Лекция 12

Програмиране във Web. Програмен език JavaScript

JavaScript е съвременен език за програмиране за World Wide Web. Той не само осигурява възможност за разработване на интерактивни Web страници, но и представлява основно средство за интегриране на Java аплети, ActiveX контроли, plug-in модули за браузери, сървърни скриптове и други.

Началото на JavaScript поставя "LiveScript" – скриптов език, разработен от Netscape. Независимо от него, Sun Microsystems разработва Java (първоначално "Oak"), като език за контрол на потребителски електронни устройства. Поради своята мощ и производителност Java намира впоследствие място в Internet за писане на малки програми, наречени "аплети".

По-нататък Netscape и Sun Microsystems обединиха своите усилия и преобразуваха LiveScript в подмножество на Java, като по-късно промениха и наименованието на езика – JavaScript.

Сега JavaScript се възприема повече като софтуерно независима платформа за разработване на Web приложения. Той може да бъде използван и за създаване на самостоятелни приложения. Като скриптов език, програмите написани на JavaScript се интерпретират, за разлика от Java на Sun, които се компилират.

В усилията си да заеме дял в пространството на Web, Microsoft разработи своя разновидност на JavaScript, която нарече JScript. В тази версия Microsoft добавя някои подобрения, във връзка с използването на своя браузър Internet Explorer.

Езиците за програмиране най-често се базират на един или няколко предходни езика, като се явяват тяхно развитие и усъвършенстване. Езикът Java, както и следващата модификация JavaScript произлиза от C.

1 Синтаксис на JavaScript

1.1 Характеристики на JavaScript

Синтаксисът на JavaScript се различава от известния скриптов език VBScript (визуална версия на Бейсик). В JavaScript операторите завършват с точка и запетая, поради което той не се нуждае от знак за продължение - докато срещне точка и запетая, интерпретаторът на JavaScript възприема всичко като част от един ред (оператор). Другата разлика е че, във VBScript една процедура може да бъде или функция, ако връща като резултат стойност, или подпрограма, ако не връща стойност. В JavaScript всяка процедура е функция, независимо от това дали връща или не връща стойност (както в език C++).

```
function Funct_Name (arg1, arg2, ..) {
     expression_1;
     expression_2;
     ……
     expression_n;  }
```

В този фрагмент от JavaScript програмен код е представена конструкция на функция. Използва се служебната дума `function`, а след нея се задава наименованието на функцията. В кръглите скоби се задават аргументите на функцията, а тялото на функцията съдържа определен брой изрази или оператори (`expression)`, оградени във фигурни скоби.

Езикът JavaScript, подобно на C++ и XML е чувствителен към регистъра на символите. Това означава, че JavaScript различава малките и главни букви и потребителя трябва да държи сметка за начина на записване на текста в програмата.

Коментарите в JavaScript са организирани, както в език C++. Едноредовите коментари започват с две наклонени черти '//' и целият текст след тези символи се разглежда като коментар (игнорира се от интерпретатора). Коментар може да бъде разположен и върху повече от един ред, но тогава се използва отваряща комбинация от символи '/*' (наклонена

черта и звездичка) и затваряща комбинация ‘*/’. Текстът, намиращ се между отварящата и затварящата комбинация символи се разглежда като коментар от JavaScript.

JavaScript е език за програмиране във Web. Затова той се явява просто допълнителен маркиран текст във Web страниците. За въвеждане на JavaScript команди във HTML документ се използва елементът (тагът) **<SCRIPT>** . Когато се започва изучаване на нов език, традиционният като първи пример се използва малка програма, която извежда на екрана поздравяващ текст “Hello World”. Ето как изглежда в JavaScript поздравяващата програма HellWorld:

```
<HTML>
<HEAD>
<TITLE> Hello World Program </TITLE>
</HEAD>
<BODY>
<SCRIPT language="JavaScript">
<!-- hide script
document.writeln("<h1>Hello World<\/h1>")
// end hiding -->
</SCRIPT>
</BODY>
</HTML>
```

Елементът **<script>** се използва за добавяне на JavaScript в HTML документ. Това се прави за да могат JavaScript съвместимите браузери да определят кой текст представлява скриптови команди и кой трябва да се изобрази директно в Web страницата. <Script> е контейнер, който използва атрибут language за задаване на скриптовия език. В последните версии на браузерите този атрибут може да не се използва, но се въвежда друг атрибут type, който задава използвания език за скриптиране.

