

JAVASCRIPT (1.3)

1	NOZIONI GENERALI.....	2
1.1	Introduzione	2
1.2	Struttura lessicale.....	2
1.3	Tipi di dati e valori	4
1.4	Dichiarazione di variabili, visibilità e passaggio.....	7
1.5	Espressioni e operatori	8
1.6	Alcuni oggetti	9
2	ISTRUZIONI.....	12
2.1	Istruzione “if”	12
2.2	Istruzione “switch”	12
2.3	Cicli	12
2.4	Inserimento di etichette e salti incondizionati	13
3	LE FUNZIONI.....	14
3.1	Definizione di una funzione	14
3.2	Importazione ed esportazione di funzioni (Navigator 4).....	15
3.3	Un primo esempio: Calcolo dei fattoriali	15
3.4	Un secondo esempio: Calcolo delle rate di un mutuo	16
4	OGGETTI.....	18
4.1	Definizione dell’oggetto e proprietà.....	18
4.2	Metodi.....	18
4.3	Prototipazione.....	19
4.4	Ereditarietà.....	20
5	ARRAY	21
6	ESPRESSIONI REGOLARI	22
6.1	Definizione di un’espressione regolare	22
6.2	Metodi di String per il confronto di pattern.....	23
6.3	Oggetto RegExp	24
7	INCORPORAMENTO DEL CODICE JAVASCRIPT	25
7.1	Incorporamento tramite Script.....	25
7.2	Incorporamento di file.....	25
7.3	Incorporamento degli archivi compressi	25
7.4	Incorporamento negli eventi.....	25
7.5	Collegamento del codice ad un evento (no Navigator)	25
7.6	Interprete in linea.....	26
7.7	Sintassi del foglio di stile JSS (Navigator 4)	26
7.8	Entità JavaScript (da Navigator 3) e commenti condizionali.....	26
8	PARTICOLARITA’	27
8.1	Finestre, Frames e Moduli.....	27
8.2	Sincronizzazione.....	27
8.3	COOKIES	28

1 NOZIONI GENERALI

1.1 Introduzione

JavaScript è un linguaggio interpretato orientato agli oggetti incorporato nei browser web più famosi, in modo che possa essere incapsulato nei testi HTML in maniera semplice e immediata. JavaScript, inoltre, implementa diversi tipi di oggetti tramite i quali si è in grado di manipolare l'intera struttura del web e delle finestre.

Sintatticamente è un linguaggio C-like, ma si discosta dal suo progenitore per quanto riguarda la tipizzazione (è debolmente tipizzato), l'ereditarietà e l'uso degli oggetti. Inoltre JavaScript non è Java. Il suo nome originariamente era LiveScript, modificato successivamente in quello attuale principalmente per questioni di mercato.

1.2 Struttura lessicale

Poche regole sono necessarie per la composizione di programmi lessicalmente corretti. Le principali sono già note ai programmatori C, Pascal e linguaggi simili.

1. JavaScript è case-sensitive; “Casa” sarà diverso da “casa”, “CASA”, “cAsa”, ecc.

2. I commenti si possono inserire in diversi modi.

Commenti di una riga: vengono preceduti dalla doppia barra (//);

Commenti di più righe: il commento inizia con una barra ed un asterisco (/*) e termina con un asterisco ed una barra (*).

Ad esempio le righe seguenti sono tutti commenti JavaScript corretti.

```
//questo è un ipotetico commento in un programma JavaScript
/*ma anche quest'altro è
un commento JavaScript valido */
```

3. Letterali: un letterale è un valore che compare esplicitamente all'interno di un programma; ad esempio sono letterali i seguenti numeri e testi:

1	1.2	"ciao a tutti"	'ciao a gli altri'	true	false	null
---	-----	----------------	--------------------	------	-------	------

4. Come si può notare un testo può essere delimitato indifferentemente tra virgolette (“) e apici (‘).

5. Identificatori: In JavaScript gli identificatori sono dei nomi utilizzati per battezzare variabili e funzioni.

6. Parole riservate: JavaScript possiede una serie di parole riservate che non possono essere utilizzate come identificatori all'interno dei programmi.

Parole chiave riservate da JavaScript

break	do	function	null	typeof
case	else	if	return	var
continue	export	import	switch	void
default	false	in	this	while
delete	for	new	true	with

Parole chiave riservate da ECMA

catch	const	enum	finally	throw
class	debugger	extends	super	try

Parole chiave riservate da Java

abstract	final	int	private	synchronized
boolean	float	interface	protected	throws
byte	goto	long	public	transient
char	implements	native	short	
double	instanceof	package	static	

Identificatori da evitare

alert	escape	Math	parseFloat	setTimeout
arguments	eval	menubar	parseInt	status
Array	find	moveBy	personalbar	statusbar
blur	focus	moveTo	print	stop
Boolean	frames	name	prompt	String
callee	Function	NaN	prototype	toolbar
caller	history	netscape	RegExp	top
captureEvents	home	Number	releaseEvents	toString
clearInterval	Infinity	Object	resizeBy	unescape
clearTimeout	innerHeight	open	resizeTo	unwatch
close	innerWidth	opener	routeEvent	valueOf
closed	isFinite	outerHeight	scroll	watch
confirm	isNaN	outerWidth	scrollbars	window
constructor	java	Packages	scrollBy	
Date	length	pageXOffset	scrollTo	
defaultStatus	location	pageYOffset	self	
document	locationbar	parent	setInterval	

1.3 Tipi di dati e valori

Il tipo di un dato può essere primitivo o composto. In particolare si ha:

Tipo				
Primitivo			Composto	
Numero	Stringa	Booleano	Oggetto	Array

Le funzioni rappresentano un tipo di dato a sé.

