ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА

КАТЕДРА “КОМПЮТЪРНИ НАУКИ И ТЕХНОЛОГИИ”

доц. д-р Огнян Железов, гл. ас. д-р Бойка Градинарова, гл. ас. Даниела Илиева

WEB Дизайн

HTML, JavaScript

второ актуализирано издание

Увод

Създаването на HTML документи и WEB сайтове е дейност, която се базира на все по-ускореното развитието на World Wide Web (WWW) като световна компютърна мрежа. От професия за сравнително малко специалисти в тази област, WEB дизайн се превръща във вид компютърна грамотност за всеки софтуерен инженер. Настоящия учебник е разработен в съответствие с учебната програма по дисциплината “WEB дизайн” за специалност “Компютърни системи и технологии” на ТУ-Варна. Авторите се е старали да дават изчерпателно описание на разглежданите елементи, обекти и методи и примери за използването им , така, че за използването на учебника да не са необходими някакви начални познания по HTML и по програмиране.

Учебникът съдържа три глави. В първа глава се разглежда HTML като език за създаване на документи, които се публикуват в световната компютърна мрежа WWW (World Wide Web). Втора глава е посветена на JavaScript - компактен, обектно ориентиран скриптов език, който дава възможност за включване в HTML документите на програмен код, който се изпълнява от потребителския браузър. В трета глава, озаглавена “Динамичен HTML”, се разглеждат някои възможности, които предоставя обектният модел на документа за динамично променяне на вида и съдържанието на HTML документите.

Разработката на настоящият учебник е осъществена при следното разпределение между авторите: първа глава – доц. О.Железов и гл. ас. Д. Илиева, втора глава - доц. О.Железов и гл. ас. д-р Б. Градинарова, трета глава – доц. О. Железов.

Във второ издание са направени следните допълнения и корекции:

В глава първа: Разгледани са някои от новите елементи на формулярите, включени в HTML5.

В глава втора: Добавени са описания на декларации на масиви и обекти в литерален формат, допълнени са темите за функции,.използване на регулярни изрази, програмни таймери

В глава трета: Добавена е тема “Мултимедийни възможности на HTML5”.

Авторите изразяват благодарност на рецензентите доц. В. Смърков и доц. Н. Николов за ценните препоръки по оформянето на учебника.

СЪДЪРЖАНИЕ

Глава I – HTML

Глава I – HTML

1 Какво представлява HTML.

HTML (Hypertext Markup Language) е език за форматиране на хипертекстови документи, който използва маркиране (Mark-up). Основното му приложение е създаването на документи, които се публикуват в световната компютърна мрежа WWW (World Wide Web). Поради това HTML документите често биват наричани Web-документи или Web-страници.

2 Кратка история на HTML и WEB браузърите. Хипертекст.

HTML е създаден от Тим Бърнардс Лий (Tim Berners-Lee) Основната идея, вложена в него е идеята за хипертекста. Хипертекстът в един документ е маркирана част от текста, която съдържа препратка към друг документ. Поради това хипертекстът, който съдържа препратка към друг документ или към друга част от документа, се нарича още хипервръзка. Докато един обикновен текст представя информацията по напълно последователен начин, хипертекстът представя не само информацията, съдържаща се в текста на документа, но и тази, съдържаща се във всички, свързани с него документи. Движението по свързаните с хипервръзки документи се нарича навигация в хиперпространството.

Използването на HTML за създаване на хипертекстови документи е неразривно свързано със специализираните програми, наречени браузъри. Основните функции на браузърите са:

- Изпращането на заявки за документи, публикувани в WWW
- Получаването чрез WWW на заявения документ.
- Форматирането на документа в съответствие с HTML маркерите и извеждането на документа в прозореца на браузъра.

Навигацията в хиперпространството се извършва чрез функциите на браузъра. Когато потребителят активира хипервръзка, съдържаща препратка, браузърът изпраща заявка за съответния документ и го зарежда в прозореца на браузъра.

Първият WEB браузър, който е осъществявал изброените по-горе основни функции е браузърът Mosaic, разработен в NCSA (Национален център за суперкомпютърни приложения, САЩ, Илиноис) от Марк Ендрийсен (Marc Andreessen) и Ерик Байл (Eric Bila). Повече информация за създаването му можете да намерите на адрес http://www.livinginternet.com/w/wi_mosaic.htm.

Браузърът се оказва изключително полезна програма за работа в глобалната компютърна мрежа WWW, и поради това много фирми разработват и постоянно усъвършенстват подобни програми. Най-

популярните WEB браузъри в момента са Internet Explorer на Microsoft (само за операционна система Windows), Netscape на Netscape Communications Corporation и Opera на норвежката фирма Opera Software.

HTML е преминал през дълъг път на рзвитие от създаването на първия WEB браузър до наши дни. Основните движещи сили за това са на първо място бързото развитие на WWW и свързаното с това усъвършенстване на браузърите на конкуриращите се фирми, основно Microsoft и Netscape Communications. В крайна сметка логиката на развитие е наложила стандартизирането на езика. Организацията, която е създала препоръчителната спецификация на HTML, към която се придържат фирмите, разрабтващи браузъри е консорциумът W3C (World Wide Web Consortium). По време на написването на този учебник последната спецификация е за HTML 4.01. Тя е публикувана в сайта на организацията на адрес http://www.w3.org/TR/html4/.

3 Какво представлява маркирането(Mark-up)

Както е посочено в даденото в т.1 определение за HTML, той е език, който използва маркиране. Могат да бъдат посочени доста езици за създаване на документи, които използват маркирането [4]. Такива са например Rich Text Format, разработен от Microsoft, но поддържан от повечето текстови редактори, XML (eXtensible Markup Language), SGML (Standard Generalized Markup Language) и др.

Mark-up произлиза от печатната индустрия. Ръкописите са били допълвани с указания за форматирането при отпечатване, които се наричали маркери. Тук се вижда една характерна черта на маркирането – то е допълнителна дейност, която не влияе на съдържанието на документа, а на неговото представяне и интерпретиране. В зависимост от значението на маркерите за обработката на документа те могат да бъдат класифицирани на:

Процедурен Markup – При тоя тип маркиране маркерът се възприема като указание за изпълнение на някаква процедура от изходното устройство. Пример за процедурен маркер е контейнерния HTML етикет <B> Той е указание за програмата, която извежда HTML документа, че трябва да изведе текста, съдържащ се между отварящата част <B> и затварящата част </B> като удебелен.

Родов Markup – При тоя тип маркиране маркерът се възприема като указание за изпълнение на макрос от изходното устройство. Родовото кодиране дава по-голяма гъвкавост. За да се промени изобразяването на документа е достатъчно да се променят един или няколко макроса. Пример за език за описание на документи, който използва родово кодиране е T_cX.

Структурен Markup – При тоя тип маркиране маркерите описват структурата на документа, а не неговото форматиране при извеждане. Структурното описание улеснява много електронната обработка на документите, тъй като дава възможност за лесно добавяне, извличане и променяне на елементи от документа. Да разгледаме един пример:

```
<група>
<студент>
   <име>Иван</име>
   <презиме>Денев</презиме>
   <фамилия>Иванов</фамилия>
   <фак_номер>046630</фак_номер>
</студент>
<студент>
   <име>Десислава</име>
   <презиме>Милева</презиме>
   <фамилия>Димитрова</фамилия>
   <фак_номер>046632</фак_номер>
</студент>
</група>
```

Пример 1 Структурно маркиране

Структурното значение на маркерите <група>, <студент>, <име>, <презиме>, <фамилия> и <фак_номер> се вижда и без обяснения. Очевидни са също и възможностите за електронна обработка на документа. Например за да се изведат данните за всички студенти с фамилия “Иванов” е достатъчно да се проверява само съдържанието на маркера <фамилия> във всеки маркер <студент> и при съвпадение с търсената фамилия да се изведе в подходящ вид съдържанието на родителския маркер <студент>.

4 SGML и HTML.

Структурното кодиране е реализирано в най-чист вид в езика за описание на документи SGML (Standard Generalized Markup Language) – стандартизиран обобщен маркерен език. Този език е разработен по поръчка на министерството на отбраната на САЩ и понастоящем е приет като стандарт от международната организация за стандартизиране ISO. Универсалността на SGML се състои в това, че той не въвежда собствено множество от маркери (тагове), а дава възможност за различните документи или типове документи да се задава различен набор от тагове и връзките между тях. Структурата на документа е описана в Document Type Definition (DTD), най-често отделен файл, който се нарича SGML приложение. Спецификацията на файла, съдържащ DTD, се задава в тага <!DOCTYPE>, разположен в началото на документния файл.

HTML като език за описание на документи, с основно предназначение публикуване във WWW, е по същество SGML документ със стандартизиран набор от маркери (тагове). За да бъде структуриран един HTML документ по SGML стандарта в началото му трябва да бъде включен тага <!DOCTYPE>, в който да бъде указан DTD за HTML документ, например:

<!DOCTYPE html PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN">

Този таг на практика не е задължителен за HTML документите, защото Web браузърите не използват DTD когато обработват и изобразяват HTML документи. Ако Web браузърът срещне таг, който не може да разпознае, той просто не обработва неговото съдържание.

5 Структура на HTML документите.

5.1 Формат на HTML етикетите

HTML документите са документи в текстов формат, които освен текстово съдържание могат да включват определени маркери, които по-нататък ще наричаме етикети или тагове. Етикетите са инструкции с различно предназначение, напр. задаване на параметри на текста, включване на изображения, таблици, формуляри и др. Етикетите винаги започват с лява ъглова скоба (<) и завършват с дясна ъглова скоба (>). Името на етикета се поставя между тези два знака, като не се прави разлика между малки и големи букви в тях. Обикновено, но не винаги, етикетите имат отваряща и затваряща част, като ограждат частта от HTML документа, върху която се прилага действието на етикета. Такива етикети се наричат ограждащи или контейнери. Друг вид етикети са разделящите. Те не притежават затваряща част и се наричат още неконтейнерни. Чрез тях в HTML документа се вмъкват нови редове, изображения и хоризонтални линии.

<име_на_етикет атрибут1="стойност" атрибут2="стойност" .... > съдържание </ име_на_етикет>

Формат на обграждащ HTML етикет

Например етикетът <P align="center">Hello</P> е обграждащ HTML етикет за параграф, който има един атрибут, с който се задава центриране на текста. Съдържанието на етикета се намира между отварящата част <P align="center"> и затварящата част </P>.

<име_на_етикет атрибут1="стойност" атрибут2="стойност" .... >

Формат на разделящ HTML етикет

Например етикетът <HR width=”50%” color=”blue”> е разделящ етикет, който извежда хоризонтална линия с дължина 50% и син цвят.

5.2 Структура на HTML документ

Стандартният HTML документ се състои от две основни части: заглавна част на документа и тяло на документа.

```
<HTML>
<HEAD>

<!-- Заглавна част на HTML документа -->

</HEAD>
<BODY>

<!-- Тяло на HTML документа -->

</BODY>
</HTML>
```

Пример 2 Структура на HTML документ

5.3 Заглавна част на документа <HEAD>

В заглавната част на документа се дефинират свойствата, валидни за целия документ (например заглавието на документа, кой и кога го е създал или актуализирал и какви вътрешни хипервръзки съдържа документът с други Web документи). Главата на документа може да липсва - не е задължително HTML документът да притежава заглавна част.

Заглавната част на документа се задава с контейнерния етикет HEAD. Всички настройки, отнасящи се до документа се намират между отварящата част <HEAD> и затварящата част </HEAD>. Тук спадат например заглавието на документа, както и отнасящите се до този документ ключови думи, които се използват от машините за търсене в Интернет.

Етикети, които могат да се включват в заглавната част на HTML документа

1) <TITLE> .... </TITLE> Заглавие на документа – използва се от Web браузера или от друг Интернет софтуер за различни цели. Повечето Web браузери извеждат заглавието в горната лента на прозореца. Някои Web браузери използват заглавието и като Bookmark/отметка или когато визуализираната в момента Web страница трябва да се съхрани в списъка Favorites.

2) <META> .... </META> Допълнителни сведения за документа. Поради глобалната си валидност те се наричат META- данни и се предават на Web браузера чрез елемента META. META-елементът служи за предаване на допълнителни свойства на документа по пътя му към Web браузера, които се различават от обичайните настройки.

Всеки META-елемент се прилага с атрибутите name и content , при това винаги като двойка, състояща се от свойство и стойност, например **<META name=”Author” content=” Ivan Ivanov”>**

Атрибутът name е наименование на свойството. Няма ограничения за съдържанието на name. Могат да се задават както общоприети имена на атрибути (напр”Author”), както и такива, дефинирани от потребителя.

Атрибутът content е стойност на свойството. Така чрез атрибутите name и content се формират двойки име-стойност, които се задават чрез етикета <META>

<META scheme=схема>. Профил. Чрез атрибутът scheme Web браузера получава информация каква профилна схема да се приложи към този документ.

Друг вид на етикета <META> е **<META http-equiv=”наименование” content=”стойност”>**. Чрез този вид на етикета се предават на браузъра параметри, които се включват в заглавната част на HTTP протокола. Поради това като двойки наименование-стойност се задават такива, които браузърът и сървърът разпознават и отработват По-долу са дадени два полезни META етикета от този вид:

```
<meta http-equiv="pragma" content="no-cache">
```

Пример 3 – META етикет, който забранява кеширането на HTML документа.

По правило WEB сървърите съхраняват временно изпращаните HTML документи в буферна памет (кеш) и при повторно заявяване на същия документ го извличат от буферната памет а не от дисковото устройство. С това се ускорява обработката на заявките, но при променяне на съдържанието на HTML документа новата му версия става достъпна едва след като изтече времето за кеширане на старата версия. В случай че разработчикът иска актуализиране веднага, той трябва да включи горния етикет в заглавната част на коригирания HTML документ.

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
```

Пример 4 – META етикет, който задава типа на съдържанието и кодовата страница на HTML документа.

Извеждането на кирилица от WEB браузъра изисква задаването на съответната кодова страница. Пример 6 съдържа META етикет, който задава тип ="text/html и кодова страница 1251, необходима за правилно извеждане на текст на кирилица.

Чрез META данните могат да се предават ключови думи за машините за търсене По правило при търсене на HTML документи по зададен ключ търсещите програми прочитат само заглавната част на HTML документа и търсят ключовата дума в етикетите TITLE и META. С помощта на ключовите думи маркираните документи могат по-лесно да се регистрират и по-късно отново да се намират.

3) <STYLE …>дефиниции на стилове</STYLE> Чрез този етикет в заглавната част на HTML документа се включват дефиниции на стилове за форматиране на HTML елементи.
4) <SCRIPT …>код на JavaScript</ SCRIPT> Чрез този етикет в заглавната част на HTML документа се включва програмен код (по подразбиране на JavaScript), който (също по подразбиране) се изпълнява от Web браузъра.

5.4 Тяло на HTML документа. Етикет <BODY>

Тялото на документа съдържа видимото през Web браузера съдържание на документа или присъединяваната от framesets структурна информация. Съществуват два основни начина за оформяне на тялото на документа:

- оформяне на тялото на документа чрез контейнерния етикет < BODY>.

  Всички HTML елементи, вписани в контейнера <BODY> се анализират от Web браузера и се визуализират в прозореца.
- представяне чрез фреймове, зададени посредством етикетите <FRAMESET>.

  В тоя случай структурирането се извършва чрез рамки (фреймове), в които се определя структурата, външния вид и съдържанието на всеки отделен фрейм (вижте т. 14 “Фреймове (рамки) в прозореца на браузъра”)

5.5 Атрибути на етикета BODY

Контейнерният етикет BODY включва съдържанието на HTML документа, което се извежда в прозореца на браузъра. Атрибутите на етикета BODY задават различни глобални характеристики на документа.

Атрибути [1*]BGCOLOR, TEXT, LINK, ALINK и VLINK

Атрибутите задават цветове на различни елементи на HTML документа. фонов цвят на прозореца на браузъра при извеждане на страницата. Цветовете се кодират с шестнадесетични кодове или посредством наименованието им, както е описано в т.5. Значението на атрибутите е както следва:

<BODY bgcolor= фонов цвят>. Задава фоновия цвят на целия документ

<BODY text=цвят>. Задава цвят на текста.

<BODY link=цвят>. Цвят на хипервръзките. При повечето браузери подразбиращият се цвят на хипервръзките е син.

<BODY vlink=цвят>. Цвят на посетените хипервръзки. Подразбиращият се цвят при повечето браузери е виолетов.

<BODY alink=цвят>. Цвят на активираните хипервръзки. Ако потребителят щракне върху хипервръзка или тя бъде активирана по някакъв друг начин, браузърът оцветява текста в цвета alink.

```
<BODY BGCOLOR=”#FFFFF0”>
```

Пример 5 Задаване на фонов цвят на HTML документ

В примера цветът е зададен като шестнадесетично число с яркост FF16=255_{10} за червения и зеления компонент и яркост $F0_{16}$=240_{10} за синия цвят. Като резултат се получава бледожълт цвят на фона на HTML документа.

Атрибут BACKGROUND

Атрибутът задава фоново изображение, което се зарежда на страницата. Като стойност на атрибута се задава адрес на файл (пълен или относителен спрямо директорията в която е записан HTML файла), който съдържа фоновото изображение. Файлът може да бъде във формат[*] .jpg, .gif или .png.

```
<BODY BACKGROUND=”images/fon1.jpg”>
```

Пример 6 Задаване на фоново изображение на HTML документ

[*] В HTML етикетите не се прави разлика между малки и големи букви, така че в случая изписването на името на атрибутите с главни букви е само за по-добро разграничаване от останалия текст.

[*] Вижте т. 10.1 – Графични файлови формати, поддържани от браузърите

Даденият пример показва задаване на фоново изображение от файл, който се съдържа в поддиректория images на директорията, в която е записан HTML файла.

5.6 Задаване на цвят в HTML

Повечето HTML елементи притежават атрибути, чрез които се задава цвят на елементите. HTML предоставя две възможности за задаване на цвят на елементите – чрез числена стойност и чрез име на цвета.

5.6.1 Задаване на цвят посредством числена стойност

В тоя случай цветът се задава с число в шестнадесетична бройна система, чрез което се представя яркостта на трите основни съставящи на цвета – червен, зелен и син. Цветовете се кодират с шестнадесетичен код според т.нар. RGB схема, която е стандарт във Windows (R=RED, G=GREEN, B=BLUE) във формат #RRGGBB

```
<body bgcolor=#FFABD7>
```

Пример 7 Задаване на цвят на фона HTML документ чрез числена стойност.

Първите две цифри (FF) в кода са за червения цвят.

Средните две цифри (AB) в кода са за зеления цвят.

Последните две цифри (D7) в кода са за синия цвят.

Двуразрядните числа, с които се задава яркостта на всеки от трите основни цвята, се записват в шестнадесетична бройна система. Минималната им стойност е 00, а максималната – FF_{16} (което е 255 в десетична бройна система). По-долу е дадено кратко обяснение на преобразуването на тези числа в десетична бройна система. Това не е предмет на този курс, но все пак за WEB дизайнера е полезно да може оценява приблизително тези стойности.

Коефициентите във всеки разряд могат да имат стойност от 0 до 15, но тъй като трябва да се записват с един символ, за коефициентите със тойност над 9, за които няма цифри от десетичната бройна система, се използват буквите от латинската азбука A, B, C, D, E и F – съответно за 10, 11, 12, 13, 14, 15. За да преобразуваме шестнадесетично число в десетично трябва да умножим по правилата на десетичната аритметика стойността на всеки коефициент по 16, повдигнато на степен, равна на номера на разряда (най-младшият разряд е с номер 0). Така $FF_{16} = 15*16^1 + 15*16^0 = 255_{10}$. , а $D7_{16} = 13*16^1 + 7*16^0 = 215$.

Специализираните програми за създаване на HTML документи - HTML редакторите – предоставят възможност за избор на цвят от палитра или от друг елемент и записват в HTML кода числовата стойност, съответстваща на избрания цвят. Така че на Web дизайнера по правило не му се налага да задава цветовете чрез въвеждане на шестнадесетичните им кодове.

5.6.2 Задаване на цвят посредством наименованието му.

За да се улесни Web дизайнера, в HTML 3.2 е предвидена възможност за задаване на 16 основни цвята посредством наименованието им. По-долу са дадени наименованията на тези цветове, които (както можеше и да се очаква) са на английски език.

aqua	black	blue	gray
fuchsia	green	lime	maroon
navy	olive	purple	red
silver	teal	white	yellow

В следващите версии на HTML са добавени още имена на цветове, повечето от който са производни на горните с представка light за по-светъл или dark за по тъмен, например lightgreen за светлозелен и darkgreen за тъмнозелен цвят.

```
<body bgcolor = "yellow">
```

Пример 8 Задаване на фонов цвят на HTML документ чрез име на цвета

6 Каскадни стилове CSS

Каскадните стилове (Cascading Style Sheets, съкратено CSS) дават възможност за по-пълен контрол на вида на документа (шрифтове, цветове, интервали, размери. връзки). Могат да се посочат много ограничения на класическия HTML, които са преодолени с въвеждането на стиловете. Например без използването на стил не може да се укаже точен размер в пиксели на символите от текста, както това се прави в текстообработващите програми. Без прилагане на стил текстовите бутони са винаги със сив фон, а хипервръзките – винаги с подчертан текст. Когато видите HTML документ, в който текстовите кутии не са с бял фон и черен текст, можете да бъдете сигурни, че това е направено с прилагане на стил.

6.1 Дефиниране на стил

Дефинициите на стил във външен файл (а също и в заглавната част на HTML документа) трябва да спазват следния синтаксис:

селектор {атрибут:стойност;атрибут:стойност; …}

където:

селектор - определя HTML елемента, към който ще се прилага стила. Може вместо един елемент да съдържа списък от елементи, разделени със запетая. Стил за елемент със зададен с идентификатор се задава със символа # преди идентификатора. Например стила #p1 {color:blue} ще се приложи за елемент с атрибут id=”p1” например Hello .

атрибут - определя атрибута, чиято стойност се дефинира

стойност – задава стойността, която се присвоява на атрибута. Може да бъде зададен списък от стойности, разделени със запетая.

Задължителен разделител между атрибута и стойността е двоеточие “**:**”. Задължителен разделител между две двойки атрибут-стойност е символа точка и запетая “**;**”.

6.2 Селектор за елемент по зададен идентификатор

Селектор за елемент със зададен идентификатор , т.е. атрибут id със зададена стойност се дефинира със стойността на идентификатора, предшествана от символа “#” (шарп). Например ако имаме елемент Hello то дефиницията на стил за цвят и размер на текста за този елемент ще има вида:

```
#p1 {color:blue;font-size:14px}
```

CSS допуска задаване на списък от селектори в дефиниция на стил. Например за син цвят и центриране на текста на заглавни надписи с размер H1 и H2 и елемент с id=”p2”, можем да зададем следната дефиниция:

```
H1, H2, #p2 {text-align:center;color:blue}
```

Пример 9 Задаване на списък от селектори в дефиниция на стил.

6.3 Дефиниране на класове в CSS

Класовете в CSS дават възможност да се дефинира стил, който да не е свързан с определен HTML етикет или идентификатор Класът се дефинира

с уникално (т.е. неповтарящо се) име и може да бъде приложен към всеки етикет чрез атрибута class.Дефиницията на клас има следния синтаксис:

.име_на_клас {атрибут:стойност;атрибут:стойност; …}

където:

име_на_клас – задава уникално име на класа.

атрибут - определя атрибута, чиято стойност се дефинира

стойност – задава стойността, която се присвоява на атрибута. Може да бъде зададен списък от стойности, разделени със запетая.

Да разгледаме един пример за дефиниране и прилагане на клас.

```
<HTML><HEAD><STYLE>
H2 {color:blue}
.p14 {font-size:14pt}
</STYLE></HEAD><BODY>
<H2> Този текст е с размер H2 </H2>
<H2 class=”p14”> Този текст е с размер 14 </H2>
</BODY></HTML>
```

Пример 10 Деклариране на клас и прилагането му към HTML етикет чрез атрибут class

В този пример декларацията на класа е включена в заглавната част на HTML документ чрез етикет <STYLE>. При зареждане на документа в прозореца на браузъра ще се появи текст с размер H2 (заглавен текст), и текст с размер 14 пиксела. Разликата ще се прояви при променяне на подразбиращият се размер на текста чрез функцията на браузъра (от менюто View – Text Size за Internet Explorer). Ако променим например подразбиращият се размер от “Medium” на “Largest”, то тогава първият текст ще увеличи размера си, но втория, чийто размер е зададен чрез стил, ще отстане 14 пиксела. Стилът е единственият начин да зададем точен размер на текста в HTML.

6.4 Селектори по стойност на атрибут

Селекторът на CSS по стойности на атрибути предоставят ефективен начин за стилизиране на HTML елементи, като един такъв селектор може да замести цяла група прости селектори. По-долу е даден кратък списък с атрубутни селектори и тяхното значение:

[attribute]	селектира елементи, които имат зададен атрибут

[attribute="value"]	селектира елементи, които имат зададен атрибут със зададена стойност
[attribute~="value"]	селектира елементи, които имат зададен атрибут със стойност, която съдържа думата "value"
[attribute|="value"]	селектира елементи, които имат зададен атрибут със стойност value", след която съдържа символа "-" (тире)
[attribute^="value"]	селектира елементи, които имат зададен атрибут със стойност, която започва с низа "value"
[attribute$="value"]	селектира елементи, които имат зададен атрибут със стойност, която завършва с низа "value"
[attribute*="value"]	селектира елементи, които имат зададен атрибут със стойност, която съдържа стойност "value"

Да разгледаме един пример:

```
a[href$=".pdf"] {
  text-decoration-line: overline underline;
  text-decoration-style: wavy;
}
```

Пример 11 Декларация на стил, който се прилага за елементи с атрибут href, чиято стойност завършва с низа ".pdf"

6.5 Наследяване и предефиниране на стилове

Ако за един селектор са дефинирани стойности на няколко атрибута и след това с друга дефиниция се зададат нови стойности на част от тях, това означава, че старите стойности се предефинират (и вече не са валидни) а стойностите, които не са дефинирани повторно, се наследяват. Полезно е да запомните следното просто правило – при каскадните стилове за всеки атрибут е валидна последната дефиниция. Например ако след дефиницията от предния пример включим дефиниция само за етикета H2, с която се задава червен цвят на текста,

```
H2 {color:red}
```

Пример 12 Предефиниране на стойността на атрибут color за етикет H2.

то като резултат текстовете в елементи H1 и H2 и в елемент с id=”p2” ще бъдат центрирани, текстът в елементи с етикет H1 и с id=”p2” ще бъде със син цвят, а текстът с размер H2 ще бъде с червен цвят. В това се проявява каскадността на каскадните стилове (CSS) – всяка следваща дефиниция може да предефинира всяка от предидущите, а тези, които не са

предефинирани се наследяват. Трябва също да се има в предвид, че стойностите на атрибути, зададени със стил имат приоритет пред стойностите, зададени по “класическия” начин.

Приоритет на селекторите

В CSS се прилага следния приоритет на селекторите: най-висок приоритет имат селекторите, зададени за идентификатор, следвани от тези за клас, и най-нисък - тези за етикет (таг). На първо място обаче приоритетите на стиловете зависят от това, къде са включени в HTML документа.

6.6 Включване и прилагане на стилове в HTML документ.

WEB дизайнерът има три възможности за включване на дефиниции на стилове в HTML документите:

- включване в документа на стилове, дефинирани във външен файл
- Дефиниране на стилове в етикет <STYLE> в заглавната част на документа
- включване на атрибут style в конкретен етикет от документа.

Ще разгледаме последователно особеностите на всеки от тези три вида включване на стил в HTMLдокумент.

6.6.1 Включване на стилове от външен файл – етикет <LINK>

Етикет <LINK> дава възможност на дизайнера да създаде външен файл, който да съдържа декларации на стилове и да го свърже с един или повече HTML документи. Декларациите, включени в HTML документ чрез етикет <LINK> имат най-ниското трето ниво на приоритет в CSS и поради това могат да бъдат надписвани (заменяни) от декларации, включени чрез етикет <STYLE> (с второ ниво) и тези, включени с атрибут style (с първо ниво). Синтаксисът на етикет <LINK> в тоя случай трябва да има вида:

<link href="адрес_на_файла" rel="stylesheet" type="text/css">

където:

адрес_на_файла е пълен или относителен интернет адрес на файл (URL), който съдържа декларации на стилове. Например, ако искаме да получим същият резултат, както в пример 33, можем да запишем във файл MyStyle.css един текстов ред с декларацията:

```
H2 {color:blue}
.p14 {font-size:14pt}
```

Пример 13 Съдържание на файл **MyStyle.css** с декларации на стилове.

и да свържем HTML документ с този файл посредством етикета <LINK>

```
<HTML>
<HEAD>
<link href="Test_CSS.css" rel="stylesheet" type="text/css">
</HEAD>
<BODY>
<H2> Този текст е с размер H2 </H2>
<H2 class=”p14”> Този текст е с размер 14 </H2>
</BODY>
</HTML>
```

Пример 14 Включване на стилове от външен файл посредством етикет <LINK>

Ползата от създаването на външен файл с декларации на стилове е очевидна. Дизайнерът може да декларира всички необходими стилове за сайта си в един или няколко файла и да свърже всички страници от сайта с тях. Така сайтът ще има еднакъв вид на елементите на всички страници, а при необходимост от корекция на някоя декларация, ще се променя само един файл, а не всички HTML документи, в които той се прилага.

6.6.2 Включване на стилове от външен файл – директива @IMPORT

Втората възможност за включване на дефиниции на стилове от външен файл в HTML документ е използването на директивата @import. Тя се включва в контейнерния етикет <STYLE>, който от своя страна, който от своя страна трябва да бъде включен в заглавната част на HTML документа. Да разгледаме един пример, аналогичен на предходния, при който вместо етикет <LINK> се използва директива @IMPORT:

```
<HTML><HEAD>
<style type="text/css">
<!--
@import url("Test_CSS.css");
-->
</style>
```

```
</HEAD>
<BODY>
<H2> Този текст е с размер H2 </H2>
<H2 class=”p14”> Този текст е с размер 14 </H2>
</BODY></HTML>
```

Пример 15 Включване на стилове от външен файл посредством директива @IMPORT

Резултатът ще бъде същия, както в предния пример. Обърнете внимание на това, че съдържанието на етикета <STYLE> е включено в HTML маркери за коментар <!-- … -->. Това е направено за да се избегне изобразяването на включеното между отварящата и затварящата част на <STYLE> като HTML код. Предполагаме, че вашият браузър е съвременна версия и отработва етикета <STYLE>, но лесно можете с него да имитирате какво би се получило ако не го отработваше. Просто изтрийте от кода на HTML документа редовете, съдържащи <style type="text/css"> и </style>. Като резултат ще се изчезне форматирането, зададено с декларациите във файла Test_CSS.css. Ако обаче изтриете и редовете, съдържащи маркерите за коментар “<!—“ и “-->”., тогава в прозореца на браузъра ще се появи текста “@import url("Test_CSS.css");”.което очевидно е нежелателно. Поради това включването на съдържанието на етикета <STYLE> в маркерите за коментар е полезно, и това добрите HTML редактори като Dreamweaver го правят автоматично.

6.6.3 Включване на декларации на стилове в заглавната част на HTML документа – етикет <STYLE>

Ние вече разгледахме използването на етикет <STYLE> като контейнер на директивата @IMPORT, чрез която в документа се включват стилове, декларирани във външен файл. Основното приложение на етикета <STYLE> обаче е включването на декларации на стилове непосредствено в заглавната част на HTML документа. По този начин те се съдържат в кода на документа и естествено са валидни само за него. Декларациите, включени в HTML документ чрез етикет <STYLE> имат второ ниво на приоритет и следователно могат да надписват (да заменят) декларациите с трето ниво на приоритет, включени чрез етикет <LINK>. Етикета p> има само един атрибут, който е от значение за WEB дизайнера:

Атрибут:

type=”тип_на_съдържанието - задава езика за описание на стиловете, който е използван във файла. Този атрибут няма подразбираща се стойност,

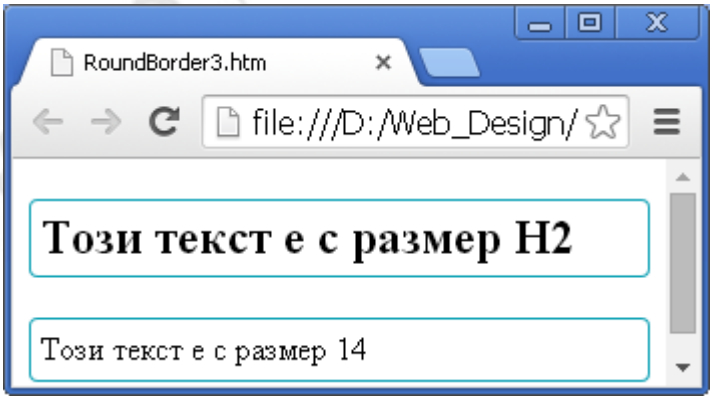
поради което се препоръчва да се включва type="text/css" в етикета <STYLE>. Добрите HTML редактори като Deamweaver включват автоматично type="text/css" при използване на етикета <STYLE>

Да разгледаме един пример:

```
<HTML><HEAD>
<style type="text/css">
H2,p {
border: 1px solid #0ba1b5; //Задаване на дебелина, вид и цвят на рамката
-moz-border-radius: 4px;    //Задаване на закръгление с радиус 4 пиксела
-webkit-border-radius: 4px;
border-radius: 4px;
padding: 5px 5px 5px 5px;}
</style>
</HEAD>
<BODY>
<H2> Този текст е с размер H2 </H2>
<p class=”p14”> Този текст е с размер 14 </p>
</BODY></HTML>
```

Пример 16 Включване на декларации на стилове в заглавната част на документа посредством етикет <STYLE>.

Декларацията задава жълт фон[2] на текста, включен в етикети H2. В случая в документа няма включена декларация за клас p14, и поради това и двата реда текст ще бъдат с еднакъв размер – използването на недекларирана клас не води до генериране на грешка от браузъра.



Фиг. 1 Задаване на рамка на текст посредством декларация на стил, включена в етикет <STYLE>

[2] Вижте т. 5 - задаването на цвят в HTML посредством числена стойност.

Допуска се включването в етикета <STYLE> едновременно на директиви @IMPORT и декларации, но директивите трябва да предхождат декларациите, както е показано в следващия пример:

```
<HTML>
<HEAD>
<style type="text/css">
<!--
@import url("Test_CSS.css");
H2 {background-color:#FFFF00}
-->
</style>
</HEAD>
<BODY>
<H2> Този текст е с размер H2 </H2>
<H2 class=”p14”> Този текст е с размер 14 </H2>
</BODY></HTML>
```

Пример 17 Включване на директива @IMPORT и декларация на стил в един и същ етикет <STYLE>.

В тоя случай ще се приложат както стиловете, декларирани във файла Test_CSS.css, така и декларацията, включена в документа. В резултат двата реда текст ще бъдат с различен размер на символите, както и в предните примери 14 и 15, но с жълт цвят на фона, както е зададено в декларацията.

6.6.4 Включване на декларация на стил в HTML етикет – атрибут style

HTML дава възможност на Web дизайнера да включи декларация на стил непосредствено в HTML етикети посредством атрибут style. В тоя случай стилът се прилага само за елемента или елементите, включени в HTML етикета, който съдържа декларацията. Синтаксисът малко се различава от декларациите на стилове във външен файл и етикет <STYLE>:

```
<име_на_етикет style=”атрибут:стойност;атрибут:стойност; …”>
```

Разликата в сравнение с дефинирането на стил в заглавната част или във външен файл е, че в тоя случай липсва селектор, а атрибутът style получава като стойност символен низ, съдържащ двойките атрибут-стойност на стил.

Да разгледаме един пример:

```
<HTML>
<BODY>
<H2 style=”background-color:#FFFF00”> Този текст е с размер H2 </H2>
<H2 style =” background-color:#FFFF00;font-size:14pt”> Този текст е с размер 14 </H2>
</BODY>
</HTML>
```

Пример 18 - Включване на дефиниции на стил в HTML етикети

Резултатът ще бъде същия, както за пример 37, но при използване на атрибута style декларациите трябва да се включват във всеки етикет поотделно, което е трудоемко за реализация и трудно за променяне в последствие. Затова задаването на стил с атрибут style (inline styles) има смисъл да се използва само тогава, когато се задава стил, който е наистина специфичен само за един конкретен елемент.

6.7 Приоритет на CSS декларациите. Атрибут !important

Декларациите, включени в HTML етикет чрез атрибут <STYLE> имат най-висок приоритет и следователно могат да надписват (да заменят) декларациите с трето ниво на приоритет, включени чрез етикет <LINK> и декларациите с второ ниво на приоритет, включени в заглавната част на документа чрез етикет. <STYLE>. За декларациите, зададени на едно ниво на приоритет, са валидни приоритетите за селектори, дадени в т.7.3. Съществува обаче възможност да се дефинира стил, който да има приоритет, по-голям от този на inline стиловете. Това става с атрибута !important. Да разгледаме един пример:

```
<HTML>
<HEAD>
<style type="text/css">
    #hello {color:red !important}
</style>
</HEAD>
<BODY>
<p id="hello" style="color:blue">Hello, CSS World</p>
</BODY>
</HTML>
```

В примера към елемента <p> се прилагат два стила - единият е дефиниран в блок <style> , а другият е вложен в самия елемент <p> чрез атрибут style. Ако в дефиницията в блока <style> параметърът !important липсваше, вложеният inline стил щеше да има по-висок приоритет и текстът в параграфа щеше да се изведе със син цвят. С атрибута !important обаче първият стил получава най-висок приоритет и текстът ще се изведе с червен цвят.

Online информация за кастадните стилове:
http://www.w3.org/TR/html401/present/styles.html
http://www.htmlcodetutorial.com/style/style.html

6.8 Управление на извеждането на HTML елементите – стил display

CSS стилът display определя как се позиционира и оразмерява даден елемент спрямо останалите елементи. Web браузърът извежда HTML елементите по две схеми на позициониране и оразмеряване: като вградени (inline) елементи и като блокови елементи. Разликата между двете схеми е, че вграденият елемент се извежда непосредствено след предходният елемент – той не заема самостоятелна област в прозореца на браузъра. Блоковият елемент заема самостоятелна област в елемента-контейнер, в който е вложен, като чрез стилове може да се управлява размерът и позицията на тази област

Свойството display приема много различни стойности като inline, inline-block, block, table и други, които влияят върху оформлението и представянето на даден елемент на Web страницата. Освен това, чрез стил display със стойности flex, grid и table могат да се позиционират група елементи като мрежа или таблица. По-долу е дадено значението на някои от по-честоизползваните стойности на CSS display:

inline	Показва елемент като вграден елемент (като). Свойствата за височина и ширина не не могат да се управляват.
block	Показва елемент като блоков елемент (като <p>). Започва на нов ред и заема цялата ширина на контейнера
contents	Скрива контейнера, правейки дъщерните елементи дъщерни на елемента на следващото ниво нагоре в DOM
flex	Показва елемент като гъвкав контейнер на ниво блок (block-level flex container)
grid	Показва елемент като мрежов контейнер на ниво блок (block-level grid container)
inline-	Показва елемент като вграден (inline) блоков контейнер. Самият

block	елемент е форматиран като вграден елемент, но могат да приложат стойности за височина и ширина.
inline-flex	Показва елемент като гъвкав вграден контейнер.
inline-grid	Показва елемент като гъвкав вграден мрежов контейнер
none	Елементът се премахва напълно

Даденият по-долу пример семонстрира разликата между вграден (inline) и блоков елемент:

```
<!DOCTYPE html>
<html><head>
<style>
#inline,#block {
  width:100px; height:100px;
  text-align:center;
  border: 1px solid blue;
  margin: 2px;
}
#inline {display: inline;}
#block  {display: block; }
</style>
</head>
<body>
<span id="inline">Inline span</span>
<span id="block">Block span</span>
</body></html>
```

Пример 19 Задаване на стилове за вграден и блоков елемент

HTML страницата в примера съдържа два елемента , за единият от които е зададен стил „display: inline;”, а за другият – стил „display: block;”. В резултат на това, въпреки че и на двата елемента са зададени размери „width:100px; height:100px;” само блоковият елемент получава този размер. Размерът на вградените (inline) елементи

7 Слоеве в HTML документите.

Слоевете са въведени в HTML за пръв път в Netscape Navigator (NE) 4.0. Слоят дефинира правоъгълна област (контейнер), в които може да се извежда всяко HTML съдържание (включително и вложени слоеве). Netscape Corporation и Microsoft създават различни етикети за слоеве.

Netscape създава етикета <LAYER>, а Microsoft създава етикета <DIV>. Необходимостта от стандартизация изисква да се приеме единия от двата етикета и в крайна сметка печели Microsoft – W3C стандартизира етикета <DIV>.

7.1 Създаване на слой в HTML документ – етикети <DIV> и <SPAN>

Както беше посочено по-горе, контейнерният етикет <DIV> е стандартизиран от W3C, така, че понастоящем той се всички браузъри. Етикетът <SPAN> е просто вариант на <DIV>. Разликата между двата етикета се проявява само ако слоя, който се създава чрез тях, се разполага вътре в останалото съдържание на документа "inline" без да се позиционира със стил. Тогава слоят, създаден с етикета <DIV> се разполага като параграф – с преминаване на нов ред преди началото и след края на слоя и с въвеждане на разстояние между слоя и останалото съдържание. Слоят, създаден с етикета <SPAN> се разполага последователно с останалото съдържание на документа и без въвеждане на допълнително разстояние. Етикетите<DIV> и <SPAN> предоставят набор от атрибути, който дават възможност за задаване на позицията, размерите и съдържанието на слоя.

Основни атрибути:

id	="идентификатор" – Задава идентификатор, чрез който се извършва програмен достъп до слоя.
style	="декларация на стил" – Задава декларация на стил за форматиране на слоя.
class	="име_на_клас" – Задава име на CSS клас за форматиране на слоя.
title	="подсказка" – Задава подсказка (tooltip) – малко прозорче със зададения текст, което се извежда, когато курсора на мишката се задържи над слоя.

Използването на CSS за форматиране на съдържанието на слой не се различава от използването на CSS за фоматирането на други HTML елементи, както е описано в т. 6 "Каскадни стилове CSS". Да разгледаме един пример:

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
```

```
<style type="text/css">
<!--
#div1, H1, .blue_div {background-color:blue;font-size:16}
-->
</style>
</HEAD>
<BODY>
<DIV id=”div1”>Първи слой със син фон </DIV><BR>
<H1 id=”div1”>Заглавен текст със син фон </H1>
<DIV id=”div2” class=”blue_div ”>Втори слой със син фон </DIV>
</BODY></HTML>
```

Пример 20 Задаване на стил на слой чрез декларация за идентификатор и чрез CSS клас.

В примера в блока <STYLE> е включена декларация на стил за идентификатора div1 на първия слой, за HTML етикет <H1> и за клас blue_div. В тялото на HTML документа са включени слой с идентификатор id=”div1”, заглавен текст в етикет <H1> и слой с идентификатор id=”div2.При извеждане и трите елемента имат син фонов цвят и размер на шрифта 16.

Етикетът <DIV> притежава следните атрибути – манипулатори на събития:

onClick, onDblClick, onMouseDown, onMouseUp, onMouseOver, onMouseMove, onMouseOut, onKeyPress, onKeyDown, onKeyUp

Манипулаторите на събития дават възможност за изпълнение на код на JavaScript при възникването на определени събития. Значението на дадените манипулатори на събития и използването им са разгледани във втора глава.

7.2 Позициониране на съдържанието на слой – стилове justify-content и align-items

Съдържанието на слой може да се позиционира в хоризонтална посока чрез CSS стила justify-content. Очевидно това има смисъл тогава, когато общата ширина на съдържанието е по-малка от ширината на слоя. На стила justify-content могат да се задават следните стойности:

flex-start	Стойност по подразбиране. Елементите се позиционират в началото (т.е. вляво) на контейнера
flex-end	Елементите се позиционират в края (т.е. вдясно) на контейнера

center	Елементите се позиционират в средата на контейнера
space-between	Елементите се позиционират с равно разстояние между тях като първият е плътно вляво, а последният – плътно вдясно
space-around	Елементите се позиционират с равно разстояние между тях и между контейнера
inherit	Наследява стойността на стила от контейнера

Съдържанието на слой може да се позиционира във вертикална посока чрез CSS стила align-items. Аналогично на хоризонталното позиционирана това очевидно има смисъл тогава, когато общата височина на съдържанието е по-малка от височината на слоя. На стила align-items могат да се задават следните стойности:

normal	По подразбиране. Държи се като „stretch“ за flexbox и мрежови (grid) елементи или „start“ за мрежови елементи с определен размер на блока.
stretch	Елементите се разтягат до размерът на контейнера
center	Елементите се позиционират в средата (вертикално) на контейнера
start	Елементите се позиционират в началото (т.е. в горната част) на контейнера
end	Елементите се позиционират в началото (т.е. в долната част) на контейнера
baseline	Елементите са разположени в основата на текста в контейнера
inherit	Наследява стойността на стила от контейнера

Да разгледаме един пример:

```
<!DOCTYPE html>
<html><head>
<style>
#container,#child {
  display: flex;
  justify-content: center; /* центриране хоризонтално */
  align-items: center; /* центриране вертикално */
}
#container {
```

```
  width:300px; height:300px;
  background-color: lightgreen;
}
#child {
  width: 100px; height: 100px;
  background-color:coral;
}
</style>
</head>
<body>

<h1>justify-content & align-items properties</h1>

<div id="container">
<div id="child">
<span>Content</span>
</div></div>
</body></html>
```

Страницата в примера съдържа слой "container" и вложен в него слой "child", който съдържа блок с текст в него. На слоевете са зададени стилове "display: flex;" , "justify-content: center;" и "align-items: center;". Като резултат слоят "child" се центрира хоризонтално и вертикално в слоят "container", а блокът се центрира се центрира хоризонтално и вертикално в слоят"child".

8 Форматиране на текст

Текстът е най-информативната част от HTML документите. За добрия външен вид на HTML документа е особено важно да се форматира по подходящ начин неговото текстово съдържание. За форматиране на текст най-често се използват каскадните стилове (CSS), които се разглеждат в т.6. Тук се представят възможностите, които предоставя HTML за форматиране на текст без използване на стилове.

При форматиране на текст е важно да се подберат подходящи цветове на текста и фона. В Интернет могат да се видят HTML страници с неподходящо подбрано съотношение от цвят на фон и цвят на текст, което ги прави почти неразбираеми за потребителя. Други често срещани грешки при форматирането на текст са:

- Прекалено малкия размер на символите, който, ако е зададен със стил, не може да се променя с опциите на браузъра (View ->Text Size за

Internet Explorer) без мащабиране на страницата. Това може да направи страницата неразбираема за някои потребители, въпреки, че според разработчика всичко изглежда добре.

- Задаване на вид шрифт, който не се включва по подразбиране при инсталиране на операционната система на потребителския компютър. Например при страници с текст на кирилица и неподходящо зададен шрифт, потребителят може да види неразбираем низ отсимволи.

При въвеждане на текст в HTML документ трябва да се помни това, че браузърът по подразбиране не извежда текста така, както правят това текстообработващите програми. По подразбиране не се отработват символите за преминаване на нов ред (които се въвеждат с натискане на бутона **Enter**), не се отработват символите за табулация а последователните празни интервали се извеждат като един. Поради това, въпреки че по принцип HTML документи могат да се създават и с обикновен текстов редактор, е по-добре да се използват специализирани HTML редактори като Dreamweaver, които при въвеждане автоматично заместват "специалните" символи с подходящи етикети или кодови комбинации. Например при натискане на бутона **Enter** програмата Dreamweaver вместо символ с код 13 (CR) въвежда в HTML документа етикет <P> за начало на параграф.

8.1 Форматиране на заглавен текст

Посредством контейнерните етикети <H1>,<H2>,<H3>,<H4>,<H5> и <H6>. в HTML се задават шест размера на символи за заглавен текст. Етикетът има атрибут ALIGN, който дава възможност за подравняване на текста вляво (left), вдясно (right) и центриран (center).

<HX align=left/center/right>. Подравняване на заглавието

където X може да бъде съответно 1, 2, 3, 4, 5 или 6. С най-голям размер се извежда текстът, включен в етикет <H1>, а с най-малък – текстът, включен в етикет <H6>.На практика етикетите се използват просто като начин за променяне на размера на какъв да е (не обязателно заглавен) текст, така, както това се прави и с етикета <FONT>.

```
<html>
<head>
<title>моята Web страница</title>
</head>
<body >
<h1 align="center">заглавие 1</h1>
<h2>заглавие 2</h2>
<h3>заглавие 3</h3>
```

```
<h4>заглавие 4</h4>
<h5>заглавие 5</h5>
<h6>заглавие 6</h6>
</body>
</html>
```

Пример 21 Задаване на размер на заглавен текст посредством етикети H1-H6

Атрибути на етикетите <H1> … <H6>

align – задава подравняване на текста. Допустимите стойности са “left”, “right” и “center” съответно за ляво, дясно и центрирано подравняване.

8.2 Параграф <P>

Етикетът за параграф <P> по принцип е контейнерен, което означава, че има затваряща част </P>. Той предизвиква включване на един празен ред преди <P>, започване на текста от началото на следващия ред и включване на един празен ред след затварящата част </P>. Съвременните браузъри са доста “търпеливи” към грешки в структурата на HTML документите. Неправилните етикети просто не се отработват, но не предизвикват извеждане на съобщения за грешка. Така ако пропуснете затварящата част </P> на етикета за параграф, нищо лошо няма да се случи.

Атрибути:

align – задава подравняване на текста. Допустимите стойности са “left”, “right” и “center” съответно за ляво, дясно и центрирано подравняване.

```
<html>
<head>
<title>моята Web страница</title>
</head>
<body>
<p>погледни</p>
<p>тези</p>
<p>параграфи</p>
</body>
</html>
```

Пример 22 Използване на етикета за параграф <P>

8.3 Нов ред

В някои случаи е необходимо текстът да премине на нов ред без да се вмъква разстояние до предходния текст. В такива случаи се използва

етикетът
. Етикетът за нов ред
 не е контейнерен, което означава, че за разлика от <P> той няма затваряща част.

Атрибути:

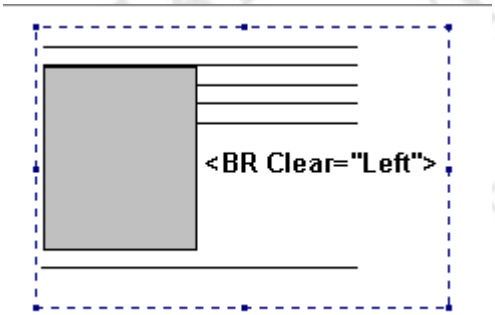
Етикетът
 има един специфичен атрибут – Clear с възможни стойности “Left”, “Right” и “All”. Атрибутът Clear задава начина на преминаване на следващ ред в случай че текста “обтича” правоъгълна област в HTML документа, която е отделена за извеждане на съдържанието на зададен обект (най-често това е изображение, включено с етикет <IMG>). Важно е да се отбележи, че за да има резултат от атрибута **Clear** изображението трябва да бъде подравнено с атрубут **align**. Значението на стойностите на атрибута **Clear** е следното:

“left”- следващият ред започва от позиция, непосредствено до лявата рамка на прозореца на документа.

“right” - следващият ред започва от позиция, от която може да завърши до дясната рамка на прозореца на документа.

“all” - следващият ред започва от тази позиция, непосредствено до лявата рамка на прозореца на документа, от която може да завърши до дясната рамка на прозореца на документа, т.е. стойността “all” извършва това, което би се получило ако можеше да се приложат “Left” и “Right” едновременно.

Да поясним това с пример:



Фигура 2 Позициониране на следващия ред на текста след етикет <BR Clear=”Left”>

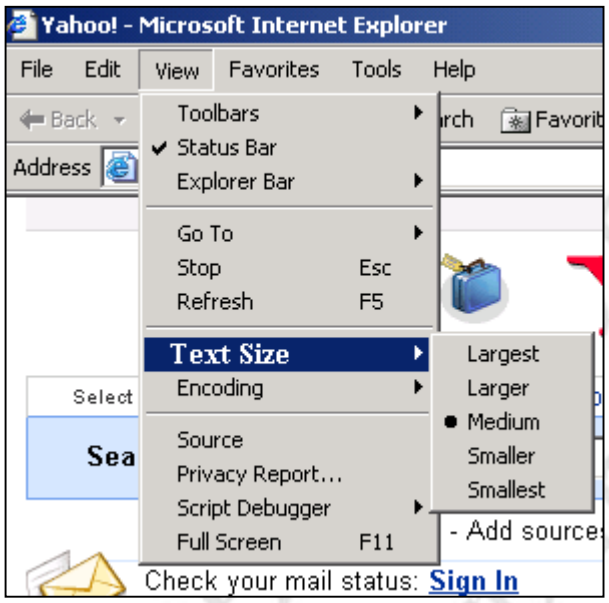
На дадената фигура със сив правоъгълник е обозначено изображение, вмъкнато в HTML документа. Редовете текст, обозначени с линии “обтичат” отдясно изображението. След етикета <BR Clear=”Left”> обаче обтичането отдясно се прекратява и следващият ред текст започва от позиция под изображението непосредствено от лявата граница на документа, обозначена с пунктир. В този пример ако стойността на атрибута Clear беше “All”, резултата щеше да бъде същия, защото отдясно нищо не пречи на текста да стигне до дясната граница на документа.

Задаване на параметри на шрифта

HTML предоставя голям набор от етикети, с които се задават параметри на извежданите символи. На практика всички възможности за форматиране,

които предоставят текстовите редактори като WORD, са достъпни и в HTML документите, като дори има дублиране на функциите на някои етикети за форматиране. По-долу е дадено кратко описание на етикетите според тяхното предназначение.

8.4 Задаване на вид, размер и цвят на шрифта – етикет <FONT>



Фиг. 3 Избор на подразбиращ се размер на шрифта в Internet Explorer

Контейнерния етикет FONT е основния етикет за форматиране на текст. Той притежава следните атрибути:

- **FACE** задава вид (име) на шрифта.
- **SIZE** задава размер на шрифта.
- **COLOR** задава цвят на шрифта.

На Фиг.2 е показана функцията в менюто View на Internet Explorer, в която чрез опция Text Size се избира подразбиращ се размер на шрифта.

Цветът на шрифта се задава чрез атрибут **COLOR** според изискванията за задаване на цвят в HTML:

a) Чрез име на цвета, напр. COLOR=”green” за зелен цвят
b) Чрез шестнадесетична стойност на яркостта на трите основни цвята (червен, зелен, син) както е описано в т 5.

Други етикети за форматиране на текст

HTML предоставя група контейнерни етикети без атрибути (с изключение на атрибутите за задаване на стил, които са въведени в последствие и могат да се използват на практика с всички контейнерни етикети за форматиране). Всеки от тези етикети (тагове) задава определен пааметър на текста и може да се използва в комбинация с останалите.