Някои стари браузери не разпознават JavaScript и е възможно да се направи опит за неправилно интерпретиране на информацията в <SCRIPT> елемента. За да се предпазят не-JavaScript браузерите от неправилна интерпретация скриптовите команди се разполагат между отварящ и затварящ таг на елемента за коментар в език HTML:

```
<script type = text/JavaScript>
<!--
скриптови команди
//   - ->
```

В представения пример е използвана командата document.writeln (), която се нарича метод в JavaScript. Методът е функция (подпрограма) вградена в даден обект, позволявайки му да извършва определени действия. В случая методът writeln () може да ‘пише’ в обекта ‘document’. Следователно, document.writeln () е команда, която изписва някакъв текст във текущата Web страница.

Специфична форма има затварящия таг на елемента за заглавие <h1>. Той има вида <\/h1>, тъй като символът ‘/’ в комбинация с някои символи се използва като управляваща комбинация при изобразяване на текстови данни. За да се избегне неправилно тълкуване на комбинацията ‘/h’ се добавя символ ‘\’.

1.2 Служебни (резервирани) думи и променливи

Резервираните (служебни) думи са въведени за специфичните потребности на синтаксиса на езика. Част от резервираните думи са специализирани термини за JavaScript, но по-голямата част от тях са термини взаимствани от Java и C++. По тази причина, те няма да бъдат изброени тук, но в хода на изложение на материала, част от тях ще бъдат обяснени, а пълният набор може да бъде намерен във всеки справочник по JavaScript.

Наименованията на променливите в JavaScript удовлетворяват стандартните изисквания за образуване на наименования (идентификатори) в повечето програмни езици. Наименованията трябва да започват с буква от латинската азбука или знак за подчертаване. В останалата част от наименованието може да има и цифри, като е недопустимо използването на символа интервал и не е препоръчително използването на резервираните думи за наименования на променливи.

Съществена, в сравнение с голяма част от другите програмни езици, е разликата при дефиниране на променливите в JavaScript. Програмистите на JavaScript обикновено използват променливите без да ги декларират, което е допустимо от синтаксиса на езика. Това обаче, не е добра практика на програмиране. Възможността за използване на променливи, които не са декларирани е обусловено от факта, че програмният код се интерпретира (не се компилира). В този случай, интерпретаторът изпълнявайки последователно операторите от програмата, заделя място за недекларираните променливи, които се срещат за първи път и ги записва в списъка на променливите (за да не се заделя ново място при повторното им използване). Така са организирани нещата с променливите и в скриптовия език VBS. При езиците, които използват компилатори, се извършва предварително заделяне на памет за променливите (не и при динамичното резервиране на памет), което изисква и предварително деклариране на променливите.

JavaScript допуска и изрично деклариране на променливи, което се извършва с помоща на служебната дума **var** (подобно на езика Pascal).

```
var glbVar ;
var glbVar_Init = 0 ;
```

Както се вижда, възможни са два начина за използване на служебната дума **var**. При първия начин се извършва само декларация на променливата, а при втория, освен деклариране се извършва и присвояване на начална стойност (инициализация).

В една декларация могат да се декларират повече от една променлива:

```
var x,y,r ;
```

Друга особеност на скриптовия език JavaScript, е че в декларациите не е необходимо да се уточнява тип на променливите. Езикът работи с три основни типа променливи:

- **Числови променливи** - съдържат числови данни. Не се iизисква спецификация типа на числовите данни (цели, реални). Пример:

```
var numberOfStudents = 45 ;
```

- **Низови (стрингови) променливи** - съдържат текст, който се огражда в кавички. Пример:

```
var Name = "Димитър" ;
```

- **Логически (булеви) променливи** - съдържат логически стойности (ture и false).

JavaScript разглежда резервираната дума **null**, като 'празна' променлива. Типът на променливата се специфицира, когато се присвои стойност. По време на изпълнението на програма на JavaScript, на една променлива могат да се присвояват стойности от различен тип. Следователно, в JavaScript променливите могат да се разглежда като контейнери в които се съхраняват различен тип данни. Това обаче изисква, с тях да се работи с по-голямо внимание, тъй като са възможни сериозни грешки. Ето един пример:

```
var FirstVar = 10 ;
var SecondVar = 35 ;
var SumVar ;
SumVar = FirstVar + SecondVar ;
```

Резултат от изпълнението на тези оператори на JavaScript е че променливата `SumVar` получава стойност 45. Ако обаче, на променливата `SecondVar` се присвои стойност '35', т.е. `SecondVar = '35' ;`, то последният оператор ще формира стойност за променливата `SumVar` = '1035'. Това е така, защото променливата `SecondVar` се разглежда като текстов низ и съгласно синтаксиса на JavaScript, променливата `FirstVar`

трябва да се преобразува също в текстов низ (‘10’), а знака ‘+’ се разглежда като операция за сливане на два низа (конкатенация).