1.3.1 Numeri

Letterali interi: `0|((1-9)(0-9)*)`
 Letterali ottali: `0(0-7)*`
 Letterali esadecimali: `0(x|X)(0-9|a-f|A-F)*`
 Letterali a virgola mobile: `[cifre][.cifre][(E|e)[(+|-)]cifre]`

Valori numerici speciali:
 Number.MAX_VALUE Massimo numero rappresentabile
 Number.MIN_VALUE Minimo numero negativo rappresentabile
 Number.NaN Valore speciale: Not a number
 Number.POSITIVE_INFINITY Valore speciale che rappresenta “più infinito”
 Number.NEGATIVE_INFINITY Valore speciale che rappresenta “meno infinito”

Assegnamento: `NomeVariabile = ValoreNumerico;`
 Definizione di variabile: `var NomeVariabile [= ValoreNumerico];`

1.3.2 Stringhe

Letterali stringa: Qualunque sequenza di caratteri chiusa tra apici singoli o doppi
 (“...” oppure ‘...’)

Sequenze di escape nei letterali stringa:

<code>\b</code>	Backspace
<code>\f</code>	Avanzamento pagina (form feed)
<code>\n</code>	Avanzamento riga (new line)
<code>\r</code>	Ritorno a capo (carriage return)
<code>\t</code>	Tabulazione
<code>\'</code>	Apostrofo o singolo apice
<code>\"</code>	Doppio apice
<code>\\</code>	Backslash
<code>\XXX</code>	Il carattere in codifica Latin-1 specificato da tre cifre ottali
<code>\xXX</code>	Il carattere in codifica Latin-1 specificato da due cifre esadecimali

Sequenze di escape Unicode per caratteri speciali

<code>\u0009</code>	Tabulazione
<code>\u000B</code>	Tabulazione verticale
<code>\u000C</code>	Avanzamento pagina (form feed)
<code>\u0020</code>	Spazio
<code>\u000A</code>	Avanzamento riga (line feed)
<code>\u000D</code>	Ritorno a capo (carriage return)
<code>\u000b</code>	Backspace
<code>\u0009</code>	Tabulazione orizzontale
<code>\u0022</code>	Doppio apice
<code>\u0027</code>	Singolo apice
<code>\u005C</code>	Backslash

Assegnamento: `NomeVariabile = (“stringa”|’stringa’)`

1.3.3 Valori Booleani

Valore booleano:	<code>(true false)</code>
Verifica booleana	<code>a == b</code>

1.3.4 Funzioni

Definizione di una funzione

```
function NomeFunzione(param1, param2, ..., paramN)
{
    istruzione1;
    istruzione2;
    return Risultato;
    ...
    istruzioneN;
}
```

Esempio: Calcolo del quadrato di un numero

```
function square(x)
{
    return x * x;
}
```

Letterale funzione (v 1.1) `var square = new Function("x", "return x * x;");`

Letterale funzione (v 1.2) `var square = function(x) { return x * x; };`

1.3.5 Oggetti

L'utilizzo degli oggetti è il consueto:

Assegnamento ad una proprietà:	<code>NomeOggetto.NomeProprietà = Valore;</code>
Riferimento ad una proprietà:	<code>Valore = NomeOggetto.NomeProprietà;</code>
Esecuzione di un metodo:	<code>NomeOggetto.NomeMetodo(parametri);</code>
Oggetti con Array di proprietà:	<code>NomeOggetto["NomeProprietà"];</code>

Creazione di oggetti:

`var NomeOggetto = new CostruttoreOggetto;`

Letterali oggetto (esempio):

DEFINIZIONE CONVENZIONALE

```
var point = new Object( );
point.x = 2.3;
point.y = -1.2;
```

LETTERALE OGGETTO

```
var point = {x:2.3, y:-1.2}
```

Letterali oggetto annidati

```
var rectangle = { upperleft : {x:3, y:2}
                  lowerright: {x:4, y:5}};
```

l'assegnamento sarà del tipo:

```
rectangle.upperleft.x = 4;
```

1.3.6 Array

Dato il loro impiego intuitivo ed analogo ad altri linguaggi si procede subito con un esempio.

```
var a = new Array ( );
a[0]=1.2;
a[1]="Stringa di esempio";
a[2]=true;
a[3]={x:1, y:3};
```

Come si può notare anche gli array sono debolmente tipizzati.

Dalla versione 1.2 è possibile utilizzare i letterali array come segue.

```
var a = [1.2, "Stringa di esempio", true, {x:1, y:3} ];
```

Eventualmente anche annidati.

```
var matrice = [ [1, 2, 3] , [4, 5, 6] , [7, 8, 9] ];
```

1.3.7 Valori speciali

null
undefined
NaN
...

