցված են հետևյալ որոշիչները՝

```
<head>
```

```
<title>Սպիտակ արջերի բազմացումը Հայաստանի
```

```
պայմաններում</title>
```

```
<meta name="description" content="Սայթ, որը նվիրված է  
Հայաստանի Սյունիքի մարզում սպիտակ արջերի բազմացման  
պրոբլեմներին" />
```

<meta name="keywords" content="սպիտակ արջեր,ՍՊԻՏԱԿ ԱՐ-
ՋԵՐ,արջ,ԱՐՋ,բազմացում,ԲԱԶՄԱՑՈՒՄ,կենդանիներ,ԿԵՆԴԱՆԻՆԵՐ
" />

</head>

Իսկ այժմ ենթադրենք օգտվողը դիմելով փնտրող մեքենայի
ռեսուրսներին հավաքել է թեմատիկ հարցման դաշտում “սպիտակ
արջեր” տեքստը: Որոշ ժամանակ անց տվյալ թեմային նվիրված
բոլոր գտնված ռեսուրսների ընդհանուր ցուցակում փնտրող
սերվերը կգրանցի նաև հետևյալը՝

- ◆ Սպիտակ արջերի բազմացումը Հայաստանի պայմաններում:
- ◆ Սայթ, որը նվիրված է Հայաստանի Սյունիքի մարզում սպիտակ
արջերի բազմացման պրոբլեմներին:
- ◆ http://www.կայքի_հասցեն:

Հետևյալ երկու որոշիչները սահմանում են կայքում ներկա-
յացված ինֆորմացիայի գործողության տիրույթը՝

<meta name="distribution" content="Global" />,

<meta name="rating" content="General" />:

Իհարկե եթե կայքը նշանակված է միայն Ձատկի կղզու բնա-
կիչների համար, կարելի փոխել content ատրիբուտների արժեք-
ները, սակայն մյուս բոլոր դեպքերում խորհուրդ է տրվում գրանցել
այդ որոշիչները վերը բերված տեսքով:

Այն փնտրման մեքենաների համար, որոնք չեն կարողանում
մշակել <title> տեգի պարունակությունը (փաստաթղթի վերնագի-
րը) նշանակված է հետևյալ որոշիչը՝

<meta name="title" content="Փաստաթղթի վերնագիրը" />:

Web կայքի ստեղծման դատան թելադրվում է փնտրող
մեքենաներին հետևյալ որոշիչի օգնությամբ՝

<meta name="site-created" content="04-06-2005" />:

Սերվերներին կարելի է նաև թելադրել կայքի վերահինդեքսաց-
ման ժամկետները, օրինակ՝

<meta name="revisit" content="10 days" /> կամ

<meta name="revisit-after" content="10 days" />:

Հնարավոր է նաև թույլ տալ կամ արգելել փնտրող ռոբոտներին
ինդեքսավորել կայքը: Օրինակ եթե հետևյալ դիրեկտիվը

<meta name="robots" content="noindex" />

արգելում է կայքի ինդեքսավորումը, իսկ

<meta name="robots" content="all" />

դիրեկտիվը թույլ է տալիս ինդեքսավորել կայքի բոլոր էջերը:

Կարելի է գրանցել նաև հեղինակային իրավունքներին վերաբեր-
վող ինֆորմացիան՝

<meta name="copyright" content="© 2005 Pumpkin Ltd" />

<meta name="autor" content="Չեղինակի անունը" />:

Բերենք առաջին խմբի որոշիչների ևս մի քանի օրինակներ՝
<meta name="owner" content="Սեփականատիրոջ կամ կոմպանի-
այի անվանումը" />

<meta name="address" content="Օֆիսի հասցեն" />

<meta name="owner-type" content="personal/company" />

<meta name="home-url" content="URL հասցեն" />

<meta name="generator" content="Notepad" />:

Երկրորդ խմբի որոշիչները name ատրիբուտի փոխարեն պա-
րունակում **http-equiv** ատրիբուտը: Ի տարբերություն առաջին խմբի
որոշիչների, որոնք կրում են զուտ ինֆորմատիվ բնույթ, երկրորդ
խմբի meta-ները որոշակի դեր են խաղում http արձանագրության
միջոցով փաստաթղթերի հաղորդման գործում: Օրինակ՝

<meta http-equiv="refresh" content="N;url=http://www.կայքի_անուն/
էջի_անուն" />

դիրեկտիվը էջը բացվելուց N վարկյան հետո ավտոմատ կերպով
ապահովում է անցումը content ատրիբուտում նշված հասցեով: Այդ
հնարավորությունը կոչվում է redirect կամ օգտվողի բրաուզերի
վերաուղղում որևէ այլ ռեսուրսին: Սովորաբար այդ հնարքը
կիրառվում է այն դեպքերում, երբ փոխվում է կայքի հասցեն: Տվյալ
տեղը տեղադրվում է index.html էջում որևէ բացատրող տեքստով,
օրինակ՝ “Ներեցեք, մեր կայքը տեղափոխվել է հետևյալ հասցեով՝
նոր հասցեն”: Եթե տեղը գրանցվի հետևյալ տեսքով՝

<meta http-equiv="refresh" content="N" />, ապա ընթացիկ էջը
կբեռնվի բրաուզեր ամեն N վարկյանը մեկ: Հաջորդ տեղը՝

<meta http-equiv="content-type" content="text/html; cahrset=ISO
8859-1" />

“խիստ” կերպով թելադրում է բրաուզերին էջի կիրառական կոդա-
վորման տիրույթը կամ, այսպես կոչված տառանիշերի կոդերի
բազմությունը: Տեղը հարկավոր է օգտագործել զգուշորեն, քանի
որ եթե էջում ներկայացված տեքստի կոդավորումը չհամընկնի
տեղում նշվածի հետ, ապա տեքստը կդարձա անընթերցելի:

§ 2.9. Web-խմբագրիչները

Ինտերնետի սկսնակ օգտվողների շրջանում կարծիք է ստեղ-
ծվել, որ անհատական web-կայքեր ստեղծելու համար տեքստի
գծանշման լեզվի կառուցվածքների ուսումնասիրությանը պարտա-
դիր չէ, քանի որ բացի web-փաստաթղթերի ստեղծման “դասական”
եղանակից (ծեռքով) գոյություն ունի նաև ևս մեկ բավականին
տարածված՝ web-խմբագրիչների կիրառությունը: Իհարկե, վի-

զուլալ խմբագրիչները հանդիսանում են բավականին հարմար գործիքներ ինտերնետի բազմաթիվ ռեսուրսների մշակման համար, սակայն HTML-էջերի վերջնական հարդարումը և կարգավորումը վերջ ի վերջո բերվում է սկզբնական ծրագրային կոդի խմբագրմանը: Իսկ դրա համար պետք է իմանալ և լավ պատկերացնել, թե ի՞նչ է անում խմբագրիչով ավտոմատ կերպով գեներացված կոդի յուրաքանչյուր տողը:

Բոլոր դեպքերում խմբագրիչների օգտագործման առավելությունները ակնհայտ են՝ բացառվում է մեծածավալ ծրագրային կոդի ամբողջովին ձեռքով հավաքելու անհրաժեշտությունը: Գործնականում ցանկացած web-խմբագրիչ հնարավորություն է ընձեռում տեղադրել աղյուսակներ, ստեղծել շրջանակներ և, նույնիսկ, կցել որոշ սկրիպտեր: Իսկ այսպես կոչվող “պրոֆեսիոնալ” խմբագրիչների մեծամասնությունը կարողանում է աշխատել ոճերի բարդ աղյուսակների հետ և իրագործել DHTML-ին հատուկ բազմաթիվ գործառնություններ: Վերջիններից կարելի է նշել Microsoft FrontPage 2002, Adobe GoLive, Macromedia Dreamweaver և Netscape Composer խմբագրիչները: Առավել լայն կիրառում են ստացել Macromedia Dreamweaver և Microsoft FrontPage 2002-ը:

Պրոֆեսիոնալ web-դիզայներների մեծամասնության կարծիքով կայքերի ստեղծման լավագույն գործիքն է հանդիսանում Dreamweaver-ը: Այն չափազանց հարմար է HTML-էջերի ստեղծման վիզուալ և տեքստային մոտեցումների համատեղումով: Ինտերֆեյսը ձևավորված է այնպես, որ օգտվողը կարող է աշխատել գործիքային վահանակների միմյանց քանակով, քանի որ մեկ վահանակում հնարավոր է տեղադրել առավել անհրաժեշտ և հաճախ օգտագործվողները, տեղափոխելով դրանք մյուս վահանակներից:

Dreamweaver-ը ընդգրկում է նաև հզոր տեքստային պրոցեսորի հնարավորությունները: Օրինակ՝ էկրանի ստորին մասի Properties (հատկություններ) պատուհանը թույլ է տալիս անմիջապես փոփոխելով էջի առանձնացված էլեմենտների հատկությունները զուգահեռ տեսնել նաև ծրագրային կոդում կատարված փոփոխությունները: Ընդ որում եթե խմբագրիչի կողմից կատարված փոփոխությունները ձեզ չեն բավարարում, ծրագրային կոդը հնարավոր է նաև խմբագրել “ձեռքով”:

Խմբագրիչն ունի երեք ռեժիմներ՝ ծրագրային կոդի դիտարկման, արդյունքային էջի դիտարկման և կրկնակի պատուհանի: Վերջինի վերին մասում արտացոլվում է ծրագրային կոդը, իսկ ստորինում՝ դրա գրաֆիկական ներկայացումը: Երկու պատուհաններն էլ ակտիվ են՝ կարելի է խմբագրել և կոդը, և գրաֆիկան,

ընդ որում փոփոխությունները անմիջապես կատարվում են մյուս պատուհանում:

Մյուս առանձնահատկություններից հարկ է նշել JavaScript-ի ներկառուցված կարգավորիչը, որի միջոցով կարելի է գտնել և ուղղել սխալները սկրիպտերում: Բացի այդ, քանի որ որ Dreamweaver-ը Macromedia ֆիրմայի արտադրանքն է, ապա հնարավոր է հենց խմբագրիչի ներքո ստեղծել որոշակի անհմացիոն էֆեկտներ:

Կարելի է ասել, որ Dreamweaver-ն ունի մեկ թերություն՝ տվյալների բազաների հետ աշխատելու հնարավորության տեսակետից: Սակայն և այդ թերությունը կարելի է վերացնել, եթե Dreamweaver-ի հետ զուգահեռ տեղադրվի նաև Dreamweaver UltraDev-ը, որն իրենից ներկայացնում է հզոր փաթեթ ASP, JSP և Cold Fusion սատարումով:

FrontPage-ը Microsoft Office ֆիրմայի կողմից արտադրված խմբագրիչ է, որն ապահովում է web-կայքերի պատրաստման ավտոմատացումը: Կայքեր պատրաստողներին այն տրամադրում է հարմար գործիքներ, որոնք ապահովում են դրանց ստեղծման լրիվ ցիկլը՝ ուսուցողական բնույթի կայքերից մինչև խոշոր կորպորատիվ կայքերը:

Նմուշների (Template) օգտագործումը ընձեռում է կայքերի ստեղծման լրացուցիչ հնարավորություններ: Նմուշները կիսապատրաստ կայքեր են որոշակի տրամաբանական և ֆայլային կառուցվածքներով, էջերի դիզայնով, գերհղումների վահանակների և ինֆորմացիոն բլոկների մակետներով: Այդ մակետներում անհրաժեշտ արժեքները գրանցելով կարելի է պատրաստել նման կայքեր:

FrontPage համակարգի գործառնությունը իրագործվում է մի շարք համալիր ծրագրերի և ընդլայնումների միջոցով (FrontPage Extensions), որոնցից հիմնականներն են FrontPage Explorer-ը և FrontPage Editor-ը: Դրանք գործածվում են կայքերի ֆայլային կառուցվածքների ստեղծման և էջերի ինֆորմացիոն և գրաֆիկական բաղադրամասերի ձևավորման համար:

Աշխատելով FrontPage Explorer ծրագրի հետ նախագծողը կարող է ստեղծել նոր ֆայլեր, այնուհետև անցնելով FrontPage Editor ծրագրին՝ ձևավորել դրա ինֆորմացիոն բաղադրամասերը: Դրանից հետո կարող է նորից անցնել FrontPage Explorer ծրագրին, դիտել site-ի ֆայլային կառուցվածքը, ստեղծել նոր ֆայլ, նորից վերադառնալ FrontPage Editor ծրագրին և շարունակել կատարել աշխատանքներ էջերի ինֆորմացիոն բաղադրամասերի հետ:

Որպես Microsoft Office-ի մի մաս FrontPage-ին հատուկ են բազմաթիվ օժանդակ վարպետներ, հուշարարներ, օգնականներ և այլն: Հենց այդ էլ բերում է նրան, որ այն ամենը ինչ ստեղծվում է FrontPage-ի միջոցով ստացվում է բավականին ստանդարտ տեսքով: Իսկ, քանի որ FrontPage-ը խիստ կախում ունի ընդլայնումներից, ապա պարտադիր պահանջվում է վերջիններս տեղադրել սերվերի վրա (առանց դրա շատ մշակումներ պարզապես չեն աշխատի): Դա էլ հանդիսանում է պատճառ, որ խմբագրիչը հիմնականում օգտագործվում է միայն ոչ մեծ անհատական կայքեր ստեղծելու համար:

Լաբորատոր աշխատանքների առաջադրանքներ

Լաբորատոր աշխատանք N1

1.1. Կազմել փաստաթուղթ, որն ունի տեքստի հետևյալ գծանշումը՝

Մարքետինգի սահմանումները կարելի է միավորել երկու հիմնական խմբերի՝ **դասական** (սահմանափակ) և **ժամանակակից** (ընդհանրացված):

Մարքետինգի “ավանդական” սահմանումները ենթադրում են , որ գլոբալորր ապրանքների և ծառայությունների ֆիզիկական տեղաշարժն է և դրա հետ կապված ունեն մի շարք թերություններ:

1.2. Օգտագործելով <pre></pre> տեգը կազմեք հետևյալ աղյուսակը՝

Տարածաշրջան	Չեռքի ածխատանք	Մտավոր աշխատանք
Հարավ-արևմուտք	\$ 40	\$ 50
Հարավ-արևելք	\$ 35	\$ 45
Հյուսիս-արևմուտք	\$ 40	\$ 65
Հյուսիս-արևելք	\$ 25	\$ 35

1.3. Կազմեք էջ, որն արտացոլի հետևյալ չկարգավորված ցուցակը՝

ԱՄՆ-ի ստանդարտ արդյունաբերական դասակարգիչը

- Գյուղական և անտառային տնտեսություն
- Ձկնորսություն
- Շինարարություն
- Արդյունաբերության
- Տրանսպորտ
- Հաղորդակցման ուղիներ
- Էլեկտրա և գազամատակարարում
- Բժշկական սպասարկում
- Մեծածախ և մանրածախ առևտուր

- ֆինանսներ
- Ապահովագրություն և անշարժ կայք ծառայություններ

1.4. 1.3-ում բերված չկարգավորված ցուցակը ներկայացնել կարգավորված (համարակալված) ցուցակի տեսքով:

1.5. Կազմել էջ, որը արտացոլի հետևյալ ցուցակը՝

Травяные чаи

I Ромашка

II Мятный чай

◆ Перечная мята

◆ Обычная мята

III Женшень

IV Шиповник

1.6. Հավասարեցրեք կամայական պատկեր ձախից (լողացող պատկեր), այնպես, որ այն շրջանցվի հետևյալ տեքստով՝

Ջրոսաշրջիկների համար առավել գրավչական են Ինուն և Սպայուն ձմեռային տեսարանները:

Լաբորատոր աշխատանք N2

2.1. Կազմել էջ, որը արտացոլի հետևյալ աղյուսյակը՝

0719			
Գերազանց	Լավ	Բավարար	Անբավարար
30	40	20	10

Տարբեր տեսակների կոնֆետների վաճառքի ճակարդակը

Տեսակը	Մեկինն	Աշոտ	Ռուբեն
Շոկոլադե կոնֆետներ	50	25	50
Մոգային կոնֆետներ	50	40	30
Կարամել	15	25	40

2.2. Կազմել էջ, որն արտացոլի հետևյալ աղյուսակը

2.3. Կազմել էջ, որը արտացոլի հետևյալ աղյուսակը՝

Լաբորատոր աշխատանք N3

3.1. Կազմել էջ, որը լուսաբանի հետևյալ շրջանակները

Աշխատանքային տիրույթ			
Հղումներ	Հղումներ	Հղումներ	Հղումներ

3.2. Կազմել էջ, որը լուսաբանի հետևյալ շրջանակները և կազմել յուրաքանչյուր շրջանակի համար փաստաթուղթ, որն արտապատկերի հետևյալ ինֆորմացիան՝

Գովազդի ֆրեյմ	
Ժամանակ	Փաստաթղթերի արտապատկերման հիմնական էջ (Լաբորատոր աշխատանք 2.2-ի աղյուսակը)
Հղումների մեկուկ	

ԳԼՈՒԽ 3. ԴԻՆԱՄԻԿ WEB-ԷՋԵՐԻ ՍՏԵՂԾՈՒՄ, JAVASCRIPT ՍՏԵՆԱՐԱՅԻՆ ԼԵՁՈՒՆ

Կայք ստեղծելիս յուրաքանչյուր հեղինակ, որպես կանոն, նախատեսում է, որ այն պետք է լինի ինտերակտիվ, իսկ դա մշակակում է այցելուների հետ այս կամ այն աստիճանի փոխհամագործակցության կազմակերպում: Web-էջերը կարող են հետադարձ կապ ունենալ օգտվողների հետ, ապահովել պատվերների ընդունում, տարբեր հարցերի քննարկում և, մույնիսկ ընծեռել միմյանց հետ շփվելու հնարավորություն: Սայթերի ինտերակտիվությունը և դինամիկությունը ապահովվելու համար ամիրաժեշտ պայմաններից մեկը, ինչպես արդեն մշակվել է առաջին գլխում՝ սերվերում տեղադրվող հատուկ ծրագրերի ստեղծումն է:

Սակայն ոչ բոլոր դեպքերում է նպատակահարմար օգտվողից ստացվող տվյալները կամ հարցումները մշակել սերվերում: Web-էջի տարբեր էլեմենտների հետ աշխատելու, օգտվողի կողմից մուտքագրման պրոցեսին փոխազդելու, ֆորմաների տվյալները որոշակի ալգորիթմների միջոցով մշակելու և բազմաթիվ այլ՝ սերվերային մշակում չպահանջող խնդիրներ լուծելու համար օգտագործվում են, այսպես կոչված, **սցենարային լեզուները**, որոնցից առավել տարածվածները՝ JavaScript և VBScript լեզուներն են:

Սցենարային լեզուները աշխատում են անմիջականորեն բրաուզերի ներքո (հիարկե եթե համատեղելի են վերջինի հետ) և օգտագործվում են՝ ինչքան էլ դա տարրիմակ կարող է թվալ, կամ պարզ, ոչ մեծ, կամ չափազանց բարդ խնդիրների լուծման համար:

Սակայն նախկան սցենարային լեզուները և դրանցով կազմած սցենարները (հետագա շարադրության ընթացքում սցենարները

մենք կանվանենք սկրիպտեր) ուսումնասիրելը անհրաժեշտ է պարզաբանել երկու կարևորագույն հիմնադրիչ հասկացություններ, որոնց միջոցով էլ յուրաքանչյուր web-էջ հնարավոր է դարձնել վառ, պատկերազարդ և դինամիկ: Դրանք են՝ HTML ֆորմաները և փաստաթղթի օբյեկտային մոդելը (Document Object Model - DOM):

§3.1. HTML ֆորմաները

Ֆորմաները թույլ են տալիս հարցումների միջոցով օգտվողից ստանալ անհրաժեշտ ինֆորմացիան, ընդ որում տվյալների մի մասը կարող է ձևակերպվել որպես պատրաստի մենյուներ (ցանկեր), որոնցից օգտվողը կարող է ընտրել անհրաժեշտ կետը: Հարցումները կարող են լինել հետևյալ տեսակների՝ հարցեր, որոնց պատասխանը անհրաժեշտ է հավաքել հատուկ տեքստային դաշտերում, հարցեր, որոնց պատասխանը պետք է ընտրվի առաջարկվող ցուցակից և, որը կարող է կազմակերպված լինել մենյուների կամ փոխարկիչների համախմբի տեսքով, հարցեր, որոնց միակ պատասխանը ընտրվում է փոխարկիչների համախմբից:

Ֆորմաները ստեղծվում են HTML այնպիսի տեգերի միջոցով, որոնք սահմանում են մուտքի դաշտեր, փոխարկիչներ, դրոշանշիչներ և այլ ղեկավարման էլեմենտներ: Յուրաքանչյուր այդպիսի ղեկավարման էլեմենտ պետք է ունենա եզակի անուն, որը հետագայում կօգտագործվի որպես փոփոխական և այն ընտրությունը, որը կկատարի օգտվողը կդառնա այդ փոփոխականի արժեքը: Օրինակ, եթե մուտքագրման տեքստային դաշտին շնորհիվ “city” անուն, իսկ օգտվողը գրանցի այդ դաշտում հարազատ քաղաքի անունը՝ Երևան (կամ որևէ այլ), ապա դա կնշանակի, որ city փոփոխականին շնորհիվ է “Երևան” արժեքը (city=“Երևան”):

Փոփոխականները և դրանց արժեքները հիմնականում հաղորդվում են կամ սցենարին և մշակվում անմիջապես օգտվողի համակարգչի վրա կամ սերվերին, որը սերվերային ծրագրերի միջոցով վերծանում է և մշակում ստացվող տվյալները: Երկու դեպքում էլ մշակման արդյունքում web-էջի տեսքը փոփոխվում է: Օգտվողի կողմից գրանցված տվյալներից կախված, էջում կարող են հայտնվել նոր տվյալներ կամ որևէ պարզ պատասխան, օրինակ “Շնորհակալություն, որ դիմեցիք մեզ”:

Ֆորմաները ստեղծվում են հատուկ **<form>** տեգի միջոցով, որը հանդիսանում է կոնտեյներ այլ էլեմենտների համար: Տեգի գրանցման ընդհանուր քերականությունը հետևյալն է՝

<form>

Ֆորմայի էլեմենտները

</form>

Այն դեպքերում, երբ օգտվողի պատասխանները պետք է մշակվեն սերվերում տեղն ունի նվազագույնը երկու ատրիբուտներ՝

1. **action** – պարունակում է ֆորմայի տվյալները մշակող ծրագրի URL հասցեն: Եթե դա սերվերային ծրագիր է ապա այն ունի, օրինակ հետևյալ տեսքը՝ “http://www.mysite.am/cgi-bin/program.pl”:

2. **method** – որոշում է սերվերին տվյալները հաղորդելու եղանակը և կարող է ընդունել հնարավոր երկու արժեքներ՝

- ♦ **get** – այս եղանակը օգտագործելու դեպքում հաղորդվող տվյալները կցվում են URL հասցեին: Մեծ մասամբ դա լրջորեն սահմանափակում է տվյալների ծավալը (սովորաբար ոչ ավել, քան 100 սիմվոլ): Իհարկե, եթե ֆորման բաղկացած է մեկ - երկու էլեմենտներից և պահանջվում է ապահովել հաղորդման մեծ արագություն, ապա նախընտրելի է օգտագործել հենց այս մեթոդը;

- ♦ **post** – օգտագործվում է այն դեպքերում, երբ ֆորմաները պարունակում են զգալի թվով մեծածավալ ինֆորմացիա: Տվյալները հաղորդվում են URL հասցեից առանձին և ծավալը գործնականում սահմանափակված չէ:

Վերը նշված երկու հիմնականներից բացի, ֆորմաները կարող են ունենալ նաև որոշ լրացուցիչ ատրիբուտներ: Սցենարների (սկրիպտերի) և սերվերային ծրագրերի օգտագործման պայմաններում առավել կարևոր են ֆորմաների իդենտիֆիկացման համար ծառայող՝ **name** և **id** ատրիբուտները (այդ ատրիբուտները կարևոր նշանակություն ունեն ֆորմայի առանձին էլեմենտների համար՝ դա մենք կքննարկենք քիչ ուշ): Քանի որ **name** ատրիբուտը չի ընդգրկված XML ստանդարտում հարմար է առայժմ օգտագործել այդ երկու ատրիբուտները միաժամանակ:

Նշենք, որ ի տարբերություն այլ կոնտեյներ տեգերի (օրինակ՝ **<p>**, **<frame>**, **<table>** և այլն) **<form>** տեգերը չի կարելի ներդնել միմյանց մեջ՝ հարկավոր է լիովին սահմանել մեկ ֆորման, փակել այն **</form>**-ով և նոր միայն անցնել մի այլ ֆորմայի սահմանումին:

3.1.1. Տեքստային դաշտեր

Ֆորմաների կիրառության առավել տարածված նպատակներից մեկն է՝ ստանալ օգտվողից որոշակի տեքստային տվյալներ, որոնց

միջոցով հնարավոր լինի հետադարձ կապ կազմակերպել կայքում: Դրանք կարող են լինել մեկնաբանություններ, կայքի բովանդակության կամ ձևավորման վերաբերյալ դիտողություններ և այլն: Տեքստի մուտքագրման համար տիրույթը ստեղծվում է`

```
<textarea name="անուն" id="անուն" rows="քիվ" cols="քիվ">
```

Ըստ լռելյայն տեքստը

```
</textarea>:
```

rows և cols ատրիբուտների միջոցով կարելի է սահմանել պատուհանում արտապատկերվող տողերի և սյունակների քանակը, սակայն մուտքագրվող տեքստի ծավալը դրանով չի սահմանափակվում` կարելի է գրանցել այնքան ինֆորմացիա, ինչքան անհրաժեշտ է: Տեգի ներսում սովորաբար գրանցվում է այն տեքստը, որը պետք է արտապատկերվի ըստ լռելյայն, ընդ որում այն դուրս է բերվում էկրանին այնպես, ինչպես հավաքվել է` <pre> տեգի նման: name և id ատրիբուտները միարժեքորեն իդենտիֆիկացնում են տեքստային դաշտը: Կարելի է հիշել ևս մեկ ատրիբուտ` **readonly**, որի միջոցով արգելվում է տեքստային դաշտի պարունակության խմբագրումը` **readonly="readonly"**: Ստորև բերված է տեքստային դաշտի ստեղծման ծրագրային կոդի օրինակ (տես` պատկեր 3.1.1.)`

```
<html><head><title></title></head>
```

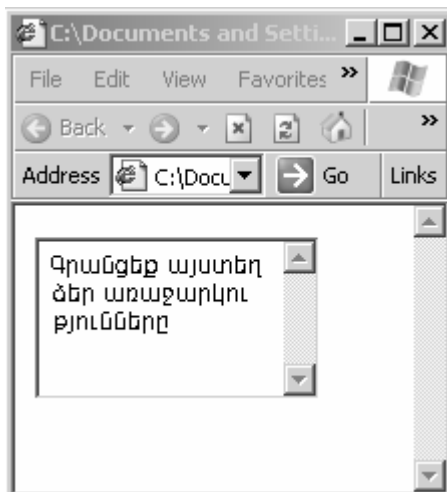
```
<body style="font-family:Arial Armenian;font-size:10pt">
```

```
<textarea name="textarea" id="textarea" rows="5" cols="20">
```

Գրանցեք այստեղ ձեր առաջարկությունները

```
</textarea>
```

```
</body></html>
```



Պատկեր 3.1.1. Տեքստային դաշտի օրինակ

3.1.2. Ղեկավարման էլեմենտներ

Տեքստային դաշտը տեքստային տվյալների մուտքագրման հիմնական գործիքն է՝ դրա օգնությամբ կարելի է մուտքագրել որոշակի ծավալի կամայական տեքստ: Սակայն շատ հաճախ անհրաժեշտ է լինում սահմանափակել օգտվողի կողմից տրված պատասխանների քանակը: Կամայական տեքստի մշակումը չափազանց բարդ խնդիր է և հասկանալի է, որ ավելի հեշտ է մշակել սահմանափակ քանակով նախապես հայտնի արժեքները: Այդ խնդիրը լուծվում է **<input />** տեգի օգտագործման միջոցով: Այն թույլ է տալիս ստեղծել մեկ կամ մի քանի ղեկավարման էլեմենտներ, որոնց տեսակը գրանցվում է տեգի հատուկ՝ **type** ատրիբուտում: Բոլոր տեսակների **<input />** ղեկավարման էլեմենտները միարժեքորեն իդենտիֆիկացվում են **name** ատրիբուտի միջոցով՝

<input type="տեսակ" name="անուն" />:

Մուտքագրման դաշտը` **<input type="text" />** դա նույնն է ինչ և տեքստային դաշտը այն տարբերությամբ, որ թույլ է տալիս մուտքագրել միայն մեկ, ոչ մեծ տող: Սովորաբար մուտքագրման դաշտը կիրառվում է օգտվողի վերաբերյալ որոշակի տվյալներ (ազգանուն, անուն, բնակավայր, ծննդյան դատան և այլն) գրանցելու նպատակով: Ատրիբուտների միջոցով կարելի է սահմանել դաշտի չափսերը (**size**) և նիշերի մաքսիմալ հնարավոր քանակը (**maxlength**), այսինքն կարելի է ստեղծել դաշտ, որի չափսերը գերազանցում են (կամ հակառակը՝ զիջում են) մուտքագրվող տողի չափսերին: ցանկության դեպքում **value** ատրիբուտի միջոցով կարելի նաև սահմանել դաշտի ըստ լրելայն արժեքը: Օրինակ՝ **<input type="text" name="city" size="30" maxlength="40" value="Yerevan" />**:

Ծածկագրի մուտքագրման դաշտը` **<input type="password" />** տարբերվում է սովորական մուտքի դաշտից միայն նրանով, որ մ հավաքվող տեքստը արտապատկերվում է հատուկ նիշերի տեսքով (սովորաբար աստղանիշերով), ինչը թույլ է տալիս քողարկել ծածկագիրը օտար հայացքներից: Նշենք, որ “պաշտպանվածությունը” հենց դրանով էլ սահմանափակվում է, քանի որ ինքը՝ ծածկագիրը բրաուզերը հաղորդում է ակնհայտ ձևով:

- Պաշտպանվածությունը կարելի է որոշ չափով բարելավել, օգտագործելով պաշտպանված HTTP արձանագրություն (այն կոչվում է SHTTP), որը հաղորդում է տվյալները ծածկագրված եղանակով:

Դրոշանշիչները` `<input type="checkbox" />` թույլ են տալիս սահմանել դրոշակի պարամետրերի տրամաբանական արժեքները: Հիմնականում դրոշակները կիրառվում են այն դեպքերում, երբ փոփոխականը կարող է ընդունել երկու հնարավոր արժեքներ (օրինակ` այո կամ ոչ): Օգտագործելով `checked` ատրիբուտը կարելի է սահմանել դրոշակի սկզբնական վիճակը, օրինակ` `checked="checked"` (ըստ լռելայն դրոշակը նշված չէ, այսինքն `checked` չէ): Սովորաբար, նույնիսկ մի քանի դրոշանշիչների առկայության դեպքում, դրանք իրարից անկախ են, այսինքն կարելի է կատարել միաժամանակ մի քանիսի ընտրությունը: Ստորև բերված է դրոշանշիչի օրինակ`

`<input type="checkbox" name="flag" checked="checked" />`

Փոխարկիչները` `<input type="radio" />`, ինչպես և դրոշանշիչները նշանակված են արժեքների տրված բազմությունից ընտրություն կատարելու համար: Տարբերությունը կայանում է նրանում, որ փոխարկիչների միջոցով կատարվում է միակ արժեքի ընտրությունը տրված բազմությունից, իսկ դրոշանշիչները ընծեռում են երկուական ընտրության (այո կամ ոչ) հնարավորություն բազմության յուրաքանչյուր արժեքի համար: Դրանով է բացատրվում այն փաստը, որ փոխարկիչների ամբողջ խմբին շնորհվում է միակ միասնական անուն: Օրինակ հետևյալ ծրագրային հատվածում ստեղծվում է փոխարկիչների խումբ, որի միջոցով կայքի հեղինակները ստանում են ինֆորմացիա այն մասին, թե որտեղից է օգտվողը ստացել ինֆորմացիան կայքի գոյության վերաբերյալ`

Որտեղի՞ց է ձեզ հայտնի դարձել մեր կայքի հասցեն

`<input type="radio" name="where" value="web" checked="checked" />` Ինտերնետի փնտրող ծրագրից կամ
 հղումից

`<input type="radio" name="where" value="advert" />`

Հեռուստա կամ ռադիոգովազդից

`<input type="radio" name="where" value="press" />`

Թերթերից

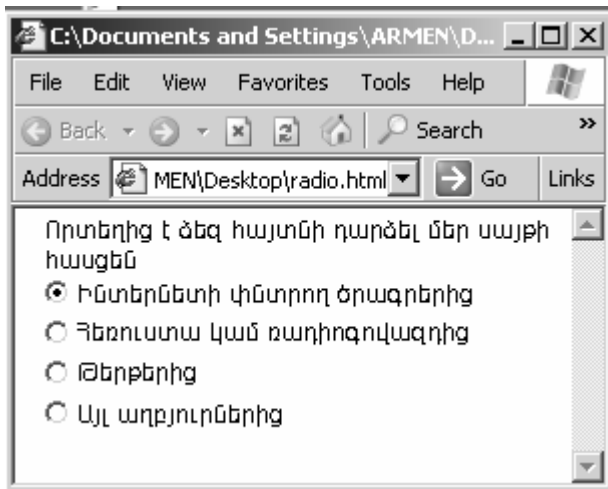
`<input type="radio" name="where" value="other" />`

Այլ աղբյուրներից

Ինչպես տեսնում ենք սկզբնական վիճակում միացված է առաջին փոխարկիչը (`checked="checked"`): Դա նշանակում է, որ ըստ լռելայն **where** փոփոխականին շնորհված է `web` արժեքը (`where="web"`), այսինքն ենթադրվում է, որ ինֆորմացիան կայքի գոյության վերաբերյալ օգտվողը ստացել է ինտերնետից: Քանի որ խմբի բոլոր փոխարկիչներն ունեն միևնույն `where` անունը, ապա

այն դեպքում, երբ օգտվողը միացնի մի այլ փոխարկիչ՝ where փոփոխականին ավտոմատ կերպով կշնորհվի վերջինիս value ատրիբուտում գրանցված արժեքը (ընդ որում նախորդ փոխարկիչը կանջատվի): Պատկեր 3.1.2-ում ցուցադրված է, թե ինչպես կարտապատկերի բրաուզերը օրինակում բերված փոխարկիչների խումբը:

- Փոխարկիչների խումբ ստեղծելիս միշտ հարկավոր է սահմանել ըստ լռելայն ամբողջ (այսինքն նախատեսել փոխարկիչներից մեկի միացած վիճակը), քանի որ եթե օգտվողը անտեսի այդ խումբը՝ տվյալ անունով փոփոխականի արժեքը կմնա անորոշ:



Պատկեր 3.1.2. Փոխարկիչների խմբի օրինակ

Քողարկված դաշտը՝ `<input type="hidden" />` իրականում ոչ մի կապ չունի տվյալների մուտքագրման հետ: Այն օգտագործվում է տվյալների նախապես հայտնի արժեքները պահպանելու համար և չի արտապատկերվում էկրանին: Օրինակ՝ եթե միևնույն ծրագիրը (սկրիպտը) օգտագործվում է մի քանի ֆորմաների տվյալները մշակելու համար, ապա քողարկված դաշտի միջոցով կարելի է հաղորդել՝ ո՞ր ֆորմայի տվյալներն են հաղորդվում ծրագրին: Էլեմենտը ունի երկու ատրիբուտներ՝ name և value, օրինակ՝ `<input type="hidden" name="identify" value="form3" />`:

Մաքրման կոճակը՝ `<input type="reset" />` օգտագործվում է ֆորմայի դաշտերը մաքրելու և դրոշակներն ու փոխարկիչները սկզբ-

նական վիճակի բերելու համար: Այն ունի ոչ պարտադիր value ատրիբուտ, որը թույլ է տալիս միայն փոխել կոճակի վրայի գրանցումը և, եթե այն չօգտագործվի, ապա կոճակի վրա ըստ լռելայն կգրվի “Reset”:

Պնդման (կամ ուղարկման) կոճակը` <input type=“submit” /> ի տարբերություն մաքրման կոճակի հավաստում է ֆորմայում կատարած բոլոր փոփոխությունները և պատրաստում է տվյալները սերվերին ուղարկելու: Այն նույնպես ունի ոչ պարտադիր value ատրիբուտը` կոճակի անվանումը փոխելու համար (ըստ լռելայն կոճակի վրա գրված է “Submit”):

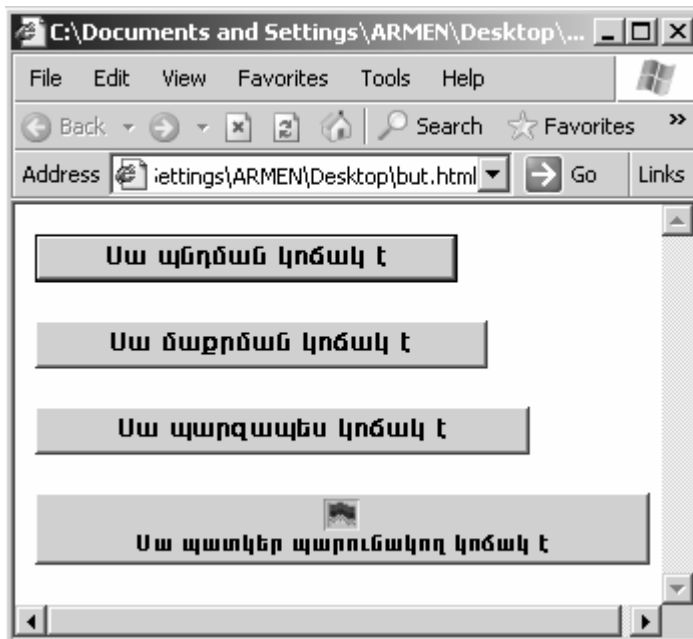
Անհատական կոճակներ ստեղծելու համար նոր ստանդարտը նախատեսում է <button>...</button> կոնտեյներ տեղը: Այն ունի երկու ատրիբուտներ` name և type: Որպես պարտադիր հետագայում հանդիսանալու է միայն կոճակը իդենտիֆիկացնող name ատրիբուտը: type ատրիբուտը կարող է ընդունել երկու հնարավոր արժեքներ` **submit**, **reset**: Դրանց միջոցով ստեղծվում են, համապատասխանաբար պնդման և մաքրման կոճակները: Բոլոր դեպքերում այն ինչ անհրաժեշտ է գրանցել կոճակի վրա տեղադրվում է տեգի ներսում, օրինակ (տես` պատկեր 3.1.3.)`

<button type=“submit”>Սա պնդման կոճակ է</button>

<button type=“reset”>Սա մաքրման կոճակ է</button>

<button>Սա պարզապես կոճակ է</button>

<button>
Սա պատկեր պարունակող կոճակ է</button>



Պատկեր 3.1.3. Ղեկավարման կոճակների օրինակներ

Ցանկերը (մենյու) ստեղծվում են այն դեպքերում, երբ օգտվողին առաջարկվում է ընտրություն կատարել, սակայն չի թույլատրվում մուտքագրել սեփական տեքստը: Ցանկ ստեղծելու համար օգտագործվում է `<select>...</select>` կոնտեյներ էլեմենտը: Պարտադիր է միայն `name` ատրիբուտի առկայությունը (հասկանալի է, որ ծրագրային մշակման դեպքում տեղը միարժեքորեն իդենտիֆիկացնելու համար): Հաճախ օգտագործվում է `size` ատրիբուտը, որի միջոցով կարելի է սահմանել էկրանին միաժամանակ դուրս բերվող կետերի քանակը, օրինակ, եթե գրանցվի՝

```
<select name="products" size="3">...</select>
```

ապա միաժամանակ կցուցադրվեն ցանկի երեք կետերը:

Ցանկի յուրաքանչյուր կետ սահմանվում է `<option>...</option>` տեգով, որն ունի մեկ պարտադիր `value` ատրիբուտը, որում գրանցվում է տվյալ կետի արժեքը: Երբ օգտվողը ընտրում է որոշակի կետ ամբողջ ցանկին շնորհվում է այդ կետի արժեքը: Ըստ լռելյայն սկզբնական վիճակում ընտրված է լինում առաջին կետը: Անհրաժեշտության դեպքում կարելի է որպես նախապես ընտրված՝

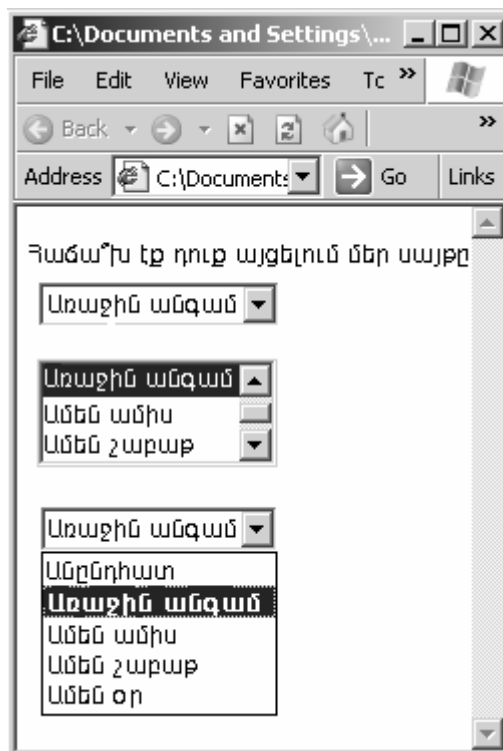
ցուցադրել կամայական կետը, շնորհելով համապատասխան <option> տեգի selected ատրիբուտին “selected” արժեքը: Ստորև բերված օրինակում առաջին ցանկում սկզբնական վիճակում ընտրված է երկրորդ կետը (ըստ լռելյայն ցուցադրվում է միայն մեկ կետ): Երկրորդում նույնպես ընտրված է երկրորդ կետը, սակայն size ատրիբուտի միջոցով (size="3") ցուցադրվում են միաժամանակ երեք կետեր (արտապատկերումը տես՝ պատկեր 3.1.4.)՝

Հաճախ էք դուք այցելում մեր կայքը:

```
<select>
<option value="infinite">Անընդհատ</option>
<option value="first" selected="selected">Առաջին անգամ</option>
<option value="monthly">Ամեն ամիս</option>
<option value="weekly">Ամեն շաբաթ</option>
<option value="daily">Ամեն օր</option>
</select>
<select size="3">
<option value="infinite">Անընդհատ</option>
<option value="first" selected="selected">Առաջին անգամ</option>
<option value="monthly">Ամեն ամիս</option>
<option value="weekly">Ամեն շաբաթ</option>
<option value="daily">Ամեն օր</option>
</select>
```

Պատկեր 3.1.4-ում ցուցադրված է նաև (ստորին մասում) առաջին ցանկը բացված վիճակում:

Երբեմն անհրաժեշտություն է առաջանում միաժամանակ ընտրել ցանկի մի քանի կետեր: Այդպիսի դեպքերի համար նախատեսված է <select> տեգի **multiple** ատրիբուտը: Եթե տեղում գրանցվում է՝ <select multiple="multiple">, ապա օգտվողը կարող է ընտրել ցանկացած քանակության կետեր և նույնիսկ կարող է առհասարակ չընտրել ոչ մեկը:



Պատկեր 3.1.4. Մեկ և բազմատող ցանկեր

Էլեկտրոնային ֆորմաների ձևավորմանը ներկայացվող հիմնական պահանջը՝ պարզությունն է, այսինքն ոչ ուրիշ ընկալումը, էլեմենտների մեկ խմբից մյուսին անցումը, տրամաբանական բաժանումը ըստ թեմաների: Անհրաժեշտ է հաշվի առնել նաև օգտվողների մտածելակերպը՝ նրանց ներկայացվող պահանջների պարզ և հակիրճ շարադրումը ավելի գրավիչ կլինի, քան երկարատև բացատրությունները և խնդրանքները:

§3.2. Ծանոթություն JavaScript լեզվի հետ

JavaScript լեզուն ստեղծվել է Netscape ֆիրմայում HTML-ի հետ համատեղ աշխատանքի համար՝ դինամիկ և ինտերակտիվ

էջեր ստեղծելու նպատակով: Այն լիովին սատարվում է Internet Explorer (IE) և Netscape Navigator (NN) բրաուզերների կողմից (ի տարբերություն VBScript-ի, որը սատարվում է միայն Microsoft ֆիրմայի արտադրանքի կողմից): JavaScript-ն ու նրա մոտակա “ազգականները” ոճերի աղյուսակների և HTML-ի հետ միասին կազմում են այն ինչ կոչվում է Դինամիկ HTML:

Հարկ է նշել, որ JavaScript-ը ոչ մի առնչություն չունի Java-ի հետ: Վերջինս, որը ստեղծվել է Sun Microsystems ֆիրմայում համդիսանում է C++-ի (Java-ն C++-ի անմիջական ժառանգն է) կարգի լիարժեք (կոմպիլացվող) ծրագրավորման լեզու, ընդ որում, կոդմոորոշված լինելով վիրտուալ մեքենաների և համակարգերի զաղափարի վրա, այն կարող է աշխատել տարբեր օպերացիոն համակարգերի ներքո, այսինքն կախված չէ պլատֆորմից: Անվանումների նմանությունը բացատրվում է լեզուների հրամանների գրանցման քերականության նմանությունից:

Շատ բրաուզերներ ունեն Java սատարում, որը նման է մուլտիմեդիայի սատարմանը: Հավելվածները աշխատում են անմիջապես բրաուզերի պատուհանում և տպավորություն է ստեղծվում, իբր օգտվողը աշխատում է սեփական համակարգչի վրա տեղադրված ծրագրի հետ:

Համակարգչի սովորական ծրագրային և տեխնիկական ապահովումը (մասնավորապես՝ պրոցեսորը) պետք է լինեն համատեղելի: Օրինակ Macintosh-ի ծրագրագրային ապահովումը չի աշխատի Windows-ի համար նախատեսված պրոցեսորի և սարքավորումների վրա: Սակայն Java-ի միջոցով կարելի է ստեղծել ստանդարտ համակարգիչ, այսինքն վիրտուալ մեքենա, որը հիմնված է միայն ծրագրային էլեմենտների վրա: Եվ Windows-ի կամ Macintosh-ի համար ծրագրավորելու փոխարեն, հնարավորություն է ընձեռվում ստեղծել հավելվածներ վիրտուալ մեքենաների համար, որոնք կախում չեն ունենա համակարգչային պլատֆորմներից:

Ներկայումս JavaScript-ը առավել լայն կիրարկող լեզուն է սկրիպտեր գրելու համար: VBScript-ը խիստ հետ է մնում, սակայն զբաղեցնում է պատվավոր երկրորդ տեղը web-սցենարներ գրելու գործում, չնայած սատարվում է միայն Microsoft տեխնոլոգիաներով և միայն Internet Explorer բրաուզերներով:

Քննարկենք այժմ այն երկու գլխավոր հասկացությունները, որոնց վրա են հիմնվում բոլոր web-սցենարները՝ **“Ֆունկցիա”** և **“իրադարձությունների (պատահարների) մշակում”**: Սցենարների մեծամասնությունը բաղկացած է երկու մասերից՝

- առաջինը գտնվում է փաստաթղթի վերնագրային (<head>) մասում կամ, առհասարակ, առանձին ֆայլում և պարունակում է մշակման ալգորիթմների ծածկագրերը (կոդերը)՝ մշակող ֆունկցիաները,
- երկրորդում, որը գտնվում է փաստաթղթի մարմնում (<body>) գրանցվում են այդ ֆունկցիաների կանչերը:

Կանչերի միջոցով ֆունկցիաներին հաղորդվում են արգումենտների փաստացի արժեքները, որոնք անհրաժեշտ են աշխատանքի համար: Կանչը ընդունելուն պես սկսվում է ֆունկցիայում գրանցված մշակման ալգորիթմների կատարումը, այսինքն հաշվարկներն ու ձևափոխությունները: Մշակման արդյունքները տարբեր եղանակներով վերադարձվում են սկրիպտին:

Իհարկե այդ պրոցեսը իմաստ չէր ունենա, եթե ֆունկցիաների կանչերը կատարվեին հաջորդաբար՝ այդ դեպքում կանչերը առհասարակ պետք չէին լինի և ամբողջ սկրիպտը կարելի էր տեղադրել փաստաթղթի մեկ մասում: Սակայն ֆունքցիաների արժեքը հենց նրանում է, որ դրանք կարելի է օգտագործել բազմակի անգամ և կամայական հաջորդականությամբ՝ հենց այստեղ է սկսում գործել պատահարների մշակման սկբմունքը: Այն հետևյալն է՝ սկրիպտը “սպասում է” որևէ պատահարի (օրինակ ստեղծի սեղմում, թայմերի որևէ արժեք, մկնիկի նշիչի շարժում և այլն), որի կատարման դեպքում փաստացի արգումենտները հաղորդվում են ֆունկցիային և սկսվում է դրանց մշակումը: Գործարկվում են միայն այն ֆունկցիաները, որոնք անհրաժեշտ են դրված խնդիրների լուծման համար, ընդ որում միայն այն դեպքում, երբ տվյալ պատահարը ճանաչվում է սքրիպտի կողմից:

- Նշենք, որ պատահարը դա միայն օգտվողի կողմից կատարվող գործողություն չէ: Այն կարող է լինել համակարգի թայմերի որևէ արժեք, էջի բեռնման փաստ, որևէ մեծության արժեքի փոփոխում և այլն: Ամեն ինչ կախված է նրանից, թե ինչ է նախատեսել ծրագրավորողը սցենարում:

3.2.1. Սցենարների ներդրումը փաստաթղթերում

Սցենարներ գրելու համար չեն պահանջվում հատուկ հավելվածներ: Այն ամենն ինչ անհրաժեշտ է՝ դա հնարավորին չափ պարզ տեքստային խմբագրիչն է, որը չի զբաղվում “ինքնագործունեությունով” և չի ֆորմատավորում հավաքած տեքստը յուրովի (ինչպես դա անում են բարդ խմբագրիչները կամ տեքստային պրոցեսորները, օրինակ, Word-ը): Սցենարը ներդրվում է էջի վերնագրային մասում **<script>...</script>** կոնտեյներ տեղի օգնու-

թյամբ: Տեգի **type** ատրիբուտի միջոցով կարելի է նշել նաև սկրիպտի համար օգտագործվող ծրագրավորման լեզուն, օրինակ՝ `<script type="text/javascript">`(կամ `<script type="text/vbscript">`)
Սցենարի ֆուկցիաների նկարագրությունը (կոդը)
`</script>`

- Հին բրաուզերների հետ համատեղելիությունը ապահովելու համար կարելի է **type** ատրիբուտի հետ մեկտեղ օգտագործել **language** ատրիբուտը, օրինակ՝ `<script type="text/javascript" language="javascript">`: Հին բրաուզերները սցենարը կարող են հասկանալ որպես HTML գծանշում և դրանից խուսափելու նպատակով խորհուրդ է տրվում նաև “քոդարկել” այն մեկնաբանության նշաններով՝
`<script type="text/javascript">`
`<!--`

Սցենարի ֆուկցիաների նկարագրությունը (կոդը)

`//→`

`</script>`:

Երկու շեղ գծերը `“/”` գրանցվում են, որպեսզի `javascript-`ի ինտերպրետատորը չմեկնաբանի `-->` նշանները, որպես երկու միմուս և “մեծ է քան” համեմատության նշան:

Ասում են, որ աշխարհում առաջին կազմած ծրագրի կատարման արդյունքում ելքի սարքում ստացվել է հետևյալ արտահայտությունը՝ “Hellow World” (Բարև Աշխարհ): Այն դարձել է պատմական և ցանկացած լեզու ուսումնասիրելիս առաջին ծրագիրը ըստ սովորության կատարում է նույնը: Կազմենք մեր առաջին սցենարի օրինակը, որը դուրս կբերի բրաուզերի էկրանին այդ բարևը: Նպատակը իրականացնելու համար նախ ուսումնասիրենք հետևյալ հրամանը՝

document.writeln():

JavaScript-ում (և առհասարակ օբյեկտակողմնորոշված ծրագրավորման լեզուներում) այդպիսի հրամանները կոչվում են օբյեկտի **մեթոդներ** (տվյալ դեպքում `document` օբյեկտի): Մեթոդը կարելի սահմանել հետևյալ կերպով:

- ❖ **Մեթոդը ֆունկցիա է, որը ներկառուցված է որևէ օբյեկտում և, որի միջոցով օբյեկտը կարող է ավտոմատ կերպով կատարել որոշակի գործողություններ:**

Տվյալ դեպքում `writeln()` մեթոդը թույլ է տալիս տող “գրանցել” (`write line`) “փաստաթուղթ” (`document`) օբյեկտում: Ընդ որում այդ տողը, որը հաղորդվում է մեթոդին որպես արգումենտ, կարող է պարունակել և սովորական տեքստ, և HTML գծանշում: Օգտա-

գործելով այդ մեթոդը կազմենք մեր առաջին սցենարը պարունակող փաստաթուղթը՝

```
<html>
<head><title> Hello World </title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script language="JavaScript">
  <!-- document.writeln("<h1>Hello World!</h1>") // -->
</body>
</html>
```

Այժմ քննարկենք վերը բերված օրինակում հանդիպող որոշ առանձնահատկությունները:

1. <script> կոնտեյները ներդրված է անմիջապես փաստաթղթի մարմնում: Սովորաբար դա արվում է այն դեպքերում, երբ սկրիպտը օգտագործվում է անմիջապես web-էջի մարմնում որևէ բան ստեղծելու համար և պետք չէ նախապես նկարագրել ֆունկցիան <head> բաժնում, քանի որ այն կատարվում է մեկ անգամ՝ փաստաթղթի բեռնման հետ համընթաց և չի պահանջում որևէ հատուկ իրադարձության տեղի ունենալուն: Իհարկե այն դեպքերում, երբ ստեղծվում են բարդ սցենարներ, որոնցում պահանջվում է որոշակի պատահարներին արձագանքող բազմակի անգամ օգտագործվող ֆունկցիաների առկայությունը, վերնագրային մասում տեղադրված և ֆունկցիաների նկարագրությունը պարունակող <script> տեգի գրանցումը դառնում է անհրաժեշտություն (հետագայում մենք կքննարկենք նաև սցենարների կառուցումը առանձին ներդրվող կամ կապակցվող ֆայլերի տեսքով):

2. document.writeln("<h1>Hello World!</h1>") մեթոդի արգումենտը իրենից ներկայացնում է սովորական HTML կոդի տող և կարելի է նկատել, որ </h1> փակող տեգում գրանցված է “\” հակադարձ շեղ գծիկը: Այդ գծիկը ծրագրային կոդում դրվում է հատուկ սիմվոլների առջև, որպեսզի ինտերպրետատորը չհասկանա դրանք որպես կատարվող գործողություն, օրինակ “/n” սիմվոլը հասկացվում է որպես տողադարձ, “/t” – տաբուլյացիայի նշան: Շեղ գծիկից արջև դրված “\” նշանը հուշում է ինտերպրետատորին, որ դրան հաջորդելու է ոչ թե հատուկ իրաման, այլ HTML կոդ կամ սովորական տեքստ:

3. Քանի որ վերնագրային մասում գրանցված <meta> տեգում արդեն սահմանված է սկրիպտերի ծրագրավորման լեզուն՝ <script> տեգում չի օգտագործվել type ատրիբուտը, սակայն հին բրաու-

զերների հետ համատեղելիությունը ապահովելու նպատակով, համեմայն դեպս գրանցված է language="JavaScript" ատրիբուտը:

Անվանելով ֆայլը օրինակ helloworld.html և բացելով այն բրաուզերում կտեսնենք այն ինչ ցուցադրված է պատկեր 3.2.1-ում:



Պատկեր 3.2.1. Առաջին սցենարի աշխատանքի արդյունքը

3.2.2. Ֆունկցիաների ստեղծումը JavaScript-ում

Ֆունկցիան կարելի է ներկայացնել որպես փոքրիկ ծրագիր սկրիպտի ներքո: Ֆունկցիայի նշանակումը կայանում է նրանում, որ այն պետք է կատարի որևէ առաջադրանք և վերադարձնի արդյունքը կանչող ծրագրին: Ֆունկցիայի աշխատանքը սկսվում է հաշվարկները կամ համապատասխան գործողությունները կատարելու համար անհրաժեշտ փոփոխականների արժեքները ստանալուց (ընդ որում պարտադիր չէ արգումենտների տեսքով): Դրանից հետո ֆունկցիան կամ վերադարձնում է ստացված արդյունքը սքրիպտին (եթե դա զուտ հաշվարկներ են), կամ, կատարելով պահանջվող գործողությունները (օրինակ՝ որևէ պատկեր է նկարում էկրանին) ավարտում է աշխատանքը:

Կարելի է իհարկե կազմել սկրիպտը որպես պրոցեդուրաների անընդհատ բլոկ, սակայն ավելի ճիշտ է բաժանել այն երկու առանձին մասերի՝ ֆունկցիաների նկարագրությանների և ֆունկցիաների կանչերի: Այդպիսի մոտեցումը տալիս է երկու անվիճելի առավելություններ:

- ◆ Նախ՝ քանի որ պարտադիր չէ օգտագործել ֆունկցիաները որոշակի հաջորդականությամբ, ապա կարելի է դասավորել դրանց նկարագրությունները կամայական հաջորդականությամբ:
- ◆ Երկրորդ՝ ֆունկցիաները կարող են բազմակի անգամ օգտագործվել պարամետրերի փոփոխված արժեքները մշակելու և,

համապատասխանաբար, նոր արդյունքներ ստանալու համար: Այսինքն միևնույն ֆունկցիան տարբեր սկզբնական տվյալների դեպքում կվերադարձնի տարբեր արդյունքներ:

Բնական է մինչև օգտագործելը ֆունկցիաները պետք է նկարագրվեն (ծրագրավորման “լուրջ” լեզուներում դրանք սկզբում հայտարարվում են և, ապա նկարագրվում): Այն դեպքերում, երբ սկրիպտը ծավալուն չէ դա սովորաբար կատարվում է <head> սեկցիայում, չնայած տեխնիկական տեսակետից դա պարտադիր չէ՝ սկրիպտը կարող է տեղադրվել փաստաթղթի ցանկացած մասում <script> կոնտեյնների ներքո, քանի որ իրականում JavaScript-ը դիտարկվում է բոլոր <script> էլեմենտները որպես մեկ միասնական սկրիպտ:

```
Ֆունկցիայի նկարագրության ֆորմատը հետևյալն է՝
<script type="text/javascript">
<!-- թաքցնում ենք ենք սկրիպտը
function ֆունկցիայի_անունը (ֆորմալ պարամետր[ներ])
{
... ֆունկցիայի ծրագրային կոդը...
return վերադարձվող արժեքը (եթե այն կա)
}
//ավարտում ենք թաքցնել սկրիպտը →
</script>
```

Անհրաժեշտ է հիշել, որ երբ ֆունկցիան կանչվում է, դրան շատ հաճախ փոխանցվում է որոշակի պարամետրի փաստացի արժեքը, որը տեղադրվում է ֆորմալ պարամետրի փոխարեն: Այդ դեպքերում չպետք է մոռանալ անվանակոչել ֆորմալ պարամետրը և փաստորեն ստեղծել ներքին (լոկալ) փոփոխական: Օրինակի համար դիտարկենք ֆունկցիա (անվանենք այն getSumma), որը ստանում է որոշակի արժեք և գումարելով դրան 10 վերադարձնում սկրիպտին: Ֆունկցիայի նկարագրությունը կունենա հետևյալ տեսքը՝

```
<script type="text/javascript">
<!--
function getSumma (myNumber)
{
    summa=myNumber+10;
    return (summa);
}
//→</script>
```

Քննարկենք գրանցված կոդը: Հայտարարվել է getSumma անունով ֆունկցիա: Այն ունի myNumber անունով ֆորմալ պար-

րամետր: Երբ ֆունկցիան կանչվում է, ֆորմալ պարամետրին (myNumber-ին) շնորհվում է փաստացի արժեք, դրանից հետո այդ արժեքին գումարվում է 10, ստացված արդյունքը շնորհվում է summa ներքին փոփոխականին (summa=myNumber+10) և այն վերադարձվում է սկրիպտին՝ return (summa):

Ինչպես տեսնում ենք՝

- ամբողջ հաշվարկային մասը տեղադրվում է ձևավոր փակագծերի միջև (ֆունկցիայի, այսպես կոչված, մարմնում),
- կատարվող գործողությունները (հրամանները) իրարից առանձնացվում են կետ-ստորակետերով “,”;
- արդյունքը վերադարձվում է return օպերատորի միջոցով, որը միաժամանակ ազդարարում է ֆունկցիայի աշխատանքի ավարտը (այն դեպքերում, երբ արժեք չի վերադարձվում return օպերատորը չի գրանցվում):

Որպեսզի ֆունկցիան սկսի աշխատել անհրաժեշտ է կազմակերպել դրա կանչը և փոխանցել ֆորմալ պարամետրին (եթե այն առկա է) փաստացի արժեք: Կանչը կարող է տեղադրվել կամ <script> կոնտեյնների ներսում (ինչպես վերը բերված “Hello World” օրինակում), կամ որպես պատահարների մշակիչը որոշող ատրիբուտի արժեք, կամ URL հասցեում:

Օրինակ՝ եթե փաստաթղթի որևէ էլեմենտը սահմանող տեգում գրանցվի հետևյալ կանչը՝ mySumma(5) (օրինակ՝ <p onclick="mySumma(5)">), դա նշանակում է, որ երբ օգտվողը կտտացնի մկնիկով այդ էլեմենտի վրա (կատարվում է onclick պատահարը), ապա անմիջապես կանչվում է mySumma ֆունկցիան, ընդ որում myNumber ֆորմալ պարամետրին փոխանցվում է փաստացի “5” արժեքը:

- Եթե ֆունկցիան ունի ֆորմալ պարամետր, ապա կանչելիս դրան պարտադիր կերպով պետք է փոխանցել փաստացի արժեք, ընդ որում արժեքների տեսակները պետք է լինեն համատեղելի: Օրինակ եթե փախանցենք մաթեմատիկական հաշվարկներ կատարող ֆունկցիային որևէ սիմվոլային տող, ապա դժվար թե կարողանանք արդյունքում ստանալ որևէ օգտակար ինֆորմացիա:

Օգտագործելով ձեռք բերված գիտելիքները կազմենք թվի քառակուսին հաշվող ֆունկցիայի կանչի օրինակ՝

```
<html>
```

```
<head><title>Թվի քառակուսու հաշվարկ</title>
```

```
<meta http-equiv="Content-Script-Type" content="text/javascript" />
```

```
<script type="text/javascript">
```

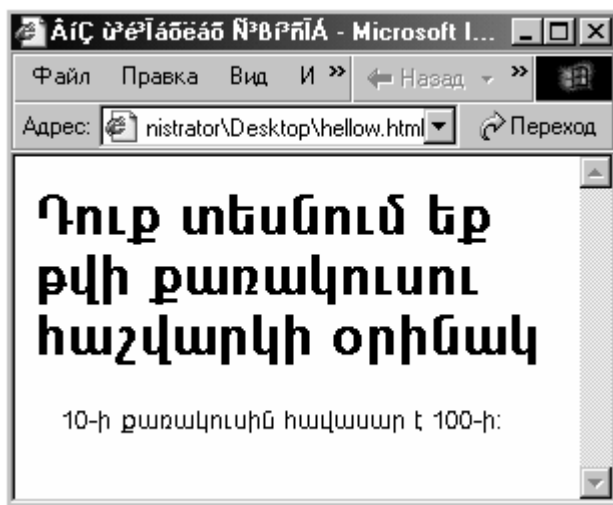
```
<!--
```

```

function getSquare (number)
{ square=number * number;
return (square); }
//→
</script>
</head>
<body>
<h1>Դուք տեսնում եք թվի քառակուսու հաշվարկի օրինակ</h1>
<script language="JavaScript">
<!--
myNum=10;
Sqr=getSquare(myNum);
document.writeln( myNum+" թվի քառակուսին հավասար է "+Sqr);
// -->
</script>
</body>
</html>

```

Ծրագրի աշխատանքի արդյունքը ցուցադրված է պատկեր 3.2.2-ում:



Պատկեր 3.2.2. Ֆունկցիայի կանչը անմիջապես սկրիպտից կազմակերպելու օրինակ
3.2.2. Գործողություններ փոփոխականների հետ

Սկրիպտերում օգտագործվում են տվյալների երկու հիմնական տեսակներ՝

- ♦ լիտերալներ,
- ♦ փոփոխականներ:

Լիտերալների արժեքները հաստատուն են, այսինքն չեն կարող փոփոխվել, լինի դա 5, “Բարև ձեզ” կամ 6,02: Դրանք կարելի է շնորհել փոփոխականներին կամ օգտագործել ինքնուրույն՝ հաշվարկների ընթացքում:

Փոփոխականների արժեքները կարող են փոփոխվել (դա երևում է հենց անվանումից): Սցենարներում յուրաքանչյուր փոփոխական սահմանվում է եզակի իդենտիֆիկատորի միջոցով, որն ավելի հաճախ կոչվում է **փոփոխականի անուն**: Անունը պետք է սկսվի լատինական այբուբենի տառով կամ էլ ընդգծման սիմվոլով: Դրանց կարող են հաջորդել մեծատառեր, փոքրատառեր, թվանշաններ կամ ընդգծման նշաններ: **Անուններում արգելվում է օգտագործել պրոբելի (բացակի) նշանը**: JavaScript-ում անունները կախում ունեն ռեգիստրից, օրինակ myVar և MyVar անունները չեն ընկալվի որպես միևնույն փոփոխական: Ընդունված է շնորհել փոփոխականներին իմաստալից անուններ և, եթե դրանք բաղկացած են մի քանի բառերից, ապա ամջատել վերջիները ընդգծման նշաններով, կամ սկսել ամեն հաջորդ բառը մեծատառով: Օրինակ՝ myNumber, my_number, TotalSales, total_sales և այլն: Կարելի է իհարկե շնորհել մեկ տառից բաղկացած անուններ (հասկանալի է, որ դա պետք է լինի տառ)՝ այդպիսի անունները շնորհվում են սովորաբար ծառայողական բնույթ կրող փոփոխականներին, որոնց նշանակումը հետագայում պետք չէ հիշել ծրագրի մեջ փոփոխություններ կատարելու դեպքում:

JavaScript-ում օգտագործվում են չորս տիպի փոփոխականներ:

- ♦ **Ամբողջ թվեր** – կարող են լինել բացասական և դրական, օրինակ՝ n=5, p=20, m= -18, c=123456 և այլն:
- ♦ **Սահող ստորակետով թվեր** – տասական կոտորակներ են, օրինակ՝ pi=3,14, k=5,21 և այլն:
- ♦ **Սիմվոլային տողեր** – հատուկ տեսակի փոփոխականներ են, որոնց արժեքները չակերտների մեջ վերցված սիմվոլների հաջորդականություններ են: Դրանք կարող են լինել բառեր, նախադասություններ, տառաթվային տողեր, թվային տողեր: Օրինակ՝ city=“Երևան”, zip=“375012”, congr=“Շնորհավորանքներ 50-ամյակի կապակցությամբ” և այլն: Սիմվոլային տողերը ի տարբերություն այլ փոփոխականների չեն մասնակցում թվաբանական գործողություններում, չնայած այն դեպքերում,

երբ սիմվոլային տողը պարունակում է թվանշաններ՝, այն կարելի է ներկայացնել թվի տեսքով:

- ♦ **Բուլյան (տրամաբանական) մեծություններ** - այս տիպի փոփոխականները կարող են ընդունել ընդամենը երկու արժեքներ՝ **true** և **false** (ճշմարիտ է և կեղծ է):

Փոփոխականները և լիտերալները կարող են մասնակցել տարբեր հանրահաշվական գործողություններում (թվաբանական գործողությունների օպերատորները բերված են աղյուսակ 3.2.1-ում): Օրինակ՝ `myVar+2, x*5, 12%7`, “Բարև ”+”Ձեզ”: Սակայն օրինակում բերված հաշվարկները անհմաստ են, եթե դրանց արդյունքները չշնորհվեն փոփոխականներին որպես արժեքներ հետագա օգտագործման նպատակով: Արժեքները (այդ թվում նաև գործողությունների արդյունքները) շնորհվում են փոփոխականներին “=” օպերատորի միջոցով: Օրինակ՝ `newVar=myVar - 4, congr=`”Շնորհավորում ենք”, `x=x+1`:

- JavaScript լեզվում ցանկացած հրամանից հետո պետք է դրվի կետ-ստորակետ “;” նշանը:

Աղյուսակ 3.2.1.