От казаното дотук, става ясно, че JavaScript притежава само един тип числови данни. Той обхваща, както целочислените, така и реалните данни (с плаваща точка). Числата могат да бъдат представени в десетичен формат (например 12, 4.37e-12 и т.н.), като осмични числа (основа 8 на бройната система) или шестнадесетични (основа 16). Осмичните числа се записват, като преди стойността на числото се запише 0 (нула), а шестнадесетичните - пред стойността се запише ‘0x’ (нула и ‘x’).

При представяне на низовете могат да се използват и специални знаци. Така например, специалният знак “\n” указва на интерпретатора да продължи представянето на текста на нов ред:

```
" Това е текст, който се показва \n на два реда"
```

Ляво наклонената черта е символ, който показва на интерпретатора, че предстои поява на специален знак. Ако трябва в текстовия низ да се включат кавички (не като начало или край на низа), пред тях трябва да се постави ляво наклонена черта. Специалните знаци са показани в следващата таблица:

Символ	Означение (интерпретация)
\\	Ляво наклонена черта
\’	Апостроф
\”	Кавички
\b	Символ backspace
\f	Символ form feed (нова страница)
\n	Нов ред
\r	Символ carriage return (връщане в началото на реда)
\t	Табулатор

JavaScript позволява да се създават нови обекти. За тази цел се използва служебната дума **new**.

Освен стандартните типове данни, често се налага използването на набор от еднотипни елементи (данни), обръщението към които се извършва по номера на елемента. Подходящата структура за дефиниране на такива елементи е **масивът**. Масивите в JavaScript се специфицират като прости и сложни. Като пример за прост масив може да се използва, описанието на списък с компании занимаващи се с компютърни технологии:

```
var companyNames = new Array(7) ;
companyNames[0]=”Microsoft”;
companyNames[1]=”HP” ;
companyNames[2]=”IBM”;
companyNames[3]=”Addobe”;
companyNames[4]=”Borland”;
companyNames[5]=”Epson”;
companyNames[6]=”Compaq”;
```

В този фрагмент програмен код се декларира масив от елементи с наименование `companyNames`, който съдържа наименованията на фирмите. В JavaScript (след версия 1.1) масивите се разглеждат като обекти. Всеки масив се явява инстанция (обект) на класа `Array`, дефиниран в JavaScript. За създаване на нов обект в JavaScript се използва служебната дума `new`. Цифрата в скобите при декларацията на масива указва броя на елементите които съдържа масива.

В горния масив, за обръщение например, към `”Microsoft”` се използва `companyNames[0]`, а към `”Addobe”`- `companyNames[0]`. Номерът на елемента се указва посредством квадратни скоби, като номерацията започва от 0 - за първия елемент, 1 - за втория елемент и т.н.

Вграденият в JavaScript клас `Array` дава и други възможности за деклариране и инициализиране на масивите:

```
var companyNames = new Array("Microsoft","HP","IBM",
"Adobe","Borland","Epson","Compaq") ;
```

Доста свободен е и начинът на обръщение към елементите на масивите. Така например, долните два израза са еквивалентни в JavaScript:

```
companyNames[5]
companyNames["Epson"]
```

Тъй като масивите са обекти, в сила и общото правило за обръщение към елементите на обект. Така следната конструкция е еквивалентна на представените по-горе две възможности за обръщение към елемент от масива:

```
companyNames.Epson
```

Когато трябва да се съхраняват повече параметри за даден елемент от някакво множество, се използват многомерни (сложни) масиви. Многомерните масиви в JavaScript са масиви от масиви. Например, ако за всяка от горните компании, освен наименование се изисква да се зададе и броя на работещите във тях, се получава таблица с две колони - наименование и брой на служители. В този случай, всеки ред от таблицата може да се разглежда като масив, а обединението на тези масиви също е масив. Ето един начин на представяне на тези данни:

```
var microsoft = new Array ("Microsoft","software") ;
var hp = new Array ("Hp","hardware") ;
var ibm = new Array ("IBM","hardware") ;
var adobe = new Array ("Adobe","software") ;
var borland = new Array ("Borland","software") ;
var epson = new Array ("Epson","hardware") ;
var compaq = new Array ("Compaq","hardware") ;

var companyNames = new Array (microsoft, hp, ibm, adobe,
borland, epson, compaq) ;
```

В масива `companyNames` обръщението към "IBM" може да се извърши посредством израза `companyNames[2][0].` Както се вижда, синтаксисът на обръщение към масиви е както в езика C и C++ (индексите по редове и колони са в отделни квадратни скоби).

