PROGRAMMAZIONE WEB

SCRIPTING CLIENT-SIDE IN JAVASCRIPT INTRODUZIONE

Prof. Ada Bagozi
ada.bagozi@unibs.it

LE BASI DEL LINGUAGGIO

Javascript - Definizione

- ✓ JavaScript è un linguaggio di scripting lato-client, ossia un linguaggio di programmazione interpretato dal browser che prevede la scrittura di script
- ✓ Uno script è un piccolo programma (contenuto o importato in una pagina HTML) che viene interpretato ed eseguito dal browser
- ✓ Mediante l'uso di script è possibile creare dinamicamente i contenuti di una pagina web e aggiungere interattività alla pagina stessa (con un occhio alle prestazioni)

- ✓ Javascript NON É Java: JavaScript e Java sono due linguaggi di programmazione differenti!!!

Javascript – Lo script

Gli script JavaScript possono essere:

- ✓ contenuti in uno o più file di testo con estensione **`js`** e linkati al file HTML con il tag **`script`** che va inserito fra i tag **`head`**

```
<head>
   <script type="text/javascript"
      src="myScript.js"/>
<head>
```

- ✓ contenuti nel file HTML, inseriti come testo all'interno del tag script

```
<script type="text/javascript"> ... </script>
```

Inserire lo script in una pagina

- ✓ Esistono inoltre alcuni attributi HTML in cui si può incorporare del codice:
  - ✓ gli attributi per la gestione degli eventi, come **`onclick`**, possono contenere frammenti di codice (ma non dichiarazioni), da eseguire al verificarsi dell'evento
  - ✓ L'attributo **`href`** del tag **`<a>`** può fare riferimento a una funzione javascript con la sintassi:

    `javascript:nome_funzione(parametri)`;

    in questo caso, il click del link eseguirà la chiamata alla funzione

Esecuzione di uno script

- ✓ Tutte le funzioni e le variabili dichiarate negli script diventano disponibili (quindi possono essere usate e chiamate) non appena il parser analizza il punto della pagina in cui sono dichiarate
- ✓ Se uno script contiene codice immediato, cioè scritto al di fuori di funzioni, questo viene eseguito non appena il parser analizza il punto della pagina in cui il codice compare
- ✓ Gli script possono utilizzare liberamente funzioni e variabili dichiarate in altri script inseriti nella stessa pagina.

Partiamo dalla sintassi …

JavaScript è un linguaggio di programmazione *case sensitive* ossia fa distinzione tra lettere maiuscole e minuscole

`num` è diverso da `Num`

Ogni singola istruzione va conclusa con il punto e virgola

```
alert("Hello world!!!");
```

I commenti all'interno di uno script vanno inseriti tra i caratteri `/*` e `*/` oppure dopo `//` per commenti su una riga

```
/* questo è un commento */
```

Tipi di dato

In Javascript possiamo avere i seguenti tipi di dati

Numeri - in JavaScript non vi è differenza tra numeri interi e numeri in virgola mobile

Stringhe - una stringa è formata da una sequenza di zero o più caratteri racchiusi tra apici singoli o doppi (‘ o “):

“`casa`” è la stringa `casa`
“`casa ‘Pisa’`” è la stringa `casa ‘Pisa’`
‘`casa “Pisa”`’ è la stringa `casa “Pisa”`

Valori Booleani - è un dato che esprime un “valore di verità” e può assumere solo due valori `true` e `false`

Array o **Oggetti**

Dichiarazione di variabili

Una variabile viene dichiarata mediante l'uso della parola chiave **`var`**:

```
var i;
```
dichiara la variabile `i`

```
var j = 0;
```
dichiara la variabile `j` e le assegna il valore `0`

```
var s = "casa";
```
dichiara la variabile `s` e le assegna il valore `"casa"`

Se una variabile viene ri-dichiarata, non perde il suo valore

Variabili locali e globali

`var s = "pluto";` variabile `s` *locale* di tipo stringa con valore iniziale `"pluto"`

`var n = 3;` variabile `n` *locale* di tipo numerico con valore 3

`t = "paperino";` variabile `t` *globale* di tipo stringa con valore iniziale `"paperino"`

