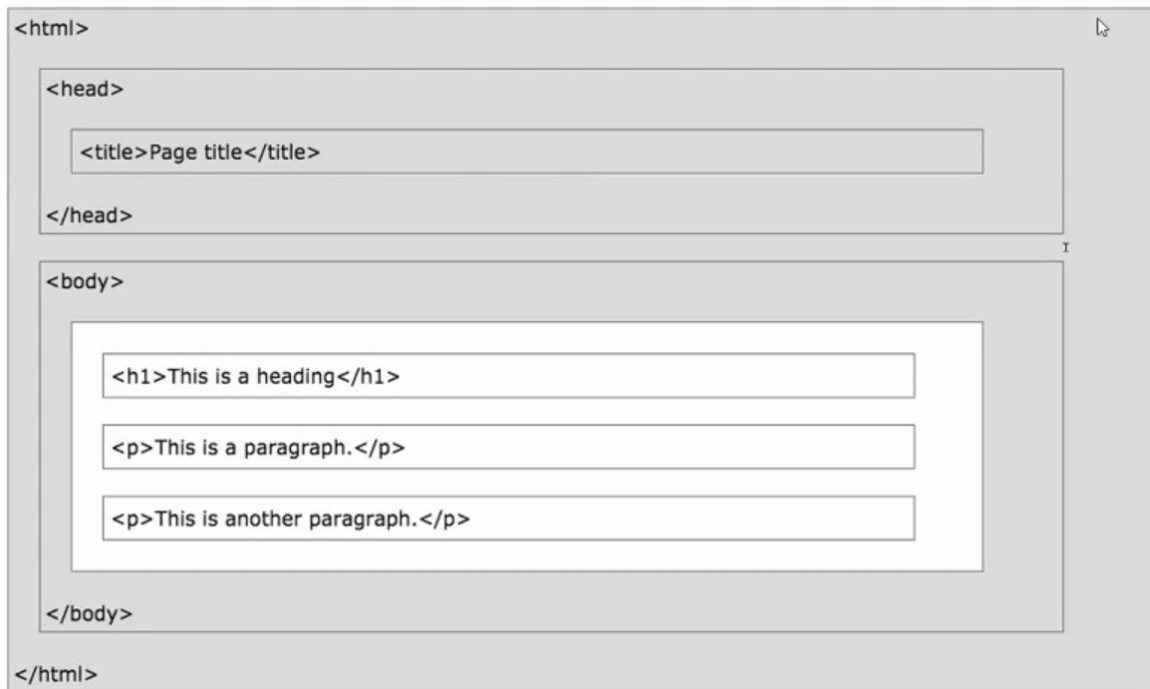
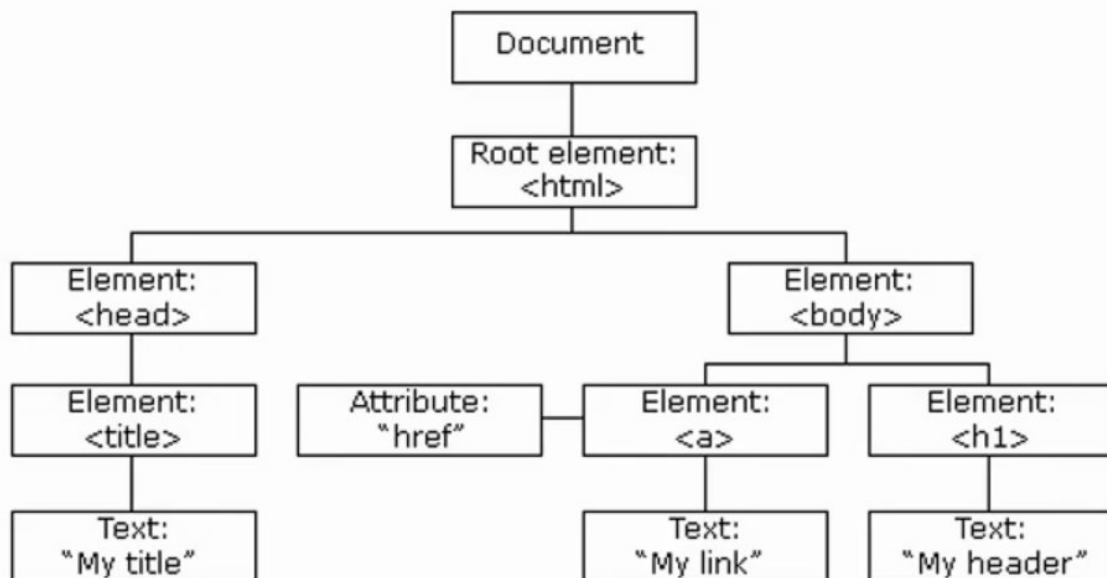


