ПРОГРАМИРАНЕ В ИНТЕРНЕТ

I. ВЪВЕДЕНИЕ В JAVASCRIPT

1. Какво е JavaScript?

JavaScript прави Web страницата по-динамична и привлекателна. Позволява на страницата да реагира на действията на посетителите, дава възможност за използване на специални ефекти (визуални и др.). JavaScript не се нуждае от компилатор (за разлика от Java) и е по-снизходителен в някои области като синтаксиса.

JavaScript е обектно-базиран скриптов език от страна на клиента, който се използва, за да се направят по-динамични Web страници. Обектно-базиран означава, че могат да се използват елементи като обекти; от страна на клиента означава, че JavaScript се изпълнява от софтуера, който използва посетителя, а не Web сървъра на сайта, обслужващ тази страница; скриптов език означава, че не изисква компилиране преди изпълнение.

2. Поставяне на JavaScript в HTML файл.

Използва се тагът <SCRIPT> (</SCRIPT>) със следните атрибути:

- language – определя скриптовия език;

```
Пример:
<SCRIPT language="JavaScript">
<!--
document.write ("Първи ред, написан с JavaScript");
//-->
</SCRIPT>
```

За скриване на JavaScript кода при по-стари браузъри между отварящата и затварящата част на тага <SCRIPT> се поставят HTML коментари. Двете наклонени черти пред затварящия HTML коментар са кодът за JavaScript коментар. Поставя се, защото JavaScript знае, че трябва да игнорира отварящия HTML коментар, но не и затварящия.

- src – извикване на външни скриптове (текстови файлове, съдържащи само JavaScript код и с разширение .js).

```
Пример:
<SCRIPT language="JavaScript"  src="my script.js">
</SCRIPT>
```

3. Метод document.write.

Това е функция, която изписва текстов низ в документ. Низът се поставя в кавички и скоби. Знакът ";" обозначава края на JavaScript конструкция. (Вж. първия пример).

4. Коментари в JavaScript.

Едноредови коментари се записват след обозначението "//".

Многоредове коментари се записват между обозначенията "/*" и "*/".

5. Създаване на външен JavaScript файл.

1) Създава се нов документ само с кода на JavaScript;

2) Документът се записва с разширение .js;

3) Създава се HTML документ с атрибут src="… .js" в тага <SCRIPT>;

4) HTML документът се записва с разширение .html.

II. ПРОМЕНЛИВИ

1. Променлива – представлява или съдържа стойност.

а) деклариране на променливи:

var име_на_променлива;

б) присвояване на стойности на променливи

Става чрез знака "=". Може да се декларира променлива и да й се присвои стойност на един ред:

var име_на_променлива=стойност;

Всяко ново присвояване на стойност на променлива се записва на нов ред. Допустимо е при някои версии да се изпускат "var" или ";", но за по-сигурно е добре да се изписват.

2. Правила за именуване на променливите:

- JavaScript променливите са чувствителни към регистъра (т. е. има значение дали се пишат главни или малки букви. Напр. abc, ABC, Abc, AbC са различни променливи.);

- Името на променлива може да съдържа латинска буква, цифра или долно тире, като не може да започва с цифра;

- Името не може да съвпада със запазена дума в JavaScript.

3. Типове променливи.

а) числа

Може да се работи с числа, представени в обикновен или експоненциален вид:

var suma=-34.15;

var resultat=4.5*e*5; (4.5*e*5=4,52.10^5=452000)

б) низове (последователност от букви, цифри, символи, интервали...)

var име_на_променлива="текстов низ"

var text="информатика"

var text='информатика'

Може да се използват кавички и апострофи, но трябва от двете страни на низа символът да е един и същ. JavaScript низовете са чувствителни към регистъра (напр. "информатика" и "Информатика" са два различни низа).

За някои символи се използва специален код, който започва със знака "\":

Символ	Специален код
Наклонена черта	\\
Изтрива един символ назад	\b
Нова страница	\f
Нов ред	\n
Табулация	\t

Нов ред в JavaScript влияе само на външния вид на кода, не и на извеждането от браузъра. Ако искаме при извеждане в браузъра даден текст да е на нов ред се използва тагът
. Ако даден низ е ограден с кавички и искаме като част от низа да изведем кавички, то пред тях се поставя знакът "\". Същото важи и когато при низ, ограден с апострофи искаме да изведем апострофи. За форматиране на текста в страницата се използват познатите HTML тагове, които могат да се поставят в скобите след document.write и да са част от заграден с кавички текст

в) булеви стойности

Булевата променлива може да прием две стойности: true (истина) и false (лъжа). True и false се използват без да се ограждат с кавички. Вместо true може да се използва стойността 1, а вместо false – 0.

var m=true;

г) null

С null се бележи, че променливата няма стойност. Това не е интервал, нито нула, а просто нищо.

4. Използване на променливите в скриптове.

При използването си променливата не се нуждае от кавички. Ако стойността й се извежда с някакъв текст, то този текст се огражда в кавички, а между текста и променливата се поставя знакът "+". Ако искаме преди или след стойността на променливата да има интервали, то те се поставят в текста с кавичките.

Пример1:

```
var myname="Петър";
document.write ("Аз се казвам "+myname+" и живея в България. ");
```

Резултат:

Аз се казвам Петър и живея в България.

Пример2:

```
var myname="Петър";
var text1="Аз се казвам "
var text2=" и живея в България. "
document.write (text1+myname+text2);
```

Резултат:

Аз се казвам Петър и живея в България.

III. ИЗПОЛЗВАНЕ НА ФУНКЦИИ

1. Какво е функция.

Функцията е малък скрипт, който се намира в рамките на по-голям скрипт. Нейното предназначение е да изпълнява едно или последователност от няколко действия. Напр. една функция може да изписва текстов ред в браузъра или да изчислява числова стойност и да връща тази стойност към главния скрипт.

2. Структуриране на функциите.

Една функция трябва да се декларира заедно със своето име и кода, който съдържа. Във функцията може да се импортира една или повече променливи, които се наричат параметри. Може да се върне

стойност от функцията към основния скрипт с помощта на конструкция за връщане на стойност

а) Деклариране на функция.

За да се декларира функция най-напред се пише запазената дума function, следвана от името на функцията и две скоби. Ако функцията използва параметри, те се описват в скобите; а ако не използва – скобите остават празни. След тях не се поставя точка и запетая.

б) Дефиниране съдържанието на функция.

Кодът на функцията се загражда с фигурни скоби. Добре е за по-голяма прегледност всяка скоба да е на нов ред.

```
Пример:
function  maxnumber ( )
{
……
}
```

в) Именуване на функциите.

Важат същите правила както при имената на променливите. Добре е имената да са смислени.

г) Добавяне на параметри към функции.

Параметрите на функциите представляват променливи в рамките на функцията. Записват се в скобите след името на функцията без да се използва запазената дума var и между отделните параметри, ако са повече от един, се поставя запетая. Може функцията въобще да няма параметри.

д) Конструкции за връщане на резултат.

Поставя се като последен ред на функцията, преди затварящата фигурна скоба. Състои се от запазената дума return, името на променливата, чиято стойност се връща към главния скрипт и точка и запетая.

```
Пример:
function  maxnumber ( )
{
……
return  max;
}
```

3. Обръщение към функция.