1.4 Dichiarazione di variabili, visibilità e passaggio

Le variabili sono debolmente tipizzate e si dichiarano in uno dei modi seguenti.

```
var NomeVariabile;
var NomeVariabile1, NomeVariabile2;
var Stringa = "valore" ;
```

Nel caso in cui non venga utilizzata l'istruzione "var", il primo utilizzo della variabile corrisponde alla sua creazione come "GLOBAL" (si veda oltre). E' da ricordare che la definizione di una variabile ha valore nell'intero ambito in cui viene definita, comprese le istruzioni precedenti.

```
var scope = "globale";
function f( )
{
    alert (scope);           //scrive "undefined"
    var scope = "locale";
    alert (scope);           //scrive "locale"
}
f ( );
```

Per quanto riguarda il passaggio dei parametri, i tipi primitivi sono passati per valore, gli altri per riferimento.

Esempio di passaggio per valore

```
var a = 3;
var b = a;
a = 4;
alert (b); // scrive "3", cioè il valore passato all'assegnamento
```

Esempio di passaggio per riferimento

```
var a = [1, 2, 3];
var b = a;
a [0] = 99;
alert (b); // scrive "[99, 2, 3]", cioè il valore attuale di "a"
```

1.5 Espressioni e operatori

Verifica di uguaglianza: ==
 Verifica di identità: ===

Operatori logici

Operatore	Logico	A livello di bit
AND	&&	&
OR		
NOT	!	~
XOR		^

- Post incremento: NomeVar + +
- Post decremento: NomeVar - -
- Pre incremento: + + NomeVar
- Pre decremento: - - NomeVar
- Somma e assegnamento: + =
- Operatore condizionale: ? :
- esempio:* a ? b : c → iif (a , b , c) → if a then b else c
- Operatore typeof(a) restituisce in una stringa il tipo della variabile
- Costruttore di oggetti: new NomeOggetto
- Distruzione di oggetti: delete NomeVariabile | Proprietà
- Operatore "vuoto": void
- esempio:* parola

1.6 Alcuni oggetti

JavaScript è composto da molti oggetti che consentono di controllare molteplici aspetti della macchina e del browser. Ci limitiamo ad elencarne alcuni.

1.6.1 Oggetto Document

Rappresenta un documento HTML. Per maggiori informazioni si veda anche `window.document`.

Proprietà:

<code>alinkColor;</code>	<code>domain;</code>	<code>lastModified;</code>	<code>referrer;</code>
<code>anchors[];</code>	<code>embeds[];</code>	<code>linkColor;</code>	<code>title;</code>
<code>applets[];</code>	<code>fgColor;</code>	<code>links[];</code>	<code>URL;</code>
<code>bgColor;</code>	<code>forms[];</code>	<code>location;</code>	<code>vLinkColor;</code>
<code>cookie;</code>	<code>images[];</code>	<code>plugins[];</code>	

Proprietà per Navigator 4:

<code>classes;</code>	<code>ids;</code>	<code>tags;</code>
<code>height;</code>	<code>layers[];</code>	<code>width;</code>

Proprietà per Internet Explorer 4:

<code>activeElement;</code>	<code>charset;</code>	<code>defaultCharset;</code>	<code>parentWindow;</code>
<code>all[];</code>	<code>children[];</code>	<code>expando;</code>	<code>readyState;</code>

Metodi:

<code>clear();</code>	<code>close();</code>	<code>open();</code>	<code>write();</code>	<code>writeln();</code>
-----------------------	-----------------------	----------------------	-----------------------	-------------------------

Metodi in Navigator 4:

<code>captureEvents();</code>	<code>contextual();</code>	<code>getSelection();</code>	<code>releaseEvents();</code>	<code>routeEvent();</code>
-------------------------------	----------------------------	------------------------------	-------------------------------	----------------------------

Metodi in Internet Explorer 4:

<code>elementFormPoint();</code>

1.6.2 Oggetto Window

Proprietà:

closed	length	offscreenBuffering	status
defaultStatus	location	opener	top
document	Math	parent	window (self)
frames[]	name	screen	
history	navigator	self	

Proprietà di Navigator:

cripto	menubar	pageXOffset	scrollbars
innerHeight	netscape	pageYOffset	statusbar
innerWidth	outerHeight	personalbar	sun
java	outerWidth	screenX	toolbar
locationbar	Packages	screenY	

Proprietà di Internet Explorer:

clientInformation
event

Metodi:

alert()	confirm()	prompt()	scrollTo()
blur()	focus()	resizeBy()	setInterval()
clearInterval()	moveBy()	resizeTo()	setTimeout()
clearTimeout()	moveTo()	scroll()	
close()	open()	scrollBy()	

Metodi di Navigator:

atob()	find()	releaseEvents()	stop()
back()	forward()	routeEvent()	
btoa()	handleEvent()	setHotkeys()	
captureEvents()	home()	setResizable()	
disableExternalCapture()	print()	setZOptions()	

Metodi di Internet Explorer:

navigate()

Gestori di eventi:

onblur	onerror	onload	onresize
ondragdrop	onfocus	onmove	onunload

1.6.3 Oggetto Math

L'oggetto Math consente di usufruire di una serie di funzioni e costanti matematiche predefinite.

