JavaScript - 2

Ambiente di esecuzione Javascript

L'ambiente del web browser – 1

- Per capire come funziona JavaScript lato client bisogna capire la struttura dell'ambiente di programmazione offerto da un web browser
- Questo ambiente di programmazione possiede tre importanti caratteristiche

1. Esiste l'oggetto Window che serve sia da oggetto GLOBALE sia da contesto globale di esecuzione di codice JavaScript lato client

L'ambiente del web browser – 2

2. Esiste una gerarchia di oggetti lato client ed il modello oggetto del documento (*Document Object Model* – DOM) che forma una parte di essa
3. Esiste un modello di programmazione azionato dagli eventi (*event-driven*)
   - Ad esempio, se si preme un bottone o si carica una pagina si genera un evento che può essere controllato

L'oggetto **Window** – 1

- In JavaScript lato client l'oggetto **Window** è un oggetto globale, esso rappresenta la finestra (o frame) che mostra il documento
- La finestra corrente è il contesto di esecuzione di JavaScript lato client
- Definisce varie proprietà e metodi che ci permettono di manipolare la finestra del browser
  - Ha anche proprietà auto-referenziali window, this e self
  - Definisce anche proprietà che si riferiscono ad altri oggetti importanti ad esempio Document referenziato tramite document

L'oggetto **Window** – 2

- Dato che l'oggetto **Window** è un oggetto globale, in Javascript lato client, tutte le variabili globali sono definite come proprietà di **Window**
- Il seguente codice fa essenzialmente la stessa cosa

```
var variabile = 14;
window.variabile = 14;
```

- Possiamo omettere window per accedere alle proprietà di **Window**
- Ogni frame genera un suo oggetto **Window**

Nota

- L'oggetto Window ha al suo interno varie proprietà tra cui document che è un'istanza dell'oggetto Document
- Document rappresenta il documento correntemente mostrato nella finestra del browser
- Maggiori dettagli su Window nelle prossime lezioni

Due modelli ad oggetti

- Un modello ad oggetti definisce l'interfaccia ad i vari aspetti del browser e del documento che possono essere manipolati con JavaScript
- In JavaScript sono utilizzati due modelli ad oggetti
  - Un modello ad oggetti del browser (BOM)
  - Un modello ad oggetti del documento (DOM)

Il BOM

- Il BOM fornisce l'accesso alle varie caratteristiche di un browser e dell'ambiente in cui il browser è in esecuzione come: la finestra del browser stesso, le caratteristiche dello schermo, la cronologia del browser e così via

```
windows.status = "benvenuto nella mia HP"
```

Vietato da alcuni browser

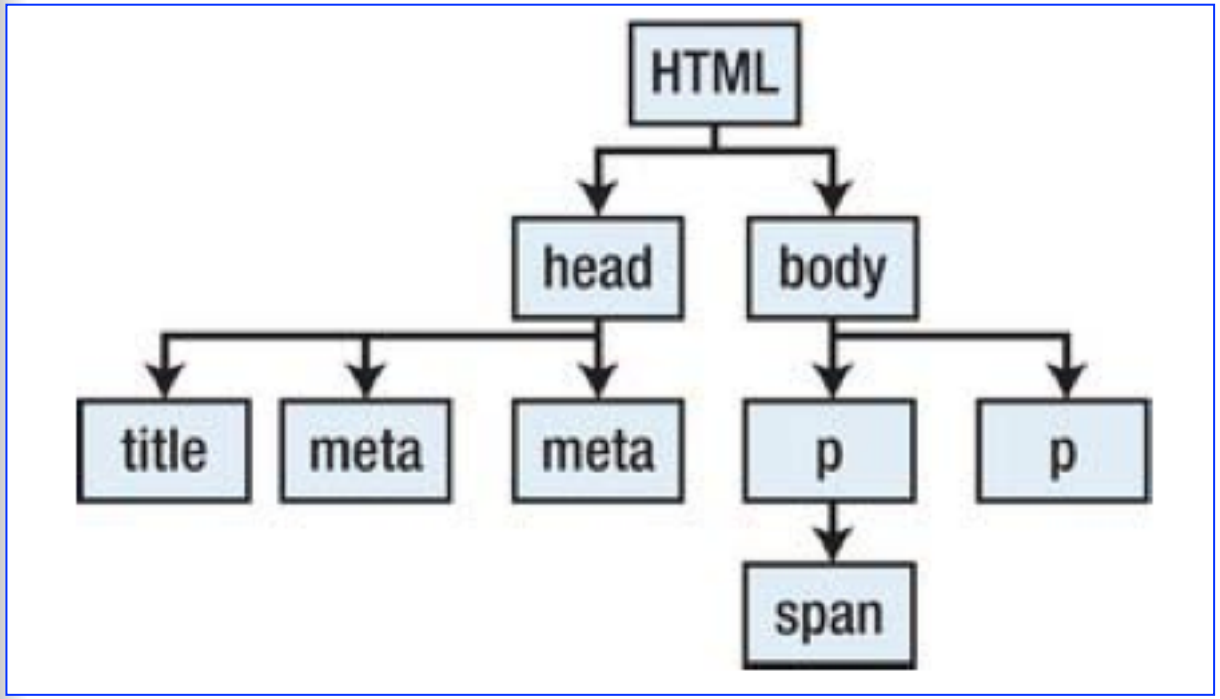
```
if(screen.height == 600)

history.back();
```