`m = n;` variabile `m` *globale* di tipo numerico con valore 3

`u = v;` la variabile `u` ha valore *undefined* (in quanto `v` non è a sua volta definita)

`var b = (3>2);` variabile `b` *locale* booleana con valore `true`

`var o = new Object();` variabile `o` *locale* di tipo `Object` (vuota)

Funzioni in Javascript (I)

Una funzione racchiude una porzione di codice JavaScript che può essere eseguito e viene definita mediante la parola chiave **`function`** nel seguente modo:

```
function nomeFunzione (par1, par2)
{
   istruzioni;
   ...
}
```

Funzioni in Javascript (II)

- ✓ Le funzioni Javascript sono in realtà variabili con valore di tipo **`Function`**
- ✓ Per fare riferimento a una funzione è sufficiente usare il suo nome, o un'espressione equivalente che abbia valore di tipo **`Function`**
- ✓ Una volta ottenuto il riferimento a una funzione è possibile:
  - ✓ chiamare la funzione passandole una lista di parametri
  - ✓ omettere uno o più parametri al termine della lista; in questo caso, tali parametri varranno **`undefined`** nel corpo della funzione
  - ✓ passare come argomento una funzione ad un'altra funzione
  - ✓ assegnare una funzione a una o più variabili
  - ✓ accedere a tutti gli elementi della funzione, per modificarla o ridefinirla, tramite le proprietà di **`Function`**
  - ✓ verificare se una funzione è definita come si farebbe con qualsiasi variabile, ad esempio testandola con un **`if(nome_funzione)`**

Funzioni in Javascript (III)

- ✓ Le funzioni restituiscono il controllo al chiamante al termine del loro blocco di istruzioni
- ✓ È possibile restituire un valore al chiamante, in modo da poter usare la funzione in espressioni più complesse, utilizzando la sintassi `return espressione`
- ✓ L'espressione può essere di qualsiasi tipo; essa viene valutata e il valore risultante è restituito
- ✓ Se la funzione non restituisce nessun valore, Javascript sottintende un "`return undefined`" implicito

Funzioni - Esempi

//funzione con due parametri, dichiarazione diretta

```
function prodotto(a,b) {
  return a*b;
}
```

//oggetto funzione assegnato a una variabile

```
var per = new Function("a","b","return a*b;");
```

```
<script type="text/javascript">
function prodotto(a,b)
{
return a*b;
}
</script>
</head>

<body>
<script type="text/javascript">
var per = new Function("a","b","return a*b;");
document.write(per(4,3));
document.write("<br>");
document.write(prodotto(4,3));
</script>
```

Funzioni – Passaggio di parametri

- ✓ Il passaggio dei parametri alle funzioni Javascript avviene in maniera diversa a seconda del tipo del parametro stesso:
  - ✓ i tipi booleano, stringa, numero e undefined sono passati per valore, cioè nella funzione è presente una copia del valore usato come argomento; cambiamenti locali alla funzione non influenzano il valore dell'argomento usato nella chiamata alla funzione stessa
  - ✓ il tipo oggetto è passato per riferimento; la manipolazione del contenuto dell'oggetto si riflette sull'oggetto usato come argomento

Javascript - Oggetti

- ✓ In Javascript non si possono definire classi, ma solo speciali funzioni dette costruttori che creano oggetti con proprietà e metodi; il nome della funzione costruttore è considerato il nome della classe dell'oggetto

- ✓ Gli oggetti si creano utilizzando l'operatore **`new`** applicato alla loro funzione costruttore: **`o = new Object()`**

- ✓ Un metodo di creazione alternativo consiste nell'utilizzo del costrutto **`{"proprieta": valore, … }`**, che crea un oggetto con le proprietà date