Costanti:

Math.E	Math.LN2	Math.LOG2E	Math.SQRT1_2
Math.LN10	Math.LOG10E	Math.PI	Math.SQRT2

Funzioni statiche:

abs()	ceil()	max()	sin()
acos()	cos()	min()	sqrt()
asin()	exp()	pow()	tan()
atan()	floor()	random()	
atan2	log()	round()	

1.6.4 Oggetto Date

Costruttore:

```
new Date( );
new Date (millisecondi);
new Date (StringaData);
new Date (anno, mese, giorno, ore, minuti, secondi, ms);
```

Metodi:

getDate();	getUTCDate();	setFullYear();	setUTCHours();
getDay();	getUTCDay();	setHours();	setUTCMinutes();
getFullYear();	getUTCFullYear();	setMilliseconds();	setUTCMonth();
getHours();	getUTCHours();	setMinutes();	setUTCSeconds();
getMilliseconds();	getUTCMilliseconds();	setMonth();	setYear();
getMinutes();	GetUTCMinutes();	setSeconds();	toGMTString();
getMonth();	getUTCMonth();	setTime();	toLocaleString();
getSeconds();	getUTCSeconds();	setUTCDate();	toString();
getTime();	getFullYear();	setUTCFullYear();	toUTCString();
getTimezoneOffset();	setUTCMilliseconds();	setDate();	valueOf();

Metodi statici:

```
Date.parse();
Date.UTC();
```

2 ISTRUZIONI

In questa sezione vengono esposte sintassi e semantica dei principali costrutti JavaScript.

2.1 Istruzione “if”

Il classico costrutti condizionale è il seguente:

```
if (espressione) istruzione1
else if istruzione2
else istruzione3
```

2.2 Istruzione “switch”

Le condizioni multiple vengono gestite come segue:

```
switch ( NomeVariabile)
{
case PrimoCaso:
    IstruzionePrimoCaso;
case SecondoCaso:
    IstruzioneSecondoCaso;
...
default:
    IstruzioneDefault;
}
```

2.3 Cicli

In JavaScript è possibile implementare i cicli in maniera differente a seconda delle necessità di elaborazione.

2.3.1 Ciclo “while”

Il ciclo “while” puro è a condizione iniziale:

```
while (espressione)
    istruzione;
```

2.3.2 Ciclo “do... while”

Questo ciclo sostituisce il costrutto “repeat” a condizione finale:

```
do
    istruzione;
while (espressione);
```

2.3.3 Ciclo “for”

Il ciclo a conteggio possiede una sintassi molto simile a quella del C:

```
| for (inizializza; test; incremento)
  |   istruzione;
```

2.3.4 Ciclo “for...in”

Il ciclo “for” può essere esteso in modo da poter enumerare gli elementi di un insieme:

```
| for (Variabile in Oggetto)
  |   istruzione;
```

2.3.5 Uscita dai cicli

Per forzare l’uscita da un ciclo o da un’istruzione “switch” è sufficiente eseguire l’istruzione seguente:

```
| break;
```

2.3.6 Istruzione “continue”

L’istruzione “continue” consente (all’interno dei cicli) di ripetere il test d’iterazione. Sono possibili due utilizzi fondamentali:

```
| continue;
| continue NomeEtichetta //nel caso di cicli annidati
```

2.4 Inserimento di etichette e salti incondizionati

L’inserimento di un’etichetta avviene in maniera semplice facendo seguire all’etichetta stessa i due punti e un’eventuale istruzione:

```
| NomeEtichetta: istruzione;
```

E’ anche possibile saltare al termine di un’istruzione specificata da un’etichetta con la sintassi seguente:

```
| break NomeEtichetta;
```

3 LE FUNZIONI

3.1 Definizione di una funzione

```
function NomeFunzione ( [arg1 [,arg2 [,... ,argN]])
{
  istruzioni;
  return Valore      //restituzione del valore;
}
```

La definizione della funzione avviene al momento del parsing e non in fase di esecuzione. Ciò comporta alcuni comportamenti anomali come il seguente:

```
alert ( f ( 4 ) ); // visualizza 16
var f = 0; // ridefinisce "f"
function f ( x )
{
  return x*x;
}
alert ( f ); //visualizza zero
```

Come già esposto la creazione di una funzione è anche possibile attraverso l'utilizzo dell'apposito costruttore.

```
| var f= new Function("x", "y", "return x*y;");
```

o, ancora, attraverso il letterale funzione.

```
| var f=function(x,y) {return x*y;};
```

Ogni funzione, infine, possiede degli oggetti e proprietà interessanti tra cui:

Oggetto `arguments` → è l'oggetto che contiene i riferimenti agli argomenti della funzione. In particolare alcune delle sue proprietà sono:

<code>arguments.length</code>	numero di argomenti disponibili
<code>arguments[i]</code>	argomento "i-esimo" della funzione
<code>arguments.callee</code>	è un riferimento a sé stessa
<code>arguments.caller</code>	è un riferimento al contesto di chiamata