Թվաբանական գործողությունների օպերատորները

Գործողությունը	Անվանումը
+	Գումարում
-	Հանում
*	Բազմապատկում
/	Բաժանում
%	Ամբողջ թվերի բաժանման մնացորդը
++ (ինկրեմենտ)	Արժեքի մեծացում մեկով
-- (դեկրեմենտ)	Արժեքի փոքրացում մեկով

JavaScript-ում հնարավոր է նաև պարզապես հայտարարել փոփոխականները անմիջապես չշնորհելով դրանց արժեքներ: Դա հարմար է այն տեսակետից, որ համակարգչի հիշողությունում նախապես տեղ է հատկացվում այդպիսի փոփոխականների համար: Հայտարարությունը կատարվում է “**var**” (**variable**՝ փոփոխական) բանալիական բառի օգնությամբ: Օրինակ՝ `var x; var myVariable;` Իհարկե կարելի է նաև փոփոխականը հայտարարելիս միանգամից շնորհել դրան արժեք: Օրինակ՝ `var y=“375001”; var milAge=1000; var color=“Գորշ մետալիկ”;` Այն փոփոխականները, որոնց շնորհված է արժեք կարող են մասնակցել նոր գործողություններում:

Սկրիպտերում առավել տարածված գործողություններից են ինկրեմենտը (ավելացնել մեկով) և դեկրեմենտը (պակասեցնել

մեկով): Առավել հաճախ դրանք օգտագործվում են այն դեպքերում, երբ պետք է հաշվել քանի անգամ է կատարվել որևէ պատահարը կամ պարբերական գործողությունը: Պարզագույն ձևով դա կարելի է իրագործել գումարման կամ հանման գործողությունների օգնությամբ՝ $x=x+1$; $y=y-1$: Գումարման և հանման գործողությունները պատկանում են այսպես կոչված **“քինար”** (երկտեղանի) գործողությունների խմբին, քանի որ պահանջում են երկու մասնակիցների (**օպերանդների**) առկայություն:

JavaScript-ը թույլ է տալիս միևնույն նպատակը իրականացնել **“ուճար”** (մեկ օպերանդ պահանջող) գործողությունների միջոցով՝ ինկրեմենտի ($++$) և դեկրեմենտի ($--$): Օրինակ՝ $x++$ գործողության արդյունքը նույնն է ինչ և $x=x+1$ գործողությանը: Ինկրեմենտը և դեկրեմենտը ունեն երկու տարատեսակներ՝ $x++$, $++x$, $y--$, $--y$:

Տարբերությունը բացատրելու համար բերենք օրինակ: Դիցու՛ք $x=2$: Եթե այժմ ինկրեմենտի արդյունքը շնորհից y փոփոխականին հերկյալ կերպով՝ $y=x++$; , ապա y -ին կշնորհվի 2 արժեքը, իսկ x -ի արժեքը կաճի մեկով և կդառնա հավասար 3-ի, քանի որ սկզբում կատարվում է x -ի արժեքի շնորհումը y -ին և դրանից հետո ինկրեմենտը: Իսկ հետևյալ գրանցման դեպքում՝ $y=++x$ և y -ին, և x -ին կշնորհվի 3 արժեքը, քանի որ սկզբում կկատարվի ինկրեմենտի գործողությունը, ապա նոր շնորհման:

Գոյություն ունի շնորհման և թվաբանական գործողությունների միաժամանակյա գրանցման այլընտրանքային (ալտերնատիվ) տարբերակը, որը ներկայացված է աղյուսակ 3.2.2-ում:

Աղյուսակ 3.2.2.

Շնորհման գորշողության գրանցման կրճատ ձևերը

Օպերատոր	Համարժեք գործողությունը
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x \% = y$	$x = x \% y$

3.2.3. Այգորիթմների կատարման ընթացքի ղեկավարումը

Այսպիսով մենք արդեն քննարկեցինք JavaScript-ին վերաբերվող հիմնական հասկացությունները՝ փոփոխականները, ֆունկցիաները, մաթեմատիկական գործողությունները և շնորհման գործողությունը: Հաջորդ քայլը՝ այն արտահայտությունների ուսումնասիրությունն է, որոնց միջոցով հնարավոր է ստեղծել ղեկավարող կառուցվածքներ: Ծրագրավորման հետ ծանոթ մարդկանց

համար պարզ է, որ խոսքը գնում է ճյուղավորվող (if...else) և ցիկլային (այսինքն բազմակի անգամ նույնատիպի գործողությունների կատարումը ապահովող՝ for, while, break, continue) կառուցվածքների մասին:

Այդպիսի կառուցվածքների համար հիմնական հասկացություն է հանդիսանում **պայմանը**: Դա ծրագրային կոդի որոշակի մաս է, որի միջոցով որոշվում է գործողությունների կամ հաշվարկների հետագա ընթացքը: Պայմանի տրամաբանությունը կարող է լինել հետևյալը՝

“Եթե մեր ենթադրությունը ճշմարիտ է, ապա կատարենք որոշակի գործողություն (գործողություններ)”:

Պայմանը կարող է ունենալ նաև այսպիսի ձևակերպում՝

“Եթե մեր ենթադրությունը ճշմարիտ է, ապա կատարենք որոշակի գործողություն (գործողություններ), եթե կեղծ է, ապա այլ գործողություն (ներ)”, կամ

“Եթե մեր ենթադրությունը կեղծ է (ճշմարիտ չէ), ապա կատարենք որոշակի գործողություն (ներ), որը (ոնք) վերջնական արդյունքում կդարձնեն այն ճշմարիտ”:

Հաճախ որպես պայման օգտագործվում է որոշակի արժեքների համեմատությունը՝ ստուգվում է հավասար են դրանք թե ոչ և, եթե ոչ, ապա ո՞րն է մեծ կամ փոքր: Այդպիսի պայմանը կարող է ունենալ հետևյալ ձևակերպումը՝ “Եթե x-ը մեծ է y-ից, ապա կատարենք հետևյալ հաշվարկը”: Համեմատության օպերատորների ցուցակը բերված է աղյուսակ 3.2.3-ում:

Աղյուսակ 3.2.3.

Համեմատության օպերատորները

Օպերատոր	Իմաստը	Օրինակ	“Ճշմարիտ” արժեքը վերադարձնելու պայմանը
==	հավասար է	$x==y$	x-ը հավասար է y-ին
!=	հավասար չէ	$x!=y$	x-ը հավասար չէ y-ին
>	մեծ է	$x>y$	x-ը մեծ է y-ից
<	փոքր է	$x<y$	x-ը փոքր է y-ից
>=	մեծ է կամ հավասար	$x>=y$	x-ը մեծ կամ հավասար է y-ին
<=	փոքր է կամ հավասար	$x<=y$	x-ը փոքր կամ հավասար է y-ին

Համեմատությունները և առհասարակ պայմանական արտահայտությունները սովորաբար վերցվում են կլոր փակագծերի մեջ: Օրինակ ($x==y$):

- Հավասարության պայմանը կազմելու օպերատորը բաղկացած է երկու == նշաններից, քանի որ մեկ = նշանը ծրագրավորման մեջ օգտագործվում է շնորհման գործողության համար, որի ձախ կողմում գտնվող փոփոխականին կամ արտահայտությանը շնորհվում է աջ կողմի փոփոխականի կամ արտահայտության արժեքը: Օրինակ՝ $x=y$ գրանցումը նշանակում է, որ x -ին շնորհվում է y -ի արժեքը, իսկ $(x==y)$ -ը պայմանական արտահայտություն է, որի արժեքը կլինի “ճշմարիտ” (true), եթե x -ը հավասար է y -ին արժեքին և “կեղծ” (false) հակառակ դեպքում:

Պայմանական արտահայտությունների արժեքները կարելի է շնորհել նաև փոփոխականներին և օգտագործել դրանք պայմանները ստուգելու համար: Օրինակ՝ $equiv=(x==y)$: $equiv$ բուլյան փոփոխականի արժեքը կլինի true, եթե կատարվում է պայմանական արտահայտությունը և false հակառակ դեպքում: Հաճախ ամիրաժեշտություն է առաջանում ստուգել մի քանի պայմանական արտահայտությունների արժեքները: Այդպիսի դեպքերում տրամաբանական օպերատորների միջոցով կառուցվում են բաղադրյալ պայմաններ:

Այն դեպքում երբ պետք է ստուգել մի քանի պայմանների միաժամանակյա կատարումը օգտագործվում է && (տրամաբանական “և”), իսկ երբ բավական է մի քանի պայմաններից որևէ մեկի կատարումը || (տրամաբանական “կամ”) օպերատորը: Աղյուսակ 3.2.4-ում բերված են տրամաբանական գործողությունների արդյունքում ստացվող արժեքները օպերանդների (այսինքն գործողությունում մասնակցող բուլյան արտահայտությունների կամ փոփոխականների) տարբեր արժեքների համար:

Աղյուսակ 3.2.4.

Տրամաբանական գործողությունների արդյունքները

A	B	A&&B	A B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Տրամաբանական ընտրություն կատարելու համար JavaScript-ում օգտագործվում է if...else կառուցվածքը (շատ հաճախ այն անվանում են ճյուղավորման օպերատոր), որի միջոցով իրականացվում է “կամ...կամ” տեսակի ընտրություն: Պայմանը այստեղ ստուգվում է միայն մեկ անգամ և պետք է պարունակի արտահայտություն կամ փոփոխական, որը կարող է ընդունել true կամ false արժեքներ: Գրանցման քերականությունը հետևյալն է՝

if (պայման) {ծրագրային կոդ}

else {ծրագրային կոդ}:

Բերենք օրինակ՝

if (x==5)

{ document.writeln("x փոփոխականի արժեքը հավասար է 5-ի"); }

else

{ document.writeln("x փոփոխականի արժեքը հավասար չէ 5-ի"); }

else մասը չի օգտագործվում այն դեպքերում, երբ անհրաժեշտ է, որ պայմանը չկատարվելու պարագայում, ծրագիրը անտեսի if կառուցվածքում պարունակվող ծրագրային կոդի կատարումը և պարզապես շարունակի ալգորիթմի կատարումը: Օրինակ՝

if (total==5) { return true; }

3.2.4. Ցիկլերի կազմակերպումը

Ցիկլեր կազմակերպելու համար օգտագործվում են կառուցվածքների երկու տեսակներ՝ **for** և **while**: Ցիկլը ծրագրային կոդի այնպիսի մասն է, որի կատարումը կրկնվում է այնքան ժամանակ մինչև չկատարվի ցիկլից դուրս գալու պայմանը:

Այն դեպքերում երբ նախապես հայտնի է կամ որևէ եղանակով ստացվել է կրկնողությունների թիվը օգտագործում են **for** կառուցվածքը: Դրա ներքո ստեղծվում է փոփոխական, որի միջոցով հաշվարկվում է կրկնությունների քանակը (այսպես կոչված **հաշվիչը**), որին տրվում է սկզբնական արժեք: Ցիկլի ամեն կատարելիս հաշվիչի արժեքը ավելացվում է (կամ պակասեցվում): Երբ այն հասնում է անհրաժեշտ թվին ցիկլի կատարումը ավարտվում է: **for**-ի գրանցման քերականությունը հետևյալն է՝

for (հաշվիչի սկզբնական արժեքի շնորհում; պայման; հաշվիչի փոփոխության եղանակը)

{ JavaScript-ի հրամանները }

Հասկանալու համար կազմենք օրինակ և քննարկենք **for**-ի կատարման հաջորդականությունը:

```
<script type="text/javascript" language="JavaScript">
```

```
for (n=0;n<10;n=n+1)
```

```
{ result=2*n;
```

```
document.writeln("Եթե 2-ը բազմապատկենք "+n+"-ով ապա կստանանք "+result+"<br />"); }
```

```
</script>
```

Օպերատորի աշխատանքի հաջորդականությունը հետևյալն է՝

1. Ցիկլի սկզբում հաշվիչին շնորհվում է սկզբնական արժեք (հաշվիչը ինիցիալիզացվում է):

2. Ստուգվում է հաշվիչի արժեքի հավասարությունը սահմանային արժեքին, որի դեպքում ցիկլի կատարումը պետք է ավարտվի:

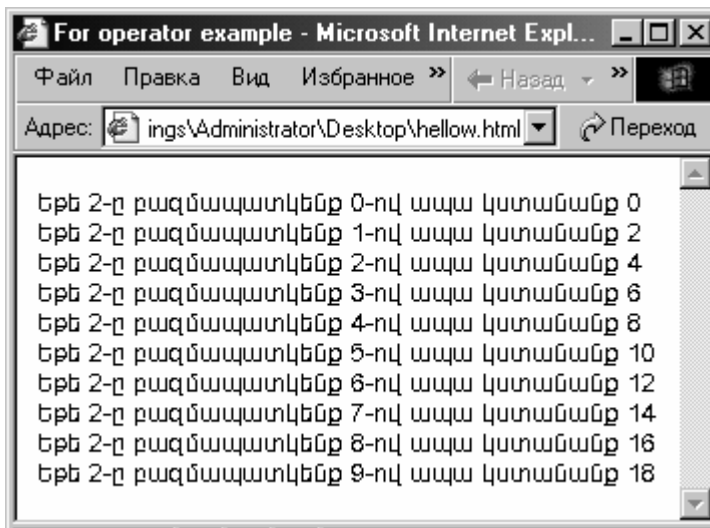
Բերված օրինակում այն հավասար է 9-ի, քանի որ պահանջվում է, որ n -ը լինի 10-ից խիստ փոքր: Եթե հավասարությունը կատարվում է, ապա ցիկլը ավարտվում է:

3. Եթե հաշվիչի արժեքը դեռ սահմանայինին չի հասել կատարվում են բոլոր հրամանները, որոնք գրանցված են ձևավոր փակագծերի մեջ (ցիկլի մարմինը): Մեր օրինակում սկզբում հաշվարկվում է n հաշվիչի արժեքի և 2-ի արտադրյալը և, ապա `document.writeln()` մեթոդի օգնությամբ ընթացիկ հաշվարկի արդյունքը արտապատկերվում է էկրանին:

4. Հաշվիչի արժեքը փոփոխվում է ըստ `for`-ի երրորդ պարամետրում տրված կանոնի: Մեր դեպքում արժեքը ավելացվում է 1-ով ($n=n+1$):

5. Ցիկլի մարմնի բոլոր հրամանները կատարելուց հետո կատարվում է անցում կետ 2-ին:

Տեղադրելով վերը բերված սկրիպտը շաբլոնի `<body>` տեգի ներսում և կազմելով HTML փաստաթուղթ արդյունքում կտեսնենք մոտավորապես այն, ինչ ցուցադրված է պատկեր 3.2.3-ում:



Պատկեր 3.2.3. `for` կառուցվածքի կիրառության օրինակ

Ինչպես կարելի է տեսնել, տողը կազմելիս օգտագործվում է քվաբանական գումարման (+) օպերատորը: Տվյալ օրինակում դրա միջոցով կատարվում է ոչ թե թվերի արժեքների գումարի հաշվարկը, այլ պարզապես իրար են կցվում էկրանին դուրս բերվող

սիմվոլային տողի առանձին մասերը, ընդ որում եթե դա լիտերալ է, ապա պառփակվում է չակետներով, իսկ եթե փոփոխականի արժեք է պարզապես գրանցվում է փոփոխականի անունը (բերված օրինակում դրանք են՝ `n` և `result` փոփոխականները):

Շատ դեպքերում ցիկլեր կազմակերպելու համար օգտագործվում է `while` կառուցվածքը, որի գրանցման քերականությունը հետևյալն է՝

while (պայման)

{ JavaScript-ի հրամանները }

Օպերատորի մարմնում գրանցված ցիկլը կրկնվում է այնքան անգամ, ինչքան “ճշմարիտ” է մնում գրանցված պայմանը: Պետք է չափազանց ուշադիր լինել պայմանը ձևակերպելու ժամանակ և չմոռանալ նաև օպերատորի մարմնում գրանցված կոդում ապահովել պայմանի խախտման նախադրյալները, քանի որ հակառակ դեպքում ցիկլը կկատարվի անվերջ: Օրինակ, եթե մենք գրանցենք՝

```
x=0;
```

```
while (x=6)
```

```
{ x=x+1;
```

```
  document.writeln("x-ի արժեքը հավասար է "+x+"-ի<br />");
```

```
},
```

ապա ցիկլը կկատարվի անվերջ և էկրանին անըդիատ կգրանցվի “x-ի արժեքը հավասար է 6-ի”, քանի որ պայմանում գրանցված է ոչ թե համեմատություն, այլ շնորհում, իսկ ինչպես կարելի է կռահել՝ շնորհումը միշտ ճշմարիտ է:

Նման արդյունք (այսինքն ցիկլի անվերջ կրկնություն) կստացվի նաև հետևյալ օրինակում՝

```
x=0;
```

```
while (x<=6)
```

```
{ document.writeln("x-ի արժեքը հավասար է "+x+"-ի<br />"); }
```

Կտարբերվի միայն արտապատկերվող տողը՝ “x-ի արժեքը հավասար է 0-ի”: Պատճառն այն է, որ ցիկլի մարմնում չի նախատեսվում հաշվիչի (`x`-ի) արժեքի փոփոխությունը: Ամեն ինչ կկարգավորվի, եթե նույն կոդը գրանցվի հետևյալ տեսքով՝

```
x=0;
```

```
while (x<=6)
```

```
{ x=x+1;
```

```
  document.writeln("x-ի արժեքը հավասար է "+x+"<br />"); }
```

Այս դեպքում ցիկլը կկատարվի մինչև `x`-ի արժեքը չգերազանցի 6:

JavaScript-ում (ինչպես նաև շատ ուրիշ լեզուներում) գոյություն ունեն բանալիական բառեր, որոնց միջոցով հնարավոր է փոփոխել

ցիկլի կատարման ընթացքը որոշակի պայմանների կատարման դեպքում: Դրանք են **break** և **continue** բառերը:

break-ը թույլ է տալիս ընդհատել (ավարտել) ցիկլի կատարումը: Օրինակ՝

```
for (n=0;n<10;n=n+1)
{
    z=getInput();
    if (z==n) break;
}
```

Վերը բերված օրինակում ցիկլի սկզբում կանչվում է `getInput()` ֆունկցիան, որի վերադարձվող արժեքը (այն կարող է, օրինակ, լինել որոշակի հաշվարկի արդյունք կամ օգտվողի կողմից ներմուծված թիվ) շնորհիվում է `z` փոփոխականին և, եթե `z`-ի և `n`-ի արժեքները լինեն հավասար ցիկլի կատարումը ավարտվում է `break`-ի միջոցով: Հակառակ դեպքում ցիկլը կկատարվի մինչև `n`-ը չգերազանցի 9:

continue-ն կիրառվում է այն դեպքերում, երբ անհրաժեշտ է շրջանցել ցիկլի կատարումը (այսինքն չկատարել ցիկլը) հաշվիչի որոշ արժեքների համար: Օրինակ՝

```
x=0;
while (x<10)
{
    x=x+1;
    if (x==5) continue;
    document.writeln("x հավասար չէ 5-ի<br \>");
}
```

Այստեղ (`x==5`) պայմանը կատարվելու դեպքում `continue`-ն ստիպում է անտեսել բոլոր ստորև գրանցված հրամանները և անմիջապես վերադառնալ `while` օպերատորի պայմանի ստուգմանը:

3.2.5. Չանգվածները

Չանգվածը տվյալների այնպիսի տեսակ է, որը թույլ է տալիս մեկ անվան տակ պահպանել մի քանի փոփոխականներ, որոնք կոչվում են զանգվածի անդամներ: Միմյանցից տարբերելու նպատակով զանգվածի անդամները համարակալվում են (ինդեքսավորվում): Չանգվածի ներքո յուրաքանչյուր փոփոխականին շնորհվում է հաջորդական համար: Չանգվածը հայտարարվում է հետևյալ եղանակով՝ **var զանգվածի_անուն = new Array();** Այն դեպքերում, երբ զանգվածի փոփոխականների քանակը շատ չէ, կարելի է մաս հայտարարելիս դրանց անմիջապես շնորհել արժեքներ, օրինակ՝

```
var employee=new Array("Սիմոնյան","Պետրոսյան","Սարգսյան",  
"Պողոսյան");
```

Օրինակում հայտարարված է չորս անդամներից բաղկացած employee անունով զանգված: Հասանելիությունը զանգվածի անդամներին իրականացվում է հետևյալ կերպով՝ զանգվածի անունից անմիջապես հետո քառակուսի փակագծերում գրանցվում է համապատասխան փոփոխականի հաջորդական համարը: Համարակալումը սկսվում է 0-ից, այսինքն զանգվածի առաջին անդամին շնորհվում է 0 համարը, երկրորդին՝ 1 և այլն: Բերված օրինակում employee[0]="Սիմոնյան", employee[1]="Պետրոսյան" և այլն:

Ինչպես և բոլոր օբյեկտները զանգվածներն ունեն հատկություններ և մեթոդներ: **length** (երկարություն) հատկությունը թույլ է տալիս որոշել զանգվածի անդամների քանակը: Օրինակ՝ վերը բերված օրինակի զանգվածի անդամների քանակը՝

employee.length, հավասար կլինի չորսի:

Երբ զանգվածը ստեղծված է, կարելի է դրան ավելացնել նոր անդամներ կամ փոխել առկաների արժեքները: Օրինակ՝
employee[4]="Սահակյան"; //ավելացվում է ևս մեկ, հինգերորդ (համար 4) անդամը,
employee[2]="Փափազյան"; // երրորդ (համար 2-ուս) անդամի արժեքը փոխվում է՝ "Սարգսյան"-ի փոխարեն դառնում է հավասար "Փափազյան":

Ցիկլերի օգտագործումը թույլ է տալիս դյուրին կերպով աշխատել զանգվածների հետ՝ տեսակավորել, շնորհել նոր և փոփոխել առկա արժեքները, ապահովել հասանելիությունը և այլն: Բերենք զանգվածների հետ աշխատելու համար ցիկլերի օգտագործման օրինակ:

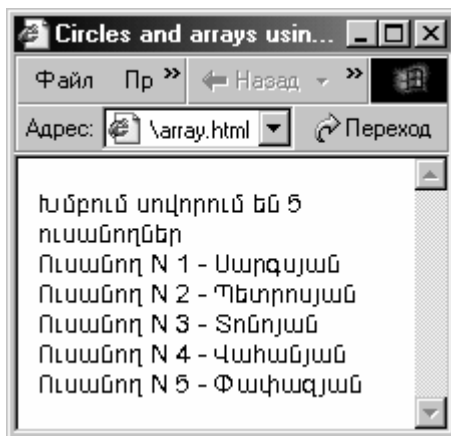
```
<html>  
<head><title>Circles and arrays using example</title>  
<meta http-equiv="Content-Script-Type" content="text/javascript" />  
</head>  
<body>  
<script language="JavaScript">  
<!--  
student = new Array ("Սարգսյան","Պետրոսյան","Տոնոյան", "Վահանյան",  
"Փափազյան");  
numStudents =(student.length);  
document.writeln("Խմբում սովորում են " + numStudents + " ուսանողներ  
<br />");  
var x=0;
```

```

while (x<numStudents)
{ document.writeln ("Ուսանող N "+(x+1)+" - "+student[x]+"<br V>");
x++; }
// -->
</script>
</body></html>

```

Ծրագրի կատարման արդյունքը ցուցադրված է պատկեր 3.2.4-ում:



Պատկեր 3.2.4. Զանգվածների և ցիկլերի համատեղ օգտագործման օրինակ

§3.3. JavaScript լեզվի օբյեկտները

JavaScript-ը, ինչպես և այլ ժամանակակից ծրագրավորման լեզուները օբյեկտակողմնորոշված լեզու է: Դա նշանակում է, որ այն հնարավորություն է ընձեռում ստեղծել և օգտագործել միանման հատկություններով օժտված օբյեկտների անվանակոչված համախմբեր, որոնք ներառում են բնութագրող փոփոխականներ և ներկառուցված ֆունկցիաներ՝ այսպես կոչված **հատկություններ և մեթոդներ** (սովորաբար այդպիսի համախմբերը անվանում են դասեր, որոնց յուրաքանչյուր կոնկրետ հատկություններ ունեցող նմուշը կոչվում է օբյեկտ):

Օգտագործելով գրանցման հատուկ ձև կարելի է դիմել օբյեկտի հատկություններին և մեթոդներին: Այն հետևյալն է՝

օբյեկտի_անուն.հատկություն կամ

օբյեկտի_անուն.մեթոդ()

Ինչպես կարելի է տեսնել օբյեկտի անունից հետո դրվում է կետ և գրանցվում համապատասխան հատկության կամ մեթոդի անվանումը:

➤ Քանի որ մեթոդը դա ֆունկցիա է, ապա մեթոդի անվանումից հետո դրվում են նաև կլոր փակագծեր (որոնց մեջ անհրաժեշտության դեպքում կարող են գրանցվել նաև արգումենտների արժեքները):

Մենք արդեն դիտարկել ենք document օբյեկտի writeln() մեթոդին դիմելու օրինակներ: HTML փաստաթուղթը (document) մեկ խոշոր օբյեկտ է: Այն ունի բազմաթիվ հատկություններ, որոնք հնարավոր է փոփոխել և հենց այդ հատկությունների ծրագրային փոփոխությունն հնարավորությունն էլ JavaScript-ի գլխավոր առավելություններից մեկն է:

3.3.1. Սեփական օբյեկտների ստեղծումը JavaScript-ում

Օբյեկտներ ստեղծելու համար օգտագործվում է new օպերատորը, որը, սովորաբար, կիրառվում է օբյեկտների նոր նմուշներ ստեղծելու և դրանց համար համակարգչի հիշողությունում դիմամիկ կերպով տեղ հատկացնելու համար: Օբյեկտի նոր նմուշի ստեղծման քերականությունը հետևյալն է՝

անուն = new կոնստրուկտորի_անվանումը ([արգումենտները])

Այն դեպքերում, երբ օբյեկտի տեսակը հայտնի է և մշակված (օրինակ՝ ներկառուցված օբյեկտների դեպքում), որպես կոնստրուկտոր օգտագործվում է այդ տեսակի անվանումը, օրինակ՝ myDate=new Date("Jan 15 2005"); // ստեղծվում է հայտնի դատա օբյեկտի նոր նմուշ:

JavaScript-ը, ինչպես և այլ օբյեկտակողմնորոշված լեզուները, թույլ է տալիս նաև ստեղծել սեփական տիպի օբյեկտներ, օժտել դրանք համապատասխան հատկություններով ու մեթոդներով և օգտագործել տվյալների պահպանման համար: Այդ նպատակը իրականացնելու համար սկզբում անհրաժեշտ է ստեղծել պահանջվող տեսակի օբյեկտների կառուցման **կոնստրուկտորի շաբլոնը**, որին տրվում է օբյեկտի բնույթին համապատասխանող որևէ անուն: Շաբլոնը իրենից ներկայացնում է ընտրված անունով ֆունկցիա, որի ներքո **this** բանալիական բառի միջոցով գրանցվում են օբյեկտի բոլոր հատկություններին արժեքների շնորհման հրամանները:

Օրինակ անշարժ գույքի (բնակարանների) վերաբերյալ տվյալներ պահպանելու նպատակով կարելի է ստեղծել home անունով օբյեկտի կոնստրուկտորի շաբլոն՝

function home(price,sqmeter,rooms,description)

```
{ this.price=price;
  this.sqmeter=sqmeter;
  this.rooms=rooms;
  this.description=description;
}
```

Ինչպես տեսնում ենք կոնստրուկտորը դա ֆունկցիա է, որն ընդունում է բնակարանի հատկությունները բնորոշող չորս ֆորմալ պարամետրեր (արգումենտներ)՝ price (գինը), sqmeter (մակերեսը), rooms (սենյակների քանակը) և description (նկարագրությունը): Ֆունկցիան շնորհում է home տիպի կոնկրետ օբյեկտին (նմուշին) նշված հատկությունների համապատասխան արժեքները:

➤ Ուշադրություն դարձրեք this բանալիական բառին: JavaScript-ում այն օգտագործվում է որպես տվյալ օբյեկտի “ցուցանակ” (այսինքն դրա միջոցով ծրագրային այլ բլոկներին հաղորդվում են օբյեկտի տեսակը ճանաչելու համար անհրաժեշտ տվյալները):

Կոնստրուկտորը կառուցելուց հետո կարելի է արդեն ստեղծել օբյեկտի տվյալ տեսակի կոնկրետ հատկություններով օժտված նմուշներ: Օրինակ, եթե մենք ցանկանում ենք պահպանել տեղեկություններ բնակարանի վերաբերյալ, որի գինը 25000 դոլար է, մակերեսը 80 ք.մ., սենյակների քանակը հավասար է 2-ի և բնակարանը կապիտալ վերանորոգված է, շուրջօրյա ապահովված է ջրով: Դա նույնպես կատարվում է **new** օպերատորի օգնությամբ՝ **home1=new home(25000, 80, 2, "Կապիտալ վերանորոգված է, ջուրը շուրջօրյա")**;

Ինչպես կարելի է տեսնել օբյեկտի նմուշի ստեղծման համար home ֆունկցիայի (կոնստրուկտորի) ֆորմալ պարամետրերին պարզապես փոխանցվում են կոնկրետ արժեքներ, որոնք արդեն շնորհվում են home1 օբյեկտին որպես հատկությունների կոնկրետ արժեքներ, այսինքն՝ home1.price=25000; home1.sqmeter=80; home1.rooms=2; home1.description=" Կապիտալ վերանորոգված է, ջուրը շուրջօրյա";

Կարելի է նաև սկզբում ստեղծել օբյեկտը և ապա մոր օժտել դրան կոնկրետ հատկություններով: Օրինակ՝ home2=new home(); home2.price=50000; home2.sqmeter=100; home2.rooms=3; home2.description="Արտակարգ տեղայնք, չտեսնված կրոնեմոնտ";

Իհարկե այնպիսի դեպքերում, երբ կոնստրուկտորի ֆորմալ պարամետրերին չեն հաղորդվում կոնկրետ արժեքներ, պետք է նախատեսել նաև ըստ լռելյայն արժեքների (օրինակ զրոական)

շնորհման հնարավորությունը, կամ կառուցել մաս առանձին՝ ըստ լռելյայն արժեքների շնորհող կոնստրուկտոր:

Հատկանշական է, որ տվյալ տեսակի օբյեկտի առանձին մոուլների հատկությունների ցանկը կարելի է ընդլայնել և դա չի ազդում այլ մոուլների վրա, մուլհիսկ եթե դրանք ստեղծված են միևնույն կոնստրուկտորի օգնությամբ: Օրինակ, եթե անհրաժեշտ է նշել մաս թաղամասը, որտեղ գտնվում է առաջին բնակարանը (home տիպի օբյեկտի home1 մոուլը), ապա դա կարելի է անել հետևյալ եղանակով՝ home1.location="Կենտրոն թաղամաս"; և դա բացարձակապես չի ազդի այդ տիպի մյուս՝ home2 օբյեկտի հատկությունների վրա:

Ինչպես արդեն նշվել է, միաբնույթ օբյեկտների յուրաքանչյուր տեսակը (դասը) կարող է բնութագրվել ոչ միայն հատկություններով (փոփոխականներով), այլ և մեթոդներով: Յուրաքանչյուր մեթոդ ձևակերպվում է որպես առանձին ֆունկցիա: Ստեղծենք օրինակ ֆունկցիա, որը էկրանին դուրս կբերի օբյեկտի բոլոր փոփոխականների (հիշենք՝ հատկությունների) արժեքները՝

```
function ShowListing()
{
    document.writeln("Գինը – "+this.price + " դոլար<br V>");
    document.writeln("Մակերեսը - "+this.sqmeter + "ք.մ.<br V>");
    document.writeln("Սենյակները - "+this.rooms + "<br V>");
    document.writeln("Նկարագրությունը՝ "+this.description + "<br V>");
    return;
}
```

Որպեսզի այդ ֆունկցիան դառնա home տիպի օբյեկտի մեթոդ՝ դրա հայտարարությունը պետք է ավելացնել օբյեկտի կոնստրուկտորում (մույն եղանակով, ինչպես և հատկությունները, մի տարբերությամբ՝ մեթոդին չի շնորհվում արժեք)՝

```
function home(price,sqmeter,rooms,description)
{ this.price=price;
  this.sqmeter=sqmeter;
  this.rooms=rooms;
  this.description=description;
  this. ShowListing= ShowListing;
}
```

Ասվածը պարզաբանելու համար կազմենք փաստաթուղթ (Ծր. 3.3.1.), որի արտապատկերումը ցուցադրված է պատկեր 3.3.1-ում:

Ծր. 3.3.1. Օբյեկտի ստեղծման օրինակ

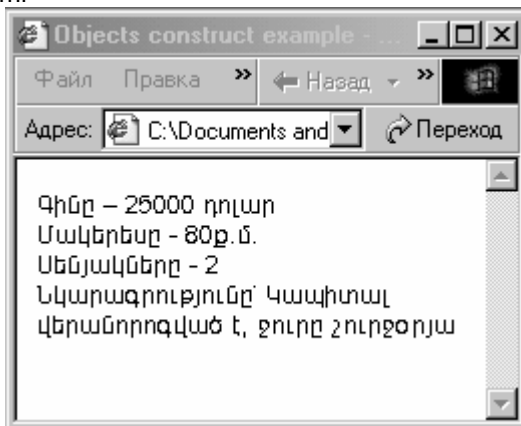
```
<html>
```

```
<head><title>Objects construct example</title>
```

```

<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head><body>
<script language="JavaScript">
<!--
function ShowListing()
{ document.writeln("Գինը – "+this.price +" դոլար<br V>");
  document.writeln("Մակերեսը - "+this.sqmeter +"ք.մ.<br V>");
  document.writeln("Սենյակները - "+this.rooms +"<br V>");
  document.writeln("Նկարագրությունը` "+this.description +"<br V>");
  return; }
function home(price,sqmeter,rooms,description)
{ this.price=price;
  this.sqmeter=sqmeter;
  this.rooms=rooms;
  this.description=description;
  this. ShowListing= ShowListing; }
home1=new home(25000,80,2,"Կապիտալ վերանորոգված է, ջուրը
շուրջօրյա");
home1.ShowListing(); // --> </script>
</body></html>

```



Պատկեր 3.3.1. Օբյեկտի մեթոդի կիրառության օրինակ

3.3.2. JavaScript-ի ներկառուցված օբյեկտները

JavaScript-ում ներկառուցված են երեք տեսակի օբյեկտներ, որոնք թույլ են տալիս սկրիպտեր գրելիս խուսափել բազմաթիվ նմանատիպ գործողությունների նկարագրությունից: Դրանք նշանակված են առավել օգտակար փոփոխականները և հաճախ օգ-

տագործվող մեթոդները պահպանելու և, ստեղծելով հարկ եղած դեպքում համապատասխան տեսակի օբյեկտներ, դրանք օգտագործելու նպատակով:

String օբյեկտը հատկացնում է տողերի (տեքստի) հետ աշխատելու համար անհրաժեշտ փոփոխականները և մեթոդները: Յուրաքանչյուր տող, որը շնորհվում է որևէ փոփոխականին որպես արժեք JavaScript-ում ավտոմատ կերպով դառնում է String տիպի օբյեկտի կոնկրետ նմուշ: Օրինակ եթե գրանցենք՝

str="Այ, հենց այսպիսի մի տող";

ապա str փոփոխականը արդեն իրենից կներկայացնի օբյեկտ և բույլ կտա օգտվել տողային տեսակի օբյեկտների բոլոր համապատասխան մեթոդներից: String օբյեկտի առավել հաճախ կիրառվող մեթոդների ցուցակը բերված է աղյուսակ 3.3.1-ում:

Աղյուսակ 3.3.1.

String օբյեկտի մեթոդները

Մեթոդը	Նկարագրությունը	Օրինակ
bold	Տողը արտապատկերվում է թավ տառաշարով	str.bold()
charAt	Ընտրում է անհրաժեշտ սիմվոլը	str.charAt(2)
fontSize	Փոխում է տառերի է չափսը	str.fontSize(10)
indexOf	Գտնում է տառի հաջորդական համարը տողում	str.indexOf("w")
lastIndexOf	Գտնում է տառի վերջին համարը տողում	str.lastIndexOf("w")
substring	Ընտրում է տողի մի մասը	str.substring(1,7)

Նշենք, որ աղյուսակում բերված բոլոր մեթոդները հանդիսանում են շնորհման գործողության “աջակողմյան” օպերանդ, այսինքն դրանց կատարման արդյունքը պետք է շնորհել որևէ փոփոխականին, որպեսզի հնարավոր լինի հետագայում այն արտապատկերել կամ օգտագործել: Պարզաբանելու համար բերենք օրինակ: Դիցուկ ստեղծում ենք string տիպի փոփոխական՝
string1="Սա փորձնական տող է՝ մեկ, երկու, երեք";

Եթե այժմ մենք ցանկանում ենք արտապատկերել այդ տողը թավ տառաշարով, ապա սկզբում օգտագործում ենք bold() մեթոդը և արդյունքը շնորհում մի այլ, ասենք՝ string2 փոփոխականին և դրանից հետո միայն ապահովում ենք անհրաժեշտ տեսակի արտապատկերումը՝

```
string2=string1.bold();
document.writeln(string2);
```


Տողի սիմվոլների համարակալումը նման է զանգվածների փոփոխականների համարակալմանը՝ սկսվում է 0-ից: Դա պետք է հաշվի առնել այն դեպքերում, երբ օգտագործվում են ինդեքսների հետ կապված մեթոդները՝ charAt(), indexOf(), lastIndexOf(), substring(): Պարզաբանենք դա օրինակի վրա: Դիցուկ myString փոփոխականին շնորհիվ էլ է հետևյալ արժեքը՝
myString="Այս տողը ծառայում է որպես օրինակ";

Ինչպես վերը նշեցինք myString-ը ավտոմատ կերպով դառնում է String տիպի օբյեկտ և, հետևաբար օժտված է դրան հատուկ բոլոր մեթոդներով: Տողում պարունակվում են 32 սիմվոլներ, քանի որ պրոբելը նույնպես համարվում է սիմվոլ: Տողի հաշվով առաջին՝ “Ա” սիմվոլի ինդեքսը հավասար է 0-ի, իսկ վերջին՝ “կ”-ինը՝ 31-ի: Դա նշանակում է, որ եթե օրինակ մենք ցանկանում ենք պարզել, թե ի՞նչ սիմվոլ է սովորական հաշվով 15-րդ սիմվոլը տողում, ապա պետք է գրենք՝ myString.charAt(14), քանի որ ինդեքսային հաշվարկը սկսվում է 0-ից:

Հարկավոր է ուշադիր լինել substring(m,n) մեթոդը կիրառելիս: Դրա միջոցով առանձնացվում է (ինդեքսային հաշվարկով) սկզբնական տողի այն ենթատողը, որն ընդգրկում է սիմվոլները սկսած m-ից մինչև (n-1)-ը՝ ներառյալ: Ստորև բերված ծրագրում ցուցադրված են String օբյեկտի որոշ մեթոդների կիրառության օրինակներ (տես նաև՝ պատկեր 3.3.2.):

Ճր. 3.3.2. String օբյեկտի մեթոդների կիրառության օրինակ

```
<html >
<head><title>String object example</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body marginleft="1">
<script language="JavaScript">
<!--
var myString="Այս տողը ծառայում է որպես օրինակ";
length=myString.length;
charat=myString.charAt(5);
indof=myString.indexOf("ա");
indofh=indof+1;
lindof=myString.lastIndexOf("ա");
lindofh=lindof+1;
substr=myString.substring(4,10);
document.writeln("<b>myString տողը՝ \"/>

```

```

document.writeln("Տողը պարունակում է <b>"+length+"</b> սիմվոլ
\\(որպես սիմվոլ ընդունվում են նաև պրոբելները)<br />");
document.writeln("<b>1. myString.charAt(5) մեթոդի
կիրառությունը</b><br />");
document.writeln("Տողի հաշվով 6-րդ սիմվոլը, որն ունի 5 ինդեքս -
<b>"+charAt+"</b>-ն է<br />");
document.writeln("<b>2. myString.indexOf(\"ա\") մեթոդի
կիրառությունը</b><br />");
document.writeln("<b>ա</b> տառը առաջին անգամ հանդիպում է
տողում - <b>"+indexOf+"</b>-րդ դիրքում
\\(հաշվով<b>"+indexOfh+"</b>-րդում)<br />");
document.writeln("<b>3. myString.lastIndexOf(\"ա\") մեթոդի
կիրառությունը</b><br />");
document.writeln("<b>ա</b> տառը վերջին անգամ հանդիպում է
տողում - <b>"+lindexOf+"</b>-րդ դիրքում \\(հաշվով
<b>"+lindexOf+"</b>-րդում)<br />");
document.writeln("<b>4. myString.substring(4,10) մեթոդի
կիրառությունը</b><br />");
document.writeln("substring(4,10) ենթատողը -
<b>\\\""+substr+\\\"</b> ընդգրկում է սկզբնական տողի սկսած
հաշվով 5-ից մինչև հաշվով 10-դ սիմվոլները: Եթե դա
արտահայտենք ինդեքսներով ապա 4-ից մինչև 9-րդ սիմվոլը
ներառյալ<br />");
// -->
</script>
</body>
</html>

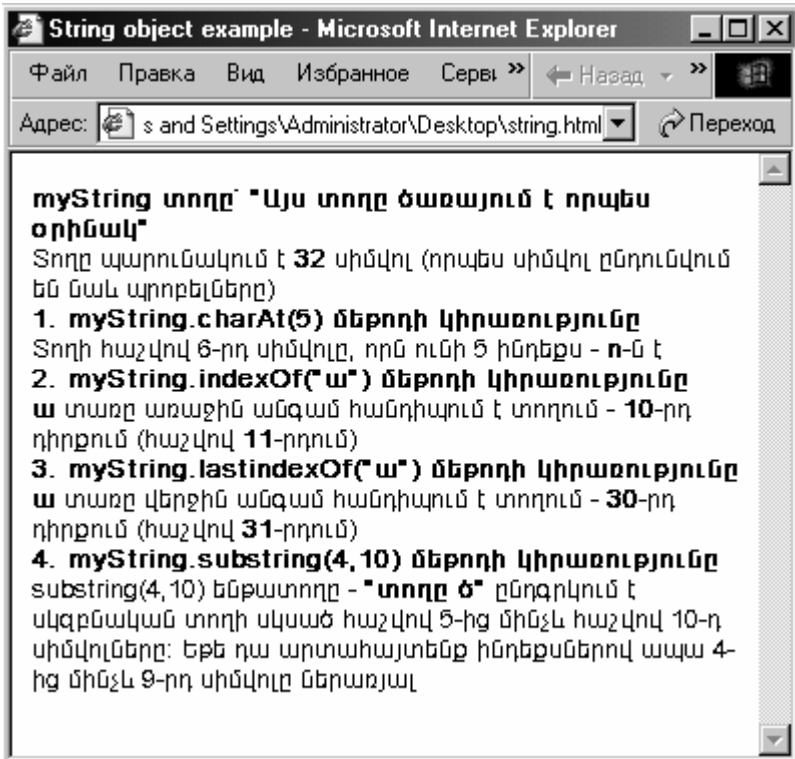
```

Ուշադրություն դարձրեք, որ document.writeln() մեթոդը կիրառելիս, այն սիմվոլների արջև, որոնք պետք է վերծանվեն որպես արտապատկերվող տեքստ կամ HTML գծանշում (օրինակ՝ փակագծերի, չակերտների և շեղ գծերի) դրված է “\” (հակառակ շեղ գիծը), որը, հիշեցնենք, թելադրում է JavaScript-ի ինտերպրետատորին, որ դրան անմիջապես հաջորդող սիմվոլը իրենից ներկայացնում է կամ սովորական տեքստ կամ HTML գծանշում, այլ ոչ ծրագրային ծառայողական սիմվոլ: Օրինակ՝ եթե մենք ուզում ենք արտապատկերել հետևյալ տեքստը՝ հյուլեի “միջուկը” բաղկացած է, ապա պետք է գրանցել՝ **document.writeln(“հյուլեի Վմիջուկը” բաղկացած է”);**

Ինչպես տեսնում ենք չակերտների մեջ գրանցված յուրաքանչյուր չակերտի արջևում դրված է “\” նշանը: Եթե հրամանը գրանցվեր հետևյալ կերպով՝

`document.writeln("հյուլեի "միջուկը" բաղկացած է");`

ապա ինտերպրետատորը լավագույն դեպքում կարտապատկերեր միայն "ատոմի" բառը, քանի որ `writeln()` մեթոդում չակերտների յուրաքանչյուր զույգ առանձնացնում է տեքստ կամ HTML գծանշում: Փոփոխականները գրանցվում են առանց չակերտների, իսկ առանձին "կտորները" (տեքստը, փոփոխականները, HTML կոդը) կցվում են միմյանց գումարման նշանի միջոցով, օրինակ՝ `document.writeln("տեքստ"+փոփոխական+"HTML կոդ տեքստ");`



Պատկեր 3.3.2. String օբյեկտի մեթոդների կիրառության օրինակ

Math օբյեկտն ունի մաթեմատիկական ֆունկցիաների և հաստատուն մեծությունների հետ աշխատելու համար անհրաժեշտ մի շարք հատկություններ և մեթոդներ: Գրանցման եղանակը հետևյալն է՝ **Math.մեթոդ** կամ **Math.հատկություն**: Աղյուսակ 3.3.2-ում բերված են այդ օբյեկտի մեթոդները:

Աղյուսակ 3.3.2.

Math օբյեկտի մեթոդները

Մեթոդը	Վերադարձվող արդյունքը	Գրանցման ձևը
abs	Թվի մոդուլը	Math.abs(թիվ)
acos	arccos	Math.acos(թիվ)
asin	arcsin	Math.asin(թիվ)
atan	arctg	Math.atan(թիվ)
cos	cos	Math.cos(թիվ)
sin	sin	Math.sin(թիվ)
tan	tg	Math.tan(թիվ)
ceil	Մոտակա մեծ ամբողջ թիվը	Math.ceil(թիվ)
floor	Մոտակա փոքր ամբողջ թիվը	Math.floor(թիվ)
exp	Էքսպոնենտը (e-ի աստիճանը)	Math.exp(թիվ)
log	Բնական լոգարիթմը	Math.log(թիվ)
pow	թ1-ի թ2 աստիճանը	Math.pow(թ1,թ2)
max	Երկու թվերից մեծագույնը	Math.max(թ1,թ2)
min	Երկու թվերից փոքրագույնը	Math.min(թ1,թ2)
round	Կլորացված թիվը	Math.round(թիվ)
sqrt	Թվի քառակուսի արմատը	Math.sqrt(թիվ)
random	Պատահական թիվ (0<թիվ<1)	Math.random()

Date օբյեկտը սատարում է ժամանակի և դատաների հետ աշխատելու համար անհրաժեշտ մեթոդները, որոնցից հիմնականները բերված են աղյուսակ 3.3.3-ում: Ընդ որում դրանք թույլ են տալիս, ոչ միայն ստանալ ընթացիկ դատան կամ ժամանակը, այլ և կատարել դրանց հետ մաթեմատիկական գործողություններ: Օբյեկտի մեթոդները օգտագործելու նպատակով ստեղծվում է դրա կոնկրետ նմուշը (դարձյալ new օպերատորի օգնությամբ)՝

todaydate=new Date();

Եթե փակագծերում ոչինչ նշված չէ, ապա օբյեկտին ըստ լռելյայն շնորհվում են ընթացիկ ժամանակը և դատան հետևյալ ֆորմատով՝

օր ամիս ամսաթիվ ժժ:ՌՌ:ՎՎ գոտի տարի:

Բերենք Date օբյեկտի մեթոդների օգտագործման օրինակ (տես՝ օր. 3.3.3. և պատկեր 3.3.3.)

Աղյուսակ 3.3.3.

