

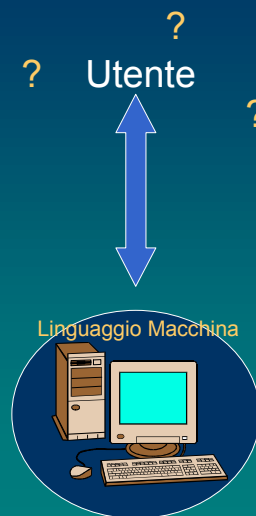
JavaScript Core Language

Introduzione alla Programmazione e JavaScript

Matteo Baldoni
Dipartimento di Informatica - Università degli Studi di Torino
C.so Svizzera, 185 - I-10149 Torino
E-mail: matteo.baldoni@unito.it
URL: <http://www.di.unito.it/~baldoni>

Programmare un computer

- Un computer è una macchina programmabile, tuttavia esso non è direttamente utilizzabile da parte degli utenti poiché richiederebbe la conoscenza sull'organizzazione fisica della specifica macchina e del suo linguaggio macchina
- Ogni macchina avrebbe le sue differenti caratteristiche
- Il linguaggio macchina è estremamente complicato e non di facile gestione



Programmare un computer: astrazione

- In altre parole desideriamo **astrarci** dai dettagli fisici della macchina in oggetto e dal suo specifico linguaggio macchina
- L'idea è quella di realizzare al di sopra della macchina reale una **macchina virtuale astratta** che abbia le funzionalità desiderate e che sia facile da utilizzare per l'utente
- L'utente interagisce con la macchina virtuale, ogni comando viene poi **tradotto** nei corrispondenti comandi sulla macchina fisica
- La macchina virtuale è realizzata mediante **software** (programmi)

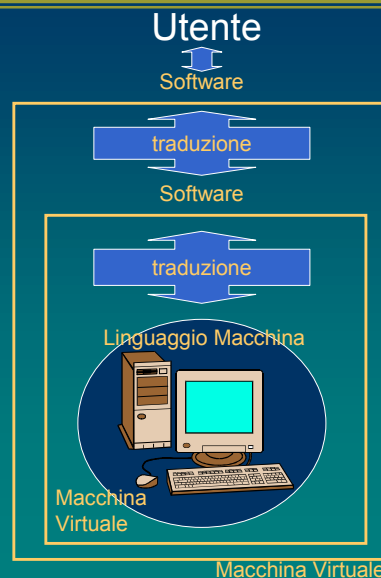


JavaScript Core Language

3

Programmare un computer: astrazione

- La macchina virtuale viene realizzata in genere mediante il **software di base**:
 - *Sistema Operativo*: file system, memoria, cpu, risorse ausiliarie, comunicazione
 - *Linguaggi e ambienti di programmazione ad alto livello*: interpreti e compilatori
- Non vi sono limiti al numero e al tipo di macchine virtuali che possono essere realizzate
- In genere nelle macchine moderne sono strutturate su più livelli (*struttura a cipolla*)

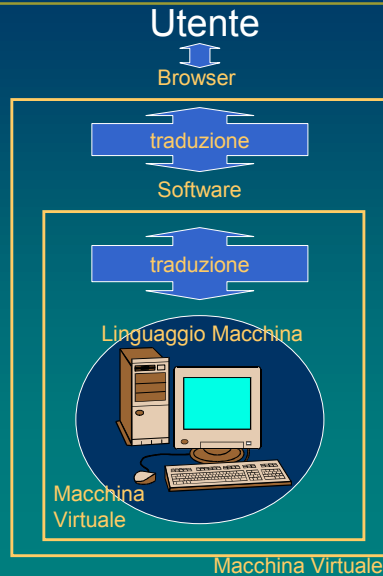


JavaScript Core Language

4

Macchine Virtuali: anche il browser lo è

- un browser è un programma
- Un browser interpreta il linguaggio HTML e visualizza le pagine sullo schermo
- Per la visualizzazione delle pagine un browser che per farlo si appoggia al software di base
- Il software di base si appoggia alla macchina fisica per realizzare effettivamente il “rendering” della pagina su video

