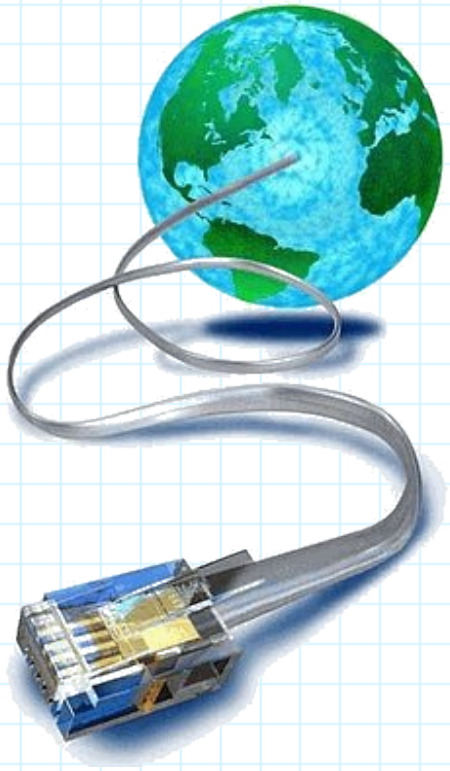
Dynamic HTML

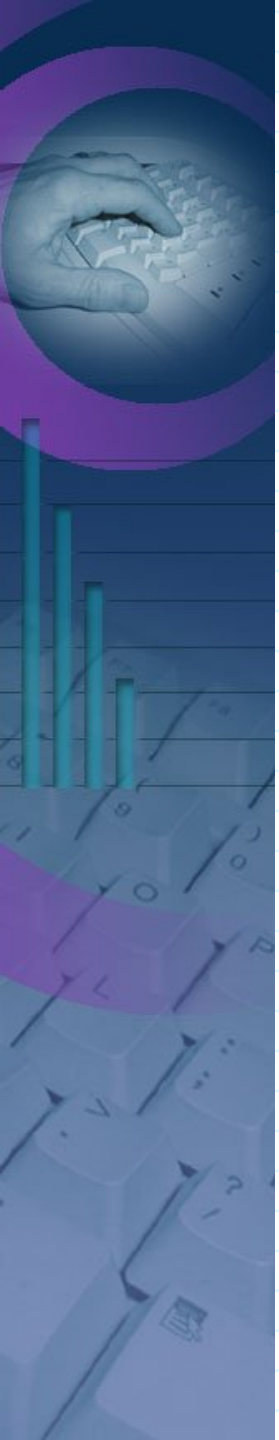


Applicazioni web

Michele Tomaiuolo

tomamic@ce.unipr.it

http://www.ce.unipr.it/people/tomamic



Sommario

- Dynamic HTML: elementi di raggruppamento; markup semantico
- CSS: sintassi; proprietà
- Javascript: sintassi, oggetti DOM, esempi

Dynamic HTML

- Non è uno standard definito dal W3C
  - Marketing Netscape e Microsoft per tecnologie supportate dalle versioni 4.x dei loro browser
- È una combinazione di tecnologie per creare pagine web dinamiche
  - W3C: termine con cui alcune aziende descrivono una combinazione di Html (≥4.0), fogli di stile e Javascript (per ottenere documenti animati ecc.)

Tag generici: div, span

- Aggiungere struttura e semantica ai documenti
  - `span` raggruppa contenuto inline
  - `div` raggruppa contenuto di livello blocco
  - Spesso assegnati attributi `id` e `class`
- Non impongono nessun altro vincolo di presentazione al contenuto
  - Gli autori possono usare questi elementi assieme a fogli di stile per adattare i documenti HTML ai loro bisogni e gusti

Id vs. class

- Mentre uno stesso valore di `class` può essere attribuito a molti elementi su una pagina...
- `id` deve essere unico all'interno del documento!
- Non si può applicare uno stesso valore di `id` a più elementi
- Si possono assegnare più classi ad un solo elemento, separate da spazio. Es.

```
<p class="news gossip">Bla bla.</p>
```

Markup semantico

- Documenti privi di markup di presentazione
- Definire un vocabolario di classi semantiche...

  da assegnare agli elementi con attributo `class`

  la cui presentazione può essere specificata in fogli di stile validi per tutto il sito
- Indicazioni di mozilla.org
  http://www.mozilla.org/contribute/writing/markup
- Accessibilità dei contenuti web
  http://www.w3.org/TR/WCAG10-HTML-TECHS/

Cascading Style Sheets

«In principio il web era popolato di semantici tag p ed h1; ma presto arrivarono font, center, color; le tabelle nascoste erano in agguato; era già scoppiata la Guerra dei Browser, tra Netscape e IE.»