Proprietà degli oggetti

- ✓ Le proprietà di un oggetto Javascript possono contenere valori di qualsiasi tipo
- ✓ Per accedere a una proprietà, si possono usare due sintassi:
  - ✓ Sintassi “a oggetti”: `oggetto.proprietà`
  - ✓ Sintassi “array”: `oggetto["proprietà"]`
- ✓ Se si tenta di leggere il valore di una proprietà non definita in un oggetto, si ottiene il valore **`undefined`** (come per ogni variabile non assegnata)
- ✓ È possibile aggiungere dinamicamente proprietà agli oggetti semplicemente assegnando loro un valore
- ✓ Non è possibile aggiungere proprietà a variabili che non siano di tipo oggetto (cioè che non siano oggetti predefiniti o creati con **`new`**)

Proprietà degli oggetti - Esempi

`var o = new Object();` // creazione esplicita di un oggetto

`var v = o.pippo;` // `v` è `undefined`

`o.pluto = 3;` // adesso `o` ha una proprietà `pluto`, di tipo `Number`, con valore 3

`v = o.pluto;` // adesso `v` è una variabile di tipo `Number` e vale 3

`v.paperino = "ciao";` // è un errore: una variabile può accettare l'aggiunta di proprietà solo se è di tipo oggetto

`var o2 = {"pippo": "ciao", "pluto": 3};` // creazione implicita di un oggetto

`v = o2["pluto"];` // equivalente a `v =o2.pluto`

`var nome = "pippo";` // adesso `nome` è una variabile di tipo `String` e vale "pippo"

`v=o2[nome];` // accesso a una proprietà con nome dinamico assegnato a una seconda variabile

Oggetti Javascript - Metodi

- ✓ I metodi di un oggetto Javascript sono semplicemente proprietà di tipo `function`
- ✓ Per accedere a un metodo si possono usare le stesse sintassi viste per le proprietà
- ✓ Per aggiungere un metodo a un oggetto, è sufficiente creare una proprietà col nome del metodo ed assegnarvi
  - una funzione già definita
  - una funzione anonima: `function(parametri) {corpo}`
- ✓ I metodi possono essere aggiunti in qualsiasi momento a un oggetto, esattamente come le proprietà
- ✓ I metodi di un oggetto, per far riferimento alle proprietà dell'oggetto in cui sono definiti, devono utilizzare la parola chiave `this`: `this.proprietà`

Oggetti e Metodi - Esempi


```
var o = new Object();
```

`o.metodo1 = function(x) {return x;}` // aggiunge la funzione specificata come `metodo1` all'oggetto

`o["metodo2"] = f;` // aggiunge la funzione `f` (se esistente) come `metodo2` all'oggetto

`o.metodo1` // questa espressione restituisce l'oggetto `function` che rappresenta il `metodo1`

```
o.metodo1(3);
```

`o["metodo1"](3);` // due chiamate equivalenti al `metodo1`

`var o2 = {"pippo": "ciao", "pluto": 3, "metodo3": function(x) {return x;}}` // definizione di un metodo all'interno della sintassi di creazione

```
var o3 = new Object();
```

`o3.metodo3 = o.metodo1` // il `metodo3` dell'oggetto `o3` è una copia del `metodo1` dell'oggetto `o`

Altri esempi


```
function myObject(a) {
this.v = a+1;
this.w = 0;
this.m = function(x) {return this.v+x;}
}

var o = new myObject(2);
```

/*

L'oggetto `o` avrà due proprietà (`v` e `w`), una delle quali inizializzata tramite il parametro della funzione costruttore, e un metodo (`m`) che restituisce il valore della proprietà `v` sommata al suo argomento

*/

`o.m(3);` // ritorna 6

`o.getW = function() {return this.w;}` // aggiunta dinamica di membri all'oggetto (NON al costruttore)

`o.getV = function() {return v;}` // SBAGLIATO! Fa riferimento alla variabile GLOBALE v!