Il DOM

- Il DOM, d'altra parte, fornisce l'accesso al contenuto della finestra del browser, cioè al documento incluso i vari elementi HTML che vanno dalle ancore alle immagini così come il testo che può essere incluso da tali elementi

```
document.images["miobanner"].src= "uno.gif";
document.modulo.testo.value = "Ciao!!!";
document.miobanner.src = "1.gif";
```

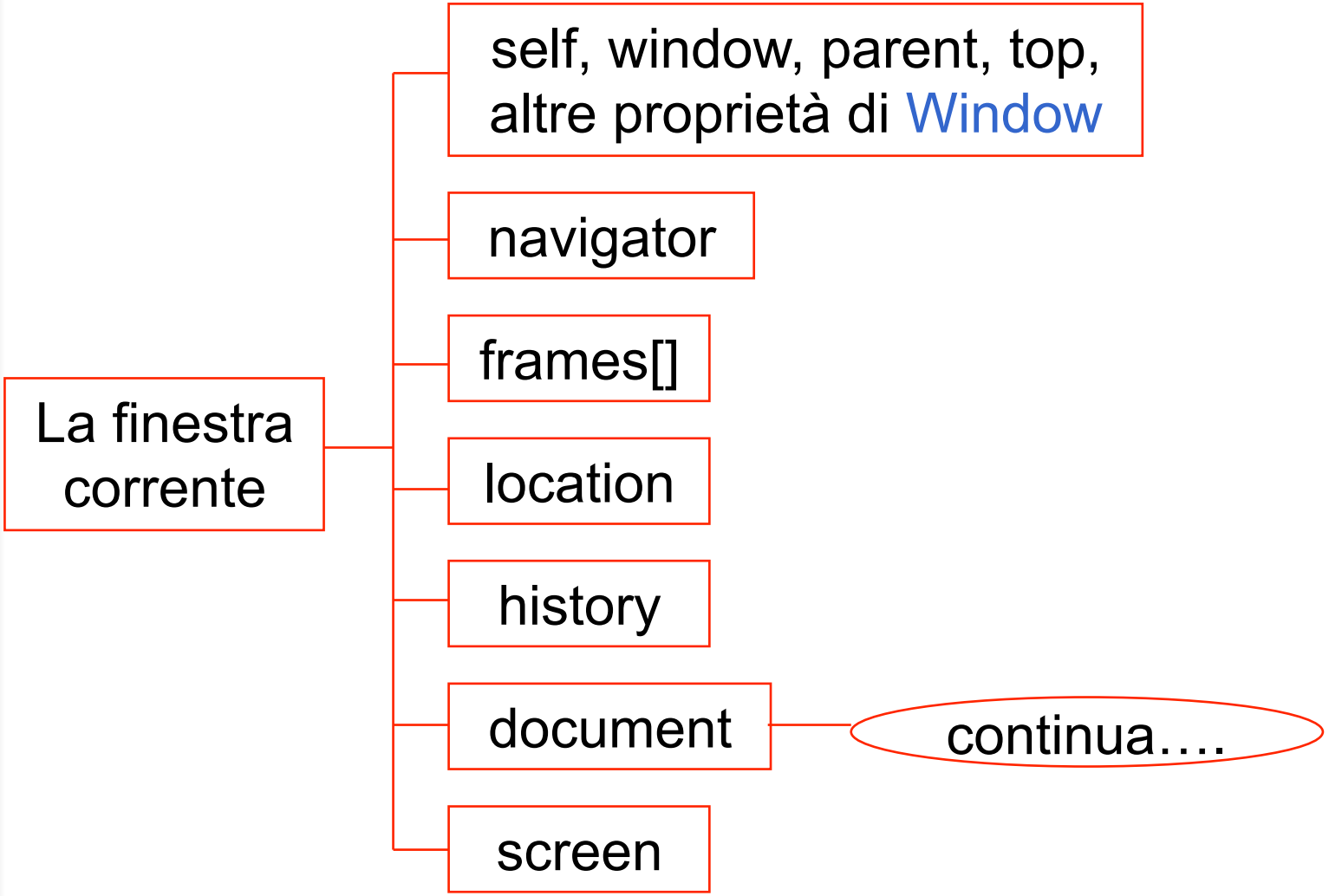


Il browser legge il documento HTML e crea un modello che preserva la gerarchia degli elementi HTML. Ogni elemento è rappresentato da un oggetto Javascript