<B> **за удебелен шрифт (BOLD)** </B>

<I> *за курсивен шрифт (italic)*</I>

<U><u>за подчертан шрифт(underline)</u></U>

<Strike>~~за зачертан шрифт(strikethru)~~</Strike>

<BLINK> (въведено за Netscape, но понастоящем се отработва и от други браузъри) </BLINK> - Задава “мигане” на текста. Трябва да се използва рядко, защото мигането се възприема като дразнещ ефект от потребителите.

<TT> за телетекст</TT> - шрифт, при който всички знаци и букви имат еднаква ширина. По този начин става възможно отделни букви или знаци да се поставят точно едни под други.

<CENTER> текст или картина </CENTER> за центриране.

Използвайте <SMALL> за да намалите стандартния текст до три пъти:

<SMALL>по-малък текст</SMALL>

<SMALL>< SMALL>още по-малък текст</ SMALL></ SMALL >

Използвайте <BIG> за да увеличите стандартния текст до три пъти:

<BIG >по-голям текст</BIG>

<BIG><BIG>още по-голям текст</BIG></BIG>

8.5 Структуриран текст

Структурираният текст предава на текстовия откъс определени структурни или характерни свойства (понякога наричани емфатичност), които могат да се представят по различен начин от различните медии. Ако, например, една дума трябва да се открои, браузерът избира дали това да стане с получерен шрифт или курсив, което зависи от различни фактори. Поради тази зависимост от вида на браузъра етикетите за структуриране понякога се наричат неявни етикети. Значителният брой етикети за структуриран текст, някои от които дават съвсем еднакъв резултат, могат да се разглеждат като свидетелство за дългият път в развитието на HTML докато се достигне до неговото стандартизиране. Съвременният Web дизайнер може изобщо да не използва тези етикети, ако използва каскадните стилове, които предоставят много по-богати възможности.

Използвайте:

<EM> за открояващ се текст (т.н. емфатичност) **</EM>**

<STRONG> за силно открояващ се текст **</STRONG>**

<DFN> за сектори, съдържащи дефиниции **</DFN>.** Повечето браузери, познаващи този елемент правят маркирания текст курсив.

<CODE> за шрифт на програмни списъци **</CODE>** Програмни списъци, елементи или обекти трябва да се форматират чрез CODE, ако друг елемент за форматиране вече не е направил това.

<SAMP> за извеждане на примерни програми **</SAMP>**. За кратки текстове, форматирани от повечето браузери като телетекст.

<KBD> за въвеждане от клавиатурата **</KBD>**. Текстът се извежда с шрифт с постоянна ширина на буквите, с което се имитира извеждане на текст от телетайп. Такъв шрифт е например **Courier.** При повечето шрифтове символите (напр. I и W) се извеждат с различна ширина

<VAR> за променливи **</VAR>**. Променливите са особен случай при форматирането на текста и затова се маркират специално с VAR. На практика резултата от прилагането на този етикет е същия както за

<CITE> за цитати **</CITE>**. Цитати от книги или от други медии се форматират със CITE, ако за това не се използват BLOCKQUOTE или Q. Повечето браузери форматират цитатите посредством елемента CITE с курсивен шрифт. Ако оригиналът на цитата е документ, достъпен в WWW, се препоръчва цитатът да се обозначи доълнително като хипервръзка (елемент A)

<ABBR> за съкращения **</ABBR>**. Съкращенията се възприемат за първи път в HTML 4.0 като градивен елемент на текста. Тук се включват всички кратки версии на имена или понятия като например WWW за World Wide Web или UFO за неидентифициран летящ обект.

<ACRONYM> за акроними **<ACRONIM/>**. Действа подобно на елемент ABBR.

8.6 Резюмета и сентенции. Етикети <Q> и <BLOCKQUOTE>

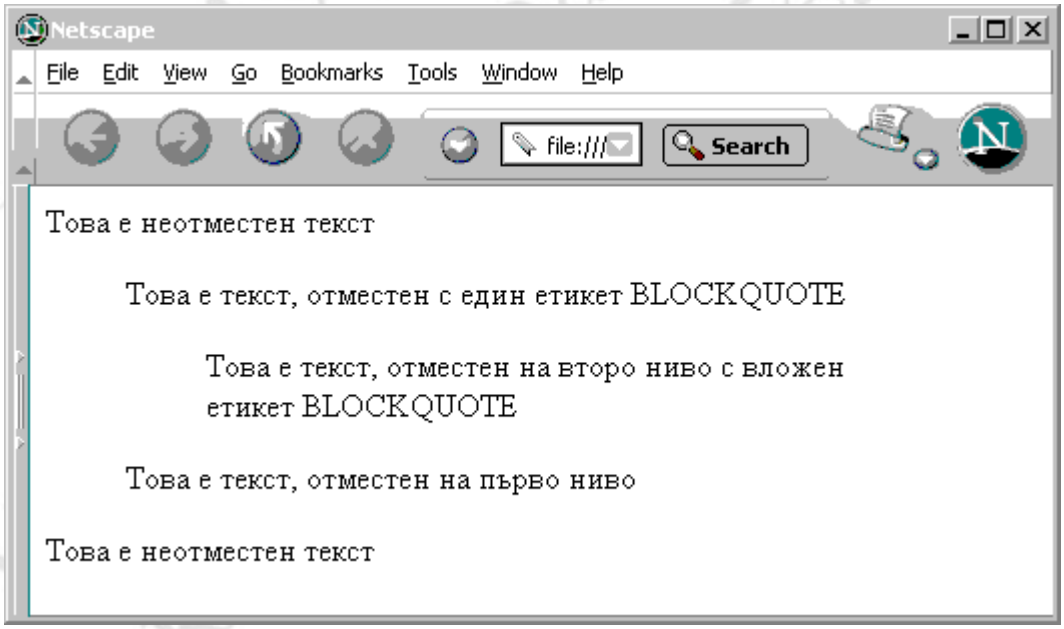
За сентенции (цитати) или цели резюмета са резервирани два особени HTML елемента – Етикетът <Q> поставят автоматично включения в него текст в кавички.), а <BLOCKQUOTE> поема грижата за форматирането на цели параграфи. Етикетът <Q> не е от особено значение, тъй като лесно можем да получим същият резултат ако сами добавим необходимите кавички. Етикетът <BLOCKQUOTE> обаче е изключително полезен, тъй като дава възможност за отместване навътре от лявата граница на включения в него текст. Допуска се влагане на етикети <BLOCKQUOTE> един в друг, с което се постига отместване навътре на няколко нива.

```
<HTML>
<HEAD>
```

```
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=windows-1251">
   </HEAD>
   <BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#0000FF"
VLINK="#800080">
   Това е неотместен текст<blockquote>Това е текст, отместен с един етикет
BLOCKQUOTE
   <blockquote>Това е текст, отместен на второ ниво с вложен етикет
BLOCKQUOTE
   </blockquote>Това е текст, отместен на първо ниво
   </blockquote>Това е неотместен текст

   </BODY>
   </HTML>
```

Пример 23 Използване на етикет <BLOCKQUOTE> за отместване навътре на текст



Фиг. 4 Резултат от използването на етикета <BLOCKQUOTE>

8.7 Горен и долен индекс

Контейнерния етикет _{долен индекс} поставя текста под основата с намален размер (като индекс). Например A₅ ще се изобрази като A_5.

Контейнерния етикет ^{горен индекс} поставя текста над основата с намален размер (като степен). Например B< SUP >3</ SUP > ще се изобрази като B^3.

8.8 Предварително форматиран текст

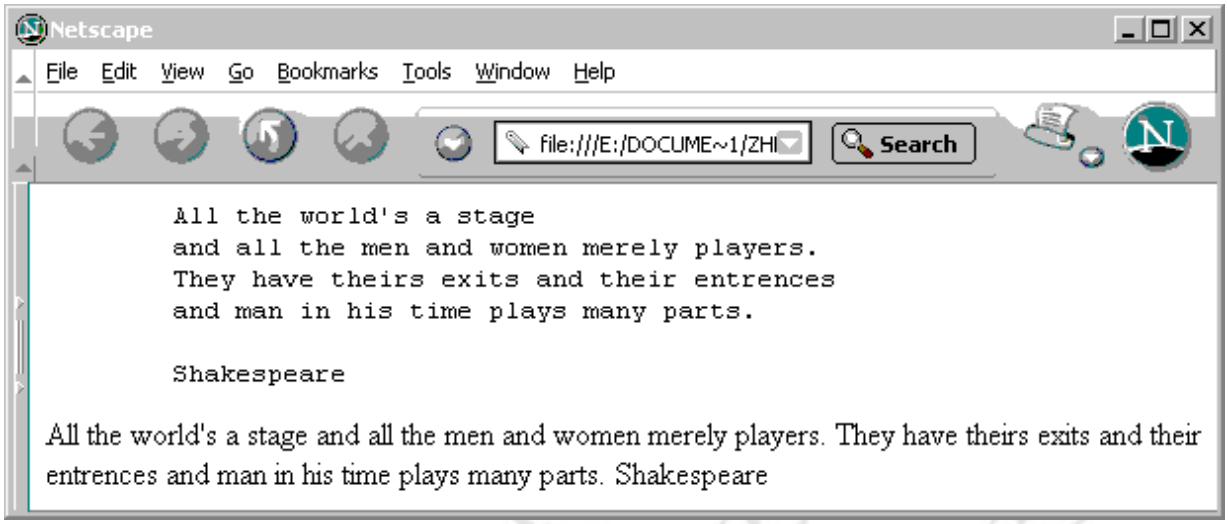
Контейнерния етикет за предварително форматиран текст <PRE> изобразява символите за празен интервал, табулация и преминаване на нов ред така, както те се изобразяват в текстовите редактори. По подразбиране браузърите изобразяват няколко последователни празни интервала само като един празен интервал. Символите за табулация и преминаване на нов ред също се изобразяват като един празен интервал. Етикетът <PRE> позволява текста да изглежда на екрана така, както е написан по редове и с разстояния, получени с табулации и с последователни празни интервали текст. Даденият по-долу пример позволява да се види разликата в изобразяването на текст, включен в етикета <PRE> и извън него. Текстът в етикета <PRE> се изобразява по редове и с отместване с една табулация навътре, докато вънщият текст извън етикета се изобразява като последователност от думи с един празен интервал между тях, като на нов ред се преминава само тогава, когато текстът достигне до дясната граница на документа.

```
   <HTML>
   <BODY>
   <PRE>
   All the world's a stage
   and all the men and women merely players.
   They have theirs exits and their entrences
   and man in his time plays many parts.

   Shakespeare
   </PRE>
   All the world's a stage
   and all the men and women merely players.
   They have theirs exits and their entrences
   and man in his time plays many parts.

Shakespeare
   </BODY>
   </HTML>
```

Пример 24 Използване на етикет за предварително форматиран текст <PRE>



Фиг. 5 – Резултат от използването на етикета <PRE>

8.9 Хоризонтална разделителна линия <HR>

HTML предлага специален неконтейнерен (т.е. без затваряща част) етикет <HR>за изобразяване на хоризонтална линия в документа. Етикетът притежава няколко незадължителни атрибути, с които се задават определени параметри на линията

Атрибути:

<HR size = 1 до 10> за дебелина на линията,

<HR width=стойност> за ширина на линията (в % или пиксели)

<HR align=left/center/right>. за подравняване на линията спрямо границите на документа

<HR noshade > за изобразяване на линията плътно и без сянка.

Пример:

```
<html>
<head>
<title>Хоризонтални линии</title>
</head>
<body>
<hr width="20%" size="5">
<hr width="50%" size="5" noshade>
<hr width="75%" >
</body>
</html>
```

Пример 25 Използване на етикета за хоризонтална линия HR

Фиг. 6 Хоризонтални линии, получени с етикета <HR>

8.10Включване на коментари в HTML документ.

HTML дава възможност за включване на коментари в документите. Коментарните редове в HTML кода са невидими за потребителите и служат за обозначаване на бележки на програмиста. В началото на коментара се записва **<!--** а в края на коментарния текст се записва.-->. Всичко, включено между тези маркери не се изобразява от браузъра, а HTML етикетите не се отработват. Например <!-- <B>Hello</B> --> няма да изведе нищо в прозореца на браузъра. Както ще видим във втората част на настоящия учебник, посветена на JavaScript, началният и краен маркер за коментар могат да се включват и в контейнерния етикет <SCRIPT>. Това се прави, за да се избегне изобразяването на кода на програмата (скрипта) като текст от по-стари браузъри.

9 HTML списъци

Списъците са части от текста,изброяващи определени елементи. Пред всеки от тези елементи стои символ или номер в зависимост от вида на списъка. HTML предлага възможност за реализиране на два основни вида списък според символа, извеждан преди елементите – номериран и неномериран. Общото в структурно отношение между различните видове HTML списъци е:

- Списъците се реализират с контейнерни етикети, като за всеки елемент от списъка в контейнерния етикет се включва етикет за елемента и съдържание на елемента.

- Списъците могат да се влагат един в друг. Влагането става, като един елемент от външния списък се замени със цял списък. Така списъкът, заменящ елемент от външния списък се оказва вложен в него.

9.1 Неномериран списък <UL>

Неномерираният списък се реализира с обграждащ етикет <UL> - съкращение от английското наименование на неномерирания списък - **Unordered List**. Елементите от списъка се съдържат в обграждащият етикет <LI> от английското наименование на указател на списък – **List Index**.

Атрибути:

type=”disc/circle/square” – задава символа, с който започват елементите от списъка. Възможни стойности: “disk” – за запълнен кръг, “circle” – за окръжност и “square” – за квадрат. Важно е да се отбележи, че за да бъде отработена зададената стойност на атрибута type, тя трябва да бъде записана с малки букви, т.е. стойността ”circle” ще се приеме, но задаването на стойност “CIRCLE” ще има за резултат използването на подразбиращата се стойност на атрибута.

9.2 Номериран списък <OL>

Номерираният списък се реализира с контейнерния етикет <OL> - съкращение от английското наименование на номерирания списък - **Ordered List**. Елементите от списъка се кодират както за неномерирания списък.

Атрибути:

type=1/a/A/i/I - указва начина на номерирането. Възможни стойности: “1” – за номериране с арабски цифри, “a” – за номериране с малки букви, “A” – за номериране с големи букви, “i” за номериране с малки римски цифри, “I” за номериране с големи римски цифри.

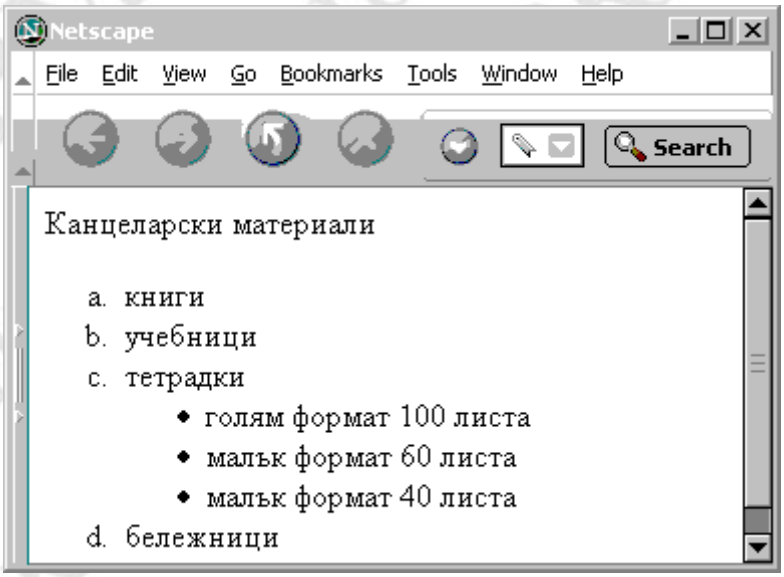
Start – указва началната номерация на елементите в списъка.

9.3 Елементи на списъка <LI>

В обграждащите <UL> и <OL> всеки елемент на списъка се означава с <LI>.или <LI value=число>. Атрибутът value се използва за елементи от номериран списък, които трябва да получат нова номерация. Value се задава винаги като число, дори когато за оформяне на външния вид на списъка са избрани букви или римски цифри. Чрез атрибута value може да се получи “прескачане” на номерацията на елементите в списъка. Ако след елемента със зададена стойност на атрибут value следват елементи без атрибут value, то за тях номерацията се променя отново последователно.

```
<HTML><HEAD>
META HTTP_EQUIV="Content-Type" content="text/html;
charset=windows-1251">
</HEAD><BODY >
<p> Канцеларски материали</p>
<OL type="a">
<li> книги</li>
<li> учебници</li>
<li> тетрадки
                    <ul type="disc">
                    <li> голям формат 100 листа</li>
                    <li> малък формат 60 листа</li>
                    <li> малък формат 40 листа</li>
                    </ul>
<li> бележници
</OL>
</BODY></HTML>
```

Пример 26 Вложени списъци в HTML



Фиг. 7 Вид на вложени списъци в HTML документ

9.4 Списъци с дефиниции <DL>

Този вид списъци се използва по-рядко. Той е структуриран за извеждане на списък с дефиниции на термини със съответни обяснения. Елементът <DL> служи за контейнер на всички термини в списъка с дефиниции. Термините се реализират посредством елементите <DT>

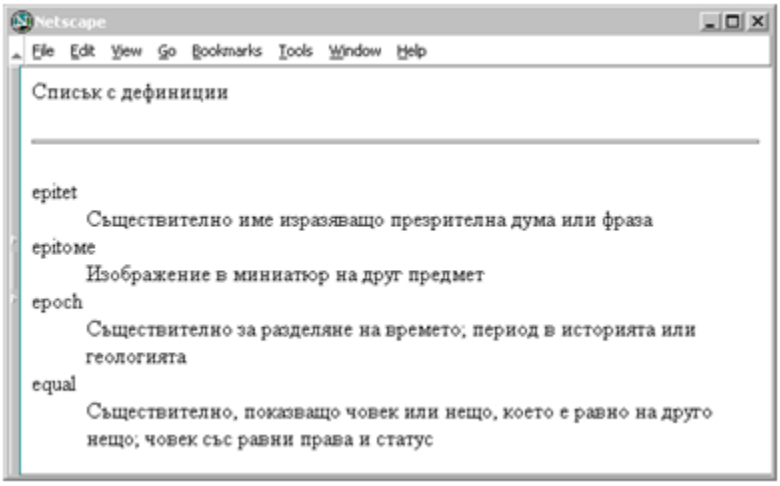
(дефиниция на термина) и <DD>(описание на дефиницията.). По-долу е дадено кратко описание на тези елементи.

<DT>...</DT>. Дефиниция на термина. Отличава се от останалия текст (получерен шрифт и ляво подравняване). След всяка дефиниция следва описание (елемент DD)

<DD>...</DD>. Описание на дефиницията. Съдържа обяснителния текст. Извежда се на следващия ред след реда с наименованието на термина с отстъп навътре

```
<HTML><HEAD>
<META HTTP_EQUIV="Content-Type" content="text/html;
charset=windows-1251">
</HEAD><body >
<p>Списък с дефиниции<br></p><hr>
<dl>
<dt> epitet</dt>
<dd>Съществително име изразяващо презрителна дума или фраза</dd>
<dt>epitome </dt>
<dd>Изображение в миниатюр на друг предмет</dd>
<dt> epoch </dt>
<dd>Съществително за разделяне на времето; период в историята или
геологията</dd>
<dt> equal </dt>
<dd>Съществително, показващо човек или нещо, което е равно на друго
нещо; човек със равни права и статус</dd>
</dl>
</body>
</html>
```

Пример 27 Дефиниционен списък



Фиг. 8 Вид на дефиниционен списък в HTML документ

9.5 Списъци с директории и списъци с менюта

HTML предоставя и други видове списъци, които обаче е дават допълнителни възможности на Web дизайнера. Пример за това са списъците <DIR>...</DIR>. за организиране на разклонени директории и списъците <MENU>...</MENU> за представяне на разклонени менюта. Тези списъци се извеждат от Web браузерите във вид на неномерирани списъци <UL>.

10 Интернет адреси (URL)

Интернет адресите са адреси на документ или друг ресурс в компютърната мрежа WWW. Изпращането на заявка за доставяне на някакъв документ, активиране на някаква услуга или изпълнение на някаква програма (скрипт) задължително изисква включването на интернет адрес, за да се укаже кой трябва да изпълни заявката и какво трябва да се достави на заявителя. Интернет адресите по правило се обозначават с URL – съкращение от Uniform Resourse Locator – адрес на ресурс във Web.

Общият формат на URL е:

protocol://hostname:port/pathname search hash

където значението на отделните елементи на адреса е както следва:

- protocol – Началото на URL до първото двоеточие. За заявка заHTML документ протоколът е **http: ,** за заявка за доставяне на файл – **ftp: ,** за активиране на услугата електронна поща – **mailto:** и т.н.
- hostname – името (или цифров адрес) на мрежовия компютър (host) който трябва да изпълни заявката. Hostname се състои от три части: host.domain.first-level domain.
- port – номер на комуникационен порт за предаване на заявката. Всъщност това е един номер, който се предава заедно със заявката и дава възможност за определяне на предназначението на заявката. Този елемент по правило се пропуска в URL, тъй като видът на протокола в повечето случаи определя и стойността на port. Например за http протокола номера на port е 80, за ftp – 21.
- pathname – пълната файлова спецификация, която включва път, име и разширение на файла. Важно е да се уточни, че файловата спецификация се изписва според правилата на операционната система Unix – като

разделител между имената на директориите се използва “/” , а не “\” , както е прието в операционната система Windows, и освен това се прави разлика между малки и големи букви. Например низа “/mysite/mypage.htm” е пример за правилно записана стойност на path.

- search – информация за отговор на въпроси или предаване на параметри, започваща с въпросителен знак. За заявки за HTML документи по правило този елемент отсъства.
- hash – име на точка за преход (котва), което започва със символа за паунд (#).

Да разгледаме един пример. Адресът

http://www.webopedia.com/TERM/U/URL.html

включва следните елементи:

- protocol - http:
- hostname - www.webopedia.com
- pathname - /TERM/U/URL.html

Детайлното разглеждане на протоколите, използвани във WWW и техните характеристики е извън предназначението и обема на настоящия курс. Повече информация можете да намерите в специализираната литература по компютърни мрежи, или в спецификациите на W3C на адрес http:// www.w3.org/Protocols.

11 Включване на изображения в HTML документ.

Изображенията са един от основните елементи на HTML документите. Те придават естетичен вид на Web страниците, дават възможност за променяне на вида на страницата при действия на потребителя и за създаване на прости анимации. HTML дава възможност за включване в документа на изображения, които се съхраняват във файл с определен формат.

11.1 Графични файлови формати, поддържани от браузърите

Съществуват голям брой файлови формати за графични изображения. Повечето от тях са свързани с определен програмен продукт и определена фирма. В HTML документите могат непосредствено да се включват само растерни изображения, записани във формати BMP, GIF, JPG и PNG. За Web дизайнера е важно да познава характеристиките на тези файлови формати, за да може да подбере подходящите графики за разработвания сайт.

Формат BMP – Основна отличителна характеристика на този формат е липсата на компресия. Размера на файла еднозначно се определя от размера на изображението и от цветовата палитра, т.е. от максималния брой цветове – черно-бяло, 16-цветно, 256-цветно или 2^{24}- цветно изображение. Поради липсата на компресия графичните файлове във формат BMP са със значително по-голям размер от аналогични изображения във формат GIF или JPG, поради което не се препоръчва този файлов формат в HTML документите.

Формат GIF – Този формат притежава няколко интересни характеристики.

- Изображенията в GIF формат могат да имат не повече от 256 различни цвята. Поради това той не е подходящ за изображения, получени чрез сканиране или цифрова фотография.
- Форматът използва беззагубна компресия (алгоритъмът за компресия е известен като LZW). Беззагубността на компресията означава, че в резултат на компресията не се влошава качеството на изображението.
- GIF форматът дава възможност за задаване на прозрачен цвят (transparent color). При задаването на един цвят като прозрачен, при извеждане на изображението в прозореца на браузъра, вместо този цвят се вижда фона на HTML документа. Така изображението изглежда като част от фона, а не като правоъгълник, поставен върху него.
- Четвърта, много полезна за дизайнера фарактеристика на GIF формáта е възможността за създаване на анимирани изображения. Форматът GIF 98a дава възможност за записване в един файл на последователност от изображения, които браузърът извежда последователност като кадри на анимацията.

Формат JPG – Този формат притежава две основни характеристики:

- Няма ограничения за броя на цветовете – цветът се кодира с 24 двоични разряда.
- Използва загубна компресия. На практика по-голямата компресия води до по-забележимо влошаване на контраста на изображението. Това се забелязва чувствително при запис на изображения, които съдържат контрастни линии (например чертежи) и дребни детайли. Форматът е подходящ за полутонови изображения с по-големи размери, например цифрови фотоснимки.

Формат PNG (Portable Network Graphics format) – Този формат е създаден след GIF формáта с намерението да бъде по-добър от него. Ето някои основни характеристики

- Няма ограничения за броя на цветовете –може да поддържа до 24-битов цвят както JPG.

- Дава 256 стойности на прозрачност (а не само прозрачен-непрозрачен както GIF)
- Прави по-добра беззагубна компресия от GIF формата

Повече информация можете да намерите на сайта http://www.libpng.org/pub/png/. Независимо от суперлативите този формат все още не намира голямо приложение. Форматът е по-сложен и при неправилен подбор на параметрите дава по-големи по размер файлове, независимо от рекламираната по-добра компресия. Затова нашата препоръка е: придържайте се към GIF за изкуствено генерирани изображения и към JPG за сканирани изображения и фотографии.

11.2Етикет <IMG>

Включването на изображения в HTML документ става с неконтейнерния етикет <IMG>. При това в документът не се включват и данните от графичния файл – за да се изведе изображението в прозореца на браузъра е необходимо а бъде достъпен и графичния файл с изображението. Това е съществена разлика между HTML и някои други документни формати, например RTF, в които включените изображения се записват във файла на документа.

Атрибути:

- src=”URL-адрес на графичния файл” – Това е основния и единствено задължителен атрибут на етикета <IMG>. Той трябва да съдържа относителен или абсолютен адрес (URL) на графичния файл[3]. По принцип адресът може да бъде кой да е достъпен интернет адрес на графичен файл. Например ако включите във вашата HTML страница етикет

• <img src=”http://kst-forum.netfirms.com/tu_varna_logo.gif”>

- при извеждането на страницата в прозореца на браузъра в нея ще се появи анимирано изображение на емблемата на ТУ-Варна. Това обаче няма да се получи ако нямате връзка с интернет мрежата или ако по някакви причини Web сървърът, който обслужва зададеният адрес не е активен. Затова на практика винаги изображенията, които са включени в HTML документа се съхраняват там ,където се съхранява и самия документ. Признак за добър стил е, ако в сайта се създаде поддиректория **images,** в която да се записват всички графични файлове.

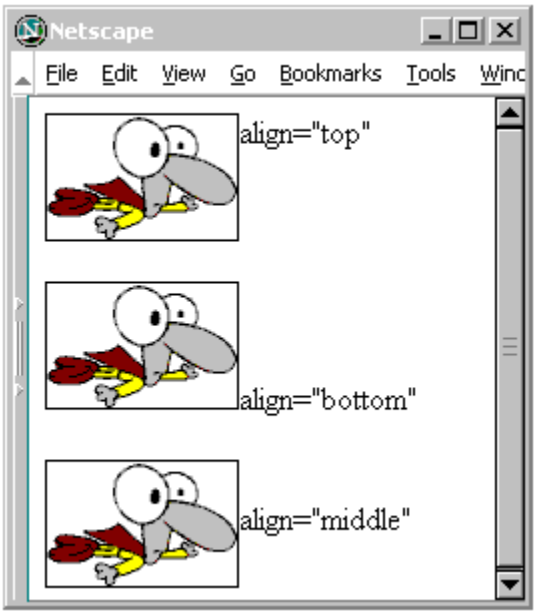
[3] Вижте т. 9 Интернет адреси

- lowsrc=”URL-адрес на графичен файл” –Този атрибут се използва тогава, когато основния файл, зададен в атрибут src, е голям по размер. Файлът, чийто адрес е зададен в атрибута lowsrc се зарежда пръв, така че докато се изтегля по-големият файл потребителят има възможност да разглежда едно по-малко или по-некачествено негово копие.
- width=”ширина”, height=”височина” – тези атрибути задават размера на изображението в прозореца на браузъра. Размерите могат да бъдат в пиксели или в проценти от размерите на прозореца. Браузърът мащабира зададеното изображение според стойностите на width и height. Ако атрибутите липсват, изображението се извежда с оригиналния си размер.
- hspace=”разстояние”, vspace=”разстояние” – тези атрибути задават хоризонталното и вертикално разстояние от изображението до съседните HTML етикети.
- border=”дебелина на рамката” – Задава дебелина на рамката, която се извежда от браузъра. Ако изображението се използва като хипервръзка и не е зададена стойност на border, около изображението се изобразява рамка с цвета на текстова хипервръзка. Начинът да се премахне тази рамка е да се зададе на border стойност 0.
- align=”top/bottom/middle/left/right” – задава подравняване на изображението. Стойностите left и right задават подравняване на изображението спрямо границите на документа (в по-общия случай спрямо контейнера – т.е. елементът, който съдържа изображението).

Стойностите top, bottom и middle задават подравняване на текста спрямо изображението, както е показано на следващия пример.

```
<HTML>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#0000FF"
VLINK="#800080">
<img align="top" border="1" src="FlyT.gif">align="top"
<BR><BR><img align="bottom" border="1" src="FlyT.gif">align="bottom"
<BR><BR><img align="middle" border="1" src="FlyT.gif">align="middle"
</BODY>
</HTML>
```

Пример 28 Подравняване на текст спрямо изображение с атрибута align на етикет <IMG>



Фиг. 9 Подравняване на текст спрямо изображение с атрибута align на етикет <IMG>

Възможностите за подреждане на текст и изображения в прозореца на браузъра са ограничени. Може да се видят примери, когато позиционирането на елементи се извършва посредством поставяне до тях малко изображение в GIF формат, съдържащо само един цвят, който е обявен като прозрачен. Такова малко изображение може да се използва на много места в HTML страницата, като му се задават подходящи размери с width и height. Това обаче е нетрадиционен начин – по-добри резултати могат да се получат с използването на таблици.

12 Хипервръзки

Хипервръзката по правило е връзка между два Web документа. Тя свързва две точки, наречени котви (anchors). Хипервръзката започва в изходния документ и сочи към котвата на целта, която може да е произволен Web документ, или пък да се намира в същия документ. В по-общия случай хипервръзката е средство за изпращане на заявка за Web документ или Web услуга.

Включването на хипервръзки в един документ дава възможност за навигация в хиперпространството. Хипертекстът представя не само информацията, съдържаща се в текста на документа, но и тази, съдържаща се във всички, свързани с него документи. Тази изключително плодотворна идея на Тим Бърнардс Лий е една от основните причини за превръщане на компютърната мрежа WWW в световна информационна система.

Подразбиращия се вид на хипервръзката е подчертан текст, оцветен в син цвят. Като хипервръзка обаче може да се използва не само текст, но и

графично изображение, а също и “горещи” области от изображение т.н. HTML карти.

Хипервръзките се създават с контейнерния етикет <A> … </A> (от anchor – котва).

Атрибути:

- href=”адрес на хипервръзката URL” – Този атрибут е задължителен за хипервръзките. Той задава Web адреса, на който се изпраща заявка. Без този атрибут хипервръзката не реагира на щракване с мишката върху нея
- name = име>. Задава име на котва (точка за преход) в рамките на документа. Етикет <A>, който съдържа атрибут name може да не съдържа атрибут href – тогава той ще създава само точка за преход (котва), но няма да създава хипервръзка., под което може да се намери в документа. Достъп до това име може да се осъществи както в рамките на URL спецификациите, така и отвън. В този случай е достатъчно присъединяването на знака # и на името на котвата към името на файла на документа.
- target="име_на_прозореца" – Задава името на прозореца (фрейма), в който трябва да се зареди заявеният документ. Като стойност на атрибута могат да се използват запазените думи “_SELF”, “_PARENT”, “_TOP” и “_BLANK”, които имат следното значение:

_SELF	Означава да се зареди заявения документ в прозореца на хипервръзката. Може да се каже, че използването на тази стойност е безмислено, тъй като същия резултат можем да получим и ако не включваме изобщо атрибут target.
_PARENT	Означава да се зареди заявения документ в родителския прозорец, т.е. в прозореца (или фрейма), който включва прозореца на хипервръзката.
_TOP	Означава да се зареди заявения документ в най-външния фрейм, който всъщност е прозореца на браузъра.
_BLANK	Означава да се отвори нов прозорец на браузъра и документът да се зареди в него.

- **title=**"подсказка” – Един полезен атрибут, на който си струва да обърнете внимание. Задава подсказка (tooltip) – малко прозорче със зададения текст, което се извежда, когато курсора на мишката се задържи над хипервръзката.

12.1Създаване на връзки към други Интернет услуги . Хипервръзки за E-Mail съобщения

В описанието на URL, дадено в т. 9, е посочено, че видът на протокола, зададен като елемент на интернет адреса URL, определя какъв документ ще бъде заявен или каква услуга ще бъде активирана. Ако искаме да активираме функцията за електронна поща трябва да зададем на атрибута *href* на хипервръзката протокол *mailto*. Ето пример за код на такава хипервръзка.

```
<A href=mailto:my_mail@yahoo.com> Send me e-mail</A>
```

Пример 29 Код на хипервръзка за изпращане на e-mail

12.2Създаване на графични хипервръзки

Web браузърите дават възможност да се използва изображение вместо текст на хипервръзка. Това означава, че ако между отварящата и затварящата част на етикета <A> се включи код за графично изображение, браузърът ще реагира на операции с мишката върху изображението както и върху текстова хипервръзка. Ето пример за код на такава хипервръзка.

```
<a href="www.fakecorp.com/"><img src="biglogo.gif"
alt=Bigcorp"></a>
```

Пример 30 Код на графична хипервръзка

12.3Използване на изображения като HTML карти

HTML картите дават възможност едно изображение да предоставя няколко хипервръзки, всяка от които е свързана с определена област от изображението. Така изображението може да съдържа различни по очертание области (т.н. “горещи” области – hot spot), всяка от които реагира на щракване с мишката върху нея като хипервръзка. Така се преодолява ограничението на графичните хипервръзки , при които хипервръзката е привързана обязателно към правоъгълна област.

За да се създаде HTML карта с дефиниция, която се съдържа изцяло в HTML документа, и следователно функционира изцяло от страна на клиента на WEB сайта, е необходимо следното:

- В HTML документа да се включи структура MAP, която дефинира областите на картата и съответните им хипервръзки.
- В етикета <IMG> за графичното изображение да се включи атрибут usemap, който да получи като стойност името на структурата MAP.

12.3.1 Структура MAP.

Структурата MAP се състои от контейнерния етикет <MAP> , в който са включени етикетите <AREA>, с които се описват областите на картата. Етикетът <AREA> има следните атрибути:

- href=”адрес на хипервръзката URL” – Този атрибут е задължителен за етикета <AREA>. Той задава Web адреса, на който се изпраща заявка. Без този атрибут областта от картата не реагира на щракване с мишката върху нея.
- shape=”вид_на_областта” – Този атрибут задава вида на областта от изображението (картата), която е привързана към зададения в href адрес. Възможните стойности са: RECT за правоъгълник, POLY за многоъгълник (полигон), CIRCLE за кръгова област и DEFAULT за всички точки извън изображението извън областите RECT, POLY и CIRCLE.
- coords=”списък_от_координати” - Този атрибут задава координатите на областта, зададена с атрибута shape. За различните видове области координатите се задават както следва:

Вид област	Координати
RECT	coords="x1,y1,x2,y2" (Координати на горния ляв и долния десен връх на правоъгълника)
CIRCLE	coords="x,y,r" (Координати на центъра и дължина на радиуса на окръжността)
POLY	coords="x1,y1,x2,y2,x3,y3,..." (Координати на върховете на многоъгълника)

- target="име_на_прозореца" – Задава името на прозореца (фрейма), в който трябва да се зареди заявеният документ. Като стойност на атрибута могат да се използват запазените думи “_SELF”, “_PARENT”, “_TOP” и “_BLANK”, които имат същото значение, както за етикета за хипервръзка <A>.
- **title=**"подсказка” –Задава подсказка (tooltip) – малко прозорче със зададения текст, което се извежда, когато курсора на мишката се задържи над областта.

Да разгледаме един пример:

```
<MAP name="MyMap">
<AREA shape="circle" coords="58,50,40" href=”p1.htm" title="Circle">
<AREA shape="rect" coords="136,11,227,89" href=" p2.htm"
  title="Rectangle">
<AREA shape="poly" coords="309,13,358,89,257,89" href="p3.htm"
  title="Triangle">
<AREA shape="default" href="p3.htm" title=”default”>
</MAP>
```

Примерът съдържа структура MAP, в която са дефинирани три области – кръгова, правоъгълна и триъгълна. За всяка от областите в съответния етикет <AREA> е включен атрибут href, който задава адреса на HTML документ, който да се зареди в браузъра при щракване с мишката върху съответната област. Етикетът <AREA> с област default задава адреса на HTML документ, който да се зареди в браузъра при щракване с мишката върху изображението извън зададените области circle, rect и polygon.

12.3.2 Атрибут USEMAP на изображение за карта.

За да може едно изображение да се използва като карта, в етикета <IMG> на изображението трябва да се включи атрибут *usemap*, който да получи като стойност името на структурата MAP, която описва областите на картата, предшествано от символа шарп “#”. Например етикета <IMG>, който може да се използва за структурата MAP, зададена в предходния пример, може да има следния вид:

```
<IMG src=”MyMapImage.gif” width=200” height=”200”
usemap=”#MyMap”>
```

13 HTML таблици

HTML таблиците са изключително ценен инструмент за Web дизайнера. Те дават възможност в прозореца на браузъра да бъде заделена правоъгълна област със зададени размери, която от своя страна да бъде разделена на клетки и във всяка клетка да бъде включено различно съдържание (включително и цели вложени таблици). Съдържанието на всяка клета може да бъде позиционирано самостоятелно както в хоризонтална, така и във вертикална посока. Структурата HTML таблицата по подразбиране (при еднакъв брой клетки във всеки ред) съответства точно на структурата на таблица от релационна база от данни (БД), и

поради това най-често извлечените от БД записи се представят в HTML документите чрез HTML таблици.

13.1Структура и атрибути на HTML таблиците

Структурата на HTML таблиците се описва с етикетите <<TABLE>, <TH>, <TR> и <TD>. Допълнително в последните версии на HTML са въведени етикетите за структуриране <THEAD>, <TBODY> и <TFOOT>. По-долу ще бъдат разгледани по-подробно значението и атрибутите на изброените етикети.

13.2Етикет за таблица <TABLE>

В него се включват всички елементи на таблицата. Атрибутите на етикета <TABLE> задават параметри на цялата таблица.

Атрибути:

align=”left/center/right”. – Задава подравняване на таблицата в хоризонтална посока. От HTML 4.01 атрибутът не се препоръчва за използване, но браузърите го поддържат. Препоръчва се подравняване с CSS стил.

- left: таблицата се подравнява спрямо левия край на Web страницата.
- center: таблицата се позиционира в средата на прозореца.
- right: таблицата се подравнява спрямо десния край на Web страницата.

width=”ширина”. Ширина на таблицата, задава се в проценти от размера на прозореца на браузъра или в пиксели. Например width=”100%” означава, че таблицата ще има хоризонтален размер, равен на размера на документа в прозореца на браузъра. При променяне на размера на прозореца ще се променя и размера на таблицата. Размер width=”100” ще означава, че таблицата ще има хоризонтален размер 100 пиксела независимо от размера на прозореца на браузъра. Поради това рзмерът в проценти ще наричаме по-нататък “относителен размер” а размерът в пиксели “абсолютен размер”.

height=”височина”. Височината на таблицата , както и ширината, се задава се в проценти от размера на прозореца на браузъра или в пиксели.

bgcolor=”фонов цвят” – Задава фонов цвят на цялата таблица. Цветът се задава по правилата за задаване на цвят в HTML, описани в т.5.

background=”фоново изображение” – Задава графичен файл, който трябва да съдържа изображение, с което се попълва като фон правоъгълната област, заета от таблицата.

border="ширина_на_рамката" – Задава ширина на външната рамка на таблицата. Ако този атрибут отсъства или му е зададена стойност 0, таблицата се изчертава с невидими линии.

frame=void/above/below/hsides/vsides/lhs/rhs/box/border >. Задава вида на външната рамка и разделителните линии между клетките. Ако таблицата трябва да извежда всички линии, този атрибут може и да не се използва – достатъчен е атрибутът **border**. Значението на възможните стойности атрибута **frame** е както следва:

void	Не се извежда външна рамка независимо от стойността на border. Може да има вътрешна рамка, ако е зададена с атрибут **border** или **rules**.
above	Извеждат се само горните линии от рамката
below	Извеждат се само долните линии от рамката
hsides	Извежда се само хоризонталните линии от рамката
vsides	Извежда се само вертикалните линии от рамката
lhs	Извежда се вертикалните линии от рамката без дясна външна
rhs	Извежда се вертикалните линии от рамката без лява външна
box/border	Извежда се пълна рамка

rules= none/rows/cols/all – Управлява вътрешната рамка на таблицата. Използването му (както и на атрибута **frame)** не е необходимо ако таблицата трябва да има всички разделителни линии – за това е достатъчен атрибутът **border**. Значението на възможните стойности атрибута **rules** е както следва:

none	липсват разделителни линии между клетките.
rows	извеждат се само хоризонтални разделителни линии между клетките.
cols	извеждат се само вертикални разделителни линии между клетките.
all	извежда се всички разделителни линии между клетките.

bordercolor="color" – Задава цвят на външната рамка на таблицата.

cellpadding="разстояние" - Задава разстояние в пиксели от рамката на клетките на таблицата до нейното съдържание.

cellspacing=" разстояние" - Задава разстояние в пиксели между рамките на клетките. По подразбиране тази стойност е 1 и поради това рамките на клетките (ако не са невидими) изглеждат като очертани с двойни линии. При стойност 0 на cellspacing се получава вътрешна рамка на таблицата от единични линии.

cols="брой колони" – Задава брой колони, което означава на практика брой клетки на ред. Може да се каже, че използването на този атрибут е излишно, защото броя клетки в даден ред от таблицата в действителност се определя от броя етикети за клетки <TD>.

13.3Стилове за таблица

Таблиците, както и останалите HTML елементи могат да се форматират и със стилове. Ето някои специфични за таблицата стилове

13.3.1 Задаване на разстояние между клетките. Стилове border-spacing и border-collapse

Чрез стилът border-spacing се задава разстояние между рамките на клетките. Пример с декларацията table { border-spacing: 10px; } се задава разстояние 10 пиксела между рамките на клетките. Същият резултат можем да получим и с атрибута cellspacing, както беше дадено по-горе, например <table cellspacing="10">

Чрез стилът border-collapse се разрешава или забранява разстояние между рамките на клетките. При подразбиращата се стойност *separate* се разрешава да има разстояние (по подразбиране 1 пиксел) а при стойност *collapse* се забранява разстоянието ! независимо от стойностите на стилът border-spacing и атрибутът cellspacing, т.е. стилът *border-collapse* има по-висок приоритет.

Важно е да се запомни, че разстоянието между клетките на таблица не може да се управлява със стилът *margin*, който се прилага за слоеве например. За таблици този стил не работи.

13.3.2 Позициониране на текст в таблица. Стилове text-align и vertical-align

Позиционирането на съдържанието на клетките от таблицата може да се управлява и посредством стиловете text-align (за хоризонтално позициониране със стойности left, center и right – ляво, центрирано и дясно) и vertical-align (за вертикално позициониране със стойности top, middle и bottom – горе, в средата и долу). Тези стилове могат да се задават за цялата таблица, за редове <tr> и клетки <td> и <th>, като с най-висок приоритет са за клетките, след това за редовете, а с най-нисък – за таблицата.

13.4Редове в таблиците <TR>

Един ред в таблицата се състои от поредица включени в него клетки, които могат да бъдат клетки със заглавия и/или клетки с данни. В съответствие с това контейнерния етикет за ред съдържа контейнерните етикети са клетки <TD> за данни и <TH> за заглавия. Атрибутите на етикета <TR> задават параметри на целия ред, които се прилагат за всички клетки, ако за тях не са зададени стойности на аналогични атрибути.

Атрибути:

align=”left/center/right”. – Задава подравняване на съдържанието на клетките в реда в хоризонтална посока. За дадена клетка предимство има аналогичният атрибут на клетката, ако е зададен.

bgcolor=”фонов цвят” – Задава фонов цвят на реда. Цветът се задава по правилата за задаване на цвят в HTML, описани в т.5. Както и за предния атрибут предимство има атрибутът за клетка.

nowrap – На този атрибут не се задава стойност. Включването му означава, че текстът в клетките на реда няма да се пренася на следващ текстов ред. На практика това означава, че дължината на текста в дадена клетка ще определя дължината на клетката , ако текстът е по-дълъг от зададената с атрибут width дължина на клетката.

valign = "top/middle/bottom" – Задава вертикално подравняване на съдържанието на клетките в таблицата. Предимство има съответния атрибут на клетка, ако е зададен.

13.5Клетки в таблиците<TH>,<TD>

Това е най-малкият елемент в таблицата и може да бъде заглавна клетка или клетка с данни.

<TH> заглавие </TH>. Тези клетки се използват основно за озаглавяването на други клетки. В такъв случай текста в клетката автоматично се центрира в средата и се показва удебелен.

<TD> данни </TD>- съдържат действителните данни в таблицата.

По подразбиране съдържанието на клетките с етикет <TH> се извежда като центрирано с удебелен текст, докато съдържанието на клетките с етикет <TD> се извежда с ляво подравняване и с нормален текст. На практика друга разлика между използването на двата вида клетки в HTML таблицата няма. Двата етикета имат еднакви атрибути, с които се задават различни параметри на клетките в таблицата. Основната част от тях съвпадат с атрибутите на етикета за ред <TR>., но имат приоритет, когато са зададени в етикет за клетка.

Атрибути:

align=”left/center/right”. – Задава подравняване на съдържанието на клетката в хоризонтална посока.

bgcolor=”фонов цвят” – Задава фонов цвят на клетката. Цветът се задава по правилата за задаване на цвят в HTML, описани в т.5.

nowrap – На този атрибут не се задава стойност. Включването му означава, че текстът в клетката няма да се пренася на следващ текстов ред. Така при по-дълъг текст размера на клетката може да определи размера на съответната колона в таблицата.

valign="top/middle/bottom" – Задава вертикално подравняване на съдържанието на клетката.

colspan="брой" – Задава броя на колоните, които се покриват от клетката. Например colspan=”2” означава, че клетката ще заема дължината на две колони в раблицата – текущата (в която е дефинирана) и следващата.

rowspan="брой" – Задава броя на колоните, които се покриват от клетката. Например rowspan=”2” означава, че клетката ще заема дължината на два реда в раблицата – текущият (в който е дефинирана) и следващият.

Вместо атрибутите align и valign за позициониране на съдържанието на клетките могат да се използват CSS стиловете, които са разгледани по-горе за етикет (таг) <table>.

13.6 Участъци в таблиците и групи от колони

Новост в HTML4.0 е препоръката таблиците да се структурират в участъци. За целта са въведени три нови елемента:

<THEAD>...</THEAD>- прилага се най-често в книги, където една таблица може да се простира върху няколко страници. Заглавията се повтарят в горния край на всяка страница.

<TFOOT>...</TFOOT> -последните редове функционират по подобие на заглавните, но се намират в края на таблицата

<TBODY>...</TBODY>-основното съдържание на таблицата се структурира в участъци. Пример за приложението на участъци в таблиците са изброените в един каталог категории продукти.

При използване на тези етикети една таблица би трябвало да се структурира в следните участъци:

```
<TABLE>
    <CAPTION>
</CAPTION>
    <THEAD>     </THEAD>
    <TBODY>     </TBODY>
    <TFOOT>     </TFOOT>
</TABLE>
```

Пример 31 Използване на етикетите са структуриране на таблица в участъци

<COLGROUP span=”брой на колоните” width=”ширина”>. Разделяне на таблиците на хоризонтални участъци, с което се образува поне една група от колони.

13.7 Заглавие на таблица<CAPTION>

С този елемент таблиците се озаглавяват в горния или долен край. Елементът се прилага само веднъж и само на едно място в дефиницията на таблицата – непосредствено след въвеждащия таг на елемента TABLE.

Етикета има един атрибут, който определя подравняването на заглавния текст

Атрибути:

align=left/right/top/bottom>. Позволява заглавието да се разположи на една от следните позиции:

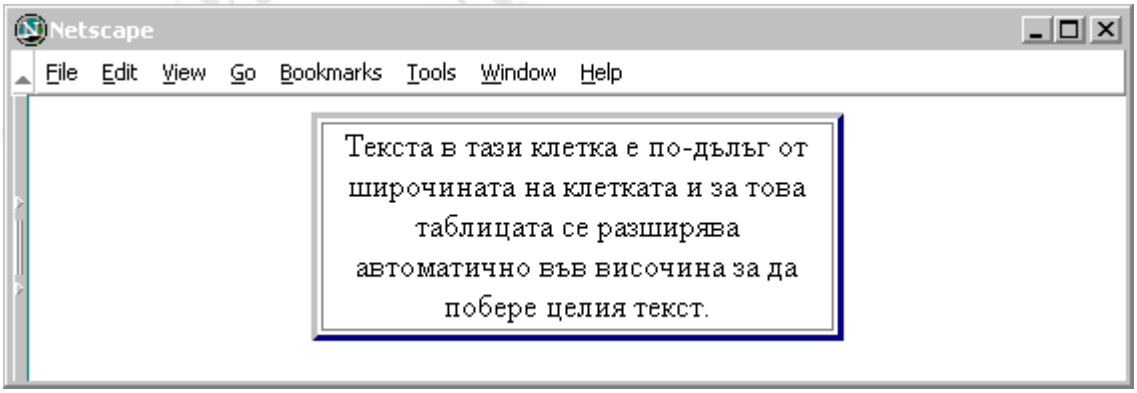
- left над таблицата, в левия край.
- right над таблицата, в десния край.
- top над таблицата, центрирано.
- bottom под таблицата, центрирано.

Примери за използване на атрибути на таблиците

По подразбиране, когато съдържанието на клетката е по голямо от размера и, тя автоматично се разширява във височина, за да го побере. Ето как изглежда по-дълъг текст в таблица с една колона и 1 ред със синя рамка с дебелина 3 пиксела и ширина 50% от екрана, подравнена в средата:

```
<table align="center" border="3"  width="50%" bordercolor="#0000FF">
<tr><td>
<p align="center">Текста в тази клетка е по-дълъг от широчината на клетката и за това таблицата се разширява автоматично във височина за да побере целия текст.
</td></tr>
</table>
```

Пример 32 Таблица, в която размерът на единствената клетка се определя от размера на текста.



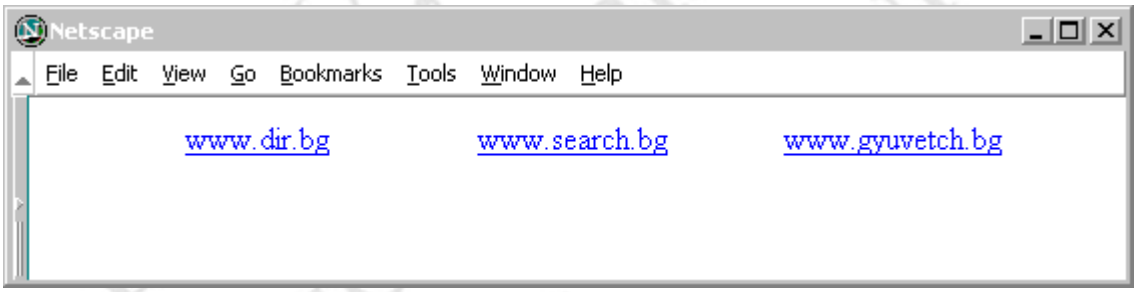
Фиг. 10 Пример за таблица, чийто размер се определя от дължината на текста

В примера вече имаме съдържание на клетка. То се поставя между елементите <TD> и </TD> и може да представлява текст, графика, или хипервръзка.

Ето как е направена таблицата в началото на тази глава с препратките до dir.bg, search.bg и gyuvetch.bg, но без рамка :

```
   <table align="center" border="0"  width="90%">
<tr>
<td width="33%">
<p align="center"><a href="http://www.dir.bg">www.dir.bg</a> </td>
<td width="33%">
<p align="center"><a href="http://www.search.bg">www.search.bg</a>
</td>
<td width="34%">
<p align="center"><a
href="http://www.gyuvetch.bg">www.gyuvetch.bg</a> </td>
</tr>
</table>
```

Пример 33 Таблица без рамка (border=”0”) която съдържа хипервръзки



Фиг. 11 Вид на HTML таблица без рамка в прозореца на браузъра

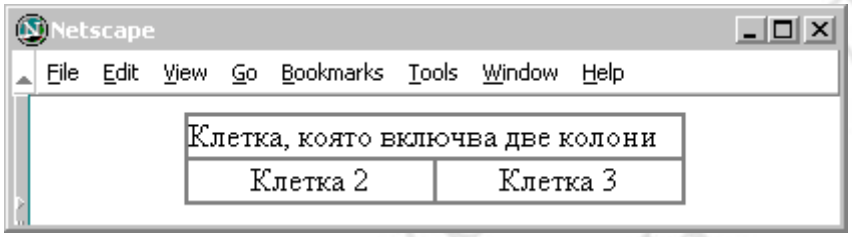
Таблицата на фиг. 10 е пример за оформяне на меню от хипервръзки. Подобно меню може да се види в много Web сайтове, като в много случаи се използват слоеве за появяване на падащо меню при преминаване на курсора на мишката над някоя от хипервръзките.

По-долу е даден пример за обединяване редове и колони в таблицата посредством атрибутите colspan и rowspan.на етикетите за клетка <TD> и <TH>. В пример 18 colspan="2" задава да се обединят съседните 2 клетки хоризонтално. Така първият ред от таблицата съдържа само една клетка, а втория – две клетки:

```
<table align="center" .border="1" cellspacing="0" cellpadding="0"
width="250">
<tr>
<td colspan="2">Клетка, която включва две колони</td>
</tr>
<tr align=”center”>
```

```
<td>Клетка 2</td>
<td>Клетка 3</td>
</tr>
</table>
```

Пример 34 Обединяване на клетки в HTML таблица посредством атрибута colspan.



Фиг. 12 Обединяване на клетки в HTML таблица посредством атрибута colspan.

В следващия пример посредством атрибута **rowspan=”2”** се обединяват две клетки от първи и от втори ред. Така в първи ред са включени етикети за две клетки – общата за двата реда и клетка 2, а във втори ред е включен етикет само за една клетка – клетка 3.

```
<table align="center" border="1" cellpadding="0" cellspacing="0"
width="250">
<tr>
<td rowspan="2"></td>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</table>
```

Пример 35 Обединяване на клетки в HTML таблица посредством атрибута rowspan

Фиг. 13 Обединяване на клетки в HTML таблица посредством атрибута rowspan.

