

Form

- Un documento può contenere più oggetti form
- Un oggetto form può essere referenziato con il suo nome o mediante il vettore **forms[]** esposto da **document**:

document.nomeForm

document.forms[n]

document.forms["nomeForm"]

- Gli elementi del form possono essere referenziati con il loro nome o mediante il vettore **elements[]**

document.nomeForm.nomeElemento

document.forms[n].elements[m]

document.forms["nomeForm"].elements["nomeElem"]

- Ogni elemento ha una proprietà form che permette di accedere al form che lo contiene (vedi esempio “calcolatrice” precedente **this.form**)

Form (2)

- Per ogni elemento del form esistono proprietà corrispondenti ai vari attributi:

id, name, value, type, className...

```
<form name="myForm">  
  Form name:  
  <input type="text" name="text1" value="test">  
  <br/>  
  <input name="button1" type="button"  
    value="Mostra il nome del form"  
    onclick="document.myForm.text1.value=  
              document.myForm.name">  
</form>
```

Form name:

In alternativa potevamo scrivere:

```
onclick="this.form.text1.value=  
          this.form.name">
```

Form (3)

- Proprietà:
 - **action**: riflette l'attributo action
 - **elements**: vettore contenente gli elementi della form
 - **length**: numero di elementi nella form
 - **method**: riflette l'attributo method
 - **name**: nome del form
 - **target**: riflette l'attributo target
- Metodi:
 - **reset()**: resetta il form
 - **submit()**: esegue il submit
- Eventi:
 - **onReset**: quando il form viene resettato
 - **onSubmit**: quando viene eseguito il submit del form

Access to the form elements (accessformelement.html)

```
<!DOCTYPE html>
<html>
  <body>

    <h3>A demonstration of how to access a FORM element</h3>

    <form id="myForm" action="/action_page.php">
      First name: <input type="text" name="fname" value="Donald"><br>
      Last name: <input type="text" name="lname" value="Duck"><br>
      <input type="submit" value="Submit">
    </form>

    <p>Click the "Try it" button to get the URL for where to send the form data when t
      form above is submitted.</p>

    <button onclick="myFunction()">Try it</button>

    <p id="demo"></p>

    <script>
      function myFunction() {
        var x = document.getElementById("myForm").action;
        document.getElementById("demo").innerHTML = x;
      }
    </script>

  </body>
</html>
```

I controlli di un form

- Ogni tipo di controllo (widget) che può entrare a far parte di un form è rappresentato da un oggetto JavaScript:
- **Text:** `<input type="text">`
- **Checkbox:** `<input type="checkbox">`
- **Button:** `<input type="button">` o `<button>`
- **Radio:** `<input type="radio">`
- **Hidden:** `<input type="hidden">`
- **File:** `<input type="file">`
- **Password:** `<input type="password">`
- **Textarea:** `<textarea>`
- **Submit:** `<input type="submit">`
- **Reset:** `<input type="reset">`

```
var checkbox = document.createElement('input');
checkbox.type = "checkbox";
checkbox.name = "name";
checkbox.value = "value";
checkbox.id = "id";
//form container
container.appendChild(checkbox);

var label = document.createElement('label')
label.htmlFor = "id";
label.appendChild(
    document.createTextNode('text...'));
container.appendChild(label);
```

Elementi comuni ai vari controlli

- Proprietà:
 - **form**: riferimento al form che contiene il controllo
 - **name**: nome del controllo
 - **type**: tipo del controllo
 - **value**: valore dell'attributo value
 - **disabled**: disabilitazione/abilitazione del controllo
- Metodi:
 - **blur()** toglie il focus al controllo
 - **focus()** dà il focus al controllo
 - **click()** simula il click del mouse sul controllo
- Eventi:
 - **onBlur** quando il controllo perde il focus
 - **onFocus** quando il controllo prende il focus
 - **onClick** quando l'utente clicca sul controllo

onblur.html

```
<!DOCTYPE html>
<html>
  <body>
```

Enter your name: `<input type="text" id="fname" onblur="myFunction()">`

`<p>`When you leave the input field, a function is triggered which transforms the input text to upper case.`</p>`

```
<script>
  function myFunction() {
    var x = document.getElementById("fname");
    x.value = x.value.toUpperCase();
  }
</script>
```

```
</body>
</html>
```


onfocus.html

```
<!DOCTYPE html>
<html>
  <body>
```

Enter your name: `<input type="text" onfocus="myFunction(this)">`

`<p>`When the input field gets focus, a function is triggered which changes the background-color.`</p>`

```
<script>
  function myFunction(x) {
    x.style.background = "yellow";
  }
</script>
```

```
</body>
</html>
```