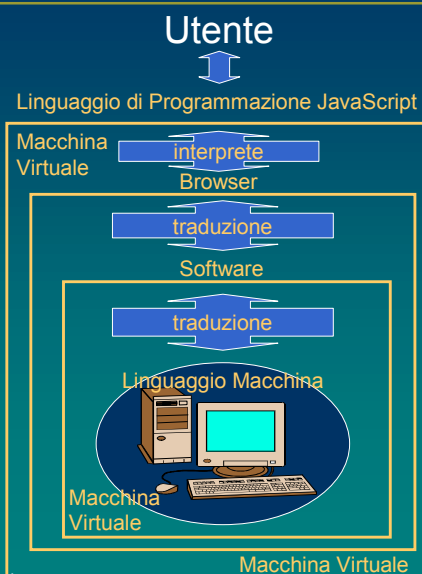


JavaScript Core Language

5

Macchine Virtuali: l'interprete JavaScript

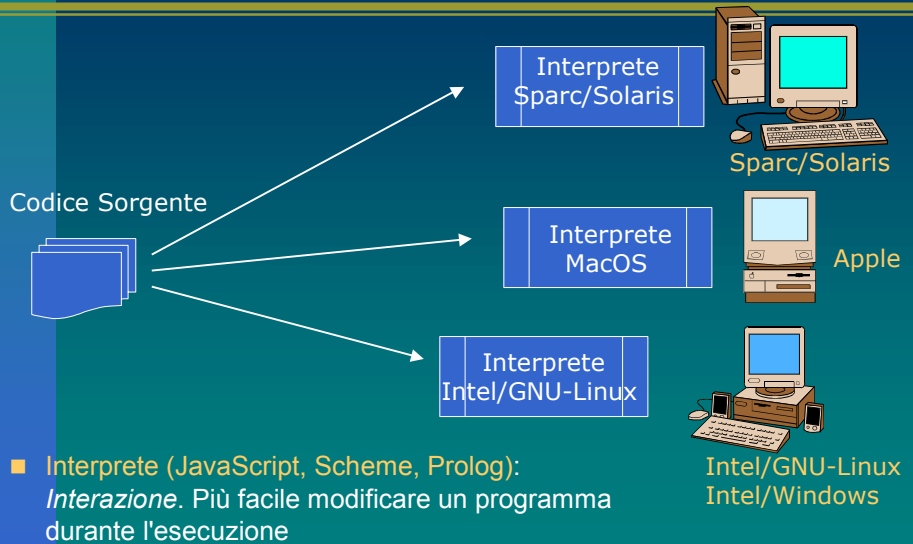
- Il linguaggio di programmazione JavaScript (in particolare il suo interprete) può essere considerato come un'ulteriore livello di astrazione, un'ulteriore macchina virtuale
- Mediante JavaScript è possibile programmare la macchina fisica
- In particolare, in JavaScript è possibile programmare in maniera più semplice quelle che sono le funzionalità offerte da un browser e quindi il controllo del “rendering” di documenti HTML



JavaScript Core Language

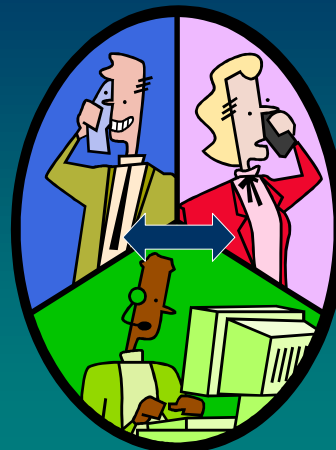
6

Interpretazione

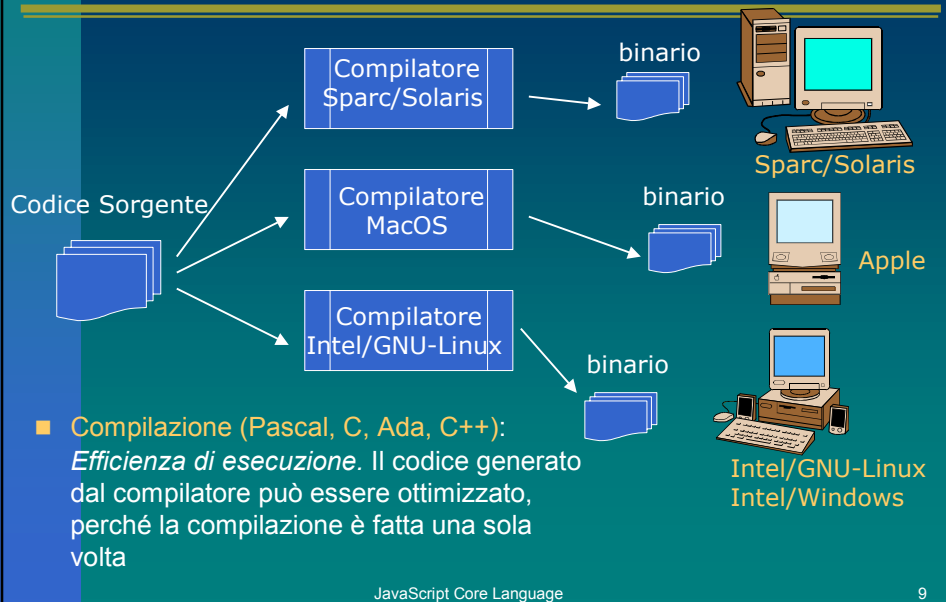


Interpretazione

- Affinché le due persone di lingua diversa possano dialogare tra di loro (nel caso nessuna delle due conosca la lingua dell'altro) è necessario che qualcuno interpreti (traduca sul momento) quanto dice una persona nella propria lingua nella lingua di chi ascolta
- Si interpreta quando è necessario una stretta interazione, quando si desidera dialogare e non solo trasmettere un messaggio