Обръщението към функция става чрез изписване на името й, скоби (с или без параметри) и точка и запетая. Функция може да се извика както в BODY секцията, така и в HEAD секцията на страницата. Може да се извика функция от друга функция. Преди да се извърши обръщение към функцията, тя трябва да бъде дефинирана. Затова обикновено дефинирането на функциите става в HEAD секцията на страницата.

а) Метод window.alert

С метода window.alert може да се извежда прозорец за предупреждение, като текстът на предупреждението се поставя в скоби и кавички. Ето един пример за извеждане чрез функция на предупреждение преди зареждането на страница (за да е преди зареждането на страницата функцията се поставя в HEAD секцията):

```
Пример:
<HTML>
<HEAD>
<TITLE> Функции </TITLE>
<SCRIPT language="JavaScript">
<!--
function preduprezhdenie ()
{
window.alert ("Това е предупреждение!");
}
preduprezhdenie ();
document.write ("Това е моята web страница");
//-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#0AAAFA">
</BODY>
</HTML>
```

б) Извикване на функция с параметри.

- Глобални променливи – това са променливите, които се дефинират извън функция и стойностите им могат да бъдат променяни на произволно място в скрипта – в или извън функция.

- Локални променливи – това са променливите, които се дефинират във функция и могат да бъдат използвани само от тази функция. Те трябва да се декларират във функцията с помощта на

служебната дума var. Като параметър на функция може да се използва не само стойност, но и променлива. Тя се поставя в скобите и при обръщението към функцията нейната стойност се присвоява на локална променлива за функцията и се работи с нея.

Пример:

```
<SCRIPT language="JavaScript">
<!--
function tochki (broi)
{
window.alert ("Вие имате "+broi+"  точки!");
}
var aktiv=1200;
tochki (aktiv);
//-->
</SCRIPT>
```

в) Извикване на функции с конструкции за връщане на стойност.

Общият вид е:

var име_на_променлива=име_на_функция ( );

По този начин променливата получава стойността върната от функцията и може да я използва но-късно в скрипта. Но за тази цел функцията завършва с return (име_на_променлива) и в скобите се записва променливата, чиято стойност се връща.

Пример:

```
<SCRIPT language="JavaScript">
<!--
function sazdavane_text ( )
{
var text1="Аз се казвам Петър"
var text2="  и живея в България."
var novtext= text1+ text2;
return (novtext);
}
var izrechenie= sazdavane_text ( );
window.alert (izrechenie);
//-->
</SCRIPT>
```

IV. ОСНОВНИ JAVASCRIPT ОПЕРАТОРИ

1. Математически оператори.

а) Оператор за събиране (+).

Могат да се събират две или повече числа, числа и стойности на променливи или само стойности на променливи. Може да се слепват текстове. Резултатът автоматично се преобразува в подходящ тип. (Напр. при събиране на цяло и дробно число, резултатът е число с плаваща запетая., при събиране на цяло число и текст резултатът е текст).

б) Оператор за изваждане (-).

в) Оператор за умножение (*).

г) Оператор за деление (/).

Трябва да се внимава да не се получи деление на нула. Резултатът в този случай е неопределен и зависи от браузъра. При деление на две цели числа, които не се делят без остатък, резултатът се преобразува до число с плаваща запетая (т. е. дробно). При някои браузъри резултатът 0.75 може да се изведе като .75.

д) Оператор за деление по модул (%).

Резултатът е остатъкът при деление на едно число на друго. Напр. резултатът при: 11%5 е 1. Ако едното число се дели точно на другото, резултатът е 0.

е) Оператор за инкрементиране (++).

Увеличава стойността на променлива с 1. Има значение дали операторът се поставя преди или след променливата. Ако е поставен преди променливата, първо се увеличава стойността с единица и после се изпълнява останалата част от конструкцията.

Пример:

```
var  num=2;
var  result=++num;
```

И това е еквивалентно на:

```
var  num=2;
num=num+1;
var result=num; (Накрая num има стойност 3, result – стойност 3)
```

Ако операторът е поставен след променливата, първо се изпълнява останалата част от конструкцията, а след това се увеличава стойността с единица.

Пример:
var num=2;
var result=num++;
И това е еквивалентно на:
var num=2;
var result=num;
num=num+1; (Накрая num има стойност 3, result – стойност 2)

ж) Оператор за декрементиране (−−).

Намалява стойността с 1. Ако е поставен преди променливата, първо се намалява стойността с единица и после се изпълнява останалата част от конструкцията. Ако операторът е поставен след променливата, първо се изпълнява останалата част от конструкцията, а след това се намалява стойността с единица.

з) Оператор за унарно отрицание (−).

Поставя се пред число или променлива и променя знака им – от положителни стават отрицателни и обратно.

2. Оператори за присвояване.

а) Оператор за присвояване (=).

Служи за задаване на нова стойност на променлива.

Пример:
var suma=1000;

б) Оператор за прибавяне и присвояване (+=).

Този оператор прибавя стойността от дясната страна към променливата от лявата страна и след това присвоява на променливата в ляво новата стойност.

Пример:
var suma=1000;
suma+=1;
И това е еквивалентно на:
var suma=1000;
suma=suma+1;

б) Оператор за изваждане и присвояване (−=).

Този оператор изважда стойността от дясната страна от променливата в лявата страна и след това присвоява на променливата в ляво новата стойност.

Пример:
var suma1=1000;
var suma2=500;
suma1-=suma2;
И това е еквивалентно на:
var suma1=1000;
var suma2=500;
suma1=suma1-suma2;

в) Оператор за умножение и присвояване (*=).

Този оператор умножава стойността от дясната страна с променливата в лявата страна и след това присвоява на променливата в ляво новата стойност.

г) Оператор за деление и присвояване (/=).

Този оператор дели променливата от лявата страна на стойността от дясната страна и след това присвоява на променливата в ляво новата стойност.

д) Оператор за деление по модул и присвояване (%=).

Този оператор дели променливата от лявата страна на стойността от дясната страна и след това присвоява на променливата в ляво остатъкът от делението.

3. Оператори за сравнение.

Операторите за сравнение служат за създаване на логически условия най-често при условни конструкции и цикли. Тези оператори могат да върнат една от двете стойности: true (истина) или false (лъжа).

Използват се шест оператора:

Оператор	Означение	Действие
Равно на	==	Връща true, ако стойностите от двете страни на оператора са еднакви.
Различно от	!=	Връща true, ако стойностите от двете страни на оператора **не** са еднакви.
По-голямо от	>	Връща true, ако стойността от ляво е по-голяма от стойността в дясно.
По-малко от	<	Връща true, ако стойността от ляво е по-малка от стойността в дясно.
По-голямо или равно	>=	Връща true, ако стойността от ляво е по-голяма или равна на стойността в дясно.
По-малко или равно	<=	Връща true, ако стойността от ляво е по-малка или равна на стойността в дясно.

4. Логически оператори.

Оператор	Озна-чение	Действие
AND	&&	Логическо "и"
OR	||	Логическо "или"
NOT	!	Логическо отрицание

5. Приоритет на операциите.

1) Скоби – ( );
2) Унарно или логическо отрицание - - !;
3) Умножение, деление, деление по модул - * / %;
4) Събиране, изваждане - + -;
5) Сравнение за неравенство - < <= > >=;
6) Сравнение за равенство - == !=;
7) Логическо "и" - &&;
8) Логическо "или" - ||;
9) Присвояване - = += -= *= /= %=.