Ad esempio per chiamare la funzione chiamante quella corrente è necessario un riferimento come quello che segue:

```
| arguments.caller.callee
```

3.2 Importazione ed esportazione di funzioni (Navigator 4)

E' possibile importare ed esportare funzioni tra un contesto JavaScript ed un altro.

```

<LAYER name="layer1" bgColor="yellow">
...
<SCRIPT LANGUAGE="JavaScript1.2">
    function sposta(x, y)
        {this.x=x ; this.y=y ;}
    export sposta ;
</SCRIPT>
</LAYER>
...
<SCRIPT LANGUAGE="javascript1.2">
    import document.layer1.* ;
    var x=0 ; y=0 ;
    setInterval (function(){sposta(x,y); x=(x+5)%200; y=(y+5)%200;},100) ;
</SCRIPT>

```

3.3 Un primo esempio: Calcolo dei fattoriali

```

<HTML>
<BODY>
<SCRIPT LANGUAGE="javascript">
document.writeln ("<h1>CALCOLO DELLA TAVOLA DEI FATTORIALI</h1>");
document.writeln ("<font size="3">");
for(contatore=1,fattoriale=1;contatore<15;contatore++,fattoriale*=contatore)
{
    document.write( contatore + "! = " + fattoriale + " <br>");
}
document.writeln ("</font>");
</SCRIPT>
<BODY>
<HTML>

```

3.4 Un secondo esempio: Calcolo delle rate di un mutuo

```

<!--
 modulo HTML per il calcolo delle rate del mutuo. Inserendo Importo,
 Tasso e Periodo di restituzione il programma calcola la rata mensile,
 il rimborso totale e l'interesse totale -->
<FORM NAME="modulodati" >
  <TABLE>
    <TR> <B>INSERIMENTO DATI</B> </TR>
    <TR> <TD>Importo del mutuo</TD>
      <TD><INPUT      TYPE="text" NAME="importo"
        SIZE="13"    onChange="calculate()"></TD> </TR>
    <TR> <TD>Tasso annuo di interesse</TD>
      <TD><INPUT      TYPE="text" NAME="tasso"
        SIZE="13"    onChange="calculate()"></TD> </TR>
    <TR> <TD>Durata in anni del rimborso</TD>
      <TD><INPUT      TYPE="text" NAME="anni"
        SIZE="13"    onChange="calculate()"></TD> </TR>
    <TR> <TD>Premi il pulsante per calcolare:</TD>
      <TD><INPUT      TYPE="button" VALUE="Calcola"
        onClick="calculate()"></TD> </TR>
    <TR> <TD><B>INFORMAZIONI SUL PAGAMENTO</B></TD> </TR>
    <TR> <TD>Rata mensile:</TD>
      <TD><INPUT TYPE="text" NAME="rata" SIZE="13"></TD> </TR>
    <TR> <TD>Rimborso totale:</TD>
      <TD><INPUT TYPE="text" NAME="rimborso" SIZE="13"></TD> </TR>
    <TR> <TD>Interesse totale sul capitale:</TD>
      <TD><INPUT TYPE="text" NAME="interessesetotale" SIZE="13"></TD><TR>
  </TABLE>
</FORM>

```

```

<!--
 ora il codice javascript che implementa l'algoritmo di calcolo
-->
function calcolate()
{
  var v_importo = document.modulodati.importo.value;
  var v_tasso   = document.modulodati.tasso.value / 100 / 12;
  var v_rate    = document.modulodati.anni.value * 12;
  var v_pot     = Math.pow(1 + v_tasso, v_rate);
  var v_mese    = (importo * v_pot * v_tasso) / (v_pot - 1);

  //verifica del risultato
  if ( !isNaN(v_mese) &&
      (v_mese != Number.POSITIVE_INFINITY) &&
      (v_mese != Bomber.NEGATIVE_INFINITY) )
  {
    document.modulodati.rata           = round(v_mese);
    document.modulodati.rimborso       = round(v_mese * v_rate);
    document.modulodati.interesstotale =
      round((v_mese * v_rate) - v_importo);
  }
  //dati non validi
  else
  {
    document.modulodati.rata           = "";
    document.modulodati.rimborso       = "";
    document.modulodati.interesstotale = "";
  }
}

function round(x)
{
  return Math.round(x*100)/100;
}
</SCRIPT>

```

4 OGGETTI

4.1 Definizione dell'oggetto e proprietà

```
| var obj = new Object();
```

oppure

```
| var Omer =
| {
|     nome: "Omer Simpson",
|     età: 34,
|     sposato = true,
|     e-mail = "omer@simpson.com"
| }
```

4.2 Metodi

All'interno di un metodo l'oggetto "this" si riferisce all'oggetto del metodo stesso.

```
| //prima si creano le funzioni da utilizzare come funzioni
| function rec_area() {return this.width * this.height;};
| function rec_perimetro() {return 2*(this.width+this.height)};
| function rec_setsize(w, h) {this.width=w, this.height=h};
|
| //si definisce, poi, il costruttore dell'oggetto
| function rettangolo(w,h)
| {
|     //proprietà...
|     this.width=w;
|     this.height=h;
|     //e metodi
|     this.area=rec_area;
|     this.perimetro=rec_perimetro;
|     this.setsize=rec_setsize;
| }
|
| //L'oggetto è pronto per l'uso
| var r = new rettangolo(5,3);
| var a = r.area();
| var p = r.perimetro();
| ...
```

4.3 Prototipazione

Dalla versione 1.1 di JavaScript è possibile definire degli oggetti attraverso il metodo della prototipazione. Bisogna partire dal presupposto che esista un oggetto “prototype” per la classe che andiamo a costruire. Ad esempio per la classe “parallelepipedo” esisterà l’oggetto “parallelepipedo.prototype”.