# Introduzione HTML

## Struttura di una pagina



## Document Object Model HTML

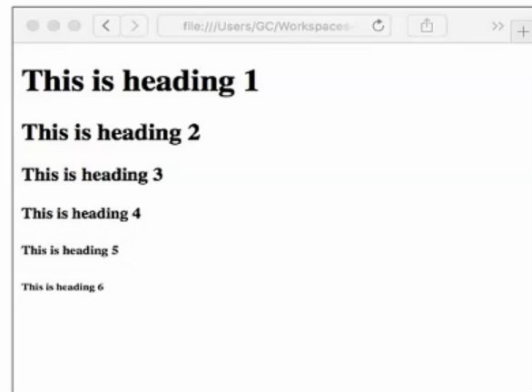


Il DOM ci consente di navigare ed operare sulla pagina e sugli elementi che la compongono a tempo di esecuzione usando **javascript**; questo aspetto ci dà un riferimento su come sia possibile intervenire sul documento per "modificarlo" anche a tempo di esecuzione.

Vediamo un elemento radice, ed ogni nodo in questo albero è un elemento che abbiamo visto in precedenza. All'interno di ogni elemento possono essere posti altri elementi: ad esempio nel `<body>` è possibile inserire degli Headers `<h1>` o testo.

# Elementi basilari

```
<!DOCTYPE html>
<html>
<body>
<h1 id="name">This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>
</body>
</html>
```



Gli heading possono rappresentare diversi livelli di intestazione. Un elemento importante da notare in questa immagine è il fatto che per ogni elemento possiamo introdurre la possibilità di associare (a quell'elemento) un **identificatore**. Possiamo poi (usando javascript) operare su quell'elemento **anche a tempo di esecuzione**.

Abbiamo anche altri elementi:

- `<tagname>Content ...</tagname>`
- Headings
  - multiple levels of heading
- Paragraphs
  - `<p>This is a paragraph</p>`
- Links
  - `<a href="http://www.unisannio.it">This is a link</a>`
  - The destination is given as an attribute "href"
- Images
  - ``
  - Source, alternative text and dimensions are specified as attributes
- Elements can be nested and define a tree



Possiamo anche creare delle tabelle:

- `<table>` to define a table
- `<tr>` to start a row
- `<td>` to define an element