V. УСЛОВНИ КОНСТРУКЦИИ

1. Блокови конструкции if/else.

а) структура:

```
if (логическо условие)
{
...     (тук стоят JavaScript командите, които се изпълняват при
}       удовлетворяване на логическото условие – стойност true)
else
{
...     (тук стоят JavaScript командите, които се изпълняват при
}       неудовлетворяване на логическото условие – стойност false)
```

Частта else с командите след нея не е задължителна. Ако не се постави, действието продължава с командите след частта с if.

Условните конструкции могат да се влагат. Втората блокова конструкция if/else може да се постави в скобите след if и ще се изпълни, ако логическото условие на първата конструкция се удовлетворява. Може да се постави в скобите след else и ще се изпълни, ако лог. условие на първата конструкция не се удовлетворява. На същия принцип могат да се влагат и повече условни конструкции.

б) сложни сравнения:

Използват се скоби за ограждане на елементите, от които се състои едно сложно сравнение.

Пр.1 Проверка дали x е от интервала (2;11]:

```
if ((x>2)&&(x<=11)) …
```

Пр.2 Проверка дали x се дели на 5 чрез признака за деление на 5:

```
if ((x%10==0)||(x%10==5)) …
```

2. Конструкция switch.

Тя позволява да се избира измежду повече от две възможности (не както при if/else измежду две).

Нейният вид е:

```
switch (име на променлива)
{
  case  стойност1:
      ...               (действия 1);
      break;
  case  стойност2:
      ...               (действия 2);
      break;
……
   case  стойностk:
      ...               (действия k);
      break;
   default:
      действия;
}
```

Ако стойността на променливата е равна на стойност1 се извършват действия 1, ако е равна на стойност2, се извършват действия 2 и т.н. Конструкцията break след извършените действия казва на браузъра да излезе от блока на кода и да премине към първия ред след блока. Така се осигурява невъзможността да се изпълнят всички случаи след този, който е върнал true. Ако стойността на променливата не съвпада с нито една стойност се изпълняват действията след default.

VI. ЦИКЛИ

1. Определение.

Цикълът е част от код, който може да се изпълнява многократно. Циклите в JavaScript са три вида: цикъл for, цикъл while и цикъл do while.

2. Цикъл for.

а) структура:

for (име на променливата-брояч=начална стойност; условие за продължаване на цикъла; правило за изменение на променливата-брояч)

```
{
  ……    (тяло на цикъла – действията, които се повтарят)
}
```

Тук променливата-брояч не е нужно да се декларира предварително с var.

Пример:

```
for (i=1; i<=10; i+=1)
{
document.write ("Цикълът се изпълнява за "+i+" път. <BR>")
}
```

В примера променливата-брояч има начална стойност 1. При всяко преминаване през цикъла нейната стойност се увеличава с 1. Цикълът ще се повтаря, ако стойността на i е по-малка или равна на 10 (т. е. 10 пъти).

б) вложени цикли:

В тялото на цикъла може да бъде вложен друг цикъл, в неговото тяло – трети и т. н.

Пример:

```
for (i=1; i<=10; i+=1)
{
document.write ("Таблица за умножение с "+i+"<BR>");
for (j=1; j<=10; j+=1)
{
var k=i*j;
document.write (i+" x "+j+" = "+k+"<BR>");
}
}
```

3. Цикъл while.

Този цикъл повтаря изпълнението си, докато даденото логическо условие връща стойност true. Общата структура е следната:

```
while (логическо условие за продължаване на цикъла)
{
  ……    (тяло на цикъла – действията, които се повтарят)
}
```

При цикъла while (за разлика от цикъла for) трябва да се декларира променливата (променливите), която се използва в логическото условие и да й се присвои стойност преди да се вмъкне в цикъла. В тялото на цикъла трябва да се променя стойността й, иначе ще се получи безкраен цикъл. Ако логическото условие не е изпълнено още в началото, командите в цикъла няма да се изпълнят нито веднъж. Цикли while също могат да се влагат.

Ето как би изглеждал примерът от 2.а) с цикъл while:

```
var i=1;
while (i<=10)
{
document.write ("Цикълът се изпълнява за "+i+" път. <BR>");
i+=1
}
```

4. Цикъл do while.

И този цикъл повтаря изпълнението си, докато даденото логическо условие връща стойност true. Но за разлика от цикъл while, цикълът do while се изпълнява поне веднъж, дори и логическото условие да не се изпълнява още в началото. Общата структура е следната:

```
do
{
  ……    (тяло на цикъла – действията, които се повтарят)
} while (логическо условие за продължаване на цикъла)
```

Ето така ще изглежда примерът от 2.а) с цикъл do while:

```
var i=1;
do
{
document.write ("Цикълът се изпълнява за "+i+" път. <BR>");
i+=1
} while (i<=10)
```

VII. ФУНКЦИИ ЗА ОБРАБОТКА НА СЪБИТИЯ

1. Какво е функция за обработка на събития.

Това е предварително дефинирана ключова дума в JavaScript, която се използва за обработване на дадено събитие от Web страница. Събитието е нещо, което се случва когато потребителят извърши някакво действие (щракване с мишката, щракване върху бутон и т. н.).

2. Използване на функции за обработка на събития.

Функцията за обработка на събитие се добавя като допълнителен атрибут към HTML тага, като стойността на атрибута може да бъде JavaScript код, заграден с кавички.

```
Пример:
<HTML>
<BODY>
<FORM>
<INPUT type="button"  onClick="window.alert (‘здравей!’); ">
</FORM>
</BODY>
</HTML>
```

3. Основни функции за обработка на събития:

а) събитието Click (onClick)

Събитието Click настъпва, когато потребителят щракне върху определени области от Web страницата. Тези области се намират в елементи от форми и връзки. В горния пример събитието Click настъпва при натискане на бутон от формуляр, създаден чрез тага <INPUT> и предизвиква поява на прозорец за предупреждение.

```
Друг пример:
<HTML>
<BODY>
<A HREF="http://www.nagradi.bg" onClick="window.alert (‘Това
е адресът на страницата с наградите’); return false"> Награди
<A>
</BODY>
</HTML>
```

Тук след появата на прозорецът за предупреждение, ако не се върне стойност false чрез return, ще се зареди страницата.

б) събитието Mouseover (onMouseOver)

Събитието Mouseover настъпва, когато потребителят премести показалеца на мишката над хипервръзка. Ако в примера от а) вместо onClick използваме onMouseOver, предупреждението се появява при посочване на хипервръзката.

в) събитието Mouseout (onMouseOut)

Това събитие настъпва, когато потребителят премести показалеца на мишката извън хипервръзка.

г) събитието Load (onLoad)

Това събитие настъпва, когато една Web страница завърши зареждането си. Функцията се поставя в тага <BODY>.

д) събитието Unload (onUnload)

Това събитие настъпва, когато потребителят напусне Web страницата (при щракване върху хипервръзка, при изписване на нов адрес в браузъра или при затваряне на прозореца). Функцията се поставя в тага <BODY>.