Si procede come segue.

```
// Si costruisce il metodo costruttore della classe
function Parallelepipedo(x, y, z)
{
    this.x = x;
    this.y = y;
    this.z = z;
}

//definiamo un metodo (volume del parallelepipedo)
function para_volume() {return this.x*this.y*this.z;};

//assegnamo il metodo ad una proprietà del prototipo
Parallelepipedo.prototype.volume = para_volume;

//oppure tutto in un passaggio
Parallelepipedo.prototype.perimetro = new Function
    ("return 4*(this.x*this.y*this.z)");

//Definizione di un'istanza dell'oggetto
var pa = new Parallelepipedo(1, 2, 3);
var v = pa.volume();
var p = pa.perimetro();
```

4.4 Ereditarietà

Per definire una classe di oggetti che erediti le caratteristiche da una superclasse è sufficiente assegnare all'oggetto prototipo della sottoclasse un'istanza della superclasse.

```
//costruttore per la classe
function complex(reale, immaginario)
{
  this.x = reale;
  this.y = immaginario;
}
//posso aggiungere proprietà e metodi a volontà
...
//definisco il costruttore per la SOTTOCLASSE
function complessissimo(reale, immaginario)
{
  this.x = reale;
  this.y = complesso;
}

//imponiamo la relazione di ereditarietà
complessissimo.prototype = new prototipo(0, 0);

//successivamente si possono aggiungere altri metodi
complessissimo.prototype.swap = function()
{
  var tmp = this.x;
  var this.x = this.y;
  var this.y = tmp;
};

//potrebbe essere necessario ridefinire il costruttore dell'oggetto
complessissimo.prototype.constructor = complessissimo;
```

5 ARRAY

Creazione di Array:

```
var a = new Array();
var a = new Array(5, 4, 3, 2, 1, "prova");
var a = new Array(DimensioneArray);
```

Letterale Array:

```
var a = [5, 4, 3, 2, 1];
var a = ['a', true, 4];
```

Bisogna ricordarsi che il 1° elemento ha indice zero.

Array composti:

```
var a = [ [1, {x:1, y:2}], [2, {x:3, y:4}] ];
```

Letture/Scrittura :

```
c = a[indice];
a[indice] = c;
```

La proprietà `length` restituisce il massimo indice più uno. Esistono, inoltre, molti metodi utili per la gestione degli array; tra cui:

Versione JS	Metodo	Note
1.1	<code>join(stringa)</code>	converte gli elementi in una stringa e li concatena. Tra un elemento e l'altro inserisce una virgola o, se specificato, "stringa"
1.1	<code>reverse()</code>	inverte l'ordine degli elementi di un array
1.1	<code>sort(f(a,b))</code>	ordina gli elementi in ordine alfabetico o, se specificato, secondo l'algoritmo implementato in f: f<0 → "a" è precedente a "b" f=0 → "a" è equivalente a "b" f>0 → "a" è successivo a "b"
1.2	<code>concat()</code>	"appiattisce" un array e lo accoda a quello esistente; ad esempio: var a = [1,2,3]; a.concat([4,5],[6,7]) → [1,2,3,4,5,6,7] a.concat(4,[5,[6,7]]) → [1,2,3,4,5,[6,7]]
1.2	<code>slice(i,f)</code>	restituisce un sottoarray che parte da "i" e finisce a "f"
1.2	<code>splice()</code>	Inserisce o rimuove elementi; ad esempio: var a = [1,2,3,4,5,6,7,8]; a.splice(4) restituisce [5,6,7,8] ma a==[1,2,3,4] a.splice(1,2) restituisce [2,3] ma a==[1,4,5,6,7,8]
1.3	<code>push(el1,el2,...)</code>	accoda agli array gli elementi specificati
1.3	<code>pop()</code>	restituisce ed elimina l'ultimo elemento di un array
no IE4	<code>unshift()</code>	inserisce un elemento a sinistra di un array
no IE4	<code>shift()</code>	toglie un elemento a sinistra di un array
1.1	<code>toString()</code>	<code>["a", "b", "c"].toString = "abc"</code>

6 ESPRESSIONI REGOLARI

6.1 Definizione di un'espressione regolare

Per implementare le espressioni regolari è necessario l'oggetto `RegExp()`; JavaScript 1.2 copre la sintassi di Perl 4 mentre JavaScript 1.3 coprirà la sintassi di Perl 5. I delimitatori delle espressioni regolari sono le singole barre (slash o "/"). Segue una definizione tipo di un'espressione regolare.

```
var pattern = /s$/;
var pattern = new RegExp("s$");
```