Тъй като масивите са обекти, те притежават и някои функции за обработка (методи). Най-често използваните медоди на обектите от типа Array са:

join (разделител) - връща низ, който съдържа всички елементи на масива. Отделните елементи са разположени последователно и са разделени с аргумента ***разделител***. Като ***разделител*** може да бъде използван определен символ или символен низ.

reverse () - обръща реда на елементите на масива.

sort () - сортира елементите на масива.

1.3 Оператори в JavaScript

JavaScript съдържа пълен набор от операции за сравнение, логически и аритметически операции и изрази. Като наследник на C++, JavaScript има структура на операциите, както и езика Java с малко изключения. Затова тук ще отбележим само разликите в използваните операции. Като език работещ предимно с Web, в JavaScript не е включена групата на побитовите операции, които се използват за работа с данни на ниво битове. Освен това, от логическите операции липсват съкратените формати на операциите И и ИЛИ, както операцията 'изключващо ИЛИ'. Всички останали операции са както при език Java.

Оператор за присвояване.

Операторът за присвояване на стойност в JavaScript е знакът “равно” (=). Той може да се комбинира с аритметически и логически операции за формиране на така наречените съкратени оператори (изрази):

Запис	*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***

Условен преход.

JavaScript предлага различни начини за избор на алтернативи в случаите, когато содът среща конструкция за условен преход.

- **Оператор switch**. Синтаксисът на оператора switch е подобен на аналогичният оператор в C++ и Java. Той позволява на програмата да провери стойността на дадена променлива или израз и в зависимост от резултата да изпълни някоя от няколко възможни савкупности от оператори. В предложеният пример, програмата проверява стойността на низовата променлива `Case_Var` и изпълнява някоя от комбинациите от оператори заключени между `case` и `break.` Всяка конструкция `case` се идентифицира с някоя от възможните стойности на `Case_Var` (в случая “Microsoft”, “HP”, “IBM”).

```
switch (Case_Var) {
case “Microsoft”:expr1 ; expr2 ; expr3 ; break ;
case “HP”:expr4 ;expr5 ; break ;
case “IBM”:expr6 ;expr7; break ;
default:expr8 ;break ;
}
```

Както и в C++ и Java, в оператора `switch` се използва секция `default`. Тя се изпълнява, ако стойността на променливата `Case_Var` не съвпадне с никоя от посчените в `case` случаи. Операторът `break` се използва за да осигури излизане от оператора `switch`, когато се изпълнят операторите за дадения случай.

- **Оператор if**. Синтактсисът на оператора за условен преход **if** има вида:

`if (expr_log) {expr1; expr2; …}`, където `expr_log` е логически израз, а във фигурните скоби се намират операторите, които ще се изпълнят, ако стойността на логическия израз е ***true***. Ако стойността на логическия израз е ***false***, операторите във фигурните скоби няма да се изпълнят.

- Оператор else. Операторът **else** се използва като разщирение на **if**. Той определя кодът, който трябва да се изпълни, когато стойността на логическият израз в оператора **if** е ***false***:

```
if (expr_log) {expr1; expr2; …}
   else {expr3; expr4 ; …}
```

Синтаксисът и начинът на използване на тези оператори са същите както в език Java. В JavaScript може да се използва и съкратеният условен оператор, познат от Java:

```
var1 = Log_expr ? expr1 : expr2 ;
```

Оператори за цикъл и управление.

Операторите за цикъл имат същата синтактическа конструкция, както аналогичните оператори от Java. Операторът за цикъл с предусловие има вида:

```
while (Log_expr) expr1 ;
```

Цикълът с постусловие е със следния синтаксис:

```
do expr1 while (Log_expr);,
```
а цикълът с управляван с променлива (цикъл for) е със следната конструкция:
```
for (init_expr; Log_expr; iter_expr) expr1 ;
```
Допълнителни възможности за управление и изход от конструкциите за цикъл могат да се реализират с помоща на операторите **break (прекъсни)** и **continue** по-същия начин както това се прави в Java.

1.4. Обекти в JavaScript.

JavaScript, както повечето съвременни програмни езици е обектно-ориентиран. Това означава, че могат да се създават специални именовани набори от данни и вградени функции (методи) за обработка на данните.