Traduzione (o Compilazione)



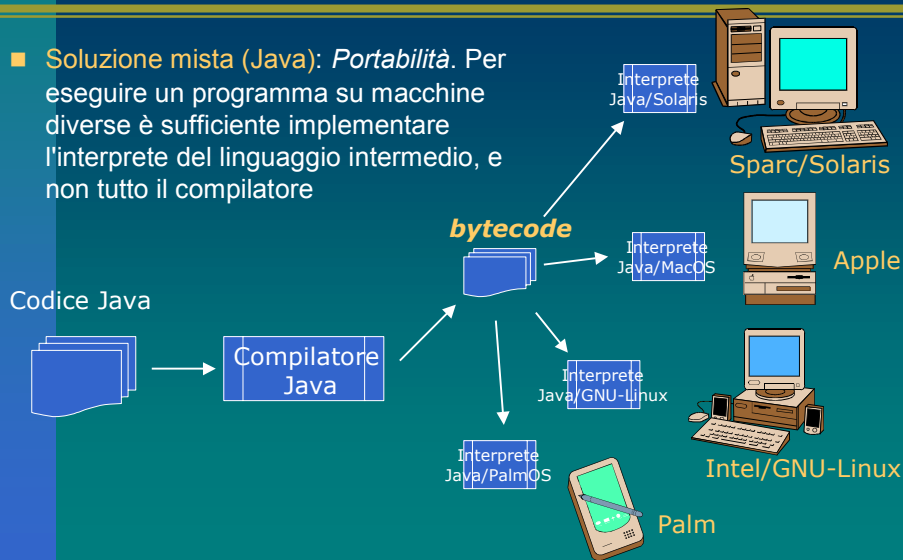
Traduzione (o Compilazione)

- La traduzione è adatta per comunicare messaggio, come ad esempio una lettera
- Anche per effettuare una traduzione è necessario l'intervento di qualcuno che sia in grado di comprendere le frasi di un linguaggio e riportarle in un altro ma questo può operare in tempi separati rispetto la scrittura del messaggio e la sua lettura
- La lettura è più rapida e semplice, il traduttore ha senz'altro avuto tempo per meglio adattare il testo
- ma si penalizza l'interattività



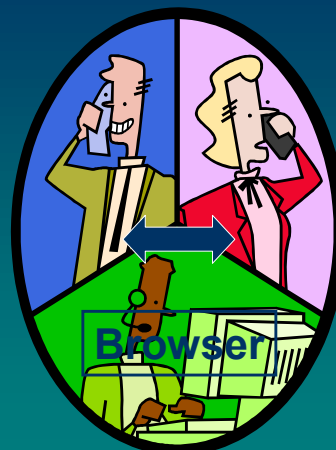
Soluzione Mista: Java

- **Soluzione mista (Java): Portabilità.** Per eseguire un programma su macchine diverse è sufficiente implementare l'interprete del linguaggio intermedio, e non tutto il compilatore



Interpretazione di JavaScript

- JavaScript è interpretato ed il suo interprete è il browser (es. Netscape, Internet Explorer, Opera)
- Non tutti i browser sanno interpretare JavaScript e non tutti lo sanno fare nella stessa maniera sebbene il linguaggio JavaScript sia uno (standard)
- Ogni comando è immediatamente tradotto in un insieme di comandi che il browser può eseguire grazie l'ausilio del software di base della macchina



Perché JavaScript?

- Perché si desidera rendere le pagine HTML dinamiche
- I programmi JavaScript possono essere “embedded” in una pagina HTML
- Gli “script” in JavaScript vengono scaricati insieme ad una pagina HTML e quindi interpretati dal browser del client



JavaScript Core Language

13

JavaScript in una pagina HTML

```
<HTML>
<HEAD>
<TITLE> ... </TITLE>
</HEAD>
<BODY>
...
<SCRIPT>
<!-- Inizio script JavaScript

var a0, b0, r;

a0 = window.prompt("Inserisci il primo numero");
b0 = window.prompt("Inserisci il secondo numero");

...

// Fine script -->
</SCRIPT>
...
</BODY>
</HTML>
```

- Il codice JavaScript è incluso in una pagina HTML mediante il tag SCRIPT
- È buona norma includere il codice JavaScript all'interno di un tag di commento HTML per visualizzare la pagina HTML correttamente anche dai browser che non supportano JavaScript (ovviamente non potranno fare uso delle funzionalità dello script!)