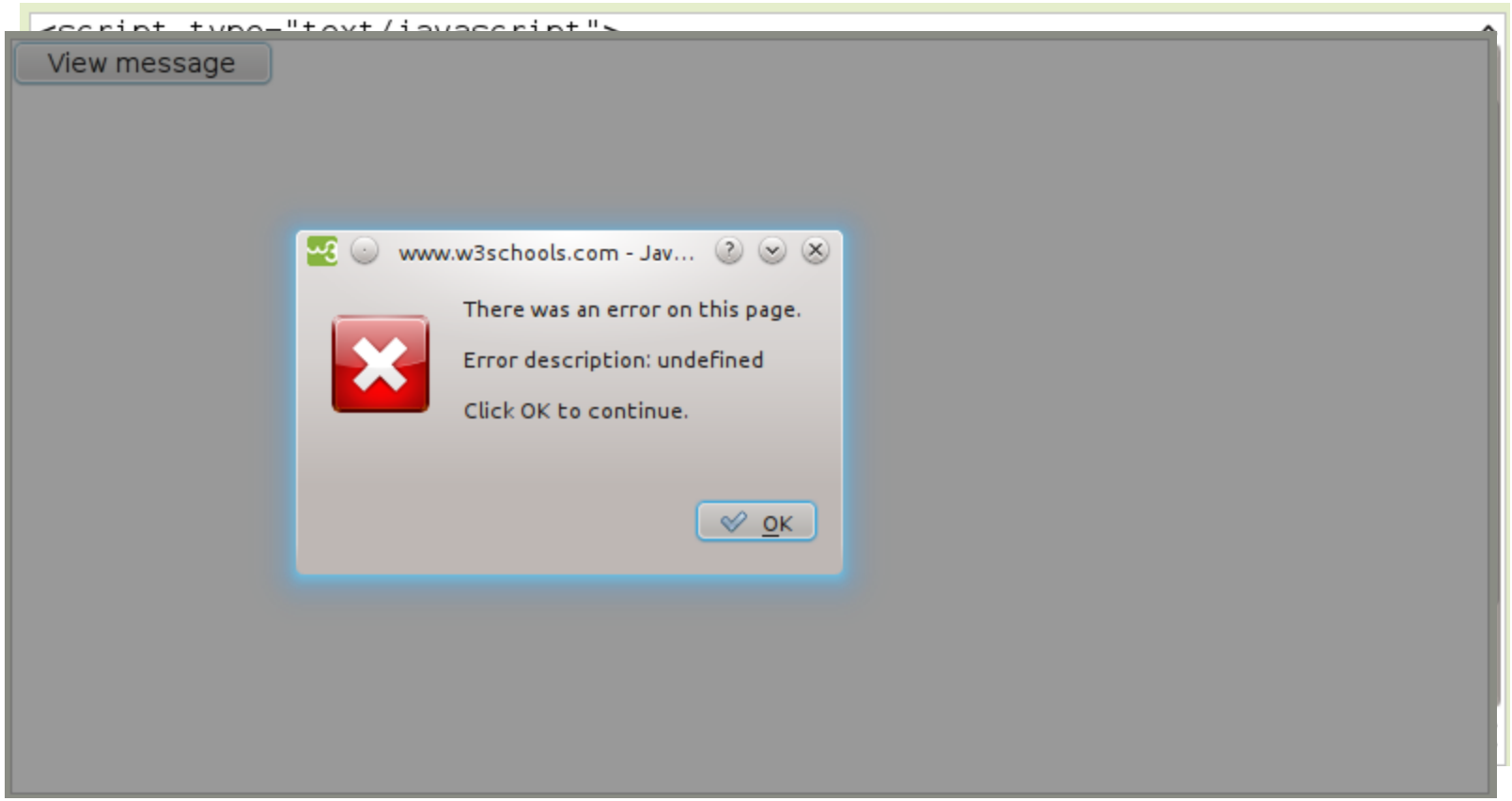
Gestione delle eccezioni

- ✓ Nelle versioni più recenti di Javascript è stato introdotto anche un sistema di gestione delle eccezioni in stile Java
- ✓ Un'eccezione segnala un imprevisto, spesso un errore, all'interno della normale esecuzione del codice
- ✓ Un'eccezione può venire sollevata dalle librerie di Javascript o dal codice scritto dall'utente, attraverso la parola chiave **`throw`**
- ✓ Per gestire le eccezioni, è possibile avvalersi del costrutto **`try…catch…finally`**

Gestione delle eccezioni - Esempio


```
<script type="text/javascript">
var txt="";
function message()
{
try
  {
  adddlert("Welcome guest!");
  }
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.description + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }
}
</script>
</head>
<body>
<input type="button" value="View message" onclick="message()" />
</body>
```