13.8 Хоризонтално подравняване на HTML таблици с CSS стил

При разглеждането на HTML таблиците беше споменато, че атрибутът ALIGN на етикета <TABLE> на таблиците е обявен в спецификацията на HTML 4.01 за остарял и непрепоръчителен (deprecated), но Web браузърите продължават да го поддържат. CSS предоставя възможност за извършване на това позициониране, но за Internet Explorer от една страна и FireFox, Opera и Chrome от друга трябва да се използват различни дефиниции на стил. По-долу е даден пример за дефиниция на стилове за хоризонтално подравняване на таблици, който ще работи за всеки от изброените най-често използвани браузъри.

```
<style type="text/css">
<!--
 body {text-align:center;} /* За Internet Explorer */
 #tb {                     /* За FireFox, Opera и Chrome */
  margin-left:auto;
  margin-right:auto;
    text-align:center;
 }
 tr,td {text-align:left;}
-->
</style>
```

Пример 36 CSS стил за хоризонтално центриране на HTML таблица

В примера центрирането на HTML таблица за IE става чрез декларацията *body {text-align:center;}*, с която се предефинира атрибута за подравняване на текст за елемента BODY. На практика резултатът ще бъде хоризонтално центриране на всички таблици в документа и съдържанието на клетките в таблиците. За да се възстанови подразбиращото се ляво подравняване на съдържанието на клетките в секцията е включена и

декларацията *tr,td {text-align:left;}*. Това наистина ще работи, но в документа променянето на подравняването в редовете и клетките на таблицата ще може да се прави само със стил.

В примера центрирането на HTML таблица за FireFox, Opera и Chrome се прави чрез задаване на стойност auto на атрибутите margin-left и margin-right за HTML таблица с идентификатор id=" tb".

Очевидно вторият метод за подравняване е по-добър, тъй като може да се приложи само за определена таблица, но няма да работи за IE, и поради това за да работи подравняването на всички браузъри трябва да се включат в документа всички дефиниции от примера.

Задаване на отзивчив (Responsive) стил за таблици

Адаптивният (Responsive) стил изисква HTML страницата да има добър вид и тогава, когато се разглежда на устройства с малък размер на екрана, например мобилни телефони и таблети. За да може таблица с голям хори-зонтален размер да се разглежда на такива устройства тя се включва в слой, съдържанието на който може при необходимост да се скоролира, както е показано на фигурата вляво.

```
<div style="overflow-x:auto;">
  <table>
    ...
  </table>
</div>
```

Пример 37 Задаване на адаптивен (Responsive) стил за таблица

14 Мрежа от елементи CSS grid

Мрежата от елементи CSS grid дава възможност за позициониране на елементи (по правило слоеве) по редове и колони в областта на зададен елемент – контейнер. В повечето случаи таблиците, създавани с таговете <table>, <tr> <th> и <td>, и мрежите CSS grid дават един и същ (по вид) резултат, така че е въпрос на избор на разработчика кой от тях ще използва. Счита се, че таблиците са по-удобни при обмен на информация с бази от данни, докато CSS grid предоставят възможност за по-голяма гъвкавост, особено при създаването на адаптивен (responsive) дизайн.

14.1 Структурни елементи на CSS grid

Основните структурни елементи на мрежата CSS grid са:

1. Контейнер на CSS grid (Grid container): Това е основният градивен елемент на CSS мрежата - HTML елементът, в правоъгълната област на който се извежда мрежата от елементи.
2. Мрежова клетка (Grid cell): Това е елемент от мрежа. Всеки елемент може да се позиционира по ред и колона и заема правоъгълна област от мрежата.
3. Област на мрежата (Grid area): съдържа една или повече клетки в област от контейнера с формата на правоъгълник.
4. Пътеки в мрежата (Grid tracks): Това са колекциите от редове и колони, дефинирани с помощта на свойствата grid-template-rows и grid-template-columns. С тези две свойства се дефинира броят и размерите на елементите в мрежата, т.н. „Explicit grid" – дефинирана чрез CSS мрежа от елементи.
5. Интервали между редовете и колоните (Grid gaps): Чрез тях се задава рзстоянието между клетките на мрежата.
6. HTML код на мрежата (Implicit grid): Това е структура в HTML кода на страницата, която се състои от кода на слоя на контейнера, в който са вложени кодовете на елементите на мрежата по правило също слоеве .

14.2Създаване на контейнер за CSS grid

За да може един слой да бъде контейнер на мрежа CSS grid трябва да му се зададе стил display: grid; По правило се дефинира CSS клас, в който освен този стил се задават и стилове, чрез които се дефинират и други свойства на мрежата като брой и разположение на елементите, както е показано в следващият пример:

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto;
  grid-gap: 10px;
  padding: 10px;
}
```

Пример 38 Дефиниция на CSS клас за Grid контейнер

В примера чрез стил grid-template-columns се дефинира броят на колоните на мрежата, чрез стил grid-gap – размерът в пиксели на празните интервали между клетките, а чрез padding – вътрешният празен интервал между контейнера и съдържанието му.

14.3Задаване на брой и размер на колоните – свойство grid-template-columns

Чрез свойството grid-template-columns се дефинират колоните в мрежата CSS grid в следния формат:
grid-template-columns: none|auto|max-content|min-content|length|initial|inherit;
където:

none	Стойност по подразбиране. Колоните се създават при необходимост.
auto	Размерът на колоните се определя от размера на контейнера и от размера на съдържанието на елементите в колоната.
max-content	Задава размера на всяка колона в зависимост от най-големия елемент в колоната.
min-content	Задава размера на всяка колона в зависимост от най-малкия елемент в колоната.
length	Задава размера на колоните, като използва допустима стойност за дължина.
initial	Задава на свойството стойността му по подразбиране.
inherit	Наследява стойността от родителския елемент.

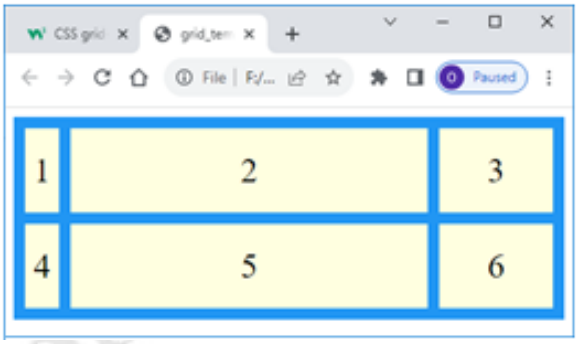
Да разгледаме един пример:

```
<!DOCTYPE html>
<html><head>
<style>
.grid-container {
  display: grid;
  /*задаване на броя и ширината на колоните */
  grid-template-columns: 30px auto 100px;
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: lightyellow;
  text-align: center;
  padding: 20px 0;
```

```
 font-size: 30px;
}
</style>
</head>
<body>
<div class="grid-container">
  <div class="item1">1</div> <div class="item2">2</div>
  <div class="item3">3</div> <div class="item4">4</div>
  <div class="item5">5</div> <div class="item6">6</div>
</div>
</body></html>
```

Пример 39 Създаване на мрежа CSS grid с четири колони и два реда

В примера в CSS кода на класа за контейнера на мрежата grid-container чрез свойството grid-template-columns се дефинира ред с три колони с размери 30px, auto и 100px. Тъй като размерите на първа и трета колона са зададени с числови стойности в пиксели, размерът на втората колона се получава от размера на контейнера, намален с размерите на първа и трета колона. В HTML кода на мрежата в слоя са вложени шест слоя за клетки. Така се получава мрежа CSS grid с три колони и два реда, както е показано на фигурата по-долу:



Фиг. 1 Мрежа CSS grid, получена с кода от пример 39

14.4Задаване на брой и размер на редовете – свойство grid-template-rows

Чрез свойството grid-template-rows се дефинират редовете в мрежата CSS grid в следния формат (изброени са възможните стойности, разделени с „|”)

```
grid-template- rows: none|auto|max-content|min-content|length|initial|inherit;
```

където значението на стойностите е същото, както за свойството grid-template-columns. Чрез свойството grid-template-rows може да се зададе

различна височина на редовете на мрежата, както е показано в следващият пример:.

```
.grid-container {
  display: grid;
  grid-template-columns: 30px auto 100px;
  grid-template-rows: 100px 300px; /*задаване на височина на редовете */
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
```

Пример 40 Задаване на височина на редовете на мрежа чрез свойството grid-template-rows

14.5Обединяване на клетки или задаване име на клетка – свойство grid-area

Свойството grid-area е свойство на клетка от мрежа CSS grid, с което се задава броят позиции по ред и колона, върху които се изобразява клетката или се задава име на клетката. Когато за клетка от мрежата не е зададена стойност на grid-area, тя се извежда на позицията на един ред и колона. Чрез свойството grid-area може да се зададе извеждане на клетката върху няколко реда и колони, т.е. получава се обединяване на позиции в мрежата в една клетка така, като това се прави в HTML таблиците с артибутите rowspan и colspan на клетка <td> или <th>.

Другото приложение на свойството grid-area – задаване на име на клетката, дава възможност за задаване на броят позиции по ред и колона за клетка (т.е. обединяване на позиции) в дефиницията на контейнера на мрежата чрез просто повторение на името на клетката в стойността, която се присвоява на свойството grid-template-areas на контейнера.

Свойството има следният синтаксис:

```
grid-area: grid-row-start / grid-column-start / grid-row-end / grid-column-end | itemname;
```

където:

grid-row-start	Задава началният ред, от който се извежда клетката.
grid-column-start	Задава началната колона, от която се извежда клетката.
grid-row-end	Задава крайният ред, до който се извежда клетката или броят редове, върху които се разполага клетката ако се добави префикс *span*.

grid-column-end	Задава крайната колона, до която се извежда клетката или броят колони, върху които се разполага клетката ако се добави префикс *span*.
itemname	Задава име на клетката

Важно е да се запомни, че с една дефиниция се задават или начални и крайни редове и колони на клетката или име на клетката, т.е. не се задават с една дефиниция заедно и редове/колони и име на клетката.

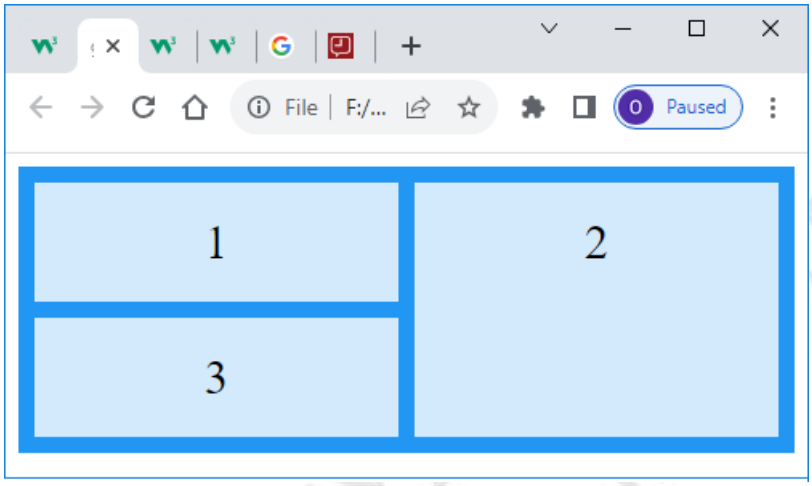
Да разгледаме един пример за използване на свойството grid-area:

```
<!DOCTYPE html>
<html><head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto; /* Задаване на две колони на ред */
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
.item2 {
  grid-area: 1 / 2 / span 2;
}
</style></head>
<body>
<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
</div>
</body></html>
```

Пример 41 Разполагане на клетка на два реда чрез свойство grid-area

В примера чрез свойството „grid-area: 1 /2 /span 2;” се задава разполагане на клетка item2 върху позиция от „първи ред-първа колона” до „втори ред-първа колона” – обединяват се позициите върху два реда чрез „span 2”. Тъй

като не се обединяват позиции от две колони не е необходимо да се добавя в края „/span 1” макар че, ако се добави, няма да се приеме за грешка.



Фиг. 2 Обединяване на позиции върху два реда чрез свойство grid-area

14.6 Дефиниране на области в контейнера на CSS grid – свойство grid-template-areas

Свойството grid-template-areas се използва за дефиниране на области в оформлението на мрежата. Свойството grid-template-areas приема като стойност един или повече символни низове – по един за всеки ред от CSS grid мрежата. Всеки низ (ограден в кавички " или апострофи ') съдържа имена на области, зададени чрез свойството *grid-area* на клетки от мрежата или символи "." (точка) разделени с празни интервали. Символите "." се поставят на тогава, когато за съответните елементи от мрежата няма зададени имена.

Така списъците образуват правоъгълна схема на съдържанието на мрежата, в която за всяка позиция от мрежата има или име на област или символ точка, както е показано в следният пример:

! Важно е да се запомни, че повтарянето на име на област в ред или колона на списъците означава разполагането на елемента в съответният брой позиции на мрежата, т.е. чрез повтарянето на имена на клетки също може да се прави обединяване на позиции в мрежата, както това се прави със свойството grid-area.

```
<!DOCTYPE html>
<html><head>
<style>
.item1 {
  grid-area: myArea;
}
```

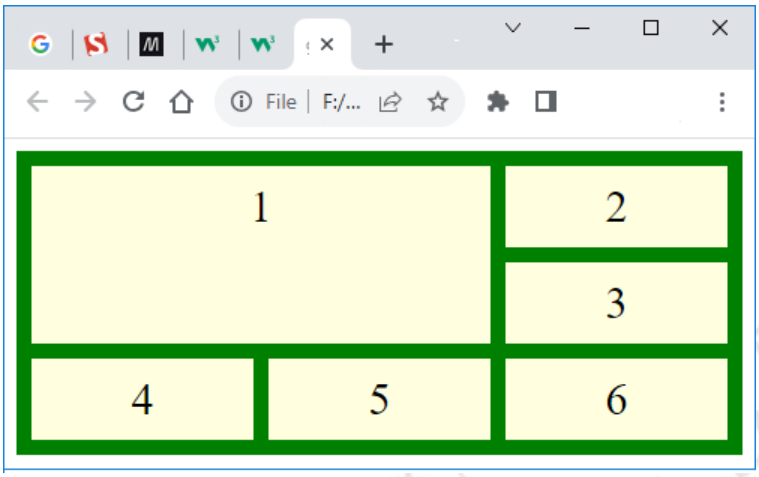
```
.grid-container {
  display: grid;
  grid-template-areas:   /* Задаване на стойност на grid-template-areas
    'myArea myArea .'  /* елементи на първи ред от мрежата */
    'myArea myArea .'; /* елементи на втори ред от мрежата */
  grid-gap: 10px;
  background-color: green;
  padding: 10px;
}
.grid-container > div {
  background-color: lightyellow;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<div class="grid-container">
<!-- HTML код за мрежата -->
  <div class="item1">1</div> <div class="item2">2</div>
  <div class="item3">3</div> <div class="item4">4</div>
  <div class="item5">5</div> <div class="item6">6</div>
</div>
</body></html>
```

Пример 42 Дефиниране на области от CSS grid мрежа чрез grid-template-areas

В примера на свойството grid-template-areas се задават като стойност два символни низа с едно и също съдържание:

```
  grid-template-areas:
    'myArea myArea .'
    'myArea myArea .';
```

като за първите два елемента и от двата реда е зададено едно и също име на област 'myArea. За последните елементи от списъка е зададен символ точка. Резултата от това е, че областта myArea ще заема в мрежата общо четири позиции в първите два реда. На останалите позиции от мрежата се разполагат елементите с CSS класове от .item2 до .item6, както е показано на следващата фигура:



Фиг. 3 Обединяване на позиции върху два реда чрез свойство grid-template-areas

Както се вижда от примера, *не е задължително* в стойността на grid-template-areas да са зададени елементите, които се разполагат на всички позиции от мрежата, но тези, които са зададени, трябва да са на първите позиции. В примера са зададени само елементите, които се разполагат на първите шест позиции (като първият елемент .item1 се разполага на общо четири позиции), а на последните три се разполагат елементите .item4, .item5 и .item6, които се съдържат в HTML кода на мрежата,

```
<div class="grid-container">
  <div class="item1">1</div> <div class="item2">2</div>
  <div class="item3">3</div> <div class="item4">4</div>
  <div class="item5">5</div> <div class="item6">6</div>
</div>
```

но за тях не са зададени позиции в стойността на grid-template-areas. Същият резултат ще да получим и ако добавим в стойността на grid-template-areas символен низ с три точки за елементите от последният ред, както следва:

```
 grid-template-areas:
   'myArea myArea .'
   'myArea myArea .'
   '. . .';
```

15 Формуляри за въвеждане на данни <FORM>

Формулярите дават на потребителя възможност да въвежда и селектира данни и да ги изпраща им към програма, която се изпълнява на Web сървъра. Всички елементи за въвеждане и селектиране на данни на

формуляра се включват в контейнерния етикет <FORM>. Етикетът . <FORM> предоставя атрибути, свързани с изпращането на данни.

15.1 Атрибути на етикета <FORM>

- **action=**”URL”. Задава адреса на сървърната програма, която трябва да получи данните.
- **method=**”GET/POST”>. Вид на предаването на данните:
  - GET: Изпращаните данни се добавят към адреса (URL) след символа „?” като двойки *име=стойност* разделени със символа „&”, зададен с атрибута action.

    Пример:

    ```
    <form method= "get" action="http://index.php?fname=John&age=22 ">
    </form>
    ```

    Методът GET е подходящ за формуляри, съдържащи малък брой обекти, от които се изпращат данни. Друг недостатък на метода е, че изпращаните данни (включително и паролите) се виждат като добавени към URL в адресната лента на браузъра.
  - POST: Този метод не притежава недостатъците на метода GET, поради което се предпочита. Данните се изпращат към сървъра в стандартния изходен поток и поради това няма ограничения за обема им и не се виждат в браузъра.

    Пример:

    ```
    <form method= "post" action="http://index.php">
    <input name="fname" value="John"><input name="age" value="22">
    </form>
    ```
- **target=**”име на фрейм” – Значението на атрибута е както на аналогичния атрибут на етикета за хипервръзка <A>. Той трябва да получи като стойност името на прозореца (фрейма), към който да се насочи отговора от сървърната програма. Валидни са същите запазени имена *SELF/*PERENT/_TOP/_BLANK както за атрибута на етикета <A>.

- **enctype=”**тип на съдържанието на предаваните данни” - Прилага се при зададен метод POST. Възможни са два установени типа, както и дефинирани от потребителя типове.

  application/x-www-form-urlencoded - подразбираща стойност на типа на съдържанието

  multipart/form-data - използва се, ако във формуляра се предават файлове(елемент INPUT с атрибут type=file)

15.2Опционални групи във формулярите

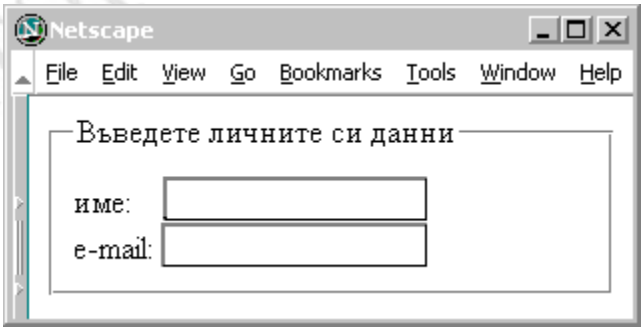
Етикетът <FIELDSET>...</ FIELDSET> - създава опционална група и. чертае тънка рамка около всички HTML формулярни управляващи елементи, намиращи се в указания контейнер.

Етикетът <LEGEND align=top/bottom/left/right>...</LEGEND> - позиционира заглавие на рамката на функционалната група, в която се намира. Елементът трябва да следва непосредствено FIELDSET. Атрибутът align задава позицията на заглавния надпис.

Web дизайнерът може да се откаже от използването на тези етикети изобщо, и вместо тях да използва таблици за позиционирането и групирането на елементите на формуляра, но все пак етикетът <FIELDSET> съвместно с <LEGEND> създават нещо по-различно по вид – рамка, в която е вмъкнат заглавният надпис (фиг. 13).

```
<HTML>
<HEAD>
<META http-equiv="text/html charset=windows-1251">
</HEAD>
<BODY>
<form>
<FIELDSET>
<LEGEND align=top>Въведете личните си данни</LEGEND>
име:    <input type="text" name="ime"><br>
e-mail: <input type="text" name="email">
</ FIELDSET>
</form>
</BODY></HTML>
```

Пример 43 Използване на етикетите <FIELDSET> и <LEGEND> за създаване на опционална група във формуляр.



Фиг. 14 Вид на опционална група, създадена с етикети <FIELDSET> и <LEGEND> в HTML формуляр

15.3Елементи на формулярите

Елементите на формулярите дават възможност за въвеждане или селектиране на данни. Те са програмно достъпни, което означава, че програма, включена като скрипт в HTML документа може да чете, а за повечето елементи и променя данните. Всички елементи на формулярите, от които ще се изпращат данни, трябва задължително да имат атрибут name=”име”, чрез който се създава име на елемента (и обекта, създаден чрез него). Елемент, който няма атрибут name не изпраща данни и не е програмно достъпен, така, че на практика е безполезен. Изключение правят само бутоните от тип “SUBMIT” и “RESET”.

Общи атрибути на елементите:

Елементите на формулярите притежават няколко атрибута, валидни за всички типове елементи. Изключение е само елемента от тип ”hidden”, който не се визуализира,и поради това няма атрибути tabindex, accesskey и title.

- **name**=”име” – Задава име на елемента и създадения чрез него обект. Задължителен за елементите, от коитосе изпращат данни. Името трябва да бъде уникално за всеки елемент и да съдържа само букви (желателно само латински), цифри и долна черта “_”.
- **value**=”стойност” – съдържа стойността на елемента. При зареждане на документа в браузъра, ако е зададен този атрибут, той определя началната стойност на елемента. В последствие операторът може да промени стойността на елемента (в зависимост от типа му) чрез въвеждане или селектиране на данни.
- **tabindex**=”номер”. С атрибута tabindex всички HTML елементи се сортират в определена последователност, като атрибутът приема стойност от 0 до 32767. Какви клавиши ще се използват за същинската навигация зависи от вида на браузъра и операционната система. При ОС Windows за придвижване между елементите се използва бутона за табулация. С клавиш Enter изборът се потвърждава и елементът се активира.
- **accesskey**="символ" – Присвоява на елемента клавиш за достъп. При натикане на бутон “Alt” и зададеният клавиш, елементът от формуляра получава фокус и ако е текстова кутия, в него се установява курсор за въвеждане на данни.
- **title=**"подсказка” – Този атрибут е аналогичен на атрибута title на хипервръзките. Задава подсказка (tooltip) – малко прозорче със зададения текст, което се извежда, когато курсора на мишката се задържи над хипервръзката.

15.4Елементи от тип INPUT

По-голямата част от елементите на формулярите се създават посредством етикета <INPUT>. Това са: текстова кутия, кутия за парола, скрит (hidden) елемент, елемент за изпращане на файл, бутон за маркиране, радио-бутон, бутон за изпращане на данни, бутон за изчистване на формуляра и обикновен бутон. Всички тези елементи притежават общите атрибути за елементите на формулярите, изброени по-горе и освен това един общ атрибут само за елементите от тип input:

Общ атрибут на елементите от тип <INPUT>:

- **type**=”тип на елемента” – Този атрибут определя конкретния елемент от тип <INPUT>, който ще бъде създаден. Ако липсва, се подразбира type=”text” и се създава едноредова текстова кутия.

Допълнителни атрибути на елементите от тип <INPUT> в HTML5

В HTML5 са добавени някои допълнителни атрибути на елементите от тип <input>. Те се поддържат от новите версии на Web браузърите (засега без IE), но не работят по-старите версии. За да проверите дали даден браузър поддържа атрибутите можете да отворите в него тестова страница от адрес http://miketaylr.com/code/input-type-attr.html. Тук ще разгледаме само три от тези атрибути:

- **pattern**=”регулярен израз” – Задава регулярен израз, с който се проверява валидността на въведените данни. (Регулярните изрази се разглеждат в глава втора). Атрибутът pattern може да се използва със следните елементи от тип <input>: text, search, url, tel, email и password.
- **placeholder**=”подсказка” – Задава пояснителен текст, който се извежда в елемента (за разлика от title, който се извежда в отделен прозорец).
- **required** – Ако е включен, указва че в елемента задължително трябва да бъде въведена стойност, за да могат данните от формуляра да бъдат изпратени. Атрибутът required може да се използва със следните елементи от тип <input>: text, search, url, tel, email, password, date pickers, number, checkbox, radio и file.

15.5Едноредова текстова кутия, TYPE=”TEXT”

Това е елемент от тип <INPUT>, който служи за въвеждане на текст. Текстът, който се извежда от текстовата кутия, се съдържа в атрибута value.

В съответствие с предназначението си елементът има някои специфични атрибути:

- **size**=”дължина” – задава дължината на текстовата кутия в брой символи.
- **maxlength**=”макс. брой символи” – задава максималният брой символи, които могат да бъдат въведени в текстовата кутия. След въвеждане на зададения брой символи текстовата кутия престава да приема символи за добавяне към въведеното.
- **readonly (само за четене) –** Забранява въвеждането на данни от потребителя. При включването му текстовата кутия не може да получи фокус, и като резултат в нея не може да се установи курсора за въвеждане на данни. Това обаче не забранява програмното записване на данни в текстовата кутия.

15.6Текстова кутия за парола, TYPE=”PASSWORD”

Този елемент е има същия вид и същите атрибути както обикновената текстова кутия, с тази разлика, че при извеждане на текста той се изобразява със “*”. Ако обаче въведените данни се изпращат към сървърната програма с метода GET, в адресната лента на браузъра ще се види действително въведения текст (в съответната двойка име-стойност на елемент). Това е един от недостатъците на метода GET, поради което се препоръчва използването на метода POST.

15.7Елемент за маркиране, TYPE=”CHECKBOX”

Този елемент извежда в прозореца на браузъра поле, в което може да се постави отметка за маркиране. Това дава възможност на попълващият формуляра да отговори положително или отрицателно на поставен въпрос, да потвърди или отхвърли предлаган отговор. В съответствие с предназначението си елементът има един специфичен атрибут

Атрибут:

- **checked** – Включването на този атрибут в етикета показва, че при начално зареждане на HTML страницата елементът ще бъде маркиран. Ако при изпращане на данните от формуляра елементът е маркиран, от него ще се изпрати съдържанието на атрибута **value**[*], в противен случай от елемента няма да се изпрати нищо.

[*] Нека напомним отново, че всъщност винаги се изпраща двойка име_на_елемент=стойност_на_елемент, а не само стойност.

15.8 Радиобутон, TYPE=”RADIO”

Понякога се налага да дадем на потребителя възможност да избира само един от няколко възможни отговори. За тази цел служат радиобутоните. Наименованието на тези елементи идва от бутоните за задаване на честотен обхват на радиоапаратите – в даден момент може да бъде натиснат само един от тях. Радиобутонът, както и обекта за селектиране, извежда в прозореца на браузъра поле, в което може да се постави отметка за маркиране. Група от радиобутони, в която селектирането на един елемент изключва останалите, се образува само, ако няколко радиобутона имат едно и също име (една и съща стойност на атрибута **name**). Такава група дава възможност на попълващият формуляра да маркира само един елемент от групата. Радиобутоните, както и елементът за селектиране, имат само един специфичен атрибут, който показва, дали елементът от групата е маркиран или не.

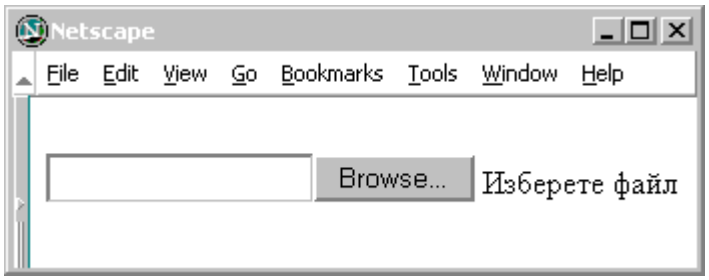
Атрибут:

- **checked** – Включването на този атрибут в етикета показва, че при начално зареждане на HTML страницата елементът ще бъде маркиран. При изпращане на данните от формуляра от дадена група ще се изпрати съдържанието на атрибута value само на този елемент, който е маркиран.

15.9 Елемент за изпращане на файл, TYPE=”FILE”

HTML 4.0 и по-новите версии дават възможност чрез формуляр да се изпрати към сървърната програма съдържанието на файл от компютъра на потребителя. Това може да се види например в сайтове за администриране на (безплатен)сайт на потребител. Потребителят може да записва файлове в своя сайт като ги изпраща чрез елемент FILE от формуляр от предоставената му страница за потребител. (при това по правило не пропускат да му напомнят: Ъпгрейтвайте за платена услуга “for only $4.99 per month” и ще ви бъде разрешена FTP* връзка.). Но да се върнем на елемента FILE. Той се извежда като текстова кутия и вдясно от нея бутон с надпис “Browse”.

* File Transfer Protocol – Прехвърлянето на файлове на сървър при разрешена FTP връзка става със специализирана програма – FTP клиент.



Фиг. 15 Вид на елемент FILE в прозореца на браузъра

При щракване върху бутона Browse се извежда стандартната диалогова кутия за избор на файл. Когато потребителят избере файл и потвърди с бутона “ОК”, в текстовата кутия се извежда пълната файлова спецификация на избрания файл (напр. C:/MySite/home.htm). При изпращане на данните от формуляра, съдържанието на избрания файл се изпраща заедно с другите данни от формуляра. В тоя случай е необходимо данните от формуляра да се изпращат обязателно по метода POST.

Елементът FILE има същите атрибути, както елементите TEXT и PASWORD (тъй като включва текстова кутия). Има обаче и някои особености – тъй като елементът дава достъп до файловата система на потребителския компютър (макар и само за четене), са взети мерки това да не може да става автоматично (без действие на потребителя). Така че дори и да се зададе първоначална стойност на атрибута value, тя не се приема – текстовата кутия се извежда като празна. По същите причини е забранен програмният запис на данни в атрибута **value**.

Атрибути:

- **size**=”дължина” – задава дължината на текстовата кутия в брой символи.

value – връща спецификацията на избрания файл. От съображения за сигурност от версия 8 на IE атрибутът връща фиктивен път и името и разширението на избрания файл, напр. ако файла е filename.ext атрибутът връща "C:\fakefile\filename.ext"

15.10 Бутон за изпращане на данни, TYPE=”SUBMIT”

Този елемент е специализиран бутон, при щракване върху който се предизвиква изпращане на данни от формуляра на адрес, зададен в атрибута action на етикета <FORM>. Като надпис на бутона се извежда съдържанието на атрибута value. За разлика от елементите, чието съдържание се изпраща като данни, на този елемент (както и на бутоните от тип RESET и IMAGE) не е необходимо да се задава атрибут name. Ако все пак зададем стойност на атрибут name, тогава от бутона също ще се изпрати двойка име-стойност, която сървърната програма може да игнорира.

15.11 Бутон за изпращане на данни, TYPE=”IMAGE”

Този елемент е също бутон за изпращане на данни от формуляра на адрес, зададен в атрибута action на етикета <FORM>. Разликата между него и бутона от тип SUBMIT е в това, че вместо текстов надпис този бутон извежда изображение. Адреса (URL) на изображението се задава като стойност на атрибута src на бутона.

Атрибути:

- **src** – задава адреса (URL) на файла, който съдържа изображението за бутона
- **width, height** – задават хоризонталния и вертикален размер на бутона в пиксели. Това са нови атрибути, въведени в HTML5, които се поддържат само от по-новите версии на браузърите.

15.12 Бутон за изчистване на формуляра, TYPE=”RESET”

Този елемент е специализиран бутон, при щракване върху който се изчистват въведените данни в елементите на формуляра и се възстановяват началните стойности (ако има зададени). Така например ако една текстова кутия има зададен атрибут value=”John”, то при щракване върху бутона RESET в нея се възстановява това съдържание, в противенслучай се изчиства въведеното. Надписа върху бутона се задава с атрибута value.

15.13 Обикновен бутон, TYPE=”BUTTON”

Това е бутон, който не е специализиран и нищо не върши ако не му се добави манипулатор на събитие. Той притежава същите атрибути, както останалите бутони с текстов надпис (това са общите атрибути за всички елементи от тип INPUT).

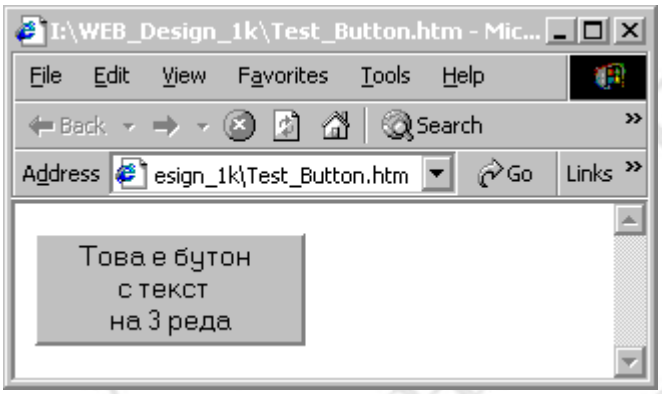
Интересна подробност, за която може би няма да се досетите веднага, е че надписите върху бутоните могат да бъдат на няколко реда, а не обязателно на един, както обикновено се прави. Достатъчно е текстът, който се задава като стойност на атрибута value да бъде записан на няколко реда. Ето един пример:

```
<HTML>
<HEAD>
<META http-equiv="text/html charset=windows-1251">
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button"
```

```
 VALUE="Това е бутон&#13;&#10;с текст &#13;&#10;на 3 реда">
</FORM>
</BODY>
</HTML>
```

Пример 44 – HTML бутон с текст на няколко реда

Ето и резултата:



Фиг. 16 HTML бутон с текст на няколко реда

Може би няма да ви хареса това, че текста е центриран, а не ляво подравнен. За съжаление, етикета INPUT няма атрибут, с който да се задава подравняването на текста. Но в HTML това, което не може да се прави по класическия начин се прави със стил. Можете да пробвате кода, даден в следващия пример – ще се изведе бутон с ляво подравняване на текста.

```
<HTML>
<HEAD>
<META http-equiv="text/html charset=windows-1251">
</HEAD>
<BODY>
<FORM>
<INPUT style="text-align:left"  TYPE="button"
 VALUE="Това е бутон&#13;&#10;с текст &#13;&#10;на 3 реда">
</FORM>
</BODY>
</HTML>
```

Пример 45 HTML бутон с ляво подравнен текст на няколко реда

15.14 Скрит елемент, TYPE=”HIDDEN”

Както се вижда от наименованието му, този елемент не се изобразява* в прозореца на браузъра. Преджазначението му е да изпраща данните, записани в атрибута cvalue на елемента, без те да се виждат от потребителя. За този елемент важи общото правило за елементите на формулярите – за да се изпратят данни от елемента, е необходимо той да има зададено име, т.е. да е зададена стойност на атрибута му name.

15.15 Нови <INPUT> елементи в HTML5

В HTML5 са въведени няколко нови <input> елемента, които са специализирани за въвеждане на определен тип данни. Тук ще разгледаме четири от тях, които считаме за най-полезни.

- type=”email” за въвеждане или избор на email адрес. Без зададена стойност на атрибута list служи за въвеждане на email адрес и извежда съобщениеако не е валиден. Ако се зададе като стойност на атрибута му list обект (елемент) datalist, служи за избор на адрес от списъка.
- type=”url” за въвеждане или избор на интернет адрес. Без зададена стойност на атрибута list служи за въвеждане на интернет адрес и извежда съобщениеако не е валиден. Ако се зададе като стойност на атрибута му list обект (елемент) datalist, служи за избор на адрес от списъка.
- type=”date” за въвеждане или избор на дата. Различните браузъри по различен начин улесняват оператора при въвеждане на дата. Например Opera извежда прозорец с календар за избор на датата.
- type=”number” за въвеждане на числа. Браузърите, които поддържат този елемент извеждат до текстовата кутия и бутони за инкремент/декремент (spinner) за постъпково изменение на числото.

Както беше посочено при разглеждане на новите HTML5 атрибути на елементите, проверка за валидност на въведените в елемент <input> данни може да се извършва с регулярен израз, зададен като стойност на атрибута му pattern.

---

* Не се “рендва”, както беше казал един студент. Често може да се чуе подобен българо-английски жаргон, който вероятно показва недостатъчно използване на документация на български език.

Елементи на формулярите, които не са от тип INPUT

Има само два елемента на формулярите, които не са от тип INPUT. Това са многоредовата текстова кутия TEXTAREA и елементът за селектиране от списък SELECT.

15.16 Многоредова текстова кутия TEXTAREA

Както се вижда от наименованието и, тази текстова кутия може да извежда текст на няколко реда. Тя се създава с контейнерния етикет <TEXTAREA>…</TEXTAREA>. Всичко, което е разположено между отварящата част <TEXTAREA> и затварящата част </TEXTAREA> се извежда като съдържание на текстовата кутия. Например текстовата кутия от следващия пример

```
<TEXTAREA name=”T1” rows=20 cols=”80”>
</TEXTAREA>
```

Пример 46 Многоредова текстова кутия, която първоначално съдържа .един празен ред.

може да изглежда без начално съдържание, но всъщност съдържа един празен текстов ред, защото отварящата част <TEXTAREA> и затварящата част </TEXTAREA> са на различни редове в кода.

Браузърите извеждат съдържанието на многоредовата текстова кутия като обикновен текст, което означава, че дори и между отварящата и затваряща част на TEXTAREA да има записани HTML етикети, те ще се изведат като обикновен текст.

```
<HTML>
<BODY>
<form><textarea rows="5" cols="40">
<H1>Hello</H1>
<BR><a href="http://www.dir.bg>"Dir.bg</a>
</textarea></form>
</BODY>
</HTML>
```

Пример 47 Многоредова текстова кутия, която съдържа HTML код

Фигура 17 Многоредова текстова кутия, която съдържа HTML код

Елементът TEXTAREA притежава общите атрибути **name**, **value, title, tabindex** и **accesskey** и няколко специфични атрибута за многоредовите текстови кутии.

Специфични атрибути:

- **rows=**”брой редове” – Задава броя редове, които може да изобрази текстовата кутия. Този атрибут на практика определя вертикалния размер на изобразяваната текстова кутия. Ако текстът в нея съдържа повече редове от зададените в rows, се изобразява вертикална лента за скролиране, с която да се превърта текста във вертикална посока.
- **cols=**”брой колони” – Задава броя символи на ред, които може да изобрази текстовата кутия. Този атрибут определя хоризонталния размер на изобразяваната текстова кутия. Ако текстът в нея съдържа повече символи на ред от зададените в cols, се изобразява хоризонтална лента за скролиране.
- **readonly (само за четене) –** Забранява въвеждането на данни от потребителя. Аналогичен на атрибута за едноредова текстова кутия. При включването му текстовата кутия не може да получи фокус, и като резултат в нея не може да се установи курсора за въвеждане на данни. Това обаче не забранява програмното записване на данни в текстовата кутия.

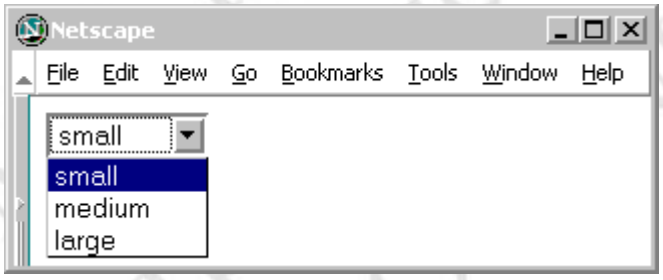
15.17 Меню/Списък от опции SELECT

Елементът SELECT извежда падащо меню или списък, от който потребителят може да селектира един или няколко реда (опции). Елементът се създава от контейнерния етикет <SELECT> , в който за всеки ред от менюто/списъка се влага по един етикет <OPTION>…</OPTION>. Елементът <OPTION> съдържа текста, който ще се появи на поредния ред

от списъка. Да разгледаме пример на елемент SELECT, който извежда падащ списък

```
<HTML><BODY>
<FORM>
<SELECT NAME="dimension">
<OPTION VALUE="s">small </OPTION>
<OPTION VALUE="m">medium </OPTION>
<OPTION VALUE="l">large </OPTION>
</SELECT>
</FORM>
</BODY></HTML>
```

Пример 48 Елемент SELECT, който извежда падащ списък



Фигура 18 Вид на падащ списък, изведен от елемент SELECT.

При извеждане на меню, елементът извежда само един ред, а падащия списък се появява при щракване с мишката върху бутона-стрелка. Както се вижда от примера, етикетите за редовете от списъка имат затваряща част. Браузърите реално не се нуждаят от затваряща част на тези етикети, тъй като края на текста се определя еднозначно от следващия етикет <OPTION> или от затварящата част на етикета </SELECT>. Въпреки това е желателно те да се включват в кода.

Елементът **<SELECT>** притежава общите атрибути **name**, **value, tytle, tabindex** и **accesskey** и няколко специфични атрибута за списъчните елементи.

Специфични атрибути на елемента <SELECT>:

- **multiple** – Задава извеждане на списък с възможност за селектиране на повече от един ред. Ако този атрибут липсва, се извежда падащ списък с възможност за селектиране само на един ред, както в горния пример.
- **size=**”брой видими редове в списъка” – Задава броя редове, които може да изобразят, като по този начин определя вертикалния размер на изобразявания списък. Ако списъкът съдържа повече редове от

зададените в size , се изобразява вертикална лента за скролиране.При задаване на стойност на атрибута size винаги се получава списък, а не падащо меню. Иаползването size без multiply дава възможност да се получи списък с възможност за селектиране само на един ред.

Атрибути на етикета <OPTION>

Етикета <OPTION>, с който се включва ред (опция) в списъка, има два атрибута – value и selected

- **value=**”стойност” – Задава стойността, която се изпраща от менюто/списъка. Ако при изпращане на данни от формуляра редът от менюто/списъка е селектиран, от негосе изпраща стойността, зададена във value. Атрибутът не е задължителен. Ако за селектирания ред няма зададен атрибут value, се изпраща текста на реда – текста, съответстващ на елемента <OPTION>.
- **selected** – Ако този атрибут е включен в етикета <OPTION>, то при начално зареждане на HTML документа в прозореца на браузъра съответния ред ще бъде селектиран. Ако се извежда меню, то реда, за който има атрибут selected (трябва да бъде само един) ще бъде избран и ще се вижда до бутона на менюто. Ако се извежда списък, всички редове, за които има атрибут selected ще се изведат като селектирани.

16 Адаптивен (Responsive) Web дизайн

Адаптивният Web дизайн (RWD) е технология за създаване на Web страници, които да изглеждат добре на всички устройства. Web страниците могат да се разглеждат с помощта на много различни устройства: настолни компютри, таблети и телефони. Адаптивният Web дизайн дава възможност за създаване на HTML страници, които да изглеждат добре на всички устройства.

Важно е да се отбележи, че адаптивният Web дизайн не използва код на JavaScript, а само HTML и CSS.

16.1 Meta тагът Viewport

Прозорецът на браузъра е областта на прозореца, в която може да се разглежда Web съдържание. За устройствата с голям екран размерът на прозореца на браузъра по правило съвпада с размера на екрана. За мобилните устройства обаче често прозорецът на браузъра се извежда в т.н. виртуален прозорец, който е с по-голям размер от екрана на устройството.

При извеждане на страницата във виртуален прозорец браузърът предоставя ленти за превъртане, за да може потребителят да превърта и да има достъп до цялото съдържание. Например, ако мобилен екран има ширина от 640px, страниците може да бъдат изобразени във виртуален прозорец за ширина от 980px и след това той може да бъде скролиран или свит, за да се побере в пространството от 640px. Това се прави за да може сайтовете, които не са оптимизирани за мобилни устройства, като цяло да изглеждат по-добре на устройства с тесен екран.

Този механизъм обаче не е толкова добър за страници, които са оптимизирани за тесни екрани с помощта на медийни заявки – ако виртуалният изглед е 980px например, медийни заявки, които стартират на 640px или 480px или по-малко, никога няма да бъдат използвани, ограничавайки ефективността на такива адаптивни дизайнерски техники. Мета тагът viewport дава възможност за по-добри решения на проблема с виртуалния прозорец viewport на устройства с тесен екран.

Мета тагът viewport, включен в HTML5, дава възможност на дизайнера да контролира параметрите на виртуалният прозорец на устройствата с тесен екран. При създаване на сайтове с използване на технологията „Адаптивен (Responsive) Web дизайн" в началото на всяка HTML страница се поставя мета таг viewport, по правило със следното съдържание:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

По-долу ще разгледаме по-подробно атрибутите на тага viewport

width: Задава ширината на прозореца за изглед. Стойността му може да бъде зададена за конкретен брой пиксели като width=600 или специалната стойност device-width, която е 100vw или 100% от ширината на прозореца на устройството. Минимум: 1. Максимум: 10000.

height: Задава височината на прозореца за изглед. Стойността му може да бъде зададена за конкретен брой пиксели като height=600 или специалната стойност device-height, която е 100vh или 100% от височината на прозореца на устройството. Минимум: 1. Максимум: 10000.

initial-scale: Задава коефициентът на мащабиране при началното зареждане на страницата. Минимум: 0,1. Максимум: 10. По подразбиране: 1.

16.2CSS медийни заявки (Media Queries)

Медийните заявки Media queries са CSS техника, въведена в CSS3. CSS медийната заявка използва правилото (конструкцията) @media, за да включи блок от CSS стилове, които се прилагат само ако зададено условие е вярно.

Синтаксис:

```
@media(media_type(media_feature) [operator media_type(media_feature) ....])
```

където:

media_type е вид устройство, например display, screen, print

media_feature е свойство на устройството със зададена стойност, напр. max-width=600px за устройство screen означава че кодът, включен в конструкцията @media ще се изпълни ако ширината на екрана не превишава 600 пиксела.

operator е логически оператор *and, or* или *not* или клауза *only*. Логическите оператори or (ИЛИ) и not (И) се използват тогава, когато трябва да с обединят условията, задавани с две или повече свойства media_feature. С клаузата only (само) се уточнява, че свойството с еотнася само за определено устройство.

Да разгледаме един пример:

```
<HTML><HEAD>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
@media only screen and (max-width: 600px) {
  body {
    background-color: lightyellow;
  }
}
</style></HEAD><BODY>
Hello Responsive HTML
</BODY></HTML>
```

Пример 49 Медийна заявка, с която се задава светложълт фонов цвят при ширина на екрана не по-голяма от 600 пиксела

В примера в медийната заявка са зададени условията „only screen” и „(max-width: 600px)”, които са обединени с логическа операция „and” (И). Резултата от изпълнението на медийната заявка е че когато призорецът на браузъра е с дължина не по-голяма от 600 пиксела за прозореца се прилага стилът „background-color: lightyellow;” и фоновият цвят на прозореца става светложълт.

Важно е да се запомни, че CSS медийните заявки трябва да се включат в блок <style>. Медийни заявки могат да се използват и в код на JavaScript, но това изисква доста по-сложен код за постигане на същите резултати.

Онлайн източници:

https://developer.mozilla.org/en-US/docs/Web/HTML/Viewport_meta_tag Viewport meta tag

https://www.w3schools.com/css/css_rwd_mediaqueries.asp Media Queries

16.3 Адаптивен табличен изглед (Responsive Grid-View)

Повечето Web страници се използват таблично (мрежово) оформление за подреждане на елементите, което означава, че страницата е разделена на редове и колони. С помощта на това мрежово оформление можем лесно и ефективно да позиционираме елементите на страницата в редове и колони. Адаптивният табличен изглед Grid-View ни позволява да разделим всеки ред на зададен брой колони (от една до дванадесет) с максимална обща ширина 100%. Когато променим размера екрана на браузъра, тези елементи на мрежата се свиват и разширяват според размера на екрана.

16.3.1 Създаване на адаптивен табличен изглед чрез CSS

За да създадем адаптивен мрежов изглед, трябва да зададем свойството box-sizing със стойност border-box за всички елементи в една уеб страница. Това ще включва външната (margin) и вътрешната (padding) рамка на елемента във височината и ширината на елемента. За целта в блока <style> на страницата се включва следният CSS код:

```
* {
      box-sizing: border-box;
}
```

При задаването на ширината на колоните във всеки един ред общата им дължина трябва да бъде 100%. Така ако в един ред трябва да има 12 колони, ширината на всяка една от тях трябва да бъде 100/12 % = 8.33% . Това може да бъде зададено с декларация на стил за CSS клас както следва:

```
.col-1 {width: 8.33%;}
```

Аналогично за всеки брой от колони n ($1 \leq n \leq 12$) можем да получим стойността на ширината на една колона по формулата width = 100/n %. За да имаме на разположение всички възможни ширини на колони можем да включим в блока <style> на HTML страницата следният CSS код:

```
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
```

```
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

Пример 50 CSS код за задаване на ширини на колони за адаптивен табличен изглед (Responsive Grid-View)

В адаптивният табличен изглед колоните вътре в реда се подравняват вляво една до друга посредством стила „float: left;”. Стилът може да се приложи за всички класове за колони, включени в предният пример със следният CSS

```
[class*="col-"] {
    float: left;
    padding: 15px;
    font-size:18px;
}
```

Пример 51 Декларация на стил за елементи с атрибут **class**, чието име съдържа низа "col-"

При такова подравняване ако редът съдържа допълнителни елементи те може да се изобразят върху колоните, за да се избегне това, в CSS за слоя за реда се прилага CSS код на клас както следва:

```
.row::after {
    content: ""; /* Добавя празен низ след кода на реда */
    clear: both; /* Указва, че трябва да се извеждат само колоните */
    display: table; /* Указва елементът да се извежда като таблица */
}
```

Да разгледаме един пример:

```
<html><head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}

.header, .menu, .main{
  border: 1px solid blue;
```

```
  padding: 15px;
}
.menu {
  width: 25%;
  float: left;
}
.main {
  width: 75%;
  float: left;
}
</style>
</head>
<body>
<div class="header">
  <h1>Responsive Web Design</h1>
</div>
<div class="menu">
  <ul>
    <li>Meta tag Viewport</li>
    <li>Media Queries</li>
    <li>CSS Responsive Grid</li>
  </ul>
</div>
<div class="main">
  <h1>CSS Responsive Grid</h1>
  <p>A responsive grid-view often has 12 columns, and has a total width of 100%, and will shrink and expand as you resize the browser window..</p>
</div>
</body></html>
```

Пример 52 Код на HTML страница, която извежда адаптивна таблица (Responsive Grid)

В примера страницата съдържа три слоя , за които са приложени стилове с CSS класове header, menu и main. За слоевете с класове menu и main са зададени ширина съответно 25% и 75% и стил „float: left;”, в резултат на което те се извеждат като колони на адаптивна (responsive grid) таблица. При намаляване на размера на прозореца ширините на двата слоя се намаляват пропорционално като се запазва съотношението 75/25%. Ако обаче в блока <style> на страницата се добави медийна заявка:

```
@media only screen and (max-width: 600px) {
  .menu, .main {
    width:100%;
```

```
    clear: both;
  }
}
```

Пример 53 Медийна заявка за подреждане на колоните на адаптивна таблица една под друга

В примера когато размерът на прозореца на браузъра е под 600 пиксела за колоните от таблицата се задава ширина „width:100%;” в резултат на което всяка колона заема цялата ширина на прозореца и така колоните от таблицата се оказват една под друга. Стилът „clear: both;” в случая би могъл и да се пропусне – той указва, че на редовете на колоните няма да се извежда друго HTML съдържание.

Онлайн източници:

https://www.w3schools.com/css/css_rwd_grid.asp Responsive Web Design - Grid-View

https://blog.logrocket.com/responsive-image-gallery-css-flexbox/ How to create a responsive image gallery with CSS flexbox

17 Глава II – JavaScript

Какво представлява JavaScript

JavaScript е компактен, обектно ориентиран скриптов* език за разработване на интернет приложения [10]. Той е създаден от Netscape Communications Corporation през 1995 година под името LiveScript, но в последствие при усъвършенстването му съвместно с фирмата Sun Microsystems езикът получава името JavaScript. JavaScript е подобен на Java – език за програмиране, разработен от Sun Microsystems, но в сравнение с него е значително опростен. JavaScript е изключително полезен за WEB дизайнера. Той дава възможност за създаване на динамични страници, които реагират на действията на потребителя като променят вида и съдържанието си при възникване на определени събития – начално зареждане на документа, натискане на бутон от клавиатурата, операции с мишката, затваряне на прозореца на браузъра и др.

18 JavaScript и Web браузърите

Съвременните браузъри поддържат JavaScript като подразбиращ се език за включване на програмен код в HTML документите. Когато браузърът изпрати заявка за HTML документ, Web сървърът изпраща като отговор пълното съдържание на документа, в това число и кода на JavaScript, включен в него. При обработването на документа браузърът създава обекти, съответстващи на HTML елементите в съответствие със стандартизираният обектен модел на HTML документ (DOM). При възникването на определени събития Web браузърът изпълнява кода (командите) на съответните събитийни процедури, ако са включени в документа. Програмите на JavaScript, вложени в HTML документите се изпълняват от Web браузъра, който в случая работи като интерпретатор – чете текста на програмата, преобразува го в команди на микропроцесора и го изпълнява (т.е. стартира неговото изпълнение и осигурява необходимата среда за това). За разлика от компилаторите, които създават изпълнима програма за многократно изпълнение, интерпретаторите, какъвто е в случая Web браузърът, всеки път, когато трябва да изпълнят даден скрипт,

* Скриптовите езици се определят като езици за писане на сценарии. Сценарият (скриптът) е програма, която автоматизира операции, които в противен случай, без скрипта, потребителят може да изпълни ръчно, като използва интерфейса на програмата.

изпълняват наново цялата последователност от операции – четене на текста на програмата, преобразуване в команди и изпълнение*.

19 Основни понятия в програмирането

19.1 Алгоритми

Всеки език за програмиране предоставя на програмиста краен брой оператори за извършване на аритметични, логически, символни и други операции. Въоръжен с тях и със средствата, които му предоставя програмната среда (библиотеки от функции, класове, предварително дефинирани програмни елементи и др.) програмистът трябва да реши поставената задача. Задължителен етап от разработването на компютърна програма е съставянето на нейния алгоритъм - определянето на необходимата последователност от операции, която да решава поставената задача.

Известни са няколко формални определения на понятието алгоритъм. Такова определение са дали А. М. Тюринг, А. А. Марков, Джон фон Нойман и др. С отчитане на спецификата на задачите за разработване на програми можем да приведем следното определение за алгоритъм [9], [10]:

D1 Алгоритъмът е последователност от правила и инструкции, предписващи определени действия със зададено множество данни с цел получаване на резултат – решение.

Алгоритмите трябва да притежават следните свойства:

1. Дискретност – алгоритъмът се състои от безкраен или краен брой стъпки. Даден алгоритъм може да се реализира като компютърна програма само ако се състои от краен брой стъпки.
2. Детерминираност – резултатите, получавани на дадена стъпка еднозначно се определят от резултатите от предишните стъпки.
3. Резултатност – използването на разработения алгоритъм за дадения клас от задачи трябва винаги да води до резултат.
4. Масовост – разработеният алгоритъм трябва да се отнася за всички задачи от даден клас, а не само за частни случаи.