Oggetti Text e Password

- Proprietà (get/set):
 - **defaultValue** valore di default
 - **disabled** disabilitazione / abilitazione del campo
 - **maxLength** numero massimo di caratteri
 - **readOnly** sola lettura / lettura e scrittura
 - **size** dimensione del controllo
- Metodi:
 - **select()** seleziona una parte di testo

Oggetti checkbox e radio

- Proprietà (get/set):
 - **checked**: dice se il box è spuntato
 - **defaultChecked**: impostazione di default

Create a checkbox

```
1  <!DOCTYPE html>
2  <html>
3      <body>
4
5          <p>Click the button to create a Checkbox.</p>
6
7          <button onclick="myFunction()">Try it</button>
8
9          <script>
10             function myFunction() {
11                 var x = document.createElement("INPUT");
12                 x.setAttribute("type", "checkbox");
13                 document.body.appendChild(x);
14             }
15         </script>
16
17     </body>
18 </html>
19
```

Validazione di un form

- Uno degli utilizzi più frequenti di JavaScript è nell'ambito della validazione dei campi di un form
 - Riduce il carico delle applicazioni server side filtrando l'input
 - Riduce il ritardo in caso di errori di inserimento dell'utente
 - Semplifica le applicazioni server side
 - Consente di introdurre dinamicità all'interfaccia web
- Generalmente si valida un form in due momenti:
 1. Durante l'inserimento utilizzando l'evento **onChange()** sui vari controlli
 2. Al momento del submit utilizzando l'evento **onClick()** del bottone di submit o l'evento **onSubmit()** del form

Esempio di validazione - 1

```
<head>
  <script type="text/javascript">
    function qty_check(item, min, max)
    {
      returnVal = false;
      if (parseInt(item.value) < min) or
        (parseInt(item.value) > max)
        alert(item.name+"deve essere fra "+min+" e "+max);
      else returnVal = true;
      return returnVal;
    }
    function validateAndSubmit(theForm)
    {
      if (qty_check(theform.quantità,0,999))
      { alert("Ordine accettato"); return true; }
      else
      { alert("Ordine rifiutato"); return false; }
    }
  </script>
</head>
```

Esempio di validazione - 2

```
<body>
  <form name="widget_order"
    action="lwapp.html" method="post">
    Quantità da ordinare
    <input type="text" name="quantità"
      onchange="qty_check(this,0,999)">
    <br/>
    <input type="submit" value="Trasmetti l'ordine"
      onclick="validateAndSubmit(this.form)">
  </form>
</body>
```

```
<form name="widget_order"
  action="lwapp.html" method="post"
  onsubmit="return qty_check(this,0,999)">
  ...
  <input type="submit" />
  ...
</form>
```


Validazione

```
<body>
```

```
<p>Enter a number and click OK:</p>
```

```
<input id="id1" type="number" min="100" max="300">
```

```
<button onclick="myFunction()">OK</button>
```

```
<p>If the number is less than 100 or greater than 300,  
an error message will be displayed.</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
function myFunction() {
```

```
    var inpObj = document.getElementById("id1");
```

```
    if (inpObj.checkValidity() == false) {
```

```
        document.getElementById("demo").innerHTML = inpObj.validationMessage;
```

```
    } else {
```

```
        document.getElementById("demo").innerHTML = "Input OK";
```

```
    }
```

```
}
```

```
</script>
```

```
</body>
```



```
function ValidateEmail(uemail)
{
    var x = uemail.value;
    var atpos = x.indexOf("@");
    var dotpos = x.lastIndexOf(".");
    if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length) {
        alert("Not a valid e-mail address");
        uemail.focus();
        return false;
    }
    else
    {
        return true;
    }
}
```