Създаването на нов обект изисква най-напред да се конструира шаблон за обекта. Шаблон на обект в JavaScript се създава помоща на декларация на функция. След това може да се създават екземпляри от този шаблон. Технологията на създаване на обекти в JavaScript е илюстрирана с пример за създаване на обект home:

```
function home (price, floor, beds, baths, location, description) {
        this.price = price ;
        this.floor = floor ;
        this.beds = beds ;
        this.baths = baths ;
        this.location = location ;
        this.description = description ; }
```

Това е декларация на шаблона за обекти от тип home. В скобите са изброени данните, които включва шаблона. Важно е да се отбележи използването на ключовта дума **this** в частта с оператори на функцията. Тя се използва за обръщение към текущия обект. На основата на този шаблон могат да се създадат обекти от типа home. За целта се използва ключовата дума new:

```
home01 = new home (35000, 250, 3, 2, "Blagoevgrad", "Тухлена, добро изложение") ;
```

Ключовата дума new създава нов обект. Същевременно със създаването на обекта се задават стойности на данните (те се нарияат свойства на обекта). Например, създадения обект home01 съдържа данни за къща, за която е е зададена цена 35000 лв., 250 м^2 площ, 3 спални и т.н. Присвояването на стойностите на параметрите в обекта се извършва като се замени this от шаблона с оператора за присвояване:

```
home01.price = prise ;
home01.floor = floor ;
.............
```

След декларацията на обекта, неговите компоненти могат да използват в различни операции:

```
document.writeln ("Тази къща струва :" + home01.price + "лв.") ;
```

Също както свойствата на обект са променливи, асоциирани с него, методите са по същество функции, асоциирани с обекта. Например един от методите, използван често за извеждане на информация е document.writeln(), е функция осигурена от JavaScript за писане на текст в XHTML документ. В случая, writeln() е функция на асоциирания обект document. Съвместимите с JavaScript браусери осигуряват някои специални базови обекти, като document, затова обръщението към тях не изисква предварително деклариране.

Методи могат да бъдат добавени за обектите които се дефинират при разработване на програми с JavaScript. За целта, наименованието на декларирана функция се присвоява на метод за даден обект. В следващия пример е добавен метод за показване на параметрите на обект от типа home в Web страница:

```
function ShowHome () {
      document.writeln("Цена на къщата:"+ this.price + "лв. <br\/>" ;
      document.writeln "Площ на къщата:"+ this.floor + " m2 <br\/>" ;
      document.writeln ("Брой спални :"+ this.beds + "  <br\/>" ;
      return ; }
```

Тази функция се присвоява на метод в декларацията на шаблон за обект:

```
      function home (price, floor, beds, baths, location,
description) {
            this.price = price ;
            . . . . . .
            this.ShowHome = ShowHome ; }
```

След тази декларация може да се използва метода `ShowHome` за показване на параметрите на обекта: `home01.ShowHome` ;

Вградени обекти

В JavaScript съществуват вградени обекти, които осигуряват някои много полезни методи. Най-често използваните вградени обекти са **String**, **Date** и **Math.**

Обектът **String** предлага набор от методи и функции за обработка на текстови низове. Всеки деклариран текстов низ в JavaScript се явява инстанция на шаблона **String** предоставен от браузерите съвместими с JavaScript. Така например операторът `MyString="Това е текстов низ"` позволява променливата `MyString` да се третира като обект **String**. Затова за нея могат да се използват методите декларирани за обекта **String**. В случая `MyString.length` е параметър (свойство), който определя дължината на стринга.