Използват се различни начини за представяне на алгоритмите:

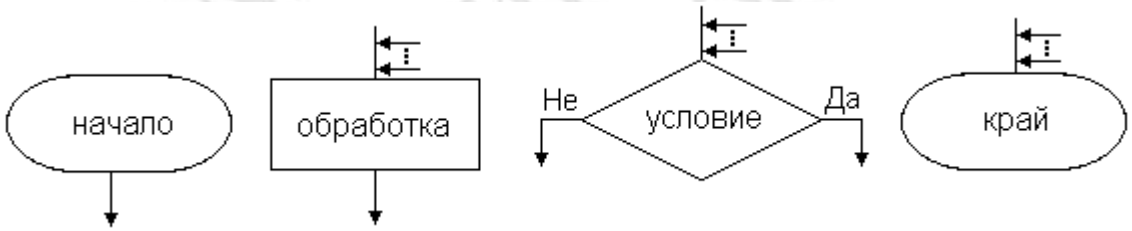
- описателен (на естествен език);

* Има някои изключения от това правило, например при използването на функции, но разглеждането им излиза извън рамките на курса по Web дизайн.

- ❖ с блокови схеми
- ❖ със специализиран език

Обикновено алгоритъмът на естествен език се описва по точки. В точките се отразява и управлението на изпълнението на алгоритъма – проверката на условия и в зависимост от резултата изпълнението на една или друга последователност от операции.

Блоковата схема на даден алгоритъм е ориентиран граф, състоящ се от възли и дъги. Ориентираните дъги свързват възлите на графа и така показват последователността на изпълнение на операциите. За изобразяване на алгоритъма се използва определено минимално множество от блокове като възли. Множеството трябва да бъде функционално пълно, за да може с него да се изобразява всеки алгоритъм, независимо от неговата сложност. Такова множество е показано на фиг. 1. То се състои от елементарните блокове за начало и край, блок за действие (обработка на информацията) и блок за разклонение (логически блок). В блока за разклонение се изчислява израз, който връща логически резултат истина (true) или неистина (false) и в зависимост от това изпълнението на алгоритъма продължава по клона, обозначен с ДА или с НЕ.



Фиг. 4 Блокове за изобразяване на алгоритми

Освен тези блокове, за по-компактно представяне на алгоритмите се използват и други блокове, като напр. блок за многократно условно разклонение, блок за подпрограма (модул), блок за преход между страници и т.н. , и освен това програмистът, който съставя блоковата схема може да представи алгоритъма с различна степен на детайлизация. Така например един блок за обработка може да представя само една програмна операция, но може да представя и цял модул от програмата, за който има известен алгоритъм или известна функция от библиотека.

В структурно-функционално отношение алгоритмите биват:

- неразклонени
- разклонени
- циклични

Неразклонени (линейни) са тези алгоритми, при които независимо от това какви входни данни са въведени, винаги се изпълняват едни и същи

действия. Блоковите схеми, които представят линейни алгоритми не съдържат блокове за условно разклонение.

Разклонени са тези алгоритми, които съдържат логически условия, в зависимост от които се изпълнява една или друга последователност от действия. В структурно отношение това означава, че в зависимост от логическите условия се преминава по едни или други клонове от графа, който представя изчислителния процес. Блоковата схема, които представя разклонен алгоритъм трябва да съдържа поне един блок за условно разклонение.

Цикъл се нарича многократно (повтарящо се) изпълнение на една и съща част от алгоритъма, като при всяко ново изпълнение се обработват нови данни.

Циклични са алгоритмите, който съдържат един или повече цикли. Всеки цикличен алгоритъм е и разклонен, защото трябва да съдържа логически условия за продължаване или прекратяване на всеки от съдържащите се в него цикли.

Според мястото на проверката на условието за продължаване на цикъла (може да се проверява и алтернативното условие за край на цикъла) циклите биват два вида:

- с префиксна проверка – тялото на цикъла започва с проверка на условието за продължаване или край на цикъла.
- с постфиксна проверка– тялото на цикъла завършва с проверка на условието за продължаване или край на цикъла.

Разликата е доста съществена – тялото на цикъла с префиксна проверка може да не се изпълни нито веднъж ако още при първата проверка се изпълни условието за край на цикъла. Ако цикълът е с постфиксна проверка, тялото на цикъла ще се изпълни поне веднъж, тъй като проверката е в края и едва след изпълнение на тялото на цикъла той може да завърши. Разликата между двата типа организация на цикъла се проявява само ако е възможно при първата проверка да не се изпълнява условието за продължаване на цикъла. Общата блокова схема на цикъл с постфиксна проверка е показана на Фиг. 9 на стр. 101.

20 Основни елементи на JavaScript

Създаването на програми на кой да е език за програмиране изисква познаване и правилно използване на елементите на езика – символи, ключови думи, константи, променливи, функции, обекти, коментари. Изучаването на тези програмни елементи, тяхното дефиниране и

използване, правилата (синтаксиса) за записване на командите и структурата на програмите са начален етап за изучаване на всеки език за програмиране. Ще разгледаме последователно тези елементи и изисквания.

20.1Константи

Всяка програма съдържа постоянна информация, която не се променя по време на нейното изпълнение. Такъв тип информация се представя посредством константи.

D2.Константата е величина, която не може да променя стойността си по време на изпълнение на програмата.и се определя от елементите {тип, стойност}.

В JavaScript константите се включват в самите команди. Ето един пример за константа, включена в команда:

```
x= 3.1416;
```

С тази команда в променливата x се записва числото 3.1416, което, както сте се досетили е константата π, зададена с точност до четвъртия знак след десетичната точка. Независимо от това колко пъти ще се изпълни тази команда в програмата, резултатът винаги ще бъде един и същ, тъй като вдясно от оператора за присвояване "=" стои константата 3.1416.

Използване на подобни константи многократно в текста на програмите се счита за лош стил на програмиране. По-добре е константите, които имат някакво специално значение за програмата да се присвоят на променливи в началото на програмата, и след това вместо тях да се използват съответните променливи. За съжаление, JavaScript (до настоящата поддържана от браузърите версия 1.3), за разлика от повечето езици за програмиране, не дава възможност за деклариране на константи с имена. Във версия JavaScript 1.5, вече е включена възможност за деклариране на константи с ключовата дума **const** , но тя не се поддържа от всички браузъри и поради това не трябва да се използва (напр. IE9 я приема за синтактична грешка).

В програмите на JavaScript се използват три типа константи – числови (целочислени и дробни), символни и логически

- Целочислени – могат да бъдат в десетична, осмична и шестнадесетична бройна система:
  - Десетични – започват с цифра, различна от 0, например 15
  - Осмични – започват с 0, следвани от цифри от 0 до7 за разрядите на числото, например 017. Преобразувана в десетична, тази константа има стойност $1*8^1+7*8^0=15_{10}$.

- Шестнадесетични – започват с 0x, следвани от цифри за разрядите на числото, като цифрите със стойност от 10 до 15 се задават с първите букви от латинската азбука от A до F. Например константата 0xAC* е шестнадесетична. Преобразувана в десетична, тази константа има стойност $10*16^1+12*16^0=172_{10}$.

❖ Дробни – могат да бъдат само десетични. Прието е като разделител между цялата и дробната част да се използва десетична точка вместо десетична запетая, например 12.54. Запетаята в JavaScript се използва като разделител например при изброяване на елементи на списъци.

❖ Символни – JavaScript дава възможност за използване на символни константи, които се записват като последователност от символи, ограничени отляво и отдясно с двойни кавички или апострофи, напр. “Hello “ или ‘World’

❖ Логически – логическите константи “истина” и “неистина” се записват с наименованията си на английски съответно като **true** и **false**.

20.2Променливи

Освен константи всеки език за програмиране включва променливи, които представят информацията, която се променя по време на изпълнение на програмата.

D3 Променливата е величина, която може да променя стойността си по време на изпълнение на програмата и се определя от елементите {име, тип, стойност}.

Стойността, съхранявана в променливата се използва чрез нейното име. За всяка променлива при изпълнение на програмата се заделя място в оперативната памет на компютъра. При избор на име на променлива трябва да се спазват следните правила:

Имената на променливите не трябва да започват с цифра и могат да съдържат букви, цифри и символи за подчертаване “_” и “$”.

Имената на променливите трябва да са уникални (различни) за всички променливи, декларирани като глобални (извън функции) или вътре в дадена функция. JavaScript (както и Java) е C-подобен език и прави разлика между малките и големи букви в имената на променливите. Например за JavaScript. x и X са две различни променливи, докато например за VB Script (скриптов език, създаден от Microsoft и поддържан от Internet Explorer) това е една и съща променлива.

* Могат да се използват и малките букви a-f.

Имената на променливите трябва да са различни от ключовите думи, дефинирани в JavaScript. По-долу е даден списък на ключовите думи на езика:

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

Таблица 1 Списък на ключовите думи в JavaScript

Ключовите думи не трябва да се използват като имена на променливи, функции, обекти, атрибути и методи. Използването им може да доведе до грешки, независимо от това, че някои от тези ключови думи са запазени за бъдещи разширения на езика.

20.2.1 Деклариране на променливи

В JavaScript декларирането на променливите не е задължително. За деклариране на променливи се използват ключовите думи **var** или **let**, следвани от име на променлива или списък от имената на променливите, разделени със запетаи.

```
var x, y, t=0;
let z=true, s="Hello";
```

Пример 54 Деклариране на променливи

Ако не се декларират с ключовата дума **var** или **let**, променливите се създават при тяхното първо използване и са глобални (т.е. достъпни навсякъде).

Променливите, декларирани с "**var**" са достъпни на ниво функция, т.е. ако са декларирани във функция, не са достъпни извън нея.

Променливите, декларирани с "**let**" са достъпни на ниво блок {…}, т.е. ако са декларирани в блок, не са достъпни извън него.

Типът на променливите в JavaScript (както и в други скриптови езици) не се задава явно. Неявно в JavaScript се разграничават следните типове променливи:

Логически – могат да съдържат стойност true (логическа истина) или false (логическа неистина).

Числови - могат да съдържат цели числа или реални числа (с дробна част с разделител десетична точка).

Символни - могат да съдържат последователност от символи (символен низ). Символните променливи могат да се инициализират (т.е. да им бъде присвоен символен низ още при декларирането им).

20.2.2 Инициализиране на променливите.

Под инициализиране се разбира задаването на начални стойности на променливите. Програмните езици, които поддържат синтаксиса на езика C* , дават възможност при деклариране на променливите те да бъдат инициализирани, т.е. да получат начална стойност. В предния пример е дадена декларация на променливите x, y и t посредством ключовата дума var. Те могат да бъдат инициализирани при декларирането си както следва:

```
var x=true, y=3.14, t=”Hello”;
```

Пример 55 Деклариране и инициализиране на променливи

Както беше посочено по-горе, декларирането на променливите в JavaScript не е задължително. В JavaScript (както е по правило в скриптовите езици) недекларираните променливи се създават при тяхното първо използване. Задължително е обаче при първото използване на променлива тя да получи стойност т.е. да бъде инициализирана. Опитът да бъде получена стойност от неинициализирана променлива води до генериране на грешка по време на изпълнение на скрипта (run-time error).

* По-нататък ще наричаме тези езици “C-подобни”.

20.3Операции, оператори и операнди.

20.3.1 Дефиниция

Всяка компютърна програма в съответствие с предназначението си извършва зададени операции над стойностите на променливи и константи. Основните елементи на една операция са операторите и операндите.

D4 Операторите в JavaScript са символи за обозначаване на операции. Константите и променливите*, които участват при изпълнението на операциите се наричат операнди.

20.3.2 Таблица на операторите

Дадената по-долу таблица съдържа обозначението на оператора, типът на операндите в реда на записването им, кратко описание на операцията и приоритетът на операцията при изчисляване на стойността на изрази.

Оператор	Тип на операндите	Операция	Приоритет
,	всеки тип	Изчислява стойността на два операнда. Връща стойността на втория операнд.	1 (най-нисък)
=	променлива, всеки тип	Присвояване на стойност	2
+=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, ^=, =	променлива, всеки тип	Присвояване на стойност с извършване на операция (присвояване с натрупване)	2
? :	логически, всеки тип	Условен оператор (тернарен)	3
||	логически	Логическо събиране (ИЛИ)	4
&&	логически	Логическо умножение (И)	5
|	целочислен	Побитово събиране	6
^	целочислен	Побитово изключващо ИЛИ	7
&	целочислен	Побитово умножение	8

* Като променливи величини в операциите могат да се използват и стойности на елементи на масиви, функции, атрибути и методи на обекти.

==, !=	всеки тип	Проверка за равенство/идентичност	9
<, <=, >, >=	Число или символен низ	Оператори за сравнение	10
<<, >>, >>>	целочислен	Измествания	11
+, -	Число или символен низ (само за +)	Събиране/Изваждане Слепване	12
*, /, %	число	Умножение/Делене/Модул	13
++,--, -	целочислен	Инкрементиране/Декрементиране/Смяна на знака (унарни)	14
~	целочислен	Побитово инвертиране (унарен)	
!	логически	Логическо отрицание (унарен)	
new	Конструктор	Създаване на нов обект (унарен)	
typeof	всеки тип	Връща символен низ с информация за типа на операнда (унарен)	
void	всеки тип	Връща недефинирана стойност (унарен)	
[ ] .	Масив, целочислен	Индекс на масив	15
( )	Функция, списък от аргументи	Извикване на функция	
.	Обект, атрибут/метод	Достъп до атрибут или метод на обект	

Таблица 2 Оператори на JavaScript

Според броя на операндите, които участват при изпълнението на операцията, операциите (и съответно операторите) на JavaScript са:

- едноместни (унарни) – с един операнд
- двуместни (бинарни) – с два операнда
- триместни (тернарни) – с три операнда

Според предназначението си операциите на JavaScript могат да бъдат класифицирани в следните основните групи:

20.3.3 Аритметични операции

Основните аритметични операции са събиране, изваждане, умножение, делене и модул. Те се задават с операторите:

+	за събиране
-	за изваждане
*	за умножение
/	за делене
%	за модул

Аритметичните операции са двуместни (бинарни) операции. Допълнително пояснение изисква само операцията модул (%). Това е операция, с помощта на която можем да определим остатъка от целочислено делене. Тя се изпълнява над целочислени оператори. Стойността, която се получава при изпълнение на операцията делене по модул е остатъкът от деленето на две цели числа, например:

5 % 3 равно на 2	5 делено на 3 е равно на 1 и **остатък 2**
0 % 3 равно на 0	Резултатът от деленето е 0, което е цяло без остатък – **остатък 0**
-10 % 3 равно на -1	частното е отрицателно число (-3) и за да получим делимото (-10) прябва да добавим **остатък -1**

Пример 56 Аритметична операция модул

JavaScript предоставя три едноместни (унарни) аритметичните операции:

++	инкрементиране (увеличаване с 1)
--	декрементиране (намаляване с 1)
-	смяна на знака (умножение с -1)

Тези операции може да се считат като по-кратък запис на бинарни аритметични операции, например операцията x++ е еквивалентна на x=x+1. Интересно е да се отбележи, че съвременните микропроцесори притежават машинни команди за изпълнение на унарните операции инкрементиране, декрементиране и смяна на знака.

20.3.4 Логически операции

Логическите операции са операции, които се извършват над логически операнди и връщат логически резултат true (логическа истина) или false (логическа неистина). Ако някой от операндите съдържа числова стойност,

рзлична от 0, при изпълнение на логическа операция стойността му се приема за true, а ако съдържа стойност 0, се приема за false. Такова неявно преобразуване на числови стойности в логически се извършва от всички C-подобни езици за програмиране JavaScript предоставя следните логически операции:

&&	логическо умножение (бинарна)
||	логическо събиране (бинарна)
!	логическо отрицание (унарна)

Логическите операции могат да се разглеждат като изчисляване на стойности на логически функции. От Булевата алгебра е известно, че логическите функции И, ИЛИ и НЕ образуват функционално пълна база, което означава, че произволно сложна логическа функция може да се представи с логически израз, в който се използват само тези функции. Логическата функция Z = X && Y (лог. умножение) се представя посредством таблицата 3a. Вижда се, че тя дава като резултат true (лог. истина) само ако и двата операнда имат стойност true. . Логическата функция Z = X || Y (лог. събиране) се представя посредством таблицата 3b. Тя дава като резултат true (лог. истина) само ако поне един двата операнда имат стойност true.

X	Y	Z
false	false	false
false	true	false
true	false	false
true	true	true

3a) логическо умножение (И)

X	Y	Z
false	false	false
false	true	true
true	false	true
true	true	true

3b) логическо събиране (ИЛИ)

X	Z
false	true
true	false

3c) логическо отрицание (НЕ)

Логически операции – таблици на истинност.

Логическата функция Z = !X (лог. отрицание), представена посредством таблицата 3c, връща стойност, обратна на стойността на операнда.

20.3.5 Операции за сравнение

Операциите за сравнение са операции, които се извършват сравняване на съдържанието на два операнда и връщат логически резултат. По принцип сравнение може да се извършва между стойности от един и същ тип. Операторите за сравнение на JavaScript дават възможност да се извършва сравнение между две числови стойности или между два символни низа. JavaScript предоставя следните оператори за сравнение:

Оператор	Значение: връща логическа истина при:
= =	равно на
! =	различно от
>	по-голямо от
<	по-малко от
>=	по-голямо или равно на
<=	по-малко или равно на

Например операцията X != 12 ще върне стойност false ако стойността на променливата X е равна на 12 и true във всички други случаи. По-подробно е необходимо да се поясни сравняването на символните низове. Два символни низа могат да бъдат равни (т.е. еквивалентни) само ако имат еднаква дължина и еднакви символи. При сравняването на два символни низа с някой от операторите “>” , “>” , “>=” или “<=” се извършва последователно сравняване на кодовете на символите от двата низа и сравнението завършва тогава, когато се открие, че различие. Резултата от последното сравнение на кодове на символи се приема за резултат от операцията сравнение на низовете. Например ако се сравняват низовете “Hello” и “Harry” различие ще се установи при сравняване на вторите по ред символи в низовете – “e” от “Hello” и “a” от “Harry”. Кода на буквата “e” (латиница) е 101, а кода на “a” е 97, и поради това при сравняването на двата низа ще се получи че първият низ има по-голяма стойност от втория. Следователно в дадения по-долу пример променливата Z ще получи стойност true.

```
var X = ”Hello”, Y = ”Harry”, Z = (X>Y);
```

Пример 57 Сравняване на символни низове

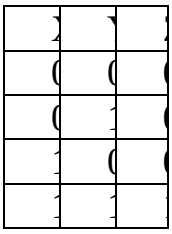
20.3.6 Побитови операции

Побитовите (bitwise) операции са бинарни операции, които се извършват над съответните битове на един или два целочислени операнда и връщат като резултат целочислена стойност. В JavaScript стойностите на числовите променливи се записват в четири байта (32 двоични разряда). При

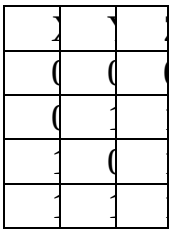
побитовите операции стойността на всеки от 32-та двоични разряда на резултата се получава като логическа функция от стойностите на същите по номер битове на двата операнда. JavaScript предоставя следните побитови операции:

&	побитово умножение AND (бинарна)
|	побитово събиране OR (бинарна)
^	побитово изключващо събиране XOR (бинарна)
~	побитово инвертиране NOT (унарна)

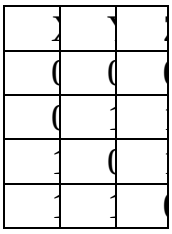
Таблиците на истинност на логическите функции, чрез които се изчисляват битовете на резултата са аналогични на таблиците на истинност за логическите операции:



4a) побитово умножение (AND)



4b) побитово събиране (OR)



4c) изключва-що събиране (XOR)

X	Z
0	1
1	0

4d) побитово отрицание (NOT)

Побитови операции – таблици на истинност.

Примери:

```
var X = 7, Y = 12 ;
var Z1 = X & Y;
var Z2 = X | Y;
var Z3 = X ^ Y;
var Z4 = ~X;
```

За да получим нагледна представа за резултата от побитовите операции на двата операнда X и Y трябва да ги представим като 32 разрядни двоични числа и да извършим операциите със съответните битове за да получим битовете на резултата. В случая можем да се ограничим до разглеждането на първите осем разряда (младшия байт) тъй като останалите разряди съдържат само нули По-долу е даден пример на побитовото умножение на операндите X и Y дава като резултат 32 разрядно число, което съдържа 1 само в разряд b2 и поради това стойността му е 4.

разряди	b7	b6	b5	b4	b3	b2	b1	b0
тегло	128	64	32	16	8	4	2	1
X	0	0	0	0	0	1	1	1
Y	0	0	0	0	1	1	0	0
Z	0	0	0	0	0	1	0	0

Пример 58 Побитово умножение Z1 = X & Y. Резултат Z=4

разряди	b7	b6	b5	b4	b3	b2	b1	b0
тегло	128	64	32	16	8	4	2	1

X	0	0	0	0	0	1	1	1
Y	0	0	0	0	1	1	0	0
Z	0	0	0	0	1	1	1	1

Пример 59 Побитово събиране Z2 = X | Y. Резултат Z=15

разряди	b7	b6	b5	b4	b3	b2	b1	b0
тегло	128	64	32	16	8	4	2	1
X	0	0	0	0	0	1	1	1
Y	0	0	0	0	1	1	0	0
Z	0	0	0	0	1	0	1	1

Пример 60 Побитово изключващо събиране Z3 = X ^ Y. Резултат Z=11

разряди	b7	b6	b5	b4	b3	b2	b1	b0
тегло	128	64	32	16	8	4	2	1
X	0	0	0	0	0	1	1	1
Z	1	1	1	1	1	0	0	0

Пример 61 Побитово инвертиране Z = ~X. Резултат $Z4=(2^{32}-1)-7$

Резултатът от побитовото инвертиране на X е 32-разрядно двоично число, което има единици във всички разряди с изключение на разрядите b0, b1 и b2, и поради това стойността му (като цяло положително число) е равна на сумата от теглата на всички разряди $(2^{32}-1)$ намалена със сумата от теглата на разрядите b0, b1 и b2*.

20.3.7 Измествания

Измествания са особен вид побитови операции. В тях участват два целочислени операнда, като при изпълнение на операцията съдържанието на левия операнд (operand1) се измества наляво или надясно на брой разряди (битове), равен на стойността на десния операнд (operand2). Съдържанието на operand1 участва в съответната операция като цяло число, представено с 32 двоични разряда (4 байта), а operand2 – като цяло положително число. JavaScript предоставя три оператора за изместване:

<<	изместване наляво
>>	аритметично изместване

* Детайлното разглеждане на побитовите операции излиза извън рамките на курса по WEB Дизайн.

надясно
>>> логическо изместване надясно

Интересно е да се отбележи, че аритметичните устройства на микропроцесорите включват схеми за извършване на операциите изместванията и поради това притежават машинни команди за извършване на тези операции. Изместванията са били въведени като команди на езика за програмиране С защото създателите му са искали програмите, написани на С да се доближават по бързодействие до програмите, написани на машинния език на процесора.

Изместване наляво <<

При операцията изместване наляво operand1 << operand2 се получава числова стойност, равна на съдържанието на operand1, изместено наляво на брой разряди, равен на operand2. При това изместване най-старшите битове от съдържанието на operand1 се «избутват» извън 32-та бита на резултата, а в най-младшите битове се «вмъкват» нули. Резултатът от тази операция е еквивалентен на умножение на operand1 по 2 на степен operand2. Резултатът обаче ще бъде такъв само ако в «избутаните» старши битове на operand1 са се съдържали само нули.

Пример: нека operand1=3. Тогава съдържанието му като 32-разрядно двоично число ще има вида:

operand1=00000000 00000000 00000000 00000011

При изпълнение на операцията operand1 << 2 резултатът, представен като 32 разрядно двоично число ще има вида:

00000000 00000000 00000000 00001100

, което, представено като десетично число е $12=3*2^2$. – съдържанието на operand1, умножено по 2^2.

Аритметично изместване надясно >>

При операцията аритметично изместване надясно operand1 >> operand2 се получава числова стойност, равна на съдържанието на operand1, изместено надясно на брой разряди, равен на operand2. При това изместване най-младшите битове от съдържанието на operand1 се «избутват» извън 32-та бита на резултата, а в най-старшите битове се «вмъкват» нули, но съдържанието на най-старшия бит b_{31} се копира (запазва стойността си).

Резултатът от тази операция е еквивалентен на делене на operand1 по 2 на степен operand2 със запазване на знака на operand1. По-точно, получава се цялата част от деленето, защото «избутаните» младши разряди на operand1 се губят.

Пример: нека operand1=-12. Тогава съдържанието му като 32-разрядно двоично число ще има вида*:

operand1=11111111 11111111 11111111 11110100

При изпълнение на операцията `operand1 >> 2` резултатът, представен като 32 разрядно двоично число ще има вида:

10111111 11111111 11111111 11111101 ,

което, представено като десетично число е $-3 = -12/2^2$. – съдържанието на operand1, разделено на 2^2.

Логическо изместване надясно >>>

При операцията логическо изместване надясно `operand1 >>> operand2` се получава числова стойност, равна на съдържанието на operand1, изместено надясно на брой разряди, равен на operand2. При това изместване най-младшите битове от съдържанието на operand1 се «избутват» извън 32-та бита на резултата, а в най-старшите битове се «вмъкват» нули, като съдържанието на най-старшия бит b_{31} също се избутва (за разлика от аритметичното изместване). Резултатът от тази операция е еквивалентен на делене на operand1 по 2 на степен operand2 , но при условие, че operand1 съдържа положително число. И при тази операция се получава се цялата част от деленето, защото «избутаните» младши разряди на operand1 се губят.

Пример: нека operand1=12. Тогава съдържанието му като 32-разрядно двоично число ще има вида:

operand1=00000000 00000000 00000000 00001100

При изпълнение на операцията `operand1 >>> 2` резултата, представен като 32 разрядно двоично число ще има вида:

* Числото е представено в т.н. допълнителен код. Детайлното разглеждане на машинното представяне на числата излиза извън рамките на настоящия курс.

00000000 00000000 00000000 00000011 ,

което, представено като десетично число е 3 = $12/2^2$. – съдържанието на operand1, разделено на 2^2.

20.3.8 Операции със символни низове

Освен операциите за сравнение на символни низове, разгледани в т 4.3.5 JavaScript предоставя само един оператор за работа със символни низове – операторът конкатенация “+”. Конкатенацията е бинарна операция, която “слепва” съдържанието на два символни низа и връща като резултат получения символен низ.

```
var X=”Hello “; Y=”World”, Z;
Z=X + Y;
```

Пример 62 Слепване на символни низове

Резултатът от операцията конкатенация X + Y в дадения пример е символния низ “Hello World”, получен от слепването на двата символни низа, съдържащи се в операндите X и Y.

Възможностите за операции със символни низове в JavaScript обаче съвсем не се изчерпват с операциите за сравнение и конкатенация. Както ще бъде разгледано по-нататък, JavaScript предоставя на програмиста обект String, който притежава богат набор от методи, чрез които се извършват операции със символни низове.

20.3.9 Условен оператор “?”

Условният оператор “?” е единственият оператор на JavaScript, който изисква три операнда. Той има следния синтаксис:

operand1 ? operand2 : operand3

където:
operand1 е логически операнд (променлива или константа) или израз, който връща логически резултат
operand2, operand3– операнди или изрази, които връщат стойност.

Стойността, която се получава при изпълнение на операцията зависи от стойността на operand1:

Ако operand1 има стойност true	операцията връща стойността на operand2
Ако operand1 има стойност false	операцията връща стойността на operand3

Условният оператор “?” не е просто съкратен запис на конструкцията if-else, защото, за разлика от конструкциите, разгледани по-нататък, операторите връщат стойност. Например като използваме условния оператор “?” можем да присвоим стойността, която се получава от него на някаква променлива или да я използване като фактически параметър при извикване на функция*.

```
var S=x ? “x има стойност true” : x има стойност false
```

Пример 63 Използване на условния оператор “?”

В примера в ако променливата x има стойност логическа истина (true) , в променливата S се записва низа “x има стойност true”, в противен случай в S се записва низа “x има стойност false”.

20.3.10 Оператори за присвояване

Операторът за присвояване “=” служи за присвояване на стойности на дадена променлива. Операцията присвояване е бинарна и има следния синтаксис:

operand1 = operand2

където operand1 може да бъде само име на променлива, а operand2– константа, променлива, функция или израз, който връща стойност. Стойността на operand2 се присвоява на operand1. Например при изпълнение на операцията `y=x+2` интерпретаторът на JavaScript ще изчисли аритметичната сума от съдържанието на променливата x и константата 2 и ще запише (или, както е прието да се казва, ще присвои) резултата на променливата y.

Операцията присвояване връща стойността, която се присвоява на operand1. Поради това например изразът* `1+(x=5)` не съдържа синтактична грешка. Първа ще се изпълни операцията в скобите, която ще върне

* Функциите се разглеждат в т. 4.5

* Изразите като програмни конструкции се разглеждат в следващата глава

стойност 5, след което ще се извърши операцията събиране и ще се получи резултат 6. Записът `1+x=5` обаче съдържа синтактична грешка, тъй като може да се присвоява стойност само на променлива, а в случая се прави опит да се присвои стойност на израза 1+x.

20.3.11 Съкратен запис на операции

Практиката в програмирането е показала [6], че най-често използваните операции са операциите за присвояване и аритметичните операции. Някои комбинации от тези операции са толкова често срещани, че в езика C и C-подобните езици като JavaScript са предвидени специални форми, които опростяват използването им. Това са конструкции от вида:

```
променлива1 = променлива1 оператор операнд
```

За подобни конструкции е предвиден съкратен запис от вида:

```
променлива1 оператор = операнд
```

В JavaScript се използват следните съкратени записи на операции.

Съкратена операция	**Значение**
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x <<= y	x = x << y
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

Таблица 3 Съкратен запис на операции в JavaScript

20.4Конструкции за управление.

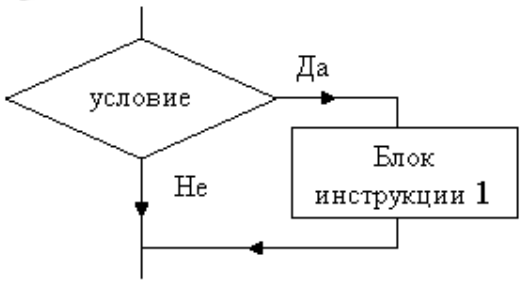
Всяка една програма, в съответствие с алгоритъма си, съдържа последователност от инструкции и конструкции за управление. Конструкциите за управление реализират условните разклонения в алгоритъма на програмата. От това следва, че всички разклонени програми съдържат конструкции за управление. JavaScript предоставя на програмиста две конструкции, които осъществяват условно разклонение - **if-else** и **switch** и три специализирани конструкции за организиране на цикли – **for**, **while** и **do-while**.

20.4.1 Условна конструкция if-else.

Тази конструкция се използва тогава, когато трябва да се изпълни една инструкция или блок от инструкции само в случай, че се изпълнява зададено условие. Клаузата else дава възможност да се изпълни друга инструкция или блок от инструкции в случай, че не се изпълнява зададеното условие. Конструкцията има следния синтаксис:

```
if (условие)
{
  Блок инструкции 1
}
```

Фиг. 5 Конструкция **if**

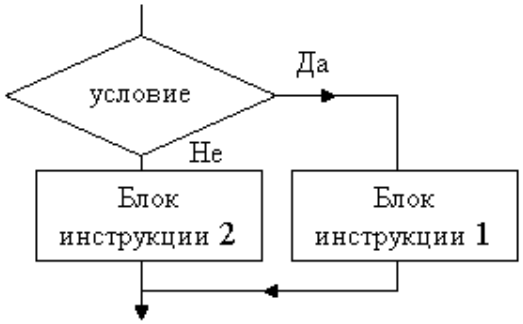


Фиг. 2 Блокова схема на конструкция if

Ако се добави клауза else конструкцията добива вида:

```
if (условие)
{
    Блок инструкции 1
}
else
{
    Блок инструкции 2
}
```

Фиг. 6 Конструкция **if – else**



Фиг. 3 Блокова схема на конструкция if-else

където:

- **условие** е израз, който връща логически стойност (true или false) или числова стойност, която се преобразува в логическа (стойност 0 се преобразува във false, а стойност, различна от 0 – във true).
- **блок инструкции 1** – Блок от инструкции, който се изпълнява, когато изразът **условие** има стойност логическа истина (true).
- **блок инструкции 2** – Блок от инструкции, който се изпълнява, когато изразът **условие** има стойност логическа неистина (false).

Ако трябва да се изпълни само една инструкция, тя може да не се включва във фигурни скоби, например конструкцията

```
if(x>5){ y=1; }
else { y=2; }
```

Пример 64 Конструкция if-else.

може да се замени с

```
if(x>5)  y=1
else  y=2;
```

Важно е да се запомни, че клаузата else трябва да следва непосредствено след блока инструкции след if, в противен случай клаузата ще представлява синтактична грешка в скрипта. Така например конструкцията

```
if(x>5)  y=1; x=5;
else  y=2;
```

съдържа синтактична грешка, защото клаузата else не следва непосредствено след инструкцията `y=1;` а а между тях е включена още една инструкция - `x=5;`.

Пример: Да се състави скрипт, който да изчислява стойността на функцията

$$y(x)=\begin{cases} x+5 \ за \ x<5 \\ x-1 \ за \ x>=5 \end{cases}$$

Както се вижда от зададената дефиниция, y(x) е по части непрекъсната функция. За да се определи по каква формула трябва да се изчислява нейната стойност, е необходимо да се направи проверка дали стойността на x е по-малка от 5 или е по-голяма или равна на 5. Тези условия са взаимно-изключващи се, следователно е достатъчна само една проверка, която в JavaScript ще се извърши посредством оператор за сравнение. За изчисляването на стойността на y може да се използва следната условна конструкция if-else:

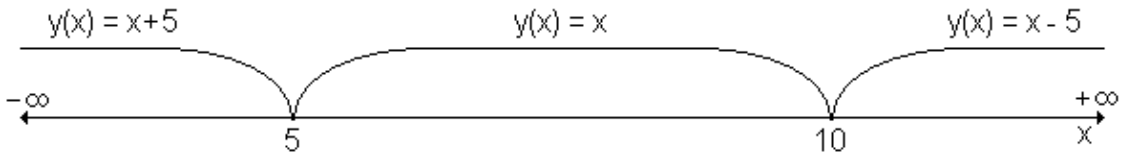
```
if(x<5) y = x + 5;
else    y = x – 1;
```

Блоковете след if и else могат да съдържат всички допустими за JavaScript инструкции, включително и други конструкции if-else. Посредством вложени конструкции if-else може да се реализира произволно сложна разклонена програма. Да разгледаме още един пример за изчисляване на стойност на функция, при който да е необходимо да се проверяват две условия:

Пример: Да се състави скрипт, който да изчислява стойността на функцията

$$y(x)=\begin{cases} x+5 \text{ за } x<5 \\ x \text{ за } 5\le x<10 \\ x-5 \text{ за } x\ge 10 \end{cases}$$

От зададеното условие се вижда, че за да се определи по каква формула трябва да се изчислява стойността на y(x) е необходимо да се провери в кой от зададените три интервала на числовата ос попада стойността на променливата x.



Фиг. 7 Разположение на интервалите от пример 52 върху числовата ос.

Тъй като трите интервала I1 (-∞ , 5) , I2 [5 , 10) и I3 [10 , +∞) покриват цялата числова ос -∞ , +∞ , т.е. всички възможни стойности на x, и не се припокриват взаимно, достатъчни са две проверки за да се установи в кой интервал попада стойността на x и така да се определи по коя формула да се изчислява стойността на y(x). Скриптът, който извършва това изчисление може да се реализира посредством две вложени конструкции if – else.

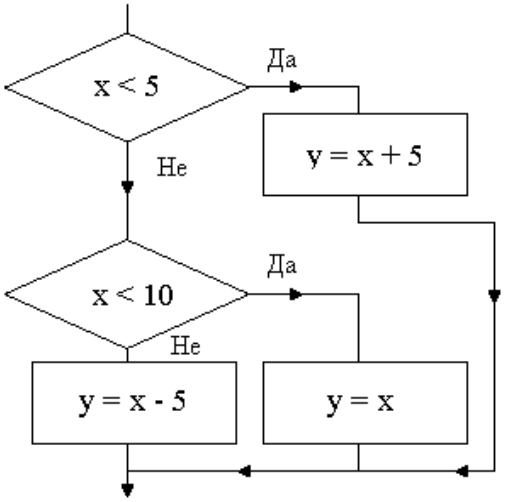
```
if(x<5)              // Проверка дали x попада в интервал I1
    {y=x+5;}         // x е в интервал I1 и y се изчислява по тази формула
else                 // x не е в интервал I1
    {
    if(x < 10)       // Проверка дали x е в интервал I2
          {y=x;}     // x е в интервал I2 и y се изчислява по тази формула
    else             // x не е в интервал I2 следователно е в интервал I3
```

```
        {y=x-5;} // y се изчислява по формулата за интервал I3
    }
```

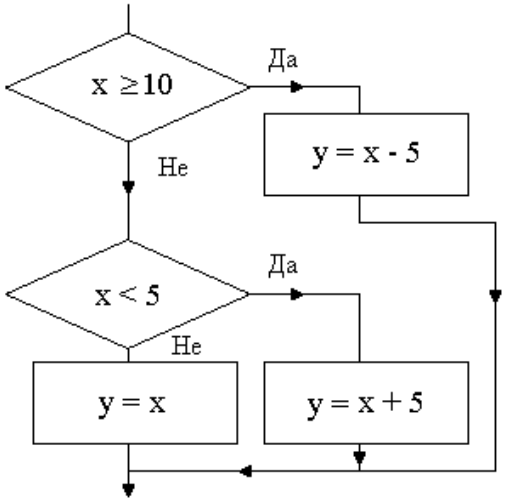
Пример 65 Вложени конструкции if – else

Както беше посочено по-горе, когато след клаузата if или else трябва да се изпълни само една инструкция, не е необходимо инструкцията да се включва в блок { … }. Така предходният скрипт може да се запише в доста по-компактен вид:

```
if(x<5)        y=x+5;      // x е в интервал I1 и y се изчислява по тази формула
else if(x < 10) y=x; // x е в интервал I2 и y се изчислява по тази формула
    else y=x-5;             // y се изчислява по формулата за интервал I3
```



Фиг. 8 Блокова схема на скрипта от пример 52



Фиг. 9 Друг вариант на блокова схема по зададеното условие на пример 52

Когато условието на задачата изисква проверката на няколко условия, са възможни различни варианти за съставяне на алгоритъм и програма, която да го реализира. Блоковата схема, дадена на Фиг. 6 е друг вариант на алгоритъм, по който може да се състави скрипт, който да извършва изчисленията по зададеното условие на пример 52. Както се вижда от схемата, разликата е в реда на проверка на условията, но крайният резултат от изчисляването на стойността на y(x) е същия.

Да разгледаме още един пример, при който се извършва сравнение на стойността на променлива или израз с набор от константи и при съвпадение се изпълнява определена инструкция или блок инструкции.

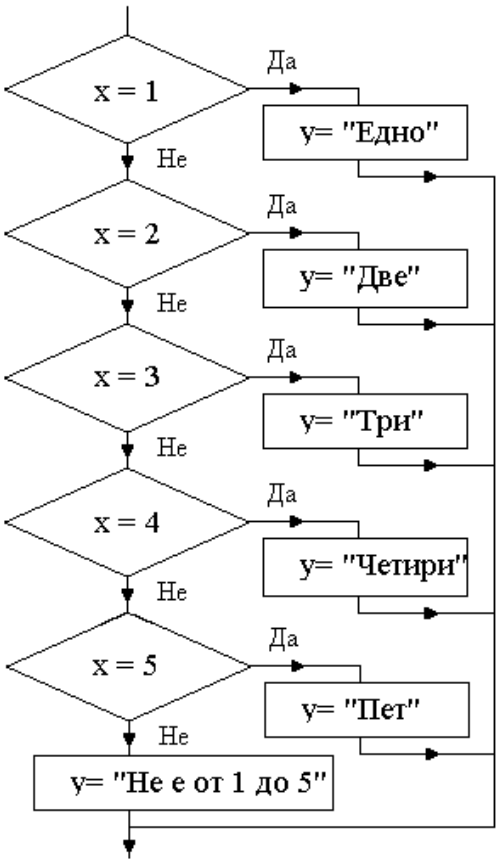
Пример: Да се състави скрипт, който да сравнява съдържанието на променлива x с числата от 1 до 5 и при съвпадение да присвоява на променлива y низ, съответстващ на стойността на променливата x. Ако стойността на x е извън интервала [1 ÷ 5] на y трябва да се присвои низ "Не е от 1 до 5". Записано както в предния пример, условието има вида:

$$
y. = \begin{cases} \text{"Едно"} & \text{за } x = 1 \\ \text{"Две"} & \text{за } x = 2 \\ \text{"Три"} & \text{за } x = 3 \\ \text{"Четири"} & \text{за } x = 4 \\ \text{"Пет"} & \text{за } x = 5 \\ \text{"Не е от 1 до 5"} & \text{за } x \notin [1 \div 5] \end{cases}
$$

Блоковата схема, представяща алгоритъма, по който може да се получи стойността на y в съответствие с поставената задача, е показан на фиг. 7. Скриптът може да се реализира посредством влагане на конструкции if else , като всяка следваща конструкция се влага в клаузата else – на предходната. Тъй като вложената if–else конструкция е единствената инструкция, която се изпълнява в клаузата else на външната if–else конструкция, не са необходими фигурни скоби за блок:

```
if(x == 1) y = "Едно";
else  if(x == 2) y = "Две";
else  if(x == 3) y = "Три";
else  if(x == 4) y = "Четири";
else  if(x == 5) y = "Пет";
else y = "Не е от 1 до 5";
```

Пример 66 Сравнение на съдържанието на променлива с набор от константи– реализация посредством вложени конструкции if – else



Фиг. 10 Сравняване на съдържанието на променлива с набор от константи

Фиг. 11 Обща блокова схема на цикъл с постфиксна проверка за край

Счита се за добър стил на оформление на текста на програмата, да се отместват по-вдясно (напр. с една табулация) блоковете след клаузите if и else, както е направено в дадения пример. Така при влагане на структури if-else за този, който разглежда сорс кода е по-лесно да се ориентира кой блок към коя конструкция принадлежи. Препоръчваме ви също така да не си спестявате времето за добавяне на коментари към сорс кода – коментарите, и дори простото отделяне с няколо празни реда на отделни фрагменти от сорс кода на програмата ще ви спести много време в последствие.

Конструкции за организиране на цикли for, while и do-while.

Както беше посочено в т. 4 и т. 5.4, конструкциите за огранизиране на цикли осигуряват многократно повторение на зададен блок от инструкции (тяло на цикъла) докато се изпълнява условието за продължаване на

цикъла. JavaScript предоставя на програмиста три специализирани конструкции за организиране на цикли - **for**, **while** и **do-while**

20.4.2 Конструкция за организиране на цикли for.

Конструкцията for е най-често използваната конструкция за организиране на цикли. Тя, за разлика от конструкциите while и do-while поема по-голяма част от организацията на цикъла, и поради това дава по-компактен сорс код на цикъла.

Синтаксис:

```
for(инициализация ; условие ; инкрементиране) инструкция
```

или

```
for(инициализация ; условие ; отброяване)
{
    блок инструкции
}
```

инициализация е инструкция или списък от инструкции, които се изпълняват еднократно като начално установяване на цикъла*.

условие е израз, който връща логическа стойност или числова стойност, която се преобразува в логическа. Много често изразът съдържа една или няколко операции за сравнение и логически операции.

отброяване по правило е инструкция, която променя стойността на индекса (брояча) на цикъла.

Блокът инструкции се повтаря до тогава, докато изразът, зададен като **условие** връща резултат логическа истина. Условието се проверява преди изпълнението на блока инструкции, следователно конструкцията for реализира цикъл с префиксна проверка. Ако в цикъла трябва да се изпълнява само една инструкция не е необходимо тя да се включва в блок {…}. Блоковата схема, която представя алгоритъма, съответстващ на конструкцията for е дадена на фиг. 9:

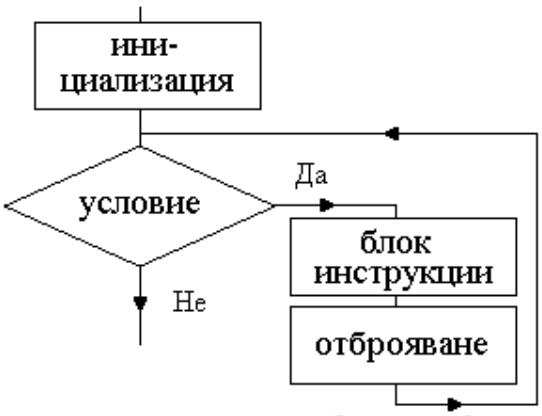
Да разгледаме един пример за реализиране на цикъл посредством конструкция for.

Пример:

Да се състави скрипт, който да изчислява сумата от първите N натурални (т.е. цели положителни) числа.

* Вижте общата блокова схема на цикъл, дадена в т. 4.

Съгласно условието, трябва да се изчисли сумата S = 1 + 2 + 3 ...+ N. Ако се използва математическия символ за сума "Σ" стойността на S, която трябва да се изчисли, може да се представи във вида: $S = \sum_{n=1}^{N} n$



Фиг. 12 Блокова схема на конструкция for

За да се реализира като цикъл, многократното сумиране трябва да се представи като последователност от операции, които да отговарят на определението, дадено за цикличен алгоритъм, дадено в т. 4.1.

За да съставим цикличен алгоритъм за решаване на поставената задача, първо трябва представим събирането на много събираеми като последователност от операции събиране на две събираеми. Това е необходимо, тъй като JavaScript не предоставя операция събиране с повече от две събираеми. Последователните операции събиране се представят във вида:

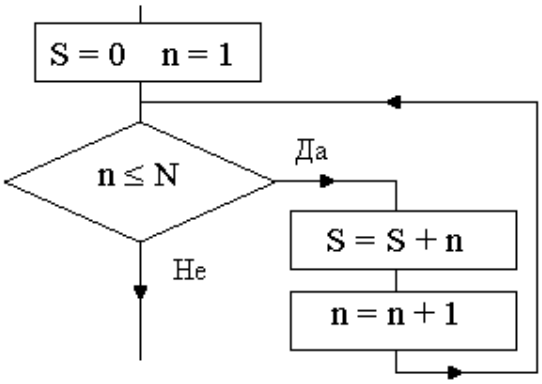
```
S = 0;
S = S + 1;
S = S + 2;
……
S = S + N;
```

Операция събиране, при която резултатът от операцията се записва в променливата, съдържаща първия операнд, се нарича натрупващо събиране. След изпълнението на последната инструкция, в S се съдържа сумата от числата от едно до N. Тази последователност от инструкции обаче не е подходяща за организиране на цикъл, защото съгласно определението за цикъл трябва да се извършва многократно повторение на едни и същи инструкции, а в предлагания алгоритъм всяка следваща функция е различна. Изходът е да се повтаря двойка инструкции, като първата прибавя към съдържанието на S съдържанието на променлива n, а втората да увеличава съдържанието на n с 1.

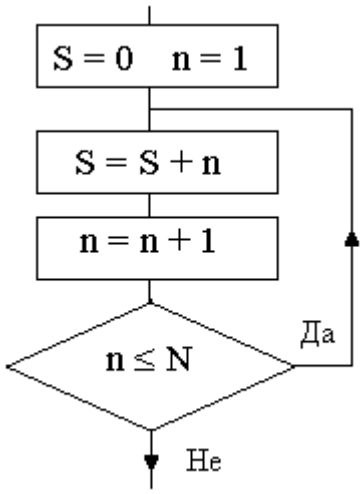
```
S = 0;        n=1;  //Начално установяване на S и n
```

$S = S + n;\ n = n + 1;$
$S = S + n;\ n = n + 1;$ } N *двойки инструкции*
……..……..……
$S = S + n;\ n = n + 1;$

След задаването на начални стойности на променливите S и n по този алгоритъм се повтаря N пъти двойката инструкции `S=S+n; n=n+1;` След N изпълнения на двойката инструкции, в променливата S се получава крайния резултат – сумата на числата от 1 до N. Блоковите схеми на цикъл по този алгоритъм съответно с префиксна и постфиксна проверка имат вида:



Фиг. 13 Блокова схема на алгоритъм за изчисляване на сумата на първите N натурални числа с префиксна проверка (за цикъл с конструкции for и while)



Фиг. 14 Блокова схема с постфиксна проверка (за цикъл с конструкция do-while)

Елементарен вход и изход чрез диалогови кутии.

За да дадем на оператора възможност да въведе стойност на параметъра N (брой натурални числа, които да се сумират) и да изведем резултата ще бъде необходимо да използваме някои функции, които ще разгледаме по-подробно в следващите глави. Това са:

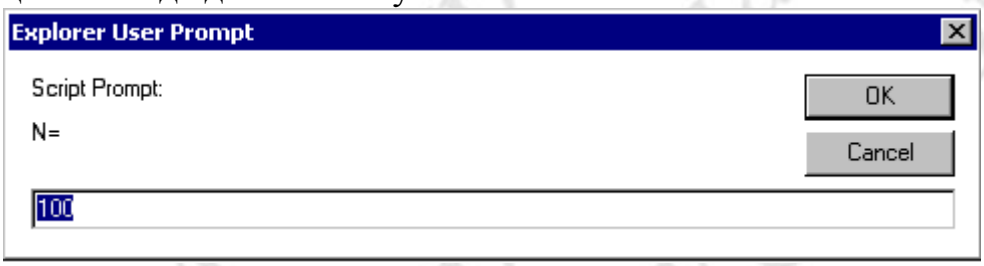
- parseInt(string) – функция, която получава като параметър символен низ и връща първото цяло число, което се съдържа в низа. Например от инструкцията X=parseInt(“100 200”) ще присвои на X числото 100. Ако символният низ не започва с число (празните позиции преди

първата цифра се пренебрегват) функцията връща недефинирана стойност (NaN).

- Функция prompt(string1[, string2]) извежда диалогова кутия с пояснителен текст string1 и незадължителен втори параметър string2, който, ако е включен, се извежда като начално съдържание на текстова кутия, в противен случай се извежда текст *undefined*. Предназначението на функцията prompt е да даде възможност на оператора да въведе текст (символен низ), който да бъде получен от скрипта. Например ако се изпълни инструкцията

```
var N=parseFloat(prompt("N=","100"));
```

ще се изведе диалогова кутия



Ако операторът щракне с мишката върху бутона OK функцията prompt ще върне символен низ, който ще съдържа това, което е въведено в текстовата кутия (по подразбиране “100”). Функцията parseInt ще преобразува низа в число.

- Функция alert(string) извежда диалогова кутия с текста, съдържащ се в string и само един бутон OK , чрез който се затваря диалоговата кутия. Ако текстът съдържа символи за преминаване на нов ред “\n” и табулация “\t”, те се отработват. Това означава, че текстът в диалоговата кутия може да бъде на няколко реда и да бъде позициониран с табулации.

Да съставим скрипта като използваме конструкцията for

За да тестваме скрипта, трябва да го включим в блок <SCRIPT> в HTML документ и да заредим документа във WEB браузър. Пълният текст на HTML документа има вида:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
    var S=0;
    var N=parseFloat(prompt("N=","100"));
    for(var n=1;n<=N;n++)
    {
```

```
            S=S+n;
        }
        alert("S="+S);
    </SCRIPT>
    </BODY></HTML>
```

Пример 67 Изчисляване на сумата на първите N натурални числа посредством цикъл for.

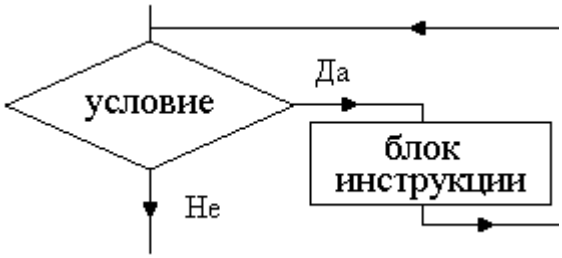
Скриптът е включен в блок <SCRIPT> в тялото на документа. Като начално установяване на цикъла се присвоява стойност 0 на променливата S, операторът въвежда стойност на променливата N чрез диалогова кутия prompt, а на променливата n се въвежда начална стойност 1 чрез конструкцията for. В тялото на цикъла N пъти се прибавя към променливата S поредната стойност на n и след това n се увеличава с 1. Когато стойността на n стане по-голяма от N, цикълът завършва и резултатът (съдържанието на S) се извежда в диалогова кутия посредством функция alert. Тъй като инструкциите не са включени в кода на функция, те се изпълняват веднага при зареждането на документа в браузъра.

20.4.3 Конструкция за организиране на цикли while.

Конструкцията while също както for реализира цикъл с префиксна проверка. Синтаксис:

```
while (условие)
{
    блок инструкции
}
```

Блокът инструкции се повтаря до тогава, докато изразът, зададен като **условие** връща резултат логическа истина. Условието се проверява преди изпълнението на блока инструкции, следователно конструкцията while, (както и for) реализира цикъл с префиксна проверка. Ако в цикъла трябва да се изпълнява само една инструкция не е необходимо тя да се включва в блок {…}. Блоковата схема, която представя алгоритъма, съответстващ на конструкцията while има вида:



Фиг. 15 Блокова схема на конструкция while

Както се вижда от описанието и блоковата схема на конструкцията, в нея, в сравнение с for, липсват клаузите инициализация и инкрементиране . Поради това, ако се реализира цикъл с while, за тези операции трябва да се включат отделни инструкции.

Така например, скриптът за изчисляване на сумата на първите N натурални числа, реализиран с конструкция while ще има вида:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
    var S=0; n=1;
    var N=parseFloat(prompt("N=","100"));
    while(n<=N)
    {
        S=S+n;
        n++;
    }
    alert("S="+S);
</SCRIPT>
</BODY></HTML>
```

Пример 68 Изчисляване на сумата на първите N натурални числа посредством цикъл while.

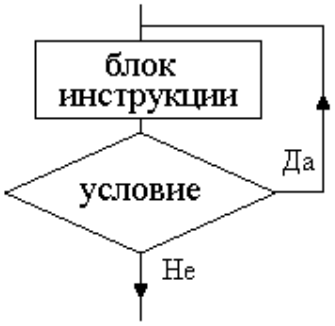
20.4.4 Конструкция за организиране на цикли do-while.

Конструкцията do-while за разлика от for и while реализира цикъл с постфиксна проверка. Синтаксис:

```
do
    {
        блок инструкции
    }
while (условие);
```

Блокът инструкции се повтаря до тогава, докато изразът, зададен като **условие** връща резултат логическа истина. Условието се проверява след

изпълнението на блока инструкции. Ако в цикъла трябва да се изпълнява само една инструкция не е необходимо тя да се включва в блок {…}. Блоковата схема, която представя алгоритъма, съответстващ на конструкцията do-while има вида:



Фиг. 16 Блокова схема на конструкция do-while

Както се вижда от описанието и блоковата схема на конструкцията, конструкцията започва с блока от инструкции и завършва с проверката на условието за продължаване на цикъла. Поради това блокът от инструкции, включени в тялото на цикъла, ще се изпълни поне веднъж, независимо от стойността на израза, зададен като условие.