Dopo specifiche W3C per HTML 4.0 e stili, tendenza a miglior supporto di standard

- CSS: migliore semantica e lavoro risparmiato!
  - Un solo file, esterno ai contenuti, controlla la presentazione di molte pagine web

http://www.w3.org/Style/CSS/

Stili a cascata

- Stili raccolti a cascata in “foglio di stile virtuale”
  - Default del browser (priorità più bassa)
  - Foglio di stile esterno (per molti documenti)
  - Foglio di stile interno (per un singolo doc.)
  - Stile inline (per un singolo elemento, priorità)
- Gli stili **non** sono HTML, hanno sintassi diversa
- Nota per le pagine seguenti
  - Contenuto dei file .html in verde, .css in blu, .js in viola

Foglio di stile esterno

- Ideale, soprattutto, per un sito di molte pagine
  - Cambiare l'aspetto, modificando un solo file
- Ogni pagina deve essere collegata al file css
  - `<head><link rel="stylesheet" type="text/css" href="mystyle.css" /></head>`
  - Browser formatta secondo `mystyle.css`
- CSS scritto con comune editor di testo:
  - ```
    body {background: url("images/back40.gif")}
    p {margin-left: 20px}
    ```
    ```

Foglio di stile interno

- Tag `<style>` nella sezione head della pagina
- Il browser leggerà le definizioni di stile, e formatterà il documento in accordo ad esse

```
<head>
  <style type="text/css">
    hr {color: sienna}
    p {margin-left: 20px}
    body {background-image: url("img/back4.png")}
  </style>
</head>
```

Stile inline

- Attributo `style` in elemento da formattare
- Si perdono molti dei vantaggi dei fogli di stile
- Si mischia il contenuto con la presentazione
- Es. cambiare il colore e il margine sinistro di un singolo paragrafo
  - ```

      This is a paragraph

    ```
    ```

Sintassi degli stili

- La più semplice regola css è composta di tre parti: un selettore, una proprietà ed un valore:
  - `selector {property: value}`
- Selettore: es. un elemento html da ridefinire
- Proprietà: aspetto cui assegnare nuovo valore
- Es. `body` ed elem. contenuti con testo in nero
  - `body {color: black}`

Attributi e selettori multipli

```
p {
   text-align: center;
   color: red;
   font-family: "sans serif";
}
h1, h2, h3, h4, h5, h6 {
   color: rgb(0, 255, 0);
}
img {
   float: right;
   padding: 5px;
   border: 1px solid #0000ff;
   margin: 10px;
}
```

Più proprietà → separate da punto-e-virgola

Valore composto da più parole → tra virgolette

Più selettori → separati da virgola

Selettori di classe e id

```
.important {
  text-align: center;
}

p.gossip {
  display: none;
}

#wer345 {
  font-size: 32px;
  font-weight: bold;
  font-style: italic;
}
```

```
<h1 class="important">
  Centered heading
</h1>

<p class="news gossip">
  Gossip's not displayed.
</p>

<h1 id="wer345">
  Some specific heading
</h1>

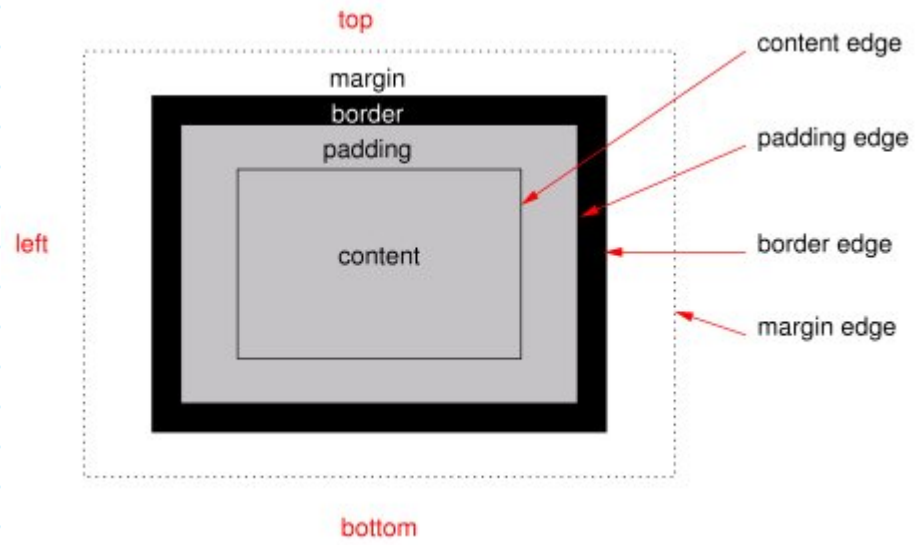
<p class="important">
  More text.
</p>
```