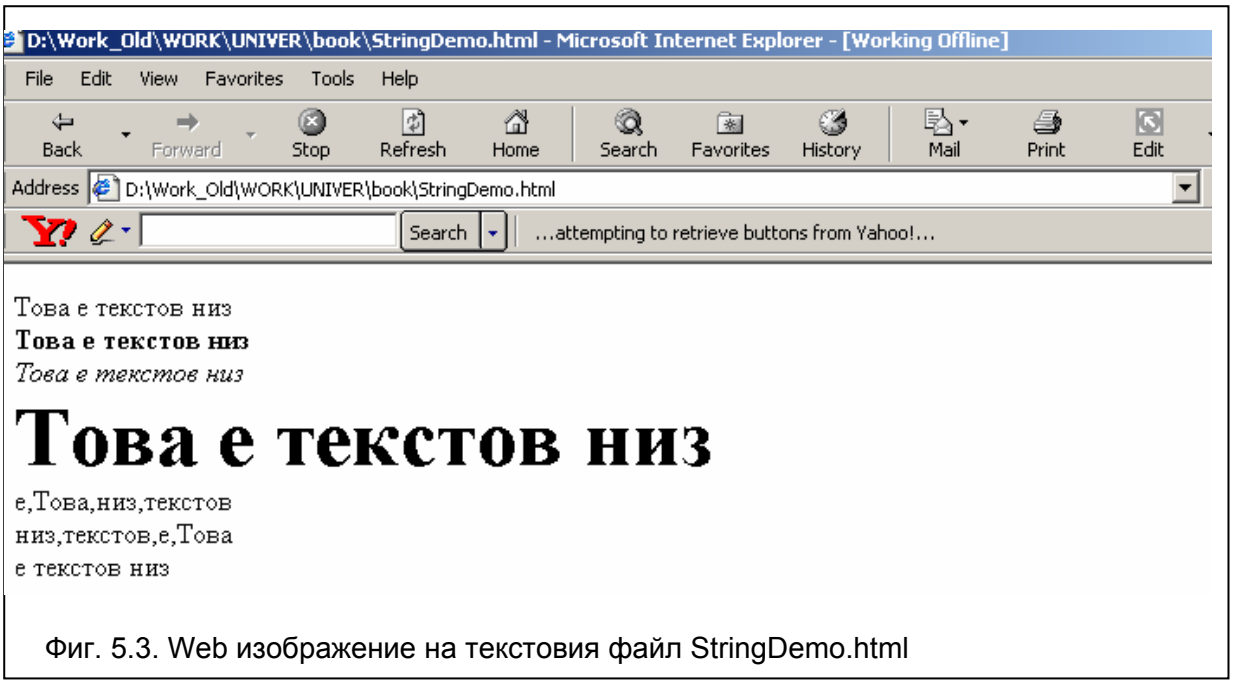
Част от функциите на вградения обект **String** са показани в примера по-долу. В него е деклариран текстов низ `s`, за който са приложени различни вградени методи. Текстовият файл е записан като "StringDemo.html" и е изобразен като Web страница на фиг.5.3..

`"StringDemo.html"`

```
<HTML>
<HEAD>
<SCRIPT>
var s=new String ("Това е текстов низ") ;
function StringDemo(string_s) {
        document.write(string_s + "<BR>") ;
        document.write(string_s.bold() + "<BR>") ;
        document.write(string_s.italics() + "<BR>") ;
        document.write(string_s.bold().fontsize(24) + "<BR>") ;
        document.write(string_s.split(" ").sort() + "<BR>") ;
        document.write(string_s.split(" ").reverse() + "<BR>") ;
        document.write(string_s.substring(5.20) + "<BR>") ;
}
</SCRIPT>
<TITLE> Използване на обект STRING </TITLE>
</HEAD>
<BODY onload = StringDemo(s)>
```

Първите три функции `bold()`, `italics()` и `fontsize()` се използват за форматиране на текста, а следващите `split()`, `sort()` и `substring()` се използват за манипулиране (обработка) на текста.

Обектът **Math** притежава методи за изпълнение на различни математични функции, а обектът **Date** дава възможност за работа с времена и дати.



Фиг. 5.3. Web изображение на текстовия файл StringDemo.html

Диалогови прозорци за предупреждение, въвеждане на данни (запитване) и потвърждение. JavaScript осигурява възможност за комуникация с потребителя посредством три прозореца:

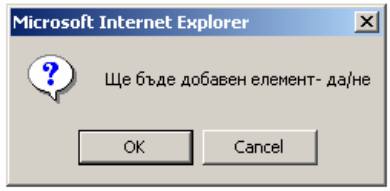
alert – извежда съобщение за предупреждение и бутон (фиг.5.4)

```
<HTML><HEAD><TITLE> Предупреждение </TITLE>
</HEAD>
<BODY>
  <SCRIPT> {
alert ("This is a warning !! ") ;}
</SCRIPT>
</BODY></HTML>
```

Фиг. 5.4. Предупреждение

confirm – извежда съобщение и два бутона (фиг. 5.5)

```
<HTML><HEAD><TITLE>Confirm </TITLE></HEAD>
<BODY>
<SCRIPT>log=confirm("Ще бъде добавен елемент – да/не") ;
</SCRIPT>
</BODY></HTML>
```



Фиг. 5.5. Потвърждение

prompt – извежда съобщение, поле за текст и два бутона (фиг.5.6)

```
<HTML>
<HEAD>
<SCRIPT> var s=new String (" ") ;
</SCRIPT>
<TITLE> Prompt </TITLE>
</HEAD>
<BODY>
<SCRIPT>prompt("Въведете
стойност",s) ; </SCRIPT>
</BODY>
</HTML>
```

Фиг. 5.6. Въвеждане на стойност

2. Добавяне на JavaScript към Web страници.