Date օբյեկտի հիմնական մեթոդները

Մեթոդ	Վերադարձվող արժեքը
getDate()	Ամսաթիվը
getDay()	Շաբաթվա օրը, որպես ամբողջ թիվ (0-ից կիրակի մինչև 6-ը՝ շաբաթ)
getHours()	Ժամը (0-ից մինչև 23-ը)
getMinutes()	Րոպեներ
getMonth()	Ամիսը, որպես ամբողջ թիվ (0-ից մինչև 11-ը)
getSeconds()	Վարկայանները
getTime()	Տարբերությունը ընթացիկ դատայի և 1970 թվի հունվարի 1-ի 00:00:00 (գրինվիչյան ժամանակով) միլիվարկայաններով
getTimeZoneOffset()	Տարբերությունը (րոպեներով) տեղական և գրինվիչյան ժամերի միջև
getYear()	Տարեթիվը, որպես ամբողջ (երկու նիշը)
parse(դատան)	Տարբերությունը փակագծերում նշած դատայի և 1970 թվի հունվարի 1-ի 00:00:00 (գրինվիչյան ժամանակով) միլիվարկայաններով
toGMTString()	Գրինվիչյանի ձևափոխված տեղական ժամանակը
toLocaleString()	Տեղականի ձևափոխված գրինվիչյան ժամանակը

Ծր. 3.3.3. Date օբյեկտի մեթոդների կիրառության օրինակ

```

<html>
<head>
<title>String object example</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body marginleft="1">
<script language="JavaScript">
<!--
var myDate=new Date();
document.writeln("Սահմանենք Date տեսակի օբյեկտ՝ myDate=new
Date\()\<br \>");
document.writeln("Ընթացիկ ժամանակը և դատան այդ օբյեկտում
ստացվում են հետևյալ ձևով՝ "+

```

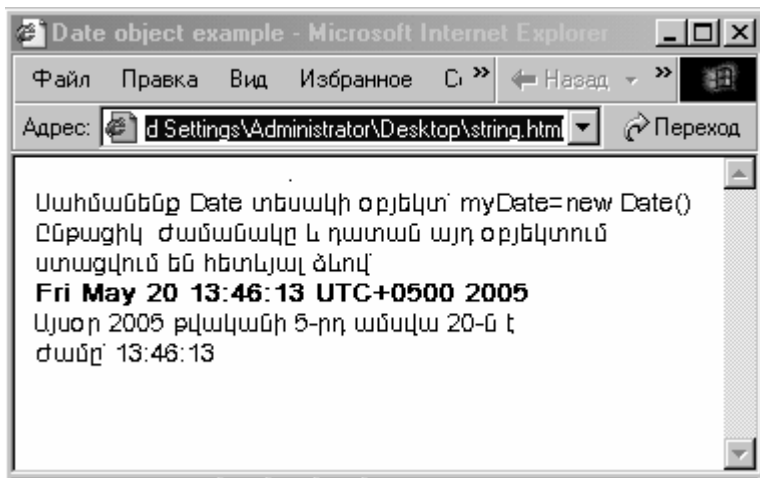
```

"<br V><b>" + myDate + "</b><br />");
myAmsativ = myDate.getDate();
myDay = myDate.getDay();
myMonth = myDate.getMonth();
myMonth = myMonth + 1;
myYear = myDate.getFullYear();
document.writeln("Այսօր " + myYear + " թվականի " + myMonth + "-րդ  

ամսվա " + myAmsativ + "-ն է<br V>");
document.writeln("Ժամը՝ " + myDate.getHours() + ":"  

+ myDate.getMinutes() + ":" + myDate.getSeconds() + "<br V>");
//→
</script>
</body></html>

```



Պատկեր 3.3.3. Date օբյեկտի մեթոդների կիրառության օրինակ

3.3.3. JavaScript-ի ներկառուցված ֆունկցիաները

Բացի ներկառուցված օբյեկտներից JavaScript լեզվում գոյություն ունեն մեթոդներ չհանդիսացող ներկառուցված ֆունկցիաներ, որոնք հնարավոր է օգտագործել առանց օբյեկտներ ստեղծելու: Հիմնականում այդ ֆունկցիաները օգտագործվում են փոփոխականների կամ օբյեկտների տեսակը փոխելու կամ ստուգելու նպատակով:

1. **escape(սիմվոլ)** ֆունկցիան ներկայացնում է արգումենտի արժեքը բոլոր տեսակի համակարկիչներին հասկանալի տաս-

վեցական կոդով հետևյալ տեսքով %XX (լատինական այբուբենի տառերը և թվանշանները վերադարձվում են անփոփոխ):

2. **eval(արտահայտություն)** ֆունկցիան վերադարձնում է արտահայտության գործողությունների կատարման արժեքը, ընդ որում արտահայտության օպերանդները պետք է պարունակեն միայն թվանշաններ: Օրինակ, եթե գրանցվի՝

```
a=25;
```

```
b=33;
```

```
c=eval(a+b);
```

ապա արդյունքում կստացվի 58, քանի որ a և b փոփոխականները թվեր են և գումարվում են որպես թվեր: Իսկ եթե՝

```
a="25";
```

```
b="37";
```

```
c=eval(a+b);
```

ապա արդյունքում կստացվի 2537, քանի որ այս դեպքում a և b փոփոխականները տողեր են, իսկ տողերը գումարման արդյունքում պարզապես կցվում են միմյանց:

3. **isNaN(արժեք)** վերադարձնում է true, եթե արգումենտի արժեքը թիվ չէ և false, հակառակ դեպքում:

4. **parseInt(տող)** ձևափոխում է փակագծերում գրանցված տողը ամբողջ թվի: Այն դեպքերում, երբ տողը պարունակում է ոչ միայն թվեր վերադարձնում է NaN արժեքը (not a number՝ թիվ չէ)

5. **typeof(օբյեկտ)** վերադարձնում է օբյեկտի տեսակը որպես տող: Չնարավոր վերադարձվող արժեքները վեցն են՝ "boolean", "function", "number", "string", "function" և "undefined":

Ստորև բերված ծրագրում բերված են ներկառուցված ֆունկցիաների օգտագործման օրինակները (տես նաև՝ պատկեր 3.3.4.):

Ծր. 3.3.4. Ներկառուցված ֆունկցիաների օգտագործման օրինակ

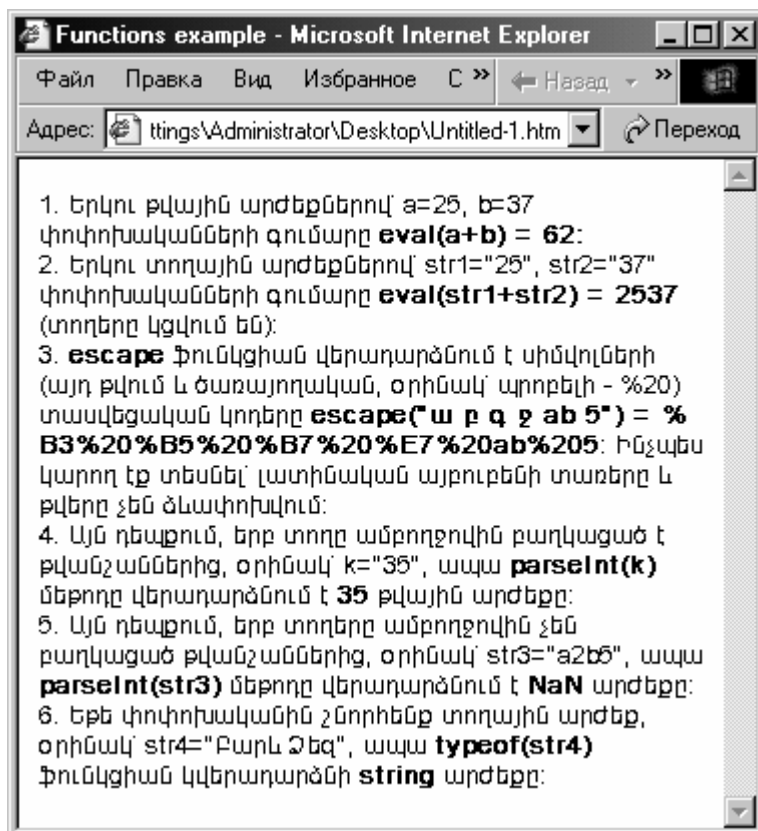
```
<html >
<head>
<title>Functions example
</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script language="JavaScript">
<!--
a=25;b=37;
```

```

document.writeln("1. Երկու թվային արժեքներով՝ a=25, b=37 փոփո-
խականների գումարը <b>eval&#40;a+b&#41;="+eval(a+b)+"</b><br />");
str1="25";str2="37";
document.writeln(
"2. Երկու տողային արժեքներով՝ str1=&#34;25\\", str2=\\'37\\'
փոփոխականների գումարը <b>eval&#40;str1+str2&#41; =
"+eval(str1+str2)+" </b>&#40;տողերը կցվում են&#41;:<br />");
resesc=escape("ա բ գ զ ab 5");
document.writeln("3. <b>escape</b> ֆունկցիան վերադարձնում է
սիմվոլների \\(այդ թվում և ծառայողական, օրինակ՝ պրոբելի -
%20\\) տասվեցական կոդերը <b>escape&#40;\\'ա բ գ զ ab 5\\'&#41;
= "+resesc+"</b>: Ինչպես կարող էք տեսնել՝ լատինական
այբուբենի տառերը և թվերը չեն ձևափոխվում:<br />");
k="35";
document.writeln(
"4. Այն դեպքում, երբ տողը ամբողջովին բաղկացած է
թվանշաններից, օրինակ՝ k=\\'35\\", ապա
<b>parseInt&#40;k&#41;</b> մեթոդը վերադարձնում է
<b>"+parseInt(k)+"</b> թվային արժեքը:<br />"
);
str3="a2b5";
int1=parseInt(str3);
document.writeln(
"5. Այն դեպքում, երբ տողերը ամբողջովին չեն բաղկացած
թվանշաններից, օրինակ՝ str3=\\'a2b5\\", ապա
<b>parseInt&#40;str3&#41;</b>
մեթոդը վերադարձնում է <b>"+int1+"</b> արժեքը:<br />"
);
var str4="Բարև Ձեզ";
document.writeln(
"6. Եթե փոփոխականին շնորհենք տողային արժեք, օրինակ՝
str4=\\'Բարև Ձեզ\\", ապա
<b>typeof&#40;str4&#41;</b> ֆունկցիան կվերադարձնի
<b>"+typeof(str4)+"</b> արժեքը:"
);
//-->
</script></body></html>

```

Ուշադրություն դարձրեք, որ ծրագրում որոշ հատուկ նշանները արտապատկերելու նպատակով օգտագործված են պրիմիտիվներ՝ բացող փակագիծ “(“ - (փակող փակագիծ “)” -):



Պատկեր 3.3.4. Ներկառուցված ֆունկցիաների կիրառությունը

§3.4. Իրադարձությունների մշակումը Javascript լեզվում

JavaScript լեզվի կարևորագույն գործառնություններից մեկը՝ օգտվողի տարբեր գործողություններին արձագանքելն է: Ինչպես վերը ասվել է այն ամենը, ինչ օգտվողը կատարում է էջում, ինչպես նաև այն ամենը ինչ կատարվում է բրաուզերում՝ իրադարձություններ են, լինի դա կոճակի կամ ստեղծի սեղմում, մկնիկի շարժում կամ որևէ օբյեկտի վրա տեղադրում, էջի ներմուծում և այլն: Իրադարձությունների մշակումը կազմակերպելու համար պետք է նախ հասկանալ, թե ինչպե՞ս են ստեղծվում իրադարձությունները մշակող ֆունկցիաների կանչերը ինչպես ընդունված է ասել՝ «մշակիչ»:

ՈՆԵՐԻ” միջոցով: Ընդհանուր դեպքում մշակիչի ստեղծման ֆորմատը հետևյալն է՝ **<էլեմենտ մշակիչ=“ֆունկցիայի կանչ”>**:

Ինչպես տեսնում ենք մշակիչը ձևակերպվում է որպես HTML էլեմենտի ատրիբուտ, որի արժեքը՝ իրադարձությունը մշակելու համար նախատեսված ֆունկցիայի կանչն է կամ, որոշ դեպքերում, անմիջապես կատարվող ծրագրային կոդը:

Գոյություն ունի HTML էլեմենտների որոշակի հավաքածու, որը սատարում է իրադարձությունների մշակիչների ստեղծումը: Հիմնականում իրադարձությունները կապված են **<form>** էլեմենտի հետ, սակայն կան մշակիչներ, որոնք “սպասարկում են” մի շարք այլ էլեմենտներ: Գոյություն ունեն մեծ մշակիչներ, որոնք հատուկ են միայն որոշակի էլեմենտներին: Աղյուսակ 3.4.1-ում ներկայացված են առավել հաճախ օգտագործվող մշակիչների նկարագրությունները: Դժվար չէ նկատել, որ մշակիչը դա պարզապես համապատասխան իրադարձության անվանումն է, որի սկզբում ավելացված է “on” մասնիկը, օրինակ՝ **onclick**, **onfocus** և այլն:

Կառուցենք սցենար, որը օգտվողին թույլ կտա ներմուծելով կամայական թիվ և ստեղծելով իրադարձություն՝ մկնիկի ստեղծիկ կտտոցը կոճակին (դրան համապատասխանում է onclick մշակիչը) ստանալ պատուհանում պատասխանը: Նպատակը իրագործելու համար սկզբում կառուցենք ֆորմա, որը բաղկացած է երկու տեքստային պատուհաններից և մեկ կոճակից՝

<form>

Ներմուծեք թիվը

<input type="text" name="usrEntry" id="usrEntry" size="2" />

**
**

Թվի քառակուսին հավասար է

<input type="text" name="result" id="result" size="5" readonly="readonly"/>

**
**

<input type="button" value="Count" />

</form>

Որպեսզի կարողանանք սցենարում դիմել տեքստային պատուհաններին դրանցից յուրաքանչյուրին շնորհիված են միաժամանակ ունիկալ անուն (հին բրաուզերների համար) և իդենտիֆիկատոր (նոր ստանդարտին համապատասխան): “usrEntry” պատուհանը ծառայում է թիվը մուտքագրելու, իսկ “result”-ը՝ արդյունքը արտապատկերելու համար: Մուտքագրելով թիվը և սեղմելով կոճակին օգտվողը կտեսնի արդյունքը:

Աղյուսակ 3.4.1.

Իրադարձությունների մշակիչները

Մշակիչը	Պատահարը	Էլեմենտները
1	2	3
onfocus	Ֆորմայի (կամ որոշ այլ) էլեմենտի լրացում (ֆոկուսի ստացում)՝ ստեղծի կամ մկնիկի սեղմում:	input,select,textarea, button,a,label
onblur	Ֆոկուսի կորուստը, երբ ստեղծը կամ մկնիկի կոճակը սեղմվում են էլեմենտից դուրս:	Նույն էլեմենտները, ինչ և onfocus մշակիչի համար:
onclick	Օգտվողը կտտացնում է էլեմենտի վրա:	Բոլոր էլեմենտների համար:
ondblclick	Կրկնակի կտտոց էլեմենտի վրա:	Բոլոր էլեմենտների համար:
onchange	Արժեքի փոփոխությունը և ելքը էլեմենտից:	Միայն input,select և textarea էլեմենտների համար
onkeydown	Ստեղծը սեղմված է պահվում, երբ էլեմենտը ֆոկուսում է գտնվում:	input,select,textarea, button,a,label:
onkeypress	Ստեղծը սեղմվում է և թողնվում, երբ էլեմենտը ֆոկուսում է գտնվում:	input,select,textarea, button,a,label:
onkeyup	Ստեղծը թողնվում է , երբ էլեմենտը ֆոկուսում է գտնվում:	input,select,textarea, button,a,label:
onload	Էջը բեռնվում է:	body, frameset
onunload	Էջը փակվում է:	body, frameset
onmouseover	Մկնիկի մշիչը էլեմենտի վրա է:	Բոլոր էլեմենտների համար:
onmousedown	Մկնիկի կոճակը սեղմվում է, երբ մշիչը էլեմենտի վրա է:	Բոլոր էլեմենտների համար:
onmouseup	Մկնիկի կոճակը թողնվում է, երբ մշիչը էլեմենտի վրա է:	Բոլոր էլեմենտների համար:
onmousemove	Մկնիկը շարժվում է, երբ մշիչը էլեմենտի վրա է:	Բոլոր էլեմենտների համար:

Աղյուսակ 3.4.1. (շարունակությունը)

1	2	3
onmousemove	Մկնիկը շարժվում է, երբ նշիչը էլեմենտի վրա է:	Բոլոր էլեմենտների համար:
onmouseout	Մկնիկը շարժվում է և նշիչը դուրս է գալիս էլե- մենտից:	Բոլոր էլեմենտների համար:
onselect	Դաշտը ընտրվում է:	Input,textarea
onreset	Ֆորմայի մաքրումը:	form
onsubmit	Ֆորմայի պնդումը:	form

Այժմ ստեղծենք կոճակի սեղմում իրադարձության մշակիչը: Քանի որ այն կապված է կոճակի հետ, ապա պետք է գրանցվի կոճակը նկարագրող տեգում, որպես ատրիբուտ՝

```
<input type="button" value="Count" onclick="fcompute()" />
```

Ինչպես կարելի է տեսնել բերված կոդից՝ ատրիբուտի արժեքը `fcompute()`՝ ֆունկցիայի կանչն է: Այժմ մեզ մնաց միայն կառուցել այդ ֆունկցիան, որը կհաշվի թվի քառակուսին և կգրանցի այն `"result"` տեքստային պատուհանում՝

```
<script>
```

```
<!--
```

```
function fcompute()
```

```
{
```

```
  numb=document.all("usrEntry").value;
```

```
  document.all("result").value=Math.pow(numb,2);
```

```
}
```

```
//-->
```

```
</script>
```

Ֆորմայի էլեմենտներին ֆունկցիայի մարմնից դիմելու, արժեքները ստանալու կամ շնորհելու համար օգտագործված է բրաուզերի օբյեկտային մոդելը (այն ավելի մանրամասն կքննարկվի հաջորդ պարագրաֆում), համաձայն որի փոստաթուղթը օբյեկտ է, որի ենթաօբյեկտներն են հանդիսանում մասնավորապես ֆորմայի էլեմենտները: Վերջիններին դիմելու համար օգտագործված է հետևյալ եղանակը՝ փաստաթղթի բոլոր ենթաօբյեկտների հավաքածույուն (կոլեկցիայում), որը նշանակվում է `all` բանալիական բառով, դիմում ենք կոնկրետ երկու օբյեկտներին դրանց իդենտիֆիկատորների միջոցով՝

`document.all("usrEntry")` և `document.all("result")`:

Սկզբում մուտքի պատուհանում օգտվողի կողմից գրանցված արժեքը՝ `document.all("usrEntry").value` շնորհիվում է `numb` փոփոխականին՝ `numb=document.all("usrEntry").value`; և, ապա `Math` օբյեկտի համապատասխան մեթոդի միջոցով հաշվարկվում է թվի քառակուսին ու շնորհիվում ելքի ("result") պատուհանին որպես արժեք՝ `document.all("result").value=Math.pow(numb,2)`; անմիջապես արտապատկերվելով էկրանին:

Միավորելով այժմ բերված հատվածները կազմենք համապատասխան փաստաթուղթը (տես՝ Ծր.3.4.1.): Պատկեր 3.4.1-ում բերված է ծրագրի աշխատանքի արդյունքը:

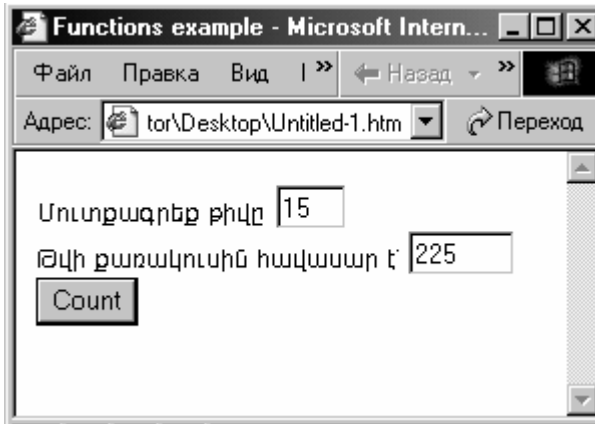
Ծր. 3.4.1. onclick մշակիչի ստեղծման օրինակ

```
<html>
<head>
<title>Functions example</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script language="JavaScript">
<!--
function fCcompute()
{ numb=document.all("userEntry").value;
  document.all("result").value=Math.pow(numb,2); }
//-->
</script>
</head>
<body>
<form name="formPow" id="formPow">
Մուտքագրեք թիվը
<input type="text" name="UserEntry" id="userEntry" size="2" /><br />
Թվի քառակուսին հավասար է <input type="text" name="result"
size="5" readonly="readonly" /><br />
<input type="button" value="Count" onclick="fCompute()" />
</form></body></html>
```

`document.all("կոնկրետ օբյեկտի իդենտիֆիկատորը")` դիմումի եղանակը թույլ է տալիս դիմել կամայական օբյեկտին անմիջականորեն, չպահպանելով օբյեկտային մոդելի հիերարխիան: Մոդելի հիերարքիայի սկզբնունքին խիստ հետևելու դեպքում անհրաժեշտ է իդենտիֆիկացնել, նաև երկու տեքստային պատուհանները պարունակող ֆորման: Այդ դեպքում ֆորմայի կամայական էլեմենտին դիմելու համար օգտագործվում է հետևյալ սխեման՝

փաստաթուղթ.ֆորմայի_անուն.էլեմենտի_անուն.հատկություն:

Օրինակ “result” էլեմենտին կարելի է դիմել հետևյալ եղանակով՝ `document.formPow.result.value`:



Պատկեր 3.4.1. onclick մշակիչի օգտագործման օրինակ

Օգտվողի գործողություններին արձագանքելու եղանակներից մեկը դա որևէ հաղորդակցության ցուցադրումն է: JavaScript-ն ունի ներկառուցված ֆունկցիա՝ `alert()`, որի միջոցով էկրանին ցուցադրվում են այսպես կոչված նախազգուշական հաղորդակցություններ: `alert` պատուհանը երկխոսության մոդալ պատուհան է, որն ունի ընդամենը մեկ կոճակ՝ “OK”: Մոդալ նշանակում է, որ մինչև այն չփակվի (չսեղմվի “OK” կոճակը) ծրագրի շարունակումը անհնար է: Բերենք օրինակ: Եթե Ծր.3.4.1-ում `<body>` տեղում ավելացնենք փաստաթղթի բեռնման իրադարձության մշակիչը՝ `onload="alert('Thank You for visit')"`, ապա պատուհանի յուրաքանչյուր բեռնման ժամանակ դուրս կբերվի փակագծերում գրանցված տեքստը: Ուշադրություն դարձրեք, որ տեքստը վերցված է եզակի չակերտների մեջ, քանի որ կրկնակի չակերտները օգտագործված են `onload` ատրիբուտի արժեքը պառփակելու համար: Պատկեր 3.4.2-ում ցուցադրված է պատուհանի բեռնման պահը՝ դուրս բերված հաղորդակցությունով:

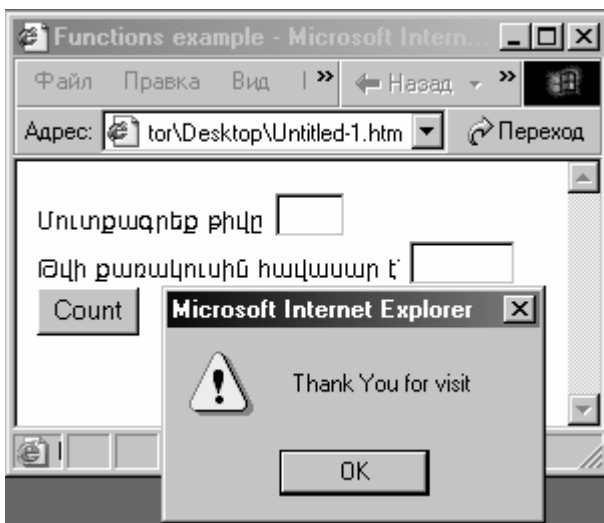
Անհրաժեշտ ինֆորմացիան պատուհաններին կան ֆունկցիաներին կարելի է հաղորդել նաև օգտագործելով `this` բանալիական բառը: Ձևափոխենք Ծր. 3.4.1-ում կառուցված ֆորմայի “userEntry” մուտքագրման պատուհանի և “Count” կոճակի ծրագրային կոդը հետևյալ կերպով՝

```
<input type="text" id="userEntry" name="userEntry" size="5"
```

```

    onchange="fCompute(this.value)" /> և
    <input type="button" value="count" />:

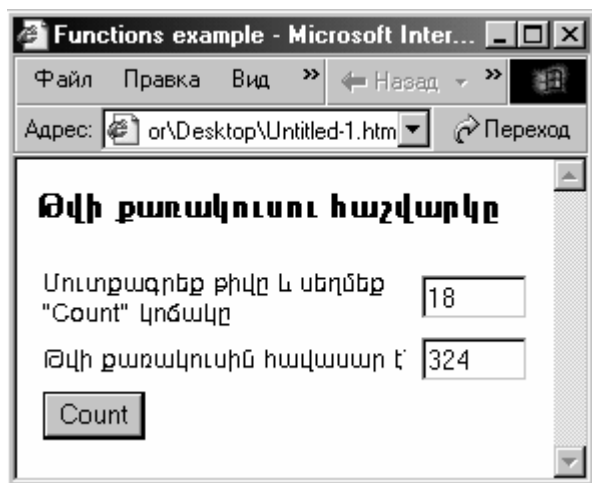
```



Պատկեր 3.4.2. onload մշակիչի և alert հաղորդակցության պատուհանի օգտագործման օրինակ

Ինչպես տեսնում ենք "userEntry" տեքստային պատուհանի տեղում ավելացված է onchange մշակիչը, որի միջոցով կազմակերպվում է մուտքագրված թվի քառակուսին հաշվարկող fCompute() ֆունկցիայի կանչը: Սակայն այս դեպքում ֆունկցիային this բառի միջոցով հաղորդվում է ընթացիկ էլեմենտի, այսինքն տեքստային պատուհանի, պարունակությունը (this.value): Ըստ էության this բառը այստեղ փոխարինում է էլեմենտի անունը: Ուշադրություն դարձրեք, որ "Count" կոճակի տեղում արդեն բացակայում է onclick մշակիչը, քանի որ մկնիկի ստեղծի կտտացնելը կոճակի (ինչպես նաև որևէ այլ էլեմենտի կամ տիրույթի, բացի "userEntry" տեքստային պատուհանից) վրա բերում է change իրադարձության կատարմանը և fcompute() ֆունկցիայի կանչին (հիարկե եթե օգտվողը փոփոխել է "userEntry" պատուհանի պարունակությունը):

Նշենք նաև, որ ֆորմայի ավելի գեղեցիկ ձևավորման նպատակով դրա էլեմենտները և ուղեկցող տեքստը տեղադրված են աղյուսակում: Ծրագրի կատարման արդյունքը ցուցադրված է պատկեր 3.4.3-ում:



Պատկեր 3.4.3. this բանալիական բառի օգտագործման օրինակ

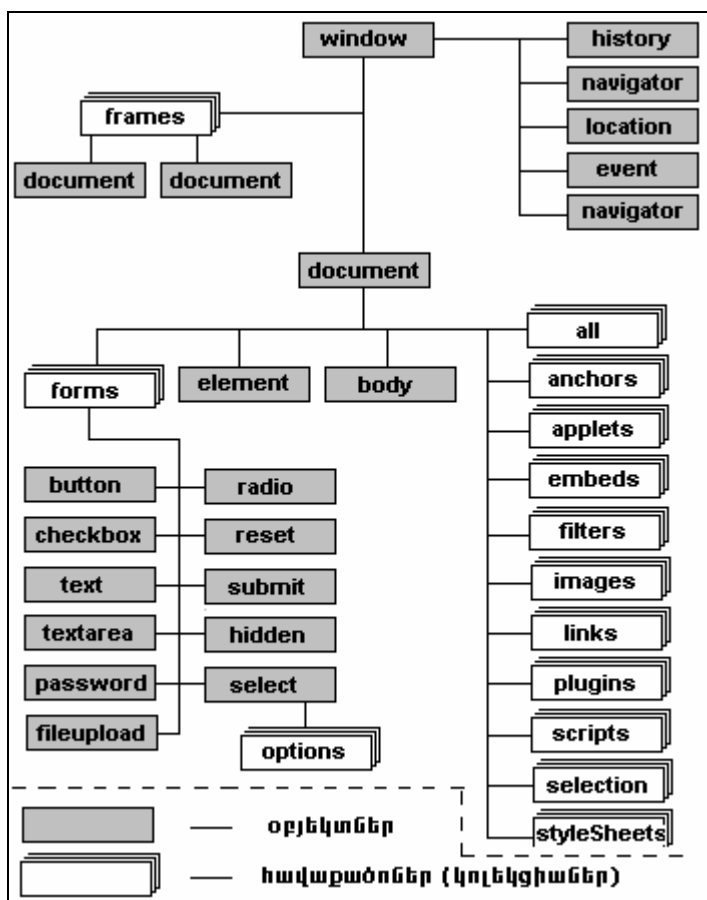
§3.5. Բրաուզերի օբյեկտային մոդելը, օբյեկտները և հավաքածուները (կոլեկցիաները)

Նախկան իրադարձությունների և ֆորմաների մշակման հարցերի քննարկումը շարունակելը անհրաժեշտ է անդրադարձնալ շատ կարևոր և հիմնադրիչ հասկացության՝ բրաուզերի օբյեկտային մոդելի (այդ թվում նաև փաստաթղթի օբյեկտային մոդելի՝ DOM – Document Object Model) պարզաբանմանը:

3.5.1. Օբյեկտային մոդելի ընդհանուր կառուցվածքը

Պատկեր 3.5.1-ում ներկայացված է բրաուզերի օբյեկտային մոդելը: window-ն բրաուզերի օբյեկտային մոդելի հիերարխիայի գլխավոր (վերին մակարդակի) օբյեկտն է: Այն վերաբերվում է տվյալ պահին ակտիվ վիճակում գտնվող պատուհանին և կարող է վերագրվել ինչպես գլխավոր, այնպես էլ <iframe> տեգում ստեղծված ֆրեյմերից յուրաքանչյուրի ներքին պատուհանին:

Չնայած այդ փաստին իրականում մոդելի հիմքն է հանդիսանում document օբյեկտը, քանի որ յուրաքանչյուր էջի և դրա բովանդակության մեծ մասը հանդիսանում են document օբյեկտի բաղադրիչներ: Յուրաքանչյուր window օբյեկտ կարող է պարունակել մի քանի փաստաթղթեր կամ ֆրեյմերի հավաքածուներ: Համարյա նույնը կարելի է ասել նաև document օբյեկտի վերաբերյալ՝ այն կարող է ընդգրկել մի շարք օբյեկտներ և հավաքածուներ:



Պատկեր 3.5.1. Բրաուզերի օբյեկտային մոդելը

Կարելի է նկատել, որ պատկեր 3.5.1-ում ներկայացված սխեմայում օբյեկտային մոդելի որոշ մասերը միավորված են հավաքածուներում (կոլեկցիաներում՝ **collections**): Կոլեկցիան դա կառուցվածք է, որը նման է զանգվածի: Այդպիսի զանգվածներում յուրաքանչյուր էլեմենտ պատկանում է ընդհանուր կառուցվածքին և կապակցված է “հարևանների” հետ: Դրանց համարակալումը կատարվում է ավտոմատ կերպով, ըստ էջում հայտնվելու (այսինքն ծրագրային կոդում գրանցելու) հաջորդականության և սկսվում է

նույնպես 0-ից (ինչպես և զանգվածներում): Որպես օրինակ դիտարկենք պատկեր 3.5.2-ում ներկայացված ֆրեյմերի կոլեկցիան:

<frame id="mainframe"> դիմումը՝ window.frames(0) կամ window.frames("mainframe")	
<frame id="lframe"> window.frames(1) window.frames("lframe")	<frame id="rframe"> window.frames(2) window.frames("rframe")

Պատկեր 3.5.2. window օբյեկտի ֆրեյմերի կոլեկցիան

Ինչպես տեսնում ենք ֆրեյմերի կոլեկցիայում համարակալումը կատարվում է ձախից աջ և վերից վար: Հատկանշական է, որ սցենարում ֆրեյմերին կարելի է դիմել ինչպես հաջորդական համարով, այնպես էլ իդենտիֆիկատորով: Օրինակ՝ "lframe" իդենտիֆիկատոր ունեցող ֆրեյմին դիմելու համար կարելի է գրանցել ինչպես window.frame(1) (քանի որ ֆրեյմերի ընդհանուր հաջորդականությունում այն հաշվով երկրորդն է), այնպես էլ՝ window.frames("lframe"):

Կոլեկցիաների օգտագործման օրինակ մենք արդեն բերել ենք Օր. 3.4.1-ում: *document* օբյեկտի **all** կոլեկցիան դա փաստաթղթի բոլոր ենթաօբյեկտների հավաքածուն է (անկախ օբյեկտների տեսակներից): Դրա օգտագործումը թույլ է տալիս դիմել կամայական օբյեկտին ըստ վերջինի իդենտիֆիկատորի: Օրինակ՝ "userEntry" տեքստային պատուհանին դիմելու նպատակով գրանցված է **document.all("userEntry")**:

Սովորաբար **all** կոլեկցիան կիրառվում է այն դեպքերում, երբ փաստաթղթի օբյեկտները բազմատեսակ են և բազմաթիվ, այսինքն դժվար է հիշել դրանց հաջորդականությունը:

Որպես ստանդարտ բրաուզերի օբյեկտային մոդելը ընդունվել է վերջերս և սցենարներում համապատասխան հատկությունները և մեթոդները օգտագործելու նպատակով օբյեկտներին դիմելու եղանակները տարբեր բրաուզերներում տարբեր են: Դա նշանակում է, որ բրաուզերից կախում չունեցող սցենարների ստեղծումը կապված է որոշակի դժվարությունների հետ՝ օբյեկտների հասցեների գրանցման տեսակետից: Ընդհանուր դեպքում օբյեկտի հատկություններին և մեթոդներին դիմելու համար անհրաժեշտ է սկսել

առավել ընդհանուրից (window) և “խորանալ” մինչև անհրաժեշտ էլեմենտը, բաց չթողնելով ոչ մի միջանկյալ կոնտեյներ:

Օրինակ՝ Ծր.3.4.1-ում բերված “textEntry” տեքստային պատուհանին դիմելու համար (որը հանդիսանում է “formPow” ֆորմայի բաղադրիչ էլեմենտ) ղեկավարվելով ընդհանուր կանոնով՝ անհրաժեշտ է սցենարում գրանցել հետևյալ ճանապարհը՝

window.document.formEntry.textEntry.հատկություն/մեթոդ

Սակայն պարզվում է, որ բոլոր ճանապարհներն ունեն մեկ ընդհանուր հատկություն՝ ազդեցության տիրույթ: Դա նշանակում է, որ գտնվելով օբյեկտների որոշակի վիրտուալ մակարդակում կարելի է անտեսել դեպի կոնկրետ օբյեկտը “տանող” ճանապարհի մի մասը, քանի որ “տվյալ տիրույթում” այն պարզապես արդիական չէ (հիշենք թե ինչպես են կառուցվում հղումային հասցեները՝ բացարձակ և հարաբերական): Քանի որ սցենարը գրելիս մենք գտնվում ենք ոչ թե “պատուհանների”, այլ փաստաթղթերի, այսինքն միևնույն պատուհանի մակարդակում, ապա window բառը կարելի է “հանգիստ խղճով” բաց թողնել (պատուհանը այդ դեպքում համարվում է ընթացիկ)՝

document.formEntry.textEntry.հատկություն/մեթոդ

- “Internet Explorer” և “Netscape Navigator” բրաուզերների վերջին վարկածներում (6-րդ) հնարավոր է դարձել կազմակերպել դիմումները ենթաօբյեկտներին, օգտագործելով միասնական եղանակ՝ getElementById() մեթոդը: Կամայական էլեմենտին կարելի է դիմել շնորհիվ ֆունկցիային որպես արգումենտ էլեմենտի իդենտիֆիկատորը: Օրինակ՝ document.getElementById(“textEntry”):

Ցուցանակների օգտագործումը թույլ է տալիս զգալիորեն հեշտացնել օբյեկտներին դիմելու եղանակը: Ցուցանակը՝ օբյեկտի հասցեն պարունակող փոփոխական է: Փոփոխենք Ծր. 3.4.1-ում ներկայացված ֆորմայի նկարագրությունը հետևյալ կերպով՝

```
<form name="formPow" id="formPow">
```

Սուտքագրեք թիվը

```
<input type="text" name="UserEntry" id="userEntry" size="2" /><br />
```

Թվի քառակուսին հավասար է՝

```
<input type="text" name="result" size="5" readonly="readonly" /><br />
```

```
<input type="button" value="Count" onclick="fCompute(formPow)" />
```

```
</form>
```

Ինչպես տեսնում ենք onclick մշակիչի fCompute() ֆունկցիայի կանչում ավելացված է իրական պարամետր՝ ֆորմայի անունը

(իդենտիֆիկատորը) - fCompute('formPow'): Եթե այժմ փոփոխենք սցենարում այդ ֆունկցիայի նկարագրությունը հետևյալ կերպով՝

```
function fCcompute(theForm)
{
  numb=theForm.userEntry.value;
  theForm.result.value=Math.pow(numb,2);
},
```

ապա կստանանք նույն արդյունքը, չգրանցելով ֆորմայի էլեմենտների լրիվ հասցեները: Դա բացատրվում է ցուցանակների, այսպես կոչված գործողության տիրույթի հատկությամբ՝ քանի որ ցուցանակը գործում է ընթացիկ պատուհանի (window) ընթացիկ փաստաթղթի (document) ներքո, ապա նա ավտոմատ կերպով ընդգրկում է վերջինների հասցեները և fCompute('formPow') գրանցումը փաստորեն նշանակում է՝ fCompute(window.document.formPow): Ինչպես տեսնում ենք օբյեկտի ցուցանակի ստեղծումը բավականին դյուրին է՝ պարզապես անհրաժեշտ է ֆունկցիայի կանչում գրանցել օբյեկտի անունը որպես հաղորդվող պարամետրի իրական արժեք (երբ օբյեկտը հանդիսանում է մի այլ օբյեկտի էլեմենտ, ապա որպես հաղորդվող պարամետրի արժեք գրանցվում է “հայրական” օբյեկտի անունը):

Ասվածից ամենևին էլ չի հետևում, որ վերին մակարդակի օբյեկտները (window-ն և նրա ենթաօբյեկտները) առհասարակ կարելի է չօգտագործել: Դրանց հատուկ են մի շարք օգտակար և հաճախ կիրառվող հատկություններ և մեթոդներ:

3.5.2. window օբյեկտի հատկությունները և մեթոդները

Ինչպես և յուրաքանչյուր օբյեկտ window-ն ունի հատկություններ և մեթոդներ (հիմնական հատկությունները և մեթոդները թվարկված են համապատասխանաբար աղյուսակներ 3.5.1. և 3.5.2-ում): Հատկությունների մի մասը կարելի է միայն “ընթերցել”, այսինքն դրանց արժեքները հնարավոր չէ փոփոխել, քանի որ պատուհանը արդեն գոյություն ունի: Կարելի է նշել window օբյեկտի հետևյալ հատկությունները՝

- ◆ **name** - դա այն տեքստն է, որը գրանցվում է պատուհանի վերնագրային մասում (պետք չէ շփոթել փաստաթղթի վերնագրի հետ);
- ◆ **length** - թույլ է տալիս ստանալ պատուհանում արտապատկերվող ֆրեյմերի քանակը;
- ◆ **self** - հղում ինքն իրեն;
- ◆ **status** - օգտակար է այն դեպքում, երբ ցանկալի է հայտնել օգտվողին հաղորդակցություններ միջանկյալ գործողություն-

ների վերաբերյալ: Սովորաբար դա էջում կատարվող իրադարձությունների հակիրճ նկարագրությունն է: Օրինակ՝ ստեղծված է հղում արտադրանքի տեսականին ներկայացնող էջին: Եթե այն գրանցվի հետևյալ կերպով՝

```
<a href="products.html" onmouseover="window.status=
'Արտադրանքի տեսականիի դիտարկում' "
onmouseout="window.status=' '>
```

ապա այն դեպքում, երբ մկնիկի նշիչը տեղադրվի հղումի վրա (դա նշանակում է, որ կատարվել է mouseover իրադարձությունը, որի մշակիչն է՝ onmouseover) պատուհանի վիճակի տողում (status-bar) կգրանցվի 'Արտադրանքի տեսականիի դիտարկում' տեքստը, իսկ երբ նշիչը հեռացվի հղումից (կատարվում է mouseout իրադարձությունը, որի մշակիչն է՝ onmouseout)՝ վիճակի տողը կմաքրվի: Կրկին նշենք՝ քանի որ փաստաթուղթը պատահանում է կոնկրետ ակտիվ պատուհանին, ապա կարելի է հղումում նույն մշակիչները գրանցել առանց window բառի օգտագործման՝
onmouseover="status='Արտադրանքի տեսականիի դիտարկում' " :

Աղյուսակ 3.5.1.

window օբյեկտի հատկությունները

Հատկությունը	Նկարագրությունը
parent	Վերադարձնում է ընթացիկ պատուհանի “ծնողականը”
self	Վերադարձնում է հղումը ընթացիկ պատուհանին
top	Վերադարձնում է հղումը գլխավոր պատուհանին
name	Պատուհանի անվանումը
opener	Ընթացիկով ստեղծված պատուհանը
closed	Հաղորդվում է, երբ այն փակված է
status	Բրաուզերի “վիճակի տողի” տեքստը
returnValue	Սահմանում է իրադարձության կամ երկխոսության պատուհանի վերադարձվող փոփոխականը
client	Հղում, որը վերադարձնում է navigator-ի օբյեկտը բրաուզերին
length	Վերադարձնում է ֆրեյմերի քանակը

window օբյեկտն ունի մի շարք մեթոդներ, որոնցից alert() մեթոդին մենք արդեն ծանոթ ենք:

Աղյուսակ 3.5.2.

window օբյեկտի մեթոդները

Մեթոդը	Նկարագրությունը
open	Բացում է բրաուզերի նոր պատուհան
close	Փակում է ընթացիկ պատուհանը
showModalDialog	Բացում է նոր մոդալ երկխոսության պատուհան
showHelp	Բացում է օգնության մոդալ պատուհան
alert	Արտապատկերում է նախազգուշացման պատուհանը հաղորդակցությունով և OK կոճակով
prompt	Արտապատկերում է տեքստային դաշտով և հաղորդակցությունով առաջարկման պատուհանը
confirm	Արտապատկերում է հաստատման պատուհանը հաղորդակցությունով, OK և Cancel կոճակներով
navigate	Բեռնվում է մի այլ էջ նշված հասցեով
blur	Հեռացնում է ընթացիկ էջը ֆոկուսից
focus	Փոխանցում է ֆոկուսը ընթացիկ էջին
scroll	Բացում է պատուհանը տրված լայնության և բարձրության
setInterval	Ապահովում է ծրագրի կամ որևէ գործողության կատարումը որոշակի պարբերությամբ (միլիվարկյաններով)
setTimeout	Ապահովում է ծրագրի կամ որևէ գործողության կատարումը էջի բեռնումից որոշակի ժամանակ անց (միլիվարկյաններով)
clearInterval	Անջատում է setInterval մեթոդով գործարկված թայմերը
clearTimeout	Անջատում է setTimeout մեթոդով գործարկված թայմերը

confirm() մեթոդը նույնպես թույլ է տալիս արտապատկերել որևէ հաղորդակցություն պարունակող մոդալ պատուհան, սակայն ի տարբերություն alert-ի, օգտվողի ընտրությունից կախված, կարող է վերադարձնել երկու հնարավոր բուլյան արժեքներ՝ true կամ false: Այդ նպատակով պատուհանը պարունակում է երկու կոճակներ OK (true) և Cancel (false): Պատուհանի փակման (փոքրիկ խաչը աջ վերին անկյունում) կոճակի սեղմումը համազոր է Cancel-ին:

prompt() մեթոդը նույնպես բացում է երկխոսության պատուհան, որը բացի հաղորդակցությունից պարունակում է նաև տեքստային պատուհան, որում օգտվողը պետք է մուտքագրի որևէ արժեք:

Քանի որ այդ երկու մեթոդներն էլ վերադարձնում են արժեքներ, ապա դրանք հետագայում օգտագործելու նպատակով անհրաժեշտ է շնորհել վերադարձվող արժեքը որևէ փոփոխականին: Օրինակ՝

```
var keepGoing=confirm("Ցանկանու՞մ եք շարունակել"),  
var e_mail=prompt("Գրանցեք ձեր էլեկտրոնային փոստի հասցեն",  
"Օրինակ՝ vpumpkin@fakecorp.com");
```

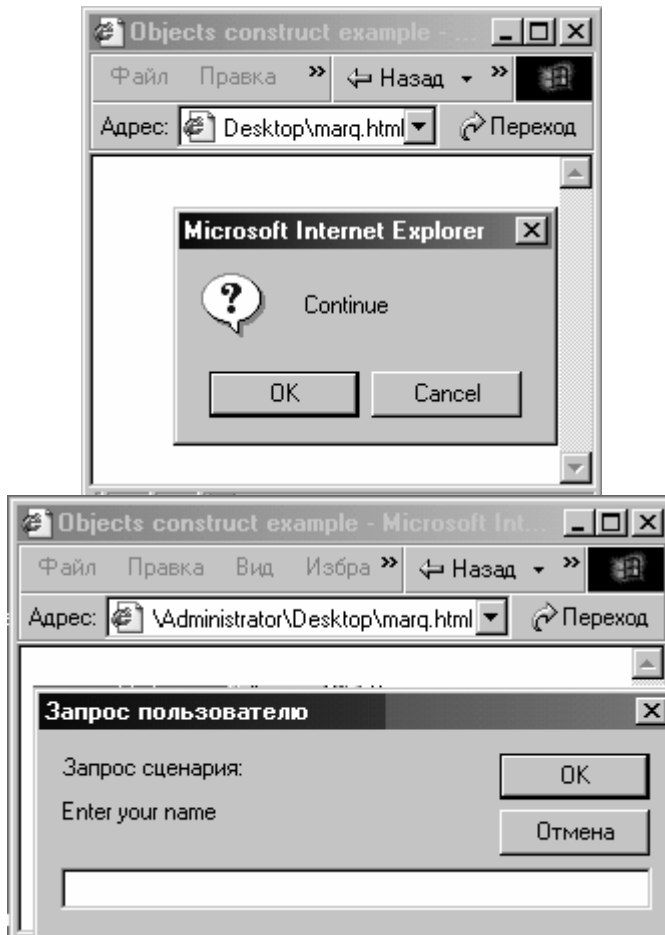
Ծր. 3.5.1-ում բերված է այդ մեթոդների օգտագործման օրինակ:

Ծր. 3.5.1. **confirm()** և **prompt()** մեթոդների կիրառության օրինակ

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-  
transitional.dtd">  
<html xmlns="http://www. w3.org/1999/xhtml" >  
<head>  
<title>Objects construct example</title>  
<meta http-equiv="Content-Script-Type" content="text/javascript" />  
<script language="JavaScript">  
<!--  
function fConfirm()  
{  
var v_name;  
var keepGoing=confirm("Continue");  
if (keepGoing)  
v_name=prompt("Enter your name", "");  
if(!v_name)  
v_name="Dolly";  
document.write("Hello "+v_name);  
}  
//-->  
</script>  
</head>  
<body onload="fConfirm()">  
</body>  
</html>
```

Պարզաբանենք Ծր. 3.5.1-ում բերված կոդը: window օբյեկտի նշված մեթոդները օգտագործելու նպատակով կազմված է սցենար: Այն պարունակում է fConfirm() ֆունկցիան, որի կանչը կազմա-

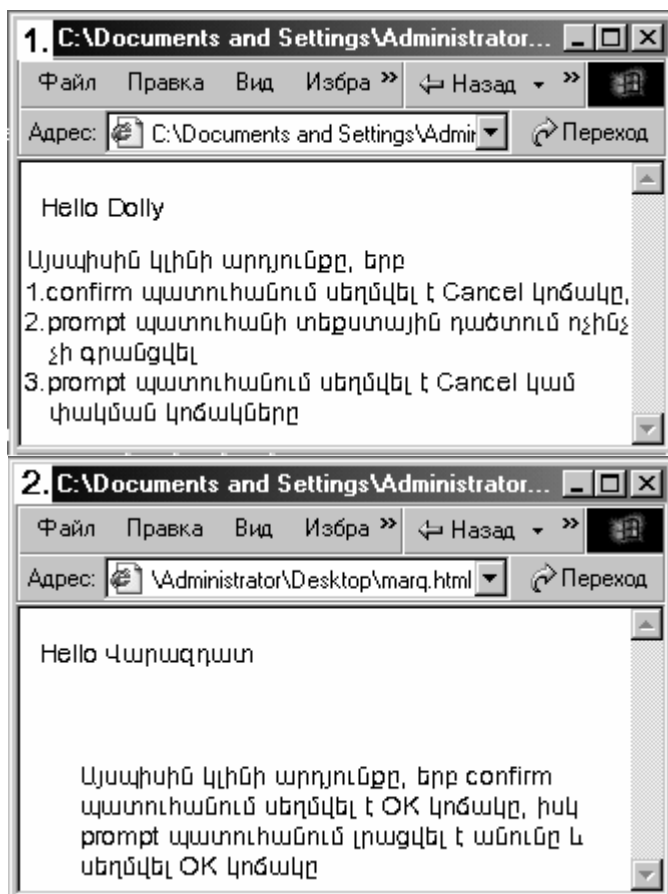
կերպելու համար <body> տեղում ստեղծվել է onload մշակիչը (դա նշանակում է, որ փաստաթղթի մարմնի բեռնման դեպքում կիրառործվի անցում fConfirm() ֆունկցիայի կատարմանը):



Պատկեր 3.5.3. confirm և prompt պատուհանների օրինակ

Սկզբում ստեղծված է v_name փոփոխականը, որին սցենարի կատարման արդյունքում կշնորհվի էկրանին դուրս բերվող տեքստը: Այնուհետև բացվում է երկխոսության confirm պատուհանը (պատկեր 3.5.3.-ի վերին մասը)

var keepGoing=confirm("Continue");



Պատկեր 3.5.4. Ծր. 3.5.1-ի արդյունքային պատուհանները՝

1. Երբ որևէ փուլում սեղմված է Cancel կամ փակման կոճակը:
2. Երբ confirm-ում սեղմված է OK, իսկ prompt-ում լրացված է անունը և սեղմված է OK:

Եթե օգտվողը սեղմում է OK կոճակը keepGoing փոփոխականին շնորհիվում է true արժեքը և, քանի որ կատարվում է առաջին if օպերատորի պայմանը, ապա բացվում է նաև prompt պատուհանը,

որի երկրորդ պարամետրին որպես արժեք շնորհվում է դատարկ տող, այսինքն պատուհանը բացվելիս տեքստի մուտքի պատուհանում ոչինչ չի գրանցվի (պատկեր 3.5.3-ի ստորին մասը):

Այժմ եթե օգտվողը գրանցի տեքստի մուտքի պատուհանում կանայական անուն և սեղմի OK կոճակը, ապա `v_name` փոփոխականին կշնորհվի գրանցված արժեքը: Երկրորդ `if` օպերատորի պայմանը չի կատարվի, քանի որ `!v_name` պայմանը նշանակում է, որ `v_name` փոփոխականի արժեքը չպետք է լինի `true` (ճշմարիտ): Օպերատորը կանոնավոր և կկատարվի `document.write("Hello "+v_name)` մեթոդը: Արդյունքում կստացվի պատկեր 3.5.4.(2)-ում ներկայացված պատուհանը (իհարկե առանց մեկնաբանության):

Բոլոր այլ դեպքերում կստացվի պատկեր 3.5.4. (1) –ում ներկայացված արդյունքը:

`window` օբյեկտին հատուկ են մահ մեթոդներ, որոնց միջոցով հնարավորություն է ընձեռնվում ցուցադրել որոշ էջեր բրաուզերի մի քանի առանձին պատուհաններում, ընդ որում ինչպես սովորական, այնպես էլ և մոդալ:

➤ Հիշեցնենք՝ մոդալ պատուհանի տարբերությունը սովորականից կայանում է գոյություն ունեցող պատուհանի հետ սահմանվող կապի եղանակում: Սովորական պատուհանները աշխատում են իրարից անկախ, այսինքն կարելի է ակտիվացնել դրանցից յուրաքանչյուրը, աշխատել դրանց հետ, փակել և, այդ ամենը չի ազդում (իհարկե եթե դա չի նախատեսված սցենարում) մյուս պատուհանների վրա: Մոդալ պատուհանի կապը գոյություն ունեցողի հետ էապես տարբեր է՝ օգտվողը չի կարող ակտիվացնել սկզբնական պատուհանը մինչև չփակվի մոդալը:

Սովորական պատուհանները օգտագործվում են այն դեպքերում երբ օգտվողին հնարավորություն է տրվում դիտարկել որոշակի ինֆորմացիա, պատկերներ, մուլտիմեդիա և այլն, ընդ որում դրան զուգահեռ շարունակել օգտվել հիմնական էջից:

open() մեթոդը օգտագործվում է նոր ոչ մոդալ պատուհան բացելու համար, օրինակ՝ `window.open("նոր պատուհանի URL հասցեն")`: Ընդհանուր դեպքում `open()` ֆունկցիան կարող է ընդունել չորս արգումենտներ:

- ◆ Առաջին արգումենտի արժեքը՝ այն փաստաթղթի հասցեն է, որը պետք է արտապատկերվի:
- ◆ Երկրորդը՝ բացվող պատուհանի անունն է: Դա ոչ պարտադիր արգումենտ է, որը կարող է կատարել `target` ատրիբուտի դերը փաստաթղթերի ֆրեյմային (շրջանակային) կառուցվածքներ-

րում, եթե որպես պատուհանի անուն տրվում է այն ֆրեյմի անունը, որում անհրաժեշտ է արտապատկերել բացվող փաստաթուղթը:

- ◆ Երրորդ արգումենտի միջոցով կարելի է բացվող պատուհանին վերագրել մի շարք հատկություններ (որոնք բաժանվում են ստորակետներով) ատրիբուտների ցուցակի տեսքով (նոր բացվող պատուհանի ատրիբուտների ցանկը բերված է աղյուսակ 3.5.3-ում):
- ◆ Զորորդ՝ նույնպես ոչ պարտադիր `replace` արգումենտի `true` արժեքը թույլ է տալիս արտապատկերել նոր բացվող էջը գոյություն ունեցող պատուհանում, դուրս մղելով նախորդը `history` օբյեկտի ցուցակում:

`window` օբյեկտի **`showModalDialog()`** մեթոդը թույլ է տալիս բացել նոր պատուհանը, որպես մոդալ երկխոսության պատուհան: Սովորաբար այդպիսի պատուհանները օգտագործվում են այն դեպքերում, երբ մինչև ծրագրի կատարման ընթացքը շարունակելը օգտվողը պետք է կատարի որոշակի ընտրություն կամ մուտքագրի որևէ անհրաժեշտ տվյալ՝ ինչպես `alert`, `confirm` `prompt` մեթոդների կիրառության դեպքում (տարբերությունն այն է, որ նոր պատուհանը ստեղծվում է ծրագրավորողի կողմից և ներկառուցված է): Մոդալ պատուհանների հատկությունների կազմը մի փոքր տարբերվում է սովորականներից (մասնավորապես դրանք չունեն գործիքների և մենյուների վահանակներ): Աղյուսակներ 3.5.3. և 3.5.4-ում բերված են այն ատրիբուտները, որոնց միջոցով կարելի է իրագործել նոր բացվող համապատասխանաբար ոչ մոդալ և մոդալ պատուհանների ձևավորումը: Մոդալ և ոչ մոդալ պատուհանների օգտագործման օրինակները ավելի մանրամասն կքննարկվեն քիչ ուշ, իսկ այժմ դիտարկենք `window` օբյեկտի ենթաօբյեկտները:

3.5.3. `window` օբյեկտի ենթաօբյեկտները

`history` օբյեկտը հիմնականում օգտագործվում է օգտվողի կողմից հաճախած և բրաուզերի հիշողությունում պահպանված URL հասցեներին հասանելիություն ապահովելու համար: Այն ունի մեկ հատկություն՝

`length` – հիշողությունում պահպանված այցելված էջերի քանակը:

Օբյեկտի մեթոդներն են՝

`back()` - բերնվում է նախորդ էջը,

`forward()` - բերնվում է հաջորդ էջը (իհարկե եթե արդեն կատարվել է գոնե մեկ անցում նախորդ էջերին),

`go(n)` – կատարվում է անցում `n` համար ունեցող էջին (եթե `n`-ը դրական է՝ դեպի առաջ, հակառակ դեպքում դեպի հետ):

Աղյուսակ 3.5.3.

open() մեթոդի երրորդ արգումենտի հատկությունները

Ատրիբուտը	Արժեքները	Նկարագրությունը
directories	yes no	Ընդգրկվում է կատալոգի կոճակները
fullscreen	yes no	Պատուհանը բացվում է ամբողջ էկրանով
height	թիվ (պիքսել)	Պատուհանի բարձրությունը
left	թիվ (պիքսել)	Պատուհանի դիրքի ձախ վերին հորիզոնական կոորդինատը
location	yes no	Ընդգրկվում է Address պատուհանը
menubar	yes no	Ընդգրկվում է ստանդարտ մենյուն
resizeable	yes no	Հնարավորություն է ընձեռնվում փոփոխել պատուհանի չափերը
scrollbars	yes no	Ընդգրկվում են հորիզոնական և ուղղահայաց պատճապավենները
status	yes no	Ընդգրկվում է վիճակի տողը
toolbar	yes no	Ընդգրկվում է գործիքների ստանդարտ վահանակը
top	թիվ (պիքսել)	Պատուհանի դիրքի ձախ վերին ուղղահայաց կոորդինատը
width	թիվ (պիքսել)	Պատուհանի լայնությունը

Աղյուսակ 3.5.4.

showModalDialog() մեթոդի հատկությունները

Հատկութ.	Արժեքները	Նկարագրությունը
border	thick thin	Շրջանակի “հաստությունը”
center	yes no	Հավասարեցում (ըստ գլխավոր պատուհանի)
dialogHeight	թիվ px	Պատուհանի բարձրությունը
dialogWidth	թիվ px	Պատուհանի լայնությունը
dialogLeft	թիվ px	Հորիզոնական դիրքը
dialogTop	թիվ px	Ուղղահայաց դիրքը
font-family	ոճի տող	Տառաչարի տեսակը
font-size	ոճի տող	Տառաչարի չափաքան
font-style	ոճի տող	Տառաչարի ոճը (ուղիղ, շեղ)
font-weight	ոճի տող	Թավությունը
help	yes no	help կոճակի ընդգրկումը
maximize	yes no	maximize կոճակի ընդգրկումը
minimize	yes no	minimize կոճակի ընդգրկումը

Օրինակ եթե գրանցենք՝ **history.go(-3)**, ապա կկատարվի անցում երեք քայլ առաջ բեռնված էջին:

location օբյեկտի հատկությունները բնութագրում են ընթացիկ էջի URL հասցեի տարբեր մասերին վերաբերվող ինֆորմացիան՝

- ◆ href – լրիվ URL հասցեն տողի տեսքով,
- ◆ hash – տողի # սիմվոլից հետո գրանցված մասը,
- ◆ host – հասցեի “հոստ-պորտ” մասը,
- ◆ hostname – հասցեի “հոստ” մասը,
- ◆ pathname – դեպի օբյեկտ տանող ճանապարհի մասը, երրորդ / նշանից հետո
- ◆ port – պորտի համարը,
- ◆ protocol – հասցեի սկզբնական՝ արձանագրությունը բնորոշող մասը,
- ◆ search – հարցման տողը կամ URL-ի տվյալները ? նշանից հետո:

Հատկանշական է, որ location օբյեկտի href հատկությունը կարելի է օգտագործել սցենարում նոր էջերին անցում կատարելու համար: Օրինակ, եթե գրանցվի հետևյալ հրամանը՝

location.href="URL հասցե";

ապա բրաուզերի պատուհանում կբեռնվի նշված հասցեով գտնվող էջը: Նույն արդյունքին կարելի է հասնել օգտագործելով window օբյեկտի navigate մեթոդը՝

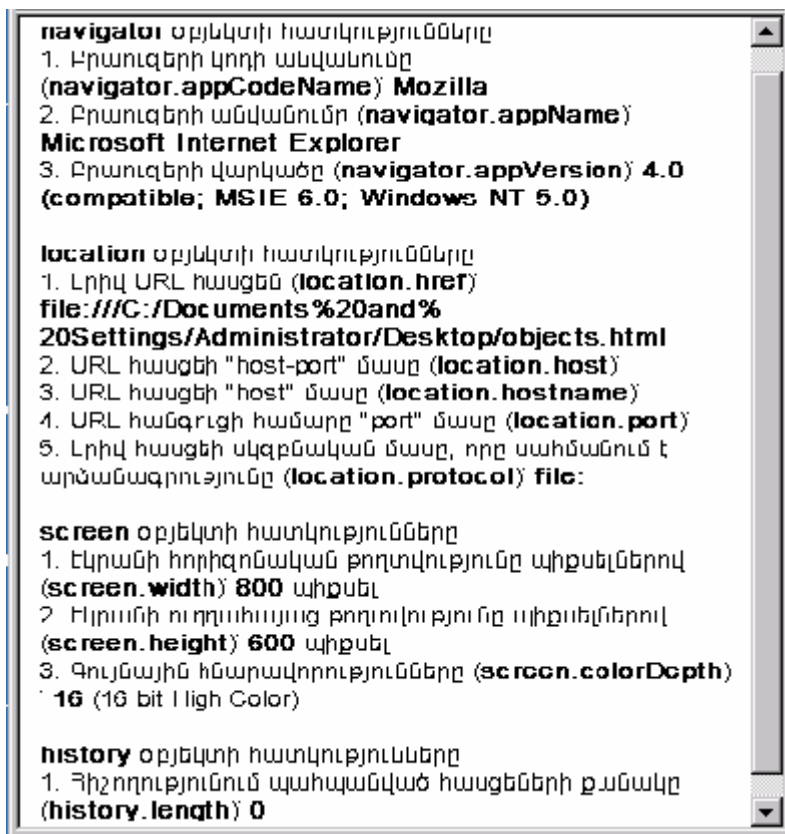
window.navigate("URL հասցե");

navigator օբյեկտը նկարագրում է ինֆորմացիան բրաուզերի ծրագրի արտադրողի, վարկածի և հնարավորությունների վերաբերյալ: Նշենք հետևյալները՝

- ◆ appCodeName - բրաուզերի կոդի անվանումը,
- ◆ appName - բրաուզերի անվանումը,
- ◆ appVersion - բրաուզերի վարկածը:

Պատկեր 3.5.4-ում ցուցադրված են քննարկված օբյեկտների որոշ հատկությունների համար ստացված արժեքները կոնկրետ բրաուզերի համար (մեր դեպքում դա Internet Explorer-ն է):

Ինչպես կարելի է տեսնել, location օբյեկտի host, hostname և port հատկությունների արժեքները բացակայում են, քանի որ ծրագիրը կատարվում է լոկալ համակարգչի վրա (սովորական file արձանագրությունով): history օբյեկտի length հատկության արժեքը հավասար է զրոյի, քանի որ էջը չի կանչվել այլ էջերից և դրանից այլ էջերի կանչեր նույնպես չեն իրագործվել:



Պատկեր 3.5.4. window օբյեկտի ենթաօբյեկտների որոշ հատկությունները

event օբյեկտը հնարավորություն է ընձեռնում ստանալ և հարկ եղած դեպքում օգտագործել սցենարում բազմաթիվ տեղեկություններ կատարվող իրադարձությունների անցման վերաբերյալ: Այդուսակ 3.5.5-ում բերված է event օբյեկտի հատկությունների ցուցակը (մեթոդներ այն չունի): Թավ թառաշարով առանձնացված են փոփոխության հնարավորություն ունեցող հատկությունները: Պատկեր 3.5.5-ում բերված է event օբյեկտի մի քանի հատկությունների ընթերցման օրինակ:

Աղյուսակ 3.5.5.

event օբյեկտի մեթոդները

Հատկությունը	Նկարագրությունը
altKey	Վերադարձնում է Alt ստեղծի վիճակը
button	Մկնիկի ստեղծը, որն առաջացրել է իրադարձ.
cancelBubble	Արգելում կամ թույլատրում է իրադարձության անցումը հիերարխիայով վեր
clientX	Էլեմենտի X կոորդինատը բացառելով շրջանակները, դաշտերը, պտտաժապավենները և այլն
clientY	Էլեմենտի Y կոորդինատը բացառելով շրջանակները, դաշտերը, պտտաժապավենները և այլն
ctrlKey	Վերադարձնում է Ctrl ստեղծի վիճակը
from Element	Վերադարձնում է էլեմենտը, որից անցել է մկնիկի նշիչը (mouseover և mouseout իրադարձությունների համար)
keyCode	Սեղմած ստեղծի ASCII կոդը
offsetX	Էլեմենտի X կոորդինատը պարունակող կոնտեյների վերաբերյալ
offsetY	Էլեմենտի Y կոորդինատը պարունակող կոնտեյների վերաբերյալ
reason	Ցուցադրում է իրադարձության անցման հաջողությունը կամ անհաջողության պատճառը
returnValue	Որոշում է վերադարձվող արժեքը
screenX	Էլեմենտի X կոորդինատը բրաուզերի էկրանի վերաբերյալ
screenY	Էլեմենտի Y կոորդինատը բրաուզերի էկրանի վերաբերյալ
srcElement	Հիերարխիայում ամենացածր էլեմենտը, որում կատարվում է իրադարձությունը
toElement	Վերադարձնում է էլեմենտը, որին տարվում է մկնիկի նշիչը
type	Վերադարձնում է իրադարձության անվանումը
x	Մկնիկի նշիչի X կոորդինատը
y	Մկնիկի նշիչի Y կոորդինատը

Պատկեր 3.5.5-ում արտապատկերված է այն պահը, երբ տեղի է ունեցել keydown իրադարձությունը՝ սեղմած է Alt ստեղծը (event.altKey=true, Alt ստեղծի կոդը՝ event.keyCode=18): Մկնիկի նշիչի կոորդինատները բրաուզերի պատուհանի ներքին մասի ձախ

վերին անկյան վերաբերյալ այդ պահին կազմում են eventClientX=348 և eventClientY=159 (նշիչի դիրքը ցուցադրված է պատկերում ձեռքի ցուցամատով): Բացատրող տեքստը և տեքստի մուտքագրման դաշտերը տեղադրված են աղյուսակում՝ դա թույլ է տալիս ներկայացնել փաստաթուղթը ավելի դյուրին ընկալվող տեսքով: Փաստաթղթի ծրագրային կոդը բերված է Ծր. 3.5.2-ում:

Մկնիկի նշիչի հորիզոնական կոորդինատը (event.clientX)	348
Մկնիկի նշիչի ուղղահայաց կոորդինատը (event.clientY)	159
Սեղմած ստեղծի ծածկագիրը (event.keyCode)	18
Սեղմած է "Alt" ստեղծը (event.altKey)	true
Սեղմած է "Ctrl" ստեղծը (event.ctrlKey)	false
Սեղմած է "Shift" ստեղծը (event.shiftKey)	false
Տեղի ունեցած իրադարձության անվանումը (event.type)	keydown

Պատկեր 3.5.5. event օբյեկտի որոշ հատկությունների կիրառության օրինակ

Ծր.3.5.2. event օբյեկտի հատկությունների օրինակ

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www. w3.org/1999/xhtml ">
<head>
```



```

<title>Event object</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script language="JavaScript">
<!--
function fMouseMove()
{
document.all("coorx").value=event.clientX;
document.all("coory").value=event.clientY;
document.all("itype").value=event.type;
window.event.cancelBubble=true;
}
function fKeyDown()
{
document.all("ikeyCode").value=event.keyCode;
document.all("ialtKey").value=event.altKey;
document.all("ictrlKey").value=event.ctrlKey;
document.all("ishiftKey").value=event.shiftKey;
document.all("itype").value=event.type;
document.all("iscrElement").value=window.event.scrElement;
window.event.cancelBubble=true;
}
//-->
</script>
</head></td><tr>
<body >
<table name="tablica" id="tablica" border="2"
style="position:absolute;top:5px;left:5px;width:350px;height:150px"
onkeydown="fKeyDown()" onmousemove="fMouseMove()">
<tr><td>Մկնիկի նշիչի հորիզոնական կոորդինատը
<b>(event.clientX)</b></td><td><input type="text" id="coorx"
/></td><tr>
<tr>
<td>Մկնիկի նշիչի ուղղահայաց կոորդինատը
<b>(event.clientY)</b></td><td><input type="text" id="coory" /></td>
<tr>
<tr><td>Սեղմած ստեղծի ծածկագիրը
<b>(event.keyCode)</b></td><td><input type="text" id="ikeyCode"
/></td><tr>
<tr><td>Սեղմած է "Alt" ստեղծը
<b>(event.altKey)</b></td><td><input type="text" id="ialtKey"/>
</td>