JavaScript Core Language

14

JavaScript: identificatori e altro

```
<HTML>
<HEAD>
<TITLE> ... </TITLE>
</HEAD>
<BODY>
...
<SCRIPT>
<!-- Inizio script JavaScript
var a0, b0, r;
a0 = window.prompt("Inserisci il primo numero");
b0 = window.prompt("Inserisci il secondo numero");
...
// Fine script -->
</SCRIPT>
...
</BODY>
</HTML>
```

- JavaScript è case-sensitive, cioè MCD e mcd sono due identificatori diversi
- Ogni istruzione termina con un punto e virgola ";"
- Gli spazi, le tabulazioni e gli a capo sono ignorati dall'interprete (quindi usarli senza timore per rendere il programma più leggibile possibile)
- Le stringhe sono rappresentate con la notazione " ... "

JavaScript: commenti

```
<HTML>
<HEAD>
<TITLE> ... </TITLE>
</HEAD>
<BODY>
...
<SCRIPT>
<!-- Inizio script JavaScript
/*
  Calcolo del
  M.C.D. tra due numeri
*/
var a0, b0, r;
a0 = window.prompt("Inserisci il primo numero");
b0 = window.prompt("Inserisci il secondo numero");
...
// Fine script -->
</SCRIPT>
...
</BODY>
</HTML>
```

- È possibile inserire dei commenti su di una singola linea facendo precedere la linea di commento con il simbolo "//"
- Commenti su più linee possono essere introdotti racchiudendoli tra i simboli "/*" e "*/"

JavaScript: convenzioni

```
<HTML>
<HEAD>
<TITLE> ... </TITLE>
</HEAD>
<BODY>
...
<SCRIPT>
<!-- Inizio script JavaScript

/*
   Calcolo del
   M.C.D. tra due numeri
*/
var a0, b0, r;

a0 = window.prompt("Inserisci il primo numero");
b0 = window.prompt("Inserisci il secondo numero");
...

// Fine script -->
</SCRIPT>
...
</BODY>
</HTML>
```

- L'identificatore assegnato ad una variabile non deve mai coincidere con quello di una parola chiave (o riservata)
- Se un identificatore è ottenuto per composizione da più parole non si deve lasciare spazi tra queste, si può rimpiazzarli dal simbolo “_” o unendo le parole e scrivendo in maiuscolo la prima lettera (metodo consigliato):

numero_intero
numeroIntero

Dichiarazioni di variabili

```
<SCRIPT>
<!-- Inizio script JavaScript

var a0, b0, r;

a0 = window.prompt("Inserisci il primo numero");
b0 = window.prompt("Inserisci il secondo numero");

var a = a0, b = b0;

document.writeln("M.C.D. (" + a + ", " + b + ") = <BR>");

if (b != 0) {
  r = a % b;
  while (r != 0) {
    a = b;
    b = r;
    r = a % b;
    document.writeln("M.C.D. (" + a + ", " + b + ") = <BR>");
  }
  document.writeln(b);
}
else
  document.writeln(a);

// Fine script -->
</SCRIPT>
```

- Per utilizzare una variabile è necessario dichiararla
- La dichiarazione di una variabile viene esseguita tramite la parola chiave var
- È possibile combinare una dichiarazione con una inizializzazione

Tipi di variabili

```
<SCRIPT>
<!-- Inizio script JavaScript

var a0, b0, r;

a0 = window.prompt("Inserisci il primo numero");
b0 = window.prompt("Inserisci il secondo numero");

var a = a0, b = b0;

document.writeln("M.C.D. (" + a + ", " + b + ") = <BR>");

if (b != 0) {
  r = a % b;
  while (r != 0) {
    a = b;
    b = r;
    r = a % b;
    document.writeln("M.C.D. (" + a + ", " + b + ") = <BR>");
  }
  document.writeln(b);
}
else
  document.writeln(a);

// Fine script -->
</SCRIPT>
```

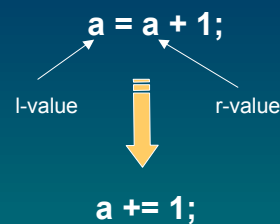
- Numeri
- Stringhe
- Booleani
- Array
- Oggetti
- Le variabili in JavaScript non hanno però alcun tipo associato alla dichiarazione:
`var x = 10;`
`x = "dieci";`
- Nota: l'operatore "+" è overloaded

JavaScript Core Language

19

Assegnamento ed operatori vari

- = indica l'assegnamento
- Operatori aritmetici: + - * / %
- la somma di un valore ad una variabile e l'assegnamento del totale alla variabile stessa si può indicare con +=
- analogamente si può utilizzare -=, *=, /=, %= e altri ancora
- Operatori di incremento (in posizione di prefisso o di suffisso di una variabile numerica): ++ e -- (es. i++; a--i;)
- Operatori relazionali e booleani: == != < > <= >= && || !
- Operatori sui bit: & | ^ ~ >> <<
- Parentesi: ()



JavaScript Core Language

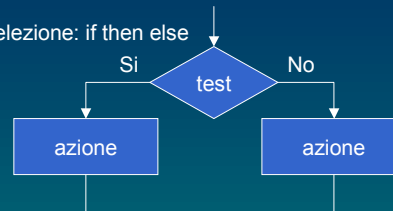
20

Strutture di controllo del flusso

■ Istruzione "if ... then ... else ..."