Разгледаните дотук примери показват някои от начините за добавяне на JavaScript към Web страници. Съществуват три основни начина за добавяне на JavaScript към Web страници:

- чрез вграждане на JavaScript кода във Web страници;
- чрез поставяне на JavaScript кода в секция <HEAD> на документа;
- чрез връзка към JavaScript код съхраняван във външен файл.

Когато се налага иаползване на малък скрипт, най-добрият избор е да се вгради в HTML документа. В този случай, JavaScript кода се записва в тялото на документа (след тага <BODY>). За обозначаване на програмния текст се използва таг <SCRIPT>.

Ако се използва код многократно в дадена страница, той може да се оформи във вид на функция и да се запише в секцията <HEAD> на документа. Този начин е използван в примера за използване на вградения обект String.

Когато се използва JavaScript в няколко документа е рационално той да бъде записан в отделен текстов файл и след това да се импортира в отделните документи. Текста във файла трябва да съдържа необходимия JavaScript код. В тага <SCRIPT>, където трябва да се разположи импортирания текст от файла се използва атрибут SCR, чиято стойност е наименованието на текстовия файл с JavaScript кода. Ето пример на използване на външен файл с JavaScript код:

```
<SCRIPT SCR=”functions.js” TYPE=”text/javascript”
```

Тук освен наименованието на текстовия файл се задава и тип на данните които се импортират: TYPE=”text/javascript”.