```

```

<tr>
<tr>
<td>Ստեղծած է "Ctrl" ստեղծողը <b>(event.ctrlKey)</b></td><td><input
type="text" id="ictrlKey" /></td><tr>
<tr>
<td>Ստեղծած է "Shift" ստեղծողը
<b>(event.shiftKey)</b></td><td><input type="text" id="ishiftKey"
/></td>
<tr>
<tr>
<td>Տեղի ունեցած իրադարձության անվանումը
<b>(event.type)</b></td><td><input type="text" id="itype" /></td>
<tr>
</table>
</body>
</html>

```

Սցենարը պարունակում է երկու ֆունկցիա, որոնց կանչերը կազմակերպված են `onkeydown="fKeyDown()"` և `onmousemove="fMouseMove()"` մշակիչների միջոցով: Կարելի է նկատել, որ յուրաքանչյուր ֆունկցիայի վերջին հրամանն է՝

`window.event.cancelBubble=true;`

`cancelBubble` հատկությունը թույլ է տալիս արգելել իրադարձության «անցումը» դեպի վեր հիերարխիայի սանդղակով: Սովորաբար նշված հատկությունը օգտագործվում է այն դեպքերում, երբ տվյալ էլեմենտը ներդրված է մի այլ էլեմենտում (կոնտեյներում) և միևնույն իրադարձության մշակումը դրանցից յուրաքանչյուրի համար պետք է կատարվի յուրովի: Պարզաբանելու նպատակով կազմենք հետևյալ աղյուսակը՝

```

<table onkeydown=" fKeyDownTbl()">
<tr><td>Մուտքագրեք ձեր անունը</td>
<td><input type="text" onkeydown="fKeyDownInp()"</td>
</tr></table>

```

Բերված ծրագրային հատվածում՝ և աղյուսակի, և ներդրված տեքստի մուտքագրման պատուհանի համար ստեղծված են `onkeydown` մշակիչները: Սակայն եթե ֆոկուսում է գտնվում տեքստային պատուհանը և սեղմվում է որևէ ստեղծ, ապա պետք է կանչվի `fKeyDownInp()` ֆունկցիան, իսկ եթե ֆոկուսում է գտնվում միայն աղյուսակը (և ոչ տեքստային պատուհանը), ապա պետք է կանչվի `fKeyDownTbl()` ֆունկցիան: Ակնհայտ է, որ եթե ֆոկուսում գտնվի պատուհանը ավտոմատ կերպով ֆոկուսում է գտնվում նաև աղյուսակը և ստեղծի սեղմումը այդ պահին կարող է բերել

միաժամանակ երկու մշակիչների կատարմանը: Հենց այդպիսի իրավիճակներում էլ կիրառվում է `cancelBubble` հատկությունը: Այսինքն եթե `keyDownInp()` ֆունկցիայի ծրագրային կոդի վերջում գրանցվի հետևյալ հրամանը՝ `window.event.cancelBubble=true`, ապա այն կարգելի `keydown` իրադարձության անցումը դեպի հիերարխիայի ավելի բարձր մակարդակի էլեմենտը՝ տվյալ դեպքում աղյուսակը:

document օբյեկտը նույնպես `window` օբյեկտի “զավակներից է”, սակայն այն ծայր աստիճան “հասուն” և ինքնուրույն օբյեկտ է: Իրականում՝ JavaScript-ով սցենարներ կազմելիս, մասնավորապես ֆորմաների մշակման համար, առավել հաճախ օգտագործվող `document` օբյեկտն է, քանի որ հենց փաստաթղթերում են պահպանվում HTML էջերի բոլոր օբյեկտները և էլեմենտները:

Օբյեկտի հիմնական հատկությունների և մեթոդների հետ կարելի է ծանոթանալ հատուկ գրականությունում, իսկ այստեղ մշենք հետևյալ հատկությունները՝

- ◆ **domain** – պահպանում է այն դոմեյնի անունը, որում գտնվում է ընթացիկ փաստաթուղթը:
- ◆ **url** – պահպանում է ընթացիկ փաստաթղթի URL հասցեն:
- ◆ **referrer** – կարող է օգտագործվել այն էջի URL հասցեն, որից կատարվել է հղումը ընթացիկ էջին (այսինքն որից օգտվողը անցել է ընթացիկին):
- ◆ **lastmodified** - փաստաթղթում վերջին փոփոխությունների կատարման դատան:
- ◆ **title** - փաստաթղթի վերնագիրը (<title></title> տեգի պարունակությունը)
- ◆ **anchors** - զանգված է, որում պահպանվում են փաստաթղթում պարունակվող բոլոր հղումները (խարիսխները):
- ◆ **images** – նույնպես զանգված է: Այստեղ պահպանվում են փաստաթղթի գրաֆիկական պատկերների URL հասցեները:
- ◆ **forms** - ևս մեկ զանգված է: Այն թույլ է տալիս դիմել (ապահովում է հասանելիությունը) փաստաթղթի տարբեր ֆորմաներին:

Հատկությունների ելությունը արդեն բնութագրվում է դրանց անվանումներով, իսկ ինչ վերաբերվում է փաստաթղթի ենթաօբյեկտների զանգվածներին, ապա դրանք օգտագործվում են ինչպես և սովորական զանգվածները: Օրինակ էջի առաջին պատկերի URL հասցեն որոշելու համար, կարող ենք գրել հետևյալ հրամանը՝

alert(document.images[0].src);

Կարելի է նաև փոխել պատկերի URL-ը և կառաջանա հղում նոր գրաֆիկական ֆայլին՝

```
document.images[0].src="images/newimage.gif";
```

Օբյեկտի որոշ մեթոդներին մենք արդեն ծանոթ ենք: Օրինակ՝ `document.writeln()` և `document.write()`: Երկուսն էլ կատարում են նույն գործառնությոնը: Տարբերությունը կայանում է նրանում, որ `writeln()` մեթոդը ավտոմատ կերպով ներդնում է տողափոխության նշանը (որը միևնույն է չի ճանաչվում բրաուզերի կողմից, եթե միայն գրանցվող կողմ չի գրանցված `<pre>` տեգում):

Հնարավոր է նաև “մաքրել” ընթացիկ էջը և փաստորեն ստեղծել լիովին նորը: Այդ նպատակով օգտագործվում են `open()` և `close()` (պետք չէ շփոթել `window` օբյեկտի `open()` մեթոդի հետ, որը բացում է բրաուզերի նոր պատուհան) մեթոդները:

Սկզբում լիովին մաքրվում է ընթացիկ փաստաթուղթը՝
`document.open();`

Դրանից հետո `write()` մեթոդով գրանցվում է այն ինչ անհրաժեշտ է և `document.close()` մեթոդը օգտագործելուց հետո նոր փաստաթուղթը պատրաստ է: Պարզաբանելու համար բերենք օրինակ (Ծր. 3.5.3.):

Ծր. 3.5.3. Նոր փաստաթղթի դինամիկ կազմավորման օրինակ

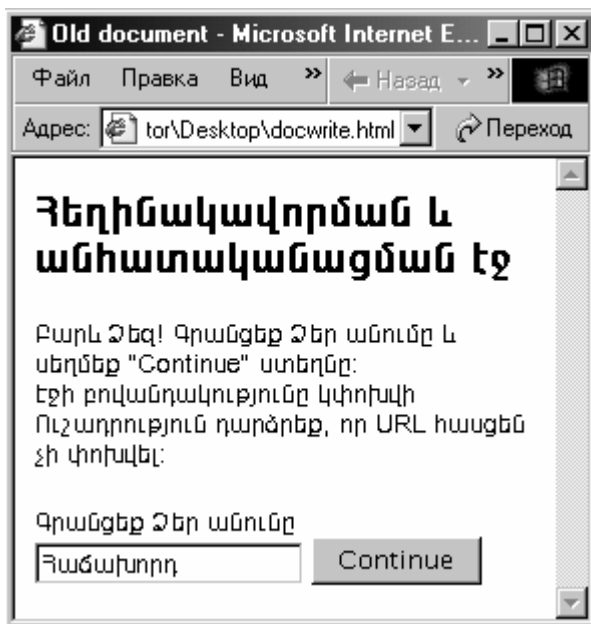
```
<html>
<head>
<title>Old document</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script language="JavaScript">
function personalizePage ()
{
var userName = personalForm.myName.value;
document.open();
document.write("<html><head><title>New document</title>
</head>");
document.write("<body><h1> Բարև " + userName + "</h1>");
document.write("<p>Այս էջը ստեղծվել է հատուկ <b>Ձեր</b>
համար</p>");
document.write("</body></html>");
document.close()
}
</script>
</head>
<body>
<h2>Հեղինակավորման և անհատականացման էջ</h2>
```

```

<p>Բարև ձեզ! Գրանցեք Ձեր անունը և սեղմեք "Continue" ստեղծը:
<br /> Էջի բովանդակությունը կփոխվի<br /></p>
<form name="personalForm">Գրանցեք Ձեր անունը
<input type="text" name="myName" id="myName" style="font-
family:Arial Armenian;font-size:10pt">
<input type="button" value="Continue" onclick="personalizePage()"
/></form></body></html>

```

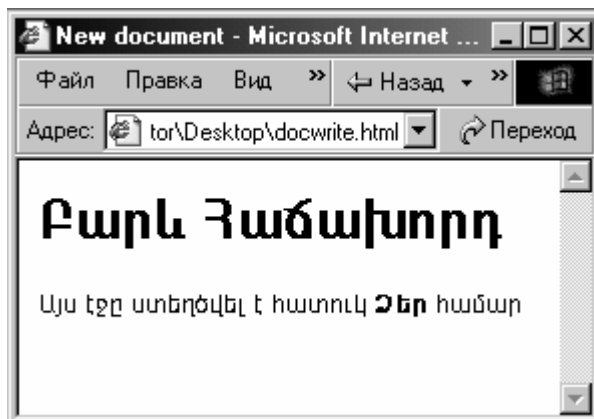
Մեկնարկային և նոր էջերը ցուցադրված են պատկերներ 3.5.7-ում և 3.5.8-ում:



Պատկեր 3.5.7. Նոր փաստաթղթի դիմամիկ ստեղծման օրինակ (մեկնարկային էջը)

Ոճերի աղյուսակների կիրառությունը և օբյեկտներին հասանելիության հնարավորությունները գործնականում թույլ են տալիս սցենարի միջոցով փոփոխել փաստաթղթի ցանկացած էլեմենտի հատկությունները: Օրինակ՝ փոխելով պատկերի (img) src հատկության արժեքը կարելի է փոխարինել մեկ պատկերը մյուսով, փոփոխելով պարբերությունում պարունակվող տեքստի տարբեր

հատկությունները (տառաշարի ընտանիքը, տառերի մեծությունը, թավությունը և այլն): Հնարավոր է փոփոխել նաև պատկերների կամ այլ էլեմենտների չափսերը, օգտագործելով width և height հատկությունները:



Պատկեր 3.5.8. Նոր փաստաթղթի դիմամիկ ստեղծման օրինակ (նոր ստեղծված էջը)

§3.6. HTML էջերի բովանդակության և տեսքի ղեկավարումը

left և **top** (էլեմենտի վերին ձախ անկյունի x և y կոորդինատները հարաբերված էկրանի նույն անկյունին) հատկությունների փոփոխությունը հնարավորություն է տալիս փոխել նաև էլեմենտների դիրքը բրաուզերի պատուհանում:

Հարկ է նշել, որ top, left, width, height հատկություններն ունեն տողային (string) ֆորմատ և դրանց արժեքների հետ ամենարեքստաբար է կատարել որևէ թվաբանական գործողություն առանց վերջին րք չափման միավորի զատելու: Սցենարներում այդ հատկությունների թվային արժեքները ավտոմատ կերպով ստանալու կամ շնորհելու նպատակով օգտագործվում է դրանց մի այլ հավաքածուն՝ posTop, posLeft, posWidth և posHeight:

Օրինակ՝ եթե շնորհենք ձախից 30 պիքսել խորությամբ տեղադրված էլեմենտի left հատկության արժեքը a փոփոխականին՝ a=element.style.left, ապա դրա արժեքը կլինի հավասար 30px: Իսկ եթե նույն փոփոխականին շնորհենք posLeft հատկության արժեքը՝ a=element.style.posLeft, ապա a-ի արժեքը կլինի հավասար պարզապես 30-ի:

z-index հատկությունը ավելացնում է նոր “չափանիւթը” (ավելի ստույգ մի շարք շերտեր) և երեք չափանի տարածության որոշակի նմանություն: Դրա միջոցով սահմանվում է թե ինչպիսի հաջորդականությամբ են առանձին շերտերը ծածկելու միմյանց: Ավելի բարձր z-index ունեցող շերտը ցուցադրվում է ավելի ցածրերի վրայ: Իրականում, քանի որ բոլոր շերտերն էլ հարթ են այդպիսի հնարավորությունը կոչվում է 2.5 չափանիւթը: Եթե էջի էլեմենտների այդ հատկությանը արժեքներ չեն շնորհված արժեքը, ապա դրանք ծածկում են միմյանց կողմն գրանցված հաջորդականությամբ (Ֆորմաների էլեմենտներն ունեն z-index-ի ավելի մեծ արժեք մյուս էլեմենտների նկատմամբ):

Էջի դիմամիկությունը ապահովող չափազանց կարևոր հատկությունն է **position**-ը, որը left և top հատկությունների հետ համատեղ թույլ է տալիս տեղադրել էլեմենտները որոշակի դիրքերում բրաուզերի պատուհանում: Այն կարող է ընդունել երեք հնարավոր արժեքներ՝

- ◆ **absolute** – նշանակում է, որ left և top արժեքները կտեղադրեն էլեմենտը համապատասխան կոորդինատներով դիրքում, հաշվարկած իրեն պարունակող կոնտեյների ծախ վերին անկյունին: Իսկ եթե էլեմենտը չունի պարունակող կոնտեյներ, ապա կոորդինատները հաշվարկվում են բրաուզերի էկրանի վերաբերյալ: Նշենք, որ էլեմենտի դիրքը կախում չունի դրա տեղի դիրքից փաստաթղթի սկզբնական կողմն:
- ◆ **relative** - այս դեպքում էլեմենտը արտապատկերվում է ըստ այն դիրքի, որը սկզբնական կողմն զբաղեցնում է դրա տեղը:
- ◆ **static** - էլեմենտը տեղադրվում է որոշակի դիրքում ֆոնի նկատմամբ և չի տեղաշարժվում էջը պտտելիս (scrolling): Հարմար է այն դեպքերում, երբ անհրաժեշտ է տեղադրել էջում վերնագիր կամ մի որևէ այլ էլեմենտ, որը պետք է արտապատկերվի մյուս էլեմենտների պտտելուց անկախ:

overflow հատկությունը կիրառվում է այն դեպքերում, երբ ներքին էլեմենտը (այդ թվում նաև տեքստը) կարող է դուրս գալ դրան պարունակող կոնտեյների սահմաններից: Այդ հատկությունը օգտագործելիս պարտադիր կերպով պետք է նախապես սահմանել պարունակող կոնտեյների չափերը width և height հատկությունների միջոցով (այսինքն նախապես սահմանվում են կոնտեյների գծային չափերը): Հատկությունը կարող է ընդունել երեք հնարավոր արժեքներ՝

- ◆ **none** - եթե էլեմենտը նույնիսկ անցնի կոնտեյների սահմաններից, միևնույն է այն կարտապատկերվի (օրինակ

տեքստի այն մասը, որը չի տեղավորվել պարունակող էլեմենտում “դուրս կգա” վերջինի սահմաններից):

- ♦ **clip** - էլեմենտի դուրս եկող մասերը կկտրտվեն:
- ♦ **scroll** – ըստ անհրաժեշտության ստեղծվում է պատման մեխանիզմը (պատաստեղներ):

visibility հատկության միջոցով կարելի է ղեկավարել էլեմենտների տեսանելիությունը դիտարկման ընթացքում: hidden արժեքը դարձնում է էլեմենտը անտեսանելի, իսկ visible՝ տեսանելի: Սակայն անկախ դրանից մյուս բոլոր էլեմենտները մնում են իրենց տեղերում:

display հատկությունը բացի էլեմենտի տեսանելիությունից ազատում կամ լրացնում է մակ դրա զբաղեցրած տեղը, այսինքն եթե գրանցվի օրինակ՝

document.all("MyElement").style.display=none;

MyElement-ը կդառնա անտեսանելի, իսկ ծրագրային կոդում դրան մախորդող և հաջորդող էլեմենտները “կկաշեն” միմյանց (եթե իհարկե դրանց դիրքավորման հատկությունը absolute չէ): Հակառակ դեպքում՝

document.all("MyElement").style.display=block,

էլեմենտը կդառնա տեսանելի և “կխցկվի” մախորդող և հաջորդող էլեմենտների միջև:

Հետևյալ օրինակը (Ծր. 3.6.1.) կօգնի ավելի լավ պատկերացնել քվարկված հատկությունների օգտագործման արդյունքները (միաժամանակ ցուցադրված են ոճերի աղյուսակների կիրառության տարբեր եղանակները):

Ծր.3.6.1. էլեմենտների դիրքավորման հատկությունների կիրառության օրինակ

```
<html>
<head><title></title>
<style>
p {font-size:10pt}
p.bluetext {font-weight:bold;color:blue}
p.reverse {font-weight:bold;color:white}
h3 {font-size:18pt;font-weight:bold;color:blue}
</style>
</head>
<body>
<div id="div1"
style="position:absolute;top:0px;left:0px;width:300px;height:100px;
margin:10px;z-index:3">
```



```

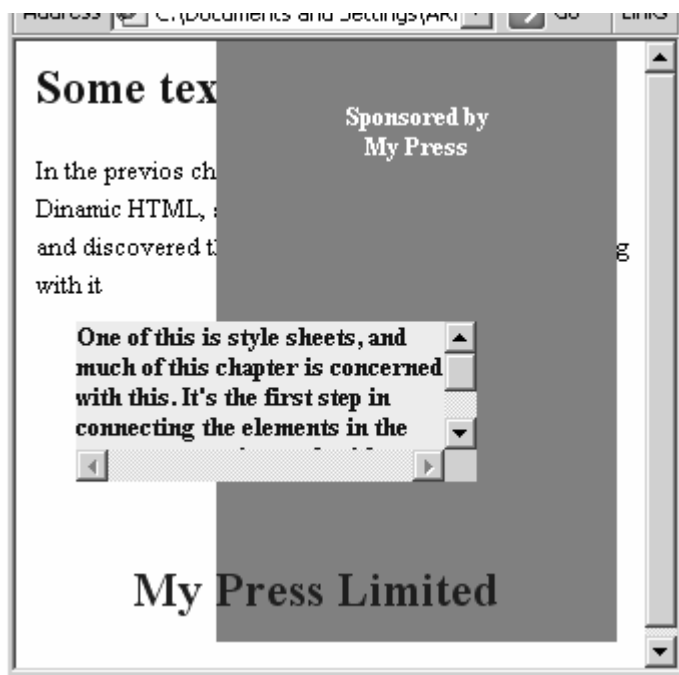
<h3>Some text about HTML</h3>
<p style="line-height:140%">In the previos chapter overwiewed
whats new in Dinamic HTML, saw some examples of dinamic pares
and discovered things we have to learn to start working with it</p>
</div>
<div id="div2"
style="position:absolute;top:0px;left:100px;width:200px;
height:300px;background-color:gray">
<center>
<p class="reverse" style="margin-top:30px">Sponsored by<br />My
Press</p></center>
</div>
<div id="div3"
style="position:absolute;top:130px;left:0px;width:200px;height:80px;
background-color:yellow;margin-top:10px;margin-left:30px;margin-
right:30px;overflow:scroll">
<p class="bluetext">One of this is style sheets, and much of this
chapter is concerned with this. It's the first step in connecting the
elements in the page to our scripts and, without scrips, the pages will
fale to come alive</p>
</div>
<div id="div4"
style="position:absolute;top:250px;left:0px;width:300px">
<center>
<h3 style="color:darkred;line-height:200%">My Press Limited</h3>
</center>
</div></body></html>

```

Պատկեր 3.6.1-ում ցուցադրված է Ծր. 3.6.1-ում բերված փոստաթղթի արտապատկերման արդյունքը: Կարելի է տեսնել, որ էջի բաժանման տրամաբանական էլեմենտները (div) ծածկում են միմյանց այն հաջորդականությամբ, որով գրանցված են ծրագրային կոդում (z-index հատկության արժեքները ըստ լռելյայն):

Բացի այդ div3 էլեմենտի overflow հատկությանը շնորհիվ է scroll արժեքը և, քանի որ դրանում պարունակվող տեքստի ծավալը գերազանցում է սահմանաված չափսերը, ապա ավտոմատ կերպով ստեղծվել են պտտաստեղներ: Պատկեր 3.6.2-ում ցուցադրված է միևնույն փոստաթուղթը, միայն div1-ի և div2-ի z-index հատկություններին շնորհիվ են ավելի մեծ արժեքներ, համապատասխանաբար 3 և 2: Դա նշանակում է, որ ցուցադրման առավել բարձր առաջնայնությունը տրված է div1-ին, ապա div2-ին,

իսկ մյուս էլեմենտների առաջնայնությունները համապատասխանում են կոդում գրանցված հաջորդականությանը:

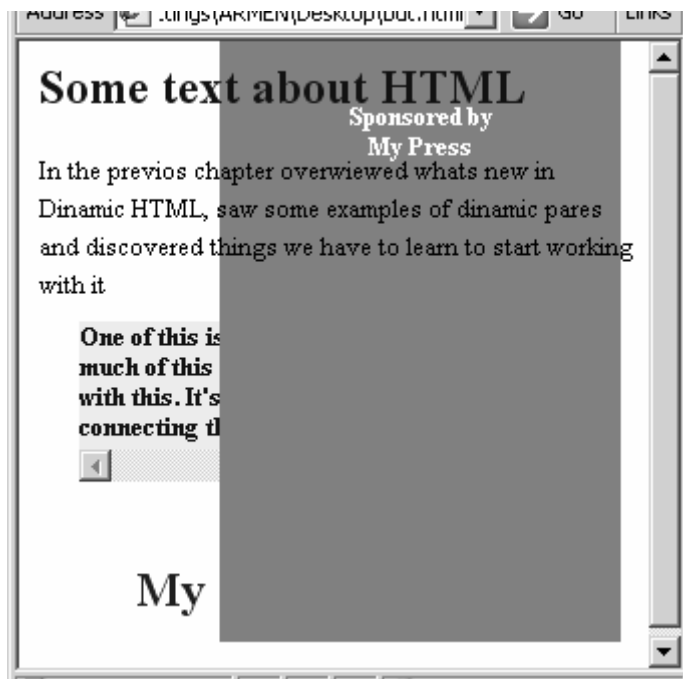


Պատկեր 3.6.1. Դիրքավորման հատկությունների կիրառության օրինակ

Բոլոր վերը նշված հատկությունների օգտագործումը սցենարում թույլ է տալիս փոփոխել էլեմենտների տեսքը, դիրքը կամ փոխարինել պատկերը մի այլ պատկերով (src հատկությունը): Սակայն գոյություն ունեն չորս հատկություններ և երկու մեթոդներ, որոնց միջոցով հնարավոր է նույնիսկ փոխարինել տեսանելի էլեմենտների մեծամասնությունը: Դրանք են՝

innerText – ընդգրակում է ամբողջ տեքստը, որը պարունակվում է տվյալ էլեմենտի ներսում, բացառելով HTML տեգերը: Օրինակ, եթե ունենք `<p id="myp">Բարև ձեզ</p>` պարբերությունը, ապա `<p>` տեգի **innerText** հատկության արժեքը (դիմումը կատարվում է հետևյալ եղանակով՝ `document.all("myp").innerText`) կլինի այն ամբողջ տեքստը, որը պարունակվում է "myp" պարբերության ներսում, այսինքն "Բարև Ձեզ" տողը: Հատկությունը կարելի է

օգտագործել նաև տեգի ներսում գրանցված տեքստը փոխարինելու համար:



Պատկեր 3.6.2. z-index հատկության կիրառության օրինակ

Օրինակ՝ `document.all("myp").innerText="Ցտեսություն"` հրամանի կատարման արդյունքում տեգը կընդունի հետևյալ տեքստը՝ `<p id="myp"> Ցտեսություն </p>`:

Սակայն եթե սկզբնական տեգում ներդրված են այլ տեգեր, ապա դրա պարունակությունը `innerText` հատկության միջոցով փոխարինելիս կվերանա ամբողջ ներքին ֆորմատավորումը (տեգերը): Օրինակ ունենք հետևյալ վերնագիրը՝

`<h1 id="heading1">Սա <i>հին</i> վերնագիրն է</h1>`:

Տվյալ տեգի `innerText` հատկության արժեքը կլինի "Սա հին վերնագիրն է" տողը (առանց HTML գծանշման `<i>` տեգի): Եթե այժմ կատարենք հետևյալ գործողությունը՝

`document.all("heading1").innerText="Իսկ սա նոր վերնագիրն է"`,

ապա կկորչի ներքին ֆորմատավորումը և տեղը կընդունի հետևյալ տեսքը՝

<h1 id="heading1">Իսկ սա նոր վերնագիրն է</h1>:

outerText – վերադարձնում է նույն արժեքը, ինչ և innerText հատկությունը: Օրինակ՝ document.all("heading1").outerText արժեքը նույնպես կլինի հավասար «Սա իմ վերնագիրն է» տողին: Սակայն երբ տվյալ հատկությունը օգտագործվում է տեգի պարունակությունը փոփոխելու հնպատակով, ապա վերանում է ոչ միայն տեգի ներքին ֆորմատավորումը, այլ և հենց ինքը տեգը: Այժմ եթե կատարվի հետևյալ հրամանը՝

document.all("heading1").outerText="Իսկ սա նոր վերնագիրն է",

ապա կվերանա ամբողջ <h1> տեգը և դրա փոխարեն պարզապես կգրանցվի «Իսկ սա նոր վերնագիրն է» տողը:

innerHTML – որպես արժեք ստացվում է էլեմենտի ներսի ամբողջ տեքստը և HTML գծանշումը: Եթե ստանաք «heading1» վերնագրային տեգի innerHTML հատկության արժեքը՝

document.all("heading1").innerHTML,

ապա այն կլինի հավասար «Սա <i>իմ</i> վերնագիրն է» տողին, որն ինչպես տեսնում ենք պարունակում է <h1> տեգի ներսում գրանցված տեքստը, և HTML գծանշումը: Տվյալ հատկությունը թույլ է տալիս կատարել էջում նոր գծանշում: Օրինակ կատարելով հետևյալ գործողությունները՝

document.all("heading1").innerHTML="Իսկ սա <i>նոր</i> վերնագիրն է: Այստեղ մենք ընդգրկել ենք նաև այս նկարը":

Արդյունքում <h1>բացող և </h1> փակող տեգերի միջև կգրանցվի նոր տեքստը և, որը առավել կարևոր է կպահպանվի տողում բերված HTML գծանշումը (այսինքն «նոր» բառը կարտապատկերվի շեղ իսկ «այս» բառից հետո կարտապատկերվի «mylmg.gif» պատկերը):

outerHTML – ընդգրկում է էլեմենտում պարունակվող ամբողջ տեքստը և HTML գծանշումը այդ թվում նաև բացող և փակող տեգերը: Օրինակ նույն «heading1» վերնագրային տեգի outerHTML հատկության արժեքն է՝

«<h1 id="heading1">Սա <i>իմ</i> վերնագիրն է</h1>» տողը:

Դա նշանակում է, որ եթե հատկությանը շնորհիվ նոր արժեք, ապա անհրաժեշտության դեպքում կարելի է փոփոխել ամբողջ տեգը: Օրինակ եթե գրանցենք՝

document.all("heading1").outerHTML="<p>Այժմ վերնագրի փոխարեն գրանցված է պարբերություն</p>",

ապա վերնագրի փոխարեն կարտապատկերվի բերված պարբերությունը:

Ծրագիր 3.6.2-ում ցուցադրված են վերը թվարկված հատկությունների կիրառության օրինակները (տես՝ պատկերներ 3.6.3-ում և 3.6.4):

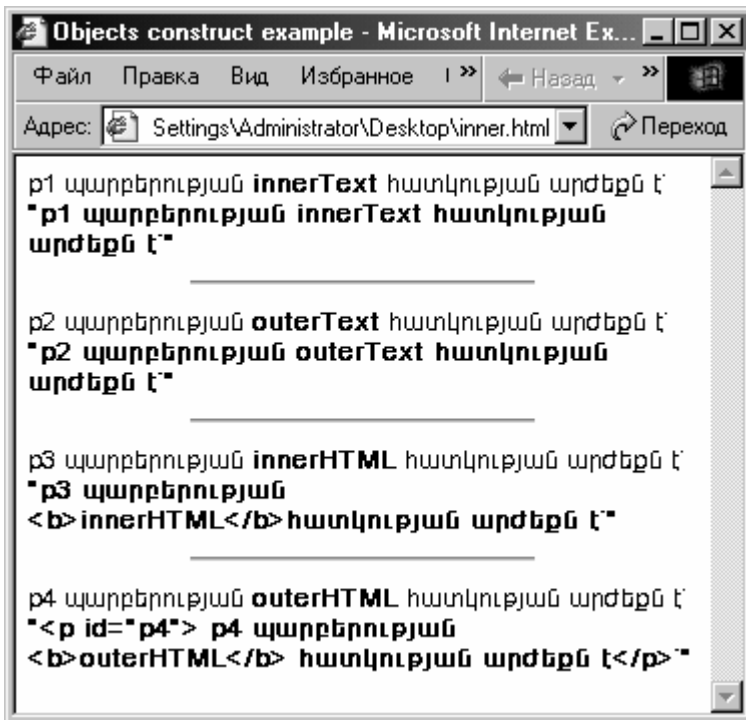
Ծր. 3.6.2. inner և outer հատկությունների կիրառության օրինակ

```
<html>
<head><title>Objects construct example</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<style>
p {margin:10px;cursor:hand}
</style>
<script language="JavaScript">
<!--
function fUpdateProperties(obj)
{
document.all("p1").innerText="Սա նոր p1 պարբերությունն է
<b>innerText</b> հատկությունը փոփոխելուց հետո";
document.all("p2").outerText="Ինչպես տեսնում ենք p3 պարբերության
գծանշումը <b>outerText</b> հատկությունը փոփոխելուց հետո
վերացել է";
document.all("p3").innerHTML="<b>innerHTML</b> հատկությունը
փոփոխելուց հետո փոփոխվում է էլեմենտի (տվյալ դեպքում
<b>p3</b> պարբերության) միայն ներքին գծանշումը";
document.all("p4").outerHTML="<b>outerHTML</b> հատկությունը
փոփոխելուց հետո լիովին փոփոխվում է սկզբնական էլեմենտի (p4
պարբերության) <i>և ներքին, և արտաքին </i>գծանշումը:";
}
// -->
</script>
</head>
<body topmargin="2" leftmargin="2" onclick="fUpdateProperties()">
<p id="p1">p1 պարբերության <b>innerText</b> հատկության
արժեքն է <b>"p1 պարբերության innerText հատկության արժեքն
է"</b></p>
<hr width="50%" />
<p id="p2">p2 պարբերության <b>outerText</b> հատկության
արժեքն է <b>"p2 պարբերության outerText հատկության արժեքն
է"</b></p>
<hr width="50%" />
```

```

<p id="p3">p3 պարբերության <b>innerHTML</b> հատկության
արժեքն է <b>"p3
պարբերության
<b>innerHTML</b>";հատկության արժեքն է"</b></p>
<hr width="50%" />
<p id="p4">p4 պարբերության <b>outerHTML</b> հատկության
արժեքն է <b>"<p id="p4">p4
պարբերության
<b>outerHTML</b>";
հատկության
արժեքն
է"</b></p>
</body>
</html>

```



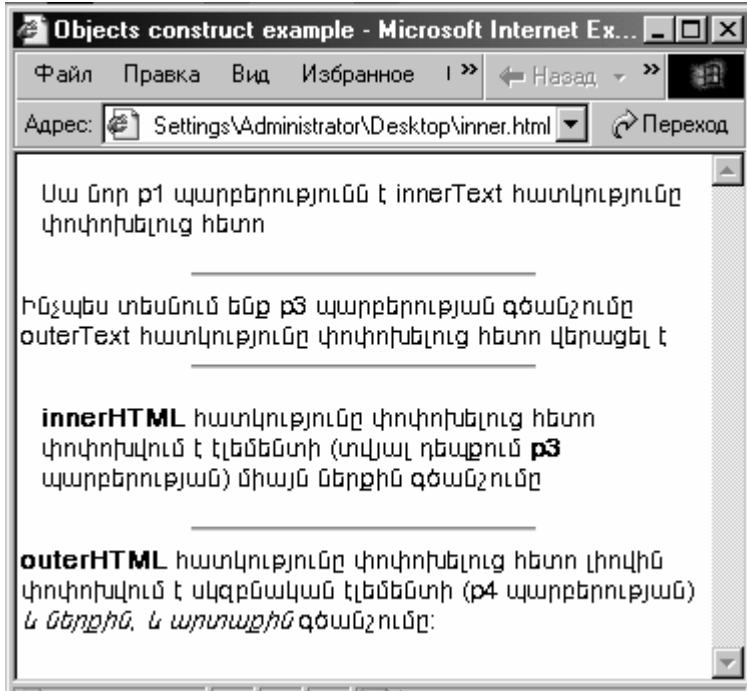
Պատկեր 3.6.3. inner և outer հատկությունների սկզբնական արժեքները

insertAdjacentText() և **insertAdjacentHTML()** մեթոդները օգտագործվում են էլեմենտի ներսում կամ որևէ կողմից, համապատասխանաբար, տեքստը կամ HTML զծանշումը մուտքագրելու

համար, ընդ որում գոյություն ունեցող տեքստը կամ գծանշումը չի փոփոխվում (“վնասվում”): Դրանց գրանցման քերականությունը հետևյալն է՝

insertAdjacentText(դիրքը, ավելացվող_տեքստը),

insertAdjacentHTML(դիրքը, ավելացվող_գծանշումը):



Պատկեր 3.6.4. Պարբերությունների տեսքը inner և outer հատկություններին նոր արժեքներ շնորհելուց հետո

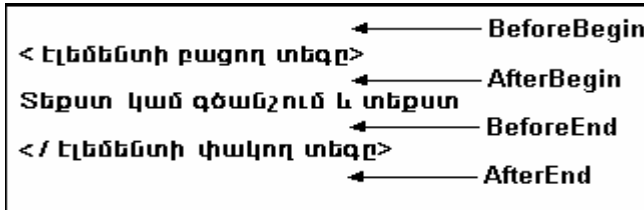
“Դիրքը” արգումենտի միջոցով որոշվում է թե որտե՞ղ պետք է ավելացվի տեքստը (կամ գծանշումը): Այն կարող է ընդունել չորս հնարավոր արժեքներ՝

- ♦ **BeforeBegin** – տեքստը կամ գծանշումը տեղադրվում է անմիջապես էլեմենտի բացող տեգի արջև:
- ♦ **AfterBegin** - տեքստը կամ գծանշումը տեղադրվում է էլեմենտի բացող տեգից հետո:
- ♦ **BeforeEnd** – տեքստը կամ գծանշումը տեղադրվում է անմիջապես էլեմենտի փակող տեգի արջև:

- ♦ **AfterEnd** - տեքստը կամ գծանշումը տեղադրվում է էլեմենտի փակող տեգից հետո:

Պատկեր 3.6.5-ում ցուցադրված է առաջին արգումենտի յուրաքանչյուր արժեքին համապատասխանող ներմուշվող տեքստի կամ HTML կոդի տեղաբաշխումը:

Երկրորդ արգումենտի արժեքը այն տեքստն է կամ HTML կոդը, որը պետք է գրանցվի առաջին արգումենտի միջոցով սահմանված տեղում:



Պատկեր 3.6.5. insertAdjacentText() և insertAdjacentHTML() մեթոդների առաջին արգումենտի արժեքները

Մեթոդների կիրառության օրինակը բերված է Ծր. 3.6.3-ում և պատկերներ 3.6.6. և 3.6.7-ում:

Ծր. 3.6.3. insertAdjacentText() և insertAdjacentHTML() մեթոդների կիրառության օրինակ

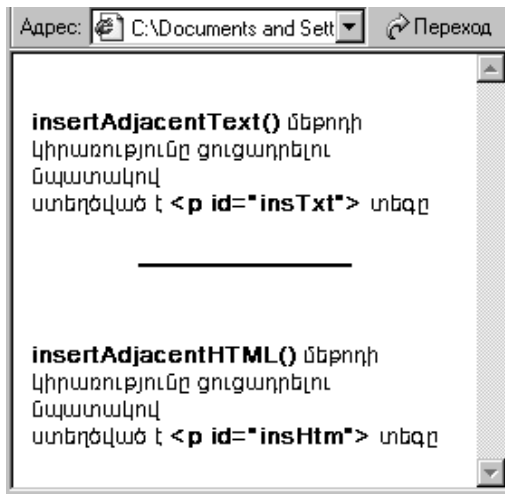
```
<html>
<head>
<style>
p {font-family:Arial Armenian;font-size:10pt}
hr {width:50%;color:darkblue}
</style>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script language="JavaScript">
<!--
function fDisplayMethods()
{var objTxt=document.all("insTxt");
var objHtm=document.all("insHtm");
objTxt.insertAdjacentText("BeforeBegin","Սա BeforeBegin արժեքին
համապատասխանող դիրքն է");
objTxt.insertAdjacentText("AfterBegin","Սա AfterBegin արժեքին
համապատասխանող դիրքն է");
```



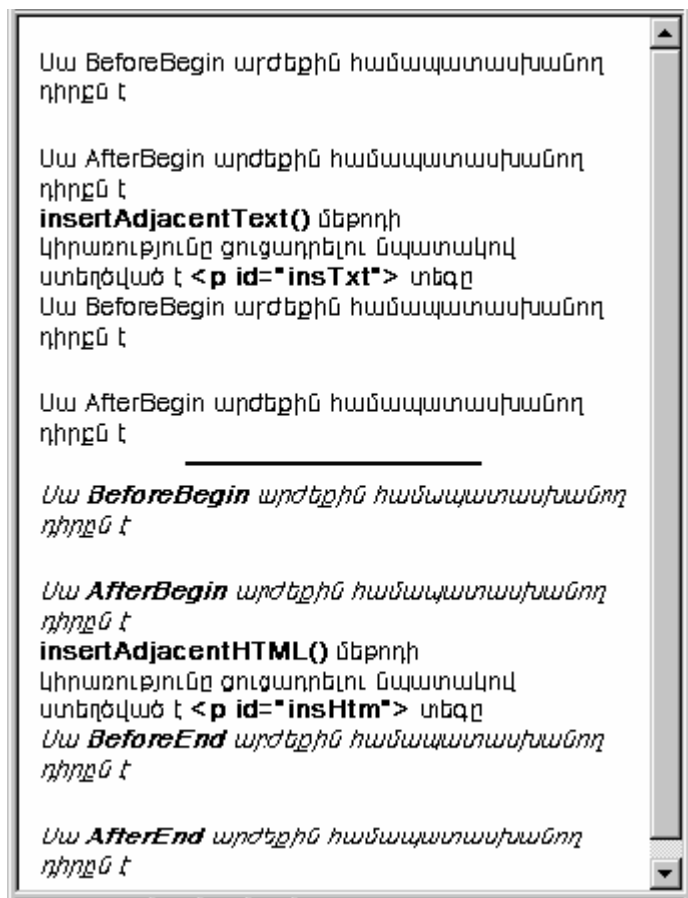
```

objTxt.insertAdjacentText("BeforeEnd","Սա BeforeBegin արժեքին
համապատասխանող դիրքն է");
objTxt.insertAdjacentText("AfterEnd","Սա AfterBegin արժեքին
համապատասխանող դիրքն է");
objHtm.insertAdjacentHTML("BeforeBegin","<i>Սա
<b>BeforeBegin</b> արժեքին համապատասխանող դիրքն է</i>");
objHtm.insertAdjacentHTML("AfterBegin","<i>Սա <b>AfterBegin</b>
արժեքին համապատասխանող դիրքն է</i>");
objHtm.insertAdjacentHTML("BeforeEnd","<i>Սա <b>BeforeEnd</b>
արժեքին համապատասխանող դիրքն է</i>");
objHtm.insertAdjacentHTML("AfterEnd","<i>Սա <b>AfterEnd</b>
արժեքին համապատասխանող դիրքն է</i>");} //--> </script>
</head><body onclick="fDisplayMetods()">
<p id="insTxt"><br /><b>insertAdjacentText()</b> մեթոդի<br />
կիրառությունը ցուցադրելու նպատակով<br /> ստեղծված է
<b>&lt;p id="insTxt"&gt;</b> տեղը<br /><p> <hr />
<p id="insHtm"><br /><b>insertAdjacentHTML()</b> մեթոդի<br />
կիրառությունը ցուցադրելու նպատակով<br /> ստեղծված է
<b>&lt;p id="insHtm"&gt;</b> տեղը<br /></p>
</body>
</html>

```



Պատկեր 3.6.6. Փաստաթղթի տեսքը մինչև insertAdjacent մեթոդների կիրառումը



Պատկեր 3.6.7. Փաստաթղթի տեսքը մինչև insertAdjacent մեթոդների կիրառումից հետո

§3.7. Նոր՝ սովորական և մոդալ պատուհանների ստեղծումը

Շարունակենք այժմ նոր պատուհանների բացման հարցի քննարկումը: Այն պարզաբանելու համար բերենք օրինակ: Կազմենք հետևյալ մեկնարկային էջը (Ծր. 3.7.1.), անվանենք այն start-page.html և պահպանենք որևէ (օրինակ My Documents) թղթապանակում:

Ծր. 3.7.1. open() և showModalDialog() մեթոդների կիրառությանը

```

<html>
<head><title>open() and showModalDialog() methods example</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script language="JavaScript">
<!--
function fOpenNewWindow()
{
window.status="";
wndFeatures="top=20,left=20,width=380,height=250,toolbar=yes,
menubar=yes,directories=no";
newWindow=window.open("newdoc.html","",wndFeatures);
window.status="Opened a new browser window";
window.event.cancelBubble=true;
window.event.returnValue=false;
}
function fOpenNewDialog()
{
window.status="";
dlgFeatures="dialogWidth=380px;dialogHeight=250px;scrollbars=no;
center=yes;border=thin;help=no;status=no";
bookTitle=window.showModalDialog("newDialog.html","MyDialog",
dlgFeatures);
window.status="Opened a new dialog window";
if(bookTitle!=null&&bookTitle!="")
{
objList=document.all("divSelect");
objList.insertAdjacentHTML("AfterEnd","<li>"+bookTitle+"</li>")
}
window.event.cancelBubble=true;
window.event.returnValue=false;
}
//-->
</script>
</head>
<body>
<center>
<div style="text-align:center;width:100%;height:20px;font-size:12pt">
<b>window.open()</b> և <b>window.showModalDialog()</b> մեթոդ-
ների կիրառության օրինակ</div>
<hr width="50%" />
<ul id="ListTag">

```


</center>

Նոր պատուհանները բացելու նպատակով ստորև ցուցադրված հիպերհղումներում ստեղծված են onclick մշակիչներ:

<hr />

Բացենք բրաուզերի նոր պատուհան գրքերի ցուցակը դիտարկելու համար

Բացենք նոր մոդալ պատուհան գիրք ընտրելու համար

<hr width="50%" />

Դուք ընտրել էք հետևյալ գրքերը՝

<ul id="ListTag">

</body>

</html>

Բերված օրինակում բրաուզերի նոր ոչ մոդալ պատուհան և նոր երկխոսության մոդալ պատուհաններ բացելու նպատակով ստեղծված են երկու հղումներ (դա կարելի է իրագործել նաև համապատասխան կոճակների օգտագործման միջոցով):

Բացենք նոր պատուհան գրքերի ցուցակը դիտարկելու համար

Բացենք նոր մոդալ պատուհան գիրք ընտրելու համար

Հղումների href ատրիբուտներին շնորհիվ են դատարկ արժեքներ, քանի որ մենք ոչ թե ցանկանում ենք բեռնել նոր փաստաթուղթ ընթացիկ պատուհանում, այլ ստեղծել նոր պատուհաններ: Դրա փոխարեն յուրաքանչյուր հղումի համար ստեղծված է onclick մշակիչ: Առաջինը կանչում է բրաուզերի նոր պատուհան բացող (onclick="fOpenNewWindow()"), իսկ երկրորդը՝ նոր երկխոսության պատուհան բացող (onclick="fOpenNewDialog()") ֆունկցիաները:

- Երբ օգտվողը դիմում է հղումին (կտտացնում է մկնիկի ստեղծը) կապակցված ծրագրի (տվյալ դեպքում կանչվող ֆունկցիաների) կատարումից անմիջապես հետո բրաուզերը փորձում է բեռնել փաստաթուղթը, որի հասցեն նշված է href ատրիբուտում (եթե այն դատարկ է սովորաբար բացվում է My Documents թղթապանակը): Դրանից խուսափելու նպատակով օգտագործվում է window օբյեկտի returnValue հատկությունը: Երբ դրան շնորհվում է false արժեքը, ինչպես դա արվել է fOpenNewWindow() և fOpenNewDialog() ֆունկցիաներում բրաուզերը չի փորձում բեռնել նոր փաստաթուղթ:

Կարելի է նկատել, որ երկու ֆունկցիաներում էլ սկզբում մաքրվում են վիճակի տողի արժեքները՝ `window.status=""` և դրանից հետո տողային փոփոխականների տեսքով կազմավորվում են բացվող պատուհանների ձևավորման համար անհրաժեշտ հատկությունները՝

```
wndFeatures="top=20,left=20,width=380,height=250,toolbar=yes,  
menubar=yes,directories=no"; (սովորական պատուհանի համար)  
և
```

```
dlgFeatures="dialogWidth=380px;dialogHeight=200px;scrollbars=no;  
center=yes;border=thin;help=no;status=no"; (մոդալ պատուհանի  
համար):
```

Հաջորդ քայլը համապատասխանաբար `open()` և `showModalDialog()` մեթոդների օգտագործումն է նոր պատուհան բացելու նպատակով: Ֆունկցիաներին հաղորդվում են բեռնվող էջի հասցեն, որպես առաջին արգումենտի արժեք, պատուհանի անունը, որպես երկրորդ և պատուհանի հատկությունները (`features`) նկարագրող տողը, որպես երրորդ: Նոր պատուհանների վերադարձվող արժեքները շնորհիվում են համապատասխան փոփոխականներին հետագա օգտագործման նպատակով՝

```
newWindow=window.open("newdoc.html","",wndFeatures);  
bookTitle=window.showModalDialog("newDialog.html","MyDialog",  
dlgFeatures);
```

Բերված օրինակում օգտագործվում է միայն մոդալ պատուհանի վերադարձվող արժեքը (`bookTitle` փոփոխականի): Ընտրած գրքի անվանումը գրանցվում է էջի վերջում կառուցված պիտակավորված ցուցակում: Սկզբնական վիճակում ցուցակը դատարկ է՝ գրանցված են միայն բացող և փակող տեգերը՝

```
<ul id="ListTag"></ul>:
```

Մոդալ պատուհանի միջոցով ընտրված յուրաքանչյուր գրքի անվանումը լրացվում է `insertAdjacentHTML()` մեթոդով՝

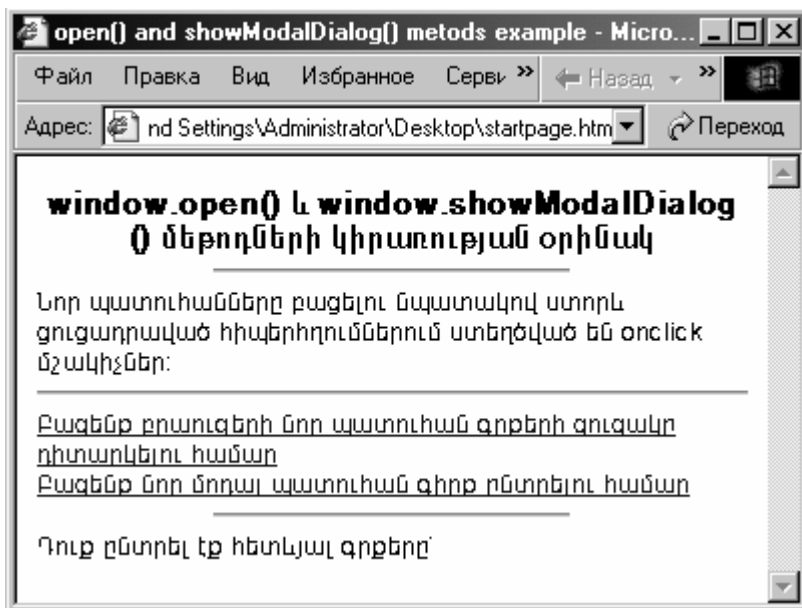
```
if(bookTitle!=null&&bookTitle!="")  
{  
objList=document.all("divSelect");  
objList.insertAdjacentHTML("BeforeEnd","<li>"+bookTitle+"</li>")  
}
```

Յուրաքանչյուր գրքի ընտրությունը կատարելիս դրա անվանումը ավելացվում է պիտակավորված ցուցակում (քանի որ ընտրված է դիրքավորման արգումենտի `BeforeEnd` արժեքը, ապա անվանումների գրանցումը կատարվում է ընտրման հաջորդականությամբ համապատասխան): Սովորական պատուհանը բացվում է միայն գրքերի ցուցակը դիտարկելու նպատակով, դրա

վերադարձվող արժեքի օգտագործումը սցենարում չի նախատեսվում: Որպեսզի բրաուզերը չփորձի ֆունկցիաների կատարումից հետո բեռնել նոր էջ (href ատրիբուտի արժեքը դատարկ է և, հետևաբար, կբացվի My Documents թղթապանակը) երկու ֆունկցիաներում էլ նախատեսվում է իրադարձության վերադարձվող կեղծ արժեքը (պետք չէ շփոթել այն պատուհանի վերադարձվող արժեքի հետ):

window.event.returnValue=false:

Պատկեր 3.7.1-ում ներկայացված է starpage.html էջի արտապատկերումը բրաուզերի պատուհանում:



Պատկեր 3.7.1. starpage.html մեկնարկային էջը

Այժմ անցնենք նոր դիմամիկ էջի ստեղծմանը, որը պետք է պարունակի դիտարկվող գրքերի ցուցակը և, կախված նրանից, թե ինչպիսի պատուհանում այն պետք է արտապատկերվի (սովորական թե երկխոսության), համապատասխան փակման և ընտրման կոճակները:

Տրամաբանորեն էջը բաժանվում է երեք տիրույթների, որոնք կառուցված են <div> տեգերի միջոցով: Գլխավոր տիրույթը (divMain) զբաղեցնում է ամբողջ պատուհանը: Դրա վերին մասում

տեղադրվում է գրքերի անվանումների մենյուն (<select id="listBooks"...>) և ընտրված գրքի շապիկի արտապատկերման համար նախատեսվող տեղը, որը սկզբնական վիճակում անտեսանելի է (visibility:hidden հատկության շնորհիվ):

Գլխավոր տիրույթի ներսում տեղադրվում են ևս երկու տիրույթ: Առաջինում (divTitle) կարմիր ֆոնի վրա արտապատկերվում է ընտրված գրքի անվանումը, իսկ երկրորդում (divText)՝ գրքի նկարագրությունը, ընդ որում դրա overflow հատկությանը շնորհիվ է scroll արժեքը: Դա նշանակում է, որ եթե տեքստի ծավալը գերազանցի տիրույթի սահմանները, ապա ավտոմատ կերպով կստեղծվեն պտտաժապավեններ տեքստը դիտարկելու համար:

Բոլոր տիրույթներն էլ ունեն բացարձակ դիրքավորում (position:absolute) և դրանց դիրքերը և չափերը ընտրված են այնպես, որ համապատասխանեն տվյալ ձեռնարկի ֆորմատին: Նոր էջի (անվանենք այն newpage.html և նույնպես տեղադրենք My Documents թղթապանակում) HTML կոդը բերված է Ծր. 3.7.2-ում:

Ծր. 3.7.2. newpage.html էջի HTML կոդը

```
<html>
<head><title>NewDocument</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<div id="divMain" style="position:absolute;top:10px;left:10px;
width:330px;height:230px">
<p style="font-family:Arial Armenian;font-size:10pt;font-
weight:bold">Ընտրեք գիրքը, որի վերաբերյալ դուք ցանկանում էք
ստանալ տեղեկություններ:
</p>
<select id="listBooks" onchange="fListChange()" style="font-
family:Arial Armenian;position:absolute;font-weight:bold;width:300px;
top:35px;left:5px">
<option value="0" selected="selected"></option>
<option value="1">Professional Active Server Pages</option>
<option value="2">ActiveX Web Database Programming </option>
<option value="3">Instant VBScript Programming</option>
<option value="4">Instant JavaScript Programming</option>
<option value="5">Professional JavaScript Programming </option>
</select>
<img src="" id="imgCover" style="position:absolute;top:65px;left:5px;
border:3px;visibility:hidden" />
```

```

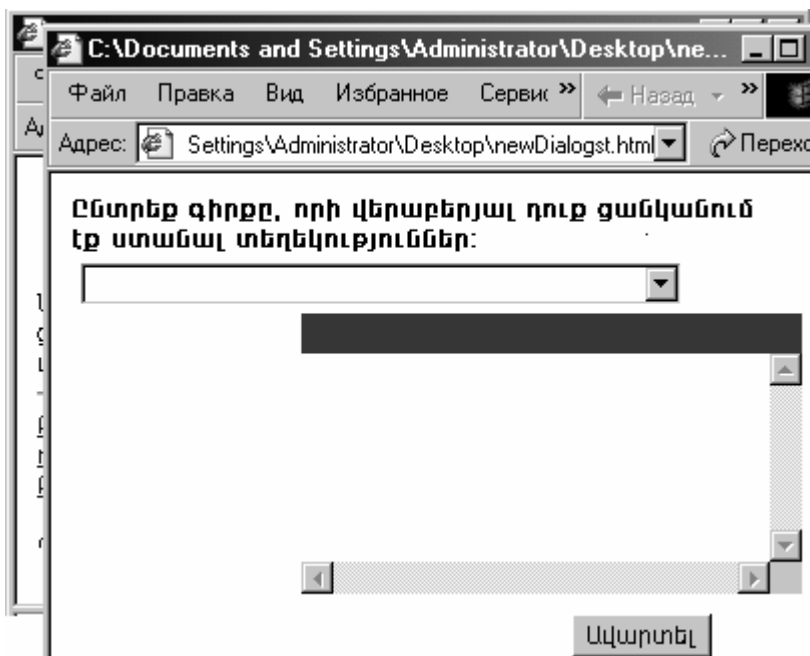
div id="divTitle"
style="position:absolute;margin:5px;top:55px;left:100px;width:250px;
height:20px;background-color:red;overflow:clip">
<p id="pTitle" style="font-size:10pt;font-weight:bold;color:white"
title="Visit our Web site for more information">
</p>
</div>
<div id="divText" style="font-family:Arial Armenian;font-size:10pt;
position:absolute;margin:5px;top:75px;left:100px;width:250px;height:
120px;overflow:scroll;text-align:justify">
</div>
</div>
<script src="newpage.js"></script>
</body>
</html>

```

Սցենարը կառուցված է, որպես առանձին ֆայլ (տես՝ Ծր. 3.7.3.) և ներդրված NewPage էջում՝ `<script src="newpage.js"></script>` (կապակցման համար օգտագործվում է `<link />` տեգը): Անվանենք սցենարի ֆայլը newpage.js (js – JavaScript ընդլայնումն է) և նույնպես տեղադրենք My Documents թղթապանակում, քանի որ հարաբերական հասցեի գրանցման եղանակը՝ `src="newpage.js"` նախատեսում է ֆայլերի տեղադրման հավասար մակարդակ:

Նշենք, որ սցենարը ներդրված է փաստաթղթի այն մասում, որտեղ պետք է արտապատկերվեն պատուհանի փակման և եթե պատուհանը մոդալ է՝ նաև գրքի ընտրության պնդման համապատասխան կոճակները և առաջին խնդիրը, որը պետք է լուծվի սցենարում՝ դա նշված կոճակների ստեղծումն է:

Այն դեպքում, երբ ստեղծվում է բրաուզերի նոր պատուհան բավական է տեղադրել ընդամենը մեկ՝ պատուհանը փակելու կոճակ: Իսկ եթե բացվելու է մոդալ պատուհան, ապա առաջանում է ևս մեկ կոճակի անհրաժեշտություն: Այդ կոճակի սեղմումը պետք է ապահովի ընտրված գրքի անվանումի վերադարձը մեկնարկային պատուհանին: Պատուհանի տեսակը ճանաչվում է window օբյեկտի dialogArguments հատկության միջոցով: `open()` մեթոդով սովորական պատուհան բացելիս երկրորդ արգումենտին շնորհիվ է դատարկ արժեք, իսկ երկխոսության պատուհանի բացման `showModalDialog()` ֆունկցիայում նոր պատուհանի “MyDialog” անունը: Հենց այդ անունն էլ շնորհիվ է `dialogArguments` հատկությանը: Պատկերներ 3.7.2. և 3.7.3-ում ցուցադրված են համապատասխանաբար նոր բացված սովորական և մոդալ պատուհանները:

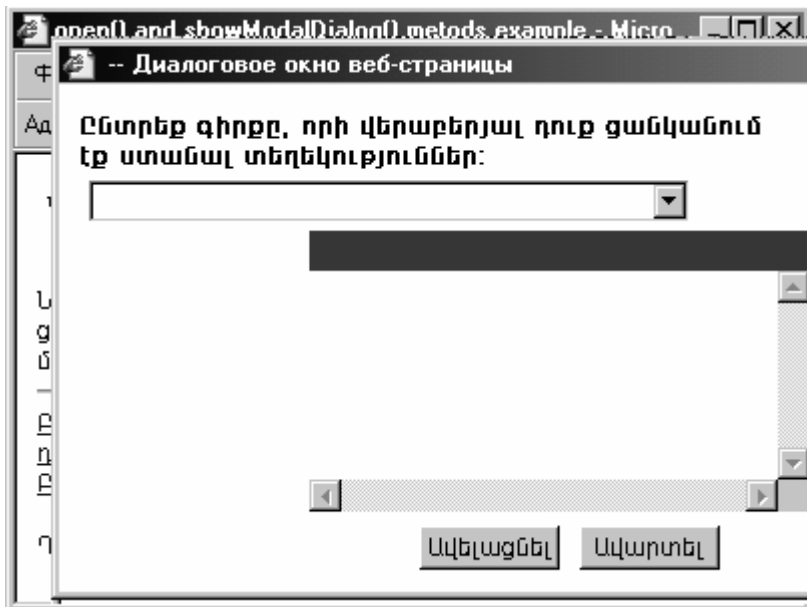


Պատկեր 3.7.2. Սովորական պատուհանը սկզբնական վիճակում

Հաջորդ քայլը սցենարի ստեղծման գործընթացում՝ օգտվողի ընտրած գրքի շապիկի և նկարագրության արտապատկերման կազմակերպումն է: Այդ նպատակով սկզբում հայտարարվում են երկու գլոբալ (որպեսզի դրանցից կարողանան օգտվել բոլոր մշակող ֆունկցիաները) փոփոխականներ՝ `strTitle`, գրքի անվանումը և `strText`, գրքի նկարագրող տեքստը պահպանելու համար: Քանի որ արտապատկերման անհրաժեշտությունը կապված է որոշակի իրադարձության հետ (փոփոխվում է մենյուի ընտրված արժեքը), ապա էջում ստեղծված է համապատասխան մշակիչը (`onchange="fListChange()"`), իսկ սցենարում՝ մշակող ֆունկցիան (`function fListChange()`):