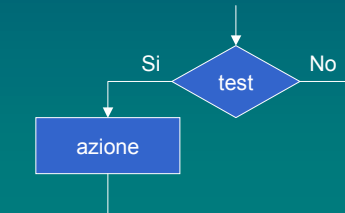
```
if (test) {  
  ... azione  
}  
else {  
  ... azione  
}
```

selezione: if then else



■ Istruzione "if ... then"

```
if (test) {  
  ... azione  
}
```



selezione: if then

Strutture di controllo del flusso

```
<SCRIPT>  
<!-- Inizio script JavaScript  
  
var a0, b0, r;  
  
a0 = window.prompt("Inserisci il primo numero");  
b0 = window.prompt("Inserisci il secondo numero");  
  
var a = a0, b = b0;  
  
document.writeln("M.C.D. (" + a + ", " + b + ") = <BR>");  
  
if (b != 0) {  
  r = a % b;  
  while (r != 0) {  
    a = b;  
    b = r;  
    r = a % b;  
    document.writeln("M.C.D. (" + a + ", " + b + ") = <BR>");  
  }  
  document.writeln(b);  
}  
else  
  document.writeln(a);  
  
// Fine script -->  
</SCRIPT>
```

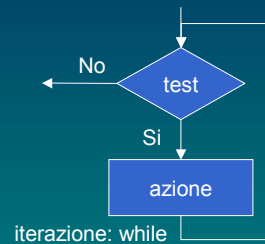
- al posto di un blocco di istruzione è possibile anche avere una sola istruzione
- queste vanno sempre terminate da un punto e virgola, anche tra l'if e l'else

Strutture di controllo del flusso

- Istruzione "while"

```
while (test) {  
  ... azione  
}
```

- Se il blocco di istruzioni contengono una sola istruzione le parentesi graffe non sono necessarie
- Come al solito le istruzioni vanno sempre terminate con un punto e virgola



Strutture di controllo del flusso

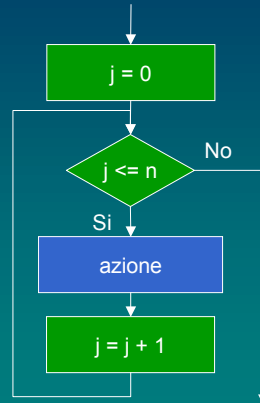
```
<SCRIPT>  
<!-- Inizio script JavaScript  
  
var a0, b0, r;  
  
a0 = window.prompt("Inserisci il primo numero");  
b0 = window.prompt("Inserisci il secondo numero");  
  
var a = a0, b = b0;  
  
document.writeln("M.C.D. (" + a + ", " + b + ") = <BR>");  
  
if (b != 0) {  
  r = a % b;  
  while (r != 0) {  
    a = b;  
    b = r;  
    r = a % b;  
    document.writeln("M.C.D. (" + a + ", " + b + ") = <BR>");  
  }  
  document.writeln(b);  
}  
else  
  document.writeln(a);  
  
// Fine script -->  
</SCRIPT>
```

Strutture di controllo del flusso

- L'istruzione di tipo for permette di ripetere sequenze di istruzioni per un numero fissato di volte

```
for (j=0; j<=n; j++) {  
  ... azione  
}
```

```
for (inizializzazione;  
     test; modifica) {  
  ... azione  
}
```



Break

- Permette di uscire da un loop
 - `break;`

```
[...]  
var i=0;  
for (i=0; i<n; i++)  
  if (elemento[i] == elementoCercato)  
    break;  
if (i != n)  
  document.writeln("Elemento trovato in pos.: " + i);  
else  
  document.writeln("Elemento non trovato");  
[...]
```

Break

- Il codice precedente senza uso del break
- il codice è più complicato, ma ...
- ... NON ABUSARNE

```
var i=0;
var trovato = false;
while (i<n && !trovato) {
  if (elemento[i] == elementoCercato) // oppure i++
    trovato = true;
  i++;
}
if (trovato)
  document.writeln("Elemento trovato in posizione: " + i);
else
  document.writeln("Elemento non trovato");
```