Скриптът за изчисляване на сумата на първите N натурални числа, реализиран с конструкция do-while ще има вида:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
    var S=0; n=1;
    var N=parseFloat(prompt("N=","100"));
    do
        {
            S=S+n;
            n++;
        }
    while(n<=N);
    alert("S="+S);
</SCRIPT>
</BODY></HTML>
```

Пример 69 Изчисляване на сумата на първите N натурални числа посредством цикъл do-while.

Както беше посочено по-горе, разлика в резултата от изпълнение на цикъл, реализиран с for или while (с префиксна проверка) от една страна, и цикъл do-while (с постфиксна проверка) от друга, ще се получи само ако при първо изпълнение изразът, зададен като условие, връща стойност логическа неистина (false). Вие можете да видите тази разлика, ако при изпълнението на скриптовете от примери 54 ÷ 56 въведете за N стойност. Тогава от скриптовете от примери 54 и 55 ще се получи резултат S = 0 (това е стойността, която S получава от началното установяване). Скрипта от

пример 56 обаче ще даде като резултат стойност S = 1, тъй като инструкцията S = S +1 ще се изпълни и едва тогава цикълът ще завърши.

20.4.5 Безусловен преход break.

Безусловният преход break дава възможност за прекратяване на изпълнението на цикъл, структура switch или блок инструкции, асоцииран с етикет.

Синтаксис:

```
break  [label]
```

където label е незадължителен етикет на инструкция или блок от инструкции.

Да разгледаме един пример:

```
for(var n=0;n<10;n++)
{
    ……..
    if(n==3)break;
}
alert(“n=”+n); //извеждане на стойността на n в диалогова кутия.
```

Пример 70 Използване на оператор break за излизане от цикъл for

В примера конструкцията for организира цикъл, който трябва да се изпълни 10 пъти (за стойности на n от 0 до 9), но когато n достигне стойност 3 се изпълнява break и цикълът завършва. Така, че на практика цикълът се изпълнява 4 пъти (за стойности на n от 0 до 3).

20.4.6 Безусловен преход continue.

Безусловният преход continue рестартира изпълнението на цикъл, структура switch или блок инструкции, асоцииран с етикет.

Синтаксис:

```
continue  [label]
```

където label е незадължителен етикет на инструкция или блок от инструкции. Етикетът в JavaScript се задава преди инструкцията с име (по синтаксиса за име напроменлива), следвано от символ ":".

Да разгледаме един пример:

```
var S=0;
for(var n=0;n<10;n++)
{
    if(n==3) continue;
    S+=n;
}
alert("S="+S); //извеждане на стойността на S в диалогова кутия.
```

Пример 71 Използване на оператор continue за рестартиране на цикъл for

В примера конструкцията for организира цикъл, който трябва получи в S сумата на числата от 1 до 10, но когато n достигне стойност 3 се изпълнява continue и цикълът се рестартира, което означава, че се прекратява текущата итерация и като резултат стойността 3 не се добавя съм сумата.

20.4.7 Конструкция за избор switch.

Конструкцията switch дава възможност за многократно разклонение чрез сравнение на стойността на зададен израз с набор от константи. Тя има следния синтаксис:

```
switch(expression)
{
    case label 1:
        инструкции 1
        break;
    case label 2:
        инструкции 2
        break;
    ……….
    [ default:
      блок инструкции default ]
}
```

При изпълнение на конструкцията първо се изчислява стойността на израза *expression* (в частност вместо израз може да бъде зададена само една променлива), след което стойността се сравнява последователно с константите label1, label2 … и ако се открие съвпадение на стойността на израза с някой етикет (константа) се изпълняват инструкциите, които следват след етикета.

Ако не се открие съвпадение с никой от етикетите, се изпълняват инструкциите след незадължителната клауза default (ако е включена).

Да разгледаме отново примера, реализиран в т. 5.4.1 чрез структура if-else.

Пример: Да се състави скрипт, който да сравнява съдържанието на променлива x с числата от 1 до 5 и при съвпадение да присвоява на променлива y низ, съответстващ на стойността на променливата x. Ако стойността на x е извън интервала [1 ÷ 5] на y трябва да се присвои низ “Не е от 1 до 5”.

Реализацията на скрипта посредством структура switch има вида:

```
switch(x)
{
    case 1:
        y=”Едно”;
        break;
    case 2:
        y=”Две”;
        break;
    case 3:
        y=”Три”;
        break;
    case 4:
        y=”Четири”;
        break;
    case 5:
        y=”Пет”;
        break;
    default:
        y=” Не е от 1 до 5”;
}
```

Пример 72 Използване на структура switch

Скриптът дава същия резултат, както скрипта, реализиран с вложени конструкции if-else, даден в пример 53.

По правило всяка последователност от инструкции, която следва след някоя клауза case, завършва с break, защото чрез този безусловен преход се излиза от конструкцията switch и изпълнението на скрипта продължава със следващата след switch инструкция. Включването на break след всеки case обаче съвсем не е задължително. Ако break липсва, скриптът ще продължи с изпълнението на следващите инструкции до достигане на края на блока на switch или до достигане на безусловен преход break. Да разгледаме един такъв пример:

```
switch(x)
{
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        y=” x е от 1 до 5”;
        break;
    default:
        y=” Не е от 1 до 5”;
}
```

Пример 73 Използване на структура switch с клаузи case с общ безусловен преход break.

В този пример стойността на x се сравнява последователно с константите 1, 2, 3, 4 и 5. Ако се получи съвпадение на x с една от тези стойности, се изпълнява инструкцията

y=” x е от 1 до 5”;

се излиза от конструкцията switch с безусловния преход break. Ако не се получи съвпадение на стойността на x с нито едно от числата от 1 до 5, ще се изпълни инструкцията след default.

20.5 Функции.

Опитът в програмирането показва, че много често в една и съща програма се налага няколко пъти да се извършват едни и същи изчисления или да се решават едни и същи задачи. В такива случаи е по-рационално такива части от алгоритъма да се отделят като самостоятелни модули с възможност за многократно използване на включения в тях код. В JavaScript, както и на практика във всички съвременни езици за програмиране, са включени структурни единици, наречени функции, които предоставят на програмиста възможност да използва многократно в програмата си кода, деклариран в тях.

D5. Функцията в JavaScript е самостоятелен модул, деклариран с ключовата дума function, име на функцията, списък от формални параметри, зададен в скоби и тяло на функцията, съдържащо декларации на променливи и набор от инструкции, зададени във фигурни скоби. Параметрите на функцията и инструкциите return реализират интерфейс за обмен на данни между функцията и извикващата програма.

Изискванията към имената на функциите са същите както към имената на променливите – името на функцията не трябва да започва с цифра и може да съдържа букви, цифри, символи за подчертаване „_” или „$”.

Използването на функции предоставя на програмиста следните предимства:

- По-малък по обем и по-четлив код на програмите
- Възможност за програмиране и тестване на функциите като самостоятелни модули.
- Възможност за използване на библиотеки от готови функции.

20.5.1 Деклариране и извикване на функции.

Ако програмистът иска да създаде своя функция, той трябва задължително да я декларира. Без деклариране могат да се използват само вградените в JavaScript функции и функциите, които са методи на създадени обекти и се извикват чрез тях. Както е указано в дадената по-горе

дефиниция, функциите, се декларират посредством ключовата дума function, следвана от име, списък от формални параметри в скоби и тяло (код) на функцията, включено във фигурни скоби. В HTML документ декларацията трябва да е включена в блок <SCRIPT> Да разгледаме един пример:

Да се декларира функция, която да изчислява лицето на триъгълник по зададена страна a и височина към нея h.

```
<SCRIPT Language=”JavaScript”>
    function triangle(a, h){
        document.writeln(“S=”+a*h/2);
    }
</SCRIPT>
```

Пример 74 Деклариране на функция.

Даденият в скрипта код включва декларацията на функцията triangle с два формални параметъра a и h. Тялото на функцията включва само една инструкция, която извежда в прозореца на браузъра низа “S=” и резултата от изчисляването на израза a*h/2.

JavaScript предоставя алтернативен синтаксис за деклариране на функция, при който създаденият обект-функция се присвоява на променлива, чрез която в последствие функцията се извиква за изпълнение. По този синтаксис функцията triangle(a, h) ще се декларира както следва:

```
    var triangle = function(a, h){
        document.writeln(“S=”+a*h/2);
    }
```

20.5.2 Предаване на параметри на функцията и връщане на резултати.

По правило при изпълнението на функция извикващата програма предава на функцията набор от параметри. Параметрите, които се предават на функцията при нейното изпълнение е прието да се наричат фактически параметри.

Достъп до параметрите на функция с псевдо-масива arguments

В JavaScript не е задължително функцията да се извиква със същия брой параметри, с който е дефинирана. В кода на функцията (или, както се

казва, в контекста на функцията) JavaScript предоставя псевдо-масива arguments, който съдържа фактическите параметри, с които е извикана функцията, индексирани по реда на предаването им. Така една функция може да бъде декларирана изобщо без параметри, а да бъде извикана с произволен набор от параметри, както е показано в следващия пример:

```
function a(){
  if(arguments.length>0)alert(arguments[0]); //Извежда първият параметър
}
```

В примера функцията a() е декларирана без параметри, но ако е извикана с параметри, ще изведе стойността на първия параметър.

Все пак arguments не е масив и не предоставя непосредствен достъп до методите, които притежава обекта Array. За извикване на метод на обект Array за псевдо-масива arguments, трябва да се използва специален метод: метода call на обект Function. Например с инструкцията alert([].join.call(arguments)) , изпълнена във функция, ще се извика метод join за псевдо-масива arguments (въпреки че той реално няма такъв) и ще се изведат в диалогова кутия всички фактически параметри, с които функцията е извикана.

Предаване на параметри по стойност

По подразбиране в JavaScript извикващата програма предава параметри на функциите по *стойност*. Стойностите на фактическите параметри при извикване на функцията се копират във формалните параметри*, с които функцията е декларирана, и се използват като входни данни при нейното изпълнение.

При предаването по стойност променливите, които участват при изпълнението на операциите във функцията, са временни променливи и затова променянето на техните стойности не се предава на фактическите параметри след завършването на изпълнението на функцията.

Например ако декларираме функция addition(a1,a2,s), и я извикаме с фактически параметри x, y и z, изменението на стойността на третия параметър s вътре във функцията няма да промени стойността на фактическия параметър z, така че след изпълнението на функцията addition променливите x, y и z ще имат същите стойности, както преди нейното

* В действителност стойностите на параметрите се копират във временни променливи, които се унищожават след изпълнение на функцията, но можем да считаме, че те са асоциирани с формалните параметри. В примера във функцията addition стойностите на x и y се копират в параметрите a1 и a2 и тяхната сума се присвоява на параметъра s.

изпълнение. От това следва, че функцията alert(z) ще изведе диалогова кутия със стойност 0.

```
<HTML.<BODY>
<SCRIPT language="JavaScript">
    function addition(a1,a2,s){
        s= a1+a2;
    }
    x=5;y=7;z=0;
    addition(x,y,z); alert(z);   //Извеждане на стойността на z
</SCRIPT></BODY></HTML>
```

Пример 75 Предаване на параметри по стойност.

Предаване на параметри по адрес

При предаване на параметри по адрес физическият адрес, на който се записват измененията в съдържанието на параметъра вътре във функцията съвпада с физическия адрес на фактическия параметър извън функцията. Поради това измененията на стойността на такъв параметър се запазват след изпълнението на функцията.

Важно е да се запомни, че в JavaScript по адрес се предават само параметрите, които са обектни променливи, т.е. такива променливи, които представляват обекти. Обектният модел на JavaScript ще бъде разгледан в следващите глави. В дадените досега примери сме използвали вече обекта document с неговите методи write и writeln, с които се извежда текст в прозореца на браузъра. Да разгледаме един пример, в който при извикване на функция и се предава като фактически параметър обектът document.

```
<HTML><BODY>
Test Function parameters
<SCRIPT language="JavaScript">
    function setColor(obj)
    {
        obj.fgColor=0x0000ff;
    }
    x=5;y=7;z=0;
    setColor(document);
    alert(document.fgColor); //Извеждане на стойността на fgColor
</SCRIPT>
</BODY></HTML>
```

Пример 76 Предаване на обект като параметър на функция

В примера в декларацията на функцията setColor е зададен формалният параметър obj. При извикване на функцията като неин фактически параметър се предава обектът document. Инструкцията obj.fgColor= 0x0000ff; , включена в тялото на функцията присвоява стойност на атрибута fgColor (подразбиращ се цвят на текста) на обект document. Тъй като обектът document се предава на функцията по адрес, то измененията в него вътре във функцията се запазват след нейтото изпълнение и поради това текста, "Test Function parameters"който ще се изведе в прозореца на браузъра ще бъде със син цвят (кодира се с 0000ff), а в диаловата кутия alert ще с изведе стойността 0000ff.

Подразбиращи се стойности на параметри

JavaScript не дава възможност за задаване на подразбиращи се стойности на параметри. В повечето езици за програмиране такава възможност съществува. По-долу е даден пример за декларацията на функция в езика C с един параметър със зададена подразбираща се стойност и пример за аналогична функция на JavaScript, която при извикване без параметър му задава подразбираща се стойност.

int AddTwo(int x=10)
{
return x+2;
}
 	
Функция на C с параметър с подразбираща се стойност
	function AddTwo(x)
	{
	if(typeof x == 'undefined')x=10;
	return x+2;
	}
	
	Функция на JavaScript, която при извикване без параметър задава подразбираща се стойност на x

Както се вижда от примера, във функцията на JavaScript чрез оператора typeof се проверява дали променливата x е недефинирана, и в тоя случай и се присвоява подразбираща се стойност. Резултата е същия, както във функцията на C, но получен чрез допълнителен код.

Връщане на резултат

Функциите в JavaScript връщат резултат чрез оператора return. Той има следния синтаксис:

return expression;

където expression може да бъде всеки валиден израз на JavaScript, а в частност и просто име на променлива.

Извикващата програма получава резултат от изпълнението на функцията чрез нейното име. Например ако функцията addition трябва да връща резултат, то в нейната декларация трябва да се включи оператор return както следва:

```
<HTML.<BODY>
<SCRIPT language="JavaScript">
    function addition(a1,a2)
    {
        return a1+a2;
    }
    x=5;y=7;
    alert(addition(x,y));        //Извеждане на стойността на addition
</SCRIPT></BODY></HTML>
```

Пример 77 Връщане на резултат от функция.

В примера извикването на функцията addition е включено като параметър при извикването на функцията alert, чрез която се извежда диалогова кутия. Като резултат върнатата посредством името на функцията addition стойност ще се изведе в диалоговата кутия.

Една функция по принцип може да съдържа повече от един оператор return. След изпълнението на оператор return изпълнението на функцията завършва и изпълнението на извикващата програма продължава със следващата инструкция.

20.5.3 Променливи, декларирани във функциите.

В JavaScript променливи могат да се декларират както в кода на функциите, така и извън тях. Променливите, декларирани в скрипта извън декларациите на функциите са глобални – те са достъпни и запазват стойността си (или както се казва още, са видими) в целия код на скрипта. За разлика от тях променливите, декларирани във функциите, са локални – те са достъпни само във функцията, в която са декларирани. Тези променливи се създават при извикване на функцията и се унищожават след завършване на нейното изпълнение. Да разгледаме един пример:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
```

```
function addition(a1,a2)
{
    var s=a1+a2;
    return s;
}
    x=5;y=7;
    z=addition(x,y);
    alert("z="+z);        // Извеждане на стойността на z
    alert("s="+s);        // Извеждане на стойността на s
</SCRIPT>
</BODY></HTML>
```

Пример 78 Деклариране на локални променливи във функция

В примера във функцията addition се декларира променливата s и в същата инструкция се присвоява на s сумата от стойностите на параметрите на функцията – a1 и a2. Операторът return прекратява изпълнението на функцията и връща чрез името на функцията стойността на s. В скрипта извън функцията се присвоява на глобалната променлива z резултата от изпълнението на функцията. Стойностите на z и s се извеждат посредством функции alert. Тъй като обаче променливата s е локална – декларирана вътре във функцията addition – обръщението към нея извън функцията ще генерира грешка и ще се изведе съобщение “s is undefined” (s е недефинирана). Ако превърнем инструкцията alert(“s=”+s); в коментар, като поставим пред нея два символа наклонена черта “/”, останалата част от скрипта ще се изпълни и ще се изведе в диалогова кутия стойността на z.

20.5.4 Статични променливи във функциите.

Статичните променливи са променливи, които запазват стойността си след изпълнението на функцията. В JavaScript няма в явен вид деклариране на статични променливи във функциите, но те могат да бъдат създадени като атрибути (свойства) на функцията, тъй като в JavaScript функциите са обекти. Да разгледаме един пример:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
function showCounter()
{
    showCounter.counter++;          //Отброяване на извикването
    alert(showCounter.counter);     //Извеждане на брояча
}
showCounter();
showCounter();
```

```
</SCRIPT>
</BODY></HTML>
```

Пример 79 Деклариране на статична променлива във функция

В примера при първо извикване на функцията се създава атрибут counter на обекта showCounter на функцията и при всяко извикване на функцията стойността му се увеличава с 1 и се извежда в диалогова кутия alert.

20.5.5 Вградени функции.

JavaScript предоставя на програмиста няколко вградени функции с различно предназначение, които могат да се използват навсякъде без да е необходимо да се декларират. Ще разгледаме най-често използваните от тях

20.5.5.1 Преобразуване на низ в цяло число - функции parseInt и Math.trunc

Функцията parseInt анализира и преобразува символен низ*, зададен като параметър, в цяло число. Празните интервали в началото се пренебрегват. За край на числото се приема краят на низа или първия открит символ, който не означава цифра в избраната бройна система. Ако символният низ не съдържа число, функцията връща стойност NaN (Not a Number).

Синтаксис

parseInt(string[, radix])

където:

string е символен низ, който съдържа число в избраната бройна система

radix [незадължителен] е число, което задава основата на бройната система, в която трябва да е записано числото в низа, напр. 10 за десетична бройна система

Примери:

parseInt("F", 16)	//Числото в низа е зададено в 16-тична бройна система
parseInt("17", 8)	//Числото в низа е зададено в 8-мична бройна система

* Получил е разпространение жаргонът “парсвам”, който според тези, които го употребяват означава “анализирам и разделям”.

parseInt("15", 10)	//Числото в низа е зададено в 10-тична бройна система
parseInt(15.99, 10)	//Числото се преобразува до символа “.”, който не е десетична цифра
parseInt("FXX123", 16)	//Числото се преобразува до символа “X”, който не е шестнадесетична цифра
parseInt("1111", 2)	//Числото в низа е зададено в двоична бройна система
parseInt("15*3", 10)	//Числото се преобразува до символа “*”, който не е десетична цифра

Пример 80 Използване на функция parseInt. Всички функции връщат стойност 15.

Във версия 6 на стандарта ECMAScript е въведена функция Math.trunc (метод на обекта Math), която връща цялата част на зададена числова стойност. Пример: var x= Math.trunc(3.14); //x получава стойност 3. В код на JavaScript, който използва новите възможности, добавени с ECMA6, се препоръчва използването на Math.trunc вместо parseInt.

20.5.5.2 Преобразуване на низ в реално число – функция parseFloat

Функцията анализира и преобразува символен низ, зададен като параметър, в реално число. Празните интервали в началото се пренебрегват. За край на числото се приема краят на низа или първия открит символ, който не е цифра или десетична точка. Ако символният низ не съдържа число, функцията връща стойност NaN (Not a Number).

Синтаксис

parseInt(string)

където:

string е символен низ, който съдържа реално число в десетична бройна система

Примери:

parseFloat("3.14")	//Числото в низа е зададено като десетично реално число
parseFloat("314e-2")	// Числото в низа е зададено в експоненциален формат. Съответстващият му математически запис е 314.10^{-2}
parseFloat("0.0314E+2")	// Числото в низа е зададено в експоненциален формат. Съответстващият му математически

	запис е $0{,}00314.10^2$
var x = "3.14"; parseFloat(x)	//Параметърът на функцията е променлива, която съдържа низа "3.14"

Пример 81 Използване на функция parseFloat. Всички функции връщат стойност 3.14.

20.5.5.3 Изчисляване на стойността на израз – функция eval

Функцията изчислява стойността на израз или изпълнява последователност от инструкции, записани в символен низ. Ако параметърът на функцията не е символен низ, функцията връща стойността на аргумента без изменение.

Синтаксис:

eval(string)

където string е символен низ или символна променлива, съдържащи израз на JavaScript или последователност от инструкции на JavaScript.

Функцията eval е изключително мощна функция. По принцип тя може да изпълни цяла програма на JavaScript, ако последователността от инструкции на програмата бъде записана в символен низ.

Примери:

```
<HTML><BODY>
Test Function eval
<SCRIPT language="JavaScript">
    x=5;y=7;
    t="x+y";
    alert(“x+y=”+eval(t));
</SCRIPT></BODY></HTML>
```

Пример 82 Изчисляване на стойността на израз с функция eval.

В този пример като параметър на функцията eval се задава променлива, която съдържа символен низ, който представлява аритметичен израз. Функцията eval изчислява стойността на израза и получения резултат се извежда в диалогова кутия alert.

```
<HTML><BODY>
Test Function eval
<SCRIPT language="JavaScript">
    x=5;y=7;
    t='z=x+y;alert("x+y="+z)';
```

```
    eval(t);
</SCRIPT></BODY></HTML>
```

Пример 83 Изпълнение на последователност от инструкции чрез функция eval.

В този пример като параметър на функцията eval се задава символен низ, който съдържа последователност от две инструкции. Първата инструкция изчислява стойността на израза x+y и го присвоява на променливата z. Втората инструкция извежда стойността на z в диалогова кутия alert.

20.5.5.4 Проверка дали дадена стойност не е число – функция isNaN

Стойността NaN в JavaScript се използва, за да се обозначи, че дадена числова променлива не съдържа валидна числова стойност. Както беше посочено при описанието на функциите parseInt и parseFloat, те връщат стойност NaN ако символният низ, който им е зададен като аргумент не съдържа число. За програмиста на JavaScript е необходимо в някои случаи да може да провери дали наистина операторът е въвел число или някакъв друг текст. Такава възможност му предоставя функцията isNaN. Тя връща стойност логическа истина (true) ако аргументът на функцията съдържа стойност NaN, и логическа неистина (false) ако аргументът съдържа число. Ако като аргумент на функцията се предаде символен низ, то isNaN връща стойност true ако низът не съдържа число и стойност false ако низът съдържа число.

Синтаксис:

isNaN(testValue)

където:

testValue е константа или променлива, която съдържа числова стойност или символен низ.

Пример:

```
<HTML><BODY>
Test Function isNaN
<SCRIPT language="JavaScript">
    while(isNaN(x=parseInt(prompt("x=","0"))))
    {
        alert("Не е въведено число. Повторете въвеждането");
    }
</SCRIPT></BODY></HTML>
```

Пример 84 Използване на функция isNaN за проверка дали е въведено число.

В примера въведеният чрез диалогова кутия prompt символен низ се преобразува в цяло число от функцията parseInt и се присвоява на променливата x. Операторът за присвояване “=” връща стойността, която се присвоява на x и тя се използва като аргумент на функцията isNaN. Ако получената от parseInt стойност е валидно число, тогава функцията isNaN връща стойност false и цикълът while завършва без да се изпълни инструкцията в тялото на цикъла. Ако получената от parseInt стойност е NaN (т.е. ако операторът не е въвел число) тогава функцията isNaN връща стойност true и тогава се изпълнява инструкцията в тялото на цикъла, която извежда диалогова кутия със съобщение че трябва да се въведе валидна стойност. След затваряне на диалоговата кутия alert се изпълнява отново израза, който е аргумент на while, и се извежда отново диалоговата кутия prompt за ново въвеждане.

20.5.5.5 Кодиране на специални символи в низ – функция escape

Кодира символен низ, така, че да може да бъде добавен към интернет адрес (URL).

Синтаксис:

escape(string)

където string е символен низ или символна променлива.

Функцията escape() връща символен низ, в който буквените символи остават непроменени, а небуквените (напр. празен интервал, =, ?) се кодират с последователност "%xx", където xx е ASCII кода на символа в шестнадесетична бройна система. Например низа “Hello!” ще се се прекодира като “Hello%21”, тъй като ASCII кода на символа “!” е 21_{16}. Не се прекодират символите * @ - _ + . и / .

20.5.5.6 Декодиране на символни последователности %xx – функция unescape

Декодира символен низ, кодиран с функция escape или получен от интернет адрес (URL).

Синтаксис:

unescape(string)

където string е символен низ или символна променлива.

Функцията unescape( ) връща символен низ, в който буквените символи остават непроменени, а символните последователности %xx, ", където xx е ASCII код на символ в шестнадесетична бройна система, се заменят със съответния символ. Например низът “Hello%21” ще се декодира като “Hello!”, тъй като 21 е ASCII кода на символа “!”. Същият резултат ще получим и от прекодирането на низа "%48ello%21”, в който символа “H” е заменен със символната последователност "%48.

Функциите escape( ) и unescape( ) са намерили едно интересно приложение – скриване на JavaScript кода чрез кодирането му в нечетим вид. За повече подробности посетете сайта http://scriptasylum.com/tutorials/encdec/encode-decode.html

21 Обекти в JavaScript

21.1 Обектно-ориентирано програмиране

Основната идея на обектно-ориентираното програмиране е съставянето на програмите от отделни, относително самостоятелни части, наречени *обекти*, за разлика от процедурното програмиране, според което една програма е съставена от само от блокове от инструкции и процедури, които компютърът изпълнява. Всеки обект е способен да получава съобщения, да съдържа и обработва данни и изпраща съобщения на други обекти. [16]. Обектите предоставят програмен интерфейс за използването им, който включва достъпните извън обекта вътрешни променливи (атрибути) и вътрешни функции (методи) и могат да генерират събития, които да извикват за изпълнение външни (за обекта) функции.

Практиката от използването на обектно ориентираните езици за програмиране показва, че основната част от програмите, които програмистите създават, се състои от код на процедури (най-често това са функции, които се изпълняват, когато даден обект генерира определено събитие). Широко се използват готови библиотеки от класове, чрез които се създават обекти, но сравнително рядко се създават обекти чрез декларации, зададени от самия програмист. Съвременните средства за програмиране предоставят на програмиста среда, в която са дефинирани шаблони на определени типове проекти. Шаблоните за проекти включват йерархия от обекти, които се създават с код, който се предоставя на програмиста наготово, а от него се очаква да създаде само кода на функциите, които се изпълняват при възникване на определени събития. Така че на практика процедурно ориентираното програмиране продължава да съществува, но програмистът трябва добре да познава и да може ефективно да използва

обектите, които му се предоставят под някаква форма – най-често като готови библиотеки от декларации на класове и структури, чрез които програмистът може да създава обекти с предварително известни свойства и методи.

21.2 Обектен модел на JavaScript

Обектният модел на JavaScript е прототипно ориентиран. Това означава, че в JavaScript обектите не се създават чрез декларации на класове, така, както това се прави в C++, Java и други широко използвани обектно ориентирани езици. В JavaScript не се декларират явно класове. Поради това се използва термина «тип обект» вместо «клас». Обектите в JavaScript се създават чрез клониране и разширяване на един предварително деклариран базов обект – обектът Object. Съществуват и други прототипно ориентирани обектни езици за програмиране като Cecil, NewtonScript, Io, Slate, MOO, REBOL, Kevo и др. [17] , но те не са получили широко разпространение.

Обектите на JavaScript предоставят на програмиста:

- Атрибути (свойства) – променливи, които са декларирани в обекта. За обекта тези променливи имат някакво специално значение. Например атрибутът bgColor задава фонов цвят на HTML документ, изведен в прозореца на браузъра. Поради това атрибутите се наричат още свойства на обекта.
- Методи – функции, които са декларирани в обекта, чрез които могат да се изпълняват определени действия. Например използваната в много примери функция alert, която извежда диалогова кутия с текст и бутон с надпис “OK”, е метод на обект window.
- Манипулатори на събития – специален тип променливи, на които се присвояват функции, дефинирани от програмиста извън обекта. Обектът предизвиква изпълнението на функцията, присвоена на манипулатор, при възникване на определено за манипулатора събитие, например щракване с мишката върху бутон в HTML документ, изведен в прозореца на браузъра. Поради това такива функции по-нататък ще наричаме събитийни процедури.

Атрибутите, методите и манипулаторите на събития предоставят възможност на останалата част от скрипта да взаимодейства с обекта и поради това се наричат интерфейс на обекта.

Операциите с атрибути, методи и манипулатори на събития на обекти се извършват посредством оператора “.” , като левият операнд е обектната

променлива (името на обекта), а десният – съответно атрибутът, методът или манипулаторът на събитие.

Примери:

```
window.status=”Hello”;                  //Присвояване на стойност на атрибут
window.attributes[“status”]=”Hi”        //Присвояване на стойност на атрибут
window.alert(”Hello window”);           //Извикване на метод
window.onclick=MyClick;                 //Присвояване на функция на манипулатор //на събитие
```

Пример 85 Достъп до атрибути, методи и манупулатори на събития на обекти в JavaScript.

Първата и втората инструкция присвояват символен низ на атрибута status на обект window. Резултатът е извеждане на съобщение в статус-лентата на брозореца на браузъра. Във втората инструкция достъпът до атрибута става чрез масива attributes. Третата инструкция извиква метода alert на обект window, като му предава като параметър символен низ. Резултатът е извеждане на диалогова кутия, в която се извежда низа и бутон с надпис “OK”. Тъй като в случая обектът window се подразбира, много често (както е в разгледаните досега примери) той се пропуска. Четвъртата инструкция присвоява на манипулатора onclick на обект window функцията MyClick. Тази функция трябва да е дефинирана преди това в скрипта. Като резултат, при щракване с мишката в прозореца на браузъра ще се изпълни функцията MyClick, която по този начин изпълнява ролята на събитийна процедура за това събитие.

21.3Създаване на обекти. Конструктор на обекта.

За използване на даден обект в JavaScript, е необходимо той да бъде създаден в оперативната памет на компютъра. Ако обектът не е предварително създаден, програмистът трябва да го създаде посредством инструкция, в която се определя типа на обекта. Обектът, създаден в оперативната памет по тази инструкция притежава атрибутите и методите на обектния тип. В JavaScript обектите се създават посредством оператора new и предварително декларирана функция (метод на типа обект), името на която съвпада с името на типа на обекта. Тази функция се нарича конструктор на обекта.

D6 Конструкторът в JavaScript е метод, посредством който и оператора new се създава обект от зададен тип. Името на метода задава името на създавания обект.

По правило конструкторът е функция без параметри, но някои типове обекти, например обектът Date притежават конструктор, който може да се изпълнява с различни параметри*.

Да разгледаме един пример за конструиране на обекти:

```
var myObject = new Object( ); //конструктор без параметри
var myMess= new String(“Hello JavaScript”); //конструктор с параметър
```

Пример 86 Създаване на обекти посредством оператор new и конструктор

Първата инструкция създава обект от тип Object като използва конструктор без параметри. Втората инструкция създава обект от тип String, като му предава като параметър символен низ.

21.4Създаване на обекти чрез потребителска функция-конструктор

В JavaScript дефинирана от програмиста функция също може да се използва като конструктор за създаване на данни. При създаване на обект чрез оператора new и функцията, се инициализира променлива със запазеното име this, която е обектна променлива (референция) за създавания обект. Чрез функцията-конструктор могат да бъдат създадени публични и частни атрибути и методи на обекта.

- Публични (public) атрибути и методи на обекта: Разширяването на обекта this, чрез добавяне на атрибути и методи създава публични (т.е. достъпни извън кода на обекта) атрибути и методи на обекта, създаден чрез функцията-конструктор.
- Публични (public) атрибути и методи на прототипа на функцията-конструктор: Разширяването на прототипа prototype на функцията-конструктор чрез добавяне на атрибути и методи създава публични атрибути и методи на всички обекти, създадени чрез функция-конструктор.
- Частни (private) атрибути и методи: Променливите и функциите, създадени вътре във функцията-конструктор са локални и не са достъпни извън кода на обекта.

Да разгледаме един пример:

```
function Person(name){
```

* Обектния модел на JavaScript не включва полиморфизъм, т.е. възможност за деклариране на няколко функции с едно и също име, но с различен набор от параметри. Вместо това е предвидена възможност за извикване на функция с брой параметри, различен от този, с който е била декларирана.

```
    this.name = name; //публичен атрибут
    this.sayHello = function() { //публичен метод sayHello
       return "Hello " + this.name+" version is "+version;
    }
      version=1; //частен (private) атрибут;
}
var Peter = new Person("Peter"); //Създаване на обект
alert(Peter.sayHello());
```

Създаване на обект чрез функция-конструктор

В случай, че обектът трябва да притежава само атрибути, той може да бъде създаден с конструктор със следния по-прост синтаксис: списък от двойки "име_на_атрибут:стойност_на_атрибут" във фигурни скоби с разделител ":" между името и стойността. Да разгледаме един пример:

```
var user={name: "John ", email: "j@abv.bg"}
alert("user name: "+user.name+ ", user email: "+user.email);
```

В примера се създава обект user с два атрибута – name и email. С втората инструкция стойностите на атрибутите се извеждат в диалогова кутия.

21.5 Вградени обекти на JavaScript

JavaScript предоставя на програмиста набор от вградени обекти. За тези обекти в езика са включени декларации и те могат да се използват независимо от това дали скриптът е вложен в HTML документ или не.

21.5.1 Прототипът на обектите в JavaScript - обект Object

Обектът Object е базов обект на JavaScript. Той е прототип на останалите обекти и поради това всички обекти в JavaScript притежават неговите атрибути и методи.

Конструктор:

Обектът Object притежава само конструктор без параметри.

new Object();

Атрубути на обект Object:

constructor	Представя функцията – конструктор на обекта.
prototype	Представя прототипа на дадения обект. Добавянето на атрибути или методи чрез prototype ги добавя към всички

	инстанции на дадения тип обект.

Атрибут constructor

Значение: Предоставя достъп до функцията – конструктор на обекта.

Описание: Атрибутът constructor може да се използва за обръщение към метода – конструктор на обекта, но това може да стане и чрез непосредствено използване на името на конструктора.

Атрибут prototype

Значение: Предоставя достъп до прототипа на дадения обект

Описание: Атрибут prototype притежават всички обекти, които се създават чрез конструктор. Ако се добави атрибут или метод към прототипа на обекта, то всички обекти от този тип ще притежават добавения атрибут или метод. Важно е да се запомни, че prototype е атрибут на обектния тип (в случая Object), а не на обектите, създадени чрез неговия конструктор, т.е. Object.prototype е правилното обръщение към атрибута prototype.

Пример:

```
<HTML><BODY>
Test prototype property<HR>
<SCRIPT language="JavaScript">
    Object.prototype.myProperty=”New Property”; //Добавяне на атрибут
                                                 // към прототипа
    var myObject = new Object()      //Създаване на обект
    alert(myObject.myProperty);      //Извеждане на стойността атрибута
</SCRIPT></BODY></HTML>
```

Пример 87 Добавяне на атрибут към всички обекти от даден тип посредством атрибута prototype**.**

В дадения пример чрез атрибута prototype към прототипа на обекта Object се добавя нов атрибут myProperty и му се присвоява низа ”New Property”. След това се създава нов обект myObject чрез оператора new и конструктора Object( ) и се извежда стойността на неговия атрибут myProperty. Тъй като атрибутът myProperty е прибавен към прототипа на Object, то новосъзданеният обект myObject има такъв атрибут и той има зададената стойност ”New Property”.

Методи на базовия обект Object:

Метод toString()

Значение: Връща символен низ, представящ дадения обект

Описание: Всеки обект в JavaScript притежава метод toString, който се извиква автоматично когато обектът трябва да бъде представен като символен низ. Например във втората инструкция от дадения по-долу пример името на обекта участва в операция слепване на символни низове и трябва да бъде преобразувано в символен низ.

```
var myObject = new Object()        //Създаване на обект
document.write("Обектът е " + myObject)
```

Пример 88 Преобразуване на обект в символен низ с автоматично извикване на метод toString()

По подразбиране, методът toString се наследява от всеки обект, създаден чрез разширяване на обекта Object. Методът връща символен низ «[object type]», където type е типа на обекта. При създаване на потребителски обекти методът toString може да бъде надписан.

Метод valueOf()

Значение: Връща стойност (числова, символна или логическа), съответстваща на типа на обекта.

Описание: Всеки обект в JavaScript притежава метод valueOf, който се наследява от обекта Object. Всички вградени обекти на JavaScript предефинират метода valueOf за да може той да връща стойност, съответстваща на съответния тип обект. Ако типа обект няма присвоена стойност (както е за обекта Object) методът връща низа “[object Object]”.

21.5.2 Операции с масиви. Обект Array

Всички езици за програмиране дават възможност за използване на масиви. Могат да се цитират различни определения за масиви като например определението за масив в програмния език C++ [10] : “Масивите представляват поредици от наредени последователно в паметта елементи (променливи) от един и същ тип, които могат да бъдат адресирани поотделно, като се добави индекс към уникалното име, с което са декларирани”. В JavaScript масивът не е просто индексирана променлива, а обект, който не само предоставя достъп до елементи с едно и също име и тип на данните и с различни индекси, а предоставя на програмиста

атрибути и методи за операции с елементите на масива. JavaScript, като изцяло обектно ориентиран език, използва по-високо ниво на абстракция на данните, което означава по-голяма независимост на логическото представяне на данните от тяхното физическо представяне. За масивите в JavaScript може да се даде следната дефиниция:

D 7 Масивът в JavaScript е подредено множество от елементи, асоциирано с име на обектна променлива.

Конструктори:

Обектът Array притежава два вида конструктори:

a) Конструктор, който създава празен масив със зададен брой елементи:

new Array([arrayLength])

където arrayLength задава броя елементи на масива

b) Конструктор, който създава масив като инициализира (т.е. задава начални стойности) на елементите:

new Array([element0[, element1[, ..., elementN]]])

където element0, ... , elementN е списък от стойности на елементите.

Поведението на конструктора когато е зададен само един параметър зависи неговата стойност. Ако параметърът е число, то се преобразува в цяло число и получената стойност задава броя елементи на масива. В тоя случай елементите на създадения масив са неинициализирани, т.е. те нямат присвоена начална стойност. Ако параметърът не е число, то се създава масив с един елемент и той се инициализира със зададената стойност на параметъра.

c) Алтернативен литерален конструктор: списък от стойности в квадратни скоби. Като предимство освен по-краткия запис може да се посочи, че няма разлика в резултата, когато се създава масив с един или повече елементи. Например инструкцията var a=[12]; създава масив с един елемент със стойност 12.

Примери:

var M1 = new Array(10); //Създава обект масив (Array) с 10 неинициализирани елемента.

var M2 = new Array(“10”); //Създава обект масив (Array) с един елемент, съдържащ символния низ “10”.

var M3 = new Array(10,20); //Създава обект масив (Array) с 2 елемента със стойности 10 и 20.
var M4 = new Array(“first”,”second”,”third”); //Създава обект масив (Array) с 3 елемента съдържащи символните низове “first” , ” second” и ” third”.
var M5 = [11, “Hello”, “World”]; //Създава масив с три елемента, съдържащи една числова и две символни стойности.

Достъпът до елементите на масива се осъществява посредством техния пореден номер (индекс), като индексирането започва от стойност 0. Например можем да запишем стойност в първия елемент на масива M1 посредством инструкцията:

```
M1[0]=0;
```

Атрибути на обект Array

наследени от обекта Object: constructor, prototype

Атрибут length

Значение: Задава и връща дължината на масива.

Описание: Атрибутът length представя дължината на масива като 32 битова стойност без знак. Посредством този атрибут може по всяко време да се промени дължината на масива. Ако на length се зададе по-малка стойност от текущата дължина на масива, елементите с индекси, които са по-големи от новата стойност на length-1 губят стойността си. Ако на length се зададе по-голяма стойност от текущата дължина на масива, елементите с индекси, които са по-големи от старата стойност на length-1 получават стойност undefined.

Методи на обект Array:

Обединяване на масиви. Метод concat

Значение: Обединява два или повече масива и връща нов масив.

Синтаксис:

concat(масив2, масив3, ..., масивN)

където масив2, масив3, ..., масивN е списък от имена на масиви, които се обединяват с масива, над който се изпълнява метода.

Описание: Методът създава нов масив, който включва в себе си елементите на масива, на когото принадлежи метода и елементите на масивите, зададени като параметри на метода.

Пример:

```
alpha=new Array("a","b","c");
numeric=new Array(1,2,3);
alphaNumeric=alpha.concat(numeric); // създава масив ["a","b","c",1,2,3]
```

Пример 89 Обединяване на масиви посредством метода concat

Обединяване на елементите на масива в символен низ. Метод join

Значение: Връща символен низ, който съдържа стойностите на елементите на масива.

Синтаксис:

join(*разделител*)

където *разделител* е символен низ, който се слепва между стойностите на елементите на масивите.

Описание: Методът получава и връща символен низ, съставен от стойностите на елементите на масива, като между всеки две стойности се добявя низа *разделител*. Най-често този метод се използва при извеждане на съдържанието на масива като низ от стойности.

Пример:

```
a = new Array("Wind","Rain"); //Декларация на масива
myVar1=a.join()       // присвоява низа "Wind,Rain " на myVar1
myVar2=a.join(", ")   // присвоява низа "Wind, Rain " на myVar2
```

Пример 90 Получаване на символен низ от съдържанието на елементите на масив и зададен разделител посредством метода join.

Извличане на елементи от масив. Методи shift и pop.

Значение: Методът shift извлича елемент от началото на масива. Методът pop извлича елемент от края на масива.

Синтаксис:

shift( )
pop( )

Описание: Методът shift извлича първия елемент от масива и връща стойността му. Индексите на останалите елементи се намаляват с единица, така, че вторият елемент става първи, третият става втори и т.н. Методът pop извлича последният елемент от масива и връща стойността му. И двата метода намаляват дължината на масива с 1.

Пример:

```
a = new Array("Wind","Rain","Fire"); //Декларация на масива
document.writeln("Първи елемент, извлечен с shift = "+a.shift();
document.writeln("Последен елемент, извлечен с pop = "+a.pop();
```

Пример 91 Извличане на първи и последен елемент от масив с методи shift и pop.

Извличане на част от елементите от масив. Методи slice.

Значение: Методът slice извлича част от елементите от масива и връща масив, който съдържа извлечените елементи.

Синтаксис:

slice (*начален_индекс*[*, краен_индекс*])

Описание: Методът slice извлича от масива елементите от зададен начален индекс до зададен краен индекс (от крайният индекс не се извлича). Ако *краен_индекс* липсва, се извличат елементите до края на масива. Масивът не се променя. Методът връща масив, в който са копирани извлечените елементи.

Пример:

```
a = new Array(1,2,3,4,5); //Декларация на масива
document.writeln("Елементите с индекси 1, 2 и 3, извлечени със slice = "+a. slice(1,3); // Извежда низа "2,3,4"
```

Пример 92 Извличане на елементи от масив с метод slice.

Добавяне на елементи в масив. Метод push.

Значение: Методът push добавя елементи в края на масива.

Синтаксис:

push(елемент1, елемент2, ..., елементN)

Описание: Методът push добавя елементите елемент1, елемент2, ..., елементN в края на масива. Методът връща последния добавен елемент.

Пример:

```
a = new Array(1,2,3,4,5); //Декларация на масива
a.push(6,7,8);
document.writeln("Съдържание на масива с добавени елементи с метод push = "+a ); // Извежда низа "2,3,4"
```

Пример 93 Добавяне на елементи в масив с метод push.

Инвертиране на елементите в масив. Метод reverse.

Значение: Методът reverse записва елементите в масива в обратен ред.

Синтаксис:

reverse( )

Описание: Методът reverse записва елементите в масива в обратен ред, така, че последният елемент става първи, предпоследният – втори и т.н.

Пример:

```
a = new Array(1,2,3,4,5); //Декларация на масива
a.reverse( );
document.writeln("Съдържание на след инвертиране с метод reverse = "+a ); // Извежда низа "5,4,3,2,1"
```

Сортиране на елементите на масив. Метод sort.

Значение: Методът sort сортира елементите в масива в нарастващ лексикографичен ред или с използване на функция за сравнение.

Синтаксис:

sort([функция_за_сравнение])

Описание: Методът sort е мощен метод за сортиране на елементите на масива в зададен ред. Ако не е зададена функция за сравнение сортирането се извършва като се преобразуват стойностите на елементите в символни низове и се сортират в нарастващ лексикографичен ред (както това се прави в речниците). При такова сортиране елемент със съдържание 10 ще се сортира преди елемент със съдържание 2, тъй като те се преобразуват в символни низове, и при сравнение на първите символи символът "1" от първия низ има код, по-малък от кода на символа "2" от втория низ. Този ред не е еквивалентен на сортиране по числови стойности (тогава очевидно

2 трябва да се сортира преди 10), следователно методът не трябва да се използва за сортиране на масив, който съдържа числа без да се зададе подходяща функция за сравнение. За подреждане по големина в нарастващ ред, функцията за сравнение трябва да връща стойност, по-малка от нула когато a < b, стойност 0 – когато a = b и стойност по-голяма от нула когато a > b, където a и b са параметрите на функцията за сравнение. Както е показано в дадения по-долу пример, за подреждане на числови стойности в нарастващ ред функцията за сравнение може просто да изчислява и връща разликата от стойностите на двата параметъра a – b.

Пример:

```
<HTML><BODY>
Test Array sort method<HR>
<SCRIPT language="JavaScript">
function compareNumbers(a, b) {   return a - b; }
myArray = new Array(2,5,10,7,4);
myArray.sort();
document.writeln("<BR>Масив, сортиран без функция за сравнение " +
myArray);
myArray.sort(compareNumbers);
document.writeln("<BR> Масив, сортиран с функция за сравнение " +
myArray);
</SCRIPT></BODY></HTML>
```

Пример 94 Сортиране на елементите на масив с метод sort

В примера елементите на масива със стойности 2,5,10,7,4 първоначално се подреждат в нарастващ лексикографичен ред “10,2,4,5,7” чрез изпълнение на метода sort() без параметър. Методът writeln на обект document ще изведе съдържанието на масива като низ “10,2,4,5,7”. При сортиране с използване на функцията за сравнение compareNumbers, която е декларирана в скрипта, елементите на масива се подреждат по големина “2,4,5,7,10” като числови стойности.

Итерация на масиви. Цикъл for-in

JavaScript предоставя на програмистите една специфична конструкция за организиране на цикли, предназначена за итерация на атрибути на обекти, и в частност – на елементите на масиви. Това е конструкцията for-in, която има следния синтаксис:

```
for (variable in object)
  {
    //блок инструкции (тяло на цикъла)
```

}

където:

variable е променливата, в която при всяко изпълнение на цикъла се записва името на поредния атрибут на обекта (за месивите – индекса на поредния елемент от масива

object е обектната променлива за обекта, за който се извършва итерацията

Предимството на цикъла for-in пред цикъла for е в това, че ако се извършва итерация през всички елементи на масива (или в общия случай през всички атрибути на обекта) не е необходим брояч на цикъла. В дадения по-долу пример се извършва итерация през елементите на масива array1, като при всяко изпълнение на цикъла, в променливата field се записва индексът на поредния елемент от масива. В променливата s при всяко изпълнение на цикъла се добавя кода на ред от HTML таблица, като в първата клетка от реда се извежда индекса на елемента от масива, а във втората клетка – стойността на съответният елемент от масива.

```
<HTML><HEAD>
<script>
function a()
{
 array1=new Array(1,7,4,2.2); //Деклариране на масив
 s="<TABLE BORDER><TR><TD> field
</TD><TD>value</TD></TR>\n";
 for(field in array1){
  s+="<TR><TD>"+ field +"</TD><TD>"+ array1[field]+"</TD></TR>\n";
 }
 s+="</TABLE>";
document.write(s);  //Извеждане на таблицата в прозореца на браузъра
}
</script>
</HEAD>
<BODY>
<input type="BUTTON" value="Iteration" OnClick="a()">
</BODY></HTML>
```

Пример 95 Итерация на масив с конструкция for-in

След изпълнението на цикъла HTML таблицата се извежда в прозореца на браузъра с метода write на обект document.

При използване на цикъла for-in трябва да се има предвид, че той в действителност не е създаден за итерация на масиви, а за итерация на

атрибутите на обекти. Поради това, ако към масива се добави атрибут или метод, цикълът ще се изпълни и за него – т.е. резултатът от изпълнението на цикъла for-in ще се различава от резултата от изпълнението на цикъл for с брояч (индекс), който се променя от 0 до length-1 . В дадения по-долу пример се прави итерация на масив с цикъл for.