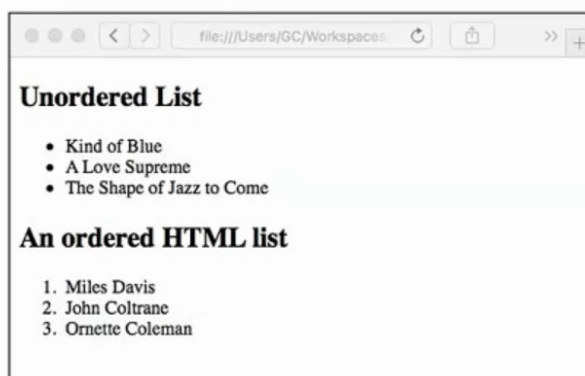


Firstname	Lastname	Age
Maria	Ronaldo	44

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Maria</td>
    <td>Ronaldo</td>
    <td>44</td>
  </tr>
  ...
</table>
</body>
</html>
```

Liste:

- **Unordered (bullet) lists**
  - `<ul>` to define the list
  - `<li>` to define the elements
- **Ordered lists**
  - `<ol>` to define the list
  - `<li>` to define the elements



<b>Unordered List</b> <ul style="list-style-type: none"> <li>• Kind of Blue</li> <li>• A Love Supreme</li> <li>• The Shape of Jazz to Come</li> </ul>
<b>An ordered HTML list</b> <ol style="list-style-type: none"> <li>1. Miles Davis</li> <li>2. John Coltrane</li> <li>3. Ornette Coleman</li> </ol>

```
<html>
<body>

<h2>Unordered List</h2>

<ul>
  <li>Kind of Blue</li>
  <li>A Love Supreme</li>
  <li>The Shape of Jazz to Come</li>
</ul>

<h2>An ordered HTML list</h2>

<ol>
  <li>Miles Davis</li>
  <li>John Coltrane</li>
  <li>Ornette Coleman</li>
</ol>

</body>
</html>
```

**Blocchi:** in HTML un block è un elemento che occupa una linea, ovvero dall'inizio (in alto) della pagina fino alla fine (in basso) o fino a quanto possibile; quindi se aggiungiamo diversi blocchi in successione questi vengono disposti verticalmente.

**Span:** gli span sono come i blocchi, ma vengono disposti orizzontalmente.

- block-level elements start on a new line and takes up the full width available
  - <h1> - <h6>
  - <p>
  - <div>
  - <form>
- <div> element is often used as a container for other HTML elements
  - Often used to define formatting using CSS

**Forms:** abbiamo già usato i form per raccogliere input che vengono inviati al server; infatti servono proprio a questo.

```
<form>
.
  form elements
.
</form>
```

**Fieldset:** ci permette di raggruppare degli elementi, usando il tag:

- <fieldset>
  - groups related data in a form
- <legend>
  - defines a caption for the <fieldset> element


Ad esempio possiamo costruire un form dove alcuni elementi vengono raggruppati grazie al fieldset:

```

<form action="action_page.php">
<fieldset>
  First name:<br>
  <input type="text" name="firstname" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
</fieldset><br><br>
<fieldset>
  <legend>Sex:</legend>
  <input type="radio" name="gender" value="male" checked> Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
</fieldset><br><br>
  <input type="submit" value="Submit">
</form>

```

Renderizzando la pagina otterremo:



First name:  
Mickey

Last name:  
Mouse

Sex:  
☒ Male  
☐ Female

Submit

If you click the "Submit" button, the form-data will be sent to a page called "action\_page.php".

Quindi il Form è uno, ma abbiamo diversi elementi raggruppati.

# Introduzione Javascript

Javascript è un linguaggio di scripting; attraverso javascript possiamo fare delle manipolazioni a tempo di esecuzione. Attraverso js potremmo anche effettuare delle modifiche pericolose: essendo js dello script recuperato dal server, è un codice che garantisce la flessibilità, ma potrebbe anche introdurre dei problemi di sicurezza; dobbiamo quindi evitare che uno script possa effettuare delle operazioni dannose sul client.

JS viene usato per intervenire a tempo di esecuzione sul browser, e quindi sugli elementi che il browser ospita. Possiamo sostituire il testo di una pagina con dell'altro testo, inserire del testo dove prima non era previsto, oppure cambiare il font/colore di un dato testo. Possiamo fare che una componente JS vada a recuperare un elemento di una pagina, per poterli utilizzare successivamente.

Abbiamo parlato della possibilità di manipolare il documento e per operare sul DOM (documento che rappresenta la pagina HTML all'interno del browser) possiamo usare l'oggetto predefinito **document** ed a partire da questo oggetto possiamo usarlo per invocare un metodo che ci permette di recuperare un oggetto dalla pagina HTML:

```
document.getElementById('demo').innerHTML = 'Hello Javascript'
```

con quest'istruzione recuperiamo l'elemento della pagina HTML attraverso il suo nome che avevamo previsto all'interno del tag. Oltre a recuperare l'elemento, grazie ad `.innerHTML = ...` stiamo inserendo una stringa con valore `"Hello Javascript"`.

Possiamo anche recuperare un elemento immagine con:

```
document.getElementById('myImage').src='some_pic.jpg'
```

 invece di usare la source (.src) prevista nel codice HTML dell'elemento, andiamo a modificarla con la stringa 'some\_pic.jpg'.

Possiamo nascondere un elemento con

```
document.getElementById('demo').style.display="none".
```

## Semplice script

```
<html>
<head><title>First JavaScript Page</title></head>
<body>
<h1>First JavaScript Page</h1>
<script type="text/javascript">
  document.write("<hr>");
  document.write("Hello World Wide Web");
  document.write("<hr>");
</script>
</body>
</html>
```



Attraverso il tag **<script>** e **</script>** possiamo specificare uno script di tipo MIME `text/javascript` e collocare lo script direttamente all'interno del body.

# Embedding JavaScript

Non è detto che il codice debba essere inserito all'interno del documento HTML, ma è possibile recuperare del codice JS usando un file separato ed andando a recuperare il file come risorsa aggiuntiva:

```
<html>
<head><title>First JavaScript Program</title></head>
<body>
<script type="text/javascript"
      src="a_source_file.js"></script>
</body>
</html>
```

Inside a source\_file.js

```
document.write("<hr>");
document.write("Hello World Wide Web");
document.write("<hr>");
```

Con l'attributo **src="..."** specifichiamo il nome del file (assumiamo che la risorsa sia presente sullo stesso server web della pagina involucro HTML). All'atto dell'interazione browser-server il client scarica il file js e lo esegue.

🚩 1:17

## Alert(), confirm() e prompt()

Possiamo inserire delle interazioni tra utente e browser/pagina web con i metodi:

```
<script type="text/javascript">
alert("This is an Alert method");
confirm("Are you OK?");
prompt("What is your name?");
prompt("How old are you?", "20");
</script>
```

The diagram illustrates the execution of three JavaScript methods: `alert()`, `confirm()`, and `prompt()`. Arrows point from the corresponding code lines in the script to the respective browser dialog boxes:

- The `alert()` call is linked to a "Microsoft Internet Explorer" dialog box with a yellow warning icon and the text "This is an Alert method".
- The `confirm()` call is linked to an "Explorer User Prompt" dialog box with the text "Script Prompt: Are you OK?".
- The `prompt()` call is linked to another "Explorer User Prompt" dialog box with the text "Script Prompt: What is your name?".

## Identificatori

JS è un linguaggio che ci consente di eseguire del codice all'interno del browser; di conseguenza si prevedono degli identificatori simili a quelli presenti in altri linguaggi. Questi identificatori devono iniziare con una lettera, simbolo '\$' o '\_'. Non possiamo usare degli identificatori riservati al linguaggio.



# Tipi di dati

---

**JS non è un linguaggio tipizzato**, di conseguenza **una variabile non ha un tipo staticamente definito**. Il tipo della variabile viene assegnato a tempo di esecuzione, di conseguenza i tipi di dati sono variabili:

```
<script type="text/javascript">
  var x;                // Now x is undefined
  alert(x);
  x = 5;                // Now x is a Number
  alert(x);
  x = "Paperino";       // Now x is a String
  alert(x);
</script>
```

All'interno di questo script la variabile x prima è di tipo intero, poi diventa di tipo stringa.

## Array

Gli arrays sono simili a quelli presenti in Java, infatti anche in questo caso definiamo un array partendo dall'esplicitazione degli elementi costituenti l'array stesso: `var languages = ["Java", "C", JavaScript];` Possiamo anche avere un array **omogeneo**, ovvero contenente **tipi diversi** (ad esempio contiene stringhe, interi, oggetti, ecc.)

## Oggetti

Gli oggetti in JS sono **collezioni di proprietà** e possono avere anche delle funzioni (simili ai metodi):