Gestione delle eccezioni - Esempio



La clausola throw


```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","");
try
  {
  if(x>10)
    {
    throw "Err1";
    }
  else if(x<0)
    {
    throw "Err2";
    }
  else if(isNaN(x))
    {
    throw "Err3";
    }
  }
catch(er)
  {
  if(er=="Err1")
    {
    alert("Error! The value is too high");
    }
  if(er=="Err2")
    {
    alert("Error! The value is too low");
    }
  if(er=="Err3")
    {
    alert("Error! The value is not a number");
    }
  }
</script>
</body>
</html>
```

Perché Javascript?

- ✓ JavaScript è utilizzato per:
  - ✓ verificare la *correttezza e la completezza dell'input dell'utente* (tipicamente attraverso controlli sui valori inseriti nei campi di una form)
  - ✓ implementare *animazioni* all'interno delle pagine HTML (modificando "on-the-fly" il codice HTML della pagina)
  - ✓ intercettare e gestire le interazioni degli utenti con le pagine HTML, attraverso un *modello "ad eventi"*
  - ✓ minimizzare la frequenza di interazioni tra client e server, gestendo lato client le operazioni che non è necessario veicolare lato server
- ✓ A tal fine, Javascript si "appoggia" sul modello del *DOM (Document Object Model)* e sull'uso della *tecnologia AJAX*
- ✓ Utilizzeremo le potenzialità di Javascript attraverso l'utilizzo di jQuery, una libreria molto diffusa

DOCUMENT OBJECT MODEL

L'ambiente del Web browser

Esistono due modelli ad oggetti per definire l'interfaccia e i vari aspetti del browser e del documento manipolabili con Javascript

- ✓ un modello ad oggetti del browser (BOM – Browser Object Model)
- ✓ un modello ad oggetti del documento (DOM – Document Object Model)

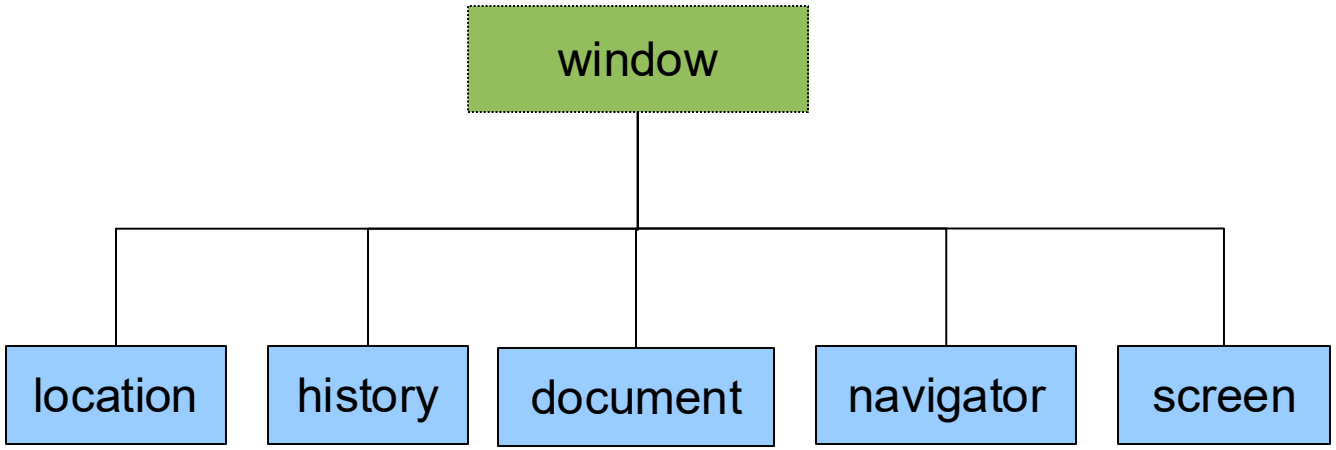
Esiste un modello di programmazione azionato dagli eventi (*event-driven*)