```
<HTML><HEAD>
<script>
function a()
{
 array1=new Array(1, 7, 4, 2.2); //Деклариране на масив
 s="<table border><tr><td>index</td><td>value</td></tr>\n";
 for(index=0; index < array1.length; index ++){
  s+="<tr><td>"+ index +"</td><td>"+ array1[index]+"</td></tr>\n";
 }
 s+="</table>";
document.write(s);  //Извеждане на таблицата в прозореца на браузъра
}
</script>
</HEAD>
<BODY>
<input type="button" value="Iteration" OnClick="a()">
</BODY></HTML>
```

Пример 96 Итерация на масив с конструкция for

При итерация на масива с конструкция for е сигурно, че итерацията се прави само за елементите на масива, тъй като те се адресират с целочислен индекс.

21.5.3 Операции със символни низове. Обект String

Обектът String е обвивка (wrapper) на типа данни “символен низ”. Това означава, че всеки обект от тип String включва в себе си символен низ, но освен това предоставя атрибути и методи за операции със символния низ. Едно от основните приложения на JavaScript е проверката на валидността на данните, въвеждани от потребителя в HTML формуляри, което се извършва посредством операции със символни низове. Поради това за програмиста на JavaScript е важно да познава добре възможностите, които ни дава обект String.

Конструктор:

new String( )

new String(символен_низ)

Ако се използва конструктор без параметри се създава обект от тип String, който съдържа празен низ.

Атрибутите и методите на обект String са достъпни и за променливи, които са инициализирани със символни константи, защото за тях JavaScript създава временен обект от тип String. Разликата между обектите от тип String и символните променливи е само в това, че ако добавим нов атрибут или метод към обект от тип String, създаден с конструктор, той ще принадлежи само на него, а атрибут или метод, добавен към символна променлива принадлежи на всички символни променливи.

Примери:

var S1 = new String( ); //Създава обект от тип String, съдържащ празен низ

var S2 = new String(“Hello World”); //Създава обект от тип String, съдържащ низа “Hello World”.

var S3 = “Hello”; // Създава символна променлива, която притежава атрибутите и методите на обект String

Атрибути на обект String:

Освен атрибутите constructor и prototupe, наследени от базовия обект Object, обектът String притежава само един атрибут – атрибутът length.

Атрибут length

Атрибутът length връща дължината на символния низ. Стойността на атрибута е 32 битово число без знак. Ако низът е празен, стойността на атрибута е 0.

Методи на обект String

Обектът String има над 30 метода. По-долу е направена класификация на методите в съответствие с тяхното предназначения и е дадено описание и примери за използването им.

Методи за търсене в символен низ – indexOf и lastIndexOf

Значение: Методите търсят последователност от символи (подниз) в символния низ на обекта от тип String.

Синтаксис:

indexOf(подниз [, начален_индекс])

lastIndexOf(подниз [, начален_индекс])

Описание: Методът indexOf търси зададен подниз в низа на обекта, като започва търсенето от начален_индекс или от началото на низа, ако незадължителния параметър начален_индекс не е зададен. Търсенето е по посока към края на низа. При първо откриване на подниза методът връща неговия начален индекс. Ако поднизът не бъде открит, методът връща -1.

Аналогично методът lastIndexOf търси търси зададен подниз в низа на обекта и връща началния индекс на първия открит подниз или -1. Разликата между lastIndexOf и indexOf е, че lastIndexOf търси подниза в обратна посока – от начален_индекс към началото на низа или (ако начален_индекс не е зададен) от края на низа към началото на низа.

Пример:

```
var S =new String(“Hello World”);
document.writeln(“Индекс на първия символ ‘o’ “ + S.indexOf(“o”));
document.writeln(“Индекс на последния символ ‘o’ “ + S.lastIndexOf(“o”));
```

Пример 97 Търсене в символен низ с методите indexOf и lastIndexOf

В дадения пример се създава обект от тип String, който съдържа низа “Hello World”. Методът indexOf(“o”) започва да търси низа “o” отляво надясно и затова открива символа “o” в края на “Hello” и връща стойност 4. Методът lastIndexOf(“o”) започва да търси низа “o” отдясно наляво и затова открива символа “o” в “World” и връща стойност 7.

Търсене и заместване чрез регулярен израз – методи search, test и replace

Значение: Методът search търси съответствие на регулярен израз (шаблон за сравнение) в символния низ на обекта от тип String и връща началния индекс на първия открит подниз, за който е получено съответствие. Ако не бъде открит подниз, съответстващ на регулярния израз, методът връща -1. Подобно е предназначението на метода test(string) на обект RegExp (регулярен израз), който връща лог. стойност true ако се открие съвпадение и false в пр. случай.

Методът replace търси съответствие на регулярен израз в символния низ на обекта от тип String и при откриване на съответствие замества първия открит съответстващ подниз със зададения заместващ_низ. Ако регулярния израз е зададен с флаг “g” се извършва заместване на всички съответстващи поднизове.

Синтаксис: search(регулярен_израз)

replace(регулярен_израз , заместващ_низ)

Описание: Регулярният израз е символен низ, който задава шаблон за сравнение. Посредством регулярен израз могат да се задават много сложни шаблони, (които обаче изглеждат като доста неразбираема последователност от символи). Детайлното им разглеждане излиза извън рамките на настоящия курс. Ще дадем два примера за регулярни изрази, които ще използваме за да демонстрираме действието на метода replace:

Конструиране на регулярен израз

Регулярен израз може да бъде конструиран по два начина:

- Чрез инициализиране със символна константа, ограничена със символи "/", например:

  re = /ab+c/

- Чрез функцията-конструктор на обекта RegExp, например:

  re = new RegExp("ab+c");

И в двата случая се създава обект от тип RegExp. Най-простият регулярен израз е символен низ (променлива или константа), който се замества еднократно. Да разгледаме един пример:

```
re = /apples/;
str = "apples are round, and apples are juicy.";
newstr=str.replace(re, "oranges");
document.write(newstr);
```

Пример 98 Използване на метода replace с регулярен израз за еднократно заместване.

Този пример показва най-простото възможно използване на метода replace. Регулярният израз, зададен като първи параметър е просто символен низ. При първо откриване на този низ в обекта от тип String (в случая това е низа str*) той се замества с низа, зададен като втори параметър – в случая това е "oranges". Като резултат методът ще върне низа " oranges are round, and apples are juicy."

Трябва да се има в предвид, че символите "[,] , (,) , + , * , ^ , $, \ , |" са специални символи (метасимволи) за регулярните изрази. Ако искаме те да бъдат търсени и заместени като обикновени символи, трябва да в регулярния израз преди тях да включваме символа "\\". Например ако искаме да бъде открит и заместен низа "1+1=2" трябва да го запишем в регулярния израз във вида "1\\+1=2". Защо обаче наклонените черти са две,

* По точно за символната променлива str се създава временен обект от тип String.

а не една, както е в действителност изискването за специалните символи в регулярните изрази? Да си припомним, че за символните константи в JavaScript “\” е начало на кодова последователност, с която се задава специален символ – например “\n” означава символа за преминаване на нов ред – CR, а “\\” за интерпретатора на JavaScript означават една наклонена черта.

Регулярните изрази обаче ни дават много повече възможности от простото еднократно заместване при точно съвпадение.

Съвпадение с всеки символ. Метасимвол “.”

Метасимволът “.” , включен в регуляен израз, дава съответствие със всеки символ. Например регулярният израз /.12/ дава съвпадение със всеки низ от три символа, ако завършва с “12”.

Повторения – метасимволи “*” , “+” , “?”, {n}, {n,m} и {n,}

Регулярните изрази предоставят няколко метасимвола, с които може да се укаже съответствие (съвпадение с шаблона) при определен брой повторения на зададен символ Например метасимволът “*” , включен в регулярен израз, дава съответствие при нулево или многократно повторение на предшестващия символ. Например регулярния израз /sa*/ ще даде съответствие с низове “s”, “sa”, “saa”, “saaa” и т.н. Метасимволът “+” , включен в регулярен израз, дава съответствие при единично или многократно повторение на предшестващия символ, т.е изисква се символът да е включен поне веднъж. Например регулярния израз /sa+/ ще даде съответствие с низове “sa”, “saa”, “saaa” и т.н, но няма да даде съответствие с низ “s”.

`*`	Нулево или многократно повторение на предшестващия символ.
`+`	Единично или многократно повторение.
`?`	Нулево или единично повторение.
`{`*n*`}`	Повторение *n* пъти.
`{`*n*`,` *m*`}`	Повторение не по-малко от *n* , но не повече от *m* пъти.
`{`*n*`,}`	Повторение не по-малко от *n* пъти.

Класове в регулярните изрази

Класовете се задават като списък от символи в квадратни скоби, например [a,b,c,d,e] или като последователни символи със зададен начален и краен символ, например [a-e]. При търсене ако символа на дадената

позиция в низа съвпада с един от символите от класа, се счита, че има съответствие. Например ако регулярния израз е /[a-z]12/ , то ще се получи съответствие със всеки низ, който започва с малка буква от латинската азбука и завършва с 12, т.е с “a12”, с “b12” ... с “z12”.

Метасимволите не са активни вътре в класовете. Например [akm$] ще даде съвпадение с всеки от символите "a", "k", "m", или "$"; Символът "$" по правило е метасимвол, но вътре в класа от символи той е лишен от особената си природа. Да разгледаме един пример:

```
<HTML><BODY onclick="a();">
<script>
function a(){
    var inVal = "12115-33"; //низ, който се проверява
    re=new RegExp("[0-9]{5}-[0-9]{2}"); //обект Regular expression
    alert(re.test(inVal)); //извеждане на резултата
}
</script></BODY></HTML>
```

Пример 99 Използване на метода test(string) на обект RegExp

С кода от примера се проверява дали в inVal се съдържа символен низ във формат XXXXX-XX където X е десетична цифра. Със зададената примерна стойност "12115-33" функцията alert извежда стойност “true”, но ако се замести някоя буква с цифра, или се промени броят на цифрите или символа „-„ се извежда стойност “false”.

Използване на отрицание в класовете

Отрицание в клас се задава чрез добавянето на "^" като първи символ от класа. Отрицанието означава, че ще има съответствие само ако сравняваният символ не съвпада с никой от символите на класа. Например, [^5] ще даде съвпадение с всеки символ, с изключение на "5". Символът "^" означава отрицание само ако е първи символ в клас. Поставен където и да е другаде, "^" има значение на обикновен символ, който изисква точно съвпадение.

Котви “^” и “$” в регулярните изрази

Котвите са специални символи, с които се задава търсене от началото или в края на на текстов ред. Търсенето от началото на ред се указва със символа “^” (същия, с който се задава отрицание в класовете), поставен в началото на регулярния израз. Търсенето на низ, съдържащ се в края на текстов ред се задава със символа “$”, поставен в края на регулярния израз. Ако символа “^” не е в началото или съответно символа “$” не е в края на

регулярния израз, те нямат значението на котви. Например /^A/ е шаблон, който търси символа “A” в началото на текстовите редове, докато /A^/ е шаблон, който търси низа “A^” навсякъде в текста. В един шаблон могат да фигурират и двете котви, напр. регулярният израз /^B$/ търси символа “B” и в началото и в края на текстовите редове.

Специални последователности с “\”.

В регулярните изрази са дефинирани няколко специални последователности, започващи със символа “\”. Те предоставят възможност за по-кратък запис на шаблон, който можем по принцип да зададем и като клас.

\d	Дава съвпадение с всяка десетична цифра; това е еквивалент на класа [0-9].
\D	Дава съвпадение с всеки символ, който не е цифра; това е еквивалент на класа [^0-9].
\s	Дава съвпадение с всеки празен символ; това е еквивалент на класа [\t\n\r\f\v].
\S	Дава съвпадение с всеки не-празен символ; това е еквивалент на класа [^ \t\n\r\f\v].
\w	Дава съвпадение с всеки буквеноцифров символ; това е еквивалент на класа [a-zA-Z0-9_].
\W	Дава съвпадение с всеки не-буквеноцифров символ; това е еквивалент на класа [^a-zA-Z0-9_].
\b	Дава съвпадение за символ, който е преди или след дума. Така рег. израз /\bHarry\b/ ще даде съвпадение за отделна дума “Harry”.

Флагове g и i

Регулярните изрази допускат използването на двата флага поотделно и в комбинация “gi”:

Флаг g означава глобално търсене. Той е от значение при изпълнението на метода replace. Ако е зададен, ще се извърши заместване при всяко откриване на съответствие на регулярния израз с част от зададения низ.

Флаг i задава нечувствителност към регистъра, т.е. не се прави разлика между малки и големи букви. Да разгледаме един пример:

```
re = /apples/gi;
str = "apples are round, and Apples are juicy.";
newstr=str.replace(re, "oranges");
document.write(newstr);
```

Пример 100 Използване на регулярен израз с ключове g и i.

Регулярният израз в примера включва ключовете g и i, и поради това ще замести всички поднизове “apples” в низа str с “oranges”, независимо от това, дали са изписани с малки или с главни букви. Като резултат ще се получи и ще се изведе низа " oranges are round, and oranges are juicy."

Задаване на флагове във функцията-конструктор на обекта RegExp

Ако регулярният израз се създава с функцията-конструктор на обекта RegExp, тогава флаговете се задават в нея в символен низ като втори параметър. Например регулярният израз от горния пример може да се създаде с инструкцията

re=new RegExp("apples", "gi");

Като използваме даденото дотук за регулярните изрази, нека да съставим регулярен израз, който да дава съответствие ако символния низ съдържа валиден e-mail адрес и несъответствие в противен случай. Валидният e-mail адрес трябва да започва със символ, който е буква или цифра и след него може да следват още няколко такива символа. Поради това започваме регулярния израз с “\w+”, където “\w” дава съответствие за буквено-цифров символ, а “+”дава съответствие за един символ или повторение. След тези символи валидния e-mail адрес трябва да включва символа “@”, който се включва в регулярния израз за да се изисква точно съвпадение. След това трябва да следват поне два или повече буквено-цифрови символи, и поради това включваме в регулярния израз “\w\w+”. След това следва символа “.”, който се включва предшестван от “\”, за да не се приеме за специален символ. E-mail адреса завършва с поне два буквено-цифрови символа. Така регулярният израз за проверка на e-mail адрес добива вида:

```
/ \w+@\w\w+\.\w\w+ /
```

Прочитане на символ и код на символ от зададена позиция – методи charAt и charCodeAt

Значение: Методът charAt прочита и връща символа от зададена позиция на символния низ на обекта от тип String. Методът charCodeAt извлича и връща кода на символа от зададена позиция на символния низ.

Синтаксис:

charAt(индекс)
charCodeAt (индекс)

Описание: Символите в низа са индексирани отляво надясно. Първият символ има индекс 0, а последният– с единица по-малък от дължината на низа (length-1). Методът charAt прочита символа от зададена позиция на символния низ и връща символен низ, който съдържа само този символ. Методът charCodeAt прочита символа от зададена позиция на символния низ и връща Unicode кода на символа (цяло число без знак в интервала от 0 до 65,535). Първите 128 Unicode стойности са същите както кода на символите от кодова таблица ASCII. Не е така обаче за символите от кирилицата. Ако на съответната позиция е записана буква А на кирилица, charCodeAt, в съответствие с кодовата таблица на Unicode ще върне стойност 1040.

Извличане на част от символен низ – методи substr и substring.

Значение: Методите извличат последователност от символи от символния низ на обекта от тип String. Методът substr връща като символен низ зададен брой символи от зададена начална позиция. Методът substring връща като символен низ символите от зададена начална позиция до зададена крайна позиция.

Синтаксис:

substr (начален_индекс [, брой_символи])
substring(начален_индекс [, краен_индекс])

Описание: Параметърът *начален_индекс* задава началната позиция, от която се извличат символите. Ако *начален_индекс* има положителна стойност, тя трябва да е в интервала от 0 до дължината на низа -1. Ако *начален_индекс* има отрицателна стойност, то методът substr го използва като отместване от края на низа, а методът substring приема стойността му за 0

Параметърът *брой_символи* трябва да има положителна стойност. При отрицателна стойност методът substr връща празен низ. . Ако *брой_символи* липсва, се извличат всички символи до края на низа.

Параметърът *краен_индекс* задава крайна позиция, до която се извличат символите, като последния, *краен_индекс* се пропуска. Например методът substring(1,4) ще извлече символите с индекси 1, 2 и 3. Ако *краен_индекс* има стойност, по-малка от начален индекс, поведението на метода е различно за различните версии на JavaScript. За версия 1.3 и следващи

методът substring в тоя случай разменя значението на двата индекса*, т.е. винаги по-малкия индекс се използва като начален, а по-големия – като краен. Ако *краен_индекс* липсва, се извличат всички символи до края на низа.

Пример:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
var str="Hello World";
document.writeln("<br>Първите 3 символа са: "+str.substr(0,3));
document.writeln("<br>Символите с индекс от 1 до 3 са: "
+str.substring(1,4));
</script></BODY></HTML>
```

Пример 101 Извличане на част от низ с методите substr и substrig на обект String

Преобразуване на регистъра на буквите – методи toLowerCase и toUpperCase.

Значение: Методът toLowerCase преобразува главните букви в низа в малки. Методът toUpperCase прави обратното - преобразува малките букви в низа в главни.

Синтаксис:

toLowerCase( )
toUpperCase ( )

Описание: Методите са полезни най-вече в случаите, когато искаме да сравняваме символни низове без да правим разлика между малки и главни букви. Например ако променливите str1 и str2 съдържат символни низове и искаме да проверим дали те са еднакви без да се прави разлика между малки и главни букви, можем да използваме следната конструкция if-else:

```
if(str1.toLowerCase() == str2.toLowerCase())
{
    // код, който се изпълнява когато низовете са еднакви
} else {
    // код, който се изпълнява когато низовете не са еднакви
}
```

* В JavaScript 1.2 при тази ситуация се генерира грешка по време на изпълнение на скрипта (run-time error).

Пример 102 Използване на метода toLowerCase при сравнение на символни низове.

Тъй като преди сравнението на низовете главните букви в тях се преобразуват в малки букви с метода toLowerCase, на практика при сравнението не се прави разлика между малки и главни букви.

Разделяне на символен низ – метод split.

Значение: Разделя символния низ и връща масив с елементи, съдържащи получените поднизове.

Синтаксис:

split([разделител][, макс_брой_разделяния])

където:

разделител е символен низ, по който се разделя низа

макс_брой_разделяния (незадължителен) е цяло число, което задава максималния брой поднизове, които могат да се отделят.

Описание:

Методът split е особено полезен при анализ на съдържанието на символни низове. Той връща масив, който съдържа отделените от символния низ части с избрания разделител. Като разделител може да се зададе низ, който съдържа един или няколко символа. Параметърът *разделител* е зададен като незадължителен (обозначено със средни скоби), но ако липсва, никакво разделяне не се прави. Всеки път, когато методът открие тази последователност в зададения низ, той добавя елемент към масива и записва в него символите, отделени след предходното разделяне.

Пример:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
var str="Hello Java  Script";
var a = str.split(“ “);
document.writeln("<br>Полученият масив е: " +a);
</script></BODY></HTML>
```

Пример 103 Разделяне на низ с метода split.

В примера като разделител е зададен празен интервал. Зададеният низ съдържа три празни интервала – един между “Hello” и “Java” и два между “Java” и “Script”. Като резултат ще се получи масив със следното съдържание: a[0]=”Hello” , a[1]=”Java”, a[2]=”” и a[3]=”Script” , и методът writeln ще изведе низа "Hello,Java,,Script". Елементът a[2] съдържа празен

низ, тъй като между втория и третия празен интервал няма никакви символи.

HTML форматиране на символен низ – методи big, small, bold, italics, fixed, sub( ), sup( ), blink, fontcolor, fontsize, link, anchor.

Значение: Методите добавят HTML етикети (тагове) към символния низ на обект от тип String.

Синтаксис:

Повечето от методите нямат параметри:

big( ), small( ), bold( ), italics( ), fixed( ), sub( ), sup( ), blink( )

С параметри са следните методи:

fontcolor(цвят)
fontsize(размер_на_шрифта)
link(URL)
anchor(име_на_котвата)

Описание: Методите са полезни най-вече в случаите когато програмно генерираме HTML код. Методите добавят към символния низ следните етикети:

Метод	Добавя етикети преди и след низ	Резултат
big	<BIG> низ </BIG>	Увеличаване на шрифта
small	<SMALL> низ </SMALL>	Намаляване на шрифта
bold	<B> низ </B>	Удебеляване
italics	<I> низ </I>	Накланяне
fixed	<TT> низ </TT>	Шрифт с фиксиран размер на символите
sub	_{низ}	Долен индекс
sup	^{низ}	Горен индекс
blink	<BLINK> низ <BLINK>	Мигане
fontcolor	<FONT COLOR=”color”> низ </FONT>	Цвят на текст
fontsize	<FONT SIZE=” размер_на_шрифта ”> низ </FONT>	Размер на шрифта

link	<A HREF=”URL”> … >/A>	Хипервръзка
anchor	<A NAME=” име_на_котвата”> низ >/A>	Точка за преход в HTML документа (котва)

Всеки от методите връща символен низ с добавени етикети преди и след низа, които съответстват на предназначението на метода. Тези етикети имат значение ако символния низ се извежда в прозореца на браузъра, например с методите write или writeln на обект document. Тогава добавените етикети ще се обработят като HTML тагове. Ако обаче получения низ се изведе по друг начин, например в диалогова кутия, добавеното от методите ще се изведе като текст. Символния низ на обекта от тип String, над който се изпълнява метода, остава непроменен. Да разгледаме един пример:

```
var str="Hello, world";
var str1= str. italics(); var str2=str1. fontcolor("red");
document.write("<BR>" + str1); //Извежда наклонен текст
document.write("<BR>" + str2); //Извежда текста с червен цвят
alert(“str2=”+str2);
```

Пример 104 Форматиране на текст с методи italics и fontcolor

В примера към низа str "Hello, world" първо се добавят етикети за наклонен текст с метода italics и към получения низ str1 се добавят етикети <FORM COLOR=”red”> и </FORM> чрез метода fontcolor. Низовете str1 и str2 се извеждат с document.write. Низа str1 се извежда с наклонен шрифт, а низа str2 – с наклонен шрифт и с червен цвят. В диалоговата кутия alert низа str2 се извежда като обикновен неформатиран текст, в който се виждат добавените от двата метода етикети.

Същия резултат може да се получи и като просто се слепят към началото и края на зададения низ символни низове, които съдържат необходимите етикети за форматиране, така, че за програмиста на JavaScript не е обязателно необходимо на познава и използва тези методи.

21.5.4 Математически константи и функции. Обект Math

Обектът Math предоставя на програмиста на JavaScript набор от математически константи и функции. Обектът се създава от интерпретатора на JavaScript и не е необходимо да програмистът да създава инстанция на обекта за да ползва неговите атрибути и методи. Например, ако искаме да изчислим квадратен корен от стойността на променливата x и да го присвоим на променлива y е достатъчно да използваме инструкцията:

```
y = Math.sqrt(x);
```

Атрибути на обект Math

Обектът Math предоставя на програмиста набор от атрибути само за четене, които съдържат често използвани математически константи:

Константа	Описание
E	Константа E, основа на натуралните логаритми.
LN2	Натурален логаритъм от 2.
LN10	Натурален логаритъм от 10.
LOG2E	Логаритъм с основа 2 от E.
LOG10E	Логаритъм с основа 10 от E.
PI	Константа PI.
SQRT1_2	Квадратен корен от 1/2.
SQRT2	Квадратен корен от 2.

Методи на обект Math

Обектът Math предоставя на програмиста набор от математически функции- основните тригонометрически функции и някои други по-често използвани функции в математическите изчисления:

Метод	Описание
abs(x)	Връща абсолютната стойност на x.
acos(x)	Връща аркус косинус от x в радиани.
asin(x)	Връща аркус синус от x в радиани.
atan(x)	Връща аркус тангенс от x в радиани.
atan2(y, x)	Връща ъгъла по посока на часовниковата стрелка между оста x axis и точката (x,y).
ceil(x)	Връща най-малката цяла стойност, по-голяма или равна на x. (закръгление нагоре).
cos(x)	Връща косинус от x, като ь е в радиани.
exp(x)	Връща експонента от x (e^x)
floor(x)	Връща най-голямата цяла стойност, по-малка или равна на x. (закръгление надолу).
log(x)	Връща натурален логаритъм от x. (логаритъм с основа

	E).
max(a, b)	Връща по-голямата стойност от a и b.
min(a, b)	Връща по-малката стойност от a и b.
pow(x, y)	Връща X^y (функция за изчисляване на произволна степен)
random()	Връща псевдослучайно число със стойност от 0 до 1.
round(x)	Закръглява x нагоре или надолу до най-близкото цяло число. Например 4.3 се закръглява до 4, а 4.7 – до 5.
sin(x)	Връща синус от x, където x е в радиани.
sqrt(x)	Връща крадратен корен от x.
tan(x)	Връща тангенс от x, където x е в радиани.

Да разгледаме един пример:

Да се състави скрипт, който да изчислява и извежда стойността на функцията sin(x) за стойности на x от 0 до 90 градуса със стъпка 1 градус.

За изчисляване на стойностите на sin(x) можем да организираме цикъл, който да се изпълнява за стойност на x от 0 до 90 със стъпка 1. Трябва обаче да имаме в предвид, че аргументът на функцията sin(x), която ни предоставя обекта Math, трябва да бъде зададен в радиани, а не в градуси. От математиката е известно, че на 180 градуса съответстват PI радиана, и следователно коефициентът, който преобразува стойност, зададена в градуси, в стойност в радиани е PI/180.0 . Можем да изведем резултата в диалогова кутия ако всеки път добавяме в низ str един текстов ред, който съдържа стойността на x и стойността на sin(x) и завършва със символа за преминаване на нов ред “\n”. Така скриптът добива вида:

```
<HTML><BODY><SCRIPT language="JavaScript">
var str=””; //инициализиране на str с празен низ
for(var x=0;x<=90;x++) //цикъл, в който x се променя от 0 до 90 със стъка
1
{    // добавя ред със стойност на sin(x)
     str+=”sin(“+x+”)=\t” + Math.sin(x*Math.PI/180.0) +”\n”;

}
alert(str); //извежда списъка от стойностите в диалогова кутия
</SCRIPT></BODY></HTML>
```

Пример 105 Изчисляване на стойности на sin(x) посредством обект Math

21.5.5 Дата и време от компютърния часовник. Обект Date

Обектът Date предоставя на програмиста на JavaScript набор от методи за работа с дата и време. Посредством get методите си обекта Date, създаден с празен конструктор, предоставя информация за текущата дата и време, които се получават от системния часовник.

Конструктори:

new Date()

new Date(милисекунди)

new Date(низ_съдържащ_дата)

new Date(година, месец, ден [, час, минути, секунди, милисекунди])

където:

милисекунди	във втория конструктор това са милисекундите от 1970 00:00:00
низ_съдържащ_дата	символен низ с датата във формат име_на_месец, дата, година, [час:минути:секунди], March 11, 1985 09:25:00
година, месец, ден [, час, минути, секунди, милисекунди]	Целочислени стойности, представящи съответните елементи от датата и часа

Примери за конструктори:

```
today = new Date();
birthday = new Date("March 11, 1985 09:25:00");
birthday = new Date(85,2,11);
birthday = new Date(85,2,11,9,25,0);
```

Get методи на обект Date за локално време.

Get методите на обект Date връщат елементи от датата и времето като целочислени стойности. За да получите чрез тях информация за текущата дата и време от системния часовник на компютъра, трябва да създадете обект от тип Date чрез конструктор без параметри, напр. theDate=new Date( ). Важно е да се запомни, че създаденият по този начин обект от тип Date ще съдържа текущото време в момента на създаването си, но неговото съдържание няма да се актуализира когато системния часовник на компютъра промени стойността си. За да се получи периодично актуализиране, трябва самия скрипт периодично да създава нов обект от

тип Date чрез конструктор без параметри. Пример за подобна актуализация е даден след описанието на метод setTimeout на обект window.

getFullYear( )	Връща целочислена стойност, съответстваща на годината.
getYear( )	Връща целочислена стойност, съответстваща на годината. За година под 2000 се връща стойност=година-1900.
getMonth( )	Връща целочислена стойност от 0 до 11, съответстваща на текущия месец.
getDate( )	Връща целочислена стойност от 1 до 31, съответстваща на датата от текущия месец.
getDay( )	Връща целочислена стойност от 0 до 6, съответстваща на деня от седмицата 0 – неделя, 1 – понеделник и т.н.
getHours( )	Връща целочислена стойност от 0 до 23, съответстваща на часа.
getMinutes( )	Връща целочислена стойност от 0 до 59, съответстваща на минутите.
getSeconds( )	Връща целочислена стойност от 0 до 59, съответстваща на секундите
getMilliseconds ( )	Връща целочислена стойност от 0 до 999, съответстваща на милисекундите в текущата секунда.
getTime( )	Връща целочислена стойност – времето в милисекунди от 1 януари 1970г. 00.00.00

Пример:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
theDate = new Date(); //Създаване на обект от тип Date
document.writeln("Датата днес е: "+
        theDate.getDate()+"."+          //Дата
        (theDate.getMonth()+1)+"."+     //Месец
        theDate.getYear());                     //Година
</SCRIPT></BODY></HTML>
```

Пример 106 Получаване на информация за текущата дата чрез методи на обект Date

Set методи на обект Date за локално време.

Set методите на обект Date записват нови стойности на елементи от датата и времето. Важно е да се запомни, че записаните стойности принадлежат на обекта от тип Date, чийто метод се изпълнява. Дори този обект да е създаден чрез конструктор без параметри (т.е. с конструктор, който не задава дата и време), set методите не променят системната дата и време, а само датата и времето, които, са записани в обекта. На всеки get метод на обект Date съответства set метод за същия елемент от датата и времето. Интересно е да се отбележи, че липсва метод setDay за задаване на ден от седмицата, тъй като деня от седмицата се изчислява чрез стойностите на годината, месеца и датата.

setFullYear(*година*)	Задава целочислена стойност, съответстваща на годината. Ако *година* е стойност от 00 до 99 включително, то към *година* се добавя 1000 и получената стойност се записва. Например 99 ще се запише като 1999 година. Ако стойността е 2000 и по-голяма, се записва както е зададена.
setYear(*година*)	Задава целочислена стойност, съответстваща на годината. Препоръчва се вместо този метод да се използва setFullYear
setMonth(*месец*)	Задава месеца чрез целочислена стойност от 0 до 11. 0 – за януари ... 11 – за декември.
setDate(*дата*)	Задава датата като целочислена стойност от 1 до 31.
setHours(*час*)	Задава часа като целочислена стойност от 0 до 23.
setMinutes(*минути*)	Задава минутите като целочислена стойност от 0 до 59.
setSeconds(*секунди*)	Задава секундите като целочислена стойност от 0 до 59.
setMilliseconds (*милисекунди*)	Задава целочислена стойност от 0 до 999, съответстваща на милисекундите в текущата секунда.
setTime (*милисекунди*)	Задава целочислена стойност – времето в милисекунди от 1 януари 1970г. 00.00.00. Задаването на тази стойност променя всички елементи на датата и времето.

```
<HTML><BODY>
<SCRIPT language="JavaScript">
theDate = new Date("December 17, 1995 03:24:00")
    document.writeln("theDate is: "+
        theDate.getDate()+"."+          //Дата
        (theDate.getMonth()+1)+"."+     //Месец
        theDate.getFullYear ()+" "+     //Година
```

```
            theDate.getHours()+":"+          //Час
            theDate.getMinutes()+":"+        //Минути
            theDate.getSeconds());           //Секунди
// Променяне на годината, месеца и датата
            theDate.setDate(12);       //Променяне на дата
            theDate.setMonth();        // Променяне на месец
            theDate.setYear(1999);     // Променяне на година
// Извеждане на датата и времето след промените
            document.writeln("theDate is: "+
            theDate.getDate()+"."+           //Дата
            (theDate.getMonth()+1)+"."+      //Месец
            theDate.getYear()+" "+           //Година
            theDate.getHours()+":"+          //Час
            theDate.getMinutes()+":"+        //Минути
            theDate.getSeconds());           //Секунди
</SCRIPT></BODY></HTML>
```

Пример 107 Променяне на дата, месец и година посредством методи setDate, setMonth и setYear.

В примера се създава обект theDate от тип Date, като с конструктора Date("December 17, 1995 03:24:00") се задават стойности на всички елементи на датата и времето. Посредством съответните get методи на обекта theDate се извличат елементите на датата и времето на обекта и полученият символен низ се извежда в прозореца на браузъра с метода writeln на обект document. Посредством методите setDate, setMonth и setYear се записват нови стойности за датата, месеца и времето и се извеждат в прозореца на браузъра.

Методът setTime задава наведнъж стойностите на всички елементи на датата и времето, тъй като задава стойността на милисекундите от 1 януари 1970 година в 00:00:00 часа. Това например може да се използва за да се създаде обект от тип Date, който да има същите стойности на всички елементи от датата и времето:

```
theDate = new Date("July 1, 1999");          //Създава обект Date
sameDate = new Date();                       //Създава втори обект Date
sameDate.setTime(theDate.getTime());  //Записва в sameDate същите
                                        стойности на датата и времето
```

Пример 108 Използване на методите getTime и setTime за записване на стойностите на датата на един обект от тип Date в друг обект Date.

В примера се създава обект от тип Date с конструктор Date("July 1, 1999"), който задава стойности на датата, часа и времето. С втората инструкция се създава втори от тип Date с конструктор Date( ) без параметри. В третата инструкция се записва в обекта.

Методи за операции с универсалното време UTC.

Универсалното време е време, отчетено спрямо Гринуичкия меридиан. Обектът Date предоставя набор от get и set методи, които съответно връщат или записват стойности на елементите на датата и времето, но отчитани спрямо универсалното време. Методите са аналогични на разгледаните по-горе методи за локалното време, но в наименованието им е добавено UTC:

Get методи за универсално време

getUTCFullYear, getUTCMonth, getUTCDate, getUTCDay, getUTCHours, getUTCMinutes, getUTCSeconds, getUTCMilliseconds, getUTCTime

Set методи за универсално време

setUTCFullYear, setUTCMonth, setUTCDate, setUTCHours, setUTCMinutes, setUTCSeconds, setUTCMilliseconds, setUTCTime

Трябва да се има в предвид, че методите за универсално време използват освен системния часовник на компютъра също така и информацията за времевата зона, в която се намира компютърът. Така че, стойностите, които се получават зависят от това дали времевата зона е правилно зададена.

21.5.6 Обработка на грешки при изпълнение на скрипта. Обект Error

При изпълнение на кода на JavaScript могат да възникнат грешки (run-time errors) по различни причини, например поради неправилно зададени операнди на операция (например опит за делене на нула) или поради недопустима стойност на аргумент на функция (например отрицателен аргумент на функция логаритъм). Такъв тип анормални ситуации се наричат *изключения*. От четвърта версия в JavaScript са включени операторите try, catch и finally, които дават възможност на оператора да изпълни собствен код, с който да реагира на такива ситуации. Синтаксисът за използването им е следния:

```
try{
```

```
    //код, който се проверява за грешка
}
catch(e){
    //код за отработване на грешката
}
finally{
    //инструкции, които се изпълняват, независимо
    //дали е възникнала грешка
}
```

Синтаксис за прехващане на грешки с try-catch-finally

Операторът try задължително се използва или с catch, или с finally, или и с двата оператора едновременно, следователно могат да се използват структури try – catch, try – finally или try – catch – finally. Действието на операторите е следното:

Операторът try { … } включва в блока си инструкциите, за които се следи за възникване на грешки.

Незадължителният оператор catch(e){ … } включва в блока си инструкциите, чрез които се отработва генерираната грешка, например, извежда се съобщение с информация за грешката. Променливата ***e*** съдържа обект Error, който е генериран за грешката. Атрибутите на обект Error съдържат информация за вида на грешката.

Незадължителният оператор finally { … } включва в блока си инструкции, които ще се изпълнят независимо от това дали е възникнала грешка или не.

Да разгледаме един пример:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
  try {
  var x = y; // Предизвиква се грешка (неинициализирана променлива  y
  alert("Continue");
} catch(e){  // Създаване на локална променлива "e".
  document.write("<br>Error "+e);     // За IE се извежда " Error [object Error]".
  }
</SCRIPT></BODY></HTML>
```

Пример 109 Прехващане на грешка посредством структура try - catch

В примера в блока try е включена инструкция `var x = y` , която използва стойност на неинициализираната променлива y. При това се генерира грешка и се извършва преход към оператора catch. Поради прехода инструкцията `alert("Continue")` няма да се изпълни, а ще се изпълни инструкцията,

включена в блока на оператора catch - document.write("
Error"+e) . Тъй като обектната променлива *e* се слепва със символен низ, се извиква неявно методът toString() на обекта Error, който връща символен низ. Резултатът ще бъде различен за различните браузъри, защото методът toString() на обект Error е реализиран по различен начин. За Internet Explorer ще се изведе низа "Error [object Error]", докато за Netscape Navigator ще се изведе по-информативното съобщение "Error ReferenceError: y is not defined".

Обект Error

Обект Error предоставя на програмиста информация за грешките, генерирани по време на изпълнение на скрипта (run time errors). Той има конструктор, но при възникване на грешка се генерира автоматично и се присвоява на променливата на оператор catch ако грешката се прехваща със структура try – catch. Обект Error е един от обектите, чиято реализация се различава за различните браузъри, напр. за Internet Explorer (IE) и Netscape Navigator (NE). И двата браузъра поддържат конструктор без параметри:

var newErrorObj = new Error()

JavaScript, от версия 1.5 поддържа различни конструктори за обект Error за различните видове грешки:

EvalError	Грешка, генерирана от функция eval().
RangeError	Генерира се, ако числова стойност излиза от допустимия диапазон на числата.
ReferenceError	Невалиден адрес (референция) на обект.
SyntaxError	Синтактична грешка в код, изпълняван от функция eval() Другите синтактични грешки не се прихващат от try/ catch/finally, и информация за тех се извежда от браузъра.
TypeError	Невалиден тип променлива.
URIError	Грешка при кодиране или декодиране на интернет адрес

Когато се генерира грешка по време на изпълнение се създава обект от клас Error. Атибутите, на обект Error, които се поддържат от всички браузъри са:

name:	Име на грешката, или, по-точно, името на функцията-конструктор, чрез която обекта error е създаден..
message:	Описание на грешката (зависи от вида на браузъра).

В IE обекта Error притежава и атрибути number и description, на които са присвоени стойности, които дават информация съответно за номера и описанието на възникналата грешка, но те не се поддържат от другите браузъри

Генериране на изключения. Оператор throw.

JavaScript дава възможност на програмиста да генерира собствени изключения, които могат да бъдат прехванати с оператор try. Оператор throw има следния синтаксис:

throw exception

където exception е константа или променлива, която може да съдържа символен низ, число, логическа стойност или обект. Стойността на exception се присвоява на генерирания обект Error. Да разгледаме един пример:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
 try {
 throw “My Exception”; // Генерира изключение
 alert("Continue");
} catch(e){  // Създаване на локална променлива "e".
 document.write("<br>Error "+e);   // извежда се " My Exception".
 }
</SCRIPT></BODY></HTML>
```

Пример 110 Генериране на изключение посредством оператор throw

В примера след генериране на изключение с оператор throw “My Exception” се извършва преход към оператора catch. Поради прехода инструкцията alert("Continue") няма да се изпълни, а ще се изпълни инструкцията, включена в блока на оператора catch - document.write("
Error"+e), която ще изведе низа “Error My Exception". Важно е да се запомни, че променливата „e” в клауза catch(e) получава типа и стойността, зададена на оператора throw. В примера променливата „e” получава символен низ.

21.6 Вградени обекти на JavaScript, свързани с потребителския компютър и WEB браузъра.

По правило JavaScript се включва в HTML документите и се изпълнява от WEB браузъра. В тоя случай JavaScript предоставя на програмиста обекти,

който притежават атрибути и методи за операции с потребителския компютър, WEB браъзъра и съдържанието на HTML документите. Прието е да се казва, че тези обекти се предоставят при изпълнение на JavaScript от страна на клиента. Основната част от тези обекти се създават при обработка на HTML документа. Тук ще разгледаме обектите screen и navigator, които предоставят атрибути и методи, свързани с потребителския компютър и WEB браузъра.

21.6.1 Обект screen.

Обектът screen е обект от най-високо ниво, който се създава от интерпретатора на JavaScript на WEB браузъра. Той предоставя атрибути, чрез които може да се получи информация за режима на монитора на потребителския компютър. Тази информация е полезна за WEB дизайнера, тъй като в много случаи се проектират HTML страници, които не се изобразяват изцяло или в приемлив вид при определени режили на дисплея, например под зададена разрешаваща способност в брой точки. Ако е желателно HTML документът да се изобразява за всички възможни режими на монитора, то за режимите с по-ниска разрешаваща способност може автоматично да се зарежда по-опростен вариант.

Основните атрибути на обект screen, които се поддържат както от Internet explorer, така и от Netscape Navigator са:

Атрибут	Описание
colorDepth	Брой битове за представяне на цвета. Най-високата разрешаваща способност по цвят за съвременните монитори е 32 бита (режим TrueColor).
height	Вертикален размер на екрана на монитора в брой точки.
width	Хоризонтален размер на екрана на монитора в брой точки.

Най-често се използва режим на монитора 1024/800, което означава размери в брой точки съответно width=1024 и height=768.

Пример:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
document.writeln("<BR>screen.width="+screen.width);
document.writeln("<BR>screen.height="+screen.height);
document.writeln("<BR>screen.colorDepth="+screen.colorDepth);
</SCRIPT>
```

```
</BODY></HTML>
```

Пример 111 Получаване на параметри на монитора посредством обект screen

21.6.2 Обект navigator.

Обектът navigator е обект от най-високо ниво, който се създава от интерпретатора на JavaScript на WEB браузъра. Той предоставя на програмиста на JavaScript информация за вида на WEB браузъра и за някои от неговите опции (т.е. избраните от потребителя режими и параметри). Не всички атрибути и методи, които се дават в документацията на JavaScript реално се поддържат от WEB браузърите. Основните атрибути на обект navigator, които се поддържат както от Internet explorer (IE), така и от Netscape Navigator (NE) са:

Атрибут	Описание
appCodeName	Връща символен низ, съдържащ кодовото име на браузъра. За IE и NE връща стойност “Mozilla”.
appName	Връща символен низ, съдържащ името на браузъра. За IE и NE връща стойности съответно “Microsoft Internet Explorer” и “Netscape”.
appVersion	Връща символен низ, съдържащ версията на браузъра. Например за IE 6.0 връща низа “4.0 (compatible; MSIE 6.0; Windows NT 5.0)”
language	Връща символен низ, съдържащ информация за езика, на който са менюто и помощната информация на браузъра. По подразбиране стойността е “en-US”.
mimeTypes	Връща масив от обекти MIME type. Поддържа се само от Netscape Navigator (NE).
platform	Връща символен низ, съдържащ информация за операционната система на потребителския компютър. За ОС Windows 2000/XP връща низ “Win32”.
plugins	Връща масив от обекти plug-ins. Поддържа се само от Netscape Navigator (NE).

userAgent	Връща символен низ с информация за вида и версията на браузъра. Ако от HTML документа се изпращат данни към интернет приложение, този символен низ се добавя към данните като т.н. “заглавна част” (user-agent header). Напр. за IE 6.0 връща низа “Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)”

Обектът navigator предоставя няколко метода, от които на практика е достъпен за използване само един – методът javaEnabled

метод javaEnabled

Методът връща логическа стойност true, ако е разрешено изпълнението на Java аплети, и стойност false – ако е забранено. Java аплетите са програми, които могат да се включват в HTML документи. Те се изтеглят и се записват на потребителския компютър, при зареждането в браузъра на HTML документа, който ги съдържа. Java аплетите се считат за безопасни за потребителския компютър, тъй като те нямат достъп до файловата система на потребителския компютър без специално разрешение. Потребителят може да забрани тяхното изпълнение от менюто на WEB браузъра – за Internet Explorer посредством функцията Tools – Internet Options – Security Settings – опция “Scripting of Java Applets”. Методът javaEnabled дава възможност да се провери програмно дали е разрешено изпълнението на аплети и ако не е да се зареди документ, който не съдържа аплети.

Синтаксис:

navigator.javaEnabled()

В много случаи се налага да се провери какъв е вида на браузъра, тъй като въпреки стандартизацията на обектния модел на HTML документа, все още има различия между браузърите. Ако искаме да определим вида на браузъра, можем например да проверим, дали в низа на атрибута userAgent се съдържа “MSIE”. Ако “MSIE” се съдържа, браузърът е Internet Explorer (IE), в противен случай може да се приеме, че е Netscape Navigator (NE). По-долу е даден скрипт, който извършва тази проверка и извежда информация за вида на браузъра.

```
<HTML><BODY>
<SCRIPT language="JavaScript">
var s=navigator.userAgent;
if(s.indexOf("MSIE")!=-1){
    document.writeln("браузърът е Internet Explorer");
```

```
}else{
    document.writeln("браузърът е Netscape Navigator");
}
</SCRIPT></BODY></HTML>
```

Пример 112 Проверка за вида на браузъра посредством атрибут userAgent на обект navigator

В примера символният низ, който се връща от атрибута userAgent на обект navigator се присвоява на променливата s. Тъй като s е символна променлива*, към нея могат да се прилагат методите на обект String. С метода indexOf на обект String се проверява дали в символния низ, получен от атрибута userAgent се съдържа “MSIE”. Ако “MSIE”се съдържа в s, тогава това методът indexOf връща неговия начален индекс (стойност, различна от -1). Това означава, че браузърът е Internet Explorer и в прозореца на браузъра се извежда съобщението “браузърът е Internet Explorer”. Ако “MSIE”се съдържа в s, тогава това методът indexOf връща стойност -1 и се извежда съобщение "браузърът е Netscape Navigator".

21.7Обектен модел на HTML документа.

D 8 Обектния модел на HTML документа (DOM)* — е платформено-независим програмен интерфейс, даващ възможност за управление на съдържанието, структурата и оформлението на HTML документите.

Моделът DOM не налага ограничения на структурата на документа. Всеки HTML документ със зададена структура с помощта на DOM може да бъде представен в вид на дърво от възли, като всеки възел содержа элемент, атрибут, текстов, графичен или някакъв друг обект. Възлите са свързани помежду си с отношения от типа родител-наследник.

Първоначално различните браузери са имали собствен документен модел DOM, несъвместим с останалите. За да се обеспечи взаимна и обратна совместимост, специалистите от международния консорциум W3C класифицирали модела DOM по нива, като за всяко ниво е създадена съответна спецификация. Всички тези спецификации са обединени в обща група, носеща наименованието W3C DOM.

В съответствие с целите на курса по WEB дизайн ние ще се запознаем с ейрархията от обектите в DOM и ще разгледаме детайлно основните обекти. Познаването на обектния модел на HTML документа, в това число

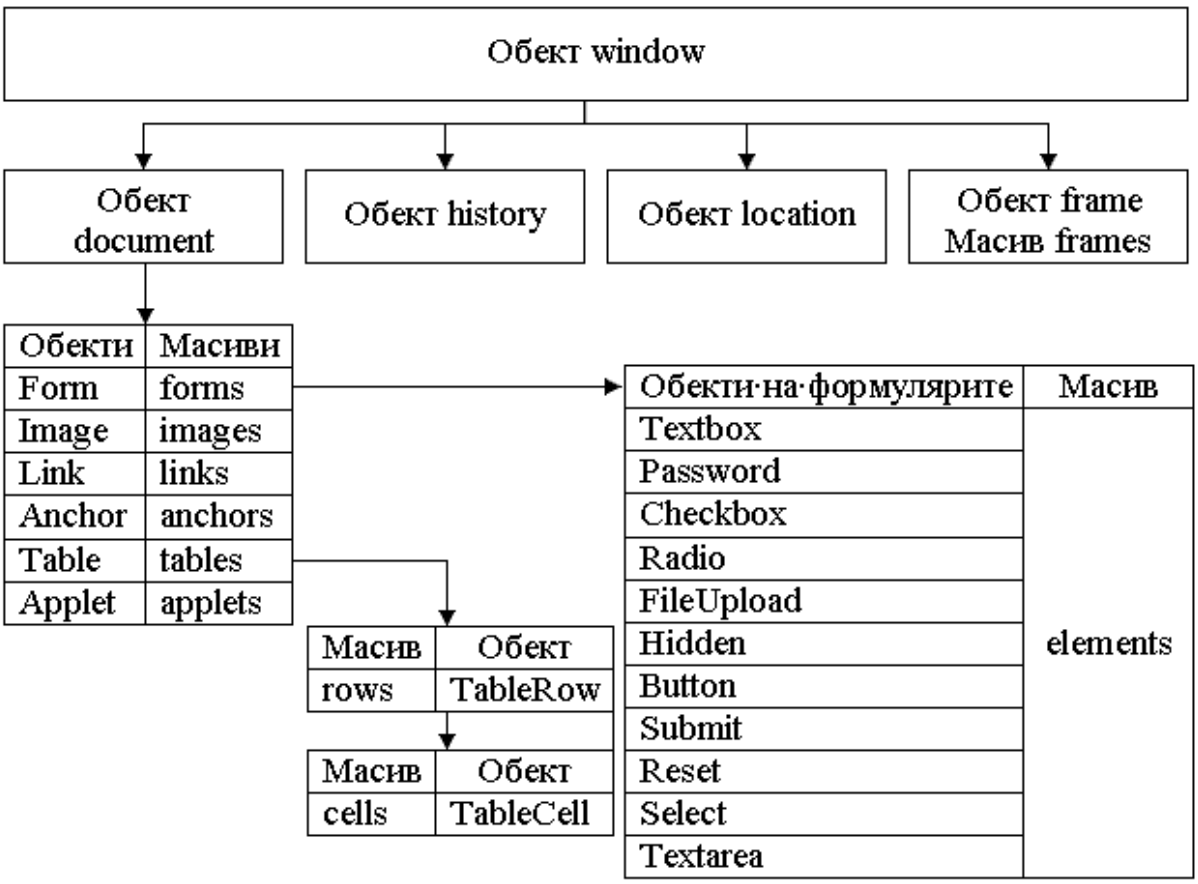
* За такива променливи се създава временен обект от тип String

* Съкращение от Document Object Model

връзките между обектите, атрибутите, методите и манипулаторите на събития, който ни предоставя всеки обект, са особено важни за WEB дизайнера, тъй като те му предоставят възможност да взаимодейства с потребителя и програмно да променя съдържанието на документа.

21.7.1 Иерархия на обектите в обектния модел на HTML документа.

Както беше посочено по-горе, обектите в DOM са свързани с отношения родител-наследник. Това означава, че обектите-наследници се създават чрез обектите-родители, и то само ако съответния обект-родител вече е създаден. Важно е да се запомни, че WEB браузърът създава обектите при обработката на HTML документа, който се зарежда в браузъра. Ако например HTML документа съдържа хипервръзки, то за всяка хипервръзка ще бъде създаден съответстващ обект link. Ако хипервързки в документа няма, такива обекти изобщо няма да бъдат създадени. При създаването на обектите за тях се създават обектни променливи, и, освен това се създават масиви от обектни променливи за основните типове обекти. Например ако в документа има три хипервръзки, ще бъде създаден масив links с три елемента, които ще съдържат обектни променливи за всяка от хипервръзките.



Фиг. 17 Йерархия на обектите в DOM

Както се вижда от схемата, на най-високо ниво в йерархията на обектите от DOM стои обект window, който представя прозореца на браузъра. На второ ниво са обектите document, history и location. Достъпът до тези обекти се извършва чрез обект window. Обектът document, който представя HTML документа, зареден и изведен в прозореца, предоставя достъп до обектите от тип Form, Image, Link, Anchor, Table, Applet, представящи съдържанието на документа. За всеки тип обекти се създават масиви forms, images, links, anchors, tables, applets, съдържащи обектните променливи на всички създадени обекти от дадения тип. Чрез обекта Form се осъществява достъп до обектите Textbox, Password, … Textarea, които се създават от елементите за въвеждане и селектиране на данни и бутоните на формулярите. Обектът Table предоставя достъп до масив rows, съдържащ обектните променливи от тип TableRow, които представят редовете на таблицата. От своя страна обектите TableRow предоставят достъп до масив cells, съдържащ обектните променливи от тип TableCell, които представят клетките в реда. По-долу последователно ще бъдат разгледани обектите от DOM.

21.7.2 Манипулатори на събития.

Механизмът за обработка на събития е основното средство за комуникация между потребителя и програмите на JavaScript. Действията на потребителя и операциите, които извършва WEB браузъра генерират събития, в резултат на което се изпълняват функции за обработка на тези събития. В действителност механизмът за обработка на събития включва източник на съобщение за събитие, механизъм за предаване на съобщенията и получател на съобщението. Този механизъм обаче е скрит за програмиста на JavaScript. От гледна точка на програмиста обектите в JavaScript, които взаимодействат с WEB браузъра, реагират на събития. Обектите предоставят възможност да се изпълни дефинирана от програмиста функция или просто последователност от инструкции. Прието е тези функции да се наричат *събитийни процедури (event procedures)*. Обектът, който реагира на дадено събитие притежава специален тип атрибут *манипулатор на събитието (event handler)*, на който трябва да се присвои функцията, която трябва да се изпълни при възникване на събитието.

Програмистът има две възможности да инициализира манипулатор на събитие – чрез атрибут на HTML елемент и чрез непосредствено присвояване на стойност на манипулатора с инструкция на JavaScript.

Инициализирането чрез атрибут на HTML елемент има следния синтаксис:

<етикет манипулатор=”код_на_JavaScript” >

Да разгледаме един пример

```
<HTML>
<BODY onKeyPress="alert('Key Press')">
<center>Test event handler</center>
</BODY>
</HTML>
```

Пример 113 Инициализиране на манипулатор чрез атрибут на HTML елемент

В примера в етикета <BODY> е включен манипулатор на събитието KeyPress (натискане на бутон от клавиатурата), на който се присвоява символен низ, съдържащ една инструкция на JavaScript – обръщение към метода alert, който извежда диалогова кутия. При натискане на бутон от клавиатурата в прозореца на браузъра се генерира събитие KeyPress и се изпълнява кода, присвоен на манипулатора onKeyPress на обект document.

Като резултат ще се изведе диалогова кутия с текст 'Key Press'. Обърнете внимание на това, че символния низ – параметър на alert, е ограничен с апострофи (‘), въпреки че по правило символните низове се ограничават с двойни кавички (“). В случая това е необходимо, тъй като низа, който се присвоява на onKeyPress е ограничен с двойни кавички и използването им вътре в низа като "alert(“Key Press”)" ще бъде синтактична грешка.

Инициализирането чрез непосредствено присвояване на стойност на манипулатора с инструкция на JavaScript има следния синтаксис:

обект.манипулатор = име_на_функция

където *обект* е обект от DOM, който притежава съответния *манипулатор*, а

име_на_функция е име на дефинирана от програмиста функция, която трябва да се изпълни при генериране на събитието за обекта.

Важно е да се запомни, че в кода на JavaScript всички манипулатори на събития се изписват изцяло с малки букви. В кода на HTML документ, когато манипулаторът се инициализира чрез атрибут на HTML елемент, не се прави разлика между малки и главни букви.

Да разгледам един пример

```
<HTML>
<BODY onKeyPress="alert('Key Press')">
<SCRIPT language="JavaScript">
 function myKeyPress(){ //декларация на функция за събитието KeyPress
    alert("Key Press");
 }
 document.onkeypress= myKeyPress; //инициализиране на манипулатора
</SCRIPT>
</BODY></HTML>
```

Пример 114 Инициализиране на манипулатор чрез присвояване на стойност с инструкция на JavaScript.

Кода на HTML документа съдържа блок <SCRIPT>, в който е декларирана функцията myKeyPress. Извън декларацията на функцията (което означава, че ще се изпълни веднага при зареждането на HTML документа в браузъра) е включена инструкцията document.onkeypress= myKeyPress; , чрез която на манипулатора onkeypress на обект document се присвоява финкцията myKeyPress. Обърнете внимание на това, че се присвоява само името на функцията / myKeyPress а не myKeyPress() /. Чрез името myKeyPress се предава на манипулатора адреса на функцията, докато

myKeyPress() би било обръщение към функцията, при което тя ще се изпълни и ще върне стойност.

По-долу е даден списък на манипулаторите на събития и обектите на JavaScript, които ги притежават.

Събитие	Манипулатор, Обекти	Описание
Abort	onAbort Обект Image	Изпълнява код на JavaScript когато потребителят прекъсва зареждането на изображение.
Blur	onBlur Обекти от формуляр, обект window, обект frame	Изпълнява код на JavaScript когато елемент на формуляр загуби фокус или когато прозорец или фрейм загуби фокус.
Change	onChange Обекти Select, Text, и Textarea	Изпълнява код на JavaScript когато обект Select, Text, или Textarea загуби фокус при условие че съдържанието му е било променено
Click	onClick Обекти document, link, всички обекти на формулярите без обект hidden.	Изпълнява код на JavaScript когато операторът щракне с мишката върху обект или прозорец.
DblClick	onDblClick Обекти document, link, всички обекти на формулярите без обект hidden.	Изпълнява код на JavaScript когато операторът щракне двукратно с мишката върху обект или прозорец.
DragDrop	onDragDrop Обект window	Изпълнява код на JavaScript когато операторът провлачи и пусне обект върху прозореца на браузъра.
Error	onError Обекти Image, window	Изпълнява код на JavaScript когато при зареждане на документ или изображение възникне грешка.
Focus	onFocus	Изпълнява код на JavaScript когато прозорец, фрейм или фреймсет

Събитие	Манипулатор, Обекти	Описание
	Обекти window, frame, frameset, обекти от формуляр без hidden.	получи фокус или когато елемент от формуляр получи фокус.
KeyDown	onKeyDown Обекти document, Image, Link, Textarea	Изпълнява код на JavaScript когато операторът натисне бутон.
KeyPress	onKeyPress Обекти document, Image, Link, Textarea	Изпълнява код на JavaScript когато натисне или задържи натиснат бутон.
KeyUp	onKeyUp Обекти document, Image, Link, Textarea	Изпълнява код на JavaScript когато операторът отпусне бутон.
Load	onLoad Обекти Image, Layer, window.	Изпълнява код на JavaScript когато браузърът завърши зареждането на документ в прозореца или във всички фреймове в етикет FRAMESET.
MouseDown	onMouseDown Обекти Button, document, Link	Изпълнява код на JavaScript когато операторът натисне бутон на мишката.
MouseMove	onMouseMove По подразбиране не се генерира от никой обект	Изпълнява код на JavaScript когато операторът премести курсора на мишката.
MouseOut	onMouseOut Обекти: Layer, Link	Изпълнява код на JavaScript всеки път когато курсорът на мишката напуска област от HTML карта или област на хипервръзка.
MouseOver	onMouseOver Обекти: Layer, Link	Изпълнява код на JavaScript еднократно когато курсорът на мишката навлиза отвън в област от HTML карта или област на хипервръзка.that object or area.
MouseUp	onMouseUp Обекти Button,	Изпълнява код на JavaScript когато операторът отпусне бутон на

Събитие	Манипулатор, Обекти	Описание
	document, Link	мишката.
Move	onMove Обект window	Изпълнява код на JavaScript когато операторът или скриптът премести прозорец или фрейм.
Reset	onReset Обект Form	Изпълнява код на JavaScript когато операторът изчисти съдържанието на формуляр (при щракане върху бутон Reset).
Resize	onResize Обект window	Изпълнява код на JavaScript когато операторът или скриптът промени размера на прозорец или фрейм.
Select	onSelect Обекти Text. Textarea	Изпълнява код на JavaScript когато операторът селектира текст в текстова кутия или многоредова текстова кутия.
Submit	onSubmit Обект Form	Изпълнява код на JavaScript когато операторът изпрати съдържанието на формуляр (при щракане върху бутон Submit).
Unload	onUnload Обект window	Изпълнява код на JavaScript когато операторът затвори документа (напр. при зареждане на нов документ в браузъра).

21.7.3 Атрибути на събитията. Обект event.

Обектът event е включен в Java Script при появяването на четвърта версия на Internet Explorer и Netscape Navigator. Той предоставя на програмиста информация за настъпилото в браузъра събитие. Може да се каже, че обектът event е "моментна снимка" на атрибутите на събитие след като то е настъпило.

Като пример да разгледаме събитието *MouseOver*. Това събитие настъпва когато курсорът на мишката премине над зададения елемент, в случая над хипервръзка, включена в документа с тага <A>.

```
<a  href="somepage.html" onMouseOver="someFunction();">
```

В манипулатора на събитието - функцията *someFunction()* можем да изпозуваме обекта event, създаден в момента на настъпване на събитието. Например, ако искаме да получим координатите на мишката при преминаването и над хипервръзката, трябва да предадем като параметри на функцията *someFunction()* съответните атрибути на обекта event - *screenX* и *screenY* :

```
<a href="somepage.html"
onMouseOver="someFunction(event.screenX,event.screenY);">
```

Ако искаме да получим във функцията достъп до целия обект event трябва да го предадем като параметър на функцията:

```
 <a  href="somepage.html"
onMouseOver=" eventOver(event);">
```

при което функцията получава копие от обекта:

```
function eventOver(evt){
window.alert("x="+evt.screenX+"  y="+evt.screenY);
}
```

Както се вижда от кода на функцията в горния пример *eventOver(evt)*, посредством този параметър в нея се получава достъп до атрибутите на обекта event. По-долу е даден пълният код на примера:

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
function eventOver(evt) {
window.alert("x="+evt.screenX+"  y="+evt.screenY);
}
</SCRIPT></HEAD>
<BODY>
<a href="somepage.html"
onMouseOver="eventOver(event);">
Move Mouse Over</a>
</BODY></HTML>
```

Пример 115 Предаване на обект event като параметър на функция

За практическото използуване на манипулатори на събития е необходимо да се познават специфичните събития на браузърите и обектите, с които са свързани.