`fListChange()` ֆունկցիայում նախ ստացվում է ընտրված գրքի ծածկագիրը՝ `strBookId=document.all("listBooks").value`, ապա (եթե գիրքն ընտրված է) տեսանելի է դարձվում գրքի շապիկը արտապատկերող նկարը: Դրանից հետո ծածկագիրը հաղորդվում է `setBookText(strBookId)` ֆունկցիային որպես արգումենտի արժեք:

setBookText() ֆունկցիան պարզապես շնորհում է արժեքներ գրքի անվանումը և նկարագրությունը պահպանող փոփոխականներին և ստացված արժեքները գրանցվում են pTitle և divText էլեմենտներում innerText մեթոդի միջոցով՝
document.all("pTitle").innerText=strTitle,
document.all("divText").innerText=strText:

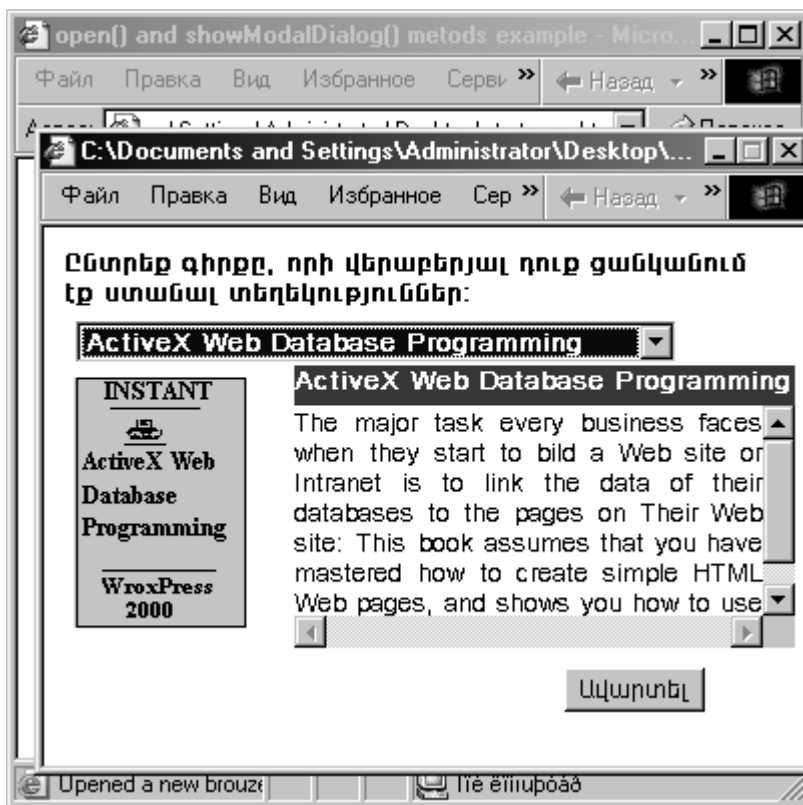


Պատկեր 3.7.3. Երկխոսության պատուհանը սկզբնական վիճակում

Պատկեր 3.7.4-ում ցուցադրված է բրաուզերի նոր պատուհանը գրքի ընտրությունից հետո:

Վերջին հարցը, որը պետք է լուծվի սցենարում՝ պատուհանների փակումն է: Քանի որ "Ավարտել" (համապատասխանում է Cancel) կոճակի սեղմումը երկու պատուհանի դեպքում էլ նշանակում է, որ գիրք ընտրված չէ, այսինքն չպետք է վերադարձվի արժեք մեկնարկային startpage.html պատուհանին, ապա երկու դեպքում էլ դիմումը կատարվում է fcancelClicked() ֆունկցիային, որը շնորհում է պատուհանի վերադարձվող արժեքին դատարկ տող և փակում է պատուհանը՝

```
window.returnValue="";  
window.close();
```



Պատկեր 3.7.4. Բրաուզերի նոր պատուհանի տեսքը, երբ գիրքն ընտրված է

Իսկ եթե գիրքն ընտրված է և օգտվողը ցանկանում է հաղորդել դրա անվանումը մեկնարկային էջին (իհարկե դա վերաբերվում է միայն երկխոսության պատուհանին), ապա սեղմվում է “Ընտրել” կոճակը, որի մշակիչը դիմում է `faddClicked()` ֆունկցիային։ Պատուհանի վերադարձվող արժեքին այդ դեպքում շնորհվում է գրքի անվանումը (որը, ինչպես հիշում ենք, պահպանվում է `strTitle` փոփոխականում) և պատուհանը փակվում է՝

```
window.returnValue=strTitle;
window.close();
```

Ստորև բերված է `newpage.js` սցենարային ֆայլի ծրագրային կոդը (Ծր. 3.7.3.):

Ծր. 3.7.3. newpage.js սցենարը

```
// JavaScript Document
var strTitle=""; // գլխավ փոփոխական գրքի անվանման համար
var strText=""; // գիրքը նկարագրող տեքստի փոփոխականը
if(window.dialogArguments=="MyDialog")
{
    // կոճակների ստեղծումը մոդալ պատուհանի համար
    stringButtons="<input title='Add book to order and return to main
window id='cmndOK' type='button' value='Ընտրել' onclick=
'addClicked()' style='font-family:Arial Armenian;position:absolute;
top:200px;left:180px;width:70px'>
<input type='button' title='Return to main window without adding
book to order'
value='Ավարտել' id='cmndCancel' onclick='cancelClick()' style='font-
family:Arial Armenian;position:absolute;
top:220px;left:260px;width:70px'>";
}
else
{
    // փակման կոճակի ստեղծումը սովորական պատուհանի համար
    stringButtons="<input type='button' title='Return to main window'
value='Ավարտել' id='cmndCancel' onclick='cancelClick()'
style='font-family:Arial Armenian;position:absolute;top:200px;
left:260px;width:70px'>";
}
// կոճակների տեղադրումը էջում
document.write(stringButtons);
//ավարտում ենք պատուհանի կոճակների ստեղծումը
function fcancelClicked()
{
    // պատուհանի փակումը երկու պատուհանների համար
    window.returnValue="";
    window.close();
}
function faddClicked()
{
    // գրքի անվանումը շնորհվում է պատուհանի վերադարձվող
    //արժեքին և պատուհանը փակվում է
    window.returnValue=strTitle;
    window.close();
}
```

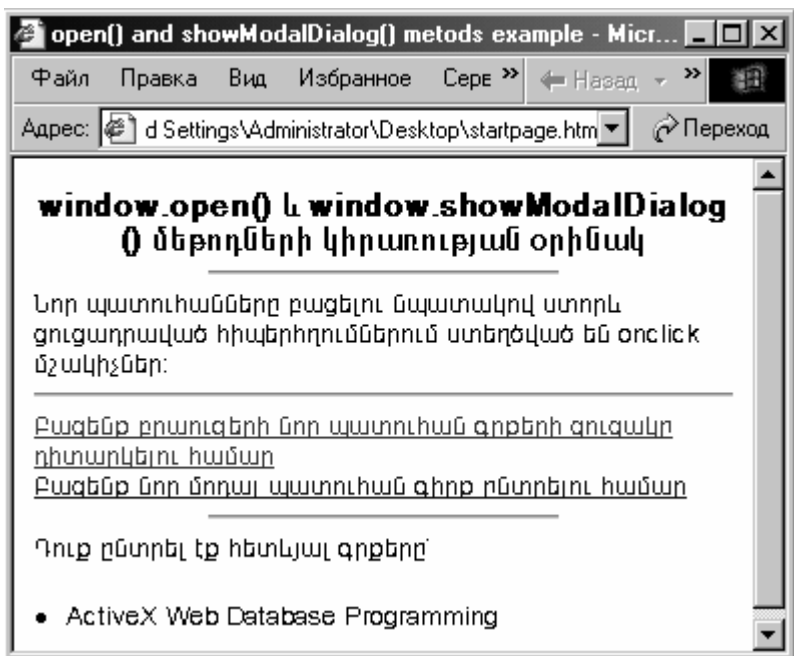
```

function BookListChange()
{
    objImgCover=document.all("imgCover");
    //ստանում ենք ընտրված գրքի ծածկագիրը
    strBookId=document.all("listBooks").value;
    if(strBookId=="0")
    {
        // եթե ընտրված չէ ոչ մի գիրք քողարկում ենք պատկերը
        objImgCover.style.visibility="hidden";
    }
    else
    {
        // տեղադրում ենք պատկերը և դարձնում այն տեսանելի
        objImgCover.src=strBookId+".gif";
        objImgCover.style.visibility="visible";
        // ստանում ենք ընտրված գրքի անվանումը և նկարագրությունը
        setBookText(strBookId);
        //տեղադրում ենք անվանումը <p id="pTitle">, իսկ նկարագրու-
        //թյունը <div id="divText"> տեղերում
        document.all("pTitle").innerText=strTitle;
        document.all("divText").innerText=strText;
    }
    function setBookText(strBookId)
    {
        //արժեքներ ենք շնորհում strTitle և strText փոփոխականներին
        if(strBookId=="1")
        {
            strTitle="Professional Active Server Pages";
            strText="Active Server Pages is simply...";
        }
        if(strBookId=="2")
        {
            strTitle=" ActiveX Web Database Programming";
            strText=" The major task every business...";
        }
        նույնը բոլոր մյուս գրքերի համար
    }
}

```

Մեկնարկային էջում երկխոսության պատուհանի վերադարձվող արժեքը ավելացվում է էջի ստորին մասում տեղադրված պիտակավորված ցուցակում: Պատկեր 3.7.5-ում ցուցադրված է

startpage.html մեկնարկային էջը գիրքն ընտրելուց և մոդալ պատուհանը փակելուց հետո:



Պատկեր 3.7.5. Մեկնարկային էջը գիրքն ընտրելուց և երկխոսության պատուհանը փակելուց հետո

§3.8. Ֆորմաների բովանդակության նախնական մշակումը

JavaScript լեզուն և դրա մշակիչները չափազանց հարմար են ֆորմաներում օգտվողների կողմից լրացվող տվյալների ճշտությունը ստուգելու համար: Իհարկե դա կարելի է անել սերվերային ծրագրերի միջոցով: Սակայն, քանի որ սերվերի հետ ինֆորմացիայի փոխանակումը հաճախ պահանջում է ժամանակի բավականաչափ մեծ ծախսեր (մանավանդ հեռախոսային կապի գծերով իրագործելու դեպքում՝ dual-up), ապա շատ ավելի հարմար է ստուգել ֆորմայի էլեմենտների լրացման ճշտությունը հենց լրացման ընթացքում՝ անմիջապես օգտվողի համակարգչի վրա:

Ֆորման, ինչպես և կամայական օբյեկտ ունի հատկություններ, որոնցից մշենք name (այն թույլ է տալիս դիմել դրան սցենարից), action, method, length և target հատկությունները: Ֆորմաներն ունեն մաս էլեմենտների հավաքածու (զանգված)՝ elements, ուր ընդգրկվում են բոլոր input, select և textarea էլեմենտները: Մեթոդներից կարելի է մշել reset() և submit()-ը:

Գործնականում կարելի է մշակել ֆորմայի կամայական էլեմենտ, սակայն առավել հաճախ դա կատարվում է տեքստային դաշտերի և մուտքագրման պատուհանների լրացման ընթացքում: Դիտարկենք օրինակ: Կազմենք փաստաթուղթ, որը պարունակում է հաճախորդի վերաբերյալ տվյալների լրացման համար ստեղծված ֆորմա և դրա մեկ էլեմենտի՝ փոստային ինդեքսի մուտքագրման պատուհանի լրացման ճշտությունը ստուգող սցենարը (Ծր. 3.8.1.):

Ծր. 3.8.1. Ֆորմայի էլեմենտների լրացման վերահսկման օրինակ

```
<html>
<head><title>Personalize page example</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<style>
input {font-family:Arial Armenian;font-size:10pt}
button {font-family:Arial Armenian;font-size:10pt}
span {font-family:Arial Armenian;font-size:11pt;font-weight:bold}
</style>
<script language="JavaScript">
<!--
function fZipCheck()
{ var zipStr=custForm.zipCode.value;
  var objAlert=document.all("spanAlert");
  if(zipStr=="")
    { objAlert.innerText="Մուտքագրեք վեցանիշ կոդ"; return (-1); }
  if(zipStr!=6)
    {objAlert.innerText="Ինդեքսը պետք է լինի վեցանիշ";return (-1); }
  return 0;
}
function fCheckSend()
{ var passCheck=fZipCheck();
  if(passCheck==1) return;
  else custForm.submit();
}
//--></script>
</head>
```

```

<body >
<h2>Խնդրում եմք լրացնել ֆորման</h2>
<form action="address.pl" method="post" name="custForm">
<table>
<tr><td>Անունը</td><td><input type="text" name="fname"></td>
</tr>
<tr><td>Ազգանունը</td><td><input type="text" name="lname">
</td></tr>
<tr><td>Հասցեն</td><td><input type="text" name="address">
</td></tr>
<tr><td>Քաղաքը</td><td><input type="text" name="city"></td></tr>
<tr><td>Փոստային ինդեքսը</td><td><input type="text"
name="zipCode" onblur="fZipCheck()"></td></tr>
<tr><td></td><td><button
onclick="fCheckSend()">Ուղարկել</button></td></tr>
<tr><td colspan="2"><span id="spanAlert"></span></td></tr>
</table>
</form>
</body></html>

```

Օր. 3.8.1-ում ֆորմայի մշակումը սկսվում է Blur իրադարձությունից, այսինքն երբ փոստային ինդեքսի մուտքագրման պատուհանը կորցնում է ֆոկուսը: Դրան համապատասխանող onblur մշակիչի միջոցով իրագործվում է fZipCheck() ֆունքցիայի կանչը, որը ստուգում է մուտքագրման պատուհանի պարունակությունը՝

```
var zipStr=custForm.zipCode.value;
```

Եթե դաշտը մնացել է դատարկ կամ լրացված է 6 նիշից պակաս կամ ավել՝ if(zipStr=="") և if(zipStr.length!=6), ապա հատուկ ստեղծված spanAlert դաշտում innerText մեթոդի միջոցով գրանցվում է համապատասխան նախազգուշացումը՝

```
objAlert.innerText="Նախազգուշացում":
```

Իհարկե օգտվողը կարող է անտեսել այդ նախազգուշացումը, շարունակել լրացումը և փորձել ուղարկել ֆորման: Այդ դեպքի համար նախատեսված է մի այլ մշակիչ, որը կանչում է ֆորմայի վերջնական ստուգման fCheckSend() ֆունկցիան, երբ օգտվողը սեղմում է "Ուղարկել" կոճակը: Եթե ինդեքսը սխալ է լրացված, ապա ֆունկցիան պարզապես վերադարձնում է օգտվողին ֆորմայի լրացման շարունակությանը (ֆորման չի ուղարկվում: Իսկ եթե ամեն ինչ կարգին է, ֆորման ուղարկվում է submit() մեթոդի միջոցով՝ custForm.submit():

Պատկեր 3.8.1-ում ցուցադրված է այն դեպքը, երբ լրացված փոստային ինդեքսի նիշերի քանակը հավասար չէ 6-ի:

Իհարկե, ծր. 3.8.1-ում ինդեքսի լրացման ստուգումը իրականաված է միայն մասամբ: Գործնականում կարելի է իրականացնել ցանկացած բարդության ստուգում: Օրինակ, կարելի է արգելել տառային սիմվոլների մուտքագրումը ինդեքսի լրացման պատուհանում, օգտագործելով String օբյեկտի charAt() մեթոդը: Այդ նպատակով fZipCheck() ֆունկցիայում ավելացնենք ծրագրային հետևյալ հատվածը՝

```
for(i=1;i<6;i++)
if(zipStr.charAt(i)<'0' || zipStr.charAt(i)>'9')
{ objAlert.innerText="Ինդեքսը պետք է պարունակի միայն թվանշաններ";
return (-1);
}
```

Պատկեր 3.8.1. Ֆորմայի լրացման վերստուգման օրինակ

Որոշ դեպքերում նույնիսկ պարտադիր չէ հասանելիությունը Ինտերնետ ծառայություններ տրամադրող կազմակերպության սերվերի CGI կատալոգին: Ստորև բերված է էլեկտրոնային ֆոր-

մայի և դրա մշակող ծրագրի օրինակ (տես՝ Ծր. 3.8.2. և պատկեր 3.8.2.):

Ծր. 3.8.2. Էլեկտրոնային ֆորմայի օրինակ

```
<html>
<head><title>Ինչպիսին է ձեր կարծիքը</title>
<style>
button {font-family:Arial Armenian;font-size:10pt}
input {font-family:Arial Armenian;font-size:10pt}
textarea {font-family:Arial Armenian;font-size:10pt}
</style>
<script>
<!--
function processForm()
{
    var newline="\n";
    var resultStr="";
    var Form1=document.form1;
    resultStr+=Form1.first_name.value+"
"+form1.last_name.value+newline;
    resultStr+=Form1.email.value+" "+newline;
    for(i=0;i<Form1.where.length;i++)
    {
        if(Form1.where[i].checked)
        {
            resultStr+=Form1.where[i].value+newline;
            break;
        }
    }
    if(Form1.desktop.checked)
        resultStr+="Սեղանի համակարգիչներ"+newline;
    if(Form1.notebook.checked)
        resultStr+="Նոութբուկներ"+newline;
    if(Form1.peripherals.checked)
        resultStr+="Պերիֆերիական սարքավորումներ"+newline;
    if(Form1.software.checked)
        resultStr+="Ծրագրային ապահովում"+newline;
    document.form2.results.value=resultStr;
    return;
}
//-->
```

```

</script>
</head>
<body>
<form name="form1" id="form1">
<table cellpadding="2" cellspacing="0">
<tr><td>Անունը</td><td><input type="text" name="first_name"
size="20" maxlength="40"></td></tr>
<tr><td>Ազգանունը</td><td><input type="text" name="last_name"
size="20" maxlength="40" /></td></tr>
<tr><td>E-mail: </td><td><input type="text" name="email" size="20"
maxlength="40" /></td></tr>
</table>
<b>Որտեղից էք դուք իմացել մեր կայքի մասին</b><br />
<input type="radio" name="where" value="Ինտերնետ"
checked="checked" />Փնտրող ծրագրից կամ Ինտերնետ
հղումից<br />
<input type="radio" name="where" value="Գովազդ" />Ռադիո կամ
հեռուստա գովազդից<br />
<input type="radio" name="where" value="Թերթ" />Թերթերից<br />
<input type="radio" name="where" value="Այլ" />Այլ
աղբյուրներից<br />
<b>Ինչպիսի արտադրատեսակների վերաբերյալ դուք
կցանկանայիք ստանալ մանրամասն ինֆորմացիա</b><br />
<input type="checkbox" name="desktop" />Սեղանի
համակարգիչներ
<input type="checkbox" name="notebook" />Նոութբուկներ
<input type="checkbox" name="peripherals" />Պերիֆերիական
սարքավորումներ
<input type="checkbox" name="software" />Ծրագրային
ապահովում<br />
<button onclick="processForm()">
Հաստատել</button>
<button name="reset" type="reset">
Մաքրել</button>
</form>
<form name="form2" id="form2"
action="mailto:survey@fakecorp.net">
<b>Ուշադիր ստուգեք գրանցումները: Եթե ամեն ինչ ճիշտ է, ստորև
կարող էք գրանցել ձեր ցանկությունները և մեկնաբանությունները
</b>
<textarea name="results" rows="5" cols="40">

```

```

</textarea>
<button name="submit" type="submit">Ուղարկել</button>
</form>
</body></html>

```

Անունը

Ազգանունը

E-mail:

Որտեղից եք դուք իմացել ձեր սայթի մասին

☐ Փնտրող ծրագրից կամ Ինտերնետ հղումից
☐ Ռադիո կամ հեռուստա գովազդից
☒ Թերթերից
☐ Այլ աղբյուրներից

Ինչպիսի արտադրատեսակների վերաբերյալ դուք կցանկանայիք ստանալ մանրամասն ինֆորմացիա

☐ Սեղանի համակարգիչներ ☒ Նոութբուկներ
☐ Պերիֆերիական սարքավորումներ ☐ Ծրագրային ապահովում

Ուշադիր ստուգեք գրանցումները: Եթե անեն ինչ ճիշտ է, ստորև կարող եք գրանցել ձեր ցանկությունները և մեկնաբանությունները

Պատկեր 3.8.2. Սցենարի միջոցով մշակվող ֆորմայի օրինակ

Բերված օրինակում սցենարի միջոցով առաջին ֆորմայում լրացված տվյալները տեքստի տեսքով տեղադրվում են երկրորդ ֆորմայի տեքստային դաշտում: Դրանից հետո օգտվողը կարող է

ուղղումներ մտցնել, ավելացնել մեկնաբանություններ և այլն: Համոզվելով, որ ամեն ինչ ճիշտ է լրացված, օգտվողը սեղմում է “Ուղարկել” կոճակը և ամբողջ կուտակված ինֆորմացիան ուղարկվում է էլեկտրոնային փոստով:

§3.9. ActiveX ղեկավարման էլեմենտների օգտագործումը

Գոյություն ունեն էֆեկտներ, որոնք հնարավոր չէ իրագործել միայն HTML-ի միջոցներով: Այդպիսի դեպքերում սովորաբար օգտագործվում են ActiveX ղեկավարման էլեմենտները կամ Java ապլետները: Դրանց կիրառությունը թույլ է տալիս ներդնել HTML էջերում արտաքին օբյեկտներ կամ կապ հաստատել տվյալների բազաների հետ: Որպեսզի հնարավոր լինի օգտագործել օբյեկտը դիմամիկ էջում այն պետք է սատարի համապատասխան հատկությունները, մեթոդները և իրադարձությունները: Սակայն պետք է հիշել, որ երբ օբյեկտը ներդրված է, էջում առկա են դրանց երկու հավաքածուներ:

օբյեկտը պարունակող կոնտեյների (օրինակ՝ <div>, <p>, <td> և այլն)

ներդրված օբյեկտի՝ դրանք տարբերվում են կոնտեյների և էջի այլ օբյեկտների հատկություններից և մեթոդներից:

ActiveX էլեմենտները էջում ներդնելու համար օգտագործվում է մեզ արդեն ծանոթ <object> տեգը: Ներդրվող էլեմենտի դիրքը և չափսերը սահմանելու (եթե, իհարկե, դա անհրաժեշտ է) համար օգտագործվում են սովորական HTML ատրիբուտները կամ հատկությունները: Բացի այդ, միայն տվյալ օբյեկտին բնորոշող հատկությունները սահմանվում են մեկ կամ մի շարք <param> ներդրված տեգերի միջոցով: Յուրաքանչյուր այդպիսի տեգ ունի երկու ատրիբուտ՝ name=“հատկության_անուն” և value=“հատկության_արժեք”:

Երբ օբյեկտը ներդրվում է, այն գրանցվում է օպերացիոն համակարգի ռեսուրսում, որպեսզի համակարգը անհրաժեշտության դեպքում կարողանա գտնել և ներդնել այն առանց կրկնակի բեռնման: Իսկ որպեսզի վստահ լինենք, որ օբյեկտը գրանցված է միայն մեկ անգամ ստեղծվում է գրանցման ունիկալ համարը պարունակող classid տողը: Բերենք ActiveX ղեկավարման էլեմենտի օրինակ (հիշենք, որ codebase ատրիբուտը սահմանում է օբյեկտի կոդի հասցեն այնպիսի դեպքերի համար, երբ այն անհասանելի է)

```
<object id="timemove" width="40" height="40" classid="
"CLSID:59ccb4a0-727D-11CF-AC36-00AA00A47DD2"
```

```
codebase="http://activex.microsoft.com/controls/iexplorer/timer.ocx"
>
<param name="interval" value="4000" />
<param name="enabled" value="true" />
</object>
```

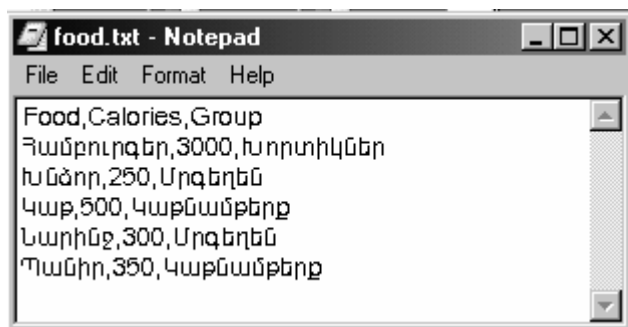
Բրաուզերի կողմից ինտրանետ և ինտերնետ միջավայրերում լուծվող կարևորագույն խնդիրներն են՝ տվյալների բազաների դիտարկումը և աշխատանքը դրանց հետ, ընդ որում ինչպես սովորական Oracle, SQL Server կամ Access տեսակների, այնպես էլ հատուկ մշակված համակարգերում: Այդպիսի խնդիրների լուծման համար գոյություն ունեն ինչպես սերվերային ծրագրավորման դասական միջոցներ (CGI ծրագրեր և սերվերային սցենարներ, գրված C, Perl և այլ լեզուներով), այնպես էլ հատուկ տեխնոլոգիաներ (ASP - Ակտիվ Սերվերային Էջեր):

Իհարկե լավ է, երբ տվյալների բազայի հետ ամբողջ աշխատանքը կատարվում է սերվերի վրա, իսկ օգտվողին մատուցվում է պատրաստի HTML էջ: Սակայն որոշ դեպքերում հաճախորդը ցանկանում է անմիջականորեն իրագործել այդ աշխատանքը՝ օրինակ, փոփոխել որոշ ինֆորմացիա կամ դիտարկել անհրաժեշտ տվյալները: Եթե այդ ամենը կատարվի սերվերի միջնորդությամբ դա կպահանջի ինչպես տվյալների բազմակի փոխանակում սերվերի և հաճախորդի միջև, այնպես էլ մշակման կրկնվող գործառույթներ սերվերի վրա: Լոկալ ցանցերում այդպիսի կրկնությունները և փոխանակությունները առանձնակի պրոբլեմներ չեն առաջացնում, սակայն ինտերնետում աշխատելիս դա պահանջում է ժամանակի մեծ ծախսեր:

Պրոբլեմի լուծումներից մեկը՝ հաճախորդի համարգչում տվյալների “քեշի” ստեղծումն է (բառացիորեն cach-ը թարգմանվում է որպես պաշար): Դա նշանակում է, որ պահանջվող տվյալները բեռնվում են հաճախորդի համակարգիչ (սովորաբար որպես ժամանակավոր՝ temporary ֆայլեր) և օգտվողը հնարավորություն է ստանում դիտարկել տվյալները ցանկացած պահին: Քեշի ստեղծումն ունի ևս մեկ առավելություն՝ օգտվողը կարող է փոփոխել աղյուսակների բազմաթիվ վանդակների պարունակությունը և միանվագ կերպով հաղորդել սերվերին բոլոր փոփոխված տվյալները:

ActiveX օբյեկտների աշխատանքի սկզբնունքները հասկանալու նպատակով դիտարկենք սկզբնական տվյալները էջի ղեկավարման էլեմենտների հետ կապակցման համար նշանակված միավորող էլեմենտի օրինակ: Որպես սկզբնաղբյուր մեր օրինակում օգտագործենք պարզագույն տեքստային ֆայլ, որը պարունակում է

տվյալներ սննդամթերքների վերաբերյալ (food.txt ֆայլը): Այն բերված է պատկեր 3.9.1-ում:



Պատկեր 3.9.1. “Սնունդ” տվյալների “բազան”

Տվյալները ֆայլում տեղաբաշխված են աղյուսակային դասավորությամբ՝ տողերով (առաջին տողում գրանցված են “սյունակների” անունները): Որպեսզի հնարավոր լինի տարբերել սյունակները, յուրաքանչյուր տողում տվյալները բաժանված են ստորակետերով (այսպես կոչված “comma” բաժանիչով – Field Delimiter):

Առաջին քայլում անհրաժեշտ է կապակցել “բազան” էջի հետ, որպեսզի տվյալները հասանելի լինեն էջի ղեկավարման էլեմենտներին: Այդ նպատակին են ծառայում հատուկ կապակցող էլեմենտները, որոնք ապահովում են տվյալների և արտապատկերվող էջի փոխհամագործակցությունը: Դրանք լինում են երկու տեսակների՝

- STD – պարզագույն աղյուսակային տվյալների (Simple Tabular Data) վերահսկման (դրանք անվանում են նաև TDC - Tabular Data Control) էլեմենտներ:
- RDS – տվյալների հեռացված սպասարկման (Remote Data Service) էլեմենտներ, որոնց միջոցով սպասարկվում են ODBC համակարգով սարքավորված տվյալների զանգվածները:

Քանի որ մեր տվյալների բազան բաղկացած է ընդամենը պարզագույն աղյուսակային տվյալներից, ապա մենք կօգտագործենք STD ղեկավարման էլեմենտը, որը ներդրվում է էջում <object> տեգի միջոցով: Շնորհից օբյեկտի իդենտիֆիկատորին “tdfood” արժեքը, DataURL հատկությանը տեքստային ֆայլի հասցեն (քանի որ երկու ֆայլն էլ գտնվում են միևնույն թղթապանակում, ապա որպես հասցե պարզապես գրանցում ենք տեքստային ֆայլի անունը), իսկ FieldDelim (դաշտերի բաժանիչ) հատկությանը “,” արժեքը: Բացի

այդ UseHeader հատկությանը շնորհիվում է True արժեքը, այսինքն օգտագործվում են առաջին տողում գրանցված սյունակների անվանումները: Միավորող TDC էլեմենտը կընդունի հետևյալ տեսքը՝

```
<object classid="clsid:333C7bc4-460f-11d0-bc04-0080c7055a83"
id="tdfood" height="1" width="1">
  <param name="FieldDelim" value="," />
  <param name="DataURL" value="food.txt" />
  <param name="UseHeader" value="True" />
</object>:
```

Տվյալների կապակցումը էջին կարելի է իրագործել երկու եղանակներով՝

տվյալների միարժեք կապակցման,

տվյալների աղյուսակային կապակցման:

Տվյալների միարժեք կապակցման դեպքում ղեկավարման էլեմենտներում ցուցադրվում են առանձին գրանցումները: Աղյուսակի յուրաքանչյուր դաշտը կապակցվում է ղեկավարման էլեմենտի հետ datasrc և datafld ատրիբուտների միջոցով: Որպես տվյալների սկզբնաղբյուր ընդունվում է կապակցող էլեմենտը՝ datasrc="#tdfood", իսկ datafld ատրիբուտին շնորհիվում է աղյուսակի անհրաժեշտ սյունակի (դաշտի) անվանումը, օրինակ՝

```
<input type="text" datasrc="#tdfood" datafld="Food" />:
```

Էջի սկզբնական բեռնման ժամանակ այդպես սահմանված մուտքագրման պատուհանում կարտապատկերվի food.txt ֆայլի Food սյունակի առաջին տողի գրանցումը՝ “Համբուրգեր”: Կապակցելով բոլոր դաշտերը առանձին էլեմենտներին կարող ենք դիտարկել աղյուսակի առանձին տողի բոլոր գրանցումները:

Գրանցումների զանգվածով տեղաշարժվելու համար օգտագործվում են ղեկավարման էլեմենտի **recordset** (recordset՝ գրանցումների բազմություն) ենթաօբյեկտի հատկությունները և մեթոդները, որոնք գործնականում նման են տվյալների բազաների կազմակերպման բոլոր տեխնոլոգիաներում առաջարկվող մեթոդներին: Դրանք են՝

eof (end of file) - ազդարարում է ֆայլի վերջը և կարող է ընդունել երկու արժեքներ՝ true կամ false,

- ◆ bof (end of file) - ազդարարում է ֆայլի սկիզբը
- ◆ moveFirst - անցում առաջին գրանցմանը (տողին),
- ◆ moveLast - անցում վերջին գրանցմանը,
- ◆ moveNext - անցում հաջորդ գրանցմանը,
- ◆ movePrevios - անցում նախորդ գրանցմանը:

Տվյալների զանգվածով տեղաշարժը ապահովելու համար սցենարը կունենա հետևյալ տեսքը՝

```
<script>
<!--
function fMovePrevious()
{ // նախապես ստուգվում է ֆայլի սկզբի ազդարարի վիճակը
if(!tdfood.recordset.bof) tdfood.recordset.movePrevious;
}
function fMoveNext()
{ // նախապես ստուգվում է ֆայլի վերջի ազդարարի վիճակը
if(!tdfood.recordset.eof) tdfood.recordset.moveNext;
}
function fMoveFirst()
{ tdfood.recordset.moveFirst;
}
function fMoveLast()
{ tdfood.recordset.moveLast;
}
//-->
</script>
```

Ստեղծելով նաև սցենարի ֆունկցիաներին դիմելու համար համապատասխան մշակիչներով կոճակներ վերջնական տեսքով կստանանք հետևյալ HTML փաստաթուղթը՝

Ծր. 3.9.1. Տվյալների միարժեք կապակցման օրինակ

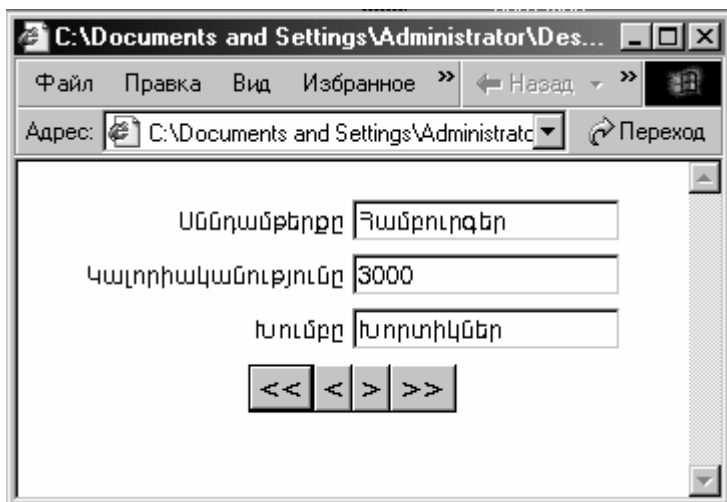
```
<html>
<head>
<title></title>
<style>
input {font-family:Arial Armenian;font-size:10pt}
</style>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script>
<!--
function fMovePrevious()
{
if(!tdfood.recordset.bof)
tdfood.recordset.movePrevious;
}
function fMoveNext()
{
```

```

if(!tdfood.recordset.eof)
    tdfood.recordset.moveNext;
}
function fMoveFirst()
{
    tdfood.recordset.moveFirst;
}
function fMoveLast()
{
    tdfood.recordset.moveLast;
}
//-->
</script>
</head>
<body>
<table width="100%">
<tr>
<td align="right">Սննդամթերքը</td><td>
<input datasrc="#tdfood" datafld="Food" /></td></tr>
<tr><td align="right">Կալորիականությունը</td><td>
<input datasrc="#tdfood" datafld="Calories" /></td></tr>
<tr><td align="right">Խումբը</td><td>
<input datasrc="#tdfood" datafld="Group" /></td></tr>
</table>
<table width="100%"><tr>
<td align="center">
<button onclick="fMoveFirst()"><b>&lt;&lt;</b></button>
<button onclick="fMovePrevious()"><b>&lt;</b></button>
<button onclick="fMoveNext()"><b>&gt;</b></button>
<button onclick="fMoveLast()"><b>&gt;&gt;</b></button>
</td>
</tr>
</table>
<object classid="clsid:333C7bc4-460f-11d0-bc04-0080c7055a83"
    id="tdfood" height="1" width="1">
    <param name="FieldDelim" value=",">
    <param name="DataURL" value="food.txt">
    <param name="UseHeader" value="True">
</object>
</body>
</html>

```

Սկզբնական բեռնման ժամանակ էջը կունենա պատկեր 3.9.2-ում ցուցադրված տեսքը: Օգտագործելով կոճակները կարելի է դիտարկել food.txt ֆայլի բոլոր գրանցումները:



Պատկեր 3.9.2. Տվյալների միաբժեք կապակցման օրինակ

Տվյալների աղյուսակային կապակցման դեպքում բրաուզերի պատուհանում աղյուսակի տեսքով ցուցադրվում են միաժամանակ մի քանի գրանցումներ: Այդպիսի արտապատկերումը ապահովելու համար անհրաժեշտ է մի քանի անգամ կրկնել էջի գրանցումները պարունակող մասը:

Առավել հաճախ այդ նպատակով ստեղծվում է փաստաթղթի հատված, որտեղ տեղադրվում է տվյալների արտապատկերման համար նախատեսված աղյուսակը: Այս դեպքում datasrc ատրիբուտը տեղադրվում է <table> տեգում, որպեսզի նշենք, որ միավորող էլեմենտի հետ կապակցված է մի ամբողջ աղյուսակ: Աղյուսակի համար ստեղծվում է վերնագրային մաս (thead), որտեղ գրանցվում են համապատասխան սյունակների անունները, և բովանդակության ավտոմատ կերպով կրկնվող մաս (tbody), որտեղ արտապատկերվում են սկզբնաղբյուր աղյուսակի բոլոր տողերը՝

```
<table datasrc="#tdfood">
<thead>
<th>Սննդամթերքը</th>
<th>Կալորիականությունը</th>
```

```

<th>Խուճը</th>
</thead>
<tbody>
<tr>
<td><span datafld="Food"></span></td>
<td align="center"><span datafld="Calories"></span></td>
<td><span datafld="Group"></span></td>
</tr>
</tbody>
</table>

```

Քանի որ <td> տեղը չի սատարում datafld ատրիբուտը, ապա դա իրականացվում է տեգերի միջոցով, որոնք ներդրվում են համապատասխան սյունակներում: Տվյալների աղյուսակային կապակցման եղանակը ցուցադրող էջի կոդը և արտապատկերումը բերված են, համապատասխանաբար, Ծր. 3.9.2-ում և պատկեր 3.9.3-ում:

Ծր. 3.9.2. Տվյալների աղյուսակային կապակցման օրինակ

```

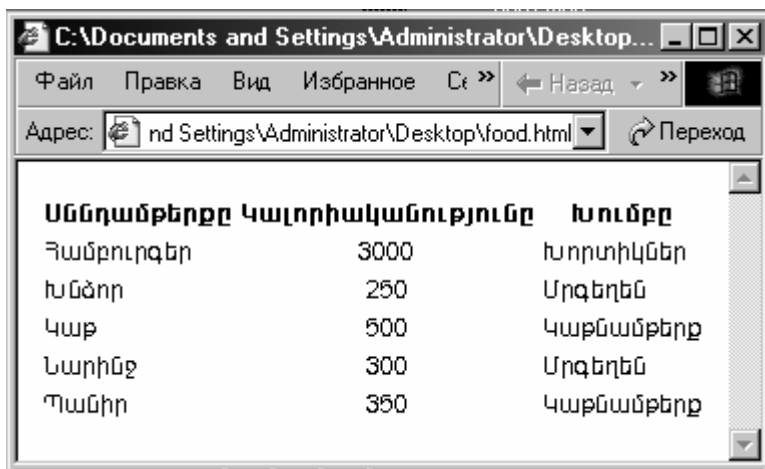
<html>
<head>
<title></title>
<style>
td {font-family:Arial Armenian;font-size:10pt}
</style>
</script>
</head>
<body>
<table id="tab1" name="tab1" datasrc="#tdfood">
<thead><th>Սննդամթերք</th><th>Կալորիականություն</th><th>
Խուճը</th></thead>
<tbody>
<tr><td><span dataFld="Food"></span></td>
<td align="center"><span dataFld="Calories"></span></td>
<td><span dataFld="Group"></span></td></tr>
</tbody>
</table>
<object classid="clsid:333C7bc4-460f-11d0-bc04-0080c7055a83"
id="tdfood" height="1" width="1">
<param name="FieldDelim" value=",">
<param name="DataURL" value="food.txt">
<param name="UseHeader" value="True">

```

</object>

</body>

</html>



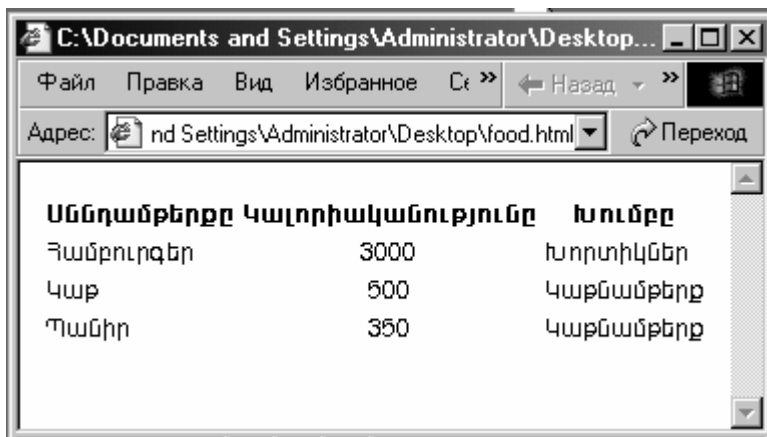
Պատկեր 3.9.3. Տվյալների աղյուսակային կապակցման օրինակ

Որոշ դեպքերում առաջանում է տվյալների տեսակավորման կամ վերադասավորման անհրաժեշտություն: Օրինակ, պահանջվում է արտապատկերել մթերքների այն տեսակները, որոնց էներգետիկ արժեքը գերազանցում է 300 կալորիա: Միաժամանակ անհրաժեշտ է տեսակավորել մթերքները ըստ խմբերի: Դա կատարվում է ֆիլտրերի և տեսակավորման հատկությունների օգտագործման միջոցով: <param> տեղերում շնորհվում են համապատասխան արժեքներ ֆիլտրվող սյունակին (FilterColumn) ֆիլտրի արժեքին (FilterValue), ֆիլտրման պայմանին (FilterCriterion) և նշվում է այն սյունակի անունը, որի արժեքները պետք է տեսակավորվեն (SortColumn): Օբյեկտը նկարագրող կոդը կընդունի հետևյալ տեսքը՝

```
<object classid="clsid:333C7bc4-460f-11d0-bc04-0080c7055a83"
  id="tdfood" height="1" width="1">
  <param name="FieldDelim" value="," />
  <param name="DataURL" value="food.txt" />
  <param name="UseHeader" value="True" />
  <param name="FilterColumn" value="Calories" />
  <param name="FilterCriterion" value=">" />
```

```
<param name="FilterValue" value="300" />
<param name="SortColumn" value="Group" />
</object>
```

Արդյունքում կստացվի պատկեր 3.9.4-ում բերված պատուհանը:



Պատկեր 3.9.4. Տվյալների ֆիլտրման և տեսակավորման օրինակ

Ինչպես կարելի է տեսնել արդյունքում ընտրվել են միայն այն մթերքները, որոնց միավորի կալորիականությունը գերազանցում է 300 կալորիա: Բացի այդ մթերքները տեսակավորվել են ըստ Group (խումբը) սյունակի արժեքների:

Բոլոր այդ գործողությունները կարելի է կատարել նաև սցենարից, օգտագործելով TDC օբյեկտի համապատասխան հատկությունները և մեթոդները: Օրինակ, որպեսզի հասնենք նույն արդյունքին, այսինքն ընտրենք մթերքները, որոնց միավորի կալորիականությունը գերազանցում է 300 կալորիա անհրաժեշտ է “tdfood” օբյեկտի Filter հատկությամբ շնորհել համապատասխան արժեքը և ապա կիրառել Reset() մեթոդը, որը կհաստատի այդ փոփոխությունը՝

```
tdfood.Filter="Calories>300";
```

```
tdfood.Reset();
```

Նույն եղանակը կարելի է օգտագործել նաև տվյալները ըստ որևէ սյունակի (դաշտի) տեսակավորելու համար: Օրինակ, եթե անհրաժեշտ է տեսակավորել գրանցումները ըստ Group դաշտի, ապա ծրագրային կոդի համապատասխան հատվածը կլինի հետևյալը՝

```
tdfood.Sort="+Group";
```

```
tdfood.Reset();
```

“+” նշանի դեպքում տեսակավորումը կատարվում է ըստ աճման կարգի: Ըստ նվազման տեսակավորելու համար Sort հատկությանը շնորհվում է “-” նշանը, որին նույնպես հետևում է տեսակավորվող սյունակի անունը, օրինակ, եթե գրենք՝

```
tdfood.Sort="-Calories";
```

```
tdfood.Reset();
```

ապա տվյալները կտեսակավորվեն ըստ Calories սյունակի արժեքների նվազման կարգի:

TDC էլեմենտը ունի նաև մեթոդներ, որոնց միջոցով կարելի է ավելացնել նոր գրանցումներ, ջնջել գրանցումները կամ փոփոխել առանձին վանդակների պարունակությունը: Սակայն այդպիսի դեպքերում սկզբնական տվյալները պետք է պահպանվեն ոչ թե “.txt” ընդլայնումով, այլ որևէ “.csv” (comma-separated values) ֆորմատում (օրինակ Excel-ի, Access-ի և այլ OLE DB տեխնոլոգիաների միջոցով ստեղծված ֆայլերը): Այդպիսի ֆորմատի կարելի վերափոխել նաև մեր կազմած food.txt ֆայլը: Եթե բացենք այ և ապա հիշեցնենք որպես (Save As) food.csv, ապա ավտոմատ կերպով կստեղծվի Excel ֆորմատի էլեկտրոնային աղյուսակ (տես՝ պատկեր 3.9.5.):

Microsoft Excel - food.csv			
File Edit View Insert Format Tools Data Window Help			
A1 = Food			
A	B	C	D
1 Food	Calories	Group	
2 Համբուրգ	3000	Խորտիկներ	
3 Խնձոր	250	Մրգեղեն	
4 Կաթ	500	Կաթնամթերք	
5 Նարինջ	300	Մրգեղեն	
6 Պանիր	350	Կաթնամթերք	
7			

Պատկեր 3.9.5. food.csv ֆայլը

Աժմ արդեն հնարավորություն է ընձեռնվում օգտագործել recordset օբյեկտի համապատասխան հատկությունները և մեթոդները գրանցումներ ջնջելու, ավելացնելու կամ փոփոխելու համար: Օբյեկտի որոշ մեթոդների կիրառությունը (moveFirst, moveNext, movePrevious, moveLast, eof, bof) մենք արդեն ուսումնասիրել ենք: Ստորև թվարկված են նաև մի քանի լրացուցիչ՝ առավել հաճախ կիրառվող հատկություններ և մեթոդներ:

- **RecordCount** - գրանցումների (տողերի) քանակը տվյալների աղբյուրում (օրինակ՝ tdfound.recordset.RecordCount հատկության արժեքը հավասար է 5-ի);
- **AbsolutePosition** – ընթացիկ գրանցման համարն է: Հնարավոր է նաև շնորհիվ այդ հատկությանը անհրաժեշտ արժեքը: Օրինակ, եթե ցանկանում ենք կատարել որևէ գործողություն i-րդ տողում գրանցված տվյալների հետ, ապա նախապես պետք է շնորհիվ տվյալ հատկությանը i արժեքը՝ tdfound.recordset.AbsolutePosition=i:
- **fields** – սյունակների (դաշտերի) հավաքածու է, որն ունի սեփական հատկություններ՝ fields.count – սյունակների քանակն է, fields(i).name i-րդ սյունակի անունն է, իսկ fields(i).value այդ սյունակի ընթացիկ (կամ ընտրած) տողում գրանցված տվյալի արժեքն է:
- **AddNew()** - տվյալ մեթոդը ավելացնում է աղյուսակի վերջում դատարկ տող (օրինակ՝ tdfound.recordset.AddNew()), որը հետագայում կարելի է լրացնել անհրաժեշտ տվյալներով, շնորհիվով ավելացված տողի բոլոր դաշտերին համապատասխան արժեքները: Օրինակ՝ tdfound.recordset. fields[1].value=400:
- **Delete()** – ջնջում է վերջին կամ AbsolutePosition հատկության միջոցով ընտրված տողը ամբողջովին:

Ծրագիր 3.9.3-ում ցուցադրվում են վերը թվարկված մեթոդների և հատկությունների կիրառության եղանակները, ընդ որում իրագործվում է տվյալների աղբյուրի և անմիջական, և աղյուսակային կապակցումը էջին: Հարկ է ուշադրություն դարձնել որոշ մշակիչների ստեղծման այլընտրանքային եղանակին: Էջի կամայական օբյեկտի համար կարելի է ստեղծել մշակիչ նաև առանձին սկրիպտի տեսքով: Իրադարձությունը մշակող հրամանները տեղադրվում են <script></script> տեգում, որը օժտվում է համապատասխան **for** և **event** ատրիբուտներով: for ատրիբուտին շնորհվում է այն օբյեկտի իդենտիֆիկատորը, որի համար մշակվելու է իրադարձությունը, իսկ event ատրիբուտին իրադարձության մշակիչի անունը: Օրինակ, աղյուսակի տողը ջնջելու համար (տես՝ պատկեր 3.9.6.) նախատեսված “Ջնջել” կոճակի (որի իդենտի-

ֆիկստորն է btnDelete) onclick մշակիչի համար նախատեսված
 օրագրային կոդը ունի հետևյալ տեսքը
 <script for="btnDelete" event="onclick">
 tdfood.recordset.AbsolutePosition = this.recordNumber;
 tdfood.recordset.Delete();
 </script>

Նշենք, որ **this.recordNumber** ցուցանակի միջոցով տվյալ
 դեպքում որոշվում է էջում արտապատկերված աղյուսակի տողի
 համարը (որը համապատասխանում է սկզբնաղբյուր աղյուսակի
 համապատասխան գրանցման համարին), քանի որ յուրաքանչյուր
 “Ջնջել” կոճակը տեղադրված է աղյուսակի որոշակի տողում, իսկ
 ինքը կոճակը չունի recordNumber հատկություն:

Այդ եղանակը թույլ է տալիս ձևակերպել մշակող հրամանները
 ոչ միայն ֆունկցիայի, այլ սովորական հաջորդական հրամանների
 տեսքով: Միաժամանակ վերանում է event.CancelBubble հատկու-
 թյան կիրառության անհրաժեշտությունը, քանի որ <script> տեգի
 ատրիբուտների արժեքները միարժեքորեն որոշում են այն օբյեկ-
 տը, որի համար կատարվելու է տվյալ իրադարձության մշակումը:

Ուշադրություն դարձրեք նաև document օբյեկտի նոր՝

document.createElement(‘էլեմենտի_դեկլարիատոր’)

մեթոդին: Այն հնարավորություն է ընձեռում էջում դինամիկորեն
 ստեղծել կամայական էլեմենտ, որի դեկլարիատորը գրանցվում է
 որպես մեթոդի արգումենտ: Իհարկե, անհրաժեշտ է նաև նախապես
 ստեղծել կոնտեյներ, որում պետք է տեղադրվի նոր էլեմենտը: Մեր
 օրինակում դա <select id="cboFields"></select> դատարկ մենյուն է,
 որի ներսում ավելացվում են <option> էլեմենտները: Յուրա-
 քանչյուր <option> էլեմենտին շնորհվում է աղյուսակի հաջորդ
 սյունակի անվանումը (cField = vRecordSet.fields[i].name;) որպես
 արժեք (oOption.value = cField) և տեքստ (oOption.text = cField):
 Ապա այն ավելացվում է <select> տեգում cboFields.add(oOption):
 Ավարտից հետո որպես ընտրված հաստատվում է առաջին
 <option> տեգը՝ cboFields.selectedIndex = 0, ընդ որում դա կա-
 տարվում է տվյալների աղբյուրի յուրաքանչյուր փոփոխության
 դեպքում (ondatachanged իրադարձությունը կապակցող tdfood
 էլեմենտի համար):

Ծր. 3.9.3. TDC և recordset օբյեկտների հատկությունների և մեթոդ- ների կիրառության օրինակ (food.html ֆայլը)

```
<html><head><title></title>  
<style>  
input {font-family:Arial Armenian;font-size:10pt}
```

```

button {font-family:Arial Armenian;font-size:8pt}
</style>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script for="tdfood" event="ondatasetchanged">
var vRecordSet = tdfood.recordset;
for (i = 0; i < vRecordSet.fields.count; i++)
{ cField = vRecordSet.fields(i).name;
  oOption = document.createElement('option')
  oOption.value = cField;
  oOption.text = cField;
  cboFields.add(oOption);
}
cboFields.selectedIndex = 0;
</script>
<script for="btnDelete" event="onclick">
tdfood.recordset.AbsolutePosition = this.recordNumber;
tdfood.recordset.Delete();
</script>
<script>
<!--
function UpdateField(cField, cValue)
{ tdfood.recordset.fields(cField).value = cValue; }
function fMovePrevious()
{ if(!tdfood.recordset.bof) tdfood.recordset.movePrevious; }
function fMoveNext()
{ if(!tdfood.recordset.eof) tdfood.recordset.moveNext;}
function fMoveFirst()
{ tdfood.recordset.moveFirst;}
function fMoveLast()
{ tdfood.recordset.moveLast;}
//-->
</script>
</head>
<body>
<object classid="clsid:333C7bc4-460f-11d0-bc04-0080c7055a83"
  name="tdfood" id="tdfood" height="1" width="1">
  <param name="TextQualifier" value="">
  <param name="DataURL" value="food.csv">
  <param name="UseHeader" value="True">
</object>
<table>

```

```

<tr><td align="right">Սննդամթերքը</td>
<td><input datasrc="#tdfood" datafld="Food" /></td></tr>
<tr><td align="right">Կալորիականությունը</td>
<td><input datasrc="#tdfood" datafld="Calories" /></td></tr>
<tr><td align="right">Խումբը</td>
<td><input datasrc="#tdfood" datafld="Group" /></td></tr>
<tr><td align="center" colspan="2">
<button onclick="fMoveFirst()"><b>&lt;&lt;</b></button>
<button onclick="fMovePrevious()"><b>&lt;</b></button>
<button onclick="fMoveNext()"><b>&gt;</b></button>
<button onclick="fMoveLast()"><b>&gt;&gt;</b></button></td></tr>
</table>
<button id="cmdAdd" title="Add a new record"
onclick="tdfood.recordset.AddNew()">Ավելացնել գրանցում
</button>
<button id="cmdDelete" title="Delete the current record"
onclick="if (tdfood.recordset.RecordCount > 0)
tdfood.recordset.Delete()">Ջնջել ընթացիկ գրանցումը</button>
<p>Ընտրեք փոփոխվող դաշտը<select id="cboFields">
</select>
<input type="text" id="txtValue" title="Enter a value for the selected
field" />
<button
onclick="UpdateFieldWithData(cboFields.options(cboFields.selectedI
ndex).value, txtValue.value)">Փոփոխել</button>
</p>
<p>
<table id="tab1" name="tab1" datasrc="#tdfood" bgcolor="#cccccc"
cellSpacing="2" border="1">
<thead><th>&nbsp;</th><th>Սննդամթերքը</th><th>Կալորի.</th><th>Խ
h>Խումբը</th></thead>
<tbody>
<tr>
<td><button title="Delete this record" id="btnDelete">Ջնջել
</button></td>
<td><span dataFld="Food"></span></td>
<td align="center"><span dataFld="Calories"></span></td>
<td><span dataFld="Group"></span></td></tr>
</tbody>
</table></p></body></html>

```

Файл Правка Вид Избранное Сервис С >>

Սննդամթերքը

Կալորիականությունը

Խումբը

<< < > >>

Ավելացնել գրանցում

Ջնջել ընթացիկ գրանցումը

Ընտրեք փոփոխվող դաշտը

Փոփոխել

	Սննդամթերքը	Կալոր.	Խումբը
<input type="button" value="Ջնջել"/>	Համբուրգեր	3000	Խորտիկներ
<input type="button" value="Ջնջել"/>	Խնձոր	250	Մրգեղեն
<input type="button" value="Ջնջել"/>	Կաթ	500	Կաթնամթերք
<input type="button" value="Ջնջել"/>	Նարինջ	300	Մրգեղեն
<input type="button" value="Ջնջել"/>	Պանիր	350	Կաթնամթերք

Պատկեր 3.9.6. TDC և recordset օբյեկտների հատկությունների և մեթոդների կիրառության օրինակ

Ինչպես նշվել է RDS էլեմենտը կիրառվում է ODBS համակարգերի միջոցով ստեղծված տվյալների զանգվածների (բազաների) հեռացված սպասարկման համար: Էլեմենտի ներդրման պարզագույն եղանակը հետևյալն է՝

```
<object classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
id="RDS" height="1" width="1">
```

```
<param name="SQL" value="SQL հարցումը" />
```

```
<param name="SERVER" value="արձանագրություն://սերվերի
անուն" />
<param name="CONNECT" value="տվյալների աղբյուրի անունը"
/>
</object>:
```

Կարելի է նկատել, որ RDS և TDC էլեմենտների ատրիբուտները տարբեր են: Դա բացատրվում է նրանով, որ հեռացված տվյալների միանալու համար անհրաժեշտ է նշել դրանց հասնելու “ճանապարհը”: Նախ և առաջ պետք է նշել հեռացված սերվերի անունը (SERVER ատրիբուտի արժեքը), որում գտնվում են տվյալները կամ որի միջոցով դրանց կարելի միանալ: Բացի այդ անհրաժեշտ է որևէ եղանակով նշել սկզբնաղբյուրի տեղաբաշխումը (CONNECT ատրիբուտի արժեքը): Օրինակ, կարելի է նշել տվյալների բազայի սիստեմային անունը, որով այն գրանցված է համակարգչում՝ dsn=բազայի_անուն: Կամ նշել տվյալների բազայի ֆայլի բացարձակ հասցեն՝ c:\.myBasa.mdb:

Տվյալների հետ կոնկրետ գործողությունները իրականացվում են SQL ատրիբուտին համապատասխան SQL հարցման արժեքը շնորհիվ միջոցով: Օրինակ, եթե գրանցենք՝

```
<object classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
id="RDS" height="1" width="1">
<param name="SQL" value="select * from authors" />
<param name="SERVER" value="http://MyServer" />
<param name="CONNECT" value="dsn=pubs" />
</object>,
```

ապա կկատարվի միացում MyServer դոմեյնային անունով սերվերի pubs տվյալների բազայի հետ և կընտրվեն տվյալները բազայի authors աղյուսակից:

RDS էլեմենտի կապակցման եղանակները նույն են, ինչ և TDC էլեմենտինը՝ անմիջական և աղյուսակային: Դրան նույնպես հատուկ է recordset օբյեկտը իր հատկություններով և մեթոդներով, որոնք թույլ են տալիս տեղաշարժվել գրանցումների զանգվածով, պարզել գրանցումների քանակը և այլն: Սովորաբար կիրառվում է էլեմենտի օգտագործման սցենարային եղանակը:

Սցենարի միջոցով կարգավորվում են կապերը էջի հետ, փոփոխվում են օբյեկտի հատկությունները, օգտագործվում համապատասխան մեթոդները: Օրինակ, օբյեկտը էջում հետևյալ եղանակով ներդնելուց հետո՝

```
<object classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
id="RDS" height="1" width="1">
</object>,
```

կարելի է արդեն սցենարի միջոցով շնորհել հատկություններին անհրաժեշտ արժեքները՝

RDS.SQL="select * from authors (SQL հատկությամբ շնորհվում է համապատասխան արժեքը),

RDS.Server="http://MyServer",

RDS.Connect="dsn=pubs",

RDS.SortColumn="au_fname",

կամ կիրառել համապատասխան մեթոդները՝

RDS.recordset.moveFirst,

RDS.Query("select * from authors") (SQL հարցումը իրագործվում է Query մեթոդի միջոցով)

և այլն:

RDS սցենարների ուսումնասիրությունը բավականին ծավալուն և աշխատատար գործընթաց է և հանդիսանում է առանձին ձեռնարկի թեմա: Այդ պատճառով տվյալ ձեռնարկում դա մանրամասն չի քննարկվում:

Լաբորատոր աշխատանքների առաջադրանքներ

Լաբորատոր աշխատանք N1

1.1. Կազմել էջ, որը արտացոլի հետևյալ ինֆորմացիան

Ազգանուն	<input type="text"/>
Անուն	<input type="text"/>
Հայրանուն	<input type="text"/>
<hr/>	
<input type="radio"/> Շտապ	
<input type="radio"/> e-mail հաստատումով	
<input type="radio"/> Տեղափոխումով	

1.2. Կազմել էջ, որը արտացոլի հետևյալ ինֆորմացիան

Ո՞ր տարիքային խմբին էք դասվում	
<input type="radio"/>	Մինչև 16
<input type="radio"/>	16 - 23
<input type="radio"/>	24 - 30
<input type="radio"/>	31 - 40
<input type="radio"/>	41-ից բարձր