е) събитието Focus (onFocus)

Събитието Focus настъпва, когато потребителят предаде фокуса на прозорец или елемент от формуляр в Web страница. Потребителят дава фокуса като щракне върху елемента. Например, когато потребителят щракне в текстово поле (преди да въведе нещо), той предава фокуса на текстовото поле. А ако щракне в неактивен прозорец, той го прави активен и му предава фокуса. Тази функция може да се използва в елемент от формуляр или в отварящия таг <BODY> (при предаване на фокуса към прозореца).

```
Пример:
<HTML>
<BODY>
<FORM> Въведете вашето име:
<INPUT type="text"  onFocus="window.alert (‘Използвайте само
главни букви!’); ">
</FORM>
</BODY>
</HTML>
```

ж) събитието Blur (onBlur)

Събитието настъпва, когато потребителят отнеме фокуса от даден елемент от форма или прозорец. За да отнеме фокуса на нещо, потребителят обикновено предава фокуса на друг елемент.

з) събитието Change (onChange)

Настъпва, когато потребителят промени нещо в рамките на даден елемент от формуляр.

```
Пример:
<HTML>
<BODY>
<FORM>Кой е любимият ви сезон?
<SELECT onChange="window.alert (‘Променяте избора си!’);">
<OPTION selected> зима </OPTION>
<OPTION> пролет </OPTION>
<OPTION>  лято </OPTION>
<OPTION>  есен </OPTION>
</SELECT>
</FORM>
</BODY>
</HTML>
```

и) събитието Submit (onSubmit)

Събитието настъпва, когато потребителят изпрати форма от web страница. Функцията onSubmit може да бъде извикана от отварящия FORM таг.

```
Пример:
<HTML>
<BODY>
<FORM onSubmit="window.alert(‘Благодарим Ви!’);">Как се казвате?<BR>
<INPUT type="text" name="thename"><BR>
<INPUT type="submit">
</FORM>
</BODY>
</HTML>
```

4. Други събития.

- **Abort (onAbort)** - Събитието настъпва, когато потребителят спре зареждането на изображение. Функцията onAbort може да се поставя в IMG тага на даден HTML документ.

- **Dragdrop (onDragDrop)** - Събитието настъпва, когато потребите-лят пусне файл в прозореца на браузъра. Функцията се поставя в BODY тага.

Пример: <BODY onDragDrop="window.alert(‘Hi!’); return false;">. Връщането на резултат false предотвратява опита на браузъра да отвори файла.

- **Error (onError)** – Събитието настъпва, когато се получи грешка при зареждането на страница или изображение. Функцията може да се използва в IMG тага или в отварящия BODY таг.

- **Keydown (onKeyDown)** – Събитието настъпва, когато потребителят натисне клавиш от клавиатурата. Функцията може да се постави в таг за текстова област или в отварящия BODY таг.

- **Keypress (onKeyPress)** – Събитието настъпва, когато потребителят задържи клавиш от клавиатурата. Функцията може да се постави в таг за текстова област или в отварящия BODY таг.

- **Keyup (onKeyUp)** – Събитието настъпва, когато потребителят отпусне клавиш от клавиатурата. Функцията може да се постави в таг за текстова област или в отварящия BODY таг.

- **Mousedown (onMouseDown)** – Събитието настъпва, когато потребителят натисне бутон на мишката, но преди завършване на щракането. Функцията може да се постави в рамките на връзка (котва) на web страница или в други области (като DIV тагове).

- **Mouseup (onMouseUp)** – Събитието настъпва, когато потребителят отпусне бутона на мишката. Функцията може да се постави в рамките на връзка (котва) на web страница или в други области (като DIV тагове).

- **Mousemove (onMouseMove)** – Събитието настъпва, когато потребителят премести курсорът на мишката. Функцията може да се постави в отварящия BODY таг или в други тагове (като DIV тагове).

- **Move (onMove)** – Събитието настъпва при преместване на прозореца. Функцията може да се използва в отварящия BODY таг.

- **Reset (onReset)** – Събитието настъпва, когато потребителят щракне върху Reset бутона на форма. Функцията се поставя в отварящият таг FORM.

- **Resize (onResize)** – Събитието настъпва при промяна в размера на прозореца. Функцията може да се използва в отварящия BODY таг.

- **Select (onSelect)** – Събитието настъпва, когато потребителят избере някакъв текст в рамките на текстово поле или текстова област от страницата. Функцията може да се постави в таг за текстов вход или в таг за текстова облас в HTML документ.

VIII. ОБЕКТИ. ОБЕКТИТЕ NAVIGATOR И DOCUMENT

1. Какво е обект.

Обектът е начин на моделиране на нещо реално, въпреки че самият обект е абстрактна величина. Когато мислим за един обект, ние си представяме нещо общо, без да се интересуваме от конкретният му вид – напр. кола. Колата е част от обекта МПС, който включва коли, микробуси, камионе и др., а напр. Мерцедес е част от обекта кола. Така може да се изгради определена йерархия.

Но колите притежават и други характеристики, които наричаме свойства – цвят, големина, брой врати и т. н. На свойствата може да се дават стойности., според типа. Достъпът до свойствата на един обект се извършва посредством оператора точка "." (обект.свойство=стойност).

Методите представляват функции, които извършват различни операции със свойствата на обектите. Всеки метод е част от даден обект. Достъпът до метод на даден обект се извършва чрез точка. (обект.метод).

Има предварително зададени обекти в JavaScript, а има и възможност за създаване на собствени обекти. Но ние ще се занимаем с някои предварително дефинирани JavaScript обекти и ще разгледаме някои от техните свойства и методи.

2. Обектът navigator.

Той дава достъп до различни характеристики на потребителския браузър, като име, номер на версията и др. Но свойствата на този обект не могат да бъдат променяни, те са зададени в режим само за четене. Могат да бъдат полезни, когато е необходимо потребителят да бъде пренасочен според характеристиките на неговия браузър.

а) свойства;

- **appName** – Това свойство съдържа типа на браузъра, който използва потребителят.

Напр. window.alert("Вие използвате "+navigator.appName);

- **appVersion** – Номер на версията на браузъра и друга допълнителна информация.

- **language** – Съдържа низ от два знака, представящ език, на който е преведен браузъра или низ от пет знака, представящ и подезик.

- **userAgent** – Съдържа по-дълъг низ с по-изчерпателна информация за браузъра.

б) методи.

- **javaEnabled()** – Връща стойност true, ако потребителят е разрешил Java в браузъра си и стойност false в противен случай. Може да е полезен, ако искаме да покажем Java аплет, което е възможно само ако потребителят е разрешил Java в браузъра.

Пример:

```
var rr=navigator.javaEnabled()
if (rr==true)
    window.alert("Вие имате Java!");
else
    window.alert("Вие нямате Java! Не можете да видите Java аплета!");
```

- **plugins.refresh()** – за опресняване на наличните в браузъра Plug-In модули.

3. Обектът document.

а) същност;

Това е обект, който се създава от браузъра за всяка нова разглеждана HTML страница.

б) свойства;

- **alinkColor** – съдържа шестнайсетичната стойност или името на цвета на активна хипервръзка в HTML документ. Поставя се в HEAD частта и не може да се променя.

Пример:

```
<HEAD>
<SCRIPT language="JavaScript">
<!--
document.alinkColor="#FF0000";
//-->
</SCRIPT>
</HEAD>
```

- **linkColor** – съдържа стойността на цвета на непосетените хипервръзки на web страницата. Работи подобно на alinkColor.

- **vlinkColor** – съдържа стойността на цвета на посетените хипервръзки на web страницата. Работи подобно на alinkColor.

- **bgColor** – съдържа стойността на цвета на фона на web страницата

- **fgColor** – съдържа стойността на цвета на текста в уеб страницата.