1. Не всяко събитие променя всички атрибути на обекта event. Променят се само атрибутите, които са свързани с възникналото събитие. Например събитието *MouseOver* не променя атрибутите, които са свързани с натискането на бутон на мишката, тъй като при това събитие бутоните на мишката не се активират.
2. Microsoft и FireFox поддържат различен набор от атрибути на обекта event, което на практика означава, че програмистът трябва да се съобразява с вида на браузъра, когато създава скрипт, който използува обекта event.

По-долу са дадени таблица с атрибутите на обект event, които се поддържат и от Internet Explorer и от FireFox, и две таблици, в които са посочени съответно атрибутите, които се поддържат само от единия от двата браузъра. Поддържането на атрибутите е тествано с IE 8 и FireFox 3.6.8.

Таблица 1. - Атрибути на обекта **event**, общи за Internet Explorer и FireFox (Netscape Navigator).

Атрибут	Описание
type	Символен низ, който съдържа името на събитието, например "Click" или "MouseOver".
screenX, screenY	Целочислени стойности, които отразяват съответно хоризонталната и вертикална координата на курсора на мишката спрямо екрана (desktop), при възникване на събитието.
clientX, clientY	Относителна хоризонтална (или съответно вертикална) координата на курсора спрямо клиентската област на браузъра (изключва се областа, заета от скрол лентите и други допълнителни елементи на прозореца).
altKey, ctrlKey, shiftKey	Стойността "true" на всеки от тези атрибути указва, че съответният бутон (**Alt**, **Ctrl** или **Shift**) е бил натиснат при настъпване на събитието, а стойност "false" - обратно (че не е бил натиснат).
button	Целочислена стойност, в която трите най-младши бита съответствуват на състоянието на бутоните на мишката при

	настъпването на събитието.Бит 0 съответствува на левия бутон на мишката, бит 1 - на десния и бит 2 - на средния бутон. Например стойност 3 ще означава, че са били натиснати левия и десния бутон на мишката.

Таблица 2. - Атрибути на обекта **event** за Internet Explorer 4.0

Атрибут	Описание
toElement	Съдържа обектна променлива на обекта, за който е генерирано събитието *MouseOver* или *MouseOut*.
srcElement	Указател към обекта, който генерира събитието
srcFilter	Указател към обекта filter, който генерира събитието *FilterChange*
keyCode	Целочислена стойност - Unicode код на бутон от клавиатурата, който генерира събитието.
x, y	Целочислени стойности, които отразяват съответно относителната хоризонтална и вертикална координата на курсора на мишката спрямо родителския обект (контейнера) при позициониране със средствата на CSS.
offsetX, offsetY	Целочислени стойности, които отразяват съответно относителната хоризонтална и вертикална координата на курсора на мишката вътре в обекта, който генерира събитието.
reason	За свързан източник на данни - целочислена стойност, показваща състоянието му по време на събитието. Възможни стойности: 0=Successful, 1=Transfer aborted, 2=Error
recordset	Съответствува на обект Recordset за свързан доставчик на данни (data provider).
repeat	Логическа стойност, която показва дали събитието се повтаря (true), например при задържане на бутон от клавиатурата.
returnValue	Логическа стойност, свързана с предаването на управлението след обработването на събитието. При стойност "true" обектът-приемник се активира след обработката му. Този атрибут е с по-висок приоритет от

	логическата стойност, върната от скрипта, който обработва събитието..
cancelBubble	Логическа стойност, която определя дали събитието повишава приоритета си в йерархията от събития на обекта-приемник. При стойност "true" приоритетът не се променя.
bookmarks	Масив от маркери, всеки от които идентифицира отделен запис в обект **recordset**, асоцииран със записите, активирани от текущото събитие.

.

Таблица 3. - Атрибути на обекта **event** за Netscape Navigator.4.0 и Firefox

Атрибут	Описание
target	Символен низ, който съдържа името на обекта-приемник на събитието - това може да бъде например HTML елемент.
layerX или x ; layerY или y	Относителна хоризонтална (или съответно вертикална) координата на курсора в слоя (layer), в който възниква събитието; В частност за събитието *Resize* атрибутът съдържа дължината (или съответно височината) на прозореца след преоразмеряването му.
pageX , pageY	Хоризонтална (или съответно вертикална) координата на курсора в прозореца, в който възниква събитието;
which	Целочислена стойност, която, в зависимост от вида на събитието указва номера на натиснатия бутон на мишката (1 за ляв и 3 за десен) или съдържа ASCII кода на натиснатия бутон от клавиатурата.
modifiers	Символна стойност, която съдържа информация за това, кой функционален модифициращ бутон е натиснат по време на възникването на събитие, свързано с мишката или клавиатурата. Възможни стойности: "ALT_MASK", "CONTROL_MASK", "SHIFT_MASK", "META_MASK"
data	Масив от символни низове, който съдържа адресите (URL) на обектите, добавени чрез събитието DragDrop в подписан скрипт, който притежава *UniversalBrowserRead privilege* - привилегия за четене.

Обект event е обектът, който е най-малко стандартизиран – основната част от атрибутите му е различна за различните браузъри. От приведените по-горе данни за Internet Explorer и Netscape Navigator се вижда, че общи са само атрибутите type, screenX и screenY.

21.7.4 Операции с прозореца на браузъра – обект window.

Обект window е обектът от най-високо ниво в обектния модел на документа (DOM). Той предоставя атрибути и методи, свързани с прозореца на браузъра. В графичния интерфейс на съвременните операционни системи основната комуникация на операционната система с приложението (в случая с WEB браузъра), както и на приложението с потребителя, се извършва посредством прозорци. В съответствие с това обект window предоставя на програмиста не само атрибути и методи, свързани с параметри на прозореца, но също така методи за извеждане и въвеждане на данни, за реализация на таймери и манипулатори на събития. По-долу ще бъдат разгледани манипулаторите на събития, атрибутите и методите на обект window.

Манипулатори на събития

Обект window притежава следните манипулатори на събития, на които можем да присвоим събитийни процедури.

Манипулатор	Вид събитие
onBlur	Загуба на фокус
onDragDrop	Операция “провлачване и пускане” в прозореца
onError	Възникнала грешка при изпълнение на скрипт
onFocus	Получаване на фокус
onLoad	Приключване на зареждането на документ
onMove	Преместване на прозореца
onResize	Преоразмеряване на прозореца
onUnload	Край на работа с документ

Атрибути на обект window

Обект window притежава голям брой атрибути, които можем да класифицираме според тяхното значение.

Атрибути, свързани с размера и позицията на прозореца. Тези атрибути не се поддържат от Internet Explorer.

pageXOffset	Задава и връща текущата позиция по x на изобразявания документ. Стойността на x се променя при хоризонтално скролиране на документа.
pageYOffset	Задава и връща текущата позиция по y на изобразявания документ. Стойността на y се променя при вертикално скролиране на документа.
screenX	Задава текущата позиция по x на горния ляв ъгъл на прозореца.
screenY	Задава текущата позиция по y на горния ляв ъгъл на прозореца.

Атрибути – обектни променливи от тип window (само за четене)

top	Съдържа обектна променлива за обект window за най-външния прозорец – прозореца на браузъра.
parent	Съдържа обектна променлива за обект window за родителския прозорец.
window, self	Съдържат обектна променлива за обект window за текущия прозорец.
opener	Ако прозорецът е отворен с метод open, съдържа обектна променлива за прозореца, чрез който е отворен.

Атрибути, съдържащи параметри на прозореца

name	(символен низ) Съдържа името на прозореца
closed	Връща логическа стойност, която показва дали прозореца е бил затворен (true) или не (false).
defaultStatus	(символен низ) Задава и връща съобщението, което се извежда по подразбиране в статус-лентата на прозореца, когато нищо друго не се извежда. Може да се променя по всяко време.
status	(символен низ) Задава и връща съобщението, което се извежда в статус-лентата на прозореца. Може да се променя

	по всяко време.

Атрибути, съдържащи масиви и параметри на масиви (само за четене)

length	Връща целочислена стойност – брой на фреймовете в прозореца
frames	Връща масив с обектни променливи за фреймовете в прозореца.

Атрибути – обектни променливи за обекти от DOM (само за четене)

document	Връща обектна променлива за обект document, който представя документа в прозореца
history	Връща обектна променлива за обект history, съдържащ информация за зареждането преди това документи.
location	Връща обектна променлива за обект location, съдържащ информация за

Методи на обект window

Методи за извеждане на диалогови кутии

Диалоговите кутии дават възможност за извеждане на съобщения (метод alert) и за получаване на отговор от оператора (методи confirm и prompt). Когато се извикват тези методи за текущия прозорец, се подразбира обект window и обектната променлива най-често се пропуска, например вместо window.alert(“Hello) е достатъчно да се запише само alert(“Hello). Изпълнението на скрипта спира докато операторът не затвори диалоговата кутия (такива диалогови кутии се наричат модални).

alert(string)	Извежда диалогова кутия с текста, съдържащ се в string и само един бутон OK , чрез който се затваря диалоговата кутия. Ако текстът съдържа символи за преминаване на нов ред “\n” и табулация “\t”, те се отработват. Това означава, че текста в диалоговата кутия може да бъде на няколко реда и да бъде позициониран с табулации.
confirm(string)	Извежда диалогова кутия с текста, съдържащ се в string и бутони с текст “OK” и “Cancel”. Ако операторът затвори

	диалоговата кутия с бутона “OK”, методът връща стойност логическа истина (true), а ако се затвори диалоговата кутия с бутона“Cancel” - връща стойност логическа неистина (false).
prompt(string 1[, string2])	Извежда диалогова кутия с пояснителен текст string1 и незадължителен втори параметър string2, който, ако е включен, се извежда като начално съдържание на текстова кутия, в противен случай се извежда текст *undefined*. Методът връща съдържащия се в текстовата кутия символен низ.

Да разгледаме един пример

```
<HTML><BODY>
<SCRIPT language="JavaScript">
  var x=prompt("Въведете стойност на x ","1");
  alert("sin(x)="+Math.sin(x));
</SCRIPT></BODY></HTML>
```

Пример 116 Използване на методи alert и prompt на обект window

В примера се въвежда стойност на променлива x с диалогова кутия prompt. Методът prompt връща символен низ, който се използва като параметър на функцията sin(x) – метод на обект Math. Методът alert извежда диалогова кутия със символен низ "sin(x)=" , към който е слепена изчислената стойност на sinus(x), преобразувана в символен низ.

Методи за създаване на програмен таймер

В програмирането таймерът е средство, с което се предизвиква еднократно или периодично изпълнение на код след зададен времеинтервал. Обектът window предоставя два метода за създаване на програмен таймер и два метода за унищожаване на създаден таймер.

Метод	Описание
timerID= setTimeout (низ_команди, интервал)	Създава програмен таймер за еднократно изпълнение на *низ_команди* след *интервал* в милисекунди и връща идентификатор на таймера *timerID*. Изпълнението на метода не блокира скрипта – изпълнението на скрипта продължава, а след изтичане на зададения времеинтервал се изпълнява низа от команди.
timerID=	Създава програмен таймер за периодично изпълнение на

setInterval (низ_команди, интервал)	*низ_команди* с период *интервал* в милисекунди. Изпълнението на метода не блокира скрипта – изпълнението на скрипта продължава, а след всяко изтичане на зададения времеинтервал се изпълнява низа от команди.
clearTimeout (timerID)	Унищожава таймер, създаден с метода setTimeout. Трябва да получи като параметър идедтификаторът на таймера timerID
clearInterval (timerID)	Унищожава таймер, създаден с метода setInterval. Трябва да получи като параметър идедтификаторът на таймера timerID

Най-често *низ_команди* съдържа обръщение към функция, декларирана в скрипта, но може да съдържа и просто няколко инструкции на JavaScript. Да разгледаме един пример:

```
<HTML><BODY>
<SCRIPT language="JavaScript">
function showTime(){
    theTime=new Date();
    window.status="Time "+ theTime.getHours()+":"+
    theTime.getMinutes()+":"+theTime.getSeconds();
    setTimeout("showTime()",600);
}
setTimeout(“showTime()”,600)
</SCRIPT>
</BODY></HTML>
```

Пример 117 Създаване на програмен часовник с метода setTimeout на обект window.

В блока <SCRIPT> на примера е включена декларация на функцията showTime, която създава обект theTime от тип Date, извлича елементите на текущото време посредством методите getHours, getMinutes и getSeconds. и извежда в статус-лантата на прозореца текущото време. Последната инструкция във функцията е обръщение към метода setTimeout, с което се предизвиква изпълнението на същата функция след 600 милисекунди. Същата инструкция setTimeout(“showTime()”,600) е включена извън декларацията на функцията и се изпълнява веднага при зареждане на документа. Така след 600 ms се предизвиква изпълнението на функцията showTime, която след това стартира сама себе си. Резултатът е, че на всеки 600 ms се извежда текущата стойност на времето в статус-лентата на прозореца.

Важно е да се запомни, че JavaScript се изпълнява от Web браузърите като само една последователност от инструкции, или, ● другояче казано – като една нишка (thread). Таймерите не създават отделни нишки, а само генерират събития, които се изпълняват в цикъла за изпълнение на JavaScript кода на виртуалната машина на JavaScript. Че това е така можем да се убедим като изведем диалогова кутия с метод alert. Докато тя не бъде затворена всички таймери спират да работят, което нямаше да се случи ако кода им се изпълняваше в отделни нишки.

Програмно извеждане на прозорец на браузъра. Методи open и close.

Методът open на обект window дава възможност програмно да бъде изведен прозорец на браузъра. Методът връща обектна променлива за прозореца, посредством която се получава достъп до атрибутите и методите на изведения прозорец.

Синтаксис:

open(URL, име_на_прозореца[, параметри])

където:

URL е символен низ, който съдържа адреса на документа, който да се зареди в прозореца. Може да бъде интернет адрес, адрес на файл от потребителския компютър или празен низ – в тоя случай се извежда празен прозорец.

име_на_прозореца е символен низ, който задава име на прозореца. Този параметър може да се използва като стойност на атрибут **target** на хипервръзка или формуляр, за да се зареди (програмно или от друг прозорец) документ в изведения с open прозорец.

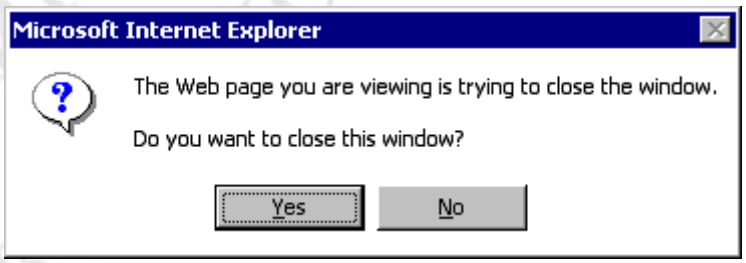
параметри е символен низ, който задава стойности на параметри на прозореца. Ако този параметър липсва, се извежда прозорец с всички опции – меню, лента с бутони, адресна текстова кутия и т.н. Ако низът с параметри е включен при извикване на метода open, то се счита, че всички опции на прозореца, за които не са зададени стойности са изключени. В низа *параметри* могат да се зададат следните опции:

опции	Описание
width	Задава дължината на прозореца в пиксели.
height	Задава височината на прозореца в пиксели.

location	При стойност yes извежда полето за избор или въвеждане от клавиатурата на адреса (URL) на документа, който да бъде зареден.
menubar	При стойност yes извежда менюто (File, Edit, …) на браузъра.
resizable	При стойност yes разрешава на потребителя да променя ръчно размерите на прозореца.
scrollbars	При стойност yes разрешава извеждането на хоризонтална и скрол ленти ако размерът на документа е по-голям от размера на прозореца.
status	При стойност yes извежда статус лента на прозореца.
toolbar	При стойност yes, извежда стандартната лента с бутони на браузъра, като например бутоните Back и Forward.

Както се вижда от таблицата, повечето от опциите, задавани в таблицата приемат стойности yes или no. Тези стойности могат да бъдат зададени съответно като 1 или 0.

Метод close на обект window дава възможност програмно да бъде затворен прозорецът на браузъра. Ако изпълните този метод за прозореца на браузъра с инструкция window.close(), ще се изведе диалогова кутия



Фиг. 18 Диалогова кутия при опит да бъде затворен текущият прозорец с window.close()

Да разгледаме един пример за използване на методите open и close.

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
var w1=0;
function win_open(){
    if(!w1)w1=window.open("","","width=300,height=100,resizable=1");
}
```

```
function win_close(){
    if(w1)w1.close()
    w1=0;
}
</SCRIPT>
</HEAD><BODY>
<a href="javascript:win_open()">Open window</a><BR>
<a href="javascript:win_close()">Close window</a>
</BODY></HTML>
```

Пример 118 Отваряне и затваряне на прозорец чрез методите open и close на обект window.

В примера в блока <SCRIPT> е декларирана глобална променлива w1, която се инициализира с 0.Чрез първата хипервръзка с текст “Open window” се извиква функцията win_open, която отваря прозорец с метода open на обект window. При това на променливата w1 се присвоява обектната променлива на прозореца*. Чрез втората хипервръзка с текст “Close window” се извиква функцията win_close, която затваря прозореца с метода close на обект window. При това на променливата w1 се присвоява стойност 0, което във функцията win_open дава възможност за проверка дали е бил затворен прозореца.

Операции преместване, преоразмеряване и скролиране на прозореца.

Обект window предоставя двойка методи за всяка от изброените операции - преместване, преоразмеряване и скролиране на прозореца, като единия метод (“By”-метода) извършва операцията относително текущото състояние на прозореца, а другия метод (“To”-метода), задава нови стойности на съответния параметър, без да отчита текущото състояние:

Метод	Описание
moveBy (±x, ±y)	Премества прозореца на разстояние ±x, ±y спрямо текущото му положение.
moveTo(x,y)	Премества горния ляв ъгъл на прозореца на координати x и y.
resizeBy	Променя размера на прозореца със стойности ±x, ±y спрямо текущия му размер. Ако прозореца е максимизиран,

* Често това наричат референция – още един “побългарен” термин от документацията на английски.

(±x, ±y)	размерите му няма да бъдат променени.
resizeTo(x,y)	Променя размерите на прозореца на x, y.
resizeBy (±x, ±y)	Скролира съдържанието на прозореца на разстояние (в пиксели) ±x в хоризонтална посока и ±y във вертикална посока спрямо текущото положение на документа в прозореца,
resizeTo(x,y)	Скролира съдържанието на прозореца на x пиксела по хоризонтала и y пиксела по вертикала.

Символите "±" означават, че съответния параметър на метода може да приема както положителни, така и отрицателни числови стойности. Важно е да се отбележи, че браузърите разрешават преместване, преоразмеряване и скролиране на прозореца посредством методите на обект window само ако в него не е отворен документ или е отворен документ от потребителския компютър или от текущата директория. Ако се опитате да изпълните някоя от тези операции за прозорец, в който е отворен документ от друг сайт, браузърът ще откаже да изпълни операцията и ще генерира грешка "Access is denied" (Отказан достъп). Това е така, защото подобни операции с документи от чужди сайтове се считат за некоректни.

Установяване на фокус върху прозорец на браузъра. Метод focus

По правило на екрана на монитора прозорецът, който е активен, (или, както се казва още, е във фокус) се изобразява над останалите прозорци. В някои случаи е желателно да можем програмно да установяваме фокуса върху определен прозорец. Такава възможност за прозорец на браузъра ни дава методът focus на обект window.

Синтаксис

focus()

Можем да установим фокус върху всеки прозорец на браузъра, за който в скрипта имаме на разположение обектна променлива. Например ако отваряме прозорец чрез код на JavaScript, и активираме кода чрез бутон или хипревръзка, след изпълнение на метода open създадения прозорец няма да има фокус и (напр. ако прозореца с хипервръзката е максимизиран) може да не се вижда върху екрана. За да избегнем това, след метода open можем да изпълним метода focus за създадения прозорец.

```
<HTML><HEAD>
```

```
<SCRIPT language="JavaScript">
var w1=0;
function win_open(){
     w1=window.open("","","width=300,height=100,resizable=1");
     w1.focus(); //установяване на фокуса върху отворения прозорец
}
</SCRIPT>
</HEAD><BODY>
<a href="javascript:win_open()">Open window</a><BR>
</BODY></HTML>
```

Пример 119 Използване на метода focus на обект window

Отпечатване на съдържанието на прозореца. Метод print

Методът print на обект window активира функцията за отпечатване на документа в прозореца на браузъра. При това се извежда стандартната диалогова кутия за отпечатване и на оператора се дава възможност да избере печатащото устройство, брой копия, страниците за отпечатване и т.н. Синтаксис:

print()

Ръчното извикване (чрез хипервръзка или бутон) на функция на JavaScript, която да изпълни метода print за прозореца на браузъра няма особен смисъл, тъй като същия резултат може да се получи и ако се щракне с мишката върху фунцията Print в менюто на браузъра. За вътрешен фрейм в прозореца на браузъра обаче поставянето на бутон или хипервръзка за отпечатване определено е полезно, тъй като не може да се очаква, че всеки потребител ще знае, че може да се извика функцията Print за фрейм чрез плаващо меню, активирано с десния бутон на мишката.

Ето например как може чрез хипервръзка да се изпълни метода print за текущия прозорец или фрейм

```
<a href="javascript:self.print()">Отпечатване</a><BR>
```

Пример 120 Хипервръзка за отпечатване на съдържанието на текущия прозорец или фрейм на браузъра чрез метод print()

21.7.5 Операции с адресите на вече зареждани документи – обект history.

Обектът History е обект от второ ниво на обектния модел на документа (DOM). Той е дефиниран в JavaScript, като достъпът до него се осъществява посредством атрибута history на обект window, затова по-нататък ще го обозначаваме като history. Той съдържа масив с информация за адресите (URL) на документите, зареждани в текущия прозорец на браузъра. В съответствие с това, обект history предоставя следните атрибути и методи:

Атрибути на обект history

Атрибут	Описание
current	Връща URL на текущия документ.
length	Връща броя записи в масива на обект history.
next	Връща URL от следващия запис в масива на обект history. Това е адрес на документ, от който сме се върнали обратно на текущия документ с бутона Back или с метод back() или go(-1).
previous	Връща URL от предходния запис в масива на обект history. Това е адрес на документ, след който е зареден текущия документ.

Методи на обект history

Обект history предоставя три метода, чрез които в прозореца на браузъра може да се зареди вече зареждан документ:

Зареждане на предидущ документ. Метод back

Зарежда в прозореца (или фрейма) на браузъра документа, който е бил зареден преди текущия. Това е документа, чийто URL е записан в масива на обект history. Ако такъв запис в масива на history липсва, методът back няма да се изпълни.

Синтаксис:

back()

Кода на хипервръзка, която да дава възможност на оператора да зареди предидущия документ ще има вида:

```
<a href="javascript:history.back()">Връщане назад</a>
```

Пример 121 Хипервръзка за връщане на предидущ документ с използване на метода back на обект history

Зареждане на следващ документ. Метод forward

Зарежда в прозореца (или фрейма) на браузъра документа, от който операторът се е върнал на текущия документ чрез бутона Back от менюто на браузъра или чрез метода back() на обект history. Това е документа, чийто URL е записан в масива на обект history след URL на текущия документ. Ако такъв запис в масива на history липсва, методът forward няма да се изпълни.

Синтаксис:

forward()

Зареждане на документ с URL от зададен запис в масива на обект history. Метод go

Методът go(n) на обект history получава като параметър номера на адреса на документа в списъка на history и зарежда в прозореца (или фрейма) на браузъра съответния документ. Ако зададеният номер запис в масива на history липсва, методът go няма да се изпълни.

Синтаксис:

go(n)

където n е номера на записа на URL в масива на обект history.

Отрицателните стойности на параметъра n съответстват на документи, които са били зареждани в нормалния ред преди текущия документ, а положителните стойности на n съответстват на документи, от които операторът се е върнал назад, напр. с бутона Back на браузъра. Така например go(-1) е еквивалентен на метод back(), а go(1) е еквивалентен на метод forward().

21.7.6 Извличане на елементи от адреса (URL) на текущия документ – обект location.

Обектът Location е обект от второ ниво на обектния модел на документа (DOM). Той е дефиниран в JavaScript, като достъпът до него се осъществява посредством атрибута location на обект window, затова по-нататък ще го обозначаваме като location. Обектът location представя пълния адрес на документа (URL) , асоцииран с обекта window. Всеки от

атрибутите на обект location представя определена част от URL. Форматът на URL е разгледан в т.9 “Интернет адреси” от първа глава. Общия формат на URL е:

protocol://hostname:port/pathname search hash

На елементите на URL съответстват атрибути на обект location както следва:

Атрибут	Описание
hash	Съдържа име на точка за преход (котва).
host	Съдържа *host* и *port*, разделени с двоеточие “:”
hostname	Съдържа името (или цифров адрес) на мрежовия компютър (host), от който е изтеглен документа.
href	Задава и връща целия URL на заредения в прозореца документ.
pathname	Съдържа пълната файлова спецификация, която включва път, име и разширение на файла.
port	Съдържа комуникационния порт, от който е зареден документа. Този елемент по правило се пропуска в URL, тъй като видът на протокола в повечето случаи определя и стойността на port. Например за http протокола номера на port е 80, за ftp – 21.
protocol	Съдържа началото на URL до първото двоеточие. За заявка заHTML документ протоколът е **http:** , за заявка за доставяне на файл – **ftp: ,** за активиране на услугата електронна поща – **mailto:** и т.н.
search	Съдържа информация за отговор на въпроси или предаване на параметри. В URL разделителят преди search е въпросителен знак. За заявки за HTML документи по правило този елемент отсъства.

Всички атрибути на обект location са само за четене, с изключение на атрибута href, който представя пълния URL на текущия документ. Ако се запише нова стойност в атрибута href и тя е валиден URL, това ще предизвика изпращане на заявка за документа и зареждането му в

прозореца. Това е най-простият начин за програмно зареждане на документ в прозорец на браузъра. Да разгледаме един пример:

```
<HTML>
<BODY OnKeyDown="location.href='http://www.yahoo.com' ">
Зареждане на документ в прозорец на браузъра чрез атрибута href на обект location
</BODY></HTML>
```

Пример 122 Зареждане на документ в прозорец на браузъра чрез атрибута href на обект location.

В примера в етикета BODY е включен манипулаторът на събитие OnKeyDown, на който се присвоява низ, съдържащ инструкция на JavaScript “location.href='http://www.yahoo.com' “ Когато операторът натисне бутон от клавиатурата се генерира събитието KeyDown и се изпълнява инструкцията, която присвоява на атрибута href адреса на началната страница на сайта на Yahoo и тя се зарежда в прозореца на браузъра.

Методи на обект location

Обект location предоставя два метода, които предизвикват изпращане от браузъра на заявка за текущия или за друг документ.

Метод reload (обект location)

Методът reload изпраща нова заявка за документа, който е зареден в прозореца на браузъра.

Синтаксис:

reload([forceGet])

където forceGet е параметър, който приема логическа стойност. Стойност true има като резултат обязателно презареждане на документа, следователно reload(true) ще има същия резултат, както изпълнението на функцията Refresh на Internet Explorer или Reload на Netscape Navigator. Стойност false на параметъра forceGet има като резултат презареждане на документа само ако той е бил променен на сървъра, или ако времето за съхраняването му в буферната памет на сървъра е изтекло. Следователно изпълнението на метода reload(false) ще има същия резултат, както щракването с мишката върху бутона “Go” на браузъра.

Метод replace (обект location)

Методът replace зарежда документ в прозореца на браузъра по зададен адрес (URL) без да прави запис в масива на обект history. Това означава, че ако бъде зареден нов документ чрез метода replace, операторът не може да се върне на преди това зареденият документ с бутона Bask на браузъра или чрез методите на обект history. Това качество на метода replace може да бъде полезно тогава, когато по някакви причини искаме програмно да пренасочим браузъра към друга страница без операторът да забележи това,

Синтаксис:

replace(URL)

където URL е символен низ, съдържащ адреса на документа, който да бъде зареден.

Да разгледаме един пример:

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
var s=navigator.userAgent;
if(s.indexOf("MSIE")==-1){
     location.replace("index_netscape.htm");
}
</SCRIPT></HEAD>
<BODY>
<center><H1>Internet Explorer Version</H1></center>
</BODY></HTML>
```

Пример 123 Пренасочване на браузъра към друга страница посредством метода replace на обект location.

В примера в блока <SCRIPT> е вложен код, с който се прави проверка за вида на браузъра*. В променливата s се записва съдържанието на атрибута userAgent на обект navigator и след това се прави проверка посредством метода indexOf на обект String дали низът в променливата s съдържа “MSIE”. Ако “MSIE” не се съдържа в низа, получен от атрибута userAgent (тогава методът indexOf връща -1), това означава, че браузърът не е Internet Explorer и посредством метода replace на обект location в браузъра се зарежда HTML страницата index_netscape.htm. Тъй като блока <SCRIPT> е вложен в заглавната част на HTML документа, то потребителят, който зареди страница с кода от примера в браузър Netscape, няма да види изобщо нейното съдържание в прозореца на браузъра, а само съдържанието на index_netscape.htm (ако такъв файл съществува в същата директория, в която е и файлът от примера).

* Вижте описанието на обект navigator

Иван Антонов — *Вие на кого вярвате – на човека или на документа?*

Чиновникът — *На документа разбира се. Хора всякакви.*

Станислав Стратиев "*Сако от велур*"

21.7.7 Програмен достъп до съдържанието на HTML документите – обект document

Обектът document е обект от второ ниво на обектния модел на документа (DOM). Той е дефиниран в JavaScript, като достъпът до него се осъществява посредством атрибута document на обект window. Обектът document преставя съдържанието на HTML документа. Достъпът до всички обекти, които представят определени елементи от HTML документа – хипервръзки, изображения, текстови кутии, списъчни обекти, бутони и т.н. , се осъществява чрез обект document.

Обектът document предоставя набор от атрибути и методи, свързани със съдържанието на HTML страницата.

Манипулатори на събития на обект document

Обект document притежава следните манипулатори на събития -

onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseUp. Програмистът на JavaScript може да присвои на всеки от тези манипулатори функция или символен низ, съдържащ команди на JavaScript, които ще се изпълнят при възникване на съответното събитие. Стойност на манипулатор на събитие за обект document може да се присвои в етикета <BODY> на HTML документа или чрез инструкция на JavaScript, както това е описано в т. 5.6.2. Да разгледаме един пример:

```
<HTML>
<BODY OnClick="alert('Click')">
Test Click and DblClick events
<SCRIPT language="JavaScript">
    function myKeyPress(){
        alert("KeyPress");
    }
    document.onkeypress=myKeyPress;
</SCRIPT>
</BODY></HTML>
```

Пример 124 Прехващане на събития Click и KeyPress на обект document.

В примера манипулаторът на събитие Click се инициализира чрез атрибута OnClick на етикета BODY, като му се присвоява символен низ, който съдържа инструкцията alert(‘Click’). На манипулатора onkeypress се присвоява функцията myKeyPress, декларирана в блока <SCRIPT>, която съдържа инструкцията alert("KeyPress"). Като резултат при щракване с мишката в прозореца на браузъра се извежда диалогова кутия с текст “Click”, а при натискане на символен бутон от клавиатурата се извежда диалогова кутия с текст “KeyPress”. Обърнете внимание на разликата в изписването на манипулатора на събитие в етикета BODY и в инструкцията на JavaScript. Етикета BODY е в кода на HTML документа, където не се прави разлика между малки и главни букви. В инструкцията на JavaScript важи синтаксиса на JavaScript и там се прави разлика между малки и главни букви. В JavaScript всички манипулатори на събития се изписват изцяло с малки букви.

Атрибути на обект document

Атрибутите на обекта document представят различни параметри на HTML документа. Всички атрибути след зареждане на HTML документа са достъпни само за четене, с изключение на атрубута bgColor (представящ фоновия цвят на прозореца на браузъра), който може да бъде променян по всяко време. Атрибутите ще бъдат класифицирани според тяхното предназначение:

Атрибути, които задават и връщат цвят на елементи от документа

bgColor	Символен низ, представящ атрибута BGCOLOR на етикет <BODY>. Задава и връща цвета на фона на прозореца.
fgColor	Символен низ, представящ атрибута TEXT на етикет <BODY>. Връща подразбиращия се цвят на текста на HTML документа.
linkColor	Символен низ, представящ атрибута LINK на етикет <BODY>. Задава и връща подразбиращия се цвят на хипервръзките.
alinkColor	Символен низ, представящ атрибута ALINK на етикет <BODY>. Задава и връща подразбиращия се цвят на активните хипервръзки.
vlinkColor	Символен низ, представящ атрибута VLINK на етикет <BODY>. Задава и връща подразбиращия се цвят на посетените хипервръзки.

Както беше посочено по-горе, всички тези атрибути са само за четене – не могат да бъдат променяни след като документът е зареден. Изключение прави само атрибута bgColor. Неговото съдържание може да бъде променяно по всяко време. Важно е да се запомни, че променянето на стойността на bgColor става по различен начин за Internet Explorer (IE) и Netscape Navigator (NE). За IE стойността на bgColor може да се зададе като число или като символен низ, но за NE трябва да се зададе обязателно като символен низ, представящ цвета, така, както това се задава за атрибута BGCOLOR на етикет <BODY>.

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
function change_bgColor(){
    var s=navigator.userAgent;
    if(s.indexOf("MSIE")==-1){
        document.bgColor="#0000ff";  //за Netscape
    }else{
        document.bgColor=0x0000ff;   //за Internet Explorer
    }
}
</SCRIPT></HEAD>
<BODY OnClick="change_bgColor()">
<center><H1>Click to change background color</H1></center>
</BODY></HTML>
```

Пример 125 Променяне на фоновия цвят на HTML документ чрез атрибута bgColor на обект document

В примера се проверява вида на браузъра посредством атрибута userAgent на обект navigator*. Ако браузърът е NE, на атрубута bgColor се присвоява символен низ, задаващ син цвят, така, като това може да се направи директно за атрубута BGCOLOR на етикета BODY. Ако браузъра е IE, тогава на атрубута bgColor се присвоява шестнадесетично число, съответстващо на кодирането на син цвят в HTML документите. Функцията change_bgColor се изпълнява при щракване с мишката в прозореца на браузъра, тъй като е присвоена на манипулатора OnClick в етикета <BODY>.

Атрибути, които връщат масиви, съдържащи обектни променливи

Атриб	Описание

* Вижте описанието на обект navigator

ут	
images	Масив, съдържащ обектни променливи за изображенията, съдържащи се в документа.
links	Масив, съдържащ обектни променливи за хипервръзките в документа.
anchors	Масив, съдържащ обектни променливи за точките за преход (котви) съдържащи се в документа.
forms	Масив, съдържащ обектни променливи за формулярите, съдържащи се в документа.
applets	Масив, съдържащ обектни променливи за аплетите, съдържащи се в документа.
layers	(само за NE) Масив, съдържащ обектни променливи за слоевете, съдържащи се в документа.
embeds	(само за NE) Масив, съдържащ обектни променливи за обектитве, включени в HTML документа с етикет <EMBED>
plugins	(само за NE) Масив, съдържащ обектни променливи за обекти Plugin – разширения на браузъра.

Стойността на атрибута length на масивите дава дава информация за броя обекти, включени в даден масив. Например броя изображения, съдържащи се в HTML документа можем да получим от атрибута length на масива images:

```
window.status=”Number of images ”+document.images.length;
```

Пример 126 Извеждане на броя изображения в HTML документ в статус-лентата на прозореца.

Други атрибути на обект document

title	Връща символен низ със съдържанието на контейнерния етикет TITLE. Ако етикетът липсва, връща стойност “undefined”.
URL	Връща символен низ, който съдържа пълния адрес (URL) на документа. Можем да получим същата стойност от атрибута href на обект location.
referrer	Връща URL на страницата, от която е извикана текущата страница. Това по принцип е полезна информация, проблема е в това, че ще я получим само ако сървъра, от който е заявен HTML документа, я изпраща.
cookie	Връща бисквидките, асоциирани с документа във вид на символен низ, като отделните бисквидки са разделени с “;”.

Няколко думи за бисквидките: Бисквидките представляват текстова информация, която се съхранява на потребителския компютър. Те се изпращат от потребителския браузър към сървъра когато браузърът изпраща заявка за документ и от сървъра към браузъра когато се изпраща документ като отговор на заявката. Потребителят може да забрани записването на бисквидки на компютъра си посредством съответната функция от менюто на браузъра (напр. за Internet Explorer това може да се направи чрез страницата “Security” в диалоговата кутия на функция “Internet Options” от менюто “Tools”. Тази възможност прави за програмиста на JavaScript бисквидките полезни само като опция.

Методи на обект document

Обект document предоставя няколко метода, свързани с извеждането на информация в прозореца на браузъра. По-долу те ще бъдат разгледани последователно. Важно е да се запомни, че тези методи не просто извеждат информация в прозореца на браузъра, а заменят информацията, която първоначално е била заредена с тази, която се извежда с тези методи. В повечето случаи такова заменяне е нежелателно и затова по-голямо практическо значение имат атрибутите и методите, които дават възможност за променяне и добавяне към съдържанието на HTML документа. Те се разглеждат в раздела “Динамичен HTML”.

Отваряне на изходен поток за извеждане в прозореца на браузъра. метод open на обект document.

Методът open на обект документ отваря изходен поток за изпращане на данни към прозореца на браузъра като дава възможност за задаване на типа на данните, който се изпращат. Изходният поток е термин, който се използва за обозначаване на операцията “изпращане на данни към някакъв приемник”. Такъв приемник може да бъде както физическо устройство (монитор, печатащо устройство) така и логическо устройство (файл). Прозорецът на браузъра е лоогическо устройство, което може да приема данни от зададен тип. Методът open има следния синтаксис:

open([mimeType, [replace]])

където:

mimeType (незадължителен) е символен низ, който задава типа на документа, в който ще се извършва запис.Ако този параметър не бъде зададен, се подразбира тип “text/html”. Това означава на практика, че в тоя

случай изпращаните към браузъра данни ще се обработват като съдържание на HTML документ.

replace (незадължителен) е символен низ “replace”, който трябва да бъде задаван само ако mimeType е “text/html”. В тоя случай стойността “replace” означава, че новия HTML документ, който ще бъде създаден, ще притежава съдържанието на масива history на текущия HTML документ и следователно ще може с бутоните Back и Forward на браузъра, а също и с методите на обект history,в него да се зареждат преди това зареждани документи.

Важно е да се запомни, че извикването на метода open изчиства текущото съдържание на HTML документа и той ще бъде празен дотогава, докато към прозореца набраузъра не бъдат изпратени данни с методите write или writeln на обект document.

Затваряне на изходния поток за извеждане в прозореца на браузъра - метод close на обект document.

Методът close затваря изходния поток към прозореца на браузъра. При това изпращането на данни приключва и в статус-лентата на браузъра се извежда съобщението “Document: Done “.

Синтаксис

close()

Изходния поток се затваря автоматично при затваряне на прозореца или при зареждане в него на друг документ.

Изпращане на данни към прозореца на браузъра - методи write и writeln.

Методите write и writeln изпращат към браузъра стойността на зададен израз или последователност от изрази. Ако преди първото изпълнение на метод write или writeln не е бил изпълнен метод open, то методът open се извиква автоматично и по тоя начин се отваря изходен поток за извеждане в документ от тип “text/html”. В тоя случай, ако текущият документ вече е бил зареден, първото изпълнение на метод write или writeln ще има за резултат изчистването на прозореца на браузъра и извеждането в него само на изпратеното от метода. Следващите изпълнения на метод write или writeln ще добавят код в създадения документ, а следователно и в прозореца на браузъра.

Синтаксис:

document.write(expr1[, ...,exprN])

document.writeln(expr1[, ...,exprN])

където expr1, ...,exprN са изрази на JavaScript.

Разликата между write и writeln е само в това, че методът writeln добавя след данните от изразите кода на символа за преминаване на нов ред (CR). В случай че се изпращат данни от тип “text/html” това не еобязателно необходимо, тъй като знаем, че по подразбиране браузърът не отработва преминаването на нов ред, ако е включено в текста в HTML документ.Все пак наличието на символи за преминаване на нов ред е полезно при разглеждане на сорс кода на документа с текстов редактор – в противен случай целия код се извежда на един текстов ред.

21.7.8 Тяло на HTML документите – обект body

Атрибути, които задават и връщат цвят на елементи от документа

Атрибут	Описание
bgColor	Символен низ, представящ атрибута BGCOLOR на етикет <BODY>. Задава и връща цвета на фона на прозореца.
background	Символен низ, представящ атрибута BACKGROUND на етикет <BODY>. Задава и връща адреса (URL) на фоновото изображение на документа.
text	Символен низ, представящ атрибута TEXT на етикет <BODY>. Задава и връща подразбиращия се цвят на текста документа
link	Символен низ, представящ атрибута LINK на етикет <BODY>. Задава и връща подразбиращия се цвят на хипервръзките в документа
aLink	Символен низ, представящ атрибута ALINK на етикет <BODY>. Задава и връща подразбиращия се цвят на активните хипервръзки в документа
vLink	Символен низ, представящ атрибута VLINK на етикет <BODY>. Задава и връща подразбиращия се цвят на посетените хипервръзки в документа

Тези атрибути имат същото значение, както аналогичните атрибути на обект document, но с тази съществена разлика, че стойностите им могат да бъдат променяни по всяко време и това предизвиква промяна на подразбиращия се цвят на съответните елементи на HTML документа. Да разгледаме един пример, в който посредством атрибута text на обект body

се променя цвета на текста в прозореца на браузъра след като HTML документа вече е бил зареден.

```
<HTML><HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<SCRIPT LANGUAGE = "JavaScript">
function change_text_color(){
    document.body.text="red";
}
</SCRIPT></HEAD><BODY>
Под една кичеста круша в една от трендафиловите градини било постлано черно-жълто китено чердже, на което седели две лебеници ...
<p align="right"> Любен Каравелов: <i> Маминото детенце</i></p>
<BR><a href="javascript: change_text_color()">Change Text Color to Red</a>
</BODY></HTML>
```

Пример 127 Променяне на цвета на текста в HTML документ посредством атрибута text на обект body.

Атрибути, които се поддържат от браузърите, но не са стандартизирани от W3C

Атрибут	Описание
id	Задава и връща the идентификатора на елемента <BODY> За IE 4 този атрибут е само за четене.
scrollLeft	Задава и връща разстоянието от лявата граница на документа до най-лявата извеждана част от съдържанието. Стойността на този атрибут показва с колко е скролирано съдържанието на документа в хоризонтална посока.
scrollTop	Задава и връща разстоянието от горната граница на документа до най- горната извеждана част от съдържанието. Стойността на този атрибут показва с колко е скролирано съдържанието на документа във вертикална посока.

Атрибутите scrollLeft и scrollTop имат същото значение както атрибутите pageXOffset и pageYOffset, отново с тази съществена разлика, че стойността им може да се променя програмно, което предизвиква скролиране на съдържанието на документа. Такова скролиране може да се направи също посредством метода scrollTo(x,y) на обект window.

Атрибути, които връщат размера на клиентската област на прозореца на браузъра

Атрибут	Описание
clientWidth	Връща размера на документа в хоризонтална посока.
clientHeight	Връща размера на документа във вертикална посока.

Атрибути, които се поддържат само от Internet Explorer

Атрибут	Описание
offsetWidth	Връща размера на документа в хоризонтална посока.
offsetHeight	Връща размера на документа във вертикална посока.

За Netscape Navigator размера на документа може да бъде получен посредством атрибутите **innerWidth** и **innerHeight** на обект window – такива атрибути в Internet Explorer обект window не притежава. И двата браузъра обаче притежават атрибутите client**Width и** client**Height** на обект body, които връщат размера на клиентската област на прозореца на браузъра, така, че в случаите, в които ни е нужен този размер, не е нужно да проверяваме типа на браузъра.

21.7.9 Операции с изображения в HTML документите – обект Image

Обектът Image представя изображение, включено в HTML документа. Обектът се създава от етикет HTML <IMG> или посредством конструктор.

Конструкторът на обект Image има следния синтаксис:

new Image([width,] [height])

където незадължителните параметри *width* и *height* задават размерите на изображението в пиксели. Конструкторът връща обектна променлива за създадения обект от тип Image, посредством която се получава достъп до атрибутите, методите и манипулаторите на събития на обекта. Ако обектът от тип Image е създаден посредством HTML етикет <IMG>, интерпретаторът на JavaScript записва обектната променлива на изображението в масива images на обект document. В тоя случай достъп до обекта може да се получи съответният елемент от масива images или посредством името на изображението – съдържанието на атрибута name на етикета <IMG>. Да разгледаме един пример:

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
```

```
function show_images(){
    var im3 = new Image(100,100);
    im3.name="im3";
    var s="<OL>";
    for(var n=0;n<document.images.length;n++){
        s+="<LI>"+document.images[n].name;
    }
    document.writeln(s+"</OL>");
}
</SCRIPT></HEAD>
<BODY onLoad="show_images()">
<img src="image1.gif" name="im1"><BR>
<img src="image2.gif" name="im2">
</BODY></HTML>
```

Пример 128 Извеждане на списък с имената на изображенията, записани в масива images на обект document.

В примера в HTML документа са включени два етикета <img>. За всеки от тях интерпретаторът на JavaScript създава обект Image и го записва в масива images на обект document. Във функцията show_images, която е присвоена на манипулатора на събитието Load, се създава обект im3 от тип Image, за който обаче не се записва обектна променлива в масива images, защото обекта е създаден с конструктор. Чрез цикъл for се преминава през всички елементи на масива images и в променливата s се записва кода на HTML списък, който съдържа имената (т.е. съдържанията на атрибутите name) на изображенията. Съдържанието на променливата s се извежда в прозореца на браузъра чрез метода writeln на обект document. В случая ще бъде изведен списъка

1. im1
2. im2

Манупулатори на събития на обект image:

onAbort	Изпълнява се когато операторът прекрати изтеглянето на изображение, например с функцията Stop от менюто на браузъра
onError	Изпълнява се когато възникне грешка при изтегляне на изображението, напр. ако изтече допустимото за браузъра време за това.
onKeyDown	Изпълнява се при натискане надолу на бутон от клавиатурата, когато изображението е във фокус.
onKeyPress	Изпълнява се при натискане на бутон от клавиатурата, когато изображението е във фокус.

onKeyUp	Изпълнява се при отпускане на бутон от клавиатурата, когато изображението е във фокус.
onLoad	Изпълнява се след успешно изтегляне на изображението.

Манипулаторите на събития onAbort и onError дават възможност за изпълнение на код на JavaScript, ако понякакви причини изображението не бъде изтеглено. Както се вижда от таблицата, обектът от тип Image не притежава манипулатори OnClick, OnMouseOver и OnMouseOut, чрез които да реагира на щракване с мишката, навлизане на курсора ма мишката върху изображението или излизане от него. Ако това е необходимо, изображението може да се включи в хипервръзка и да се полват съответните манипулатори на събития на обект link.

Атрибути на обект Image

Атрибут	Описания
name	Задава и връща стойността на атрибута NAME - символен низ, който съдържа името на обекта.
alt	Задава и връща символен низ, който се извежда като подсказка ако курсора на мишката се задържи върху изображението. Ако изображението не се зареди, символният низ на alt се извежда на мястото на изображението.
border	Задава и връща стойността на атрибута BORDER– числова стойност, която определя дебелината на рамката в пиксели.
complete	Връща логическа стойност true когато изображението е заредено изцяло.
width	Задава и връща стойността на атрибута WIDTH– числова стойност, която определя хоризонталния размер на изображението в пиксели.
height	Задава и връща стойността на атрибута HEIGHT– числова стойност, която определя вертикалния размер на изображението в пиксели.
hspace	Задава и връща стойността на атрибута HSPACE– числова стойност, която определя разстоянието по оста X от изображението до съседните елементи на HTML документа в пиксели.
vspace	Задава и връща стойността на атрибута VSPACE – числова стойност, която определя разстоянието по оста Y от изображението до съседните елементи на HTML документа в

	пиксели.
src	Задава и връща стойността на атрибута SRC, който съдържа адреса на файла, съдържащ изображението.
lowsrc	Задава и връща стойността на атрибута LOWSRC, който съдържа адреса на файла, съдържащ изображението..

Важно е да се запомни, че атрибутите на обекта от тип Image могат да се променят по всяко време. Записа в атрибута src на друг адрес на изображение предизвиква изтеглянето на изображението и извеждането му в прозореца на браузъра на мястото на преди това изведеното изображение. Също така промяната на стойността на атрибутите width или height предизвиква преизчертаване на изображението с новите размери. Да разгледаме един прмер:

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
function resize_image(){
    document.im1.width=Math.random()*80+80;
    document.im1.height=Math.random()*80+80;
}
</SCRIPT></HEAD>
<BODY onClick="resize_image()">
<img src="FlyT.gif" name="im1"><BR>
</BODY></HTML>
```

Пример 129 Преоразмеряване на изображение посредством атрибутите width и height.

В примера е декларирана функция resize_image() , която при всяко извикване записва различни стойности в атрибутите width и height на изображението im1. Стойностите се получават чрез генератора за псевдослучайни числа random() – метод на обект Math, който връща стойност в интервала от [0÷1], мащабират се до интервала [0÷80] чрез умножаване по 80 и се транслират до интервала [80÷160] чрез прибавяне на константа 80. Така минималният размер, който ще получи изображението по X и Y е 80, а максималният е 160. При всяко щракване с мишката в прозореца на браузъра се извиква функцията resize_image(), променят се стойностите на атрибутите width и height и като резултат изображението се преизчертава с новите размери.

21.7.10 Операции с хипервръзки – обект Link

Обектът Link представя хипервръзка в HTML документа. Обектът се създава от етикет <A> или от етикет <AREA>, при условие, че етикетът съдържа атрибут src. Когато открие такъв етикет при обработка на HTML документ, интерпретаторът на JavaScript създава обект Link и го записва в масива links на обект document. Обект Link е единственият от обектите, които представят елементи на HTML документа, който няма атрибут name. Това е така, защото от един и същ етикет интерпретаторът на JavaScript може да създаде два различни обекта – от етикет <A> с атрибут href се създава обект Link, а от етикет <A> с атрибут name се създава обект Anchor.

Програмен достъп до хипревръзките може да се осъществи посредством масива links на обект document или, както е описано в раздела "Динамичен HTML", посредством атрибут id.

Манипулатори на събития на обект Link

Обектът Link притежава следните манипулатори на събития:

onClick, onDblClick, onKeyPress, onKeyUp, onKeyDown, onMouseDown, onMouseUp, onMouseOver, onMouseOut

Особено внимание заслужават манипулаторите onMouseOver и onMouseOut, които са специфични за хипервръзките. Събитието MouseOver се генерира еднократно когато курсорът на мишката навлиза в областта на хипервръзката, а събитието MouseOut се генерира тогава, когато курсорът на мишката излиза от областта на хипервръзката. Тези събития често се използват за създаване на т.н. "превъртащи се" изображения (rolowed images). Да разгледаме един пример:

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
    var down1 = new Image();
    down1.src="image1_down.gif";
    var up1 = new Image();
    up1.src="image1_up.gif";
</SCRIPT></HEAD>
<BODY>
<a href="http://www.abv.bg"
 OnMouseOver="document.im1.src=down1.src"
 OnMouseOut="document.im1.src=up1.src">
<img border="0" name="im1"src="image1_up.gif"></a><BR>
</BODY></HTML>
```

Пример 130 Създаване на сменящо се изображение посредством манипулаторите OnMouseOver и OnMouseOut на хипервръзка (обект Link).

В примера в блока <SCRIPT> се създават два обекта от тип Image – down1 и up1, и в тях се записват изображенията image1_down.gif и image1_up.gif. В тялото на HTML документа е включена хипервръзка, която съдържа изображението im1, в което първоначално се изобразява image1_up.gif. В етикета на хипервръзката са включени При навлизане на курсора на мишката върху изображението, се изпълнява инструкцията document.im1.src=down1.src, която заменя изображението със записаното в down1 изображение image1_down.gif. При излизане на курсора на мишката от областта на изображението се генерира събитието MouseOut и се изпълнява инструкцията document.im1.src=up1.src, която възстановява първоначалното изображение image1_up.gif.

Атрибути на обект Link

Обекта Link притежава атрибутите на обект location:

Атрибут	Описание
hash	Съдържа име на точка за преход (котва).
host	Съдържа *host* и *port*, разделени с двоеточие ":"
hostname	Съдържа името (или цифров адрес) на мрежовия компютър (host), от който е изтеглен документа.
href	Задава и връща целия URL на заредения в прозореца документ.
pathname	Съдържа пълната файлова спецификация, която включва път, име и разширение на файла.
port	Съдържа комуникационния порт, от който е зареден документа. Този елемент по правило се пропуска в URL, тъй като видът на протокола в повечето случаи определя и стойността на port. Например за http протокола номера на port е 80, за ftp – 21.
protocol	Съдържа началото на URL до първото двоеточие. За заявка заHTML документ протоколът е **http:** , за заявка за доставяне на файл – **ftp: ,** за активиране на услугата електронна поща – **mailto:** и т.н.

search	Съдържа информация за отговор на въпроси или предаване на параметри. В URL разделителят преди search е въпросителен знак. За заявки за HTML документи по правило този елемент отсъства.

и два допълнителни атрибута – target и title:

Атрибут	Описание
target	Съдържа символен низ с името на прозорец или фрейм, в който ще бъде зареден документа, заявен чрез хипервръзката. Ако атрибутът target липсва, документът се зарежда в прозореца, в който е хипервръзката.
title	Задава и връща текст, който се извежда като подсказка, когато курсорът на мишката се задържи върху обекта. Може да се променя по всяко време.

В много сайтове прозорецът на браузъра се разделя на няколко части (фреймове), като левия фрейм съдържа меню с текстови или графични хипервръзки. При щракване с мишката върху някоя хипервръзка се зарежда документ в десния фрейм, като за целта в етикета на хипервръзката се включва атрибут target на който се присвоява името на десния фрейм.

Важно е да се запомни, че съдържанието на атрибутите href и target на обект Link може да бъде променяно по всяко време. Това означава на практика, че след зареждането на един HTML документ програмно може да се променят неговите хипервръзки, така че чрез тях да се зареждат други HTML документи, а не първоначално зададените.

21.7.11 Точки за преход в HTML документите – обект Anchor

Обектът Anchor представя точка на преход (маркер) в HTML документа. Обектът се създава от етикет <A> (както и обект Link), но при условие, че етикетът съдържа атрибут name. Когато открие такъв етикет при обработка на HTML документ, интерпретаторът на JavaScript създава обект Anchor и го записва в масива anchors на обект document. Обектът Anchor не притежава манипулатори на събития.

Атрибути на обект Anchor

Атрибу	Описание

ти	
name	Връща символен низ, съдържащ името на обекта Anchor.
title	Задава и връща текст, който се извежда като подсказка, когато курсорът на мишката се задържи върху обекта. Може да се променя по всяко време.

21.7.12 Вътрешни фреймове в HTML документ – обект Iframe.

Обектът Iframe представя вътрешен фрейм, който се извежда в правоъгълна област в HTML документа. Обектът се създава от етикет <IFRAME>.