- ✓ ad esempio, se si preme un bottone o si carica una pagina si genera un evento che può essere controllato (tramite opportuni *event handlers*)

Il BOM – Browser Object Model

- ✓ Permette di interagire con il browser, offrendo l'accesso a varie caratteristiche dell'ambiente in cui il browser è in esecuzione: la finestra del browser, le caratteristiche dello schermo, la cronologia del browser, etc.
- ✓ Nel modello BOM va posta molta attenzione alle problematiche *cross-browser*
- ✓ L'oggetto di base per ottenere tutto questo è l'oggetto **`window`**
- ✓ Tra i metodi più comuni dell'oggetto **`window`** troviamo **`alert()`**, **`prompt()`**, **`confirm()`** (si vedano gli approfondimenti)

L’albero del BOM



Per maggiori info: https://www.w3schools.com/js/js_window.asp

L’albero del DOM

- ✓ Possiamo rappresentare ogni documento HTML attraverso un albero
- ✓ Alla radice c'è il nodo che rappresenta il tag **HTML**, i suoi figli sono i nodi che rappresentano **HEAD** e **BODY**
- ✓ DOM Livello 1 fornisce le API per accedere ai nodi dell'albero

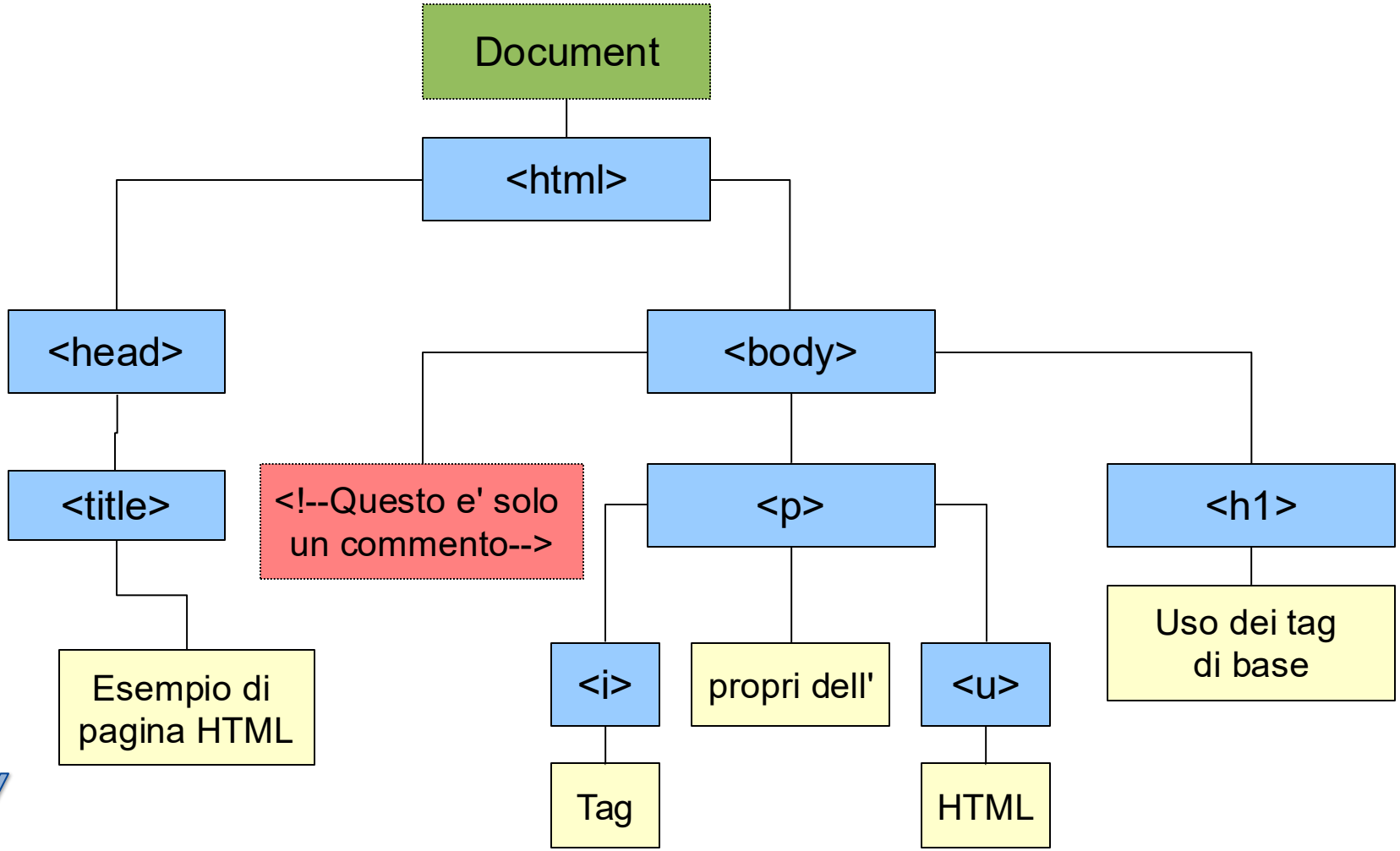
Esempio - HTML


```
<html>
   <head>
      <title>Esempio di pagina HTML</title>
   </head>

   <body>
    <!-- questo e' solo un commento -->
      <h1>Uso dei tag di base</h1>
      <p><i>Tag</i> propri dell'<u>HTML</u><p>
   </body>

</html>
```