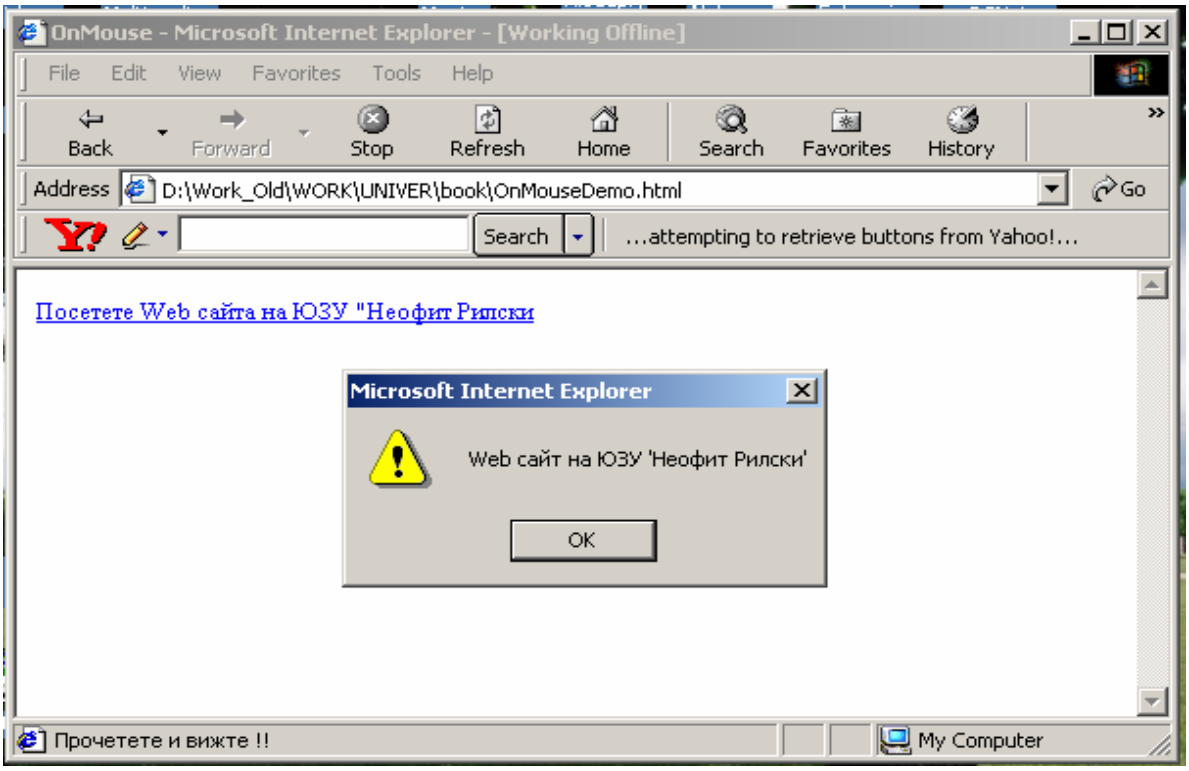
2.1. Функции за обработка на събития

Функциите за обработка на събития се използват в JavaScript за да се организира реакция (отговор) на действия на посетителя в дадена Web странице. Те осигуряват допълнителна информация за посетителя, като реагират на негови действия в страницата. Събития на които може да се реагира са свързани с движение на мишката в определени области от страницата, натискане на бутони, избор на общии от формуляри и други. Типични събития при работа с Web страници са събитията **onMouseOver** и **onMouseOut**. Първото от тези събития се генерира когато показалецът на мишката се позиционира върху обекта, а второто – при преместване на показалеца извън обекта. Като отговор на тези действия може да се използва код за смяна на текста в лентата за състояние, извеждане на предупредително съобщение (predupreditelen прозорец ***alert***), или промяна на изображението. JavaScript предлага множество функции за обработка на събития, но тук ще бъдат представени само функции за горните две събития. Другите събития се обработват по същия начин.

Обработка на събитията onMouseOver и **onMouseOut.** Обикновено тези събития се използват с тага за хипервръзка **<A>**. Може да се добавя код за обработка на тези събития, които настройват лентата за състояние чрез таймер, разменя изображението, извежда предупредителни съобщения и други. Ето как се използва обработчика на събития за разглежданите събития (HTML текст и Internet Explorer изображение – фиг.5.7):

```
<HTML> <HEAD>
<SCRIPT>
function alert_mes() {
alert ("Web сайт на ЮЗУ 'Неофит Рилски'") ;}
</SCRIPT>
<TITLE> OnMouse </TITLE> </HEAD>
<BODY>
<A HREF="http://www.aix.swu.bg/"
onMouseOver="window.status='Прочетете и вижте !!';
alert_mes() ;return true"
onMouseOut = "window.status = ''; return true">
Посетете Web сайта на ЮЗУ "Неофит Рилски </A>
```

</BODY>



Фиг.5.7 Internet Evplorer изображение

</HTML>

В примера са използван обработчици на събитията **onMouseOver** и **onMouseOut.** Когато показалеца на мишката се позиционира върху текста Посетете Web сайта на ЮЗУ "Неофит Рилски, се активира съботието **onMouseOver** и функцията обработчик формира в лентата за състояние (най-долният ред на изображението) подканящ текст: 'Прочетете и вижте !!'. Същевременно се изпълнява функция alert_mes(), която е декларирана в секцията <HEAD> на HTML документа. Тази функция генерира прозорец за предупреждение **alert**, с текст: "Web сайт на ЮЗУ 'Неофит Рилски". Другата функция е обработчик на събитие **onMouseOu** и се активира, когато показалеца на мишката се позиционира извън областта на текста Посетете Web сайта на ЮЗУ "Неофит Рилски. Нейното действие се заключава в изчистване на лентата за състояние.

Обработка на събитията onClik и **onChange.** Тези събития се активират когато се натисне бутон или се кликне върху даден обект или когато се промени поле във формуляр. Използването на тези събития е подобно на разгледаните досега.

3 Работа с обекти

JavaScript не е изцяло обектно-ориентиран език, защото той не притежава някои от възможностите на обектно-ориентираните езици. Но той е обектно-базиран език, на който са присъщи някои много важни характеристики на обектните езици.

Общите характеристики на обектно-ориентираните езици бяха дискутирани в предишния параграф (обектни характерситики на Java). Въпреки че JavaScript не подържа някои от тях, те имат важно значение защото Java аплети могат да се интегрират в скриптовите конструкции на JavaScript.

JavaScript не е истински език за програмиране, той е скриптов език. Той поддържа разработване на обектни типове и инстанциране на тези типове. Езикът осигурява добра поддържка на съставни обектни типове, но съвсем малка по отношение на модулност и многократно използване на обекти.

JavaScript притежава прост обектен модел, поддържан от известен брой предварително дефинирани обекти. В основата на обектния модел стои спецификацията на

обектни типове, които се използват за създаване на конкретни обекти (инстанции). Обектните типове в този модел се дефинират посредством свойства и методи.

Инстанциите от определен тип се създават с оператора **new**. Ако вземем предварително дефинирания обектен тип **Date**, може да се създаде променлива **currentDate**, на която да се присвои текущата дата:

```
currentDate = new Date () ;
```

В този пример конструкторът `Date ()` не получава параметри, но обектният тип `Date ()` позволява да се зададе конкретна дата при създаване на инстанцията:

```
currentDate = new Date (99,1,1) ;
```

Голяма част от работата с JavaScript се базира на използване на вградените обектни типове. Пълен списък на вградените обекти и техните свойства и методи могат да се намерят в справочниците по JavaScript.