Атрибути на обект Iframe

Атрибут	Описание
width	Задава и връща дължината на фрейма в пиксели или проценти.
height	Задава и връща дължината на фрейма в пиксели или проценти.
name	Задава и връща символен низ, съдържащ името на обекта Iframe.
src	Задава и връща символен низ, съдържащ адреса на документа (URL), който е зареден във фрейма. Записа на нова стойност на src предизвиква зареждането на нов документ във фрейма.

Вътрешните фреймове са полезен инструмент за WEB дизайнера. Важно е да запомним, че можем програмно да променяме по всяко време съдържанието и размерите на вградения фрейм чрез атрибутите на обект Iframe. Ако искаме чрез хипервръзка от друг прозорец да заредим документ в обект Iframe, трябва да включим в хипервръзката атрибут target и да му присвоим стойността на атрибута name на обекта Iframe. Програмният достъп обаче до атрибутите на обект Iframe не може да се осъществява чрез името му, а чрез идентификатор id. Да разгледаме един пример:

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
function set_width()
{
    document.getElementById("myIframe").width="50%";
```

```
   }
   </SCRIPT></HEAD>
   <BODY>
   <IFRAME id="myIframe" name="theIFRAME" src="http://www.abv.bg" width="30%"></iframe>
   <BR><a href="javascript:set_width()">Set width to 50%</a>
   <BR><a href="http://www.yahoo.com" target="theIFRAME">Go to Yahoo</a>
   </BODY></HTML>
```

Пример 131 Зареждане на нов документ и променяне на размера на вътрешен фрейм посредством атрибути на обекта от тип Iframe.

В HTML документа, даден в примера, е включен вътрешен фрейм посредством етикет <IFRAME>. Документът съдържа две хипервръзки, като с първата посредством протокола javascript се стартира функция _width() , която изпълнява инструкцията

```
document.getElementById("myIframe").width="50%";
```

с която се задава на фрейма размер 50% от размера на прозореца на браузъра. Програмният достъп до обекта Iframe се осъществява чрез метода getElementById на обект document, който получава като параметър идентификатора на id обекта.

При щракване върху втората хипервръзка, във вътрешния фрейм се зарежда началната страница на сайта на Yahoo. Това е така, защото на атрибута target на хипервръзката е присвоена стойността на атрибута name на етикета <IFRAME>.

Обект document за прозореца на <iframe> може да се получи (в зависимост от вида на браузъра) или от атрибут contentDocument на обект Iframe или от атрибут document на неговия атрибут contentWindow, както е показано в даденият по-долу пример.

```
iframe =document.getElementById("myIframe");
iframedoc = iframe.contentDocument || iframe.contentWindow.document;
```

21.7.13 Операции с формуляри – обект Form.

Формулярите дават на потребителя възможност да въвежда и селектира данни и да ги изпраща им към програма, която се изпълнява на Web сървъра. Обектът Form представя формуляр в HTML документа. Обектът се създава от етикет <FORM>. Когато открие такъв етикет при обработка на HTML документ, интерпретаторът на JavaScript създава обект от тип Form и го записва в масива forms на обект document.

Обектите от тип Form са контейнер за обектите на формулярите. Това на практика означава, че програмният достъп до обектите за въвеждане и

селектиране на данни в един формуляр се осъществява чрез обекта Form на формуляра. Обектът Form предоставя атрибути и методи, които са свързани с предназначението на формуляра - изпращането на данни към интернет приложение.

Манипулатори на събития на обект Form

Обект Form предоставя два манипулатора на събития – onReset и onSubmit, които дават възможност за изпълнение на код на JavaScript при генериране на събития Reset и Submit.

onReset	Изпълнява се се когато операторът щракне с мишката върху бутон от тип Reset във формуляра.
onSubmit	Изпълнява се се когато операторът щракне с мишката върху бутон от тип Submit във формуляра.

Важно е да се запомни, че ако кодът, който се изпълнява чрез манипулатора onSubmit завършва с инструкцията `return false`, данните от формуляра няма да бъдат изпратени. Ако липсва инструкция return или ако тя връща друга стойност, данните се изпращат. Това дава възможност събитийната процедура, която се изпълнява чрез onSubmit да се използва за проверка на валидността на въведените от оператора данни. В случай, че има въведени невалидни данни, събитийната процедура може да откаже изпращането им като изпълни инструкция return false (т.е. като върне стойност false).

Атрибути на обект Form

Както беше посочено по-горе, атрибутите на обект Form са свързани с изпращането на данни от формулярите. На всеки атрибут на HTML етикета <FORM> със същото значение и допустими стойности съответства атрибут на обекта Form , който се създава чрез този етикет.

Атрибут	Описание
name	Представя атрибута NAME на етикет <FORM>. Задава и връща символен низ, съдържащ името на формуляра.
action	Представя атрибута ACTION на етикет <FORM>. Задава и връща символен низ, съдържащ адреса (URL), на който се изпращат данните.
method	Представя атрибута METHOD на етикет <FORM>. Задава и връща символен низ, съдържащ метода, по който се изпращат данните.

target	Представя атрибута TARGET на етикет <FORM>. Задава и връща символен низ, съдържащ името на прозореца (фрейма), на който трябва да се изпрати отговора за изпратените от формуляра данни.
encoding	Представя атрибута ENCTYPE на етикет <FORM>. Задава и връща символен низ, съдържащ типа кодиране, което се използва при изпращане на данните.
elements	Връща масив, който съдържа обектни променливи за елементите от формуляра.
length	Връща целочислена стойност – размера на масива elements, който е равен на броя елементи във формуляра.

Важно е да се запомни, че можем по всяко време да променяме основните атрибути на формулярите – action и target и така да променяме адреса, на който се изпращат данните от формуляра, а също прозореца (или фрейма), на който да се получава отговора от интернет приложението – получател на данните.

Масивът elements дава възможност за програмен достъп до елементите на формуляра. Да разгледаме един пример:

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
function show_elements()
{
    var s="<center><H1>List of form's elements</H1></center><HR>";
    s+="<OL>";
    for(var n=0;n<document.F1.elements.length;n++){
        s+="<LI>name=" + document.F1.elements[n].name;
    }
    document.writeln(s+"</OL>");
}
</SCRIPT></HEAD>
<BODY>
<FORM name="F1">
<input name="textbox1" type="text" value="Hello Form"><BR>
<input name="button1" type="Button" value="Show elements"
OnClick="show_elements()"><BR>
</FORM></BODY></HTML>
```

Пример 132 Извеждане на списък на елементите на формуляр посредством масива elements на обект Form.

В примера е декларирана функция show_elements(), в която е организиран цикъл, с индекс n , който се променя от 0 до

document.F1.elements.length-1 и чрез този индекс се адресират елементите на масива elements. За всеки елемент от масива се добавя елемент в кода на HTML списък, в който се записва името (стойността на атрибута name) на елемента от формуляра. След завършването на изпълнението на цикъла в променливата s е записан заглавният надпис и кода на HTML списъка и те се извеждат в прозореца на браузъра чрез метода open на обект document.

Методи на обект Form

Обект Form притежава два метода, чрез които могат да се извършат програмно същите операции, както при щракване с мишката върху бутони от тип SUBMIT и RESET.

Метод	Описание
submit ()	Изпраща данните от формуляра на адреса, зададен чрез атрибута ACTION на етикет <FORM>.
reset()	Инициализира елементите на формуляра. В текстовите кутии се записва началното им съдържание, а списъчните обекти и обекти за селектиране се установяват на началната селекция.

Използването на метода submit() на обект Form или аналогичния метод на обект Submit са *единствения начин програмно (с код на JavaScript от страна на браузъра) да се предизвика изпращането на данни по метода POST. Изпращане на данни по метода GET може да се извърши и посредством обект location - чрез запис в атрибута href на обект location на адреса за изпращане с добавени към него данни по синтаксиса на GET.

21.7.14 Обекти на формулярите

Формулярите могат да включват обекти за въвеждане на данни (едноредова и многоредова текстова кутия), обекти за избор (обекти checkbox и радиобутони), списъчни обекти и бутони (за изпращане на данни, за изчистване и с общо предназначение). Тези обекти имат няколко общи манипулатори на събития, атрибути и методи, които ще разгледаме в тази точка и след това за всеки от обектите ще разглеждаме допълнително само неговите специфични характеристики. Изключение прави само обекта Hidden, който не се изобразява (или, както е прието да казва, “няма графичен интерфейс”) и поради това не притежава повечето от общите за останалите обекти манипулатори на събития и методи.

* Като изключим AJAX заявките, които не се разглеждат в учебника

Общи манипулатори на събития за обектите на формулярите

Всички обекти на формулярите без обект Hidden притежават следните манипулатори на събития:

Манипулатор	Описание
onFocus	Изпълнява код на JavaScript когато обекта получи фокус.
onBlur	Изпълнява код на JavaScript когато обекта загуби фокус.
onClick	Изпълнява код на JavaScript когато операторът щракне с мишката върху обекта.

Общи атрибути за обектите на формулярите

Всички обекти на формулярите (с посочените изключения) притежават следните атрибути:

Атрибут	Описание
name	Задава и връща символен низ, съдържащ името на обекта.
form	Връща обектна променлива за формуляра, съдържащ обекта
type	Връща символен низ, съдържащ типа на обекта
value	(без обект Select) Задава и връща символен низ, съдържащ асоциирания с обекта текст.
title	(без обект Hidden)Задава и връща текст, който се извежда като подсказка, когато курсорът на мишката се задържи върху обекта.

Общи методи за обектите на формулярите

Всички обекти на формулярите (без обект Hidden) притежават методите:

Метод	Описание
focus()	Установява фокуса върху обекта. Това на практика означава, че обекта ще приема въведените данни от клавиатурата. Например ако фокуса е върху бутон и се натисне бутон Enter от клавиатурата, резултата е същия, както ако се щракне с мишката върху бутона
blur()	Премахва фокуса от обекта и го премества върху друг обект.

Фокуса може да се прехвърля последователно между обектите в прозореца на браузъра с бутона за табулация. Посредством метода focus() може програмно да се установява фокуса върху определен обект. Да разгледаме един пример:

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
function check_data()
{
    if(document.F1.user.value==""){
        alert("Enter user name");
        document.F1.user.focus();
    }
    else if(document.F1.psw.value==""){
        alert("Enter password");
        document.F1.psw.focus();
    }else document.F1.submit();
}
</SCRIPT></HEAD>
<BODY>
<FORM name="F1" action="http://www.abv.bg">
user <input name="user" type="text"><BR>
password <input name="psw" type="password"><BR>
<input type="BUTTON" value="Send" OnClick="check_data()"><BR>
</FORM></BODY></HTML>
```

Пример 133 Проверка за въведени данни във формуляр

В примера HTML документа съдържа формуляр с две текстови кутии за въвеждане на име на потребител и парола и бутон за изпращане на данните. При щракване с мишката върху бутона се изпълнява функцията check_data(), която е присвоена на манипулатора на събитие OnClick. Функцията check_data() съдържа две структури if-else, с които се проверява дали в текстовите кутии има въведени данни. Ако в някоя от текстовите кутии няма въведени данни, се извежда диалогова кутия със съобщение за оператора, и след нейното затваряне фокусът се установява върху текстовата кутия с метода focus(). Ако и в двете текстови кутии има въведен текст, данните от формуляра се изпращат на адрес http://www.abv.bg с метода submit(). Изпращането на данни на този адрес е направено само за тест – в сайта на abv.bg не се прави проверка дали има изпратени данни и като отговор се връща началната страница на сайта.

Важно е да се отбележи, че методът focus() ще се изпълни успешно само ако елементът е вече изобразен. Ако методът се извиква при

зареждане на страницата по-добре е focus() да се изпълни със закъснение, напр. с извикване чрез метода за таймер setTimeout на обект window.

```
<script>window.setTimeout(“document.F1.user.focus()”,2000) </script>
```

21.7.15 Едноредова текстова кутия – обект Text

Обектът Text представя едноредова текстова кутия в HTML формуляр. Обектът се създава от етикет <INPUT> с атрибут type=”TEXT”. Обектът Text предоставя манипулатори на събития, атрибути и методи за операции със съдържанието на текстовата кутия. Достъп до съдържанието на текстовата кутия се осъществява с атрибута value на обекта. Освен общите характеристики на обектите на формулярите, разгледани в т. 5.6.13 , обектът Text притежава следните специфични манипулатори на събития, атрибути и методи:

Специфични манипулатори на събития на обект Text

onChange	Предизвиква изпълнение на код на JavaScript когато обектът Text загуби фокус и съдържанието му е било променено.
onSelect	Предизвиква изпълнение на код на JavaScript когато операторът селектира текст в текстовата кутия
onKeyDown	Предизвиква изпълнение код на JavaScript когато операторът натисне бутон.
onKeyPress	Предизвиква изпълнение на код на JavaScript когато операторът натисне или задържи натиснат символен бутон.
onKeyUp	Предизвиква изпълнение на код на JavaScript когато операторът отпусне бутон.

Специфични атрибути на обект Text

defaultValue	Задава и връща символен низ с началното съдържание на текстовата кутия след зареждането на HTML документа. defaultValue се записва в текстовата кутия също така при инициализацията на формуляра с бутон RESET или с изпълнение на метода reset().

Специфични методи на обект Text

select()	Селектира съдържанието на текстовата кутия. След изпълнението на метода целия текст в текстовата кутия се извежда с инверсен цвят. Такова селектиране е полезно тогава,

	когато се очаква, че операторът ще копира или ще замени изцяло текста.

Да разгледаме един пример.

```
<HTML><HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</HEAD>
<BODY>
<center>Въведете име и щракнете с мишката извън текстовата кутия</center><HR>
<FORM name="F1">
Име <input name="user" type="text"
 OnChange="alert('Въведено е име '+document.F1.user.value)"><BR>
</FORM>
</BODY></HTML>
```

Пример 134 Използване на манипулатора OnChange на обект Text

В примера се демонстрира използването на манипулатора OnChange на обекта Text. Ако операторът промени съдържанието на текстовата кутия и щракне в прозореца на браузъра, се генерира събитие Change и се извежда диалогова кутия с текста “'Въведено е име “ и съдържанието на текстовата кутия. Ако обаче съдържанието на текстовата кутия не е променено, събитие Change не се генерира и диалогова кутия не се извежда.

21.7.16 Многоредова текстова кутия – обект Textarea.

Обектът Textarea представя многоредова текстова кутия в HTML формуляр. Обектът се създава от контейнерния етикет <TEXTAREA>. Обектът Textarea предоставя манипулатори на събития, атрибути и методи за операции със съдържанието на текстовата кутия. Освен общите характеристики на обектите на формулярите, разгледани в т. 5.6.13 , обектът Textarea притежава специфични манипулатори на събития, атрибути и методи, повечето от които са със същото значение както за обекта Text.

Специфични манипулатори на събития; onChange, onSelect, onKeyDown, onKeyPress, onKeyUp – със същото значение както за обект Text.

Специфични атрибути на обект Textarea:

defaultValue	със същото значение както за обект Text.
rows	Задава и връща броя редове в текстовата кутия. Може да се променя по всяко време.
cols	Задава и връща броя символи на ред в текстовата кутия. Може да се променя по всяко време.

Специфични методи на обект Textarea

Метод select() - със същото значение както за обект Text.

21.7.17 Обект за маркиране Checkbox

Обектът Checkbox представя елемент за маркиране в HTML формуляр. Обектът се създава от етикет <INPUT> с атрибут type=”CHECKBOX”. Атрибута value на обекта Checkbox задава и връща символния низ, асоцииран с обекта. При начално зареждане на HTML документа в атрибута value се записва съдържанието на атрибута VALUE на етикета<INPUT>. . Освен общите характеристики на обектите на формулярите, разгледани в т. 5.6.13 , обектът Checkbox притежава следните специфични манипулатори на събития, атрибути и методи:

Специфични манипулатори на събития: onChange, onSelect, onKeyDown, onKeyPress, onKeyUp – със същото значение както за обект Text.

Специфични атрибути на обект Checkbox:

defaultChecked	Задава и връща логическа стойност, съответстваща на началното състояние на елемента за маркиране след зареждането на HTML документа – true ако е маркиран и false, ако не е маркиран.
checked	Задава и връща логическа стойност, съответстваща на текущото състояние на елемента за маркиране.

Специфични методи на обект Checkbox

click()	Симулира щракане с мишката върху обекта Checkbox, но без да се генерира събитие Click.

Да разгледаме един пример

```
<HTML><HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<SCRIPT language="JavaScript">
function show_status(){
    document.F1.T1.value=document.F1.C1.checked;
}
</SCRIPT></HEAD>
<BODY>
<center>При щракане с мишката в обекта Checkbox, в текстовата кутия се извежда състоянието му</center><HR>
<FORM name="F1"><pre>
Checkbox <INPUT type="checkbox" name="C1" OnClick="show_status()"><BR>
Status <INPUT type="text" name="T1">
</FORM></BODY></HTML>
```

Пример 135 Прочитане на състоянието на обект за маркиране Checkbox.

В примера в на манипулатора OnClick на обект Checkbox се присвоява функцията show_status(). В резултат при щракване с мишката в обекта Checkbox се изпълнява инструкцията от функция show_status() , която извежда в текстовата кутия състоянието на обекта Checkbox – true, ако е маркиран и false в противен случай.

21.7.18 Маркиране на елемент от група - обект Radio

Обектът Radio представя елемент за маркиране на обект от група в HTML формуляр. Обектът се създава от етикет <INPUT> с атрибут type=”RADIO”. Няколко такива обекта образуват група ако имат едно и също име - една и съща стойност на атрибута name. В даден момент само един обект от групата може да бъде маркиран, останалите се изключват автоматично. Атрибута value на обекта Radio задава и връща символния низ, асоцииран с обекта, който е маркиран.

Обектът Radio притежава същите манипулатори, атрибути и методи със същото значение както обекта Checkbox.

Да разгледаме един пример:

```
<HTML><HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<SCRIPT language="JavaScript">
function show_status(obj){
```

```
        document.F1.Ò1.value=obj.value;
    }
    </SCRIPT></HEAD>
    <BODY>
    <center><H1>Кой плод обичате повече ?</H1></center><HR>
    <FORM name="F1"><OL>
    <LI><INPUT    type="RADIO"    name="Fr"    OnClick="show_status(this)"
value=
    "Портокали "> Портокали
    <LI><INPUT    type="RADIO"    name="Fr"    OnClick="show_status(this)"
value=
    "Банани "> Банани
    </OL>
    Вие избрахте <INPUT type="text" name="Ò1">
    </FORM></BODY></HTML>
```

Пример 136 Прочитане на стойност от маркиран обект Radio

Във формуляра, включен в HTML документа от примера се съдържа номериран списък с два обекта от тип Radio, които образуват група, защото имат едно и също име (стойност на атрибут name). При щракване с мишката върху някой от обектите той се маркира, генерира се събитие Click и се изпълнява функцията show_status , като и се предава обектната променлива this, която е синоним на текущия обект. Във функцията show_status се записва в текстовата кутия T1 стойността на обекта Radio, върху който е щракнал оператора.

21.7.19 Меню/Списък от опции - обект Select

Обектът Select представя падащо меню или списък в HTML формуляр. Обектът се създава от контейнерния етикет <SELECT>,, в който за всеки ред от менюто/списъка се влага по един етикет <OPTION>…</OPTION>. Обектът Select предоставя манипулатори на събития, атрибути и методи за операции със списъка и неговите елементи.

Специфични манипулатори на събития на обект Select

onChange	Предизвиква изпълнение на код на JavaScript когато обектът Select загуби фокус и съдържанието му е било променено, т.е бил е селектиран друг елемент от списъка.

Специфични атрибути на обект Select

selectedIndex	Задава и връща целочислена стойност – индекс на селектирания елемент от списъка. Може да се променя по всяко време. Началната стойност на индекса е 0, но ако няма селектиран елемент от списъка атрибутът selectedIndex връща стойност -1.
options	Връща масив, който съдържа обекти от тип Option за всеки елемент от списъка. Масивът е достъпен само за четене, което означава, че не може програмно да се променя съдържанието на списъка от елементи на обект Select*.
length	(само за четене) Връща целочислена стойност – брой елементи в масива options.

21.7.20 Елемент от меню/списък от опции - обект Option.

Обектът Option представя елемент от падащо меню или списък в HTML формуляр. Обектът се създава от контейнерния етикет < OPTION >, който се влага за всеки ред от менюто/списъка. Достъп до обектите Option може да се осъществи посредством масива options на съответния обект Select. Обектът Option предоставя набор от атрибути на елементите от списъка.

Атрибути на обект Option

Атрибут	Описание
index	(само за четене) Връща целочислена стойност – индекс на елемента в масива options на обекта Select. Началната стойност на индексите е 0 (zero-bazed index).
length	(само за четене) Връща целочислена стойност – брой елементи в масива options на обекта Select..
defaultSelected	Задава и връща логическа стойност, съответстваща на началното състояние на елемента Option в списъка – true ако е селектиран и false – ако ене е селектиран. Може да се променя по всяко време.
selected	Задава и връща логическа стойност, съответстваща на текущото състояние на елемента Option в списъка – true ако е селектиран и false – ако ене е селектиран. Може да се променя по всяко време.
text	Задава и връща символния низ, съдържащ се в етикета

* Списъкът може да бъде променян със средствата на динамичния HTML, но тогава в действителност се заменя един списък с друг.

	OPTION. Може да бъде променян по всяко време.
value	Задава и връща символния низ, който се изпраща от формуляра като двойка име-стойност, заедно с името на елемента SELECT. Ако елемента OPTION съдържа атрибут VALUE, началната стойност на value е съдържанието на този атрибут. Ако атрибут VALUE липсва, началната стойност на value е празен низ.

Да разгледаме един пример за използването на обектите Select и Option.

```
<HTML><HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<SCRIPT language="JavaScript">
function show_selected(){
    var n=document.F1.fruits.selectedIndex;
    document.F1.T1.value=document.F1.fruits.options[n].text +
    " ("+document.F1.fruits.options[n].value + " )";
}
</SCRIPT></HEAD>
<BODY>
<center><H1>Кой плод обичате повече ?</H1></center><HR>
<FORM name="F1">
<SELECT name="fruits" size="2" OnClick="show_selected()">
<OPTION value="Apples"> Портокали </OPTION>
<OPTION value="Bananas"> Банани </OPTION>
</SELECT>
Вие избрахте <INPUT type="text" name="T1">
</FORM></BODY></HTML>
```

Пример 137 Във формуляра, включен в HTML документа от примера се съдържа етикета <SELECT>, който извежда списък с два реда с текст “Портокали” и “ Банани ” , и стойности на атрибута value на обектите Option съответно ="Apples" и ="Bananas". При всяко щракване с мишката върху списъка, включително и при селектиране на елемент, се изпълнява функцията show_selected(), която извежда в текстовата кутия T1 стойностите на атрибута text и атрибута value на избрания елемент от списъка. Интексът на избрания елемент от списъка се получава от атрибута selectedIndex на обекта Select.

21.7.21 Избор на файл - обект FileUpload.

Обектът FileUpload представя елемент за избор на файл в HTML формуляр. Обектът се създава от етикет <INPUT> с атрибут type=”FILE”. Този елемент на формулярите дава възможност да се изпрати към програма на WEB сървъра съдържанието на файл на потребителския компютър. Елементът се извежда като текстова кутия и бутон с надпис “Browse”, при щракване върху който се извежда стандартната диалогова кутия за избор на файл, а при затварянето и с бутона “OK” в текстовата кутия се записва спецификацията на избрания файл. Следователно обектът FileUpload осъществява достъп до файловата система на потребителския компютър, но са взети мерки да не може да се злоупотребява с това. Обектът FileUpload предоставя манипулатори на събития, атрибути и методи за операции със съдържанието на текстовата кутия на елемента.

Специфични манипулатори на събития на обект FileUpload

onChange	Предизвиква изпълнение на код на JavaScript когато обектът FileUpload загуби фокус и съдържанието му е било променено.

Специфични атрибути на обект FileUpload

value	(само за четене) Връща символен низ със съдържанието на текстовата кутия. Ограничението “само за четене” е наложено от съображения за сигурност. Със същата цел е наложено и допълнително ограничение – HTML елемента “FILE” , чрез който се създава обекта, няма атрибут “VALUE”, така че в текстовата кутия при зареждане на HTML документа няма начално съдържание – спецификацията на файла може да бъде избрана или въведена само ръчно от оператора.

Специфични методи на обект FileUpload.

select()	Селектира съдържанието на текстовата кутия. След изпълнението на метода целия текст в текстовата кутия се извежда с инверсен цвят. Такова селектиране е полезно тогава, когато се очаква, че операторът ще копира или ще замени изцяло текста.

Полезно е да не забравяме и това, че използването на елемент <INPUT> с атрибут <FILE> във формуляр изисква да се зададе на атрибута enctype на етикет <FORM> стойност “multipart/form-data”.

21.7.22 Бутони във формулярите - обекти Button, Submit и Reset.

Обектите Button, Submit и Reset представят бутони в HTML формуляр. Те се създава от етикет <INPUT> с атрибути type съответно =" BUTTON", "SUBMIT" и "RESET". Обектите Button, Submit и Reset притежават общите за всички обекти на формулярите атрибути, методи и събития. Бутонът "SUBMIT" служи за изпращане на данни от формуляр към интернет приложение на WEB сървър и в съответствие с това обектът Submit има метод submit(), който програмно предизвиква изпращане на данни от формуляра, но без да генерира събитие "Submit". Бутонът "RESET" служи за инициализиране на елементите на формуляра, и в съответствие с това обектът Reset има метод reset(), който програмно предизвиква инициализиране на обектите във формуляра.

Специфични методи на обектите Button, Submit и Reset.

click()	Симулира щракане с мишката върху обекта. Резултата от изпълнението на метода зависи от типа на обекта. И за трите обекта методът активира манипулатора на събитие OnClick. Освен това за обект Submit се предизвиква изпращане на данните от формуляра, а за обект Reset – инициализиране на елементите във формуляра.

22 Глава III – Динамичен HTML

23 Какво представлява динамичният HTML

Динамичният HTML (DHTML) е наименование, с което се обозначават разширенията на обектния модел на документа (DOM), които предоставят съвременните WEB браузъри за динамично променяне на HTML документите. Тези разширения дават възможност за:

- ❖ Динамично променяне на съдържанието.
- ❖ Динамично променяне на стила на елементите.
- ❖ Динамично позициониране.

Разширенията на DOM бяха стандартизирани от W3C* и включени в спецификацията на HTML 4.0, която се появи през септември 1998г. В тази глава ще разгледаме тези разширения и възможностите, които те ни дават за динамично променяне на HTML документите.

24 Достъп до обектите в HTML документ посредством идентификатор.

Основното средство за достъп до обект в HTML документ, стандартизирано от W3C в DOM е методът getElementById на обект document. Той връща обектна променлива (референция) за обект, създаден чрез HTML етикет със зададена уникална! стойност на атрибута id. Ако в документа липсва елемент със зададения идентификатор методът връща стойност null.

Синтаксис:

```
element = document.getElementById(element_id);
```

където

- element е обектна променлива за обекта, създаден от HTML елемент с идентификатор id.
- element_id е символен низ, съдържащ уникалния идентификатор на HTML елемента

Да разгледаме един пример.

```
<HTML><HEAD>
<script type="text/javascript">
function show_value(){
```

* World Wide Web Consortium

! т.е. неповтаряща се в рамките на HTML документа

```
    elem = document.getElementById("t1");
    alert(elem.value);
}
</script>
</HEAD>
<BODY>
<form>
<input id="t1" value="Some text here"><BR>
<input type="button" value="Show text" onclick=" show_value()">
</form>
</BODY></HTML>
```

Пример 138 Използване на метода getElementById

В примера при щракване с мишката върху бутона с надпис "Show text" се изпълнява функцията show_value(). При изпълнение на метода getElementById("t1") се получава обектна променлива за текстовата кутия и посредством диалогова кутия alert се извежда съдържанието на атрибута value – т.е. текста, който се съдържа в нея.

В случая достъпът до текстовата кутия посредством идентификатор не е обязателно необходим. Същия резултат може да се получи и посредством името на текстовата кутия (т.е. с атрибута name), например с инструкцията

```
alert(document.F1.T1.value)
```

В други случаи обаче, когато искаме да променяме съдържанието, стила или позиционирането на елементи в HTML документа, е необходимо да се получи достъп до елемента обязателно чрез неговия идентификатор id, което изисква използването на метода getElementById.

25 Динамично променяне на съдържанието на HTML документ.

За достъп до съдържанието на HTML елементи и динамичното му променяне Internet Explorer предоставя атрибутите innerText, innerHTML, outerText и outerHTML, а също и два метода, които дават възможност за добавяне на текст и HTML код към съдържанието на HTML документите – методите insertAdjacentText и insertAdjacentHTML. Тези атрибути и методи се прилагат към обектите, които се създават от браузъра при обработката на HTML документа. За съжаление тези атрибути и методи не са стандартизирани от W3C. Тестовете с Netscape Navigator 7.0 показват, че от изброените атрибути и методи той поддържа само атрибута innerHTML.

25.1Променяне на HTML елемент – атрибут innerHTML

Атрибутът innerHTML може да се прилага към всички HTML етикети с изключение на <HTML>, <HEAD> и <TITLE>. Той задава и връща текста между отварящата и затварящата част на етикета на текущия елемент. При задаване на стойност на атрибута, символният низ се обработва като HTML код.

Както се вижда от описанието, атрибутът заменят само съдържанието на HTML етикетите. Присвоената стойност се обработва като HTML код. Атрибутът може да се използва едва след като HTML документа е изцяло зареден в браузъра. Като индикатор за пълното зареждане на документа може да се използва събитието Load (с манипулатор *onload*) на обект window.

Да разгледаме един пример:

```
<HTML><HEAD>
<script type="text/javascript">
var text="<input type='text' value='Hello'>";

function inner_HTML(){
    document.getElementById('d').innerHTML=text;
}
</script></HEAD>
<BODY>
<div id="d" style="border:1px solid;background-color:#ffff00"> </div>
<p><a href="javascript:inner_HTML()">Test innerHTML</a></p>
</BODY></HTML>
```

Пример 139 Променяне на съдържанието на HTML документ с атрибут innerHTML

В примера в кода на HTML е включен слой с идентификатор id="d" и хипервръзка, с която се стартира функцията за променяне на съдържанието на HTML документа посредством атрибута innerHTML. Функцията записва в съответния атрибут символния низ "<input type='text' value='Hello'>" , който съдържа HTML код на текстова кутия. При щракване с мишката върху хипервръзката “Test innerHTML” в слоя се появява текстова кутия, тъй като низа се обработва като HTML код.

Атрибутът innerHTML ни дава пълен програмен контрол върху съдържанието на HTML документа. Ако той се приложи за етикета <BODY>, то чрез него може да се замени цялото съдържание на HTML документа (без заглавната част) с ново. Например инструкцията

```
document.body.innerHTML="<input type='text' value='Hello'>";
```

Пример 140 Заменяне на цялото съдържание на HTML документ с текстова кутия.

ще замени цялото съдържание на етикета <BODY> на HTML документ с кода на текстова кутия. Като резултат в прозореца на браузъра ще се изобрази само текстовата кутия.

26 Динамично променяне на стила на HTML елементи – обект Style.

Обектът Style представя стила на HTML елемент. Обектът document и всеки елемент в HTML документа, който може да се форматира със стил, притежават атрибут style, който връща обект от тип Style за съответния HTML елемент. Обектът Style предоставя набор от атрибути, посредством които могат програмно да се управлява форматирането (шрифтове, цветове, интервали, размери, връзки и др.) на HTML елемента. Същия резултат можем да получим и чрез включване на дефиниция на стил в HTML елемента или с прилагане на външно дефиниран стил. Обекта Style обаче ни дава възможност да правим промените динамично – при възникване на някакво събитие след като HTML документа вече е бил зареден.

За присвояване на стойност на атрибут на стила на елемент с идентификатор “id” можем да използваме следния синтаксис:

```
document.getElementById("id").style.property="value"
```

където
id е идентификатор на HTML елемента
property е име на атрибута на обект Style
value е стойността, която му се присвоява.

Обекта Style има много атрибути, с които могат да се задават различни елементи на стила. Важно е да се запомни, че имената им не съвпадат с имената на атрибутите, които се задават в CSS дефинициите. Например в CSS дефиниция на стил цветът на фона се задава с атрибут *background-color*, а съответният атрибут на обект Style е с наименование *backgroundColor*. По-долу ще изброим най-често използваните атрибути на обект Style, а по-пълна изформация можете да намерите на адрес http://www.w3schools.com/htmldom/dom_obj_style.asp.

Атрибут	Описание

backgroundColor	Задава цвета на фона на HTML елемента.
backgroundImage	Задава спецификация (URL) на файл с фоново изображение за an HTML елемента.
borderColor	Задава цвета на рамката на HTML елемента
borderStyle	Задава стила на рамката на HTML елемента, напр. “solid за рамка с плътна линия ” или “double” за рамка с двойна линия.
color	Задава цвета на текста в HTML елемента.
fontFamily	Задава вида на шрифта за текста в HTML елемента, напр. “Arial” или “Courier”.
fontSize	Задава размера на шрифта за текста в HTML елемента в пиксели, напр. “14px”.
fontStyle	Задава стила на шрифта за текста в HTML елемента, напр. “bold” или “italic”.
textAlign	Задава подравняването на текста в HTML елемента, напр. “left” или “justify”.
textDecoration	Задава специални ефекти за текста в HTML елемента, напр. “strike” (зачертан), underline(подчертан), blink(мигащ) и др.
textIndent	Задава отместване на първия ред от текста в HTML елемента в пиксели, напр. “20px”.
letterSpacing	Задава разстояние между символите в пиксели. Възможни стойности: normal (за нормално разстояние между символите) или числова стойност. Допустими са и отрицателни стойности. Пример: document.getElementById("p1").style.letterSpacing="3";

Атрибутите, свързани с позиционирането на елемент в HTML документа са разгледани в следващата точка.

Да разгледаме един пример:

```
<HTML><HEAD>
<script type="text/javascript">
function changeBackground(){
    document.getElementById('d1').style.backgroundColor="#00ff00";
}
function changeFontSize(){
    document.getElementById('d1').style.fontSize="40px";
}
</script></HEAD>
<BODY>
```

```
<div style="border:1px solid;" id="d1">Hi</div>
<OL>
<LI><a href="javascript:changeBackground()">Change Background</a>
<LI><a href="javascript:changeFontSize()">Change FontSize</a>
</OL>
</BODY></HTML>
```

Пример 141 Променяне на фонов цвят и размер на шрифта на слой посредством атрибути на обекта Style.

В примера в тялото на HTML документа е включен слой с идентификатор id="d1" и две хипервръзки, чрез които се стартират функцията changeBackground() за променяне на фоновия цвят на слоя и функцията changeFontSize() за променяне на размера на шрифта в слоя. Двете функции са включени в блок <SCRIPT> в заглавната част на HTML документа. Достъпът до обекта на слоя се осъществява посредством метода getElementById на обект document. Атрибута style на обекта връща асоциирания с него обект от тип Style и инструкцията присвоява стойност на съответния атрибут (backgroundColor за фонов цвят и fontSize за размер на шрифта).

Съвсем не е задължително достъпът до обекта да се осъществява обязателно посредством метода getElementById на обект document. Ако например кода за промяна на стила се изпълнява чрез събитийна процедура на самия обект, чийто стил ще се променя, то тогава достъпът до обекта може да се осъществи посредством обектната променлива this, която е синоним на текущия обект (т.е. представлява референция към него). Да разгледаме един такъв пример:

```
<HTML><HEAD>
<meta http-equiv="Content-type" content="text/html, charset=wondows-1251">
<script type="text/javascript">
var text="<input value='Hello'>";
function yellow(obj){
    obj.style.backgroundColor='#ffff80';
}
function white(obj){
    obj.style.backgroundColor='#ffffff';
}
</script></HEAD>
<BODY>
<center><h1> Променяне на фоновия цвят на елемент</h1><br>
Щракнете с мишката в някоя от текстовите кутии</center><hr>
<form name="F1">
```

```
<br><input name="T1" onFocus="yellow(this)" onBlur="white(this)">
<br><input name="T2" onFocus="yellow(this)" onBlur="white(this)">
</form>
</BODY></HTML>
```

Пример 142 Променяне на фоновия цвят на текстова кутия

В примера в тялото на HTML документа е включен формуляр, който съдържа две текстови кутии T1 и T2. За всяка от текстовите кутии посредством манипулатор onFocus когато текстовата кутия получи фокус се извиква функцията yellow, която задава жълт фонов цвят на текстовата кутия. Посредством манипулатор onBlur когато текстовата кутия загуби фокус се извиква функцията white, която възстановява белия фонов цвят на текстовата кутия. И двете функции получават при извикването си като параметър обектната променлива this – синоним на текущия обект.

27 Динамично позициониране на HTML елементи чрез обект Style.

Обект Style предоставя набор от атрибути, които дават възможност за програмно променяне на позицията и размерите на HTML елементи след като HTML документа е бил зареден. Важно е да се запомни, че динамично позициониране е възможно само за елементи, които още при първоначалното си зареждане са били позиционирани с използване на стил. Основните атрибути за позициониране и оразмеряване са следните:

Атрибут	Описание
left	Задава разстоянието по оста X от външния (контейнерен) елемент до лявата граница на позиционирания елемент.
top	Задава разстоянието по оста Y от външния (контейнерен) елемент до горната граница на позиционирания елемент.
width	Задава дължината на елемента
height	Задава височината на елемента.
zIndex	Задава позицията на HTML елемента по оста Z – как се подреждат елементите един над друг. По-голямата стойност означава по-горна позиция. Най-ниска позиция се задава със стойност -1.

Да разгледаме един пример:

```
<HTML><HEAD>
```

```
  <meta http-equiv="Content-type" content="text/html, charset=windows-1251">
  <script type="text/javascript">
  var text="<input value='Hello'>";
  function moveLayer(x,y){
      document.getElementById('d1').style.left=x;
      document.getElementById('d1').style.top=y;
  }
  </script></HEAD>
  <BODY>
  <div id="d1" style="border:1px solid;position:absolute;top:0;left:0">
  Преместване на слоя чрез обект Style</div>
  <br><OL>
  <LI><a href="javascript:moveLayer(100,150)">Move Down</a>
  <LI><a href="javascript:moveLayer(0,0)">Return Back</a>
  </OL>
  </BODY></HTML>
```

Пример 143 Позициониране на слой чрез обект Style

В примера в тялото на HTML документа е включен слой с идентификатор id="d1" и две хипервръзки, чрез които се стартира функцията moveLayer за за преместване на слоя с параметри, които задават координатите на горния ляв ъгъл на слоя. Функцията е включена в блок <SCRIPT> в заглавната част на HTML документа. Достъпът до обекта на слоя се осъществява посредством метода getElementById на обект document. Атрибута style на обекта връща асоциирания с него обект от тип Style и инструкциите присвояват стойности на атрибутите left и top, чрез който се задава позицията на слоя.

Свързан с позиционирането е и атрибутът visibility, с който се управлява видимостта на HTML елемент или слой с възможни стойности “visible” за видим слой, “hidden” за невидим слой и “ inherit” за слой, който наследява видимостта си от външния (родителски) слой, в който е вложен. Посредством атрибутите zIndex и visibility слоевете могат да се наслагват един върху друг и се задава кои от тях да са видими в даден момент. Освен това слоевете могат да се влагат един в друг и при скриването на външния слой се скриват и вложените в него слоеве, ако на атрибута им visibility е зададена стойност “ inherit”.

28 Динамично променяне на HTML таблици - обекти Table, TableRow и TableCell.

Обектите Table, TableRow и TableCell предоставят атрибути и методи, които дават възможност за променяне не само на форматирането на таблицата, но и променяне на нейната структура – добавяне и изтриване на редове, добавяне и изтриване на клетки в зададен ред, добавяне на заглавна част и др. При практическото използване на тези обекти е необходимо да се използват възмохностите, които ни дават DHTML и DOM за динамично променяне на съдържанието, за да може да се добави съдържание към добавените елементи на таблицата.

28.1Обект Table.

Обекта Table притежава набор от атрибути, които съответстват на атрибутите на етикет <TABLE>. Атрибутите са дадени в таблицата по-долу, с кратко описание на значението им. По подробното им описание е дадено в т. 12.2 от първа глава.

Атрибути, представящи атрибути на <TABLE>

Атрибут	Описание
border	Задава и връща ширината на рамката в пиксели.
caption	Връща символен низ, съдържащ заглавния текст на таблицата
cellPadding	Задава и връща разстоянието в пиксели от рамката на клетка до нейното съдържание.
cellSpacing	Задава и връща разстоянието в пиксели между рамките на клетките.
frame	Задава вида на външната рамка на таблицата. Допустими стойности “void” “above”, “below”, “hsides”, ”vsides”, “lhs”, ”rhs”, “box”, “border”.
width	Задава и връща ширината на таблицата в пиксели или в проценти.

Атрибути, представящи обекти на таблиците.

Атрибут	Описание
rows	Връща колекция от обекти TableRow, представящи редовете в таблицата.
rules	Задава вида на вътрешната рамка на таблицата. Допустими стойности “none”, “rows”, “cols”, “all”.
summary	Задава и връща символен низ, съдържащ описание на таблицата. По подразбиране съдържа празен низ.
tBodies	Връща колекция от обекти TBODIE, представящи секциите <TBODIE> на таблицата.
tFoot	Връща обект TFOOT, представящ секцията <TFOOT> на таблицата.

tHead	Връща обект THEAD, представящ секцията <THEAD> на таблицата.

Метод	Описание
createCaption()	Създава и връща обект HTMLTableElement, представящ етикета <CAPTION> на таблицата. Ако таблицата вече притежава елемент CAPTION, методът само връща обекта.
createTFoot()	Създава и връща обект HTMLTableElement, представящ секция <TFOOT> на таблицата. Ако таблицата вече притежава секция <TFOOT>, методът само връща обекта.
createTHead()	Създава и връща обект HTMLTableElement, представящ секция <THEAD> на таблицата. Ако таблицата вече притежава секция <THEAD>, методът само връща обекта.
deleteCaption()	Изтрива елемента CAPTION на таблицата.
deleteTFoot()	Изтрива секцията TFOOT на таблицата.
deleteTHead()	Изтрива секцията THEAD на таблицата.
insertRow(index)	Добавя към таблицата ред със зададен индекс и връща обект TableRow, представящ добавения последен елемент. Параметърът *index* трябва да има стойност от 0 до rows.lenght т.е. действителния индекс, който ще получи реда, ако се добави вътре или в края на таблицата. При стойност на *index* извън интервала [0÷rows.length] се генерира грешка “Invalid argument”.
deleteRow(index)	Изтрива от таблицата ред със зададен индекс. Параметърът *index* трябва да има стойност от 0 до rows.lenght-1. В противен случай се генерира грешка “Invalid argument”.

Да разгледаме един пример:

```
<HTML><HEAD>
<script type="text/javascript">
function insRow()
{
var tb=document.getElementById('myTable');
var x=tb.insertRow(tb.rows.length); //добавяне на ред в края на таблицата
var y=x.insertCell(0); //добавяне на клетка с индекс 0
var z=x.insertCell(1); //добавяне на клетка с индекс 1
y.innerHTML="New Cell1"; //добавяне на текст клетка с индекс 0.
```

```
z.innerHTML=" New Cell2"; //добавяне на текст в клетка с индекс 1.
}
function delRow()
{
var tb=document.getElementById('myTable');
var x=tb.deleteRow(tb.rows.length-1); //изтриване на последния ред
}
</script></HEAD>
<BODY>
<table id="myTable" border="1">
<tr><td>Row1 cell1</td><td>Row1 cell2</td></tr>
</table>
<BR><a href="javascript:insRow()">Insert Row</a>
<BR><a href="javascript:delRow()">Delete Row</a>
</BODY></HTML>
```

Пример 144 Добавяне и изтриване на редове в таблица посредством методите insertRow и deleteRow.

В примера в тялото на HTML документа е включена таблица с идентификатор id="myTable" и две хипервръзки, с които се стартират функцията за добавяне на ред към таблицата insRow() и функцията за изтриване на ред от таблицата delRow(). Обектът от тип Table за таблицата се получава чрез метода getElementById на обект document. Добавянето на ред става чрез метода insertRow на обект Table, а изтриването на ред – чрез метода deleteRow на обект Table. След добавянето на ред методът insertRow връща обект TableRow за реда и след това с метод insertCell към реда се добавят две клетки, а чрез атрибута innerHTML се добавя текст към клетките.

28.2 Обект TableRow.

Обектът TableRow представя ред от таблицата. TableRow притежава набор от атрибути и предоставя два метода, чрез които се добавят и премахват клетки на реда от таблицата.

Атрибути на TableRow

Атрибут	Описание
align	Задава хоризонтално подравняване на съдържанието на клетките в реда. Допустими стойности – “left”, “right” и “center”.
vAlign	Задава вертикално подравняване на съдържанието на клетките в реда. Допустими стойности – “top”, “middle”

	и “bottom”.
cells	Връща колекция от обекти TableCell, представящи клетките в реда.
rowIndex	Връща индекса на реда в таблицата.
sectionRowIndex	Връща индекса на реда в текущата секция thead, tfoot, или tbody

Методи на обект TableRow

Атрибут	Описание
insertCell(index)	Добавя към реда клетка със зададен индекс и връща обект TableCell, представящ добавения последен елемент. Параметърът *index* трябва да има стойност от 0 до cells.lenght т.е. действителния индекс, който ще получи клетката, ако се добави вътре или в края на реда. При стойност на *index* извън интервала [0÷cells.length] се генерира грешка “Invalid argument”.
deleteCell(index)	Изтрива от реда клетка със зададен индекс. Параметърът *index* трябва да има стойност от 0 до cells.lenght-1. В противен случай се генерира грешка “Invalid argument”.

В примера, даден по-долу се използва методът insertCell за добавяне на клетки към ред от таблицата.

```
<HTML><HEAD>
<script type="text/javascript">
function insCell()
{
var tb=document.getElementById('myTable');
var row=tb.rows[0];
var cell=row.insertCell(row.cells.length);
cell.innerHTML="New Cell "+row.cells.length
}
function delCell()
{
var tb=document.getElementById('myTable');
var row=tb.rows[0];
row.deleteCell(tb.cells.length-1);
}
</script></HEAD>
<BODY>
<table id="myTable" border="1">
<tr><td>Cell 1</td></tr>
```

```
</table>
<BR><a href="javascript:insCell()">Insert Cell</a>
<BR><a href="javascript:delCell()">Delete Cell</a>
</BODY></HTML>
```

Пример 145 Добавяне и изтриване на клетки в ред от таблица посредством методите insertCell и deleteCell.

В примера в тялото на HTML документа е включена таблица с идентификатор id="myTable" и две хипервръзки, с които се стартират функцията insCell() за добавяне на клетка към първия ред от таблицата и функцията delCell() за изтриване на клетка първия ред на таблицата. Обектът от тип Table за таблицата се получава чрез метода getElementById на обект document. Обектът от тип TableRow за първия ред се получава от колекцията rows на TableRow. Добавянето на клетка става чрез метода insertCell на обект TableRow, а изтриването на ред – чрез метода deleteCell на обект TableRow. След добавянето на клетка методът insertCell връща обект TableCell за клетката и след това чрез атрибута innerHTML на обекта TableCell се добавя текст към клетката.

28.3Обект TableCell.

Обектът TableCell представя клетка от HTML таблица. За всеки етикет (таг) <td> от ред в таблицата се създава обект TableCell и тези обекти се записват в колекцията cells на обекта TableRow. Обектът TableCell притежава набор от атрибути за достъп до параметри на клетка от таблицата.

Атрибути на TableCell

Атрибут	Описание
align	Задава хоризонтално подравняване на съдържанието на клетката. Допустими стойности – “left”, “right” и “center”.
vAlign	Задава вертикално подравняване на съдържанието на клетката. Допустими стойности – “top”, “middle” и “bottom”.
cellIndex	Връща индекса на клетката в реда.
rowSpan	Задава и връща символен низ с броя на редовете от таблицата, които заема клетката.
colSpan	Задава и връща символен низ с броя на колоните от таблицата, които заема клетката.

Да разгледаме един пример, който демонстрира използването на атрибута colSpan.

```
<HTML><head>
<script type="text/javascript">
function setColSpan()
{
var x=document.getElementById('myTable').rows[0].cells
 if(x[0].colSpan!="2"){
     x[0].colSpan="2"; x[1].colSpan="6";
 }else{
     x[0].colSpan="4"; x[1].colSpan="4";
 }
}
</script></head>
<body>
<table id="myTable" border="1">
<tr>
<td colspan="4">A cell</td><td colspan="4">A cell</td>
</tr>
<tr>
<td>A cell</td><td>A cell</td><td>A cell</td><td>A cell</td>
<td>A cell</td><td>A cell</td><td>A cell</td><td>A cell</td>
</tr>
</table>
<A href=”javascript: setColSpan()" > Change Colspan</A>
</body></html>
```

В примера в тялото на HTML документа е включена таблица с два реда, като първият ред съдържа само две клетки, всяка от които заема по четири колони от таблицата. Чрез хипервръзка се стартира функцията setColSpan, която променя стойностите на атрибута Colspan на двете клетки от първия ред. Достъпът до таблицата се осъществява посредством метода getElementById с параметър идентификатора на таблицата. Достъпът до първия ред и клетките в него се осъществява посредством колекциите rows и cols.

Информация и учебни материали в Интернет по темата:

http://www.w3schools.com/dhtml/dhtml_dom_table.asp – обект Table
http://www.w3schools.com/dhtml/dhtml_dom_tablerow.asp – обект TableRow
http://www.w3schools.com/dhtml/dhtml_dom_tablecell.asp – обект TableCell

29 Мултимедийни възможности на HTML5

HTML5 предоставя разширени възможности за операции с мултимедийните средства на компютъра. Мобилните устройства предоставят на интернет приложенията чрез API достъп до своите вградени мултимедийни устройства (камера и микрофон), които могат да се използват чрез HTML5. Засега тези възможности отсъстват в "класическите" компютри. По-долу ще бъдат разгледани два нови HTML елемента, включени в HTML5, чрез които в HTML документа, зареден във Web браузъра могат да се възпроизвеждат аудио и видео файлове. Тези HTML елементи се поддържат от новите версии на основните браузъри и могат да заместят използваните понастоящем за тази цел външни (за браузъра) контроли.

29.1 Извеждане на видео файлове в HTML5 – елемент <video>

Елементът <video> е новото средство на HTML5, чрез което могат да се възпроизвеждат както видео така и аудио файлове. Той има два алтернативни синтаксиса:

а) С използване на атрибут src:

```
<video src="URL" attribute=”value” ...>
</video>
```

където с атрибута src се задава интернет адреса на медийния файл

или

```
<video attribute=”value” ...>
  <source src="URL" type='MIME-type; codecs="list_of_codecs" ' />
  ……
</video>
```

където медийните файлове (един или повече), които се възпроизвеждат, се задават с етикети <source>.

Както се вижда, вторият синтаксис дава повече възможности, тъй като може да се зададе едновременното възпроизвеждане на видео и аудио файл и чрез атрибута type да се избере типа на файла и вида на използваните кодеци чрез атрибута codecs. Да разгледаме един пример:

```
<video>
  <source src="movie.ogg"  type='video/ogg;  codecs="theora, vorbis"'>
```

```
    HTML5 video is not supported in this browser
  </video>
```

Пример 146 Възпроизвеждане на видео файл с HTML5 елемент <video>

В примера в елемента <video> е включен етикета <source> с които се задава за възпроизвеждане мултимедиен файл movie.ogg .

Атрибути на елемента <video>

Autoplay	Задава автоматично стартиране на възпроизвеждането при зареждане на HTML документа.
Autobuffer	Задава автоматично изтегляне на мултимедийния файл в буфер след зареждане на HTML документа, с което се гарантира последващо възпроизвеждане. Този атрибут не се използва заедно с Autoplay.
Poster=”Url”	Задава адрес на изображение, което да се извежда преди да започне възпроизвеждането на мултимедийния файл.

https://developer.mozilla.org/en-US/docs/HTML/Element/video
http://www.dvdvideosoft.com/products/dvd/Free-HTML5-Video-Player-And-Converter.htm Free HTML5 Video Player and Converter
http://ru.wikipedia.org/wiki/Ogg Ogg
http://html5doctor.com/the-video-element/ The video element
http://www.w3schools.com/html/html5_video.asp HTML5 Video
http://www.html5rocks.com/en/tutorials/video/basics/ HTML5 Video
http://www.w3.org/TR/2008/WD-html5-20080122/#video The video element
http://www.dvdvideosoft.com/free-dvd-video-software.htm Free Studio
http://miketaylr.com/code/input-type-attr.html HTML5 inputs and attribute support

Литература

1. Колектив на НИСОФТ., Как да направим WEB страница …, “НИСОФТ”, С. , 1997.
2. Колектив на СОФТПРЕС., HTML 4 в лесни стъпки, “СОФТПРЕС”, 2000.
3. Колектив на НИСОФТ, Професионални WEB страници с JavaScript, “НИСОФТ”, С. , 1998.
4. Беноа Маршал, XML, С. “Инфодар” , Copyright © 2000.
5. Дж. Уорнър П., Вашиър, Dreamweaver 3 за начинаещи, “АлексСофт”, С., 2000.
6. Е. Ш. Уидок, Flash 5, Справочник на WEB дизайнера “АлексСофт”, С., 2001.
7. Колектив на НИСОФТ, Програмиране с JavaScript, “НИСОФТ”, С. , 2000.
8. Д. Богданов, И. Мустакеров, Език за програмиране С, С. “Техника”, 1989.
9. П. Бърнев, П. Азълов, Алгоритми, С. “Народна просвета”, 1978
10. Колектив на издателство СофтПрес, “Езикът С++”, С. “СофтПрес”, 2001.
11. Г. Гочев, Програмиране на Фортран 77, С. “Техника”, 1991.

Информация в интернет:

12. http://www.w3.org/ - сайт на WWW Consortium – Начална страница на сайта на WWW консорциум.
13. http://www.w3.org/TR/DOM-Level-2-HTML/html.html - дефиниция на DOM за HTML.
14. http://www.w3schools.com/ - Сайт с учебни материали за интернет технологии.
15. http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/ JavaScript Gide – Справочник за JavaScript.
16. http://www.javafile.com/ - Сайт с колекция от безплатни скриптове
17. http://javascript.about.com/library/bljver.htm – Сайт за JavaScript
18. http://bg.wikipedia.org - WEB енциклопедия – ключ за търсене “Прототипно программирование”
19. http://ru.wikipedia.org/wiki/ - Википедия, — свободная энциклопедия. – ключ за търсене “Прототипное программирование”
20. http://javascript.crockford.com/private.html - Private Members in JavaScript
21. http://www.thecounter.com/stats/2006/May/browser.php - Статистическа информация за използваните браузъри.
22. http://net.tutsplus.com/tutorials/html-css-techniques/html-5-and-css-3-the-techniques-youll-soon-be-using/ HTML5 Tutorial
23. http://www.w3.org/TR/html-markup/input.email.html input type=email

24. http://www.w3schools.com/html5/att_input_pattern.asp HTML5 <input> pattern Attribute
25. https://developer.mozilla.org/en-US/docs/Rich-Text_Editing_in_Mozilla Rich-Text Editing in Mozilla
26. https://blog.logrocket.com/creating-responsive-web-layouts-with-css-grid/#what-responsive-layout