JavaScript Core Language

27

Controllo dell'input e window.alert

```
<SCRIPT>
<!-- Inizio script JavaScript
var a0, b0, r;

a0 = prompt("Inserisci il primo numero");
while (isNaN(a0) || a0 <= 0 || a0 == null ||
  ((a0 - parseInt(a0)) != 0)) {
  alert("Il valore " + a0 + " non va bene,\n" +
    "inserire un numero intero positivo.");
  a0 = prompt("Inserisci il primo numero");
}
a0 = a0 * 1;
b0 = prompt("Inserisci il secondo numero");
while (isNaN(b0) || b0 <= 0 || b0 == null ||
  ((b0 - parseInt(b0)) != 0)) {
  alert("Il valore " + b0 + " non va bene,\n" +
    "inserire un numero intero positivo.");
  b0 = prompt("Inserisci il primo numero");
}
b0 = b0 * 1;
var a = a0, b = b0;
[...]
// Fine script -->
</SCRIPT>
```

- Desideriamo controllare che l'input inserito sia effettivamente un numero intero positivo
- Se non è un intero positivo il numero inserito si segnala l'errore e si richiede una nuova immissione
- Si noti parseInt(), isNaN(.) e alert(.)

JavaScript Core Language

28

Le Funzioni: perché

- Il precedente esempio contiene due blocchi di istruzioni simili per la richiesta e il controllo dell'input, una per ogni valore richiesto all'utente
- I due blocchi di istruzioni differiscono per:
 - la variabile su cui è memorizzato il valore in input (a0 e b0)
 - il messaggio che viene visualizzato nella finestra "prompt" ("Inserisci il primo/secondo numero")
- Ci piacerebbe poter disporre di una nuova istruzione del tipo `promptNumero(messaggio)`
- La nuova istruzione dovrebbe essere come `prompt` ma con il controllo che il valore immesso sia un intero positivo

Le Funzioni: definizione

- In altre parole: si vorrebbe definire una **funzione**, cioè *una parte di codice utilizzabile in più parti di uno stesso programma*
- In JavaScript questo è possibile farlo utilizzando la parola chiave **function**:

```
function nome_della_funzione (arg1, arg2, ..., argn) {  
    "definizione della funzione"  
}
```

The diagram illustrates the syntax of a JavaScript function definition. It shows the following code snippet:

```
function nome_della_funzione (arg1, arg2, ..., argn) {  
    "definizione della funzione"  
}
```

Arrows point from labels to specific parts of the code:

- parentesi graffe!** points to the opening curly brace `{`.
- nome della funzione** points to the function name `nome_della_funzione`.
- parametri "formali" della funzione, usati nella definizione della funzione** points to the parameter list `(arg1, arg2, ..., argn)`.
- "definizione della funzione"** points to the function body `"definizione della funzione"`.

Le Funzioni: richiamo

- La definizione di una funzione è una sequenza di istruzioni, un blocco di istruzioni
- Le istruzioni contenute in una funzione non vengono eseguite quando definite ma solo al momento del richiamo della funzione:

`nome_della_funzione(val1, val2, ..., valn)`

parametri attuali
della funzione

- Quando l'interprete incontra un richiamo di una funzione passa ad eseguire il codice contenuto nella definizione della funzione, dopo aver sostituito i parametri formali con quelli attuali

Le Funzioni: restituzione di un risultato

- All'interno di una funzione si può usare l'istruzione `return` per restituire dei valori, di solito il risultato prodotto dalla funzione stessa (es. il valore intero letto in input)
 - `return <espressione>;`: restituirà il valore computato dall'espressione
 - `return ;`: restituirà `undefined`
- In entrambi i casi si esce dalla funzione e l'interprete JavaScript passa ad eseguire l'istruzione che segue il richiamo della funzione

Le Funzioni

```
<SCRIPT>
<!-- Inizio script JavaScript

function promptNumero(messaggio) {
  var numero = prompt(messaggio);
  while (isNaN(numero) || numero <= 0 ||
        numero == null ||
        ((numero - parseInt(numero)) != 0)) {
    alert("Il valore " + numero + " non va bene,\n" +
          "inserire un numero intero positivo.");
    numero = prompt(messaggio);
  }
  return numero * 1;
}

var a0, b0, r;

a0 = promptNumero("Inserisci il primo numero");
b0 = promptNumero("Inserisci il secondo numero");

var a = a0, b = b0;
[...];
// Fine script -->
</SCRIPT> -->
```

- La funzione `promptNumero(.)` richiede un numero intero in input, verifica che il valore immesso lo sia. Il valore restituito è il numero intero positivo inserito dall'utente, dopo averlo convertito in `numero(!)`.
- Il parametro attuale è il messaggio da visualizzare nella richiesta.