HTML visto come albero

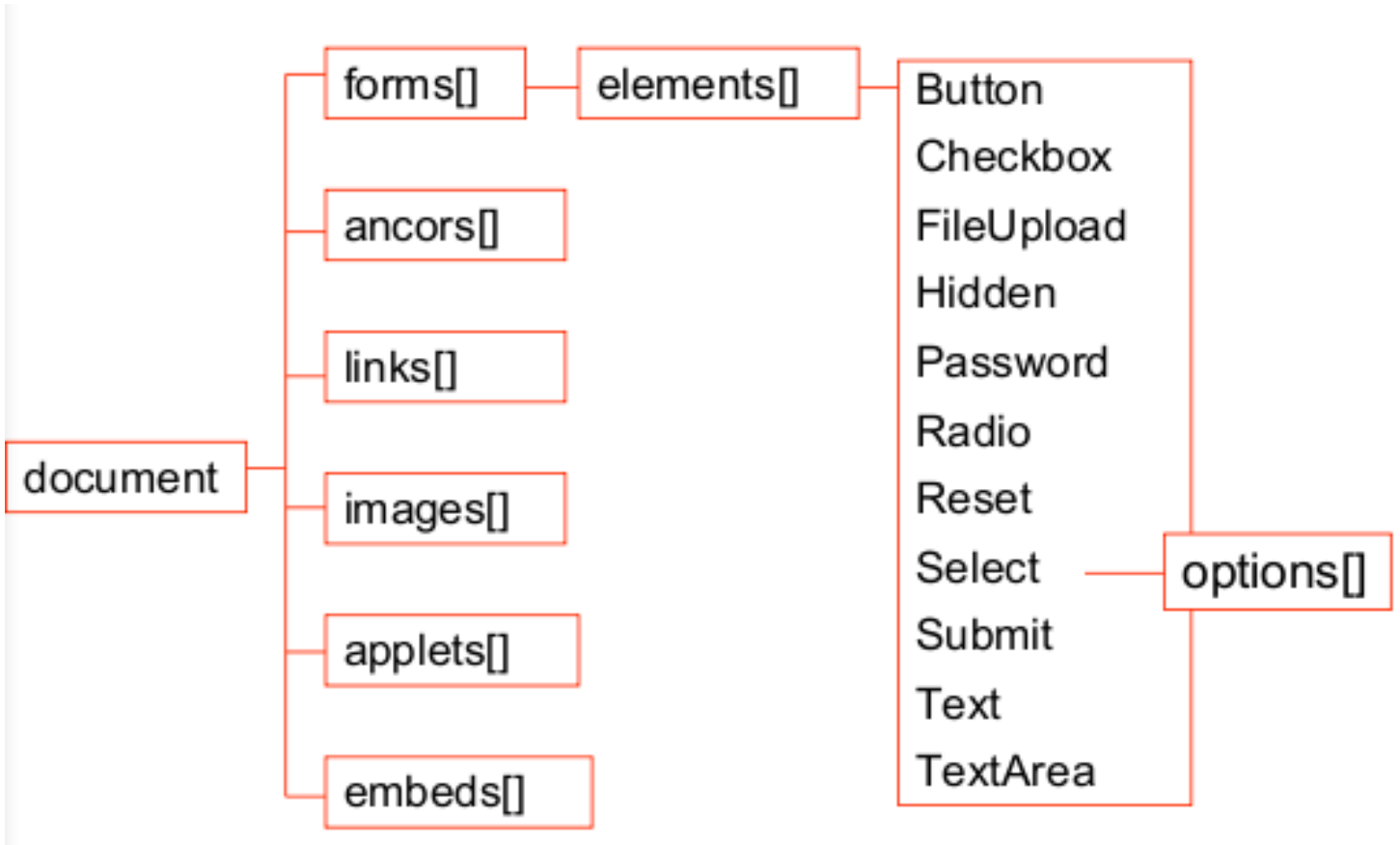


Il DOM – Document Object Model

Fornisce l'accesso al contenuto della finestra del browser, all'interno dei tag HTML

- ✓ DOM Level 0 – spesso chiamato modello ad oggetti tradizionale o classico, permette l'accesso ad alcuni elementi particolari, come immagini, link, elementi di una form, etc.
- ✓ DOM Level 1 – permette di accedere a tutti gli elementi HTML, manipolando i nodi del documento tramite l'oggetto **`document`**
- ✓ DOM Level 2 – permette di far riferimento alle proprietà CSS degli elementi, per esempio per implementare animazioni

DOM – Level 0


document
forms[]
elements[]
Button
Checkbox
FileUpload
Hidden
Password
Radio
Reset
Select
options[]
Submit
Text
TextArea
ancors[]
links[]
images[]
applets[]
embeds[]

Accedere agli elementi del DOM

Per accedere agli elementi del DOM si utilizza la classica dot notation

Ad esempio, con l'espressione seguente

- ✓ `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]`) della prima form (`forms[0]`) del documento

Il ciclo for..in

Il ciclo `for..in` è utilizzato per processare in maniera ciclica le proprietà di un oggetto

```
for (variable in object)
{
    code to be executed
}
```

Il ciclo for..in - Esempio

Esempio


```
<html>
    <head>
        <title>Esempi Javascript - Window properties</title>
    </head>
    <body>
        <script type="text/javascript">
            var prop = "";
            var i = 0;