- **images** – свойството е масив, който съдържа по един елемент за всяко изображение в web страницата. С помощта на свойството document.images.length може да се разбере колко изображения има в една страница.

- **lastModified** – съдържа датата и часа на последната промяна на документа. Но различните браузъри може да изведат различни резултати.

Пример:

```
<BODY BGCOLOR="#0AAAFA">
<SCRIPT language="JavaScript">
<!--
document.write ("<P ALIGN=RIGHT> Последна промяна: " +
document.lastModified+"</P>");
//-->
</SCRIPT>
</BODY>
```

- **links** – свойството е масив, който съхранява стойност за всяка връзка или свързана област от карта-изображение от страницата. С помощта на свойството document.links.length може да се разбере колко връзки има в една страница.

- **referrer** – съхранява URL адреса на страницата, в която се е намирал потребителят, преди да дойде на текущата страница. Понякога е възможно стойността да е 0 или да бъде неправилна, защото различните браузъри могат да приемат различни типове за това свойство.

- **title** – съхранява заглавието на HTML документа.

Пример:

```
<HTML>
<HEAD>
<TITLE> Белите мечки </TITLE>
</HEAD>
<BODY BGCOLOR="#0AAAFA">
<SCRIPT language="JavaScript">
<!--
document.write ("<H1>"+document.title+"</H1>");
//-->
</SCRIPT>
В тази страница ще прочетете много интересни факти за белите мечки.
</BODY>
</HTML>
```

- **url** – съхранява стойността на пълния URL адрес на текущия документ. Напр. може да се зададе URL адреса в края на web страницата и така при отпечатване адресът ще се появява. Макар, че това може да се извърши ръчно, при много страници е удобно да се използва именно това свойство.

Пример:

```
<HTML>
<HEAD>
<TITLE> Уроци </TITLE>
</HEAD>
<BODY BGCOLOR="#0AAAFA">
<H1> Урок №1 </H1>
.....
<P>
<SCRIPT language="JavaScript">
<!--
document.write ("Източник: "+document.URL);
//-->
</P>
</SCRIPT>
</BODY>
</HTML>
```

в) методи.

- **open()** – Методът дава възможност да се отвори нов документ и да се създаде съдържанието му изцяло чрез document.write() конструкции. Когато се извика методът, браузърът търси тези конструкции, за да може да пише в новата страница.

- **close()** – използва се за завършване на страницата, отворена с метода open().

- **write()** – Метъдът се използва за извеждане на стрингова стойност в страницата.

- **writeln()** – Методът действа по същия начин, както write(), но добавя JavaScript знак за нов ред в края на конструкцията

Посочените методи ще илюстрираме с пример, при който се създава нова страница въз основа на въведеното име на потребителя.

Пример:

```
<HTML>
<HEAD>
<TITLE> Потребителска страница </TITLE>
<SCRIPT language="JavaScript">
<!--
function newpage ()
{
var thename=document.myform.yourname.value;
if ((thename=="") || (thename==null))
     window.alert("Не сте написали името си!");
document.open();
document.write("<H1>Здравейте, "+thename+"!</H1>");
document.write("Добре дошли на "+document.URL);
document.close();
}
//-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#0AAAFA">
<B>Моля въведете Вашето име за персонализиране на страницата!</B>
<FORM name="myform">
<P>Име: <INPUT type="text" name="yourname" size="25"></P>
<P><INPUT type="button" value="click" onClick="newpage()"></P>
</FORM>
</BODY>
</HTML>
```

IX. ОБЕКТЪТ WINDOW

1. Какво представлява обектът window.

Това е обект, който се създава за всеки прозорец, който се показва на екрана. Обекти window се явяват главният прозорец, наборът от фреймове или отделен фрейм, както и всеки нов, създаден чрез JavaScript прозорец.

2. Свойства.

- **closed** – Свойството се използва, за да се провери дали потребителят е затворил даден прозорец.

- **name** – Съдържа името на текущия прозорец и позволява да дадем име на прозорец. Задаването на име на прозорец става в HEAD частта.

- **defaultStatus** – съдържа текста, който да се извежда в лентата на състоянието, когато не е указано нищо.

Пример: <BODY onLoad="window.defaultStatus='Welcome!';">

- **location** – Свойството съдържа текущият URL адрес на прозореца, като той може да бъде променян в движение, за да се пренасочи потребителя към нова страница, подобно на връзка.

Пример:

```
<FORM>
<INPUT type="button" value="За търсене в Web"
     onClick="window.locaton='http://yahoo.com';">
</FORM>
```

- **self** – Свойството е друг начин да назовем „текущият прозорец”. Използва се вместо обекта window и чрез него може да се извърши достъп до свойствата на текущия прозорец.

Пример:

```
<FORM>
<INPUT type="button" value="За търсене в Web"
     onClick="self.locaton='http://yahoo.com';">
</FORM>
```

- **status** – Свойството съдържа текста, който е изведен в лентата за състояние на прозореца.

Пример:

```
<BODY onLoad="window.defaultStatus='Welcome!';">
<A HREF="page2.html" onMouseOver="window.status='Страница
2'; return true;">Към втора страница </A>
</BODY>
```

3. Методи.

- **alert()** – Извежда прозорец с предупреждение, а потребителят трябва да щракне върху бутона ОК.

- **confirm()** – Извежда прозорец за потвърждение/отказ.

- **print()** – Методът извиква диалоговия прозорец Print, чрез който потребителят може да зададе желаните настройки и да отпечата документа.

Пример:

```
<FORM>
<INPUT type="button" value="Print" onClick="window.print();">
</FORM>
```

- **prompt()** – Методът извиква прозорец за въвеждане на данни, резултатът се присвоява на променлива.

Пример: var pr=window.prompt("Въведете година:", "2010")

- **open()** – Позволява да се отвори нов прозорец чрез JavaScript. Методът приема три параметъра, като третият задава няколко опции, които може да са необходими за прозореца. Основният синтаксис е:

Window.open("URL-адрес на HTML документа", "име на прозореца", "атрибут1=стойност1, атрибут2=стойност2... ");

Някои от атрибутите са:

Атрибут	Възможни стойности	Действие
width	число	Ширина на прозореца в пиксели
height	число	Височина на прозореца в пиксели
location	yes, no,1,0	Дефинира дали прозорецът да има или няма адресно поле за въвеждане на URL адрес
menubar	yes, no,1,0	Дефинира дали прозорецът да има или няма лента с менюта
resizable	yes, no,1,0	Дефинира дали потребителят да може да преоразмерява прозореца
scrollbars	yes, no,1,0	Дефинира дали прозорецът да има или няма плъзгачи, при голямо съдържание
status	yes, no,1,0	Дефинира дали прозорецът да има или няма лента на състоянието
toolbar	yes, no,1,0	Дефинира дали прозорецът да има или няма лента с инструменти

Всеки параметър, който не е дефиниран има стойност по подразбиране "no".

- **close()** – Позволява затварянето на отворен чрез JavaScript прозорец.

- **focus()** – Методът позволява да дадем фокуса на даден прозорец (т. е. да го изведем пред другите прозорци)

- **blur()** – Методът позволява да се постави прозорецът зад главния (т. е. отнема фокуса)

- **moveBy()** – Премества новия прозорец с определен брой пиксели хоризонтално и определен брой пиксели вертикално, които се подават като параметри при извикването на метода.