Colori e dimensioni

- Css1: 16 colori, come palette VGA di Windows
  - *aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow*
  - *Es. fuchsia == rgb(255, 0, 255) == #FF00FF*
- Dimensioni box e testo
  - px: in pixel (fissa rispetto risoluzione monitor)
  - pt: punti tipografici, 1/72 pollice
  - em: rispetto a dimensione font (carattere X)
  - %: rispetto alla dimensione (di font, o spazio) dell'elemento genitore

Box model

- margin, padding, border-width – 1-4 valori
- border-style – *none, dotted, dashed, solid...*
- border-color – Colore
- border-radius – 1-4 valori (CSS3)



Sfondo e ombra

- background-color – Colore di sfondo
- background-image – *url(…), none*
- background-image – *linear- / radial-gradient(...)* (CSS3)
- background-repeat – *repeat, repeat-x / -y, no-repeat*
- background-attachment – *scroll, fixed*
- background-position – *top left, top center...*
- background-size – Dimensioni immagine (CSS3)
- box-shadow – 2-4 valori (offset-x, offset-y, blur, distanza) + colore (CSS3)

Testo

- color – Colore del testo
- text-align – *left, right, center, justify*
- vertical-align – Inline, relativo alla riga di testo – *top, middle, bottom, baseline, sub, super*
- text-decoration – *none, underline, overline, line-through*
- font-style – *normal, italic*
- font-weight – *normal, bold*
- font-size – Dimensione
- font-family – Lista con priorità di nomi e/o di famiglie di font (*serif, sans-serif, cursive, fantasy, monospace*)

Posizionamento

- position – Modalità di posizionamento – *relative, absolute, fixed*
- float – Elementi flottanti – *left, right, none*
- z-index – *numero* (valori più alti in primo piano)
- overflow – *visible, hidden, scroll, auto*
- visibility – *visible, hidden* (occupa spazio)
- display – *block, inline, none* (non occupa spazio)

- Blocco centrato:
  - `margin-left: auto; margin-right: auto; width: 50%;`

JavaScript

- Netscape: aggiungere interattività in pagg. HTML
  - Linguaggio interpretato (no compilazione), tipizzazione debole, object-based
  - Scripting per HTML, supportato dai maggiori browser
  - Scripting in applicazioni diverse
- Java e JavaScript non sono la stessa cosa
  - In effetti, sono linguaggi completamente diversi

https://developer.mozilla.org/en/JavaScript

http://www.ecmascript.org/

JavaScript per il Web

- Sintassi JavaScript molto semplice
  - Autori HTML non sempre sono programmatori
  - Ma quasi chiunque può inserire brevi "*snippets*" di codice nelle pagine
- Inserire testo dinamicamente
- Reagire ad eventi
- Modificare la struttura degli elementi HTML
- Controllare dati

Script interni

- Tag <script> per inserire script in pag. HTML

```
<html><body>
  <script type="text/javascript">
    document.write("Hello World!")
  </script>
</body></html>
```

- Browser più vecchi mostrano lo script → tag di commento HTML

```
<script type="text/javascript">
<!--
  /* some statements */
//-->
</script>
```