Լաբորատոր աշխատանք N2

Ֆունկցիաներ: Նկարագրում և օգտագործում: Իրադեպերի մշակիչներ:

2.1. Կազմել սցենար, որը որոշի միջին եկամտի հաշվարկը:

2.2. Կազմել սցենար, որը հաշվի քառակուսու մակերեսը, դաշտի ֆոկուս ստանալու իրադեպի մշակմամբ:

2.3. Ստեղծել ուղղահայաց և հորիզոնական մենյուներ:

Լաբորատոր աշխատանք N3

Math օբյեկտը և նրա մեթոդները

3.1 Կազմել սցենար, որը որոշի տրված երեք թվերից մեծագույնը և փոքրագույնը

3.2. Կազմել սցենար, որը հաշվի կրթաթոշակի չափը:

Լաբորատոր աշխատանք N4

Պայմանական և ցիկլային պրոցեսների կազմակերպումը

4.1. Կազմել սցենար, որը ստեղծի պատկերի վիզուալ հեռացման և մոտեցման էֆեկտներ:

4.2. Կազմել սցենար, որը Switch օպերատորի օգտագործմամբ օրվա համարով որոշի նրա անունը:

4.3. Կազմել սցենար, որը իրականացնի դասախոսի ծանրաբեռնվածության արտացոլումը դիագրամայի տեսքով:

Լաբորատոր աշխատանք N5

Տեքստերի կամ պատկերի տեղադրումը աղյուսակի բջիջների ներսում

5.1. Կազմել սցենար, որը իրականացնի ընթերցողների անկետաների մշակումը

5.2. Գրել սցենար, որը կատարի պատկերների ընտրումը ցուցակից:

Լաբորատոր աշխատանք N6

Ֆրեյմների հետ աշխատանքը

6.1 Ստեղծել երեք ֆրեյմներից բաղկացած ֆրեյմային կառուցվածք:

6.2 Ստեղծել կոճակով 3 ֆրեյմ, որով հնարավոր լինի փոխանակել ֆրեյմների պարունակությունները:

Լաբորատոր աշխատանք N7

Չանգվածների հետ աշխատանքը

7.1 Կազմել սցենար, որը իրականացնի Իտերացիայի մեթոդով հավասարման լուծումը:

7.2 Կազմել սցենար, որը իրականացնի քննությունների ամփոփման ձևավորումը:

7.3 Կազմել սցենար, որը իրականացնի Երկու կարգավորված զանգվածների միավորումը:

ԳԼՈՒԽ 4. ԻՆՏԵՐԱԿՏԻՎ WEB-ԷԶԵՐ: ՍԵՐՎԵՐԱՅԻՆ ԾՐԱԳՐԱՎՈՐՈՒՄ

Մեզ արդեն հայտնի է (§1.3-ից), որ սերվերային ծրագրերը (հետագայում մենք կանվանենք դրանք սկրիպտեր՝ անկախ ծրագրի տեսակից, լինի դա CGI ծրագիր, սերվերային սցենար, թե սկտիվ սերվերային էջ) մասնագիտացված ծրագրեր են, որոնք ստանալով որոշակի տվյալներ, մշակում են դրանք և (կամ) ի պատասխան՝ ստեղծում որևէ HTML էջեր: Ամեն ինչ սկսվում է սկրիպտի կանչից, որն օրինակ կարող է կատարվել ֆորմայի պնդման (submit) կոճակի սեղմումով: Հայտնաբերելով կանչը, սերվերը գտնում է համապատասխան սկրիպտը և “ասում դրան, որ ժամանակն է մի քիչ աշխատել”: Աշխատանքը սկսելիս, սկրիպտը նախ և առաջ ստուգում է տվյալների հաղորդման եղանակը: Դրանից հետո գտնում է իրեն հայտնի փոփոխականների անունները և արժեքները և, եթե դրա հետ մեկտեղ “գիտի” ինչ պետք է անել ստացված տվյալների հետ, ապա կատարում է բոլոր անհրաժեշտ գործողությունները և ստեղծում “շնորհակալական նամակ” օգտվողին՝ ի պատասխան ֆորմայի բոլոր դաշտերը լրացնելու ծանր աշխատանքի կատարմանը: Սկրիպտերը կարող են կանչվել նաև անմիջականորեն, որպես հիպերհղումի մի մաս կամ որևէ գրաֆիկական ֆայլի URL հասցե:

Ցանկացած դեպքում սերվերային ծրագիրն ունի իմաստ, երբ առկա են օգտվողների կորեկտ տվյալները և դրանց մշակման համապատասխան մեթոդները:

§4.1. Տվյալների ստացումը ֆորմաներից

Ցանցով հաղորդվող տվյալները կոդավորելու համար օգտագործվում են երեք առավել տարածված եղանակները՝ application/x-www-form-urlencoded, multipart/form-data և (շատ հազվագյուտ) text/plain: Առավել հաճախ օգտագործվում է առաջին եղանակը (այն սովորաբար ընդունվում է ըստ լռելյայն): Երկրորդ եղանակը՝ multipart/form-data օգտագործվում է այն դեպքերում երբ անհրաժեշտ է հաղորդել սերվերին որոշակի ֆայլեր: Երրորդը՝ ներկայացնում է տվյալները սովորական տեքստի տեսքով և օգտագործվում է, օրինակ, այն դեպքերում, երբ անհրաժեշտ է ֆորմայի տվյալները հաղորդել էլեկտրոնային փոստով:

Սերվերային ծրագրերի և սերվերի (հետևաբար նաև հաճախորդների) միջև տվյալները կարող են փոխանակվել չորս հնարավոր

եղանակներով (դրանք կարող են նաև համատեղվել)՝ միջավայրի (environment) փոփոխականների միջոցով,

- հրամանային տողի,
- ելքի ստանդարտ սարքի,
- մուտքի ստանդարտ սարքի:

Երբ սկրիպտը կանչվում է ֆորմայից, ապա բրաուզերը հաղորդում է սերվերին երկար տող, որի սկզբում գրանցված է սկրիպտի “ճանապարհը” և անունը: Դրանց հետևում են մի շարք այլ տվյալներ, որոնք կոչվում են “երթուղու ինֆորմացիա” և հաղորդվում են սկրիպտին հատուկ **PATH_INFO** միջավայրի փոփոխականի միջոցով: Փոփոխականները սերվերին կարող են հաղորդվել երկու եղանակներով՝ get և post: Սկրիպտին անհրաժեշտ մեթոդի անվանումը պահպանվում է սերվերում հատուկ միջավայրի փոփոխականում, որը կոչվում է **REQUEST_METHOD** (կամայական միջավայրի փոփոխական պետք է հասանելի լինի կամայական սկրիպտին): Եթե ֆորմայի տվյալները հաղորդված են get մեթոդով, ապա ճանապարհի ինֆորմացիային հետևում է հարցական նշանը “?” և, ապա, ֆորմայի տվյալները:

Ստանալով տվյալ մեթոդով հաղորդված տվյալները, սերվերը պահպանում է դրանք մի այլ՝ **QUERY_STRING** (հարցման տող) միջավայրի փոփոխականում:

- “Տող” թերմինը ծրագրավորման մեջ նշանակում է սիմվոլների որոշակի հավաքածու, հասանելիությունը որին իրականացվում է այն պարունակող փոփոխականի անունով: Տվյալ դեպքում այդ փոփոխականը կոչվում է **QUERY_STRING** և այն նշակելու համար սերվերային ծրագրում օգտագործվում են տողային տեսակի փոփոխականներին հատուկ մեթոդները:

Բերենք օրինակ: Ենթադրենք, մեր էջը դիմում է `product.asp` սկրիպտին և փոխանցում է նրան ապրանքի կոդը (`productID`) պարունակող պարամետրը: Հարցվող փաստաթղթի URL հասցեն կընդունի մոտավորապես հետևյալ տեսքը՝

`http://www.myserver.com/products.asp/լրացուցիչ ինֆորմացիա?productID = 32`

Այն ամենն, ինչ գտնվում է հարցական նշանից առաջ ձևավորում է փաստաթղթի URL հասցեն: Հարցական նշանը առանձնացնում է այն հաղորդվող պարամետրից: Եթե պարամետրերի քանակը մեկից ավել է, ապա հարցման տողը ձևավորվում է “ամուն=արժեք” զույգերից, որոնք կցվում են միմյանց & (ամպերսանդ) սիմվոլով: Օրինակ՝

`http://www.myserver.com?l_name =Տեր+խաչ&f_name =Կամո`

Սովորաբար հարցման տողը սահմանափակված է 1000 կիլո-բայտով և տվյալների մեծ ծավալներ հաղորդելու անհրաժեշտության դեպքում get մեթոդի կիրառությունը դառնում է անհնար:

post մեթոդը փոխանցում է տվյալները մուտքի **stdin** (“standart in”) ստանդարտ սարքին: Այդ դեպքում ինֆորմացիան տվյալների ծավալի (տողի երկարության) վերաբերյալ պահպանվում է միջավայրի մի այլ **CONTENT_LENGTH** փոփոխականում: Տվյալ մեթոդի կիրառության դեպքում հաղորդվող տվյալների ծավալը գործնականում սահմանափակված չէ:

Անկախ հաղորդման մեթոդից սերվերային ծրագրի խնդիրն է՝ կատարել վերլուծություն և ընտրել անհրաժեշտ տվյալները ամբողջ հավաքածուից: Կարելի է առանձնացնել տվյալների ընդունման երկու փուլեր՝ ապակոդավորում և վերլուծություն: Կոռուստներից խուսափելու համար բրաուզերը կոդավորում է հաղորդվող տվյալները և բոլոր ոչ տեքստային սիմվոլները փոխարինվում են իրենց ASCII տասվեցական կոդերով “%” նախածանցով (պրեֆիկսով): Օրինակ՝ ամպերսանդը կոդավորվում է %26 տեսքով, իսկ տոկոսի նշանը %25–ով: Եթե պարամետրերի արժեքը պարունակում է պրոբելներ, ապա յուրաքանչյուր պրոբել փոխարինվում է %20–ով:

Ապակոդավորման արդյունքում ստացվում է հետևյալ ֆորմատի տեքստ՝

անուն1=արժեք1&անուն2=արժեք2&....:

Վերը բերված օրինակի համար կստացվի՝

l_name =Տեր%20Խաչ&f_name =Կամո

Դժվար չէ կռահել, որ փոփոխականների անունները համընկնում են ֆորմայի դաշտերի անունների, իսկ արժեքները՝ այդ դաշտերում գրանցված արժեքների հետ: Բերենք օրինակ: Կառուցենք կայքի հաճախորդների հարցման ֆորմա, որը պարունակում է երեք՝ “l_name”, “f_name”, “city” անուններով տեքստային դաշտեր, ինչպես նաև “visit” անունով սահմկող մենյու (տես՝ Օր. 4.1.1. և Պատկեր 4.1.1.):

Օր. 4.1.1. Հաճախորդների հարցման էջի օրինակ

```
<html>
```

```
<head>
```

```
<title></title>
```

```
<style>
```

```
p {font-family:'Arial Armenian';font-size:10pt;color:black;font-weight: 800}
```

```
select {font-family:'Arial Armenian';font-size:10pt;color:black;font-
```

```

weight:800}
td {font-family:'Arial Armenian';font-size:10pt;color:black;font-
weight:800}
input {font-family:'Arial Armenian';font-size:10pt;color:black;font-
weight:800}
</style>
</head>
<body>
<p>Լրացրեք համապատասխան դաշտերը</p>
<form action="hartsum.asp" name="formQuery" id="formQuery">
<table>
<tr><td>Ազգանունը</td><td><input type="text" name="l_name"
id="l_name" maxlength="40" /></td>
</tr>
<tr><td>Անունը</td><td><input type="text" name="f_name"
id="f_name" maxlength="40" /></td>
</tr>
<tr><td>Բնակավայրը</td><td><input type="text" name="city"
id="city" maxlength="40" /></td>
</tr>
<tr><td>Հաճախ եք դուք այցելում մեր կայքը</td><td>
<select name="visit" style="">
<option value="infinite">Անըդհատ</option>
<option value="first" selected="selected" >Առաջին անգամ</option>
<option value="monthly">Ամեն ամիս</option>
<option value="weekly">Ամեն շաբաթ</option>
<option value="daily">Ամեն օր</option>
</select></td></tr></table></body></html>

```

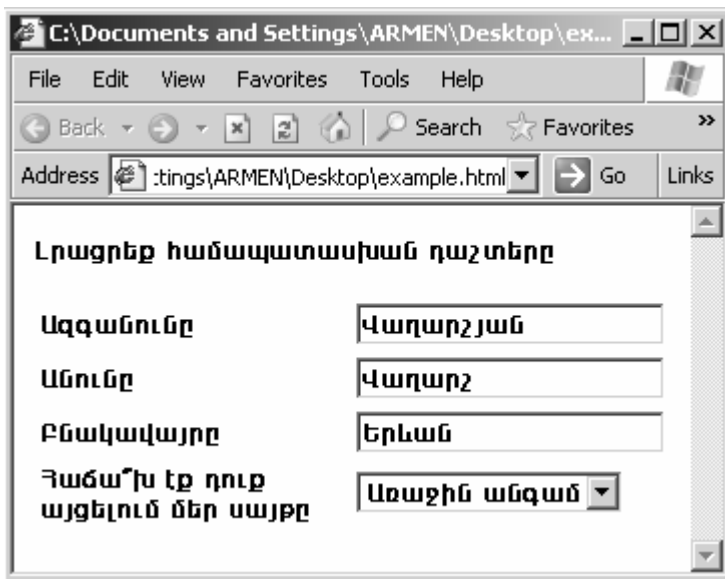
Պատկեր 4.1.1-ում ցուցադրված ձևով ֆորման լրացնելու դեպքում սերվերում տվյալների ապակողավորումից հետո կստացվի հետևյալ տողը՝

`l_name=Վաղարշյան&f_name=Վաղարշ&city=Երևան&visit=first`

Ակնհայտ է, որ փոփոխականների կոնկրետ արժեքները կհամապատասխանեն օգտվողի կողմից մուտքագրված տվյալներին:

- Գոյություն ունեն տվյալների ապակողավորման ցանցով հաղորդվող հատուկ ծրագրեր: Windows և Linux սերվերների համար դա Perl լեզվով գրված `cgi-bin.pl` ծրագիրն է:

Սակայն ամենևին էլ դժվար չէ ինքնուրույն կազմել համապատասխան ծրագիր, որը կբաժանի մուտքային ապակողավորված տողը երկու զանգվածների՝ փոփոխականների անունների և արժեքների:



Պատկեր 4.1.1. Հաճախորդների հարցման էջ

Արտաքին սարքավորումների վրա տվյալների ելքի կազմակերպումը բարդություններ չի առաջացնում: Քանի որ stdout (ելքի ստանդարտ սարքավորումների) հոսքը հաղորդվում է բրաուզերին, ապա կախված սերվերային ծրագրի տեսակից օգտագործվում են արտապատկերման համապատասխան հրամանները: Դա կարող է լինել **print** հրամանը (perl կամ մի քանի այլ սցենարային լեզուների օգտագործման դեպքում), **echo** հրամանը (Unix, Windows, PHP), **printf** (C լեզուն) և տվյալները էկրանի, կոնսոլի կամ որևէ տերմինալի վրա ելքագրելու այլ հրամանները: Դրանց օգտագործումը HTML կոդը արտապատկերելու համար ունի նույն էֆեկտը, ինչ այդ կոդը հավաքվել որևէ տեքստային խմբագրի միջոցով: Օրինակ, Perl-ով գրված CGI ծրագրի մի մասը կարող լինել այսպիսին՝

```
Print "Content-type: text/html\n\n";
Print "<html>\n<head><title>Շնորհակալագիր</title></head>\n"
Print "<body>\n<h1>Ամեն ինչ կարգին է</h1>\n<p>";
Print "Շնորհակալություն ուշադրության համար</p>\n";
Print "<p>Սեղմեք կոճակին<a href='index.html'> գլխավոր էջ";
Print"</a>վերադառնալու համար</p>\n</body></html>";
```

stdin և stdout-ը փոխարինում են սերվերի վիրտուալ օգտվողի կողմից ստեղծաշարի ստեղծների սեղմումը: Մասնավորապես stdout գործողությունը փոխարինում է փաստաթղթի ելքագրումը բրաուզերի էկրանին: Իրականում տվյալները հաղորդվում են հեռացված համակարգչի բրաուզերին, որն էլ արտապատկերում է դրանք էկրանին:

CGI ծրագրերի տեղադրումը սերվերների վրա հաճախ կապված է մի շարք դժվարությունների հետ, քանի որ հիմնականում այդ տեսակի ծրագրերը տեղադրվում են հատուկ cgi-bin կատալոգում, իսկ դրա համար, մասնավորապես, պահանջվում է սերվերի ադմինիստրատորի համաձայնությունը (կան և ուրիշ պրոբլեմներ՝ կապված կատալոգի նորացման և այլ հարցերի հետ):

Ակտիվ Սերվերային էջեր (Active Server Pages - ASP) սերվերային ծրագրերի ստեղծման տեխնոլոգիան թույլ է տալիս շրջանցել այդպիսի դժվարությունները, քանի որ բոլոր ASP ֆայլերը յուրաքանչյուր “սայտաշինարար” կարող է տեղադրել սերվերի իրեն “հասանելիք տիրույթում”:

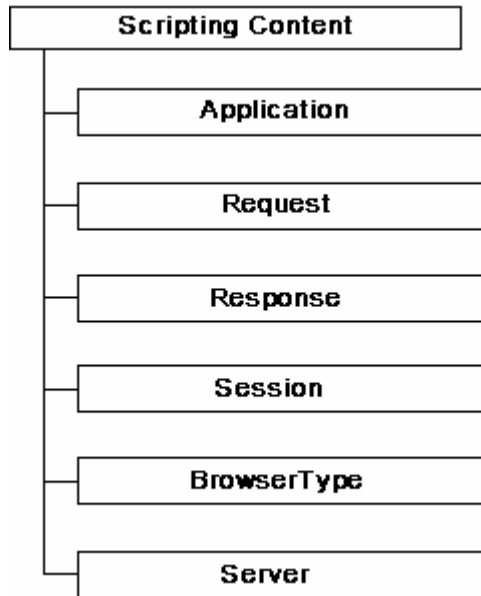
§4.2. Ակտիվ սերվերային էջեր (Active Server Pages)

4.1.1. ASP-ի օբյեկտային մոդելը

Ըստ էության Ակտիվ Սերվերային (ASP) էջերը իրենցից ներկայացնում են ստատիկ Web էջեր, որոնք բացի սովորական HTML կոդից ընդգրկում են նաև սերվերային սցենարներ, որոնք մշակվում են կամ սերվերի, կամ հատուկ սերվերային ծրագրերի (CGI ծրագրերի, Web սերվերի ընդլայնումների և այլն) կողմից: Սցենարները կարող են գրվել VBScript կամ JavaScript լեզուներով, սակայն ցանկության դեպքում անկախ ֆիրմաները կարող են ապահովել ուրիշ լեզուների սատարում օրինակ Rexx, Perl և այլն: Որպես հետևանք կամայական HTML էջ հնարավոր է հեշտորեն ձևափոխվել ASP էջի: Սցենարների միջոցով ASP էջերում ստեղծվում է HTML կոդ, որը հաղորդվում է հաճախորդին և արտապատկերվում նրա բրաուզերի էկրանի վրա: Օրինակ ASP–ն թույլ է տալիս կապակցել HTML էջին ActiveX բաղկացուցիչներ տվյալների բարդ մշակումներ կատարելու համար, աշխատել տվյալների բազաների հետ և այլն: Բոլոր ստանդարտ միջոցները, որոնք անհրաժեշտ են Web էջեր կառուցելու համար, առանձնացվում են օբյեկտներում, որոնք կարելի է բաժանել երկու հիմնական կատեգորիաների՝ ներկառուցված և բազային (տես՝ պատկեր 4.2.1.): Ներկառուցված օբյեկտների մեթոդները և հատկությունները թույլ են տալիս ստանալ մանրամասն ինֆորմացիա ընդուն-

վող հարցումների մասին, մշակել cookie ֆայլերը և կառուցել հաճախորդին վերադարձվող պատասխանները:

Ներկառուցված օբյեկտներ: ASP-ում գոյություն ունեն մի քանի ներկառուցված օբյեկտներ, որոնք ազատում են ծրագրավորողին որոշակի ոչ ստեղծագործական աշխատանքից: Օրինակ URL-ից պարամետրերի դուրս բերման, հաճախորդի cookie ֆայլերի ստացման, պահպանման կամ էլ հաճախորդին ելքային տվյալների հաղորդման համար:



Պատկեր 4.2.1. ASP-ի օբյեկտային մոդելը

Request (հարցում, պահանջ) օբյեկտը նախատեսված է ընթացիկ օգտվողին (հաճախորդին, որը տվյալ պահին ուղարկել է հարցումը) վերաբերվող ինֆորմացիան ստանալու համար: Այն հասանելի է դարձնում HTTP հարցման մեջ պարունակվող ամբողջ ինֆորմացիան: Դա կարող է լինել ընթացիկ օգտվողին բնութագրող ինֆորմացիան, որոշ տվյալներ հարցման առաջնայնության վերաբերյալ և արգումենտներ, որոնք թելադրում են ինչպես մշակել հարցումը և ձևակերպել պատասխանը: Սովորաբար Request օբյեկտը օգտագործվում է ֆորմայի էլեմենտների արժեքները կամ

հաճախորդի գոյությունն ունեցող cookie ֆայլերի պարունակությունը ստանալու համար:

Response (պատասխան) օբյեկտի միջոցով կառուցվում է հարցման պատասխանը, որը փաստորեն հանդիսանում է HTML փաստաթուղթ: Response օբյեկտը հաղորդում է տվյալները ելքային հոսքին (CGI ծրագրերում՝ stdout), որն ուղղվում է Web սերվերից հաճախորդին:

Session օբյեկտում պահպանվում է ինֆորմացիան առանձին օգտվողի մասին ընթացիկ սեանսի (սեսիայի) ընթացքում: Յուրաքանչյուր շահագործողի համար ստեղծվում է առանձին Session օբյեկտ, որի հիմնական ֆունկցիան՝ վիճակի ինֆորմացիայի պահպանելն է: Session օբյեկտում կարելի է սահմանել փոփոխականներ (web դասեր), որոնք պահպանում են իրենց արժեքները նույնիսկ հավելվածի այլ էջերին անցնելու ժամանակ: Session օբյեկտի փոփոխականների արժեքները զրոյացվում են շահագործողի հետ սեանսը ավարտելիս:

Application օբյեկտում պահպանվում է նույն ինֆորմացիան, ինչ և Session օբյեկտում, սակայն արդեն բոլոր օգտվողների վերաբերյալ: Օբյեկտի հատկությունները թույլ են տալիս ստանալ կամ շնորհել արժեքներ այն փոփոխականներին, որոնց հետ աշխատում են հավելվածի բոլոր շահագործողները:

Server օբյեկտի օգնությամբ սերվերային ծրագրերում ստեղծվում են ActiveX բաղկացուցիչների օբիեկտներ, որոնք պահպանվում են սերվերի վրա: Server օբյեկտի CreateObject մեթոդը գործնականում նույնն է ինչ որ Visual Basic-ի նույնանուն ֆունկցիան: Այն հնարավորություն է տալիս իրականացնել ActiveX բաղկացուցիչների ֆունկցիոնալ հնարավորությունները հավելվածներում: Բացի դրանից Server օբյեկտը թույլ է տալիս ստեղծել օբյեկտային փոփոխականներ և օժտել Web էջերը HTML-ի շրջանակներից դուրս եկող հնարավորություններով:

Բազային օբյեկտները ապահովում են Web ծրագրավորումում օգտագործվող ստանդարտ միջոցների աշխատանքը, օրինակ՝ տվյալների բազայի հետ աշխատանքի համար ADO (ActiveX Data Objects) բաղկացուցիչները և ֆայլային համակարգերի բաղկացուցիչները սերվերի վրա լոկալ ֆայլերը կարդալու/գրելու համար: Web հավելվածների հետ աշխատանքը պարզեցնելու նպատակով Web սերվերի ASP բաղադրիչները սատարում են մի շարք օբյեկտներ, որոնց մեթոդների օգտագործումը, ի տարբերություն ներկառուցված օբյեկտների, պարտադիր չէ, սակայն դրանք հաճախ են կիրառվում Web հավելվածների կառուցման գործընթացում:

Տվյալների բազաների սատարման օբյեկտները հնարավորություն են տալիս միանալ ցանկացած ODBC համատեղելի տվյալների բազաներին կամ OLE DB տվյալների աղբյուրին: Դրանցում օգտագործվող ADO տեխնոլոգիան թույլ է տալիս ծրագրավորողին հեշտորեն իրականացնել կապը տվյալների բազայի և ակտիվ Web էջերի միջև:

Ֆայլերին դիմումների սատարման օբյեկտները թույլ են տալիս աշխատել սերվերի վրա տեքստային ֆայլերի հետ: Սերվերային և հաճախորդի (կլիենտական) սցենարները պետք է անվտանգ լինեն համակարգի համար և այդ տեսակետից տվյալ կատեգորիայի օբյեկտները չափազանց հուսալի են, քանի որ թույլ են տալիս կարդալ և գրել միայն տեքստային ֆայլերը և մշտապես գտնվում են միայն սերվերի կոնկրետ թղթապանակում:

Էջերի միջև կապի սատարման օբյեկտները պաշտպանում են URL հասցեների ցուցակ, որը թույլ է տալիս աշխատել Web հանգույցի էջերի հետ այնպես, ինչպես գրքի էջերի հետ: Ծրագրավորողը ստեղծում է էջերի պարունակությունը, իսկ տվյալ կատեգորիայի օբյեկտները ավտոմատ կերպով ստեղծում են հղումներ նախորդ և հաջորդ Web էջերին: Օբյեկտների մեթոդները և հատկությունները թույլ են տալիս ավելացնել, հեռացնել և տեղադրել հանգույցի էջերը առանց առանձին էջերի խմբագրման:

Բրաուզերի բնութագրման օբյեկտների օգնությամբ ASP ֆայլերը կարող են որոշել հարցում կատարող բրաուզերի հնարավորությունները և դիմամիկորեն օպտիմիզացնել հանգույցի պարունակությունը կոնկրետ առանձնահատկություններին համապատասխան: Այն բրաուզերների համար, որոնց տեսակը չի որոշվում ավտոմատ կերպով, առաջանում է կրկնվող էջերի սերիայի ստեղծման (և ուղեկցման) անհրաժեշտություն: Ընդ որում դա պետք է կատարվի յուրաքանչյուր բրաուզերի համար: Հակառակ դեպքում անհրաժեշտ է հաղորդել շահագործողներին այն մասին, որ հանգույցի որոշ հնարավորությունները հասանելի են միայն որոշակի տիպի բրաուզերների համար:

Գովազդների հերթագրման օբյեկտներ: Ներկայումս բոլոր մեծ Web հանգույցները աշխատում են կոմերցիոն հիմունքներով և այդ պատճառով հաճախ առաջանում է գովազդների ղեկավարման անհրաժեշտությունը: Լուրջ Web վարպետը թույլ չի տա իրեն ձեռքով մուտքագրել գովազդը էջում: Ցանկալի է ոչ միայն հերթի դնել գովազդը, այլև ընտրել այն շահագործողի ճաշակին համապատասխան: Օրինակ՝ իմաստ չունի էլեկտրոնիկայի իրացումով զբաղվող խանութում գովազդել ավտոմեքենաներ, գրքեր և այլն:

Գովազդների հետ աշխատանքը պարզեցնելու համար ASP–ում նույնպես գոյություն ունեն հատուկ օբյեկտներ:

4.2.2. ASP էջերի ստեղծման տեխնոլոգիան

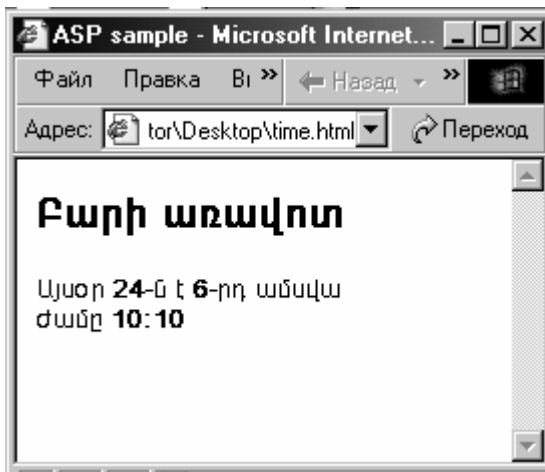
Ակտիվ Սերվերային էջ (ASP) ստեղծելու պարզագույն եղանակը՝ գոյություն ունեցող HTML փաստաթղթի փոփոխումն է կամ ASP-ի ընդլայնումը HTML–ով: Այնուհետև ֆայլը տեղադրվում է Web սերվերի համապատասխան թղթապանակում: Կազմենք օրինակ՝ DateTime.html ֆայլը, որը ցուցադրում է հաճախորդին ընթացիկ ամսաթիվը և ժամանակը (Ծր. 4.2.1. և պատկեր 4.2.2.):

Ծր.4.2.1. Ընթացիկ ամսաթիվը և ժամը արտապատկերող HTML էջը

```
<html>
<head><title>ASP sample</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script>
<!--
var v_date=new Date();
dat=v_date.getDate();
hour=v_date.getHours();
minut=v_date.getMinutes();
month=v_date.getMonth()+1;
if(hour<=18)
greeting="Բարի օր";
if(hour<=12)
greeting="Բարի առավոտ";
if(hour>18)
greeting="Բարի օր";
if(minut<10)
minut="0"+minut;
document.write(" <h2>"+greeting+"</h2>");
document.write("<p>Այսօր <b>" +dat+"</b>-ն է <b>"+month+"</b>-
րդ ամսվա<br />ժամը <b>"+hour+"."+minut+"</b></p>");
//-->
</script>
</head>
<body>
</body>
</html>
```

Այժմ վերամշակենք էջը այնպես, որ այն ելքագրի ոչ թե հաճախորդի համակարգչի, այլ սերվերի ընթացիկ ժամը: Այդ

նպատակով օգտագործում ենք Response օբյեկտը, որը նման է document օբյեկտին, սակայն սերվերը չի կարող դիմել document օբյեկտին, քանի որ վերջինս հասանելի է միայն հաճախորդի համար: Նոր էջը տալիս է նույն տվյալները, բայց ամսաթիվը և ժամը համապատասխանում է սերվերի ամսաթվին և ժամին:



Պատկեր 4.2.2. Ընթացիկ ամսաթիվը և ժամը արտապատկերող HTML էջի օրինակ

Ծրագրային կոդի այն մասը, որը կատարվում է սերվերի վրա ASP ֆայլերում ամփոփվում է `<%` և `%>` զույգ տեգերի միջև: Այդ տեգերի միջև գտնվող բոլոր հրամանները, կատարվում են միայն սերվերի վրա և անհասանելի են հաճախորդին: Եթե անհրաժեշտ է ցուցադրել հաճախորդին որևէ փոփոխականի արժեքը, ապա դրա անունը ամփոփվում է `<%= %>` տեգում՝ “=” նշանից հետո: Օրինակ՝ եթե գրանցվի՝

```
<%var greeting="Բարև Ձեզ"; %>
```

```
<b><%=greeting%></b>
```

ապա հաճախորդի բրաուզերի պատուհանում կարտապատկերվի թավ (bold) տառաչարով գրված Բարև Ձեզ տողը (և ոչ թե greeting փոփոխականի անունը):

Քանի որ ներկայումս ակտիվ սերվերային էջերը կազմվում են և VBScript և JavaScript (վերջերս նաև Microsoft-ի ստեղծած JScript – JavaScript-ի վերափոխված տարբերակով՝ այդ դեպքում էջը կոչվում է JSP) լեզուներով, ապա կամայական ASP փաստաթղթի

սկզբնամասում սովորաբար գրանցվում է համապատասխան կարգագիրը, որպեսզի սերվերը ավելի հեշտ կարողանա գործարկել համապատասխան ինտերպրետատորը՝

<%@LANGUAGE="VBSCRIPT"%>, երբ կիրառվում է VBScript-ը,

<%@LANGUAGE="JAVASCRIPT"%> JavaScript-ի և

<%@LANGUAGE="JSCRIPT"%> JScript-ի օգտագործման դեպքում:

Վերափոխված Ծր. 4.2.1-ը կընդունի հետևյալ տեսքը՝

Ծր. 4.2.2. Ընթացիկ ամսաթիվը և ժամը արտապատկերող ASP էջը

<%@LANGUAGE="JAVASCRIPT"%>

<html>

<head>

<title>ASP sample</title>

</head>

<body>

<% var v_date=new Date();

dat=v_date.getDate();

hour=v_date.getHours();

minut=v_date.getMinutes();

month=v_date.getMonth()+1;

if(hour<=18)

greeting="Բարի օր";

if(hour<=12)

greeting="Բարի առավոտ";

if(hour>18)

greeting="Բարի օր";

if(minut<10)

minut="0"+minut;

%>

<h2><%=greeting%></h2>

<p>Այսօր <%=dat%>-ն է <%=month %>-րդ ամսվա

ժամը <%=hour %>:<%=minut%></p>

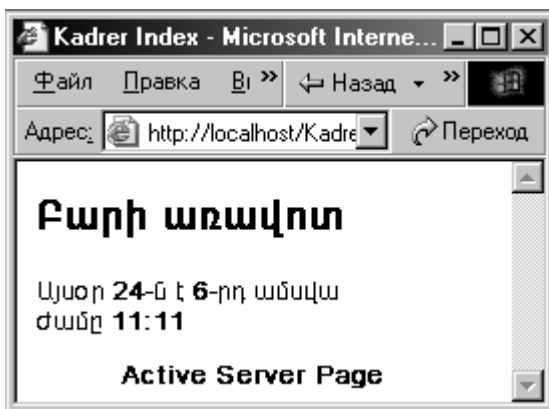
 Active Server Page

</body>

</html>

Ինչպես կարելի է տեսնել սերվերի կողմում (սերվերի վրա) կատարվող ծրագրային կողը ամփոփված է <% %> տեգում, իսկ այն փոփոխականների արժեքները, որոնք ցուցադրվելու են հաճախորդին (մեր դեպքում՝ hour, minut, month և dat) ներկառուցված են

սովորական HTML կոդում `<%= %>` տեղի միջոցով: Արդյունքը ցուցադրված է պատկեր 4.2.3-ում:



Պատկեր 4.2.3. Ընթացիկ ամսաթիվը և ժամը արտապատկերող ASP էջի օրինակ

4.2.3. Ընդգրկվող ֆայլերի օգտագործման կանոնները

Ինչպես և սովորական HTML էջերում, այնպես էլ սերվերային ծրագրերում ծրագրային կոդի որոշ մասերը կարող են օգտագործվել մի քանի սցենարներում: Ավելորդ կրկնություններից խուսափելու համար կարելի է ձևակերպել դրանք առանձին ֆայլերի տեսքով և ընդգրկել կամայական ASP ֆայլի սկզբում `#include` հրահանգի միջոցով: Հրահանգն ունի հետևյալ սինտաքսիսը՝

`<!--#include virtual="ընդգրկվող ֆայլի անունը"-->`

Մեկնաբանման տեղը (`<!-- -->`) երաշխավորում է, որ տողը սխամամբ չի փոխանցվի հաճախորդին: Եթե ֆայլը գտնվում է միևնույն թղթապանակում (կամ էլ դրան ներդրված թղթապանակներից մեկում) օգտագործվում է **virtual** բանալիային բառը:

Եթե ընդգրկվող ֆայլը գտնվում է ընդգրկողից ավելի բարձր մակարդակի կամ մի այլ թղթապանակում, ապա դրան դիմելու համար `virtual`-ի փոխարեն օգտագործում են **file** բանալիային բառը և որպես արժեք գրանցում §2.3-ում բերված կանոններով ձևակերպված ընդգրկվող ֆայլի ուղին: Օրինակ՝

`<!--#include file="../Connections/Library.asp" -->`

Ընդգրկվող ֆայլերը կարող են ունենալ ցանկացած ընդլայնում, բայց սովորաբար օգտագործվում է `.inc` ընդլայնումը (`include`

բառից): Հետևյալ հրահանգը ASP էջում ընդգրկում է DateFunc.inc ֆայլը, որը գտնվում է Support թղթապանակում:

```
<!--# include virtual ="/ASP/Support/DateFunc.inc"-- >:
```

Ընդգրկվող ֆայլերը իրենց հերթին կարող են ընդգրկել այլ ֆայլեր: Այդպիսի դեպքերում պետք է հետևել, որպեսզի չառաջանա #include հրահանգների պարբերություն: Ենթադրենք Doc1.asp ֆայլը ընդգրկում է file1.inc-ը, որն իր հերթին ընդգրկում է մի այլ file2.inc ֆայլը: Սակայն file2.inc այդ ժամանակ չի կարող ընդգրկել file1.inc ֆայլը (բացի դրանից ֆայլը չի կարող ընդգրկել ինքն իրեն): Ընդգրկող ֆայլը պետք է պարունակի ավարտուն սցենարներ: Այլ կերպ ասած սցենարը չի կարող սկսվել մեկ ընդգրկվող ֆայլում և ավարտվի մյուսում: Բացի դրանից ֆայլը չի կարող ընդգրկվել սցենարի միջնամասում:

Օրինակ հետևյալ սցենարը չի աշխատի՝

```
< %  
սերվերային հրամաններ  
<!-- # include virtual="DateFunctions.inc"-- >  
այլ հրամաններ  
% >
```

Անհրաժեշտ է այն տրոհել հետևյալ կերպով.

```
< %  
սերվերային հրամաններ  
% >  
<!--#include virtual="DateFunctions.Inc" -->  
< %  
այլ հրամաններ  
% >
```

ASP ֆայլերում կարող են պարունակվել նաև հաճախորդի կողմում աշխատող սցենարներ, որոնք ընդգրկվում են հաճախորդին ուղարկվող web էջում <script> տեգի միջոցով: ServerTime.asp ֆայլը ելքագրում է հաճախորդի համակարգիչի ժամը սցենարի հաճախորդի մասում, իսկ սերվերային ժամը՝ սցենարի սերվերային մասում: Եթե հաճախորդը և սերվերը գտնվում են տարբեր ժամային գոտիներում, տարբերությունը կազմում է ժամերի քանակը պլյուս մի քանի վայրկյան, որի ընթացքում HTML-ի կողը ստացվում է հաճախորդի համակարգիչի վրա: ServerTime.asp էջը (սցենարները գրված են VBScript-ով) ունի հետևյալ տեսքը՝

```
<html>  
<head>  
<Title> Պարզ ASP –ի ցուցադրում </Title>
```

```

<script language="VBScript">
document.Write "<font size=3 face='Arial Armenian'>"
document.Write "<h1> Բարի գալուստ ASP </h1>"
document.Write "Այսօրվա ամսաթիվն է <b>" & Date() & "</b> և այս
պահին ժամը <b>" & Time() & " է</b>"
document.Write "<p>"
</script>
</head>
<body>
<b> Սերվերի վրայի ժամն է <%=Time() %> </b>
<br />
<b> Active Server Page </b> պարունակում է որոշակի տեքստ
</body>
</html >

```

§4.3. Response և Requeste օբյեկտների կիրառությունը

4.3.1. Response օբյեկտը

Response օբյեկտը օգտագործվում է դիմամիկորեն ստեղծված HTML էջերը հաճախորդի բրաուզերին հաղորդելու համար: Օբյեկտն օգտագործում է IResponse ինտերֆեյսը, որի միջոցով հնարավորություն է ընձեռվում էջերի բուժերացումը, cookie-ների արժեքների շնորհումը, web սերվերի անվանակոչումը և բրաուզերի էջերի "կեշավորումը" (այսինքն որոշ ֆայլերի ժամանակավոր պահպանումը հենց օգտվողի համակարգչի հիշողությունում): Այն ունի բազմաթիվ մեթոդներ, որոնց միջոցով կարելի է լուծել մի շարք խնդիրներ:

ASP ֆայլերում գրանցվող սովորական HTML կոդերը անմիջապես ուղարկվում են հաճախորդին: Այն դեպքում, երբ ելքային տվյալները անհրաժեշտ է դեկավարել սցենարից, դրանք գրանցվում են Response օբյեկտի միջոցով: Հենց Response.Write մեթոդի կիրառությունն էլ հանդիսանում է ASP էջերի ստեղծման երկրորդ եղանակը:

- **Write մեթոդը:**

Response.Write մեթոդը շատ հաճախ է օգտագործվում ASP ֆայլեր ստեղծելու համար: Ենթադրենք մենք կառուցում ենք տարբեր մակարդակի վերնագրեր պարունակող էջ, ընդ որում վերնագրի մակարդակը որոշվում է ihead փոփոխականի միջոցով: Օրինակ, եթե ihead փոփոխականը հավասար է 2-ի, տեղադրում է երկրորդ մակարդակի վերնագիր՝
Response.Write "<h" & ihead & ">"

“&” նիշը օգտագործվում է VBScript-ում տողերը միմյանց հետ կցելու և մեկ տողի վերափոխելու համար: JavaScript-ում այդ նույն գործառույթը կատարում է “+” նշանը: Այսինքն, եթե վերը բերված հրամանը գրանցվի JavaScript-ով, ապա այն կընդունի հետևյալ տեսքը՝

```
Response.Write (“<h” + ihead + “>”);
```

Եթե ihead=2, ապա հրամանի կատարման արդյունքում հաճախորդի բրաուզերը կարտապատկերի <h2> տեսակի տեգ:

Բերենք երկու լեզուներով կազմած Response.Write մեթոդի կիրառության պարզագույն օրինակ:

----- VBScript-----

```
<%@LANGUAGE="VBSCRIPT"%>
```

```
<%
```

```
Response.Write “<html >”
```

```
Response.Write “<head >”
```

```
Response.Write “<title > Response.Write Demo </title >”
```

```
Response.Write “</head ><body>”
```

```
Response.Write “<h1>”
```

```
Response.Write “Response Object: Write Method”
```

```
Response.Write “</h1>”
```

```
Response.Write “</body></html>”
```

```
%>
```

----- JavaScript-----

```
<%@LANGUAGE="JAVASCRIPT"%>
```

```
<%
```

```
Response.Write (“<html >”);
```

```
Response.Write (“<head >”);
```

```
Response.Write (“<title > Response.Write Demo </title >”);
```

```
Response.Write (“</head ><body>”);
```

```
Response.Write (“<h1>”);
```

```
Response.Write (“Response Object: Write Method”);
```

```
Response.Write (“</h1>”);
```

```
Response.Write (“</body></html>”);
```

```
%>
```

- **Redirect մեթոդը:**

Redirect մեթոդը վերաուղղում է հաճախորդին այլ URL հասցեի: Օրինակ այն դեպքում, երբ web հանգույցը վերադրվում է, մեթոդը կարելի է օգտագործել հաճախորդներին նոր URL-ի ավտոմատ կերպով վերահասցեավորող ASP հավելված՝

```
<%@LANGUAGE="VBSCRIPT"%>
```

```
<html>
```



```
<% Response.Redirect NewURL % >  
</html>
```

Նոր URL հասցեն պահպանվում է NewURL փոփոխականում, կամ գրանցվում ակնհայտ եղանակով:

Առավել հաճախ Redirect մեթոդը օգտագործվում է ֆորման պարունակող էջից մի այլ (օրինակ, արդյունքների ամփոփման) էջին անցնելու համար, երբ ֆորման արդեն լրացված և ուղարկված է սերվերին:

- **Clear մեթոդը:**

Մեթոդը ջնջում է Response օբյեկտի բուֆերացված տվյալները: Մեթոդի միջոցով հնարավոր է ջնջել միայն պատասխանի (ձևավորված էջի) մարմինը և ոչ վերնագրային ինֆորմացիան (օրինակ՝ Cookie-ները): Եթե Buffer հատկությունը հավասար է FALSE, ապա մեթոդի կիրառությունը կարող է բերել ծրագրի կատարման սխալների:

- **Flush մեթոդը:**

Օգտագործելով այս մեթոդը, կարելի է կատարել նաև էջի առանձին մասերի բուֆերացումը, այսինքն ուղարկել հաճախորդին առանձին պատրաստի մասերը: Նախապես Buffer հատկությամբ պետք է շնորհել TRUE արժեք: Օրինակ՝

```
<%@LANGUAGE="VBSCRIPT"%>
```

```
<html>
```

```
<%
```

```
Response.Buffer=TRUE
```

```
... ASP հրամաններ...
```

```
Response.Flush
```

```
...ASP հրամաններ
```

```
% >
```

```
</html>
```

Բերված ծրագրային հատվածում Response.Flush հրամանի միջոցով հաճախորդին անմիջապես ուղարկվում է դրան նախորդող հրամանների կատարման արդյունքը:

- **ContentType հատկությունը:**

Այն որոշում է հաճախորդին ուղարկվող փաստաթղթի տիպը: ContentType հատկությունը վերաբերվում է ամբողջ էջին և գրանցվում ինֆորմացիայի ելքային հոսքի սկզբում Write մեթոդով: Հատկության արժեքը ըստ լռելյան հավասար է application/x-www-form-urlencoded: Հետևյալ օրինակներում հատկությամբ շնորհվում են տարբեր արժեքներ՝

```
<% Response.ContentType = "text/plain" %>
```

```
<% Response.ContentType = "image/GIF" %>
```

<% Response.ContentType = "image/JPEG" %>

- Buffer հատկությունը:

Որոշ դեպքերում առաջանում էջերի բուժերացման անհրաժեշտություն, այսինքն էջը պետք է ուղարկվի հաճախորդին միայն այն դեպքում, երբ կատարված են դրան ձևավորող բոլոր ASP հրամանները: Այդ դեպքում Buffer հատկությանը շնորհվում է "TRUE" արժեքը՝ Response.Buffer="TRUE" (բուժերացումը օգտագործվում է, օրինակ այն դեպքերում, երբ էջը պետք է վերաուղղել մի այլ URL-ի):

Ըստ լռելյայն Response օբյեկտը ելքային տվյալները ուղարկում է հաճախորդին անմիջապես՝ ASP ծրագրի հրամանների կատարման ընթացքում, չսպասելով հաջորդ հրամանների կատարմանը (այսինքն էջի լիակատար ձևավորմանը): Դա համապատասխանում է Buffer հատկության FALSE արժեքին:

Եթե էջի մշակման ժամանակ հայտնաբերվել է, որ պետք չէ հաճախորդին ուղարկել նախորդ տվյալները, ապա շնորհելով Buffer հատկությունը TRUE արժեքը, կարելի է դադարեցնել մշակումը և մաքրել ելքային հոսքը Response օբյեկտի Clear մեթոդով:

Տիպիկ սցենարը կունենա հետևյալ տեսքը՝

<%

Response.Buffer=TRUE

{սցենարի հրամանները}

If SupplierName=UserName Then

Response.Clear

Response.Redirect "/Suppliers/ AllSuppliers.html"

Response.End

End If

%>

Սցենարում ստեղծվում է էջ (օրինակ՝ կարդալով տվյալները տվյալների բազայից): Երբ հայտնաբերվում է, որ հաճախորդը իրականում հանդիսանում է առաքողներից մեկը պատրաստ ելքային տվյալները ջնջվում են, իսկ հաճախորդը վերաուղղվում է այլ էջի: Սակայն եթե հաճախորդը երկար ժամանակ չի ստանում պատասխան նա կարող է որոշել, որ իրեն հարցումը չի մշակվում: Նորմալ HTML էջերը չեն բուժերացվում, ուստի սովորական իրավիճակներում ASP էջերը նույնպես չպետք է բուժերացվեն:

Response օբյեկտը ունի ևս մի քանի հատկություններ, որոնցից կարելի է մշել Charset, Status և Expires հատկությունները: Սակայն առավել հետաքրքիր և կիրառվող է օբյեկտի միակ հավաքածուն:

- Cookies հավաքածուն:

Այն օգտագործվում է հաճախորդին Cookie ֆայլեր հաղորդելու համար: Դրանք հատուկ տողեր են, որոնք պահվում են լոկալ համակարգչի վրա և կարդացվում են ASP հավելվածի կողմից Request մեթոդի օգնությամբ: Cookie-ները ապահովում են web հանգույցների էջերի միջև որոշ` հետագա օգտագործման համար անհրաժեշտ տվյալների պահպանման հնարավորություն: Web-ում օգտագործվող HTTP արձանագրությունը չի նախատեսում վիճակ հիշելու հնարավորությունը: Երբ հաճախորդը կատարում է հարցում, այն փոխանցվում է սերվերին, որը համապատասխան սերվերային ծրագրի միջոցով պատասխանում է այդ հարցմանը: Դրանով տրանզակցիան ավարտվում է: Եթե նույն հաճախորդը հարցնում է նույն սերվերից մի այլ փաստաթուղթ, ոչ հաճախորդը և ոչ էլ սերվերը չեն հիշում նախորդ տրանզակցիան: Էջերի միջև փոփոխականների արժեքների պահպանման պրոբլեմը լուծվում է Cookie ֆայլերի օգնությամբ: Եթե Web էջերը օգտագործում են ֆոնի նույն գույնը, որը ընտրվում է շահագործողի կողմից, ապա այն կարող է պահպանվել Cookie ֆայլում և կարդացվել նոր էջի կառուցումից և ուղարկումից առաջ (պահպանված փոփոխականի արժեքը կարդալու համար օգտագործվում է Request օբյեկտի Cookies հատկությունը):

Cookie-ի ստեղծման հրամանի կառուցման քերականությունը հետևյալն է`

Response.Cookies(“անունը”)(“բանալի”) [.ատրիբուտ]=արժեք

“Բանալի” և ատրիբուտը ոչ պարտադիր պարամետրեր են և օգտագործվում են այն դեպքերում, երբ կառուցվում են Cookie-ների այսպես կոչված **բառարաններ**, այսինքն միևնույն անունով, բայց տարբեր բանալիներ ունեցող Cookie-ներ: Օրինակ եթե ստեղծվում են ընդհանուր “color” անունով մի քանի տարբեր գույնային արժեքներ ունեցող Cookie-ներ, ապա ծրագրային կոդը կլինի հետևյալը`

<%

Response.Cookies(“COLOR”)(“col1”)="green"

Response.Cookies(“COLOR”)(“col2”)="red"

Response.Cookies(“COLOR”)(“col3”)="dark blue"

%>

Response օբյեկտի վերնագրային մասում (հիշենք, որ Clear մեթոդը այն չի ջնջում) կգրանցվի`

Set-Cookie: COLOR=col1=green&col2=red&col3=dark+blue

Եթե այժմ ստեղծվի նույնանուն Cookie, որը չունի բանալի, ապա բոլոր նախորդ ստեղծվածները կջնջվեն: Օրինակ, եթե գրանցվի`

Response.Cookies(“COLOR”)="black",

ապա նախորդ երեք Cookie-ները կվերանան: Սակայն եթե գրանցենք՝

```
Response.Cookies("COLOR") ("col4") ="black",
```

ապա այդ դեպքում կպահպանվեն նաև նախորդները:

Ատրիբուտների միջոցով Cookie-ներին շնորհվում են (կամ ստուգվում) որոշակի լրացուցիչ բնութագրեր՝

- ◆ Domain - ազդարարում է, որ տվյալ Cookie-ն կուղարկվի միայն տվյալ դոմեյնից եկած կանչերին;
- ◆ Expires - սահմանում է գործածության ժամկետը,
- ◆ Path - ազդարարում է, որ տվյալ Cookie-ն կուղարկվի միայն տվյալ հասցեից եկած կանչերին;
- ◆ Secure – սահմանում է պաշտպանվածությունը (կարող է ընդունել երկու արժեք՝ TRUE և FALSE):

Օրինակ՝

```
<%
```

```
Response.Cookies("COLOR") = "black"
```

```
Response.Cookies("COLOR").Expires = "July 31, 2010"
```

```
Response.Cookies("COLOR").Domain = "msn.com"
```

```
Response.Cookies("COLOR").Path = "/www/home/"
```

```
Response.Cookies("COLOR").Secure = FALSE
```

```
%>
```

4.3.2. Request օբյեկտը

Հաճախորդների հետ նորմալ փոխազդեցության համար սկրիպտը պետք է կարողանա կարդալ հաճախորդից ստացած ինֆորմացիան, ինչպես նաև Cookie ֆայլերի արժեքները:

Request օբյեկտն ունի մեկ հատկություն՝ **TotalBytes**, որի միջոցով որոշվում է հաճախորդից ստացած հարցման մարմնում պարունակվող բայտերի քանակը: Հետևյալ սկրիպտում սահմանվում է փոփոխական, որին շնորհվում է Request օբյեկտում պարունակվող հարցման բայտերի ընդհանուր քանակը՝

```
<%
```

```
Dim bytecount
```

```
bytecount = Request.TotalBytes
```

```
%>:
```

Օբյեկտի միակ՝ **BinaryRead(count)** մեթոդը թույլ է տալիս վերադարձնել զանգվածի տեսքով post մեթոդով ուղարկված հարցման count քանակությամբ մասը և պահպանել այն SafeArray (կամ Variant) զանգվածում: Ստորև բերված օրինակում վերականգնվում են հարցման բոլոր բայտերը՝

```
<%
```

```
Dim vntPostedData, lngCount
```

```

lngCount = Request.TotalBytes
vntPostedData = Request.BinaryRead(lngCount)
%>:

```

Առավել կարևոր և հաճախ օգտագործվող են Request օբյեկտի հինգ հավաքածուները (կոլեկցիաները)՝

- ◆ **QueryString** - վերադարձնում է GET մեթոդով ուղարկված հարցման տողի “?” նշանին հաջորդող մասը (այն հադիսանում QUERY_STRING սերվերային փոփոխականի վերլուծված վարկածը):
- ◆ **ServerVariables** - վերադարձնում է սերվերի բոլոր նախապես սահմանված միջավայրի փոփոխականները:
- ◆ **Cookies** - վերադարձնում է HTTP հարցման միջոցով ուղարկած բոլոր Cookie ֆայլերը:
- ◆ **Form** - վերադարձնում է ֆորմայի POST մեթոդով հաղորդված բոլոր էլեմենտների արժեքները (որոնք տեղադրված են հարցման մարմնում):
- ◆ **ClientCertificate** – վերադարձնում է web բրաուզերի սերտիֆիկացիայի դաշտերի արժեքները:

QueryString հավաքածուն: Հարցումը կարող է ձևավորվել մի քանի եղանակներով: Հետևյալ օրինակում այն կառուցվում է խա-րիսխի (հղումի) միջոցով՝

```

<a href= "example.asp?string=սա օրինակ է">տողի օրինակ</a>:

```

Կազմելով հետևյալ հրամանը՝

```

<% Response.Write(Request.QueryString) %>

```

արդյունքում կստանանք հետևյալ վերադարձվող արժեքը՝

```

string=սա+օրինակ+է:

```

Հարցման տողը կարող է ձևավորվել նաև ֆորմայի միջոցով, երբ վերջինի հաղորդման մեթոդը get-ն է: Փոփոխականների քանակը կարելի է այդ դեպքում որոշել կիրառելով հավաքածուի Count հատկությունը՝

```

<%

```

```

Dim varCount= Request.QueryString.Count

```

```

%>

```

Նախորդ օրինակում varCount փոփոխականի արժեքը հա-վասար կլինի 0-ի, քանի որ հարցումում չկա “անուն=արժեք” տե-սակի գրանցումներ:

Առանձին փոփոխականների արժեքները կարելի ստանալ երկու եղանակներով՝ փոփոխականի համարով զանգվածում կամ փոփո-խականի անունով: Օրինակ, եթե հարցման տողն ունի հետևյալ տեսքը՝

```

name=Վաղարշ&age=25,

```

ապա ստորև բերված երկու դեպքում էլ
 Ձեր անունն է՝ <%= Request.QueryString(0) %>
 Ձեր անունն է՝ <%= Request.QueryString("name") %>
 Ելքային էջում կարտապատկերվի՝ Ձեր անունն է՝ Վաղարշ տողը:
 Եթե հարցման տողի փոփոխականների անունները հայտնի չեն,
 ապա կարելի է ստանալ դրանց արժեքները օգտագործելով ցիկլի
 որևէ օպերատոր: Օրինակ, եթե հարցման տողը հետևյալն է՝
<http://localhost/script/names.asp?name=Ազատ&name=Շողիկ>,
 ապա ստորև բերված ծրագիրը (name.asp) կարտապատկերի բոլոր
 փոփոխականների արժեքները:

```
<%
  For Each item In Request.QueryString
    Response.Write Request.QueryString (item) & "<br />"
  Next
%>:
```

Արդյունքում կստացվի՝
 Ազատ
 Շողիկ:

Նույն մոտեցումը կարելի է կիրառել նաև այն դեպքերում, երբ
 հարցման տողն ընդգրկում է միևնույն անունով մի քանի փո-
 փոխականներ, այն տարբերությամբ, որ այդ անունով փոփո-
 խականները կազմում են առանձին ենթազանգված (վերջին
 բերված օրինակում դա կլինի Request.QueryString ("name") զանգ-
 վածը: name.asp ֆայլը կընդունի հետևյալ տեսքը՝

```
<%
  For Each item In Request.QueryString("name")
    Response.Write Request.QueryString("name")(item) & "<br />"
  Next
%>
```

Արտապատկերման արդյունքը կլինի նույնը, ինչ և նախորդ օրի-
 նակում:

ServerVariables հավաքածուն: Դրա գրանցման քերականությու-
 նը հետևյալն է՝

Request.ServerVariables (փոփոխականի_անունը):

Հավաքածույում պահպանվում է հաճախորդի և սերվերի վերա-
 բերյալ առկա ամբողջ ինֆորմացիան: Թվարկենք միայն առավել
 հաճախ օգտագործվող փոփոխականները.

- ◆ SERVER_NAME - սերվերի DNS անունը, հոստի անունը կամ
 IP հասցեն, երբ այն ձևակերպվում է ինքն իրեն դիմելու համար:
- ◆ SERVER_PROTOCOL - օգտագործվող արձանագրության
 վարկածի անունը և համարը, սովորաբար՝ HTTP / 1.1:

- ◆ **SERVER_PORT** – պորտի համարը, որից ստացվում է հարցումը, օրինակ՝ 80:
- ◆ **SERVER_SOFTWARE** - սերվերի ծրագրային ապահովման անունը և վարկածը: Գրանցվում է անուն/վարկած ֆորմատում, օրինակ՝ Apache/1.3.20(Darwin):
- ◆ **SCRIPT_NAME** - կատարվող սցենարի վիրտուալ ուղին և անունը, օրինակ՝ /scripts/printany.asp:
- ◆ **QUERY_STRING** - URL-ում (?) հարցական նշանին հետևող ինֆորմացիան, օրինակ՝ name=Վաղարշ&age=25:
- ◆ **REQUEST_METHOD** - մեթոդ, որը օգտագործվում է հաճախորդի տվյալները սերվերին ուղարկելու համար: Առավել հաճախ դա GET և POST արժեքներն են (կարող է լինել նաև PUT, HEAD և այլ):
- ◆ **CONTENT_TYPE** - սերվերին ուղարկվող տվյալների տեսակը: Ըստ լռելյայն հավասար է text/html:
- ◆ **CONTENT_LENGTH** - տողի երկարությունը, որը պարունակում է տվյալները: Փոփոխականը օգտագործվում է POST մեթոդի հետ համատեղ:
- ◆ **REMOTE_HOST** – հոստի անունը, որից կատարված է հարցումը: Երբ սերվերը չունի այդ ինֆորմացիան, այն շնորհում է արժեք **REMOTE_ADDR** փոփոխականին, իսկ տվյալ փոփոխականին շնորհում է դատարկ արժեք:
- ◆ **REMOTE_ADDR** - հաճախորդի IP հասցեն, որից ստացվել է հարցումը, օրինակ 192.168.1.5.