Пример: window.moveBy(50,70)

```
<FORM>
<P><INPUT type="button" value="Премести прозореца"
onClick="window.moveBy(50,50);"></P>
<P><INPUT type="button" value="Затвори прозореца"
onClick="window.close();"></P>
</FORM>
```

- **moveTo()** – Премества новия прозорец на определен брой пиксели хоризонтално от левия край надясно и на определен брой пиксели вертикално от горния край към долния, които се подават като параметри при извикването на метода.

- **resizeBy()** – За преоразмеряване на прозореца с определен брой пиксели, които се подават като параметри при извикването на метода.

- **resizeTo()** – Зядава се определен размер на прозореца в пиксели, които се подават като параметри при извикването на метода.

- **setTimeout()** – Позволява да се изпълни дадена JavaScript функция, след като изтече определен период от време.

- **clearTimeout()** – Позволява прекратяването извикването на setTimeout(), ако се извика преди да изтече периодът на setTimeout()

X. МАСИВИ

1. Какво представлява масивът.

Масивът е начин за съхранение на данни от един тип с цел осъществяване на лесен достъп до тях. Данните се наричат елементи на масива и се състоят от две части – име на масива и номер на елемента. Първият елемент е номериран с 0, вторият с 1 и т. н. Номерата се наричат индекси. За имената на масивите важат правилата за именуване на променливи.

2. Дефиниране на масив.

а) първи начин;

Общ синтаксис:

var ime_na_masiv = new Array (element0, element1, …)

Пример:

var plist=new Array("Георги", "Мария", "Теодора", "Стоян");

С този код елементът с индекс 0 на масива plist приема стойност "Георги", този с индекс 1 приема стойност "Мария" и т. н.

б) втори начин;

Първоначално се "заделя" пространство за масива чрез задаване броя на елементите, а след това на всеки от тях се присвоява стойност. Не трябва да забравяме, че ако броят елементите е 5, то елементите на масива имат индекси от 0 до 4.

Пример:

```
var plist=new Array(4);
var plist[0]="Георги";
var plist[1]= "Мария";
var plist[2]= "Теодора";
var plist[3]= "Стоян";
```

Можем да добавим пети елемент на масива: var plist[4]= "Петър";

в) трети начин;

Създава се масив, който не съдържа елементи (не се задават параметри, т. е. не се записва число в скобите) и по-късно могат да се присвояват стойности на елементи с произволен индекс.

3. Някои свойства на масива.

- **constructor** – Съдържа като стойност кода на функцията, която е създала масива.

- **length** – Връща число, което показва от колко елемента се състои масивът.

Пример (за масива от предния пример):

```
window.alert("В ученическия съвет влизат " + plist.lenght + " ученици");
```

4. Някои методи на масива

- concat() – Комбинира елементите от два или повече масива в нов масив. Новият масив се образува, като най-напред се поставят елементите на масива, използван за извикване на метода. След тях ще се поставят елементите на масива, посочен като параметър. Ако трябва да се комбинират елементите на повече от два масива, то като параметри се подават техните имена, разделени със запетаи и в ред, в който искаме да се добавят елементите им към новия масив.

Пример:

```
var zima=new Array("януари", "февруари", "март");
var prolet=new Array("април", "май", "юни");
var lqto=new Array("юли", "август", "септември");
var esen=new Array("октомври", "ноември", "декември");
var godina=zima.concat("prolet", "lqto", "esen");
```

Резултатът тук е масив godina с елементи: януари, февруари, март, април, май, юни, юли, август, септември, октомври, ноември, декември.

- **join()** – Комбинира елементите на масив в символен низ, като елементите се разделят чрез знак, който се подава като параметър към метода. Ако не се подаде никакъв параметър, за подразбиращ се знак за разделител се използва запетаята.

Пример:

```
var zima=new Array("януари", "февруари", "март");
var z=zima.join();
```

Стойността на z е: януари,февруари,март

- **pop()** – Премахва последния елемент от даден масив. Ако се присвои резултатът от метода на променлива, тя ще съдържа стойността на премахнатия елемент.

- **push()** – Добавя елемент или елементи в края на даден масив. Параметрите на този метод са новият елемент или елементи.

- **reverse()** – Обръща реда на елементите в масива.

- **shift()** – Премахва първия елемент от масив. Ако се присвои резултатът от метода на променлива, тя ще съдържа стойността на премахнатия елемент.

- **unshift()** – Добавя елемент към началото на масив. Елементът или елементите за добавяне се подават като параметри на метода.

- **slice**() – Премахва елементи от масива. Ако се присвои резултатът от метода на променлива, тя ще съдържа стойността на премахнатите елементи. Общият синтаксис на метода е:

ime_na_masiv.slice(index1,index2);

Тук индекс1 е индексът на първия елемент от отрязъка, а индекс2 е индекса, който следва след последния елемент от отрязъка.

Пример:

```
var mec=new Array("януари", "февруари", "март", "април", "май", "юни" );
var prolet=mec.slice(2,5);
```

Този код преобразува масива mec и сега неговите елементи са: януари, февруари, юни. А масивът prolet съдържа елементите март, април, май.

- **splice()** – Премахва или заменя елементи на масив. Параметрите, които могат да се подават са: индекса, от който да започне отрязъка; брой елементи за премахване/замяна; елементите за замяна/добавяне.

Пример1 (за премахване):

```
var mec=new Array("януари", "февруари", "март", "април", "май", "юни" );
var prolet=mec.splice(2,3);
```

Пример2 (за замяна):

```
var mec=new Array("януари", "февруари", "март", "април", "май", "юни" );
var pr=mec.splice(2,5, "август", "септември", "октомври");
```

Полученият масив има елементи: януари, февруари, август, септември, октомври, юни.

Ако вторият параметър е 0, няма да се заменят елементи, а ще се добавят нови на съответното място.

- **sort()** – Сортира масив в азбучен ред.

5. Масиви и цикли.

Обикновено за работа с масиви се използват цикли. Те могат да се използват за въвеждане на елементи на масив, за обхождане на масив с цел промяна, извличане на информация или извеждане на съдържанието му по определен начин и др.

Ще илюстрираме въвеждане на елементи на масив, сортиране и извеждане с помощта на цикли чрез следния пример:

```
<HTML>
<HEAD>
<TITLE> Ученически съвет </TITLE>
<SCRIPT language="JavaScript">
<!--
var plist=new Array()
i=0;
do
{
var element=window.prompt("Въведете член на ученическия съвет при 12а клас. Ако вече сте въвели всички имена, оставете полето празно и изберете ОК.", "Име");
plist[i]=element;
i++;
}
while (element!="");
plist.pop();
plist.sort();
//-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#0AAAFA">
<H2>Ученически съвет при 12<sup>а</sup> клас:</H2>
<SCRIPT language="JavaScript">
<!--
for(i=0; i<plist.length; i++)
{
document.write(i+1+". "+plist[i]+ "<BR>");
}
//-->
</SCRIPT>
</BODY>
</HTML>
```

XI. ОБЕКТИ MATH И DATE

1. Какво представлява обектът Math.

Този обект ни дава възможността да извършим определени математически изчисления

2. Някои свойства на обекта Math.

- **E** – константа на Ойлер *e* (приблизително 2,71828...)

Пример: window.alert(Math.E);

- **PI** – стойността на числото π (приблизително 3,14159...)

3. Някои методи на обекта Math.