Script esterni

- Script in un file di testo separato, es. `xyz.js`
  - `document.write("This script is external")`
  - Stesso script eseguito su più pagine
  - Nessun problema di versione e sintassi
- Poi si può richiamare lo script da qualsiasi pagina, usando l'attributo `src` di `script`
  - ```
    <html><body>
    <script type="text/javascript"
       src="xyz.js"></script>
    </body></html>
    ```
    ```

Linee guida JavaScript

- Maiuscole/minuscole
  - Linguaggio *case-sensitive*; es. parole diverse:
  - `example, Example, EXAMPLE`
- Punti e virgola
  - Istruzioni possono terminare con punto e virgola
  - … anche se non è obbligatorio
  - Necessario se più istruzioni su una sola riga
- Spaziatura
  - Nelle istruzioni, come HTML: ignorati multipli spazi, tabulazioni, a capo
  - Riconosciuti nelle stringhe

Variabili

- Si può creare una variabile con l'istruzione `var`:
  - `var someName = someValue`
- Ma anche senza `var` :-)
  - `someName = someValue`
- Var. locale: dichiarata dentro una funzione
  - Accessibile solo nella funzione, distrutta all'uscita
  - Variabili locali con lo stesso nome, dichiarate in funzioni diverse, sono variabili distinte
- Var. globale: dichiarata fuori da ogni funzione
  - Tutte le funzioni possono accedervi
  - Esistenza comincia con la dichiarazione e termina con la chiusura della pagina

Operatori

- Aritmetici
  - `+, -, *, /, %, ++, --`
- Assegnamento
  - `=, +=, -=, *=, /=, %=`
- Confronto
  - `==, !=, >, >=, <, <=`
- Logici
  - `&&, ||, !`

Istruzioni conditionali

- If
  - `if (condition) {`
    `statements1`
    `}`
- if… else
  - `if (condition) {`
    `statements1`
    `} else {`
    `statements2`
    `}`
- Switch
  - `switch (expr) {`
    `case label1:`
    `Statements1;`
    `break;`
    `case label2:`
    `statements2;`
    `break;`
    `…`
    `default:`
    `statements;`
    `}`

Istruzioni di ciclo

- for
  - ```
    for (initialization; condition; increment) {
      statements
    }
    ```
- do-while
  - ```
    do {
      statements
    } while (condition)
    ```
- while
  - ```
    while (condition) {
      statements
    }
    ```
    ```

Funzioni

- Per creare una funzione, bisogna specificarne il nome, gli argomenti e le istruzioni
  - ```
    function total(a, b) {
      var result = a + b
      return result
    }
    ```
- Poi si può chiamare la funzione
  - ```
    sum = total(2, 3)
    ```

Oggetti

- Paradigma object-based (senza classi); ogni oggetto ha i suoi *campi* e i suoi *metodi*
- Si può creare un oggetto in due passi
  - Si scrive un metodo costruttore (def. tipo)

```
function Car(make, model, year) {
  this.make = make
  this.model = model
  this.year = year
}
```

  - Si crea una istanza con l'operatore `new`

```
myCar = new Car("Eagle", "Talon TSi", 1993)
```

- Si può poi rimuovere l'oggetto con op. `delete`

```
delete myCar
```

Proprietà degli oggetti

- Proprietà e array associativi sono intimamente correlati
- Diverse interfacce per la stessa struttura dati

```
myCar.make = "Ford"
myCar.model = "Mustang"
myCar.year = 1969
```

```
myCar["make"] = "Ford"
myCar["model"] = "Mustang"
myCar["year"] = 1969
```

Metodi degli oggetti

- Metodo: funzione associata ad un oggetto
  - ```
    function displayCar() {
      var result = "A Beautiful " + this.year
         + " " + this.make + " " + this.model
      return result
    }
    ```
- Si può poi associare ad un oggetto esistente
  - ```
    function Car(make, model, year) {
      this.make = make
      this.model = model
      this.year = year
      this.displayCar = displayCar
    }
    ```
- Infine si può invocare nel contesto di un oggetto
  - ```
    car1.displayCar()
    ```
    ```

Oggetto String

- Wrapper (automatico) per il tipo primitivo *stringa*

```
s1 = "foo" // creates a string literal
values2 = new String("foo") // an object
```

- Proprietà
  - `length` – Numero di caratteri in una stringa
- Metodi
  - `charAt, indexOf, concat, split, substring, substr, toLowerCase, toUpperCase` ...

Oggetto Array

- Ogni valore associato ad un indice numerico
  - ```
    var family_names = new Array(3)
        family_names[0] = "Tove"
        family_names[1] = "Jani"
        family_names[2] = "Stale"
    ```
- Proprietà
  - `length` – Numero di elementi nell'array
- Metodi
  - `concat`, `join`, `reverse`, `sort`, `slice` …
    ```

Oggetto Date

- Memorizza istanti di tempo
  - Internamente, millisecondi da 1970-01-01
- Molti metodi, nessuna proprietà
  - `getTime(), setTime(x)`
  - `getFullYear(), getMonth(), getDate(), getHours(), getMinutes(), getSeconds(), getMilliseconds(), getDay()`
  - `setFullYear(x), setMonth(x)` **...**
  - `parse(x), toString()`