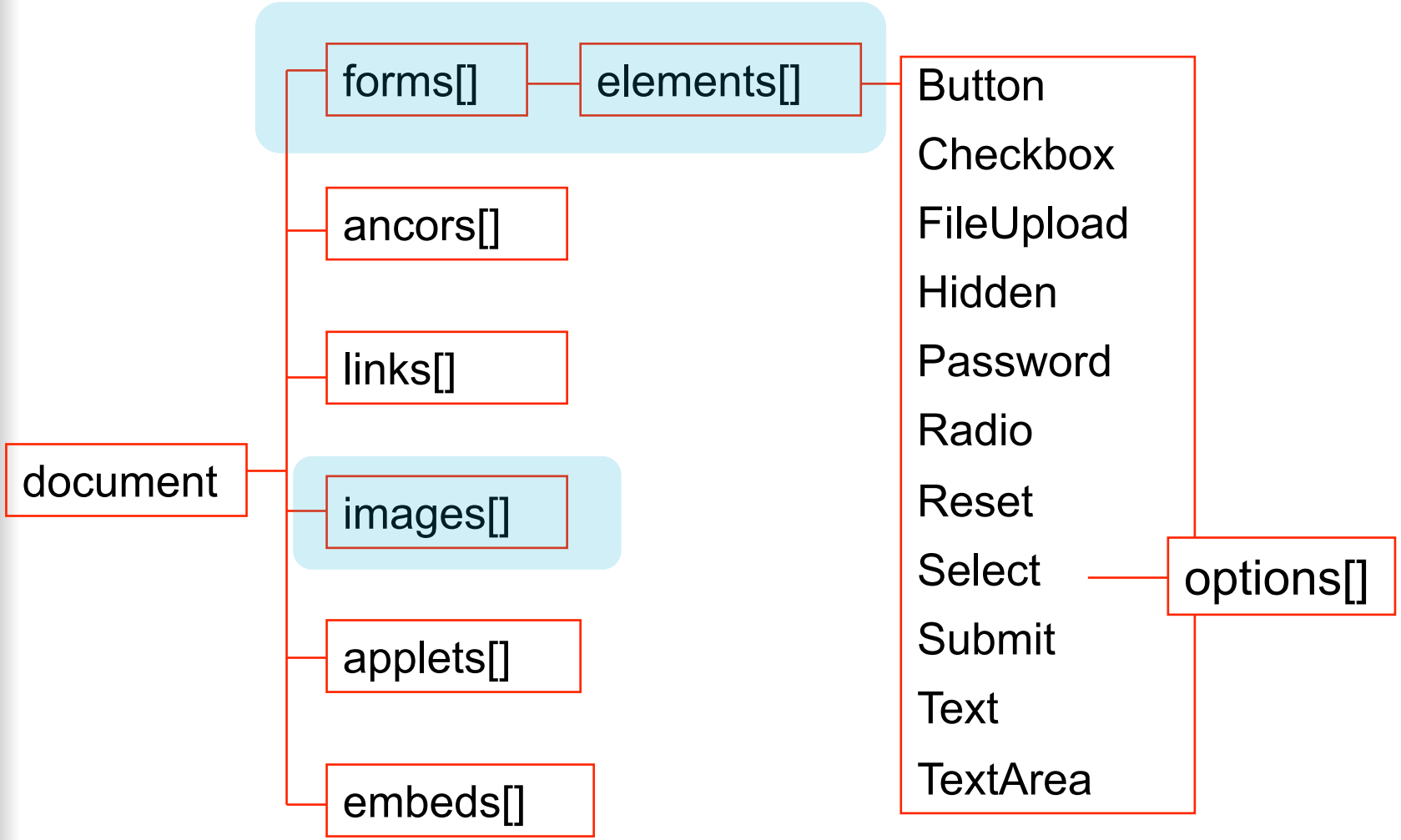
DOM

- **DOM** (livello 0)
  - Grossolanamente equivalente al modello a oggetti di NN3. Spesso è chiamato modello a oggetti *classico* o *tradizionale*, si accede solo a parte del documento
- **DOM** (livello 1)
  - Si può far riferimento a tutti gli elementi HTML. Fornisce la possibilità di manipolare i *nodi* del documento attraverso metodi dell'oggetto document
- **DOM** (livello 2)
  - Si può far riferimento a proprietà CSS degli elementi

Gerarchia oggetti in Window



Gerarchia oggetti DOM (livello 0)



Ma cosa sono quegli array?