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Can Validate Input</h1>
<p>Please input a number between 1 and 10:</p>

<input id="numb">
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>

<script>
function myFunction() {
    var x, text;

    // Get the value of the input field with id="numb"
    x = document.getElementById("numb").value;

    // If x is Not a Number or less than one or greater than 10
    if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

JavaScript Can Validate Data

Personal Information

Title

 select title 

☐ **Ph.D. Student**

First name

 enter first name

Last name



 enter last name

Affiliation

 enter affiliation

Personal Information



Title

 select title  



Select a title or select the empty one

☐ **Ph.D. Student**



First name

 michele 

Last name

 risi 

Affiliation

 enter affiliation 

The affiliation is required and cannot be empty

Validazione Form

Javascript error management

- The **try** statement lets you test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.
- The **finally** statement lets you execute code, after try and catch, regardless of the result.
-

JavaScript Errors - Throw and Try to Catch

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
try {
    adddler("Welcome guest!");
}
catch(err) {
    document.getElementById("demo").innerHTML = err.message;
}
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test
Input</button>

<p id="message"></p>

<script>
function myFunction() {
    var message, x;
    message = document.getElementById("message");
    message.innerHTML = "";
    x = document.getElementById("demo").value;
    try {
        if(x == "") throw "is empty";
        if(isNaN(x)) throw "is not a number";
        x = Number(x);
        if(x > 10) throw "is too high";
        if(x < 5) throw "is too low";
    }
    catch(err) {
        message.innerHTML = "Input " + err;
    }
    finally {
        document.getElementById("demo").value = "";
    }
}
</script>
</body>
</html>
```

The finally Statement

```
try {
    tryCode - Block of code to try
}
catch(err) {
    catchCode - Block of code to handle errors
}
finally {
    finallyCode - Block of code to be executed regardless of the try / catch result
}
```


The debugger keyword

- The **debugger** keyword stops the execution of JavaScript, and calls (if available) the debugging function
- This has the same function as setting a breakpoint in the debugger
- If no debugging is available, the debugger statement has no effect

```
var x = 15 * 5;  
debugger;  
document.getElementById("demo").innerHTML = x;
```

Chrome Developer Tools: dalla scheda
Sources (nella modalità Inspector)
inserire i breakpoint sul codice Javascript

JavaScript e JQuery: quali differenze?

- *Sebbene molti aspiranti sviluppatori Web si chiedano quale sia la differenza tra JavaScript e JQuery, in realtà è bene sapere che sono la stessa cosa!!*
- Sinteticamente, JQuery è una libreria specifica per codice JavaScript (sviluppata da terzi) pensata appositamente per semplificare l'attraversamento del DOM di una pagine HTML, la sua animazione, la gestione di eventi, e interazioni Ajax
- Prima di JQuery, gli sviluppatori provvedevano a creare il proprio "framework JavaScript"; ciò permetteva loro di lavorare su specifici bug senza perdere tempo nel debugging di features comuni
- Ciò ha portato alla realizzazione da parte di gruppi di sviluppatori di librerie open source e gratuite

JavaScript Declarations are Hoisted

- In JavaScript, a variable can be declared after it has been used
- In other words; a variable can be used before it has been declared

```
x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                      // Display x in the element

var x; // Declare x
```

Performance

```
for (i = 0; i < arr.length; i++) {
```

```
  l = arr.length;  
  for (i = 0; i < l; i++) {
```

```
    obj = document.getElementById("demo");  
    obj.innerHTML = "Hello";
```

```
var fullName = firstName + " " + lastName;  
document.getElementById("demo").innerHTML = fullName;
```

```
document.getElementById("demo").innerHTML = firstName + " " + lastName
```

Delay JavaScript Loading

- If possible, you can add your script to the page by code, after the page has loaded:

```
<script>
window.onload = downScripts;

function downScripts() {
    var element = document.createElement("script");
    element.src = "myScript.js";
    document.body.appendChild(element);
}
</script>
```

Riferimenti

- Tutorial (JavaScript e HTML DOM):
 - **<http://www.w3schools.com/js/default.asp>**
- JQuery:
 - **<http://api.jquery.com/>**