Oggetto Math

- Predefinito: costanti e funzioni matematiche
- Proprietà
  - `E, PI` – Numero di Neplero e Pi greco
- Metodi
  - `abs(x), max(x,y), min(x,y)`
  - `floor(x), ceil(x), round(x)`
  - `sin(x), cos(x), tan(x)` ...
  - `exp(x), log(x), pow(x,y), sqrt(x)` ...
  - `random()` – Restituisce num. casuale tra 0 e 1

Manipolazione di oggetti

- For ... in : su ciascun elemento di un array o sulle proprietà di un oggetto, eseguite istruzioni
  - `for (variable in object) { statements }`
- With : stabilisce un oggetto di default per un insieme di istruzioni
  - Ogni nome non qualificato: si tenta di risolverlo sulle proprietà dell'oggetto di default
  - Altrimenti, è una variabile locale o globale
  - `with (object) { statements }`

Document Object Model

- API W3C per documenti strutturati HTML e XML
  - Non è una particolare applicazione o prodotto
  - È una interfaccia che i browser devono implementare per connette le pagine web agli script e ai linguaggi di programmazione
- Per gli sviluppatori, fornisce:
  - Rappresentazione strutturata del documento
  - Accesso a questa struttura da script
  - Gestione di una pagina web come un gruppo strutturato di nodi

Document Object Model

- Es. tutti i browser conformi implementano per il documento il metodo `getElementsByTagName`
  - Restituito array con tutti gli elementi `<P>` della pagina HTML
  - ```
    paragraphs =
       document.getElementsByTagName("p");
       // paragraphs[0] is the first <p> element
       // paragraphs[1] is the second, etc.
       alert(paragraphs[0].nodeName);
    ```

http://www.w3.org/DOM/
    ```

Window

- Proprietà
  - `name, location` – Nome e url della finestra
  - `document` – Intero documento
  - `event` – Evento attuale
  - `navigator` – Caratteristiche del browser
  - `self, parent, top` – Gerarchia dei frame
- Metodi
  - `alert(msg), confirm(msg), prompt(msg)` – Visualizza una finestra di dialogo
  - `open(url, name, ...), close()` – Apre o chiude una finestra
  - `setTimeout(expr, millis)` – Valuta una espressione dopo un certo intervallo di tempo

Navigator

- Informazioni sul browser usato dall'utente
- Proprietà
  - `appName, appVersion, userAgent` – Nome, versione browser, agente-utente HTTP
  - `cookieEnabled` – Cookie abilitati, o no?
  - `onLine` – Il sistema è on-line, o no?
  - `cpuClass` – Stringa per classe CPU
  - `platform` – Piattaforma che ospita il browser
  - `userLanguage, browserLanguage, systemLanguage`

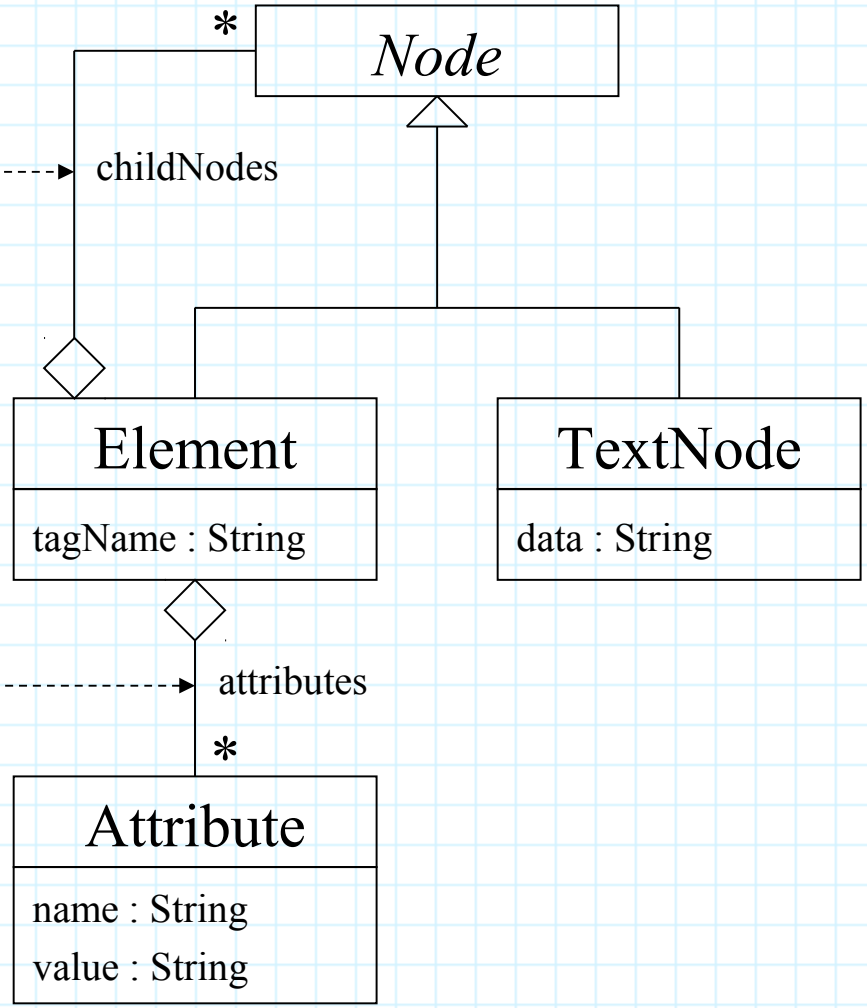
Document

- Documento HTML contenuto nella finestra
- Proprietà
  - `cookie` – Cookie del documento
  - `title` – Titolo del documento
- Metodi
  - `open(), write(text), close()` – Apre, scrive, conclude il documento
  - `getElementsByTagName(tag)`
  - `getElementById(id)` – Trova elemento, dato id
  - `createElement(type), createTextNode()`

Class diagram

List: è importante l'ordine

Set (o map): nome attributo unico, per il suo elemento



Element

- Interfaccia condivisa da tutti gli elementi
  - Interfacce più specializzate per elementi particolari
- Proprietà
  - `childNodes` – Array dei nodi figlio dell'elemento
  - `innerHTML` – Tutto il contenuto, con il markup, all'interno di un dato elemento
  - `style` – Le regole di stile per l'elemento
- Metodi
  - `appendChild(node)` – Aggiunge il nodo specificato come figlio dell'elemento
  - `getElementsByTagName(name)`
  - `getAttribute(name)`, `setAttribute(name, value)`

HTMLFormElement

- Proprietà
  - `action` – Attributo action del form
  - `method` – Metodo http per sottomettere il form
  - `target` – Finestra dove visualizzare la risposta
- Metodi
  - `reset()` – Cancella i valori inseriti dall'utente
  - `submit()` – Sottomette il form

Es. Validare un form