            for (prop in window) {
                i++;
                document.write(i + ". <b>");
                document.write(prop + ":</b>");
                document.write(window[prop] + "<br>");
            }
        </script>
    </body>
</html>
```

Il ciclo for..in - Esempio

Esempio

1. **window:**[object Window]
2. **self:**[object Window]
3. **document:**[object HTMLDocument]
4. **name:**
5. **location:**https://elearning.unibs.it/pluginfile.php/836621/mod_resource/content/19/windowProperties.html
6. **customElements:**[object CustomElementRegistry]
7. **history:**[object History]
8. **navigation:**[object Navigation]
9. **locationbar:**[object BarProp]
10. **menubar:**[object BarProp]
11. **personalbar:**[object BarProp]
12. **scrollbars:**[object BarProp]
13. **statusbar:**[object BarProp]
14. **toolbar:**[object BarProp]
15. **status:**
16. **closed:**false
17. **frames:**[object Window]
18. **length:**0
19. **top:**[object Window]
20. **opener:**null
21. **parent:**[object Window]
22. **frameElement:**null
23. **navigator:**[object Navigator]

229. **i:**229
230. **TEMPORARY:**0
231. **PERSISTENT:**1
232. **addEventListener:**function addEventListener() { [native code] }
233. **dispatchEvent:**function dispatchEvent() { [native code] }
234. **removeEventListener:**function removeEventListener() { [native code] }

PROGRAMMAZIONE WEB

SCRIPTING CLIENT-SIDE IN JAVASCRIPT INTRODUZIONE

Prof. Ada Bagozi
ada.bagozi@unibs.it