Հետևյալ ծրագիրը `servvar.asp` արտատպում է բոլոր սերվերային փոփոխականները աղյուսակի տեսքով՝

Ծր. 4.3.1. Սերվերային փոփոխականները արտատպող `servvar.asp` ֆայլը

```
<%@LANGUAGE="VBSCRIPT"%>
<html>
<head><title>ASP Process Environment</title>
</head>
<body>
<table border="1">
<thead><th>Փոփոխականը</th><th>Արժեքը</th></thead>
<% For Each strKey In Request.ServerVariables %>
<tr>
<td><%= strKey %></td>
<td><%= Request.ServerVariables(strKey) %></td>
</tr>
```

<% Next %>

</table>

<body>

</html>

Պատկեր 4.3.1-ում ցուցադրված է արդյունքում ստացվող աղյուսակի մի մասը (քանի որ ամբողջ աղյուսակը բավականաչափ ծավալուն է):

APPL_MD_PATH	/LM/W3SVC/1/ROOT
APPL_PHYSICAL_PATH	c:\inetpub\wwwroot\
CONTENT_LENGTH	0
CONTENT_TYPE	
GATEWAY_INTERFACE	CGI/1.1
HTTPS	off
PATH_INFO	/MySite/index.asp
PATH_TRANSLATED	c:\inetpub\wwwroot\MySite\index.asp
QUERY_STRING	
REMOTE_ADDR	127.0.0.1
REMOTE_HOST	127.0.0.1
REMOTE_USER	
REQUEST_METHOD	GET
SCRIPT_NAME	/MySite/index.asp
SERVER_NAME	localhost
SERVER_PORT	80
SERVER_PORT_SECURE	0
SERVER_PROTOCOL	HTTP/1.1
SERVER_SOFTWARE	Microsoft-IIS/5.0

Պատկեր 4.3.1. Սերվերի միջավայրի փոփոխականները

Cookies հավաքածուն. Հնարավորություն է ընձեռում ստանալ Cookie ֆայլերի արժեքները: Գրանցման քերականությունը հետևյալն է՝ **Request.Cookies(անուն)[(բանալին)].ատրիբուտը**:

Պարզագույն դեպքում, երբ Cookie-ն բառարան չէ (այսինքն չի պարունակում ենթաբանալիներով տարբերվող մի քանի արժեքներ), դրա արժեքը կարելի է ստանալ հետևյալ հրամանի օգնությամբ՝

```
<%= Request.Cookies("myCookie") %>:
```

Եթե երկու Cookie-ն ունեն միևնույն անունը և չեն տարբերվում ենթաբանալիով, ապա վերը բերված գրանցման դեպքում կարտատալի այն Cookie-ն, որի ուղու կառուցվածքը ավելի երկար է: Օրինակ՝ եթե երկու նույնանուն Cookie-ից մեկի ուղու կառուցվածքն է /www/, իսկ մյուսինը՝ /www/home/ և հաճախորդի բրաուզերում դրանք տեղադրված են /www/home/ դիրեկտորիայում, ապա Request.Cookies հրամանը կվերադարձնի միայն երկրորդ Cookie-ն:

Սակայն հնարավոր է ստանալ և բոլոր Cookie-ների անունները, և բոլոր ենթաբանալիների անվանումները: Դա կարելի է իրագործել, օրինակ, հետևյալ ծրագրի միջոցով՝

```
<%
```

```
For Each strKey In Request.Cookies
```

```
Response.Write strKey & " = " & Request.Cookies(strKey) & "<br />"
```

```
If Request.Cookies(strKey).HasKeys Then
```

```
For Each strSubKey In Request.Cookies(strKey)
```

```
Response.Write "- >" & strKey & "(" & strSubKey & ") = " & _
```

```
Request.Cookies(strKey)(strSubKey) & "<br />"
```

```
Next
```

```
End If
```

```
Next
```

```
%>
```

Form հավաքածուն: Ինչպես և QueryString-ը Form հավաքածուն պարունակում է պարամետրերի արժեքներ: Սակայն եթե QueryString-ում ընդգրկվում են սերվերին հաղորդվող բոլոր պարամետրերը, ապա Form հավաքածուն պարունակում է միայն ֆորմայի էլեմենտներին վերաբերվող տվյալները:

Առանձին էլեմենտի արժեքը ստանալու համար անհրաժեշտ հրամանի գրանցման քերականությունը հետևյալն է՝

Request.Form(էլեմենտի անունը)

Եթե էլեմենտը կարող է ունենալ մեկից ավելի արժեքներ (այսինքն ինքնել է հանդիսանում հավաքածու, օրինակ <select> սահմնող մեյուրն), ապա դրա արժեքների քանակը կարելի է որոշել Count հատկության միջոցով՝

```
Request.Form(էլեմենտի անունը).Count,
```

ընդ որում դրանց համարակալումը սկսվում է 1-ից:

Այն դեպքում, երբ էլեմենտը չունի բազմակի արժեքներ, դրա Count հատկությանը հավասար է 1-ի, իսկ երբ տվյալ անունով էլեմենտ չի գտնվել՝ 0-ի: Դիտարկենք օրինակ: Դիցուկ սերվերին են հաղորդվում հետևյալ ֆորմայի արժեքները՝

```
<form action = "/scripts/submit.asp" method = "post">
<p>Ձեր անունը: <input name = "firstname" size = "40"></p>
<p>Պաղպաղակի ո՞ր տեսակն էք դուք նախընտրում: <select name = "flavor">
<option>վանիլային</option>
<option>ելակի</option>
<option>շոկոլադե</option>
<option>ընկույզով</option></select></p>
<p><button type = "submit">Ուղարկել</button>
</form></p>
```

Եթե գրանցվի հետևյալ հրամանը՝ `<%= Request.Form %>`, ապա կարտատպվի չմշակված հարցման տողը, օրինակ՝ `firstname=Արմեն&flavor=ելակի:`

Իսկ եթե ձևակերպենք՝

Բարև Ձեզ, `<%= Request.Form("firstname") %>`:

Դուք սիրում եք `<%= Request.Form("flavor") %>` պաղպաղակ:

կարտատպվի հետևյալ տողը՝

Բարև Ձեզ, Արմեն: Դուք սիրում եք ելակի պաղպաղակ:

ClientCertificate հավաքածուն: Գրանցման քերականությունը հետևյալն է՝

`Request.ClientCertificate(Key[SubField])`

Սերտիֆիկատը կարող է պարունակել հետևյալ դաշտերը՝

- ◆ Certificate – երկուական տող է, որը ցուցադրում է սերտիֆիկատի բովանդակությունը ASN.1 ֆորմատում;
- ◆ Issuer – տող է, որը բաղկացած է հրատարակողի վերաբերյալ տվյալները պարունակող ենթադաշտերից: Օգտագործելով SubField բանալիները կարելի է ստանալ այդ արժեքները առանձին առանձին: Եթե գրանցենք օրինակ՝
- ◆ `<%Request.ClientCertificate("IssuerCN")["C"]%>`, ապա կստանանք հրատարակողի գտնվելու վայրի անվանումը (US, RU և այլն): Իսկ եթե պարզապես գրանցենք՝
- ◆ `<%Request.ClientCertificate("IssuerCN")%>`, ապա տեղեկությունները կձևակերպվեն աղյուսակային (comma-separated values) տեսքով;
- ◆ SerialNumber – սերտիֆիկատի սերիալ համարն է, օրինակ՝ 04-67-F3-02;

- ◆ Subject – սերտիֆիկացման սուբյեկտի վերաբերյալ տվյալներն են, որոնց ստացման եղանակը նույնն է, ինչ Issuer-ինը;
 - ◆ ValidFrom – սերտիֆիկատի ուժի մեջ մտնելու ժամկետը;
 - ◆ ValidUntil – սերտիֆիկատի ուժը կորցնելու ժամկետը:
- Թվարկերը մասնա Issue և Subject-ի հնարավոր ենթադասերը՝
 C - երկրի (տարածաշրջանի) անունը,
 CN – օգտվողի ընդհանուր անունը,
 GN – օգտվողի հատուկ անունը,
 I – ինիցիալները,
 L – հասցեն,
 O – կազմակերպության անվանումը,
 OU – կազմակերպության միավորի անունը:

§4.4. Session և Application օբյեկտների կիրառությունը

Session օբյեկտը օգտագործվում է առանձին հաճախորդի ընթացիկ սեանսի վիճակի վերաբերյալ ինֆորմացիան պահպանելու և վերադարձնելու համար: Օրինակ, պատկերացնենք, որ կայքը պարունակում է երկու հիմնական էջեր՝ առաջինում անփոփոխելի են ապրանքների վաճառքին վերաբերվող տվյալները, իսկ երկրորդը հնարավորություն է տալիս հաճախորդին կատարել գնումներ: Ընթացքում մենք պետք է համոզված լինենք, որ սպառողները դիտում են միայն առաջին էջի բովանդակությունը, միաժամանակ հաշվարկելով նրանց կատարած գործարքների քանակը:

Հաճախ անհրաժեշտություն է առաջանում օգտագործել միևնույն ինֆորմացիան web կայքի տարբեր էջերում: Օրինակ, երբ պետք է պարզել օգտվողի անունը և փոխանցել այն մի այլ էջ կամ ցուցադրել տվյալ հաճախորդի վերջին այցելության ժամկետը և այլն: Դա կարելի է իրականացնել տարբեր եղանակներով: Օրինակ՝

- Պահպանել ինֆորմացիան Session և Application օբյեկտներում, որոնց փոփոխականները հնարավոր է օգտագործել բոլոր էջերում:
- Պահպանել ինֆորմացիան Cookie ֆայլերում, կատարելով անհրաժեշտ փոփոխությունները յուրաքանչյուր նոր հաճախման ժամանակ:
- Պահպանել ինֆորմացիան տվյալների բազայում, կապակցելով այն կայքին և օգտագործել համապատասխան գործիքային ապահովումը տվյալները վերլուծելու և տեսակավորելու համար:

Երբ առաջին հաճախորդը կատարում է դիմում ASP կայքին (հատուկ գրականությունում այն, ինչպես և ցանկացած ծրագրային արտադրանք, անվանում են application) սերվերը անմիջապես ստեղծում է մեկ Application օբյեկտ, որը գոյություն է ունենում կայքի աշխատանքի ամբողջ ընթացքում և կազմալուծվում է այն պահին, երբ վերջին հաճախորդը “դուրս է գալիս” կայքից:

Session օբյեկտը ստեղծվում է յուրաքանչյուր առանձին օգտվողի համար այն պահին, երբ նա կատարում է առաջին դիմումը կայքին և վերանում է, երբ այդ օգտվողը կանոնի (անջատում է բրաուզերը) կամ ստիպողական եղանակով (կիրառվում է Session.Abandon մեթոդը) լքում է կայքը: Ի տարբերություն Application-ի, որը գլոբալ է բոլոր հաճախորդների համար, Session օբյեկտը գլոբալ է միայն առանձին հաճախորդի տեսակետից:

Փոփոխականներ ստեղծելու և դրանց արժեքներ շնորհելու եղանակը երկու օբյեկտների համար էլ միևնույնն է՝
 Application(“փոփոխականի_անուն”) = արժեք
 Session(“փոփոխականի_անուն”) = արժեք
 արժեք = Application(“փոփոխականի_անուն”)
 արժեք = Session(“փոփոխականի_անուն”)

Օրինակ, IIS սերվերում կիրառվում է Global.asa ֆայլը որը ավտոմատ կերպով գործարկվում է, երբ գործարկվում է կամ փակվում կայքը կամ, երբ յուրաքանչյուր առանձին հաճախորդ սկսում և վերջացնում է սեանսը: Երբ կայք է “մտնում” առաջին հաճախորդը գործարկվում է Global.asa ֆայլի Application_OnStart իրադարձության մշակիչը, որն ունի մոտավորապես հետևյալ տեսքը՝

```
----- Application_OnStart-----
Sub Application_OnStart()
Application(“counter”) = 0
End Sub
```

Application օբյեկտի counter փոփոխականին շնորհվում է 0 արժեքը: Յուրաքանչյուր հաջորդ հաճախորդի առաջին դիմումից հետո գործարկվում է Session_OnStart իրադարձության մշակիչը, որը մեկով ավելացնում է ընդհանուր հաճախումների քանակը և միաժամանակ շնորհում է ստացված արժեքը Session օբյեկտի նույնանուն փոփոխականին՝

```
----- Session_OnStart-----
Sub Session_OnStart()
iCount=Application(“counter”)
iCount=iCount+1
```

```
Application("counter")=iCount
Session("counter")=iCount
End Sub
```

Եթե կայքի մեկնարկային էջը պարունակում է ստորև բերված կոդը՝

```
<body>
<h1>Բարի Գալուստ</h1>
Դուք <%=Session("counter")%> - ող հաճախորդն էք
<%=Application("Counter")%> - ից
</body>
```

ապա էջը առաջին անգամ հաճախելիս օգտվողը կտեսնի, որ երկու արժեքները համընկնում են: Սակայն եթե նա վերադառնա մեկնարկային էջին մի այլ էջից, ապա դրանք կարող են լինել տարբեր, քանի որ կարող են ավելանալ նոր հաճախորդներ:

Session և Application օբյեկտները պահպանում են միայն դինամիկ (ընթացիկ) ինֆորմացիան և, եթե անհրաժեշտ է պահպանել այն հետագա օգտագործման համար, ապա պետք է ընտրել մի որևէ այլ միջոց: Օրինակ կարելի է օգտվել Global.asa ֆայլի համապատասխան Application_OnEnd և Session_OnEnd մշակիչներից կամ պահպանել ինֆորմացիան Cookie ֆայլերում:

Session օբյեկտն ունի չորս հատկություններ՝

- ◆ CodePage հատկությունը սահմանում է կոդավորման այն համակարգը ըստ որի պետք է ապակոդավորվեն կայքի էջերը (տարբեր լեզուների համար դրանք տարբեր են):
- ◆ LCID հատկության միջոցով սահմանվում է, թե ինչպիսի ֆորմատում պետք է գրանցվեն տարեթվերը, ժամանակը և այլ մեծությունները (օրինակ, տարադրամը): Դրանք նույնպես տարբեր են տարբեր աշխարհագրական գոտիների համար: Օրինակ, Բրիտանիայի համար այն հավասար է 2027 և եթե գրանցվի հետևյալ ծրագրային կոդը՝

```
<%
```

```
Session.LCID=20207
```

```
Dim curNumb
```

```
curNumb=FormatCurrency(125)
```

```
Response.Write(curNumb)
```

```
%>
```

ապա կարտապատկերվի՝ 125 (ֆունտերի նշանով):

- ◆ SessionID հատկությունը վերադարձնում է ունիկալ իդենտիֆիկատորը, որը ավտոմատ կերպով ստեղծվում է սերվերը յուրաքանչյուր “սեսիան” բացելիս:

- ◆ Timeout հատկության միջոցով սահմանվում է այն ժամանակահատվածը, որի ընթացքում օգտվողը կարող է դիտարկել որևէ էջ առանց որևէ փոփոխության կամ այլ էջի դիմելու: Հակառակ դեպքում օբյեկտը անմիջապես վերացվում է:

Session օբյեկտի հավաքածուներն են՝

- ◆ Contents հավաքածուն պարունակում է բոլոր ստեղծված փոփոխականները և այն օբյեկտները, որոնք ստեղծվել են առանց <object> տեգը օգտագործելու:
- ◆ StaticObjects հավաքածուում պահպանվում են բոլոր այն օբյեկտները, որոնք ստեղծվել են <object> տեգի միջոցով:

Հետևյալ ծրագրային հատվածում Session օբյեկտին շնորհվում է մեկ զանգված, մեկ փոփոխական և տվյալների բազայի հետ կապակցման օբյեկտ (վերջինը ստեղծվում է Server օբյեկտի միջոցով, որը կքննարկվի քիչ ուշ)

```
<%@ LANGUAGE="VBSCRIPT" %>
```

```
<%
```

```
Dim sessitem
```

```
Dim anArray(3)
```

```
response.write "SessionID: " & Session.SessionID & "<p>"
```

```
anArray(0)="one"
```

```
anArray(1)="second"
```

```
anArray(2)="third"
```

```
Session("anArray")=anArray
```

```
Session("scalar")="1234567890ABCDEFGH"
```

```
set objConn=Server.CreateObject("adodb.connection")
```

```
set Session("object")=objConn
```

```
%>
```

Session օբյեկտն ունի մակ երեք մեթոդներ, որոնցից երկուսը՝

Remove() և **RemoveAll()** կիրառում են Content հավաքածուից փոփոխականներ և/կամ օբյեկտներ հեռացնելու համար:

Remove() մեթոդի միջոցով կարելի է հեռացնել առանձին փոփոխականներ կամ օբյեկտներ, դիմելով դրանց անունով կամ ինդեքսային (հաջորդական) համարով, օրինակ՝

```
Session.Content.Remove("anArray")
```

հրամանը հեռացնում է Content հավաքածուի anArray զանգվածը:

RemoveAll() մեթոդը հեռացնում է ամբողջ Content հավաքածուն՝

```
Session.Content.RemoveAll();
```

Abandon() մեթոդի միջոցով կարելի է արհեստականորեն հեռացնել Session-ում պահպանվող բոլոր օբյեկտները, սակայն միայն այն բանից հետո, երբ կավարտվի ընթացիկ էջում

գրանցված հրամանների կատարումը: Օրինակ, եթե գրանցենք հետևյալ ծրագրային կոդը՝

```
<%  
Session.Abandon()  
Session("MyName")="Վաղարշ"  
Response.Write(Session("MyName"))  
%>
```

ապա կարտապատկերվի Վաղարշ անունը: Սակայն եթե փորձենք օգտագործել MyName փոփոխականի արժեքը մի այլ էջում, ապա այն կլինի արդեն դատարկ:

Application օբյեկտին նույնպես հատուկ են Content և StaticObjects հավաքածուները, ինչպես նաև Remove() և RemoveAll() մեթոդները, ընդ որում դրանց կիրառության եղանակները բացարձակապես նույնն են:

Բացի թվարկվածներից օբյեկտին հատուկ են նաև **Lock()** և **Unlock()** մեթոդները: Lock() մեթոդը կիրառվում է այն դեպքերում, երբ անհրաժեշտ է արգելել բոլոր հաժախորդներին բացի մեկից փոփոխել Application օբյեկտում պահպանված տվյալները: Unlock() մեթոդը կիրառվում է այն դեպքում, երբ վերանում է այդ սահմանափակման անհրաժեշտությունը: Բերենք օրինակ:

```
<%  
Application.Lock()  
Application("numVisits") = Application("numVisits") + 1  
Application("datLastVisit") = Now()  
Application.Unlock()  
%>
```

Բերված կոդում հաճախումների քանակը ավելացվում է մեկով և վերջին հաճախման դատային շնորհվում է ընթացիկ դատայի արժեքը: Որպեսզի այդ գործողությունները հասանելի լինեն միայն տվյալ էջում այդ պահին գտնվող հաճախորդին սկզբում կիրառվում է Lock մեթոդը և, միայն անհրաժեշտ փոփոխությունները կատարելուց հետո այն լուծարվում է Unlock մեթոդի օգնությամբ:

§4.5. Server օբյեկտը: Աշխատանքը տվյալների բազաների հետ

Սերվեր օբյեկտը ծառայում է web դասերի գործարկման համար անհրաժեշտ օբյեկտների կառուցման և սերվերի հատկությունների կարգավորման համար: Պատկերացնենք, օրինակ, որ որևէ իրադարձությունների մշակման գործընթացում օգտագործվում է այսպես կոչված “բիզնես օբյեկտների” (ծրագրային արտադրանքների) գրադարան և դրանցից մեկը պետք է օգտագործվի կայքի մի շարք

էջերում՝ հաճախորդի ամբողջ սեսիայի ընթացքում: Այդպիսի դեպքերում հարմար է պահպանել այն Session օբյեկտում, մասնապես ստեղծելով Server օբյեկտի CreateObject մեթոդի միջոցով: Ստորև բերված ծրագրային կոդը ցուցադրում է, թե ինչպես կարելի է իրագործել այդ նպատակը:

```
Dim bObj as SomeBusinessObject
```

```
Set bObj=Server.CreateObject(MyCompany.SomeBusinessObject.1)
```

```
...բիզնես օբյեկտի կանչման և օգտագործման հրամաններ...
```

```
Set Session("bObjInstance")=bObj
```

Server օբյեկտն ունի մեկ հատկություն և մի շարք մեթոդներ: Դրա հատկությունն է՝

ScriptTimeout – սահմանում է մաքսիմալ ժամանակը, որի ընթացքում սկրիպտը կարող է գործել մինչև սերվերը դադարեցնի դրա աշխատանքը: Հատկությունը ուժի մեջ է մտնում միայն այն դեպքում, երբ ավարտում են աշխատանքը սերվերի կոմպոնենտները: Գրանցման քերականությունը հետևյալն է՝

```
Server.ScriptTimeout = ժամանակը վարկյաններով
```

Հատկության արժեքը չպետք է գերազանցի ըստ լռելյայն սահմանվածին, որը սովորաբար 90 վարկյանի (այն կարելի է նաև փոխել, օգտագործելով սերվերի AspScriptTimeout հատկությունը): Բերենք օրինակ՝

```
<% Server.ScriptTimeout=100 %>
```

Հետևյալ օրինակում վերադարձվում է հատկության սահմանված արժեքը և պահպանվում timeOut փոփոխականում՝

```
<% timeOut = Server.ScriptTimeout %>
```

Դիտարկենք server օբյեկտի մեթոդները և դրանց կիրառության որոշ եղանակները:

4.5.1.CreateObject() մեթոդի միջոցով էջում ստեղծվում և ներդրվում են են սերվերային ծրագրերի աշխատանքի համար անհրաժեշտ՝ տարբեր կոմպոնենտներ: Մեթոդի գրանցման քերականությունը հետևյալն է՝

```
Server.CreateObject(progID),
```

որտեղ progID արգումենտի ֆորմատն ունի հետևյալ տեսքը՝

```
[Առաքողը].[Ներդրվող_կոմպոնենտը].[Վարկածը]
```

Օրինակ՝

```
Server.CreateObject(MSWC.AdRotator)
```

```
Server.CreateObject(MSWC.Tools)
```

```
Server.CreateObject(MSWC.PageCounter)
```

```
Server.CreateObject(ADODB.Recordset)
```

և այլն (MSWC – Microsoft Web Components):

CreateObject մեթոդով ստեղծված բոլոր օբյեկտների “կյանքի տևողությունը” ըստ լռելյայն սահմանափակվում է այն էջով, որում դրանք ստեղծվել են: Սերվերը ավտոմատ կերպով վերացնում է դրանք, երբ էջի աշխատանքը ավարտվում է (իհարկե գոյություն ունեն դրանց պահպանման և այլ էջերին հաղորդման եղանակներ, օրինակ, Session օբյեկտի կամ Transfer մեթոդի օգնությամբ):

Ներդրվող կոմպոնենտների տեսականին բավականաչափ մեծ է: Դրանց լրիվ ցուցակին և կիրառության եղանակներին կարելի է ծանոթանալ հատուկ գրականությունում: Մենք կքննարկենք տվյալների բազաների հետ աշխատելու համար անհրաժեշտ կոմպոնենտները, որոնք կոչվում են Տվյալների Բազաներին Հասանելիության Կոմպոնենտներ (Database Access Components) - DAC: DAC-ն ընդգրկում է ADO (ActiveX Data Objects), ODBC (Open Database Connectivity) և OLEDB (Object Linking and Embedding for Databases) տեխնոլոգիաներով ստեղծված կոմպոնենտները:

ADO-ն հանդիսանում է առավել հաճախ օգտագործվող տեխնոլոգիաներից մեկը, քանի որ այն դյուրին է օգտագործման տեսակետից: Դրան հատուկ են 7 օբյեկտներ, որոնք կարելի է ստեղծել Server օբյեկտի միջոցով: Տվյալների բազաների հետ աշխատելիս դրանցից առավել հաճախ են օգտագործվում հետևյալ երեքը՝

- ◆ **Connection** – սահմանում է բաց կապ տվյալների աղբյուրի հետ:
- ◆ **Command** - սահմանում է հատուկ հրաման որի միջոցով կատարվում է որոշակի գործողություն տվյալների հետ:
- ◆ **Recordset** – հնարավորություն է ստեղծում “երթևեկել” բազայի աղյուսակների տողերի (գրանցումների՝ record) բազմությունում (օբյեկտի որոշ հատկությունները և մեթոդները քննարկվել են §3.9-ում):

Connection օբյեկտի կիրառությունը նման է RDS-ի կիրառությանը: Օբյեկտը ստեղծելուց հետո անհրաժեշտ է արժեքներ շնորհել դրա համապատասխան հատկություններին՝ տվյալների աղբյուրի հետ բաց կապ ապահովելու համար: Սովորաբար դա իրագործվում է ConnectionString (կապակցման տող) հատկության միջոցով, որի ներքո հնարավոր է շնորհել արժեքներ նաև այլ հատկություններին: Ստորև բերված օրինակում ստեղծվում է Connection օբյեկտը և ConnectionString-ի օգնությամբ ապահովվում է կապը տվյալների բազայի հետ (VBScript լեզվով)

<%

Dim Conn, stringConn

```

Set Conn = Server.CreateObject("ADODB.Connection")
stringConn = "Provider= Microsoft.Jet.OLEDB.4.0;Data
Source=MyServer;Initial Catalog=MyBaza;User Id=sa;Password=;"
Conn.Open( stringConn)
%>

```

Բերված ծրագրային հատվածում հայտարարվում են Conn և stringConn փոփոխականները, որոնցից առաջինին որպես արժեք շնորհվում է ստեղծված օբյեկտը (երբ փոփոխականին որպես արժեք շնորհվում է օբյեկտ, պարտադիր պետք է օգտագործել Set բանախական բառը)՝

```

Set Conn = Server.CreateObject("ADODB.Connection"),
իսկ երկրորդին կապակցման տողի արժեքը, որը պարունակում է
առաքողի անունը, տվյալների բազան պարունակող թղթապանակի
անունը (MyBaza), գաղտնաբառը, սերվերի դոմեյնային անունը
(օրինակ՝ http://localhost) և այլն (լոկալ ցանցերում աշխատելիս,
սերվերի դոմեյնային անունի փոխարեն գրանցվում է տվյալների
բազայի ֆիզիկական հասցեն, օրինակ, c:\MyBaza\baza.mdb)
stringConn = "Provider= Microsoft.Jet.OLEDB.4.0;Data Source=
MyServer;Initial Catalog=MyBaza;User Id=sa;Password=;"

```

Նշենք, որ Conn և stringConn փոփոխականները ստեղծվում են, որպեսզի հաջորդող ծրագրային կոդում օգտագործելու դեպքում գրանցումը լինի հակիրճ:

Դրանից հետո հաստատվում է կապը տվյալների բազայի հետ, ինչի համար օգտագործվում է Connection օբյեկտի Open() մեթոդը, որին որպես արգումենտ փոխանցվում է կապակցման տողի արժեքը՝ Conn.Open(stringConn):

Արդյունքում հնարավորություն է ընձեռվում ազատ աշխատել տվյալների բազայի հետ: Սակայն բազայի աղյուսակների գրանցումները դիտարկելու, փոփոխելու, հեռացնելու, այլ գործողություններ կատարելու, ինչպես նաև բազայով “երթնեցնելու” համար անհրաժեշտ է ստեղծել և կարգավորել ևս մեկ գործիք, որը մեզ արդեն քաջ ծանոթ է՝ դա Recordset օբյեկտն է:

Այն նույնպես ստեղծվում է CreateObject մեթոդի կիրառությամբ՝

```

<%
Set rsCustomers = Server.CreateObject("ADODB.Recordset")
strSQLCustomers = "select CompanyName, ContactName, City
from Customers"
Set rsCustomers = Conn.Execute(strSQLCustomers)
%>

```

Այստեղ ստեղծվում է Recordset օբյեկտ տվյալների բազայի Customers աղյուսակի համար՝

```

Set rsCustomers = Server.CreateObject("ADODB.Recordset"),
որը հետագայում կարգավորվում է, այսինքն, կառուցվում է
համապատասխան SQL հարցումը՝
strSQLCustomers = "select CompanyName, ContactName, City
from Customers",
(դա նշանակում է՝ ընտրել Customers աղյուսակի CompanyName,
ContactName, City սյունակների տվյալները) և ապա Recordset
օբյեկտին շնորհվում է ստեղծված Connection օբյեկտի Execute
մեթոդի վերադարձվող արժեքը, որը իրենից ներկայացնում է
աղյուսակից ընտրված բոլոր համապատասխան գրանցումները
(որպես արգումենտ մեթոդին փոխանցվում է SQL հարցման տողը)՝
Set rsCustomers = Conn.Execute(strSQLCustomers):
Նույն նպատակին կարելի է հասնել նաև, օգտագործելով
Recordset օբյեկտի Open() մեթոդը, օրինակ՝
<% Set rsCustomers = Server.CreateObject("ADODB.Recordset")
strSQLCustomers = "select CompanyName, ContactName, City
from Customers"
rsCustomers.Open(strSQLCustomers, Conn, adOpenDynamic,
adLockPessimistic, adCmdText) %>

```

Open() մեթոդի առաջին երկու արգումենտներին որպես արժեք փոխանցվում են՝ տվյալների աղբյուրը (դա կարող է լինել SQL հարցում կամ աղյուսակի անուն) և կապակցման օբյեկտը: Հաջորդ երկուսը սահմանում են տվյալների աղբյուրի և դրա ուղեկապման տեսակը՝ համապատասխանաբար CursorType և LockType: Դրանց արժեքները կարելի փոխանցել տառային (ինչպես մեր օրինակում) կամ թվային հաստատուն մեծության տեսքով: Աղյուսակներ 4.5.1. և 4.5.2-ում բերված են այդ արգումենտների հնարավոր արժեքները և դրանց նկարագրությունը:

Ուղեկապումը օգտագործվում է, երբ որևէ օգտվող փոփոխություններ է կատարում գրանցումներում և անհրաժեշտ է սահմանափակել այլ օգտվողների հասանելիությունը փոփոխվող գրանցումներին: Եթե ուղեկապում չօգտագործվի, ապա կարող են առաջանալ կոնֆլիկտային իրավիճակներ, երբ միաժամանակ մի քանի օգտվող ցանկանան պահպանել միևնույն գրանցման մեջ կատարած փոփոխությունները:

Վերջին արգումենտի միջոցով բացահայտվում է առաջին պարամետրի տեսակը՝ հանդիսանում է դա SQL հրաման, թե՞ աղյուսակի անուն, թե՞ պահպանված պրոցեդուրա և այլն: Տվյալ դեպքում asCmdText (կամ թվային արտահայտմամբ 1) նշանակում է, որ առաջին արգումենտը SQL հարցում է:

Աղյուսակ 4.5.1.

CursorType արգումենտի հնարավոր արժեքները

Տառային	Թիվ	Նկարագրությունը
adOpenForwardOnly	0	Միակողմանի կողմնորոշում ունեցող տվյալների ընտրանք, որը հնարավոր է դիտարկել սկսած առաջին գրանցումից մինչև վերջինը (և ոչ հակառակ ուղղությամբ):
adOpenKeyset	1	Տվյալների սահմանափակ ընտրանք (միայն ընթերցման համար), որը չի արտացոլում մյուս օգտվողների կողմից կատարված փոփոխությունները:
adOpenDynamic	2	Տվյալների դինամիկ ընտրանք, որը ընդգրկում է բոլոր գրանցումները և արտացոլում այլ օգտվողների կողմից կատարված փոփոխությունները:
adOpenStatic	3	Տվյալների սահմանափակ ընտրանք (միայն ընթերցման համար), որը պարունակում է ընտրանքը կատարելու պահին առկա գրանցումները:

Աղյուսակ 4.5.2.

LockType արգումենտի հնարավոր արժեքները

Տառային	Թիվ	Նկարագրությունը
adLockReadOnly	1	Ընտրանքի գրանցումները հասանելի են միայն ընթերցման համար: Ըստ լռելյայն արժեքն է:
adLockPessimistic	2	Պեսսիմիստիկ ուղեկապում (բլոկիրովկա): Գրանցումը ուղեկապվում է անմիջապես այն պահին, երբ օգտվողը սկսում է փոփոխել գրանցման դաշտերը:
adLockOptimistic	3	Օպտիմիստիկ: Գրանցումը ուղեկապվում է միայն նոր արժեքների գրանցման պահին:

Տվյալների կապակցումը էջի հետ, ինչպես և TDC և RDS օբյեկտների օգտագործման դեպքում, կարելի է իրագործել երկու եղանակներով՝ անմիջական և աղյուսակային: Տարբերվում են միայն իրագործման մեթոդները: TDC և RDS օբյեկտների ներդրման դեպքում կապակցվող մուտքագրման դաշտի կամ աղյուսակի datasrc և datafld ատրիբուտներին որպես արժեքներ շնորհվում էին, համապատասխանաբար, կապակցող օբյեկտի իդենտիֆիկատորը և սկզբնաղբյուր աղյուսակի սյունակների անունները:

ADODB օբյեկտների կիրառության դեպքում դա արվում է այլ կերպ՝ մուտքագրման պատուհանում կամ աղյուսակի սյունակում գրանցվում է recordset օբյեկտի համապատասխան դաշտը: Օրինակ, եթե գրանցենք՝

```
<input type="text" value="<%=rsCustomers("CompanyName")%>",
```

ապա դա կնշանակի, որ մուտքագրման պատուհանում կարտացոլվեն տվյալների բազայի Customers աղյուսակի CompanyName սյունակի տվյալները: Գրանցումները դիտարկելու համար կարելի է կազմակերպել էջում նավիգացիոն կոճակներ (ինչպես դա արված է §3.9-ում) և օգտագործել recordset օբյեկտի MoveFirst, MoveNext, MoveLast և MovePrevious մեթոդները:

Հենց MoveNext մեթոդն է օգտագործվում տվյալների աղյուսակային կապակցման համար: ASP սցենարում ցիկլի օպերատորի միջոցով կազմակերպվում է աղյուսակի տողերի լրացումը հաջորդական գրանցումներով, ինչպես դա արված է ստորև բերված ծրագրային հատվածում, ընդ որում ցիկլի յուրաքանչյուր քայլի սկզբում ստուգվում է գրանցումների ավարտի պայմանը՝ rsCustomers.EOF:

---տվյալների աղյուսակային կապակցում---

```
<table>
  <!-- Վերնագրային տողի սկիզբը-->
  <tr>
    <th>Կազմակերպության անվանումը</th>
    <th>Կապի տեսակը</th>
    <th>Քաղաքը</th>
  </tr>
  <!--Customer աղյուսակի տվյալների ցուցադրման ցիկլը-->
  <% Do Until rsCustomers.EOF %>
  <tr class="tbody">
    <td> <%=rsCustomers("CompanyName")%> </td>
    <td> <%=rsCustomers("ContactName")%></td>
    <td> <%=rsCustomers("City")%> </td>
  </tr>
```

```

<%
rsCustomers.MoveNext
Loop
%>
</table>

```

Այժմ միավորենք տվյալների բազայի հետ կապակցման և էջում տվյալների արտացոլման փուլերը և կազմենք ավարտուն ASP ֆայլ, որը օգտվողին կմատուցի աղյուսակի տեսքով ձևավորված տվյալները (անվանենք այն displayTable.asp Ծր. 4.5.1.):

Ծր. 4.5.1. displayTable.asp ֆայլը

```

<%
' Հայտարարում ենք փոփոխականները
Dim Conn, stringConn
Dim rsCustomers, strSQLCustomers
'բացում ենք կապը բազայի հետ
Set Conn = Server.CreateObject("ADODB.Connection")
stringConn = "Provider= Microsoft.Jet.OLEDB.4.0;Data
Source=http://localhost;Initial Catalog=MyBaza;User
Id=;Password=;"
Conn.Open( stringConn)
'Ստեղծում և բացում ենք recordset օբյեկտը
Set rsCustomers = Server.CreateObject("ADODB.Recordset")
strSQLCustomers = "select CompanyName, ContactName, City
from Customers"
rsCustomers.Open(strSQLCustomers, Conn, adOpenDynamic,
adLockPessimistic, adCmdText)
%>
<table>
<!-- Վերնագրային տողի սկիզբը-->
<tr>
<th>Կազմակերպության անվանումը</th>
<th>Կապի տեսակը</th>
<th>Քաղաքը</th>
</tr>
<!--Customer աղյուսակի տվյալների ցուցադրման ցիկլը-->
<% Do Until rsCustomers.EOF %>
<tr CLASS=tbody>
<td> <%=rsCustomers("CompanyName")%> </td>
<td> <%=rsCustomers("ContactName")%></td>
<td> <%=rsCustomers("City")%> </td>

```

```

</tr>
<%
rsCustomers.MoveNext
Loop
%>
</table>
<%
'Փակում ենք recordset և կապի օբյեկտները
rsCustomers.Close()
Conn.Close()
'Փոփոխականներին շնորհում ենք դատարկ արժեքներ
Set rsCustomers = Nothing
Set Conn = Nothing
%>

```

Տվյալների բազայի հետ բաց կապակցման անհրաժեշտությունը առաջանում է բազմապրոֆիլային էջերի կառուցման դեպքերում, այսինքն երբ տվյալ էջը նշանակված է և տվյալների դիտարկման, և որոշակի փոփոխությունների կատարման համար: Ընդ որում այդպիսի փոփոխությունները իրենց հերթին պահանջում են նաև պնդման, հետդարձի և այլ տրանզակցիաների հետ:

Այն դեպքերում, երբ էջը ընդգրկում է միայն մեկ ֆորմա, որի միջոցով օգտվողը կատարում է մեկ գործողություն և չի պահանջվում տվյալների լրացուցիչ դիտարկումը (մանավանդ, երբ էջում կազմակերպվում է նաև ֆորմայի բովանդակության նախնական մշակումը, տես՝ §3.8), սովորաբար օգտագործվում է Command օբյեկտը:

Command օբյեկտը նույնպես ստեղծվում է CreateObject մեթոդի միջոցով, օրինակ (այստեղ օգտագործված է JavaScript լեզուն)՝
var editCmd = Server.CreateObject("ADODB.Command")

Բազայի հետ կապը հաստատելու համար ստեղծված օբյեկտի ActiveConnection հատկությանը շնորհվում է համապատասխան արժեքը՝

```

editConnection = "Provider= Microsoft.Jet.OLEDB.4.0;Data Source=
http://localhost;Initial Catalog=MyBaza;
UserId=;Password=";
editCmd.ActiveConnection = editConnection;

```

Կապը հաստատելուց հետո համապատասխան SQL հրամանը շնորհվում է օբյեկտի CommandText հատկությանը՝
editQuery = "insert into Customers (CompanyName, ContactName, City) values ('արժեք1','արժեք2','արժեք3')";
editCmd.CommandText = editQuery;

Վերը բերված SQL հրամանը նշանակում է, որ Customers աղյուսակի CompanyName, ContactName և City սյունակներում պետք է մուտքագրվեն, համապատասխանաբար, 'արժեք1', 'արժեք2' և 'արժեք3' մեծությունները (որոնք ստացվում են հարցում կատարող էջի ֆորմայից):

Դրանից հետո Execute() մեթոդի միջոցով հրամանը կատարվում է և Command օբյեկտի կապը բազայի հետ անմիջապես ընդհատվում է (Close() մեթոդով)՝

```
editCmd.Execute();
```

```
editCmd.ActiveConnection.Close();
```

Ստորև բերված է JavaScript ASP սցենարի օրինակ (insertCust.asp ֆայլը), որի միջոցով ֆորմայում մուտքագրվող տվյալները գրանցվում են բազայի Customers աղյուսակում: Բազայի հետ կապակցումը ապահովելու համար ստեղծված է առանձին asp ֆայլ (Baza.asp), որը ներդրված է էջում #include հրահանգի միջոցով: Ուշադրություն դարձրեք, թե ինչպես են մշակվում ֆորմայից ստացված տվյալները և ինչպես է կառուցվում SQL հրամանը:

-----Baza.asp ֆայլը-----

```
<%
```

```
// FileName="Connection_odbc_conn_dsn.htm"
```

```
// Type="ADO"
```

```
// DesigntimeType="ADO"
```

```
// HTTP="false"
```

```
// Catalog=""
```

```
// Schema=""
```

```
var MyBaza_STRING = "Provider= Microsoft.Jet.OLEDB.4.0;Data
```

```
Source=http://localhost;Initial Catalog=MyBaza;
```

```
UserId=;Password=;";
```

```
%>
```

-----insertCust.asp ֆայլը-----

```
<%@LANGUAGE="JAVASCRIPT" CODEPAGE="1252"%>
```

```
<!--#include file="../Connections/Baza.asp" -->
```

```
<%
```

```
var editAction = Request.ServerVariables("SCRIPT_NAME");
```

```
if (Request.QueryString) {
```

```
    editAction += "?" + Server.HtmlEncode(Request.QueryString);
```

```
}
```

```
var abortEdit = false;
```

```
var editQuery = "";
```

```
if (String(Request("insert")) == "form_insert")
```



```

{
    var editConnection = MyBaza_STRING;
    var editTable = "Customers";
    var editRedirectUrl = "insertCust.asp";
    var          fieldsStr          =
"CompanyName|value|ContactName|value|City|value";
    var          columnsStr        =
"CompanyName|none,none,NULL|ContactName|none,none,NULL|Ci
ty|none,none,NULL";
    // create the fields and columns arrays
    var fields = fieldsStr.split("|");
    var columns = columnsStr.split("|");
    // set the form values
    for (var i=0; i+1 < fields.length; i+=2) {
        fields[i+1] = String(Request.Form(MM_fields[i]));
    }
    // append the query string to the redirect URL
    if (editRedirectUrl && Request.QueryString &&
Request.QueryString.Count > 0) {
        editRedirectUrl += ((editRedirectUrl.indexOf('?') == -1)?"?":"&") +
Request.QueryString;
    }
}
// append the query string to the redirect URL
if (editRedirectUrl && Request.QueryString &&
Request.QueryString.Count > 0) {
    editRedirectUrl += ((editRedirectUrl.indexOf('?') == -1)?"?":"&") +
Request.QueryString;
}
}
// Insert Record: construct a sql insert statement and execute it
if (String(Request("MM_insert")) != "undefined") {
    // create the sql insert statement
    var tableValues = "", dbValues = "";
    for (var i=0; i+1 < fields.length; i+=2) {
        var formVal = fields[i+1];
        var typesArray = columns[i+1].split(",");
        var delim = (typesArray[0] != "none") ? typesArray[0] : "";
        var altVal = (typesArray[1] != "none") ? typesArray[1] : "";
        var emptyVal = (typesArray[2] != "none") ? typesArray[2] : "";
        if (formVal == "" || formVal == "undefined") {

```

```

        formVal = emptyVal;
    } else {
        if (altVal != "") {
            formVal = altVal;
        } else if (delim == "") { // escape quotes
            formVal = "" + formVal.replace(/'/g, "'") + "";
        } else {
            formVal = delim + formVal + delim;
        }
    }
    tableValues += ((i != 0) ? "," : "") + columns[i];
    dbValues += ((i != 0) ? "," : "") + formVal;
}
editQuery = "insert into " + editTable + " (" + tableValues + ") values
(" + dbValues + ")";

if (!abortEdit) {
    // execute the insert
    var editCmd = Server.CreateObject('ADODB.Command');
    editCmd.ActiveConnection = editConnection;
    editCmd.CommandText = editQuery;
    editCmd.Execute();
    editCmd.ActiveConnection.Close();

    if (editRedirectUrl) {
        Response.Redirect(editRedirectUrl);
    }
}
%>
<html>
<head>
<title></title>
</head>
<body>
<form          action="<%=editAction%>"          method="post"
name="form_insert">
<input type="text" name="CompanyName" />
<input type="text" name="ContactName" />
<input type="text" name="City" />
<input type="hidden" name="insert" value="form_insert">

```

```
</form>
</body>
</html>
```

4.5.2.Execute() մեթոդի միջոցով կազմակերպվում է “.asp” ընդլայնումով ֆայլերի կանչը և կատարումը: Գրանցման քերականությունն է՝

Server.Execute(path),

որտեղ path-ը կատարվող .asp ֆայլի հասցեն է, ընդ որում այն չպետք է պարունակի հարցման տող, այսինքն ? նշանից հետո գրանցված մաս:

Մեթոդի միջոցով կարելի է հասանելիություն ապահովել .asp սկրիպտերի մի ամբողջ գրադարանին և դինամիկ կերպով ապահովել այդ սկրիպտերի կանչը ու կատարումը:

Մեթոդի կիրառության դեպքում կանչող էջի բոլոր ներկառուցված օբյեկտների հատկությունները, հավաքածուները, մեթոդները և փոփոխականները հասանելի են դառնում կանչվող էջի համար: Նույնպես հասանելի են բոլոր ստեղծված փոփոխականները և օբյեկտները: Սակայն կանչվող ֆայլին անհասանելի են #include դիրեկտիվի միջոցով ներդրված ռեսուրսները:

Ստորև բերված օրինակում, կախված սահմանված լեզվից welcome.asp ֆայլում կազմակերպվում է համապատասխան asp սկրիպտի կատարումը, որի միջոցով կանչող էջում արտապատկերվում է համապատասխան լեզվով գրված ողջույնը:

---welcome.asp ֆայլը---

```
<html>
```

```
<body>
```

```
<h1> Company Name</h1>
```

```
<%
```

```
Lang = Request.ServerVariables("HTTP_ACCEPT_LANGUAGE")
```

```
Lang = Left(AcceptLang, 2)
```

```
Server.Execute(Lang & "Welcome.asp")
```

```
%>
```

```
</body>
```

```
</html>
```

```
<--enWelcome.asp ֆայլը (անգլերեն)→
```

```
<% Response.Write "Welcome to my website!" %>
```

```
<--deWelcome.asp ֆայլը (գերմաներեն)→
```

```
<% Response.Write "Willkommen zu meinem website!" %>
```

```
<--esWelcome.asp ֆայլը (իսպաներեն)→
```

```
<% Response.Write "Recepcign a mi website!" %>
```

HTTP_ACCEPT_LANGUAGE միջավայրի փոփոխականի արժեքից կախված, արտապատկերման արդյունքում կստացվեն հետևյալ ողջույնները՝

----անգլերեն---

Company Name

Welcome to my website!

----գերմաներեն---

Company Name

Willkommen zu meinem website!

----իսպաներեն---

Company Name

Recepcign a mi website!

4.5.3.GetLastError() մեթոդը վերադարձնում է կատարված սխալի տեսակի նկարագրությունը, օրինակ՝ պրեպրոցեսորի հրամանների և փոփոխականների սահմանման, կոմպիլացման (կատարվող ծրագրի հրամանների կոդերի ձևակերպման) և կատարման (այսինքն անթույլատրելի գործողությունների օգտագործման, օրինակ գրոյի բաժանման): Մեթոդի գրանցման քերականությունը հետևյալն է՝

Server.GetLastError ()

4.5.4.HTMLEncode() մեթոդի միջոցով տրված տողում պարունակող HTML կոդը ներկայացվում է պրիմիտիվների տեսքով: Օրինակ հետևյալ սկրիպտի

```
<%= Server.HTMLEncode("Պարագրաֆի տեգը - <p>")%>
```

կատարման արդյունքում կստացվի՝

Պարագրաֆի տեգը - <p>

4.5.5.MapPath() մեթոդի միջոցով կանչվող ֆայլերի վիրտուալ կամ հարաբերական հասցեները վերածվում են սերվերային դիրեկտորիաների լրիվ ֆիզիկական հասցեների: Օրինակ, ընթացիկ ֆայլի ֆիզիկական հասցեն ստանալու համար կարելի է օգտագործել հետևյալ սկրիպտը՝

```
<%
```

```
= Server.MapPath(Request.ServerVariables("PATH_INFO"))
```

```
%><br />
```

Սկրիպտի կատարման արդյունքը կարող է լինել այսպիսին՝

c:\inetpub\wwwroot\script\test.asp

Քանի որ հաջորդ երկու օրինակներում հասցեները տրված են հարաբերական տեսքով և չեն սկսվում շեղ գծիկով, ապա դրանց ավելացվում է ընթացիկ թղթապանակի հասցեն (օրինակներում դա c:\inetpub\wwwroot\script-ն է)՝

```
<%= Server.MapPath("data.txt")%><br />
```

```
<%= Server.MapPath("script/data.txt")%>
```

Արդյունքում կստացվի՝

```
c:\inetpub\wwwroot\script\data.txt
```

```
c:\inetpub\wwwroot\script\script\data.txt
```

Մեկնարկային էջի ֆիզիկական հասցեն (այսպես կոչված home directory-ն) կարելի է ստանալ հետևյալ սկրիպտի միջոցով՝

```
<%= Server.MapPath("/")%>
```

Արդյունքում կստացվի՝

```
c:\inetpub\wwwroot
```

4.5.6.Transfer() մեթոդի միջոցով որևէ .asp ֆայլի կատարման ընթացքում կուտակված ամբողջ ինֆորմացիան հաղորդվում է (այսինքն հասանելի է դառնում) մեկ այլ .asp ֆայլին: Հրամանը գրանցվում է հետևյալ տեսքով՝

Server.Transfer (path),

որտեղ path-ը այն ֆայլի հասցեն է, որին հաղորդվում է կուտակված ինֆորմացիան, ընդ որում դա նույնիսկ կարող է լինել մի այլ կայքի (այլ application-ի) ֆայլ: Բերենք փոկրիկ օրինակ: Դիցուկ ցանկանում ենք փոխանցել asp1.asp ֆայլում կուտակված ինֆորմացիան asp2.asp ֆայլին: Կազմենք երկու ֆայլեր՝

```
---asp1---
```

```
<html>
```

```
<body>
```

```
<%
```

```
Dim sessvar1
```

```
Response.Write(Session.SessionID)
```

```
Response.Write("<br />")
```

```
Response.Write("I am going to asp2 <br />")
```

```
Server.Transfer("/Myasps/asp2.asp")
```

```
%>
```

```
</body>
```

```
</html>
```

```
---asp2---
```

```
<html>
```

```
<body>
```

```
<% Response.Write(Session.SessionID) %>
```

```
</body>
```

```
</html>
```

Աշխատանքի	արդյունքում	կստացվեն	հետևյալ
արտապատկերումները՝			

```
---asp1---
```

սեսիայի ID-ն

I am going to asp2

---asp2---

սեսիայի ID-ն:

4.5.7.URLEncode() մեթոդը կոդավորում է որպես արգումենտ գրանցված հասցեն URL կանոնների համաձայն: Գրանցման քերականությունը հետևյալն է՝

Server.URLEncode(տող)

Օրինակ, հետևյալ asp հրամանի

```
<%Response.Write(Server.URLEncode("http://www.microsoft.com"))%>
```

կատարման արդյունքում կարտապատկերվի

http%3A%2F%2Fwww%2Emicrosoft%2Ecom

տողը:

§4.6. XML լեզվի կիրառության բնագավառները

Առաջին հայացքից XML լեզուն նմանվում է HTML-ն: Երկուսի հիմքում էլ դրված է SGML (Standard Generalized Markup Language) լեզուն: Երկուսն էլ ունեն ստեղծման միանման գործիքային բազմություն: Սակայն լեզուները տարբերվում են երկու հիմնական ուղղություններով՝ քերականության և սեմանտիկայի:

Քերականական տարբերությունների և դրանք հաղթահարելու հնարավորության (XHTML լեզվի օգտագործման, ատրիբուտների արժեքների չափերտների մեջ գրանցման, փակող տեգի ‘/’ նշանի պարտադիր օգտագործման և այլն) մասին մենք արդեն խոսել ենք նախորդ շարադրության ընթացքում:

HTML-ը հիմնված է մեկ “բառարանի” օգտագործման վրա: Օրինակ գրանցումը միշտ էլ միևնույն ձևով է հասկացվում HTML պրոցեսորների կողմից: Ի տարբերություն HTML-ի XML-ը թույլ է տալիս ստեղծել գծանշման անհատական բառարան կամ ընտրել ցանցում առաջարկվող բառարանների տարբերակներից իրագործվող նպատակներին առավել համապատասխանողը: Սխեմաների և փաստաթղթերի տեսակների սահմանումների (DTD) օգտագործումը թույլ է տալիս կազմել պահանջվող տեսակի բառարաններ, սակայն փաստաթղթերը կարելի է ստեղծել նաև ֆորմալ սահմանումներ չպարունակող բառարանների օգնությամբ: Այսպես կոչված “անվանադաշտերը” (namespaces) հնարավորություն են ընձեռնում իդենտիֆիկացնել օգտագործվող բառարանը:

HTML-ի և XML-ի տարբերության վերաբերյալ սկզբնական պատկերացում ստանալու համար համեմատենք երկու լեզուներով կազմած միևնույն փաստաթուղթը:

---- HTML փաստաթուղթ-----

```
<head>
<title>printer
</head>
<body>
<h1>Արտադրողի_անվան. wizbang 3000 dot matrix printer</h1>
<ul>features:
<li>40 pages per minute
<li>60 dpi printing
</ul>
<p>$200.00
<p>10 lbs.
</body>
```

-----XML փաստաթուղթ-----

```
<?xml version="1.0"?>
<manufacturer>komputersource
<product>
  <class>printer
    <type>dot matrix</type>
  </class>
  <name>wizbang3000</name>
  <features>
    <speed units="ppm">40</speed>
    <quality units="dpi">60</quality>
  </features>
  <price units="usd">
<retail>200</retail>
    <wholesale>110</wholesale>
  </price>
  <weight units="lbs">10</weight>
</product>
</manufacturer>
```

Համեմատելով երկու փաստաթղթերը կարելի է տեսնել, որ XML փաստաթղթում սպառիչ կերպով բացահայտվում է բովանդակությունը և էլեմենտների հիերարխիան (չնայած չի բացահայտվում, թե ինչպե՞ս պետք է այն ֆորմատավորվի): Դա հնարավոր է դառնում միևնույն դեսկրիպտորներ (անուններ) ունեցող բացվող և փակվող տեգերի պարտադիր առկայության և դրանց ճշգրիտ ներդրվածության հաշվին: Օրինակ, պարզ է, որ product (արտադրատեսակ) տեգը հանդիսանում է manufacturer տեգի “զավակը”, այսինքն տվյալ դեպքում “ծնողական” տեգում հնարավոր է

տեղադրել մի քանի “զավակներ”։ Յուրաքանչյուր արտադրատեսակ իր հերթին ունի բազմաթիվ “զավակներ”, օրինակ դաս (class), անուն (name), բնութագրեր (features) և այլն։

Իհարկե, վաճառողի տեսանկյունից յուրաքանչյուր արտադրատեսակ կարող է ունենալ բազմաթիվ արտադրողներ և, այդ դեպքում փաստաթղթի էլեմենտների հիերարխիան պետք է կառուցվի այլ կերպ։ Ցանակացած դեպքում XML-ի կառուցվածքային սխեման հասկանալի է և մարդու և համակարգչի համար։ Ընդ որում տեգերը և կառուցվածքը ավտոմատացման ավելի հարուստ հնարավորություններ են ընձեռում։ Օրինակ՝ շատ հեշտ է կատարել տվյալների տեսակավորումը ըստ տարբեր հայտանիշների, փոփոխել արտադրատեսակի տարբեր բնութագրերը և այլն։

Այդպիսի կառուցվածքը թույլ է տալիս օգտագործել XML ֆայլերը որպես տվյալների փոկրիկ բազաներ, այսպես կոչված, “տվյալների կղզյակներ” (data islands), ընդ որում ոչ միայն պարզագույն տվյալներ պահպանելու, այլ և ծրագրավորման տարբեր լեզուներով գրված ծրագրային հատվածներ և ամբողջական ծրագրեր։

Բերենք փոքրիկ օրինակ։ Դիցուկ մենք ցանկանում ենք պահպանել տվյալներ ձեռնարկության աշխատակիցների վերաբերյալ՝ անունը, ազգանունը, հայրանունը, հասցեն, զբաղեցրած պաշտոնը և աշխատավարձի չափը։ Որպես արմատային հասկացություն ընդունենք աշխատողների ցուցակը (employee list)։ Առանձին աշխատողի վերաբերյալ գրանցումները անվանենք (employee record) և այդպես հիերարխիայով դեպի վար։ Ցուցակը կարելի է ներկայացնել հետևյալ XML փաստաթղթի տեսքով՝

```
<?xml version="1.0"?>
```

```
<employeeelist>
```

```
<employeeRecord>
```

```
<name>Վաղարշյան Վաղարշ</name>
```

```
<homeAddress>ք. Երևան, փ. Վաղարշյան, շ.1,  
բն. 1</homeAddress>
```

```
<jobTitle>Բաժնի պետ</jobTitle>
```

```
<salary>$175,000</salary>
```

```
</employeeRecord>
```

```
...Գրանցումներ (<employeeRecord>...</employeeRecord>) մյուս  
աշխատողների մասին...
```

```
</employeeelist>
```

Հիերարխիայի մակարդակների քանակը կախված է նրանից, թե ինչ նպատակների համար պետք է ծառայի տվյալ փաստաթուղթը։ Եթե անհրաժեշտ է լինելու կատարել տեսակավորումներ, ֆիլ-

տրում ըստ տարբեր հայտանիշների և այլն, ապա պետք է ավելացնել կամ ստորաբաժանման խորությունը, կամ հիերարխիայի մակարդակները: Օրինակ, եթե աշխատակիցներին պետք է տեսակավորել ըստ բնակավայրի, ապա կարելի է ավելացնել <city> տեղը և գրանցել այնտեղ քաղաքի անունը: Ըստ ազգանունների տեսակավորելու համար պետք է առանձնացնել անունը և ազգանունը առանձին տեղերում: Եթե անհրաժեշտ է աշխատավարձը արտահայտել դրամով կամ այլ արժույթով, դրա չափման միավորը կարելի է ներկայացնել որպես <salary> տեղի ատրիբուտ՝ <salary currency="usd">: Վերափոխված փաստաթուղթը կարող է ընդունել, օրինակ, հետևյալ տեսքը՝

```
<?xml version="1.0"?>
```

```
<employeeelist>
```

```
<employeeRecord>
```

```
<firstname>Վաղարշ</firstname>
```

```
<lastname>Վաղարշյան</lastname>
```

```
<city>Երևան</city>
```

```
<homeAddress>փ. Վաղարշյան, շ.1, բն. 1</homeAddress>
```

```
<jobTitle>Բաժնի պետ</jobTitle>
```

```
<salary currency="usd">175,000</salary>
```

```
</employeeRecord>
```

```
...Գրանցումներ մյուս աշխատողների մասին...
```

```
</employeeelist>
```

XML փաստաթղթերի տվյալների օգտագործման եղանակները նույնն են, ինչ և այլ փաստաթղթերինը՝ ֆայլերը կարելի է գրանցել անմիջապես HTML փաստաթղթի մեջ, ներդնել <xml src="հասցե"> տեղի, #include դիրեկտիվի կամ <object> տեղի միջոցով և, վերջապես, կապակցել <link> տեղի օգնությամբ: Իսկ տվյալների կապակցման եղանակները նույնն են, ինչ և TDC կամ RDS էլեմենտների համար՝ անմիջական և աղյուսակային: Պարզաբանելու համար բերենք երկու օրինակներ: Առաջին օրինակում վերը կազմած աշխատակիցների ցուցակը տեղադրված է անմիջապես HTML փաստաթղթում (Ծր. 4.6.1.): Երկրորդ օրինակում XML փաստաթուղթը ձևակերպված է որպես առանձին ֆայլ և ներդրված է HTML փաստաթղթում <xml> տեղի միջոցով (Ծր. 4.6.2.):

Ծր. 4.6.1. XML փաստաթղթի գրանցումը անմիջապես HTML ֆայլում

```
<html><head><title></title></head>
```

```
<body>
```

```

<xml id="empList">
  <?xml version="1.0"?>
  <employeeelist>
    <employeeRecord>
      <firstname>Վաղարշ</firstname>
      <lastname>Վաղարշյան</lastname>
      <city>Երևան</city>
      <homeAddress>փ. Վաղարշյան, շ.1, բն. 1</homeAddress>
      <jobTitle>Բաժնի պետ</jobTitle>
      <salary currency="usd">175,000</salary>
    </employeeRecord>
    ...գրանցումներ նյութ աշխատողների մասին...
  </employeeelist>
</xml>
<table datasrc="#empList">
  <tr>
    <td><div datafld="firstname"></div></td>
    <td><div datafld="lastname"></div></td>
    <td><div datafld="city"></div></td>
    <td><div datafld="homeAddress"></div></td>
    ...
  </tr>
</table>
</body>
</html>

```

Ծր. 4.6.2. XML ֆայլի ներդրումը HTML ֆայլում

```

-----emploee.xml ֆայլը-----
<xml id="empList">
  <?xml version="1.0"?>
  <employeeelist>
    <employeeRecord>
      <firstname>Վաղարշ</firstname>
      <lastname>Վաղարշյան</lastname>
      <city>Երևան</city>
      <homeAddress>փ. Վաղարշյան, շ.1, բն. 1</homeAddress>
      <jobTitle>Բաժնի պետ</jobTitle>
      <salary currency="usd">175,000</salary>
    </employeeRecord>
    ...գրանցումներ նյութ աշխատողների մասին...
  </employeeelist>

```

```

</xml>
-----employee.html ֆայլը----
<html><head><title></title>
<xml id="empList" src="employee.xml"></xml>
</head>
<body>
<table datasrc="#empList">
<tr>
<td><div datafld="firstname"></div></td>
<td><div datafld="lastname"></div></td>
<td><div datafld="city"></div></td>
<td><div datafld="homeAddress"></div></td>
...մյուս սյունակները...
</tr>
</table>
</body>
</html>

```

Script բաղադրիչներ պարունակող XML ֆայլերը (script component file) նման են HTML ֆայլերին, սակայն պարունակում են հատուկ էլեմենտներ, որոնց միջոցով սահմանվում են այդ բաղադրիչները և դրանց “վարվելակերպը” (behavior): Script բաղադրիչներ պարունակող ֆայլերում օգտագործվում են հետևյալ հիմնական էլեմենտները՝

<component> և **<package>** - **<component>** էլեմենտը ընդգրկում է միայն մեկ script-ի սահմանում: Այն դեպքում երբ բաղադրիչների քանակը մեկից ավելին է, դրանք կարող են պահպանվել “.wsc” ընդլայնումով ֆայլում, սակայն պետք է ամփոփվեն մեկ **<package>** էլեմենտում: Ընդ որում մեկ **<package>**-ում կարող են ընդգրկվել ծրագրավորման տարբեր լեզուներով գրված բաղադրիչներ:

<registration> - պարունակում է script-ի գրանցման համար անհրաժեշտ ինֆորմացիան: Սակայն այն դեռ չի հանդիսանում պարտադիր էլեմենտ, քանի որ դեռ ոչ բոլոր բրաուզերներն են ճիշտ մեկնաբանում Windows տեսակի ռեգիստրացիան:

<public> - պարունակում է այն հատկությունների, մեթոդների և իրադարձությունների սահմանումները, որոնք նկարագրվում են script բաղադրիչում: Համապատասխան մշակող ֆունկցիաները և փոփոխականները նկարագրվում են դրան հարակից **<script>** էլեմենտում:

<script> - պարունակում է այն ծրագիրը, որի միջոցով էլ իրագործվում է script բաղադրիչի ստեղծման նպատակը:

<resource> - պարունակում է այն տվյալները, որոնք օգտագործվում են սկրիպտում, սակայն չեն կազմում ծրագրային կոդի մասը (ներդրվում է ծրագրում <object> տեգի միջոցով)

<reference> - պարունակում է հղումների այն ծրագրերին, որոնք անհրաժեշտ է օգտագործել տվյալ script-ում:

Ստորև բերված է այդպիսի ֆայլի կառուցվածքային կմախքը (Ծր. 4.6.3.):

Ծր. 4.6.3. Script component fail-ի գրանցման քերականությունը

```
<?XML version="1.0"?>
```

```
<package>
```

```
<?component error="true" debug="true"?>
```

```
<comment>
```

This skeleton shows how script component elements are assembled into a .wsc file.

```
</comment>
```

```
<component id="MyScriptlet">
```

```
<registration
```

```
progid="progID"
```

```
description="description"
```

```
version="version"
```

```
clsid="{00000000-0000-0000-000000000000}"/>
```

```
<reference object="progID">
```

```
<public>
```

```
<property name="propertyname"/>
```

```
<method name="methodname"/>
```

```
<event name="eventname"/>
```

```
</public>
```

```
<implements type="COMhandlerName" id="internalName">
```

(interface-specific definitions here)

```
</implements>
```

```
<script language="VBScript">
```

```
<![CDATA[
```

```
dim propertyname
```

```
Function methodName()
```

```
' Script here.
```

```
End Function
```

```
]]>
```

```
</script>
```

```
<script language="JScript">
```

```
<![CDATA[
```

```

function get_propertyname()
{ // Script here.
}
function put_propertyname(newValue)
{ // Script here.
  fireEvent(eventname)
}
}]>
</script>
<object id="objID" classid="clsid:00000000-0000-0000-
000000000000">
  <resource ID="resourceID1">string or number here</resource>
  <resource ID="resourceID2">string or number here</resource>
</component>
</package>

```

Կարելի է նկատել, որ `<script>` էլեմենտներում ծրագրային կոդը ամփոփված է **![CDATA[]]** հատվածում: Դա արված է հետևյալ պատճառով: Քանի որ XML-ում արգելված է “<”, “>”, “&” և որոշ այլ սիմվոլների օգտագործումը, դրանք պետք է գրանցվեն պրիմիտիվների տեսքով, օրինակ՝ “<”-ն որպես “<”, “>”-ն որպես “>” և այլն: Սակայն ծրագրերում այդ նշանները հնարավոր չէ օգտագործել պրիմիտիվների տեսքով և հակասություններից խուսափելու նպատակով ծրագրային մասը ամփոփվում է **![CDATA[]]** բլոկում:

Որպեսզի հասկանանք, թե ինչպես են օգտագործվում ծրագրային բաղադրիչները (կոմպոնենտները) բերենք մի պարզ օրինակ: Կառուցենք ֆակտորիալի հաշվարկի ծրագրային բաղադրիչ (ծր. 4.6.4.):

Ծր. 4.6.4. Ֆակտորիալի հաշվարկի ծրագրային բաղադրիչը

```

<?XML version="1.0"?>
<component id="MyComponent">
<public>
  <method name="factorial"/>
  <method name="random" internalName="getRandomNumber">
    <parameter name="upperBound"/>
    <parameter name="seed"/>
  </method>
</public>
<script language="VBScript">
Function factorial(n)

```

```

<![CDATA[
If isNumeric(n) Then
  If n <= 1 Then
    factorial = 1
  Else
    factorial = n*factorial(n-1)
  End If
Else
  factorial = -2 ' Սխալի կոդը.
End If
End Function
Function getRandomNumber(upperBound, seed)
  getRandomNumber = Cint(upperBound * Rnd(seed) + 1)
End Function
]]>
</script>
</component>

```

Ինչպես տեսնում ենք <public> էլեմենտում հայտարարված են երկու մեթոդներ (ֆունկցիաներ)՝ factorial և getRandomNumber, որոնց ծրագրային կոդը տեղադրված է <script> սեկցիայում: Երկու էլեմենտներն են ընդգրկված են MyComponent իդենտիֆիկատորով բաղադրիչում (<component> էլեմենտում): Եթե դուք ծանոթ եք ծրագրավորման C++ օբյեկտային լեզվին, ապա կարող եք նկատել, որ ծրագրային բաղադրիչի ստեղծումը նման է դասի ստեղծմանը՝ հայտարարվում են և, ապա, նկարագրվում դասի (տվյալ դեպքում ծրագրային բաղադրիչի) մեթոդները: Ֆունկցիայի արգումենտների անունները հայտարարվում են <parametr> տեգում:

Նույն սկզբնումքով են հայտարարվում նաև բաղադրիչի հատկությունները (այսինքն փոփոխականները) և իրադարձությունները: Օրինակ, ստորև բերված ծրագրային հատվածում <public> տեգում հայտարարվում են name և tagVar հատկությունները (փոփոխականները), իսկ <script> տեգում դրանց շնորհիվ են սկզբնական արժեքները (արժեքների շնորհիվ կարող է կատարվել նաև ֆունկցիաներում, կամ որոշակի հաշվարկների արդյունքում)՝

```

<public>
  <property name="name"/>
  <property name="tag" internalName="tagVar"/>
</public>
<script language="VBScript">
  <![CDATA[

```

```

Dim name
name = "script component" ' name հատկությանը շնորհվում է
'սկզբնական արժեք
Dim tagVar
tagVar = "10" ' Initializes the value of tag property.
]]>
</script>

```

Ծրագրային բաղադրիչի մեթոդների և հատկությունների օգտագործումը նույնպես նման է դասի մեթոդների և հատկությունների օգտագործմանը՝ ստեղծվում է բաղադրիչի օբյեկտ նմուշ և դրա միջոցով իրականացվում է դիմումը մեթոդներին և հատկություններին: Օրինակ, որպեսզի օգտագործվի Ծր. 4.6.4-ում սահմանած մեթոդները սցենարում կարող ենք ստեղծել MyComponent-ի նմուշ և դրա միջոցով դիմել մեթոդներին՝

```

Set component = CreateObject("component.MyComponent")
n = component.factorial(4)
rand=component.getRandomNumber(100,3)

```

Մեթոդներից որևէ մեկը (միայն մեկը) կարելի է հայտարարել նաև, որպես ըստ լռելյան ընդունվող ընտրված մեթոդի dispid (dispatch identifier) ատրիբուտին շնորհվում է "0" արժեքը: Օրինակ, եթե գրանցենք՝

```

<public>
<method name="factorial" dispid="0"/>
<method name="random" internalName="getRandomNumber">
<parameter name="upperBound"/>
<parameter name="seed"/>
</public>,

```

ապա ծրագրում մեթոդին կարելի է դիմել հետևյալ կերպով՝
Set component = CreateObject("component.MyComponent")
n = component(4):

Ավելի մանրամասն XML- կիրառության բնագավառներին (դա հանդիսանում է առանձին ձեռնարկի նյութ) կարելի է ծանոթանալ, օգտվելով հատուկ գրականությունից կամ Microsoft ֆիլմայի MSDN գրադարանից:

§4.7. PHP լեզվի կիրառության բնագավառները

PHP-ն ("PHP: Hipertext Preprocessor" բառակապակցության ռեկուրսիվ հապավումը) դա՝ բաց սկզբնական կոդով, ընդհանուր նշանակման ծրագրավորման լեզու է: Այն կառուցվել է հատրապես սերվերային web-մշակումներ ստեղծելու նպատակով և կարող է

ներդրվել HTML ծրագրային կոդերում: PHP-ին օգտագործելու համար սերվերում տեղադրվում է հատուկ Apache սերվերային մոդուլը (այսպես կոչված, parser-ը՝ լեզվական վերլուծության հատուկ ծրագիր), որը “թարգմանում է” ծրագրային կոդը օպերացիոն համակարգի համար: PHP-ի ուսումնասիրությունը հանդիսանում է առանձին ձեռնարկի թեմա, սակայն համառոտակի քննարկենք լեզվի կիրառության որոշ առանձնահատկությունները:

PHP-ի կիրառության եղանակը սերվերային ծրագրավորման տեսակետից բացարձակապես նման է ASP ծրագրավորման տեխնոլոգիային, իսկ լեզվի քերականությունը՝ JavaScript կամ C++ լեզուներին: Բերենք մի պարզ սերվերային ծրագրի օրինակ՝

```
<html>
<head>
<title>
PHP լեզվի կիրառության օրինակ
</title>
</head>
<body>
<?php
echo “Բարև ձեզ: Ես PHP ծրագիրն եմ”;
?>
</body>
</html>
```

Բերված օրինակից երևում է, որ PHP կոդի ներդրումը սերվերային ծրագրում շատ նման է ASP կոդի ներդրմանը: Տարբերությունը՝ բացող և փակող փակագծերի տեսքն է՝ ASP-ում բացող և փակող փակագծերն են, համապատասխանաբար, “<%” և “%>”, իսկ PHP-ում՝ “<?php” և “?>”: Ցանկացած դեպքում, այդպիսի փակագծերը թելադրում են՝ ASP-ի համար, ինտերպրետատորին, իսկ PHP-ի՝ պրեպրոցեսորին, որ դրանցում պառփակված է սերվերի վրա կատարվող ծրագրային հաղվածք: Եվս մեկ տարբերությունը՝ օգտվողի բրաուզերին HTML կոդի, տեքստի և փոփոխականների արժեքների հաղորդման եղանակն է՝ PHP-ում դա իրագործվում է հատուկ **echo** օպերատորի միջոցով, ընդ որում տեքստը և գծանշանման կոդը վերցվում են կրկնակի չակերտների մեջ, իսկ փոփոխականները մատնանշվում են (այսինքն պրեպրոցեսորին հայտնվում է, որ դա փոփոխական է) դոլարի նշանի (\$) միջոցով:

Տարբերությունը ավելի լավ պատկերացնելու նպատակով կազմենք միևնույն սերվերային ծրագիրը, օգտագործելով ASP և PHP տեխնոլոգիաները: Ծրագրում պարզվում է սերվերին հարցում կա-

տարած հաճախորդի (օրինակ՝ կայքի այցելույի) բրաուզերի տեսակը և ստացված արդյունքը հետ է ուղարկվում օգտվողին HTML էջի տեսքով: Ինչպես գիտենք նման ինֆորմացիան պարունակվում է սերվերի միջավայրի փոփոխականների զանգվածում: Կոնկրետ այդ փոփոխականի անունն է “HTTP_USER_AGENT”: Ինչպես մենք արդեն գիտենք, ASP ծրագրերում միջավայրի փոփոխականների արժեքները կարելի է ստանալ Request օբյեկտի ServerVariables զանգվածից: Այսինքն, եթե գրանցենք՝ Request.ServerVariables(“HTTP_USER_AGENT”), ապա զանգվածից կստանանք բրաուզերի բնութագրերը պարունակող փոփոխականի արժեքը: PHP-ում նույն փոփոխականը պարունակվում է \$_SERVER[] զանգվածում՝ \$ _SERVER[“HTTP_USER_AGENT”]:

Այժմ ներկայացնենք երկու եղանակների օգտագործումը միևնույն խնդիրը լուծելու համար:

-----ASP-ով գրված ծրագրային հատվածը-----

```
<body>
<b>
<%
=Request.ServerVariables("HTTP_USER_AGENT");
%>
</b>
</body>
```

-----PHP-ով գրված ծրագրային հատվածը-----

```
<body>
<b>
<?php
echo $_SERVER["HTTP_USER_AGENT"];
?>
</b>
</body>
```

Երկու դեպում էլ հաճախորդի բրաուզերի պատուհանում կարտապատկերվի հետևյալ ինֆորմացիան (իհարկե, եթե բրաուզերի տեսակը Internet Explorer-ն է)՝

Mozilla 4.0 (Compatible; MSIE 5.01; Windows NT 5.0)

Քանի որ PHP-ին հիմնականում կողմնորոշված է սերվերային սկրիպտերի ստեղծմանը, ապա դրա միջոցով հնարավոր է իրագործել բացարձակապես այն ամենն, ինչ անում են CGI կամ ASP սկրիպտերը՝ մշակել ֆորմաների տվյալները, դիմամիկորեն ստեղծել և հաղորդել օգտվողներին HTML էջերը, աշխատել տվյալների բազաների հետ և այլն: Հիմնականում PHP-ն

օգտագործվում է միջին չափի՝ MySQL SPQL-ի հետ, սակայն ունի բազմաթիվ ֆունկցիաներ, որոնք հնարավորություն են տալիս աշխատել նաև մի շարք այլ բազաների, օրինակ, Adabas D, dBase, Informix Oracle, Sybase և, կամայական ODBC տեխնոլոգիան սատարող, բազաների հետ:

Ֆորմաների Յուրաքանչյուր էլեմենտի պարունակությունը ավտոմատ կերպով հասանելի է դարձնում PHP սերվերային ծրագրերին: Այդ նպատակի համար օգտագործվում են \$_POST[] (եթե հաղորդման եղանակը POST է) կամ \$_GET[] (եթե հաղորդման եղանակը GET է) զանգվածները, օրինակ՝
\$_POST[ֆորմայի էլեմենտի անունը]:

Ֆորմայից տվյալների ստացման ASP-ում և PHP-ում դիտարկենք §4.3-ում բերված օրինակի վրա: Երկու դեպքում էլ օգտագործվում է միևնույն ֆորման՝

```
<form action = " " method = "post">
<p>Ձեր անունը: <input name ="firstname" size ="40"></p>
<p>Պաղպաղակի ո՞ր տեսակն էք դուք նախընտրում:
<select name = "flavor">
<option>վանիլային</option>
<option>ելակի</option>
<option>շոկոլադե</option>
<option>ընկույզով</option>
</select></p>
<p><button type ="submit">Ուղարկել</button>
</form>
</p>
```

միայն action ատրիբուտի արժեքը ASP ֆայլի համար կլինի՝
action = "/scripts/submit.asp"
իսկ PHP ֆայլի համար՝
action = "/scripts/submit.php"

ASP էջի ծրագրային կոդը կլինի հետևյալը՝

```
<html>
<head>
<title>
Ֆորմայի էլեմենտների արժեքների ստացումը (ASP)
<title>
</head>
<body>
Բարև Ձեզ, <%= Request.Form("firstname") %>:
Դուք սիրում էք <%= Request.Form("flavor") %> պաղպաղակ:
</body>
```

```

<html>
  PHP էջի ծրագրային կոդը կլինի հետևյալը՝
<html>
<head>
<title>
Ֆորմայի էլեմենտների արժեքների ստացումը (PHP)
</title>
</head>
<body>
Բարև Ձեզ, <?php echo $_POST["firstname"] ?>:
Դուք սիրում էք <?php echo $_POST["flavor"] ?> պաղպաղակ:
</body>
</html>

```

Երկու դեպքում էլ հաճախորդի (օրինակ, Արմենի, որը սիրում է ելակի պաղպաղակ) բրաուզերի պատուհանում կարտապատկերվի հետևյալը՝

Բարև Ձեզ, Արմեն: Դուք սիրում էք ելակի պաղպաղակ:

Եթե ըվյալների աղբյուրը չունի որոշակի արժեք (այսինքն նշված չէ տվյալների հաղորդման մեթոդը), ապա կարելի է օգտագործել գերգլոբալ \$ _REQUEST զանգվածը, որը ընդգրկում է GET, POST, FILE և COOKIE տվյալների խարնուրդը: Հասկանալի է, որ համապատասխան արժեքը ստացվում է կրիչի անվանը համապատասխան: Օրինակ, վերը բերված օրինակի տվյալները կարելի է ստանալ հետևյալ եղանակով՝

```

Բարև Ձեզ, <?php echo $_REQUEST["firstname"] ?>:
Դուք սիրում էք <?php echo $_REQUEST["flavor"] ?> պաղպաղակ:

```

Սակայն PHP-ի հնարավորությունները չեն սահմանափակվում web-էջերի և սերվերային ծրագրերի ստեղծումով: Լեզվի հնարավորությունները թույլ են տալիս նաև դինամիկորեն ստեղծել PDF ֆայլեր և, նույնիսկ Flash վիդեո ֆայլեր:

Սերվերների մեծամասնության համար PHP-ն առաքվում է որպես սատարող մոդուլ, իսկ CGI ստանդարտը սատարող սերվերների համար ծառայում է որպես CGI պրոցեսոր: Դա ազատ ընտրության հնարավորություն է ստեղծում ինչպես օպերացիոն համակարգերի և web-սերվերների տեսակետից, այնպես էլ օբյեկտակողմնորոշված կամ կառուցվածքային ծրագրավորման տեխնոլոգիաների ընտրության տեսակետից:

PHP-ն ընդգրկում է բազմաթիվ ներկառուցված դեկլարող և տարբեր օբյեկտների (մաթեմատիկական արտահայտությունների, տողային փոփոխականների, զանգվածների, ցանցերի և առանձին

ֆայլերի և շատ այլ օբյեկտների) հետ աշխատելու համար ֆունկցիաներ:

Առաջադրանքներ 3-րդ և 4-րդ գլուխների թեմաներով

Ա.1. Հինգ աշխատակիցներից յուրաքանչյուրի համար անկետայում բերվում են տվյալներ, ազգանունը, աշխատավարձը և երեխաների քանակը: Պահանջվում է գրել սցենար որոշելու համար ընտանիքում մեկ մարդու եկամուտը: Բացի դրանից պահանջվում է որոշել այն աշխատակիցների քանակը որոնք մեկ մարդու հաշվով ունեն մինիմալ եկամուտ: Կառուցել դիագրամա, որը արտացոլի ընտանիքում մեկ մարդու հաշվով եկամուտը:

Ա.2. Վեց աշխատակիցներից յուրաքանչյուրի համար անկետայում բերվում են տվյալներ: Ազգանունը և աշխատանքի ընդունվելու տարին: Պահանջվում է գրել սցենար որոշելու համար աշխատանքի ստաժը և նույն ստաժն ունեցող աշխատակիցների մաքսիմալ քանակը: Կառուցել դիագրամա, որը արտացոլի աշխատակիցների աշխատանքային ստաժը:

Ա.3. Վեց աշխատակիցներից յուրաքանչյուրի համար անկետայում բերվում են տվյալներ: Ազգանունը և աշխատավարձը : Որոշված է յուրաքանչյուր աշխատակցին նշանակել պարգևատրում հետևյալ սկզբունքով: Եթե աշխատավարձը փոքր է քան միջին աշխատավարձը, ապա պարգևատրման չափը կազմում է աշխատավարձի 50%-ը մնացած դեպքերում աշխատավարձի 30%-ը: Պահանջվում է գրել սցենար որոշելու համար աշխատակցին վճարվող ընդհանուր գումարը: (աշխատավարձ պլուս պարգևատրում): Բացի դրանից պետք է որոշել այն աշխատակիցների քանակը, որոնք ստացել են մաքսիմալ աշխատավարձ:

Ա.4. Բերվում են հինգ անվանումներով ապրանքների գնումների մասին տվյալներ: միավորի գինը և ձեռք բերված օրինակների քանակը: Գրել սցենար, որը որոշի ապրանքների ձեռք բերման վրա ծախսված գումարը: Որոշել թե արդյոք կան ապրանքներ, որոնց վրա ծախսվել է նույն գումարը և քանիսն են նրանք: Կառուցել դիագրամա, որը արտացոլի տարբեր ապրանքների ձեռք բերման վրա ծախսված գումարը:

Ա.5. Անկետայում լրացվում է ինֆորմացիա վեց ուսանողների մասին: Ազգանունը և կիսամյակի 4 գնահատականները: Գրել սցենար որը, որոշի ուսանողի կատեգորիան և յուրաքանչյուր կատեգորիայում ուսանողների քանակը: Կատեգորիան որոշվում է հետևյալ կերպ՝ բոլոր քննությունները 5 – ով հանձնած ուսանողները դասվում են «գերազանցիկներ» կատեգորիային,

նրանք, որոնք ունեն թեկուզ մեկ 2 «ամբավարարներ» կատեգորիային իսկ մնացածները «բավարարներ» կատեգորիային

Ա.6. Տրված է իրական թվերի միաչափ զանգված: Գրել սցենար, որը որոշի զանգվածի դրական էլեմենտների քանակը:

Ա.7. Տրված է իրական թվերի միաչափ զանգված: Գրել սցենար, որը որոշի զանգվածի բացասական էլեմենտների քանակը:

Ա.8. Տրված է ամբողջ թվերի միաչափ զանգված: Գրել սցենար, որը որոշի մինիմալ էլեմենտների քանակը:

Ա.9. Տրված է ամբողջ թվերի միաչափ զանգված: Գրել սցենար, որը որոշի 7– ին բազմապատիկ էլեմենտների քանակը:

Ա.10. Տրված է ամբողջ թվերի միաչափ զանգված: Գրել սցենար, որը որոշի վերջին մինիմալ արժեքի համարը:

Ա.11. Գրել սցենար, որը որոշի ամբողջ թվերի միաչափ զանգվածում առաջին մեծագույն արժեքի համարը:

Ա.12. Գրել սցենար, որը որոշի միաչափ զանգվածում տրված թվի հետ համընկնող թվերի քանակը:

Ա.13. Գրել սցենար, որը որոշի թե զանգվածում կան արդյոք էլեմենտներ, որոնց արժեքները համընկնում են:

Ա.14. Գրել սցենար, որը որոշի թե զանգվածի բոլոր էլեմենտները տարբեր են թե ոչ:

Ա.15. Գրել սցենար, որը որոշի թե քանի տարբեր թվեր կան տրված զանգվածում:

ՀԱՎԵԼՎԱԾՆԵՐ

Հավելված 1. Ոճերի աղյուսակների հատկությունների տեղեկատու

Հ1.1. Չափման միավորները

Հարաբերական միավորներ:

em – ընթացիկ տառաշարի “m” տառի բարձրությունը;

en - ընթացիկ տառաշարի “n” տառի բարձրությունը;

ex - ընթացիկ տառաշարի “x” տառի բարձրությունը;

px – պիքսել՝ էկրանի բացվածքի միավոր էլեմենտ (կետ), որը կախված է մոնիտորի կարգավորումներից և վիդեոքարտից:

Internet Explorer բրաուզերում em-ը և ex-ը փոխարինված են pt-ով, իսկ en-ը px-ով:

Բացարձակ միավորներ:

in - դյույմ (inch)՝ 1 դյույմը մոտավորապես հավասար 2.5 սմ;

cm – սանտիմետր;

mm – միլիմետր;

pt – հավասար է 1/72 դյույմի;

pc – հավասար է 12 pt:

Բացարձակ միավորները խորհուրդ է տրվում օգտագործել միայն այն դեպքերում, երբ հայտնի են ելքի սարքի աշխատանքային մակերեսի չափսերը:

Հ1.2. Տառաշարի հատկությունները

font-family - սատարվում է բոլոր էլեմենտներով:

Ժառանգվում է:

Սահմանում է տառաշարի ընտանիքը, օրինակ՝

font-family: “Arial Armenian, Times Armenian”: Ոճերի աղյուսակում պետք է վերցվի կրկնակի չափերսների մեջ:

Գրանցումը սկրիպտում (սցենարում)՝ fontFamily:

font-size - սատարվում է բոլոր էլեմենտներով:

Ժառանգվում է:

Տառերի չափսը կարող է տրվել բացարձակ միավորներով, կամ տոկոսային հարաբերությամբ ծնողական (տվյալ տեղը պարունակող) էլեմենտի հանդեպ:

Գրանցումը սկրիպտում՝ fontSize:

font-style - սատարվում է բոլոր էլեմենտներով:

ժառանգվում է:

Արժեքները՝

normal – սովորական,

italic – շեղ,

oblique – նույնն է, ինչ italic:

Գրանցումը սկրիպտում՝ `fontStyle`:

font-weight - սատարվում է բոլոր էլեմենտներով:

ժառանգվում է:

Արժեքները՝ թվային 100-ից մինչև 900, 700-ը համապատասխանում է bold արժեքին, իսկ 400-ը՝ normal: Կարելի է օգտագործել հարաբերական արժեքներ (հարաբերված ծնողական էլեմենտի)՝ normal, bold, bolder, lighter:

Գրանցումը սկրիպտում՝ `fontWeight`:

font-variant - սատարվում է բոլոր էլեմենտներով:

ժառանգվում է:

Արժեքները՝

normal – ըստ լրելայն,

small-caps – տեքստը գրանցվում է մեծատառերով, սակայն նույն չափսի, ինչ փոքրատառերը:

Գրանցումը սկրիպտում՝ `fontVariant`:

Չ1.3. Գույնը և ֆոնը

color - սատարվում է բոլոր էլեմենտներով:

ժառանգվում է:

Սահմանում է էլեմենտի տեքստի գույնը:

Արժեքները՝

գույնի անվանումով, օրինակ՝ red, white, blue, green,

RGB սանդղակով, օրինակ՝ rgb(255,0,0), rgb(255,255,255),

rgb(0,0,255), rgb(0,255,0),

տասնվեցական սանդղակով, օրինակ՝ #ff0000, #ffffff, #0000ff, #00ff00:

Օրինակ՝ color:green, color:rgb(0,255,0), color:#00ff00,

Ըստ լրելայն սև է՝ black կամ rgb(0,0,0) կամ #000000:

Գրանցումը սկրիպտում՝ `color`:

background-attachment - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Որոշում է սահեղութի (scroll) ֆոնը (հետմապատկերը), թե ոչ:

Արժեքները՝

fixed - այո,

scroll – ոչ:

Օրինակ՝ “background-attachment:scroll”:

Գրանցումը սկրիպտում՝ backgroundAttachment:

background-color - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Արժեքները՝ շնորհիվում են ինչպես և color հատկությանը: Ըստ լռելյայն սպիտակ է:

Գրանցումը սկրիպտում՝ backgroundColor:

background-image - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Արժեքները ֆոնային պատկերի URL հասցե:

Օրինակ՝ “background-image:url(հասցե)”:

Գրանցումը սկրիպտում՝ backgroundImage:

background-position - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Կիրառվում է այն դեպքերում, երբ տրված է ֆոնային պատկեր:

Որոշում է պատկերի սկզբնական դիրքը:

Արժեքները՝

Տրվում են թվերի զույգով, որոնք սահմանում են ֆոնային պատկերի ձախ վերին անկյան կոորդինատները (ուղղահայաց և հորիզոնական): Կարելի է սահմանել նաև հետևյալ բանալիական բառերի միջոցով՝ top, center, bottom, left, right:

Օրինակ՝ background-position:10,10 կամ background-position: top left:

Ըստ լռելյայն ձախ վերին անկյան կոորդինատներն են՝ top, left:

Գրանցումը սկրիպտում՝ backgroundPosition, backgroundPositionX, backgroundPositionY:

background-repeat - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Որոշում է՝ պե՞տք է թե ո՞չ կրկնել ֆոնային պատկերը (եթե այն կա):

Արժեքները՝

repeat – կրկնվում էերկու ուղղություններով, ընդունվում է ըստ լռելյան,

repeat-x – կրկնվում է միայն հորիզոնական,

repeat-y – կրկնվում է միայն ուղղահայաց,

no-repeat – չի կրկնվում:

Գրանցումը սկրիպտում՝ backgroundRepeat:

Ըստ լռելյայն արժեքը՝ repeat:

Չ1.4. Տեքստի հատկությունները

letter-spacing - սատարվում է բոլոր էլեմենտներով:

ժառանգվում է:

Որոշում է տարածությունը տառերի միջև:

Արժեքները՝

թվային է pt-ով, px-ով և այլն:

Գրանցումը սկրիպտում՝ letterSpacing:

line-height - սատարվում է բոլոր էլեմենտներով:

ժառանգվում է:

Որոշում է ընթացիկ տողի բարձրությունը:

Արժեքները՝

թվային են և որոշում են ընթացիկ էլեմենտի տառաչարի բարձրությունը բազմապատկած այդ թվով:

Օրինակ՝ line-height:1.2:

Գրանցումը սկրիպտում՝ lineHeight:

text-decoration - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Որոշում է տեքստի ձևավորումը (ընդգծված, ջնջած և այլն):

Արժեքները՝

underline – ընդգծված,

line-through – ջնջած (օրինակ՝ ջնջած),

box – շրջանակով,

blink – թարթող:

Ըստ լրելայն՝ none:

Գրանցումը սկրիպտում՝ textDecoration:

text-transform - սատարվում է բոլոր էլեմենտներով:

ժառանգվում է:

Արժեքները՝

capitalize – դարձնում է մեծատառ ամեն մի բառի առաջին տառը,

lowercase - ամբողջ տեքստը գրանցվում է փոքրատառերով,

uppercase - ամբողջ տեքստը գրանցվում է մեծատառերով,

none – հանում է բոլոր նախկին կարգավորումները:

Գրանցումը սկրիպտում՝ textTransform:

text-indent - սատարվում է բոլոր էլեմենտներով:

ժառանգվում է:

Որոշում է տեքստի գրանցման դաշտերի խորությունը:

Արժեքները՝

բացարձակ մեծություններ կամ տոկոսներ (ծնողական էլեմենտի դաշտից):

Օրինակ՝ text-indent:10px կամ text-indent:80%:

Գրանցումը սկրիպտում՝ textIndent:

vertical-align - սատարվում է ներդրված էլեմենտներով:

Չի ժառանգվում:

Որոշում է ընթացիկ էլեմենտի ուղղահայաց հավասարեցումը:

Արժեքները՝

baseline – հավասարեցում ծնողական էլեմենտի օրինակով,

middle – միջին գիծը հավասարեցվում է ըստ ծնողականի գումարած վերջինի տողի բարձրության կեսը,

super – փոխադրում է էլեմենտը վերին ռեգիստր,

sub – փոխադրում է էլեմենտը ներքևի ռեգիստր,

text-top – հավասարեցնում է էլեմենտը ըստ ծնողականի տեքստի վերի գծի,

text-bottom – հավասարեցնում է էլեմենտը ըստ ծնողականի տեքստի ստորին գծի,

top – հավասարեցնում է էլեմենտը ըստ ընթացիկ տողի ամենաբարձր էլեմենտի վերին գծի,

bottom – հավասարեցնում է էլեմենտը ըստ ընթացիկ տողի ամենացածր էլեմենտի ստորին գծի:

Գրանցումը սկրիպտում՝ verticalAlign:

text-align – սատարվում է բոլոր էլեմենտներով:

Ժառանգվում է:

Որոշում է տեքստի հորիզոնական հավասարեցումը պարունակող էլեմենտում:

Արժեքները՝

left – ձախ,

right – աջ,

center – կենտրոնով,

justify – ամբողջ լայնքով:

Գրանցումը սկրիպտում՝ textAlign:

31.5. Չափսերի, դիրքի և տեսանելիության հատկությունները

width – սատարվում է div, span և փոխարինվող էլեմենտներով:

Չի ժառանգվում:

Սահմանում է էլեմենտի լայնությունը:

Արժեքները տրվում են չափման միավորներով կամ տոկոսներով:

Ըստ լրելայն արժեքը auto:

Գրանցումը սկրիպտում՝ width, pixelWidth, posWidth:

height – սատարվում է div, span և փոխարինված էլեմենտներով:

Չի ժառանգվում:

Սահմանում է էլեմենտի բարձրությունը:

Արժեքները տրվում են չափման միավորներով կամ տոկոսներով:

Ըստ լրելայն արժեքը՝ auto:

Գրանցումը սկրիպտում՝ height, pixelHeight, posHeight:

position – սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Սահմանում է, թե ինչպես պետք է հաշվարկվի էլեմենտի դիրքը էկրանի հարթությունում:

Արժեքները՝

absolute – էլեմենտը գտնվելու է որոշակի դիրքում ֆոնի հանդեպ և էջը պտտելու դեպքում շարժվում է ֆոնի հետ:

static – էլեմենտը գտնվելու է որոշակի դիրքում ֆոնի հանդեպ, սակայն էջը պտտելիս չի շարժվում:

relative – էլեմենտը տեղադրվում է նախորդ էլեմենտից անմիջապես հետո՝ համապատասխան սկզբնական կողմում գրանցման հաջորդականության:

Ըստ լրելայն արժեքը՝ **relative**:

Գրանցումը սկրիպտում՝ **position**:

left – սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Սահմանում է էլեմենտի հորիզոնական կոորդինատը:

Արժեքները կարող են տրվել չափման միավորներով կամ տոկոսներով՝ ծնողական էլեմենտի լայնությանը հարաբերված:

Գրանցումը սկրիպտում՝ **left**, **pixelLeft**, **posLeft**:

top – սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Սահմանում է էլեմենտի ուղղահայաց կոորդինատը:

Արժեքը կարող է տրվել չափման միավորներով կամ ծնողական էլեմենտի բարձրության տոկոսներով:

Գրանցումը սկրիպտում՝ **top**, **pixelTop**, **posTop**:

visibility – սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Թույլ է տալիս էլեմենտը դարձնել տեսանելի կամ անտեսանելի:

Անտեսանելի էլեմենտները զբաղեցնում են նույն տեղը և նույն կերպով են ազդում այլ էլեմենտների դասավորության վրա, ինչպես և տեսանելիները, սակայն դարձնում են թափանցիկ: Այդ հատկությունը կարող է օգտագործվել միևնույն տեղը զբաղեցնող մի քանի էլեմենտներից միայն մեկը արտապատկերելու նպատակով:

Արժեքները՝

visible – էլեմենտը տեսանելի է,

hidden – էլեմենտը անտեսանելի է,

inherit – էլեմենտը ժառանգում է ծնողական էլեմենտի տեսանելիությունը:

Ըստ լրելայն արժեքը՝ **visible**:

Գրանցումը սկրիպտում՝ **visibility**:

z-index - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Սահմանում է կարգը, ըստ որի էլեմենտները պետք է միմյանց ծածկեն: Ավելի բարձր z-index ունեցող էլեմենտները արտապատկերվում են ցածր z-index ունեցող էլեմենտների “վրայից”, ստեղծելով տարածաչափական արտապատկերման տպավորություն (կոչվում է 2.5-չափանիություն)

Արժեքները՝ ամբողջ թվեր են, օրինակ “z-index:3”, “z-index:2”:

Գրանցումը սկրիպտում՝ z-index:

display - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Սահմանում է՝ ցուցադրվելու է էլեմենտը, թե ոչ: Չցուցադրվող էլեմենտի համար ի տարբերություն visibility հատկությունից տեղ չի նախատեսվում և այն չի ազդում մյուս էլեմենտների դասավորության վրա:

Արժեքները՝

block, inline, list-item – էլեմենտը ցուցադրվում է,

none - էլեմենտը չի ցուցադրվում:

Ըստ լրելայն արժեքը՝ block:

Գրանցումը սկրիպտում՝ display:

clip - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Որոշում է էլեմենտի տեսանելի մասի տիրույթը: Այն ամենը, ինչ գտնվում է հատկության մեջ նշված տիրույթից դուրս անտեսանելի է (կտրատվում է):

Արժեքները՝

auto – ցուցադրվում է ըստ ծնողական էլեմենտի չափսերի,

rect (top, right, bottom, left) – սահմանվում է ուղղանկյունի փակագծերում նշված կոորդինատներով:

Ըստ լրելայն արժեքը՝ auto:

Գրանցումը սկրիպտում՝ clip:

float - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Թույլ է տալիս ստեղծել “լողացող” բլոկներ, որոնք շրջափակվում են տեքստով կամ այլ էլեմենտներով:

Արժեքները՝

none - էլեմենտը ցուցադրվում է այն տեղում, որում գրանցված է ծրագրային կոդում:

left – տեքստը կամ այլ էլեմենտները շրջանցում են տվյալ էլեմենտը ձախից,

right - տեքստը կամ այլ էլեմենտները շրջանցում են տվյալ էլեմենտը աջից:

Ըստ լրեյայն արժեքը՝ none:

Գրանցումը սկրիպտում՝ float:

overflow - սատարվում է բոլոր էլեմենտներով:

Չի ժառանգվում:

Կարգավորում է էլեմենտի պարունակության ցուցադրումը այն դեպքերում, եթե այն լիովին չի տեղավորվում էլեմենտում: Որպեսզի հատկությունը աշխատի անհրաժեշտ է, որպեսզի դրա չափսերը լինեն սահմանված (չափման միավորներով):

Արժեքները՝

none – նշանակում , որ եթե պարունակությունը գերազանցի էլեմենտի սահմանները այն միևնույն է կցուցադրվի (նույնիսկ սահմաններից դուրս),

clip – սահմաններում չտեղավորվող մասերը կկտրատվեն,

scroll – օգտվողը կկարողանա դիտարկել չտեղավորվող մասերը պտտման ստեղծների օգնությամբ:

Ըստ լրեյայն արժեքը՝ none:

Գրանցումը սկրիպտում՝ overflow:

Չ1.6. Շրջանակների և լուսանցքների հատկությունները

border-top, border-right, border-left, border-bottom, border – սատարվում է բոլորային էլեմենտներով:

Չի ժառանգվում:

Սահմանվում է հիմնականում շրջանակների հաստությունը չափման միավորներով:

Օրինակ՝ border:5px:

Գրանցումը սկրիպտում՝ borderTop, borderRight, borderLeft, borderBottom, border:

border-top-color, border-right-color, border-left-color, border-bottom-color, border-color – սատարվում է բոլորային էլեմենտներով:

Չի ժառանգվում:

Սահմանում են շրջանակների գույնը:

Ըստ լրեյայն՝ անգույն:

Օրինակ՝ border-color:#ff000, border-color:red, border-color:rgb(255, 0,0):

Գրանցումը սկրիպտում՝ borderTopColor, borderRightColor, borderLeftColor, borderBottomColor, borderColor:

border-top-style, border-right-style, border-left-style, border-bottom-style, border-style - սատարվում է բոլորային էլեմենտներով:

Չի ժառանգվում:

Սահմանում է շրջանակների նկարելու ոճը:

Արժեքները՝

none - շրջանակ չի նկարվում,

solid - շրջանակը նկարվում է թավ գծով,

dotted - շրջանակը նկարվում է ընդհատ կարճ գծիկներով,

dashed - շրջանակը նկարվում է ընդհատ գծով,

double – նկարվում է կրկնակի շրջանակ (հաստությունը պետք է լինի 3 պիքսելից ոչ պակաս),

groove - էլեմենտը թվում է ներս ընկած,

ridge - էլեմենտը թվում է “ուռուցիկ”,

inset - ստեղծվում է ձախ վերին անկյունից լուսավորվածության տպավորություն,

outset - ստեղծվում է աջ ստորին անկյունից լուսավորվածության տպավորություն:

Ըստ լրելայն՝ none:

Գրանցումը սկրիպտում՝ borderTopStyle, borderRightStyle, borderLeftStyle, borderBottomStyle, borderStyle:

border-top-width, border-right-width, border-left-width, border-bottom-width, border-width - սատարվում է բլոկային էլեմենտներով:

Չի ժառանգվում:

Սահմանում է շրջանակի հաստությունը երեք հնարավոր արժեքներով (ըստ լրելայն արժեքը՝ medium)

thin - բարակ՝ 1 պիքսել,

medium – միջին՝ 2 պիքսել,

thick – հաստ՝ 5 պիքսել:

Գրանցումը սկրիպտում՝ borderTopWidth, borderRightWidth, borderLeftWidth, borderBottomWidth, borderWidth:

margin-top, margin-right, margin-left, margin-bottom, margin - սատարվում է բլոկային էլեմենտներով:

Չի ժառանգվում:

Սահմանում է արտաքին լուսանցքների լայնությունը (տվյալ էլեմենտի հեռավորությունը այլ էլեմենտներից) չափման միավորներով կամ ծնողական էլեմենտի լայնության տոկոսներով:

Ըստ լրելայն՝ 2պիքսել:

Գրանցումը սկրիպտում՝ marginTop, marginRight, marginLeft, marginBottom, margin:

padding-top, padding-right, padding-left, padding-bottom, padding - սատարվում է բլոկային էլեմենտներով:

Չի ժառանգվում:

Սահմանում է ներքին լուսանցքների լայնությունը (էլեմենտի պարունակության հեռավորությունը շրջանակներից) չափման միավորներով կամ ծնողական էլեմենտի լայնության տոկոսներով:

Ըստ լրելայն՝ 2պիքսել:

Գրանցումը սկրիպտում՝ paddingTop, paddingRight, paddingLeft, paddingBottom, padding:

Հավելված 2. Առավել հաճախ կիրառվող պրիմիտիվների ցուցակը

Սիմվոլը	Կոդը նիշերով	Թվային կոդը	Նկարագրությունը
A - Z (մեծատառ)		A - Z	A-A B-B և այլն
a - z (փոքրատառ)		a - z	a-a b- b և այլն
0 - 9		0 - 9	0-0 1- 1 և այլն
	 	 	Անխզելի պրոբել
!		!	Բացականչական նշան
“	"	"	Ուղիղ չափերտներ
#		#	Համար
\$		$	Դոլար
%		%	Տոկոս
&	&	&	Ամպերսանդ
'		'	Ապոստրոֆ
((Չափ կլոր փակագիծ
))	Աջ կլոր փակագիծ
*		*	Աստղանիշ
+		+	Գումարած
,		,	Ստորակետ
.		.	Կետ
/		/	Շեղ գծիկ
:		:	Երկու կետ
;		;	Կետ ստորակետով
<	<	<	Փոքր է
=		=	Հավասար է
>	>	>	Մեծ է
?		?	Հարցական
@		@	Առևտրային էտ
[[Ուղղանկյուն փակագիծ
]]	Ուղղանկյուն փակագիծ
\		\	Հակառակ շեղ գծիկ
		|	Ուղիղ գծիկ
{		{	Չափոր փակագիծ
}		}	Չափոր փակագիծ

Ինտերնետ ծրագրավորման հիմունքներ- քննական տեստ հարցաշար

Հավելված

Գնահատականը ավանդական համակարգում	Ռեյտինգային բալերը
5-գերազանց	90-100
4-լավ	70-89
3-բավարար	50-69
2- անբավարար	<50

Նշել ճիշտ պատասխան(ներ)ը կամ լրացնել բաց թողնված տեղերը

Հարց N1

Քանի տասական մեկ բայտանոց իդենտիֆիկատորներից է բաղկացած IP հասցեն

- ☐ մեկ
- ☐ երկու
- ☐ երեք
- ☐ չորս
- ☐ հինգ
- ☐ վեց
- ☐ յոթ
- ☐ ութ
- ☐ իննը
- ☐ տաս

Ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N2

Սերվերի և հաճախորդի միջև կապի համար FTP արձանագրությունը քանի՞ TCP -ի միացումներ է օգտագործում, որո՞նք են դրանք

- ☐ ղեկավարող
- ☐ ճյուղավորվող
- ☐ տեղեկատվական

Ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N3

Թվարկեր ինտերնետում մատուցվող ծառայությունները

- ☐ **WWW**
- ☐ **BBC**
- ☐ **BBS**
- ☐ **E-mail**
- ☐ **MMM**
- ☐ **Usenet**
- ☐ **Telnet**

Ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N4

Ինչպիսին են լինում տեգերը

- ☐ **Կենտ**
- ☐ **Եզակի**
- ☐ **պարզ**
- ☐ **զույգ**
- ☐ **հոգնակի**
- ☐ **բաղադրյալ**
- ☐ **բաց**
- ☐ **փակ**

Ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N5

Ինչո՞վ են իրարից բաժանվում ատրիբուտները

- ☐ **ստորակետով**
- ☐ **կետով**
- ☐ **կետստորակետով**
- ☐ **բացատով**
- ☐ **վերջակետով**

Ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N6

Համաձայն XMLստանդարտի բոլոր տեգերի և ատրիբուտների անունները ինչպես պետք է գրանցվեն

- ☐ **փոքրատառով**
- ☐ մեծատառով
- ☐ սկսվի մեծատառով
- ☐ սկսվի փոքրատառով
- ☐ եթե ճիշտ է ապա պետք է վերցվեն անկյունային փակագծերի մեջ
- ☐ ձևավոր փակագծերի մեջ
- ☐ չակերտների մեջ
- ☐ քառակուսի փակագծերի մեջ
- ☐ կենտ չակերտների մեջ

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N7

Նոր ստանդարտով թույլատրվում են դատարկ արժեքներ թե ոչ:

- ☐ **այո**
- ☐ **ոչ**

Ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N8

Նոր ստանդարտում Html տեգը ունի մեկ ատրիբուտ, որի միջոցով փաստաթղթին միացվում են XML անունների բազմությունը, ինչպես է այն կոչվում:

- ☐ **xlnms**

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N9

Web էջում գծանշելու նպատակով օգտագործվում են հատուկ տեգեր, որոնք կարելի է բաժանել երկու հիմնական խմբերի: Ինչպես են նրանք կոչվում:

- ☐ տեքստի ֆորմատավորման ոճի
- ☐ տեքստի պայմանական ոճի
- ☐ տեքստի ֆիզիկական ոճի

- ☐ տեքստի տրամաբանական ոճի
- ☐ տեքստի ձևավորման ոճի

Ծիշտ պատասխանը գնահատվում է 1 բալ

Հարց N10

Քանի՞ մակարդակի են լինում վերնագրերը:

- ☐ մեկ
- ☐ երկու
- ☐ երեք
- ☐ չորս
- ☐ հինգ
- ☐ վեց
- ☐ յոթ
- ☐ ութ
- ☐ ինը
- ☐ տաս

Ծիշտ պատասխանը գնահատվում է 2 բալ

Հարց N11

HTML լեզվում օգտագործվում են երեք տեսակի ցուցակներ: Թվարկեք դրանք:

- ☐ կարգավորված կամ համարակալված ցուցակներ
- ☐ որոշումների ցուցակներ
- ☐ Չկարգավորված կամ պիտակավորված ցուցակներ
- ☐ հայտարարությունների ցուցակներ
- ☐ սահմանումների ցուցակներ

Ծիշտ պատասխանը գնահատվում է 2 բալ

Հարց N12

Հասցեները հիպերհղումների ժամանակ լինում են երկու տիպի: Որոնք են դրանք

- ☐ անվանական
- ☐ **բացարձակ**
- ☐ լոկալ
- ☐ **հարաբերական**
- ☐ գլոբալ

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N13

Հիպերիդումները լինում են երկու տեսակի: Որոնք են դրանք:

- ☐ բաց
- ☐ **ներքին**
- ☐ փակ
- ☐ **արտաքին**

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N14

Համաձայն XML նոր ստանդարտի name ատրիբուտը փոխարինվում է ի՞նչ ատրիբուտով:

- ☐ **id**

Ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N15

Ըստ ֆունկցիոնալ նշանակության գրաֆիկական բաղադրիչները կարելի է պայմանականորեն բաժանել երեք ծավալուն խմբերի: Թվարկեք դրանք:

- ☐ **պատկերազարդ գրաֆիկա**
- ☐ ծաղկազարդ գրաֆիկա
- ☐ **ֆունկցիոնալ գրաֆիկա**
- ☐ **դեկորատիվ գրաֆիկա**
- ☐ ղեկավարող գրաֆիկա

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N16

Թվարկեք Align ատրիբուտի ստանդարտ արժեքները:

- ☐ button
- ☐ **bottom**
- ☐ width
- ☐ **right**
- ☐ **left**
- ☐ height

Ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N17

Web էջերի և մուլտիմեդիայի էլեմենտների համատեղման 2 մոտեցումներ գոյություն ունեն: Որոնք են դրանք:

- ☐ մեկնարկման
- ☐ ներդրման
- ☐ կապակցման
- ☐ համակարգման

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N18

Աղյուսակի լայնությունը կարելի է տալ 2 եղանակով: Որոնք են դրանք:

- ☐ վերացական
- ☐ իրական
- ☐ **բացարձակ**
- ☐ պայմանական
- ☐ **հարաբերական**

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N19

Համաձայն նոր ստանդարտի աղյուսակի տողերը կարելի է բաժանել երեք անկախ տրամաբանական խմբերի: Որոնք են դրանք:

- ☐ tr
- ☐ **thead**
- ☐ td
- ☐ **tbody**
- ☐ th
- ☐ **tfoot**

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N20

Ֆրեյմերի օգտագործման ժամանակ <body> տեգը փոխարինվում է <frameset>,</frameset> կոնտեյներով, վերջիններս ի՞նչ էլեմենտներ կարող են ունենալ:

☐ **frame**

ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N21

Թվարկեք "target"ատրիբուտի հատուկ արժեքները:

- ☐ **_top**
- ☐ **_left**
- ☐ **_self**
- ☐ **_right**
- ☐ **_bottom**
- ☐ **_parent**
- ☐ **_middle**
- ☐ **_blank**
- ☐ **_derived**

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N22

META որոշիչները բաժանվում են 2 խմբի և առանձնանում են ատրիբուտներով, առաջինը name երկրորդը content, վերջինս name -ի փոխարեն պարունակում է ի՞նչ ատրիբուտ:

☐ **http-equiv**

ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N23

Այն դեպքերում երբ օգտվողի պատասխանները պետք է մշակվեն սերվերում <Form>տեգը ունի նվազագույնը երկու ատրիբուտներ, որոնք են դրանք;

- ☐ **action**
- ☐ **method**
- ☐ **get post**

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N24

Նրանցից մեկը ըստ հաղորդման եղանակի կարող է ընդունել երկու արժեքներ, Որոնք են դրանք:

- ☐ **select**
- ☐ **option**

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N25

Մենյուները ստեղծելու համար ի՞նչ կոնտեյներ էլեմենտ է օգտագործվում, և նրա ներսը յուրաքանչյուր կետ ի՞նչ տեգով է սահմանվում:

- ☐ **<script>**
- ☐ **</script>**

Ճիշտ պատասխանը գնահատվում է 1 բալ

Հարց N26

Ի՞նչ տեգերի օգնությամբ է սցենարը ներդրվում էջի վերնագրային մասում:

- ☐ օպերանդներ
- ☐ լիտերներ
- ☐ փոփոխականներ
- ☐ օպերատորներ

Ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N27

Սցենարում օգտագործվում են տվյալների երկու հիմնական տեսակներ: Որոնք են դրանք:

- ☐ երկար թվեր
- ☐ ամբողջ թվեր
- ☐ ֆիբոնաչիի թվեր
- ☐ սահող ստորակետով թվեր
- ☐ սինվոլային տողեր

- ☐ Արմսթրոնգի թվեր
- ☐ բուլյան փոփոխականներ

Ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N28

Javascript -ում օգտագործվում են չորս տիպի փոփոխականներ:
Որոնք են դրանք:

- ☐ "=="
- ☐ "+"
- ☐ "-"
- ☐ "!="
- ☐ "**"
- ☐ "/"
- ☐ ">"
- ☐ "<"
- ☐ "%"
- ☐ ">="
- ☐ "<="
- ☐ "++"
- ☐ "--"

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N29

Այս նշաններից որոնք են համեմատման օպերատորներ:

- | | | |
|-----------------------------|-------|----------|
| <input type="checkbox"/> 29 | if | continue |
| <input type="checkbox"/> 29 | break | while |
| <input type="checkbox"/> 29 | for | do |

Ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N30

Նշվածներից որոնք են ցիկլի օպերատորներ:

- | | | |
|-----------------------------|-------|----------|
| <input type="checkbox"/> 29 | if | continue |
| <input type="checkbox"/> 29 | break | while |

☐ 29

for

do

ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N31

Ինչ բանալիական բառերով է հայտարարվում զանգվածը:

☐ new array()

ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N32

սեփական տիպի օբյեկտները ի՞նչ բանալիական բառի օգնությամբ է ստեղծվում:

☐ անունը =new

կոնստրուկտորի_անվանումը[(արգումենտներ)]

☐ փոփոխականը =int ֆունկցիայի_անուն[(արգումենտներ)]

☐ MyArray =new Array()

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N33

Javascript -ում ներկառուցված են երեք տեսակի օբյեկտներ:Որոնք են դրանք:

☐ String

☐ Math

☐ Date

ճիշտ պատասխանը գնահատվում է 2 բալ

Հարց N34

Նշվածներից որոնք են String մեթոդի օբյեկտներ:

getHours()

getSeconds()

getYear()

toLocalString()

blink()

fixed

indexOf

getMinutes()

getTime()

getTimeZoneoffsetSet()

parse(Date)

anchor()

bold

fontcolor

italics

getMonth()

toGMTString()

big()

charAt()

fontsize

LastIndexOf

link	small	strike
sub	SubString	sup
toLowerCase	toUpperCase	
getHours()	getMinutes()	getMonth()
getSeconds()	getTime()	getTimeZoneoffsetSet()
getYear()	parse(Date)	toGMTString()
toLocalString()	anchor()	big()
blink()	bold	charAt()
fixed	fontcolor	fontsize
indexof	italics	LastIndexOf
link	small	strike
sub	SubString	sup
toLowerCase	toUpperCase	

Գիշտ պատասխանը գնահատվում է 3 բալ

Հարց N35

Նշվածներից որոնք են Math մեթոդի օբյեկտներ:

getHours()	getMinutes()	getMonth()
getSeconds()	getTime()	getTimeZoneoffsetSet()
getYear()	parse(Date)	toGMTString()
toLocalString()	anchor()	big()
blink()	bold	charAt()
fixed	fontcolor	fontsize
indexof	italics	LastIndexOf
link	small	strike
sub	SubString	sup
toLowerCase	toUpperCase	

Գիշտ պատասխանը գնահատվում է 3 բալ

Հարց N36

Նշվածներից որոնք են Date մեթոդի օբյեկտներ:

getHours()	getMinutes()	getMonth()
getSeconds()	getTime()	getTimeZoneoffsetSet()
getYear()	parse(Date)	toGMTString()
toLocalString()	anchor()	big()
blink()	bold	charAt()
fixed	fontcolor	fontsize
indexof	italics	LastIndexOf

link	small	strike
sub	SubString	sup
toLowerCase	toUpperCase	

Ճիշտ պատասխանը գնահատվում է 3 բալլ

Հարց N37

Նշվածներից որոնք են ներկառուցված ֆունկցիաներ:

- ☐ onfocus
- ☐ onblur
- ☐ onclick
- ☐ ondblclick
- ☐ onchange
- ☐ onkeydown
- ☐ onkeypress
- ☐ onkeyup
- ☐ onload
- ☐ onunload

Ճիշտ պատասխանը գնահատվում է 3 բալլ

Հարց N38

Նշված իրադարձություններից որոնք չեն վերաբերվում Input տիպի էլեմենտին

window	anchors	text
frames	aplets	textarea
document	embeds	
	password	
history	filters	
	fileupload	
navigator	images	radio
location	links	reset
event	plugins	submit
forms	scripts	hidden
element	selections	select
body	stylesheets	options
all	button	
	checkbox	

Ճիշտ պատասխանը գնահատվում է 3 բալլ

Հարց N39

Որո՞նք են նշվածներից բրաուզերի օբյեկտային մոդելի օբյեկտներ:

window	element	
	stylesheets	
frames	body	button
document	all	
	checkbox	
history	anchors	text
navigator	aplets	textarea
location	embeds	
	password	
event	filters	
	fileupload	
forms	images	radio
plugins	links	reset
scripts	hidden	submit
selections	select	options

Ճիշտ պատասխանը գնահատվում է 3 բալլ

Հարց N40

Որո՞նք են նշվածներից բրաուզերի օբյեկտային մոդելի հավաքածուներ:

alert	status	
	navigate	
prompt	defaultstatus	blur
confirm	returnValue	focus
parent	client	scroll
self	length	
	SetInterval	
top	opener	
	SetTimeout	

name	close
	ClearInterval
opener	ShowModalDialog
	ClearTimeOut
closed	showhelp
	ExecScript

Ճիշտ պատասխանը գնահատվում է 3 բալլ

Հարց N41

Որոնք են նշվածներից Window օբյեկտի հատկություններ

alert	defaultstatus	navigate
prompt	returnvalue	blur
confirm	client	focus
parent	length	scroll
self	opener	SetInterval
top	close	SetTimeout
name	ShowModalDialog	ClearInterval
opener	showhelp	ClearTimeOut
closed	status	ExecScript

Ճիշտ պատասխանը գնահատվում է 3 բալլ

Հարց N42

Որոնք են նշվածներից Window օբյեկտի մեթոդներ:

Ճիշտ պատասխանը գնահատվում է 3 բալլ

Հարց N43

Ինչո՞վ են տարբերվում սովորական և մոդալ պատուհանները

- ☐ չափով
- ☐ գույնով
- ☐ վերադարձվող արժեքի նշակմամբ
- ☐ պատուհանի դաշտերի մաքրմամբ

Ճիշտ պատասխանը գնահատվում է 3 բալլ

Հարց N44

Html էջերի բովանդակության և տեսքի ղեկավարման համար գոյություն ունեն 4 հատկություններ և երկու մեթոդներ: Այս հատկություններից որոնք են հատկություններ:

- ☐ absolut
- ☐ innerText
- ☐ overflow
- ☐ outerText
- ☐ visibility
- ☐ innerHTML
- ☐ outerHTML
- ☐ z-Index
- ☐ insertAdjacentText
- ☐ insertAdjacentHTML

Ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N45

Html էջերի բովանդակության և տեսքի ղեկավարման համար գոյություն ունեն 4 հատկություններ և երկու մեթոդներ: Այս հատկություններից որոնք են մեթոդներ:

- ☐ absolut
- ☐ innerText
- ☐ overflow
- ☐ outerText
- ☐ visibility
- ☐ innerHTML
- ☐ outerHTML
- ☐ z-Index
- ☐ insertAdjacentText
- ☐ insertAdjacentHTML

Ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N46

Ինչ տեգի օգնությամբ է ActiveX էլեմենտը տեղադրվում էջում

<object>

Ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N47

գոյություն ունի բազան էջի հետ կապելու երկու կապակցող էլեմենտ: որոնք են դրանք:

- ☐ STD (TDC)
- ☐ RDS

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N48

Նշեք գրանցումների տեղաշարժվելու համար օգտագործվող դեկլարացիայի recordset ենթաօբյեկտի մեթոդները ա) ազդարարում է ֆայլի վերջը eof, բ) ազդարարում է ֆայլի սկիզբը bof գ) անցում առային գրանցմանը , դ) անցում վերջին գրանցմանը

- ☐ eof
- ☐ bof
- ☐ moveFirst
- ☐ moveLast
- ☐ moveNext
- ☐ movePrevious

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N49

Թվարկեք հետևյալ հատկությունների և մեթոդների անունները

- ☐ recordcount
- ☐ AbsolutePosition
- ☐ fields
- ☐ AddNew()
- ☐ Delete()

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N50

ASP-ում նշվածներից որոնք են հանդիսանում օբյեկտներ

- ☐ Application
- ☐ Redirect
- ☐ clear
- ☐ Flush

- ☐ Write
- ☐ Request
- ☐ Response
- ☐ Session
- ☐ BrowserType
- ☐ server
- ☐ contentType
- ☐ Buffer

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N51

ASP ֆայլերում ծրագրային կոդի այն մասը, որը կատարվում է սերվերի վրա ի՞նչ զույգ տեգերի միջև է ամփոփվում

- ☐ <%
- ☐ %>

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N52

ASP ֆայլի սկզբում ընդգրկվող ֆայլերը ինչ հրահանգի միջոցով է իրականացվում

#include

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N53

Որոնք են հանդիսանում տվյալների բազաներին հասանելիության բաղկացուցիչներ(DAC)

- ☐ ADO
- ☐ OCDB
- ☐ ODBC
- ☐ ODA
- ☐ CBDO
- ☐ DBOC
- ☐ OLEDB
- ☐ LEBEDO

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N54

Նշվածներից որոնք են հանդիսանում server օբյեկտի հատկություն

- ☐ CreateObject()
- ☐ connection
- ☐ ScriptTimeout
- ☐ adOpenForwardOnly
- ☐ Execute
- ☐ GetLastError()
- ☐ HTMLEncode()
- ☐ MapPath()
- ☐ Transfer
- ☐ URLEncode()

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N55

Ոճերի աղյուսակում հատկությանը արժեքի վերագրումը ինչ սինվոլի օգնությամբ է իրականացվում

""

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N56

ոճերի աղյուսակում հատկությունները ցուցակում ինչ սինվոլի օգնությամբ են առանձնացում

,

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N57

Ինչպես են կոչվում ոճերի աղյուսակում թվարկած էլեմենտները ----
-, իսկ այն ամենը ինչ գրանցված է ձևավոր փակագծերում ինչպես է կոչվում

սելեկտորներ

սահմանումներ

ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N58

Ունիվերսալ դասի սահմանման ժամանակ, ինչ է դրվում դասի անունից առաջ

Ճիշտ պատասխանը գնահատվում է 3 բալ

Հարց N59

Քանի տեսակի են լինում պրիմիտիվները և որոնք են դրանք

տառանիշային

թվանիշային

Ճիշտ պատասխանը գնահատվում է 3 բալ