Classi di caratteri:

[abc] "a" OR "b" OR "c"
 [^abc] NOT ("a" OR "b" or "c")

In particolare:

[...] qualsiasi carattere tra parentesi
 [^...] qualsiasi carattere non tra parentesi
 . qualsiasi carattere tranne "new line"; equivale a [^\n]
 \w qualsiasi carattere alfanumerico; equivale a [a-zA-Z0-9]
 \W qualsiasi carattere non alfanumerico; equivale a [^a-zA-Z0-9]
 \s qualsiasi carattere di spazio vuoto; equivale a [\t\n\r\f\v]
 \S qualsiasi carattere di spazio non vuoto; equivale a [^\t\n\r\f\v]
 \d qualsiasi cifra; equivalente a [0-9]
 \D qualsiasi carattere non cifra; equivalente a [^0-9]
 [\b] backspace letterale

Ripetizioni:

[\d\d\d] tre cifre esatte
 ^d{2,4}/ coincidenza tra 2 e 4 cifre

In particolare:

{n,m} coincidenza con l'espressione precedente almeno "n" volte ma non più di "m"
 {n,} coincidenza con l'espressione precedente almeno "n" o più volte
 {} coincidenza con l'espressione precedente esattamente "n" volte
 ? coincidenza per zero o una volta
 + coincidenza per una o più volte
 * coincidenza per zero o più volte
 | alternativa (/ab|cd/ indica "ab" OR "cd")

Riferimento alle sottoespressioni

E' possibile assegnare degli attributi a parte delle espressioni regolari in modo che sia possibile compiere un parziale parsing contestuale. Gli attributi vengono assegnati con numeri preceduti dalla barra inversa (back slash). Ad esempio:

```
/( [ \ " ] [ ^ \ " ] * \ 1 /
```

il "\1" indica che dovrà ripetersi esattamente l'istanza della prima sottoespressione ([\ "]). Ciò significa che se il parser ha incontrato in singolo apice dovrà ripetersi necessariamente il singolo apice; lo stesso vale per il doppio apice.

Caratteri di ancora e attributi:

Infine, è possibile inserire caratteri che si colleghino a determinate parti dell'espressioni o attributi che consentano di specificare il tipo di parsine. Nella tabella seguente sono riportati i principali elementi.

Caratteri di ancora

^	inizio riga
\$	fine riga
\b	fine parola
\B	interno di una parola

Attributi

/i	non case sensitive
/g	confronto globale

Le espressioni su più linee vengono accettate se la proprietà "multiline" dell'espressione regolare è impostata a "true".

6.2 Metodi di String per il confronto di pattern

METODO

UTILIZZO

<code>S.search (RegExp);</code>	restituisce la posizione del carattere iniziale della prima sottostringa coincidente oppure "-1". Non supporta ricerche globali. Ad esempio <code>"ProvaEspressioneRegolare".search(/Espres/)</code> restituisce 5
<code>S.replace (RegExp,T);</code>	Questo metodo sostituisce le occorrenze di "RegExp" con la stringa "T" all'interno della stringa "S". Se è specificato l'attributo "g" la sostituzione è globale, altrimenti si ferma alla prima istanza. Ad esempio <code>S.replace(/javascript/gi,"JavaScript")</code> sostituisce tutte le occorrenze di JavaScript a prescindere dalle maiuscole all'interno di "S" sostituendole con quella corretta (solo "J" ed "S" maiuscole). Inoltre se nella stringa sostitutiva compare un carattere "\$" seguito da una cifra, questo metodo sostituisce tali due caratteri con il testo coincidente con la sottoespressione specificata. Ad esempio si voglia sostituire il carattere tilde alle virgolette di una citazione. <code>S.replace(/"([^\"]*)" /g, "~\$1~");</code>
<code>S.match (RegExp);</code>	restituisce un array contenente tutte le istanze positive del confronto eseguito tra "S" e l'espressione regolare "RegExp". Ad esempio

`S.split(RegExp);` "1 + 2 = 3".match(/\\d+/g) restituisce ["1", "2", "3"];
 suddivide in un array di sottostringhe "S" utilizzando l'argomento come separatore. Già visto come metodo, implementa, come separatore, le espressioni regolari.

6.3 Oggetto RegExp

Costruttore:

```
var Espressione = new RegExp("espressione regolare", "attributi");
```

Metodi

`.test(Stringa)` restituisce "true" se c'è occorrenza, "false" altrimenti
`.exec(Stringa)` restituisce "Null" se non c'è occorrenza, altrimenti un array delle occorrenze

Proprietà istanza

`.global` restituisce "true" se nell'espressione regolare è definito l'attributo "g"
`.ignoreCase` restituisce "true" se nell'espressione regolare è definito l'attributo "i"
`.lastIndex` con `.global=true` restituisce la posizione di partenza dell'occorrenza di sola lettura, contiene il testo dell'espressione regolare

Proprietà statiche

`.$n` n-esima sottoespressione
`.input` testo da confrontare se non ne viene specificato nessun altro
`.lastMatch` ultimo confronto positivo
`.lastParen` ultima sottoespressione
`.multiline` riconoscimento di modelli su più linee
`.leftContext` il testo a sinistra del testo corrispondente
`.rightContext` il testo a destra del testo corrispondente

7 INCORPORAMENTO DEL CODICE JAVASCRIPT

Esistono diversi modi di incorporare o eseguire codice JavaScript all'interno di una pagina web. Alcuni dipendono dalla tipologia di browser, altri dal risultato che si vuole ottenere.