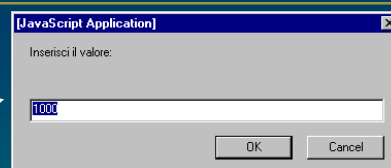
JavaScript Core Language

33

Finestre di dialogo

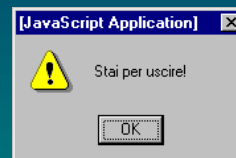
- `prompt("Inserisci il valore", "1000");`

restituisce la stringa
inserita nel campo



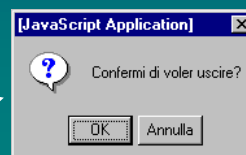
- `alarm("Stai per uscire!");`

non restituisce nulla



- `confirm("Confermi di voler uscire?");`

restituisce true o false



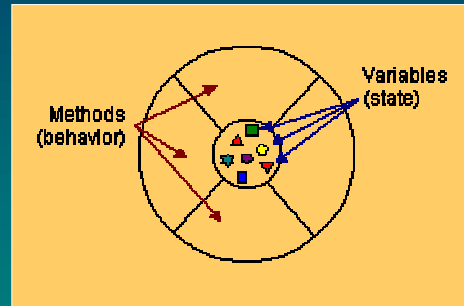
JavaScript Core Language

34

Oggetti

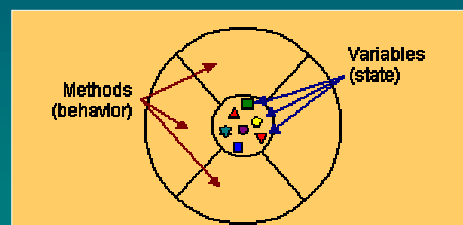
Un **oggetto** rappresenta un dato, ed è costituito da:

- *stato*: collezione di variabili (**campi** o **proprietà**)
- *comportamento*: collezione di operazioni (**metodi**)
- JavaScript è un linguaggio a oggetti



Incapsulamento

- I dati e le procedure che li manipolano sono raggruppati in un'unica entità, l'**oggetto**.
- Il mondo esterno ha accesso ai **dati** solo tramite un insieme di operazioni (**metodi**) che costituiscono l'**interfaccia** dell'oggetto. I dettagli dell'implementazione sono nascosti (**INFORMATION HIDING**)

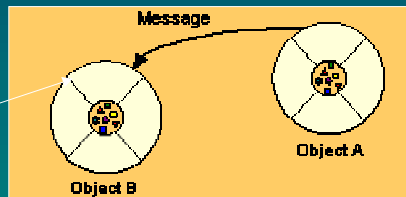


- Un oggetto realizza una **ASTRAZIONE DEI DATI**

Invio di Messaggi

- Gli oggetti sono **dinamici**, creati e distrutti durante l'esecuzione del programma.
- Un oggetto A, per agire su un altro oggetto B, **invia un messaggio** a B chiedendogli di eseguire uno dei metodi della sua interfaccia.

Es.: esegui il metodo m1



- Lo scambio di messaggi è l'unico modo di comunicare tra oggetti

JavaScript Core Language

37

Oggetti in JavaScript

- Gli oggetti vengono **creati** mediante l'operatore **new** e l'uso di un **costruttore**
- Esistono molti oggetti predefiniti in JavaScript (con proprietà e metodi), ad esempio le stringhe sono oggetti ed il loro costruttore è **String()**
- Notazione **"a punto"** per accedere alle proprietà di un oggetto o invocare metodi (inviare messaggi)

```
var nome = "Matteo Baldoni";  
var nome = new String("Matteo Baldoni");  
alert(nome.length);  
var nomeTuttoMaiuscolo = nome.toUpperCase();  
alert(nomeTuttoMaiuscolo);
```

JavaScript Core Language

38

Il costruttore Object()

- Un oggetto “generico” è creato mediante l'operatore **new** e il costruttore **Object()**
- Object è un oggetto “generico” senza proprietà e metodi, un oggetto vuoto
- Notazione “a punto” per accedere alle proprietà di un oggetto o invocare metodi (inviare messaggi)
- E' possibile definire costruttori di oggetti e metodi (si veda [Flanagan, 1998] per maggiori dettagli)

```
var poligonoRegolare = new Object();  
  
poligonoRegolare.nome = "quadrato";  
poligonoRegolare.lato = 10;  
poligonoRegolare.nLati = 4;  
  
var poligonoRegolare = {  
  nome : "quadrato",  
  lato : 10,  
  nLati : 4  
}  
  
function perimetro() {  
  return this.lato * this.nLati;  
}  
  
function PoligonoRegolare(nome, lato, nLati) {  
  this.nome = nome;  
  this.lato = lato;  
  this.getPerimetro = perimetro;  
}
```