```
function validate() {
  field = document.getElementById('id1');
  if (field.value.length > 0) {
    return true;
  } else {
    alert('Text field empty!'); return false;
  }
}
```

```
<html>
<head><script src="script.js"></script></head>
<body>
  <form onsubmit="return validate()">
    <input id="id1" name="field1" />
    <input type="submit" />
  </form>
</body>
</html>
```

Es. Nascondere elementi

- ```
  function hide(elm) {
    document.getElementById(elm).style.visibility =
  'hidden';
  }
  function show(elm) {
    document.getElementById(elm).style.visibility =
  'visible';
  }
  ```

- ```
  <html>
  <head><script src="hide.js"></script></head>
  <body>
    <div id="id1" onclick="hide('id3')">Hide</div>
    <div id="id2" onclick="show('id3')">Show</div>
    <div id="id3">Some text</div>
  </body>
  </html>
  ```

Es. Muovere un elemento

```
var t = null; var x = 0;
function move() {
  x += 5; if (x > 500) x = 0;
  document.getElementById('id3').style.left= x+'px';
  t = window.setTimeout('move()', 500);
}
function home() {
  window.clearTimeout(t); x = 0; t = null;
  document.getElementById('id3').style.left = '0';
}
```

```
<html>
<head><script src="move.js"></script></head>
<body>
  <div id="id1" onclick="if (t == null) move()">
    Move</div>
  <div id="id2" onclick="home()">Home</div>
  <div id="id3" style="position:absolute;left:0;">
    Some text</div>
</body></html>
```

Es. Creare un elemento

- ```
  function create() {
    var id2 = document.getElementById('id2');
    id2.innerHTML = 'Enter your password: ';
    var newElement = document.createElement('input');
    newElement.setAttribute('type', 'password');
    id2.appendChild(newElement);
  }
  ```
- ```
  <html>
  <head><script src="create.js"></script></head>
  <body>
    <div id="id1" onclick="create()">Create</div>
    <form id="id2">Some text</form>
  </body>
  </html>
  ```