```
var person = {firstName:"Eugenio",
               lastName:"Zimeo",
               eyeColor:"green"};
```

Possiamo definire un oggetto all'atto di assegnazione dello stesso ad una variabile

## Funzioni

---

Le funzioni in JS vengono dichiarate in questo modo:



```
// A function can return value of any type using the
// keyword "return"

// The same function can possibly return values
// of different types
function foo (p1) {
    if (typeof(p1) == "number")
        return 0; // Return a number
    else if (typeof(p1) == "string")
        return "zero"; // Return a string

    // If no value being explicitly returned
    // "undefined" is returned.
}

foo(1); // returns 0
foo("abc"); // returns "zero"
foo(); // returns undefined
```

Siccome non esiste il concetto di classe, possiamo definire le funzioni liberamente. Non dobbiamo prevedere né il **tipo di parametro** né il **valore di ritorno**.

In JS è possibile gestire i parametri di una funzione come un array, invece di specificarli come nell'immagine precedente.

```
function sum ()
{
    var s = 0;
    for (var i = 0; i < arguments.length; i++)
        s += arguments[i];
    return s;
}
```

Questa funzione addiziona tutti i parametri passati e ritorna la somma

Se usiamo **arguments** possiamo quindi gestire i parametri come un array, selezionando uno specifico parametro usando `arguments[i]`. Di conseguenza possiamo avere un **numero indefinito di parametri**.

## Funzioni incorporate in JS

- **eval(expression)** L'espressione può essere sia numerica che la visualizzazione di un alert.
  - `eval("3+4")` ritorna 7
  - `eval("alert('Hello')")` chiama la funzione `alert('hello')`
- **isFinite(x)** verifica se il numero passato è finito
- **isNaN(x)** verifica se il numero passato è un numero.
- **parseInt(s)**

- **parseInt(s,radix)**: convertiamo una stringa che contiene nella parte iniziale un numero:
  - parseInt("3 chances") ritorna 3
  - parseInt(" 5 alive") ritorna 5
  - parseInt("come stai fra?") ritorna NaN
  - parseInt("17" , 8) ritorna 15 (boh)

## Eventi

---

Nei documenti HTML possiamo catturare degli eventi causati dall'interazione dell'utente con il browser. Ad esempio un evento potrebbe essere il click su di un link, o su di un button.

Questi eventi possono essere catturati ed associati ad handler, un po' come accade nella programmazione delle interfacce grafiche di java. Se possiamo caratterizzare degli eventi, possiamo associarvi un **handler** in modo da eseguire del codice quando questo evento avviene.

```
<a href="..." onClick="alert('Bye') "> Other Website</a>
```

All'evento **onClick** viene associato l'handler **alert()** (che è una funzione js)

## Event Handlers JS

Event Handlers	Triggered when
onChange	The value of the text field, textarea, or a drop down list is modified
onClick	A link, an image or a form element is clicked once
onDbClick	The element is double-clicked
onMouseDown	The user presses the mouse button
onLoad	A document or an image is loaded
onSubmit	A user submits a form
onReset	The form is reset
onUnLoad	The user closes a document or a frame
onResize	A form is resized by the user

### Esempio onClick

```

<html>
<head>
<title>onClick Event Handler Example</title>
<script type="text/javascript">
function warnUser() {
    return confirm("Are you a student?");
}
</script>
</head>
<body>
<a href="ref.html" onClick="return warnUser()">
<!--
    If onClick event handler returns false, the link
    is not followed.
-->
Students access only</a>
</body>
</html>

```

Vediamo che viene usata la funzione **warnUser()** (JS) come reazione ad un evento.

## Creazione degli oggetti

Esistono diversi modi per creare oggetti:

### Prima opzione

Possiamo usare l'operatore **new Object()** per creare un nuovo oggetto. Possiamo poi associare alla variabile che referencia l'oggetto delle **proprietà**.

Possiamo anche definire delle funzioni dell'oggetto sempre tramite la variabile che lo referencia con `person.sayHi = function(){...}`