7.1 Incorporamento tramite Script

```
...
<SCRIPT LANGUAGE="javascript">
...
</SCRIPT>
...
```

7.2 Incorporamento di file

```
...
<SCRIPT SRC="urlfile.js"> </SCRIPT>
...
```

7.3 Incorporamento degli archivi compressi

```
...
<SCRIPT ARCHIVE="urlfile.jar" SRC="fileinterno.js"> </SCRIPT>
...
```

7.4 Incorporamento negli eventi

```
...
<INPUT TYPE="checkbox"
  NAME="opzione" VALUE="ignore_case"
  onClick="ignore_case=this.checked;"
>
...
```

7.5 Collegamento del codice ad un evento (no Navigator)

Non valido per Navigator

```
...
<INPUT TYPE="checkbox"
  NAME="opzione" VALUE="ignore_case"
>
...
<SCRIPT FOR"opzione" EVENT="onClick">
  ignore_case=this.checked;
</SCRIPT>
...
```

Valido anche per Navigator

```

<FORM NAME="modulo1">
  <input type="button" value="Pulsante" name="b3">
</FORM>
...
<SCRIPT LANGUAGE="javascript">
  document.modulo1.b3.onclick = new Function("alert('prova');");
</SCRIPT>

```

7.6 Interprete in linea

E' possibile inserire righe di codice JavaScript direttamente nella barra degli indirizzi dei browser Internet Explorer e Netscape. La riga deve essere preceduta da "javascript:". In particolare in Netscape inserendo solo l'identificativo di protocollo si attiverà l'interprete in linea.

```
| javascript:alert("ciao");
```

7.7 Sintassi del foglio di stile JSS (Navigator 4)

```

<STYLE TYPE="text/javascript">
  tags.H1.fontstyle = "bold";
  tags.H1.color      = "red" ;
</STYLE>

```

7.8 Entità JavaScript (da Navigator 3) e commenti condizionali

Il codice può comparire nei valori degli attributi HTML

```

//entità
<INPUT TYPE="text" NAME="casella" VALUE="defaults.cognome">
...
//commenti condizionali
<!--&{navigator.platform=="win95"};
<SCRIPT>
\& ...
</SCRIPT>
-->

```

8 PARTICOLARITA'

8.1 Finestre, Frames e Moduli

Per la gestione delle finestre e dei frames è necessario principalmente l'oggetto window e due dei suoi oggetti: "navigator" e "screen"

Precedentemente sono già stati esposti i metodi e le proprietà dell'oggetto "window". In particolare l'array "frames" di "window" consente di accedere a tutti i frames disponibili (magari unitamente all'oggetto "parent").

8.1.1 Oggetto Navigator

Proprietà

.appName	.language (NN4)	.plugins[]	.userLanguage (IE4)
.appVersion	.mimeTypes[]	.systemLanguage (IE4)	
	.platform	.userAgent	

Metodi

.javaEnabled()	.preference() (NN4)	.tainEnabled() (deprecato)
.plugins.refresh()	.savePreferences() (NN4)	

8.1.2 Oggetto Screen (1.2)

Proprietà

.availHeight	.availTop	.colorDepth	.pixelDepth
.availLeft	.availWidth	.height	.width

E' particolarmente accedere agli oggetti di un modulo nel seguente modo (ovunque esso sia, in qualunque frame, ci riferiremo ad esso come "document"): `document.forms[...].elements[...]`.

Naturalmente per un form chiamato "modulo1" è possibile un riferimento del tipo:

`document.modulo1` oppure `document.forms["modulo1"]` o, ancora, `document.forms[i]` con "i" numero tale per cui `document.forms[i].name=="modulo1"`.

8.2 Sincronizzazione

Esecuzione differite:

```
//imposta l'esecuzione differita
setTimeout("nomefunzione o codice", MillesimiDiSecondo);
//interrompe l'attesa
clearTimeout();
```

Impostazione di un timer:

```
setInterval("nomefunzione o codice", intervallo, "argomenti");
// "argomenti" è utile se si chiama una funzione
```

8.3 COOKIES

I cookies consentono di salvare delle informazioni sul client. Esistono diverse possibilità:

```
//SCRITTURA DI VALORI IN UN COOKIE

//informazione valida solo per tutta la vita del browser
document.cookie = "attributo = valore";

//informazione con data di scadenza
document.cookie = "attributo = valore ; expires = data";

//LETTURA DI UN VALORE

var valoreletto = document.cookie;
//l>alert seguente visualizza una stringa del tipo
//"nome1=valore1; nome2=valore2; nome3=valore3"
alert(valoreletto);
```

altre proprietà:

```
;path
;domain
;secure
```