Gli oggetti come array associativi

- Gli oggetti possono anche essere visti come **array associativi**
- Si può accedere alle proprietà mediante l'operatore “[]”
- Si noti che nella notazione “a punto” il nome della proprietà è un **identificatore** mentre nel caso degli array associativi è una **stringa**
- Il costrutto **for/in** permette di **enumerare** le proprietà degli oggetti

```
var poligonoRegolare = new Object();  
  
poligonoRegolare["nome"] = "quadrato";  
poligonoRegolare["lato"] = 10;  
poligonoRegolare["nLati"] = 4;  
  
for (proprietà in persona)  
  alert(proprietà);  
  
for (proprietà in persona)  
  alert(persona[proprietà]);  
  
for (proprietà in persona)  
  alert(persona.proprietà);
```

Gli array sono oggetti

- Un array in JavaScript è un oggetto
- Per creare una variabile di tipo array si usa l'operatore new e il costruttore Array()
- La proprietà length contiene la lunghezza corrente di un array
- Esistono diversi metodi associati ad un array, quali, ad esempio, join(), reverse(), sort(), ecc.

```
var tavolo = new Array();  
var tavolo = new Array(3, 1, 5);  
var tavolo = new Array(3);  
alert(tavolo[0]);  
alert(tavolo.join());
```

Ordinamento per Selezione in JavaScript

```
<SCRIPT>  
<!-- Inizio script JavaScript  
var tavolo = new Array();  
  
function promptNumero(messaggio) {  
  var numero = prompt(messaggio);  
  while (isNaN(numero) || numero <= 0 || numero == null) {  
    alert("Il valore " + numero + " non va bene,\n" +  
      "inserire un numero intero positivo.");  
    numero = prompt(messaggio);  
  }  
  return numero * 1;  
}  
  
var i=0;  
  
while (confirm("Vuoi introdurre una carta?")) {  
  tavolo[i] = promptNumero("Inserisci il valore della carta n. "  
    + (i + 1) + ".");  
  i++;  
}
```

Creazione dell'oggetto di tipo array

Togliere questo controllo se si desidera inserire anche valori di carte negativi

Chiede se si desidera immettere altri valori

Ordinamento per Selezione in JavaScript

```
[...]  
var j;  
var posizione_min;  
var temp;  
  
for (i=0; i < (tavolo.length-1); i++) {  
  posizione_min = i;  
  for (j=i; j<tavolo.length; j++) {  
    if (tavolo[posizione_min] > tavolo[j])  
      posizione_min = j;  
  }  
  temp = tavolo[i];  
  tavolo[i] = tavolo[posizione_min];  
  tavolo[posizione_min] = temp;  
}  
  
for (i = 0; i<tavolo.length; i++)  
  document.write(i + ": " + tavolo[i] + "<BR>");  
  
// Fine script -->  
</SCRIPT>
```

Contiene il numero di carte immesse nell'array

JavaScript Core Language

43

Ordinamento di un array con *sort()*

```
<SCRIPT>  
<!-- Inizio script JavaScript  
  
var tavolo = new Array();  
  
function promptNumero(messaggio) {  
  var numero = prompt(messaggio);  
  while (isNaN(numero) || numero <= 0 || numero == null) {  
    alert("Il valore " + numero + " non va bene,\n" +  
      "inserire un numero intero positivo.");  
    numero = prompt(messaggio);  
  }  
  return numero * 1;  
}  
  
var i=0;  
while (confirm("Vuoi introdurre una carta?")) {  
  tavolo[i] = promptNumero("Inserisci il valore della carta n. " + (i + 1) + ".");  
  i++;  
}  
  
tavolo.sort();  
  
for (i = 0; i<tavolo.length; i++)  
  document.write(i + ": " + tavolo[i] + "<BR>");  
  
// Fine script -->  
</SCRIPT>
```

Metodo built-in per gli array per effettuare un ordinamento

JavaScript Core Language

44

Copia di oggetti

```
<SCRIPT>
<!-- Inizio script JavaScript
var tavolo = new Array();
var tavolo2 = tavolo;

function promptNumero(messaggio) {
  var numero = prompt(messaggio);
  while (isNaN(numero) || numero <= 0 || numero == null) {
    alert("Il valore " + numero + " non va bene,\n" +
      "inserire un numero intero positivo.");
    numero = prompt(messaggio);
  }
  return numero * 1;
}
var i=0;
while (confirm("Vuoi introdurre una carta?")) {
  tavolo[i] = promptNumero("Inserisci il valore della carta n. " + (i + 1) + ".");
  i++;
}
tavolo.sort();
for (i = 0; i<tavolo.length; i++)
  document.write(i + ": " + tavolo[i] + "<BR>");

for (i = 0; i<tavolo2.length; i++)
  document.write(i + ": " + tavolo2[i] + "<BR>");
// Fine script -->
</SCRIPT>
```

Cosa succede di tavolo2 ??