```

var person = new Object();

// Assign fields to object "person"
person.firstName = "Eugenio";
person.lastName = "Rudd";

// Assign a method to object "person"
person.sayHi = function() {
    alert("Hi! " + this.firstName + " " + this.lastName);
}

person.sayHi(); // Call the method in "person"

```

## Seconda opzione - Literal Notation

Questa notazione ci consente anche di capire i legami tra JS e servizi REST:

```
var person = {  
  // Declare fields  
  // (Note: Use comma to separate fields)  
  firstName : "Eugenio",  
  lastName : " ",  
  
  // Assign a method to object "person"  
  sayHi : function() {  
    alert("Hi! " + this.firstName + " " +  
      this.lastName);  
  }  
}  
  
person.sayHi(); // Call the method in "person"
```

**firstname** è una proprietà a cui assegnamo un valore attraverso ':', così come le altre proprietà. Abbiamo inoltre la funzione **sayHi** che come nelle proprietà assegnamo un valore **funzione** con ':{'.

Possiamo anche avere delle **proprietà innestate**:

```
var triangle = {  
  // Declare fields (each as an object of two fields)  
  p1 : { x : 0, y : 3 },  
  p2 : { x : 1, y : 4 },  
  p3 : { x : 2, y : 5 }  
}  
  
alert(triangle.p1.y); // Show 3
```

## Costruttore dell'oggetto

Per poter definire un template da cui creare più istanze di oggetti (una sorta di classe) possiamo scrivere una funzione vista come un costruttore:

```
function Person(fname, lname) {  
  // Define and initialize fields  
  this.firstName = fname;  
  this.lastName = lname;  
  
  // Define a method  
  this.sayHi = function() {  
    alert("Hi! " + this.firstName + " " +  
      this.lastName);  
  }  
}  
  
var p1 = new Person("Mario", "Rossi");  
var p2 = new Person("Riccardo", "Bianchi");  
  
p1.sayHi();  
p2.sayHi();
```

Implicitamente assumiamo che, usando **this.**, queste variabile saranno gestite in maniera separata in istanze diverse degli oggetti. Di conseguenza possiamo usare la funzione come oggetto, invocando **new Person()**. Andando ad invocare **sayHi** sui diversi oggetti otterremo "risultati" diversi, proprio perchè questi sono **due istanze diverse dello stesso oggetto** (template di oggetto).

## Oggetti incorporati in JS

Object	Description
Array	Creates new array objects
Boolean	Creates new Boolean objects
Date	Retrieves and manipulates dates and times
Error	Returns run-time error information
Function	Creates new function objects
Math	Contains methods and properties for performing mathematical calculations
Number	Contains methods and properties for manipulating numbers.
String	Contains methods and properties for manipulating text strings

## Gestione delle eccezioni in JS

Le eccezioni possono essere gestite in modo simile a quello già visto in Java, usando la struttura di controllo **try/catch**

```
<html>
<body>
<p id="demo"></p>
<script>
try {
    alert("Welcome guest!");
}
catch(err) {
    document.getElementById("demo").innerHTML = err.message;
}
</script>
</body>
</html>
```

All'interno della sezione script prevediamo il costrutto try/catch che "controlla" un elemento di HTML tramite il suo id.

## L'oggetto XMLHttpRequest

Possiamo usare all'interno di programmi JS un oggetto particolare **XMLHttpRequest** che funziona come il client che abbiamo visto negli esempi dell'esercitazione 10, ovvero possiamo costruire dei messaggi di richiesta HTTP direttamente all'interno del client (programma).