- L'array images[] è un array contenente i riferimenti a tutte le immagini (**oggetti di tipo Image**) presenti nella pagina
  - images[0] si riferisce all prima immagine presente nel documento
  - images[images.length-1] si riferisce all'ultima immagine del documento
- Gli elementi della gerarchia precedente che non sono array rappresentano degli oggetti

Accesso agli elementi del DOM

- Per accedere agli elementi del DOM usiamo la classica *dot notation*

- Ad esempio, con la seguente espressione

document.forms[0].elements[3].options[2].text

Accediamo al testo (text) del terzo item di una lista di selezione (options[2]) che è il quarto elemento (elements[3]) del primo modulo (forms[0]) presente nel documento

Accesso agli elementi HTML

- Ci sono altri modi per accedere agli elementi (images[], forms[].elements[]) di una pagina HTML
- Si usa sempre la *dot notation*, ma invece di indicare l'indice dell'elemento possiamo indicare il suo nome
  - Ad ogni tag HTML possiamo associare un nome attraverso l'attributo NAME
  - In questo modo anche se la struttura del documento cambia, non dobbiamo modificare lo script

Esempi – indipendenti dalla posizione fisica degli elementi nel documento

```
<IMG src="0.gif" name="banner">
document.images["banner"].src = "1.gif";
```

```
<FORM NAME="modulo">
<INPUT TYPE="TEXT" NAME="testo">
</FORM>
document.forms["modulo"].elements["testo"].value
    = "Ecco un altro testo!!!";
```

…meglio ancora

- Nel caso di immagini o moduli si può anche evitare di indicare il tipo di elemento (images[], forms[].elements[])
- È sufficiente assegnare dei nomi alle immagini, moduli, controlli dei moduli e, sempre tramite la *dot notation* accediamo all'elemento

Esempi

```
<FORM NAME="modulo">
<INPUT TYPE="TEXT" NAME="testo">
</FORM>
document.modulo.testo.value = "Funziona!!!";
```

```
<DIV name="primo"> <P name="secondo">
<SPAN name="terzo">
<IMG SRC="0.gif" NAME="immagine">
</SPAN></P></DIV>
document.immagine.src = "1.gif";
```

Riepilogando

- Se la terza immagine nel documento HTML si chiama "casa" possiamo far riferimento ad essa in uno dei seguenti modi
  - document.images[2]
  - document.images["casa"]
  - document.casa
- I vantaggi dell'ultimo approccio sono evidenti

E con gli altri elementi?

- Esistono tag HTML che non hanno associato nessun “array”
- Come possiamo accedere ai paragrafi (**<P>**) o alle sezioni (**<DIV>**)?
- Vedremo che potremo far riferimento direttamente ad un qualsiasi elemento HTML purché abbia l’attributo ID settato (level 1)
  - document.getElementByID(id);

Oppure

- document.

  getElementById(<id>)

  getElementsByClassName(<class>)

  getElementsByName(<name>)

  getElementsByTagName(<tag>)

  querySelector(<selector>)

  querySelectorAll(<selector>)

La programmazione *Event-Driven*

- JavaScript è un linguaggio *event-driven*
  - L'utente interagisce con la pagina muovendo il mouse, cliccando, digitando qualcosa in una casella di testo, …
  - La browser può "reagire" opportunamente a queste sollecitazioni
  - Il *gestore dell'evento* è una funzione che risponde all'azione dell'utente (o dell'ambiente)
  - Possiamo catturare gli eventi e gestirli

Il browser e gli eventi

- L'interprete JavaScript 'monitora' un insieme di eventi
  - Possono essere generati dall'utente attraverso le proprie azioni sul documento
  - Possono essere generati dal browser
    - caricamento delle immagini
    - caricamento del documento
    - è arrivata la riposta ad una richiesta AJAX

Eventi generati dall'utente

- spostamento del focus tra gli elementi del documento
  - da un campo all'altro di un modulo con il tab
- movimenti del mouse
  - passaggio del mouse sulle immagini
  - passaggio del mouse sui link
- click
  - pressione del tasto del mouse
  - rilascio del tasto del mouse