- **abs()** – Връща абсолютната стойност на числото, което е подадено като параметър.

Пример: window.alert(Math.abs(-30));

- **sqrt()** – Връща квадратен корен от подаденото като параметър число.

- **exp()** – Връща *e* на степен подаденото като параметър число.

- **log()** – Връща естествения логаритъм подаденото като параметър число.

- **sin()** – Връща синус от подаденото като параметър число в радиани.

- **cos()** – Връща косинус от подаденото като параметър число в радиани.

- **tan()** – Връща тангенс от подаденото като параметър число в радиани.

- **round()** – Връща закръглената стойност на подаденото като параметър число.

- **ceil()** – Връща най-малкото цяло число, което е по-голямо или равно на подаденото като параметър число.

- **floor()** – Връща най-голямото цяло число, което е по-малко или равно на подаденото като параметър число.

- **random()** – Връща случайно число между 0 и 1, не изисква параметър.

Често се налага да получим не случайно число между 0 и 1, а случайно цяло число между 0 и A. За тази цел се налага да умножим случайното дробно число между 0 и 1 с A и да вземем цялата му част с floor().

Пример1: var R=Math.floor(Math.random()*5);

Пример2:

В този пример ще покажем как може да се използва методът random() за извеждане на случайна фраза.

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
<!--
var fraza = new Array(10);
fraza[0]= "Мъдър е този, който знае не много, а нужното. Есхил";
fraza[1]= "Ако можеш да си орел не се напъвай да си пръв сред гаргите. Питагор";
fraza[2]= "Мързелът се влачи толкова бавно, че бедността бързо го настига. Франклин";
fraza[3]= "Ако не умееш да говориш, научи се да мълчиш. Помпоний";
fraza[4]= "Който е приятел на всички, не е приятел на никого. Аристотел";
fraza[5]= "Не се страхувай от съвършенството - никога няма да го достигнеш. Дали";
fraza[6]= "Човек е истински мъдър тогава, когато знае, че не знае нищо. Сократ";
fraza[7]= "Въображението е по-важно от знанието. Айнщайн";
fraza[8]= "Проблемът на света е, че глупавите са самоуверени, а умните винаги се колебаят. Ръсел";
fraza[9]= "По-лесно е да се бориш за принципите си, отколкото да живееш според тях. Адлер";
var r = Math.floor(Math.random()*10);
//-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR=#0AAAFA">
<H2>Цитат на деня:</H2><BR>
<SCRIPT language="JavaScript">
<!--
document.write(fraza[r]);
```

```
//-->
</SCRIPT>
</BODY>
</HTML>
```

По подобен начин можем да изобразяваме случайна картинка в уеб страница.

- **pow()** – Връща стойността на първия параметър на степен стойността, подадена като втори параметър.

Пример: var st=Math.pow(2, 10) (st получава стойност 2 на 10 степен)

- **max()** – Връща по-голямото от двете числа, подадени като параметри.

- **min()** – Връща по-малкото от двете числа, подадени като параметри.

4. Какво представлява обектът Date.

Този обект позволява да се работи с дата и време. За да може да се използва трябва най-напред да бъде зададен.

Пример: var ime = new Date();

5. Някои методи на обекта Date.

- **getDate()** – Връща деня от месеца според локалното време на потребителя.

- **getDay()** – Връща деня от седмицата (0-6) според локалното време на потребителя.

- **getHours()** – Връща броя часове (0-23) в деня според локалното време на потребителя.

- **getMinutes()** – Връща броя минути (0-59) в часа според локалното време на потребителя.

- **getMonth()** – Връща номера на месеца (0-11) в годината според локалното време на потребителя.

- **getSeconds()** – Връща броя секунди (0-59) в минутата според локалното време на потребителя.

- **getTime()** – Връща числова стойност на времето според локалното време на потребителя.

- **getYear()** – Връща годината (две цифри) според локалното време на потребителя.

- **getFullYear()** – Връща годината (четири цифри) според локалното време на потребителя. Работи само в по-нови браузъри.

- **setDate()** – Задава деня то месеца.

- **setHours()** – Задава броя часове.

- **setMinutes()** – Задава броя минути.

- **setMonth()** – Задава месеца.

- **setSeconds()** – Задава секундите.

- **setYear()** – Задава годината (две цифри).

- **setFullYear()** – Задава годината (четири цифри). Работи само в по-нови браузъри.

- **toLocaleString()** – Връща низ, представляващ датата според локалния формат.

Пример1: (извеждане на датата в страница)

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
<!--
var d = new Date();
var den=d.getDay();
var mes=d.getMonth();
var denmes=d.getDate();
var godina=d.getYear();
var dni = new Array(7);
dni[0] = "понеделник";
dni[1] = "вторник";
dni[2] = "сряда";
dni[3] = "четвъртък";
dni[4] = "петък";
dni[5] = "събота";
dni[6] = "неделя";
mes+=1;
if (godina<2000)
    godina+=1900
//-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#023FA8">
<P>Ден:
<SCRIPT language="JavaScript">
```

```
<!--
document.write(dni[den]+ ", "+mes+"/"+denmes+"/"+godina);
//-->
</SCRIPT>
</P>
</BODY>
</HTML>
```

Пример2: (извеждане на часа в лентата на състояние)

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
<!--
function chasovnik()
{
var d = new Date();
var chas=d.getHours();
var minuta=d.getMinutes();
var secunda=d.getSeconds();
if (chas<10)
     chas="0"+chas;
if (minuta<10)
     minuta="0"+minuta;
if (secunda<10)
     secunda="0"+secunda;
window.status=chas+":"+minuta+":"+secunda;
}
setInterval("chasovnik()",1000);
//-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#023FA8">
<P>Часът е изведен в лентата на състоянието</P>
</BODY>
</HTML>
```

XII. ОБРАБОТКА НА НИЗОВЕ

1. Какво представлява обектът String.

Този обект осигурява свойства и методи за получаване на информация за низове или за модифициране на низове. Създаването на обекта става по следния начин:

var t = new String(niz);

2. Някои свойства на обекта String.

- **length** – Връща дължината на низа, т. е. броя на символите в него.

3. Някои методи на обекта String.

- **bold()** – Добавя таговете и около низова стойност.
- **charAt()** – Определя кой символ се намира на дадена, подадена като параметър позиция в низа.

Пример (резултатът е буквата "t"):

```
var t="JavaScript";
window.alert("Последната буква е: "+charAt(t.length-1));
```

- **charCodeAt()** – Намира символния код на символ от дадена позиция в низа.
- **concat()** – Слепва два или повече низа.
- **fontcolor()** – Добавя таговете <FONT color="цвят"> и </FONT> около низова стойност за смяна на цвета.

Пример (резултатът е буквата "t"):

```
var t="Червен цвят";
window.alert(t.fontColor("red");
```

- **fontsize()** – Добавя таговете <FONT size="число"> и </FONT> около низова стойност за смяна на цвета.
- **italics()** – Добавя таговете и около низова стойност.
- **indexOf()** – Търси в низа символ, предаден като параметър. Ако бъде намерен, връща, връща позицията на първото срещане на символа, иначе връща -1.
- **lastIndexOf()** – Търси в низа символ, предаден като параметър. Ако бъде намерен, връща, връща позицията на последното срещане на символа, иначе връща -1.
- **link()** – Създава хипервръзки чрез използване на низа като текст на връзката и свързване с URL адреса, предаден като параметър.

- **replace()** – Открива дали един регулярен израз съвпада с низ и след това замества съвпадналия низ с нов низ.

- **search()** – Изпълнява търсене на съвпадение между регулярен израз и указан низ.

- **split()** – От низа създава масив от елементи, разделени от подадения като параметър символ.

- **sub()** – Добавя таговете <> и около низова стойност.

- **sup()** – Добавя таговете и около низова стойност.

- **substr()** – Отрязва и връща от даден низ символен низ от дадена позиция с дадена дължина, като двете числа се подават като параметри.

Пример (резултатът е "Script" – от символа с позиция 4 /започваме от 0/ вземаме 6 символа):

```
var t="JavaScript";
window.alert(t.slice(4,10));
```

- **substring()** – Отрязва и връща от даден низ символен низ от дадена позиция до дадена позиция, като двете числа се подават като параметри.

Пример (резултатът е "Script" – от символа с позиция 4 /започваме от 0/ до символа с позиция преди 10):

```
var t="JavaScript";
window.alert(t.slice(4,10));
```

- **toString()** – Конвертира в символен низ.

- **toLowerCase()** – Връща символния низ, изписан с малки букви.

Пример:

```
var t="JavaScript";
window.alert(t.toLowerCase());  // резултатът е с малки букви
```

- **toUpperCase()** – Връща символния низ, изписан с големи букви.

XIII. JAVASCRIPT И ФОРМИ

1. Какво представлява обектът за форма.

Всяка двойка тагове <FORM> и </FORM> в HTML документ създава елемент в масива forms в реда, в който се появяват в документа. Достъп до първата форма в документа става чрез document.forms[0] (номерирането започва както при всички масиви от 0).

Другият начин за достъп до форма става чрез името й, зададено в отварящия таг <FORM> по следния начин: document.ime_na_forma

Използват се две свойства length:

- document.forms[x].length – намира броя елементи на формата с индекс x;

- document.forms.length – намира броя форми в страницата.

2. Някои свойства на обекта Forms.

- **length** – Свойството е обяснено по-горе.

- **action** – Стойността на атрибута action в FORM тага.

- **method** – Стойността на атрибута method в FORM тага.

- **name** – Стойността на атрибута name в FORM тага.

- **target** – Стойността на атрибута target в FORM тага.

- **elements** – Масив, включващ елементи за всеки елемент (текстово поле, радиобутон и др.) от една HTML форма. Достъпът до всеки елемент на форма става с посочване на индекса на елемента (document.ime_na_forma.element[0]). Освен по този начин, до елемент от формата достъп може да има и чрез името му, зададено в съответния таг (document.ime_na_forma.ime_na_element).

Всеки елемент на форма има собствени свойства и методи.

Име на обекта (тип на елемента)	Свойства	Методи
button (бутон)	form, name, type, value	blur(), click(), focus()
checkbox (поле за отметка)	checked, defaultChecked, form, name, type, value	blur(), click(), focus()
hidden (скрито поле)	form, name, type, value	-
radio (радио бутон)	checked, defaultChecked, form, name, type, value	blur(), click(), focus()
reset (бутон за връщане в начално състояние)	form, name, type, value	blur(), click(), focus()
select (поле за избор)	form, name, options, selectedIndex, type	blur(), focus()
submit (бутон за изпращане)	form, name, type, value	blur(), click(), focus()
text (текстово поле)	defaultValue, form, name, type, value	blur(), focus(), select()
textarea (текстова област)	defaultValue, form, name, type, value	blur(), focus(), select()

== **checked** – Свойството има стойност true, ако полето или бутонът е избран и стойност false, ако не е.

== **defaultChecked** – Свойството има стойност true, ако полето за отметка или радиобутонът имат атрибут checked (т. е. зададено е избиране на съответното поле или бутон по подразбиране) и стойност false, ако нямат.

== **defaultValue** – Съдържа подразбиращата се стойност, която е зададена в атрибута value на тага на елемента във формата.

== **form** – Това свойство се използва често с ключовата дума this за обръщане към формата, съдържаща елемента.

Напр. ако във форма с име info_form искаме в текстовото поле с име favurl да променим стойността, това може вместо с document.info_form.favurl.value= "http://www.yahoo.com" да стане с по-краткото this.info_form.favurl.value= "http://www.yahoo.com".

== **name** – Съдържа стойността на атрибута name на дадения елемент.

== **options (масив)** – Свойството е масив, който съдържа елемент за всяка опция, изброена в поле за избор във формата. Индексните номера започват от 0, а всяка опция се поставя в масива в реда, в който е записана в HTML кода.

== **selectIndex** – Съдържа индексния номер на опцията (в масива options), която е избрана от потребителя. Ако е избрана първата опция, стойността е 0, ако е избрана втората – стойността е 1 и т. н.

== **type** – Съдържа стойността на атрибута type на дадения елемент от формата.

== **value** – Съдържа текущата стойност на елемента

== **blur()** – Метод за премахване на фокуса от елемента

== **click()** – Методът позволява в кода да се създаде събитие за щракване върху бутон.

== **focus()** – За придаване на фокус на даден елемент на формата.

== **select()** – Методът позволява да се избере автоматично съдържанието на текстово поле или текстова област. Полезно, ако сме задали стойност по подразбиране и искаме потребителят да може да изтрие бързо тази стойност.

3. Някои методи на обекта Forms.

- **reset()** – Връща формата в начално състояние

- **submit()** – Изпраща формата без потребителят да щрака върху бутона submit.

4. Валидизиране на форми с JavaScript

Валидизирането на форми само с JavaScript не винаги е разумно, тъй като потребителят може да е дезактивирал JavaScript в браузъра си. Но все пак валидизирането преди пращането към някакъв сървърен скрипт може да спести време и да намали натоварването на сървъра.

Когато потребителят щракне върху бутона submit, възниква събитие за изпращане, което може да се прихване от функцията onSubmit в отварящия таг <FORM>. Там може да се извика функция, която да изпълни проверка за валидност на данните. За да свърши работата си функцията, трябва, ако данните не минат валидирането да не се изпратят данните. За да стане това във функцията за обработка на събития onSubmit трябва да се постави return. Подразбиращото се действие (изпращането) ще се осъществи, ако се получи onSubmit=" return true" и няма да се изпълни при onSubmit=" return false".

Пример: (извършва се проверка дали във форма полета за име и e-mail са попълнени и дали в e-mail-а се съдържат символите "@" и ".")

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
<!--
function proverka()
{
var rezultat = true;
var ime = document.forma.txt1.value;
var poshta = document.forma.txt2.value;
if ((ime=="")||(ime==null))
{
alert("Попълнете име!");
rezultat = false;
}
else if ((poshta.indexOf("@")==-1)||(poshta.indexOf(".")==-1))
{
alert("Попълнете валиден e-mail!");
rezultat = false;
}
```

```
return rezultat;
}
//-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#023FA8">
<FORM name="forma" action="http://site.com/cgi-bin/form.cgi"
onSubmit="return proverka();">
Име: <INPUT type="text" name="txt1"><BR><BR>
E-mail: <INPUT type="text" name="txt2"><BR><BR>
<INPUT type="submit" value="submit"><BR>
</FORM>
</BODY>
</HTML>
```

Разбира се, тук проверката е опростена и например имейлът *a@a.a* ще мине валидацията, макар да не е валиден.