Eventi in JavaScript

- Abort
- Blur
- Change
- Click
- DblClick
- Error
- Focus
- KeyDown
- KeyPress
- KeyUp
- Load
- MouseDown
- MouseMove
- MouseOut
- MouseOver
- MouseUp
- Move
- Reset
- Resize
- Select
- Submit
- Unload

Gestori di eventi

- Ad ogni evento è associato un *gestore dell'evento*
- Per ottenere il nome del gestore di un evento basta aggiungere il prefisso **on** al nome dell'evento stesso
- Il gestore di un evento permette di associare ad un evento un programma (funzione) JavaScript
  - Ad ogni tag HTML è associato un insieme di gestori di eventi
  - È possibile associare eventi anche ad oggetti JavaScript (dettagli alla fine delle slide)

Esempi di gestori di eventi

- click onclick
  - Attivato quando l'utente clicca su un elemento
- mouseover onmouseover
  - Attivato quando il cursore del mouse si muove sopra un elemento
- mouseout onmouseout
  - Attivato quando il cursore del mouse si sposta fuori un elemento

Gestori di eventi in HTML

- Vengono indicati tra gli attributi dei tag HTML
- La sintassi solitamente è del tipo

  < … on*Evento*="*comandoJS*" … >

- dove *comandoJS* può essere
  - la semplice invocazione di una funzione,
  - ma anche una sequenza di comandi complessi (sconsigliato) separati da **;**

Il modello ad eventi è più complesso di quello descritto. Maggiori dettagli in una lezione successiva

```
<SCRIPT TYPE="text/javascript">
function ciao() { alert("Sei passato sull'immagine"); }

function addio() { alert("Hai lasciato l'immagine"); }
</SCRIPT>

<IMG SRC="0.gif"
        onMouseOver="ciao()"
        onMouseOut="addio()">
```

Si possono registrare gli eventi attraverso il metodo addEventListener
Dettagli in seguito con il modello ad eventi

Rollover – HTML

```
<BODY>
<img width="200" height="200" name="foto"
        src="immagini/out.png"
        onMouseOver="over()"
        onMouseOut="out()"
        onMouseDown="giu()"
        onMouseUp="su()">

<form name="info">
<input type="text" name="stato"> </form>
</BODY>
```

Registrazione eventi
Lo si può fare anche con javascript
Dettagli alla fine del corso

Rollover – JavaScript

```
<SCRIPT TYPE="text/javascript">

function over() { document.info.stato.value = "over";
                  document.foto.src = "immagini/over.png";  }

function out() { document.info.stato.value = "out";
                 document.foto.src = "immagini/out.png"; }

function giu() { document.info.stato.value = "giu";
                 document.foto.src = "immagini/giu.png"; }

function su() { document.info.stato.value = "su";
                document.foto.src = "immagini/su.png"; }
</SCRIPT>
```

Rollover – HTML V.2

```
<BODY>
<img width="200" height="200" name="foto"
        src="immagini/fotoout.png"
        onMouseOver="cambia('foto','over')"
        onMouseOut="cambia('foto','out')"
        onMouseDown="cambia('foto','giu')"
        onMouseUp="cambia('foto','su')">

<form name="info">
        <input type="text" name="stato" size="4">
</form></BODY>
```

Rollover – JavaScript V.2

```
<SCRIPT TYPE="text/javascript">
function cambia(nome,stato) {
    document.info.stato.value = stato;
    img="immagini/"+nome+stato+".png";
    document.images[nome].src=img;
}
</SCRIPT>
```

Perché non scriviamo “document.nome.src=img“ ?

- È necessario che le immagini abbiano un nome composto dal nome usato nell’attributo **NAME** di **IMG** seguito dal loro stato
- Non c’è pre-caricamento delle immagini. Quando si scatena l’evento, l’immagine necessaria viene caricata dal server e memorizzata in cache

Nomi di proprietà ed oggetti

- Per accedere ad una proprietà di un oggetto Javascipt possiamo
- Usare la dot notation
  - document.foto.src = …..
- Vedere l'oggetto come un array associativo
  - document,images["foto"] .src = …..

Pre-caricamento immagini

- Per far pre-caricare al browser le immagini, la cosa da fare è dichiarare un array di oggetti di tipo **Image** ed associare alla proprietà src il nome dell'immagine da pre-caricare

```
var ImgCaricate = new Array();
var directory = "immagini/";
var nome =  document.images[i].name;
var su   = directory + nome + "su.gif";
ImgCaricate [nome+"su"] = new Image();
ImgCaricate [nome+"su"] .src = su;
```