JQUERY – 3.7.1

Se andiamo oltre la superficie fatta di interazione sociale, contenuti generati dagli utenti e partecipazione attiva, ci accorgiamo che il cosiddetto web 2.0, con cui ci confrontiamo ogni giorno, poggia in gran parte sull'utilizzo intensivo di JavaScript. Un tempo associato soprattutto a fastidiosi popup o pratiche poco sicure, negli ultimi anni questo linguaggio ha guadagnato nuova considerazione, grazie alla diffusione di tecniche come AJAX e alla possibilità di creare effetti dinamici complessi senza dover ricorrere a plugin esterni come Flash.

Dal punto di vista degli sviluppatori, però, l'esecuzione lato client ha sempre rappresentato una fonte costante di complicazioni. Ogni browser integra infatti un proprio motore JavaScript, con peculiarità e differenze (soprattutto Internet Explorer), il che rende difficile garantire che uno script funzioni in maniera uniforme su tutte le piattaforme. A questo si aggiungono limiti strutturali del linguaggio, che lo rendono meno coerente rispetto a linguaggi server-side come PHP o Ruby.

Per risolvere questi ostacoli, nel tempo sono nate librerie e framework pensati per semplificare l'uso di JavaScript, offrendo compatibilità cross-browser e ampliando le funzionalità di base del linguaggio.

Uno dei primi strumenti diffusi in questa direzione è stato Prototype, che affrontava il problema estendendo direttamente il DOM e gli oggetti globali di JavaScript, arricchendoli di nuove funzionalità. In questo modo si colmavano molte lacune del linguaggio, un po' come avviene con l'installazione di plugin nei software. Anche Mootools, nelle sue versioni iniziali, seguiva questo approccio, mentre successivamente si è evoluto in una libreria più vicina alla programmazione a oggetti, mantenendo comunque alcune estensioni per array e stringhe. Il rovescio della medaglia di questo metodo, però, è che modificare gli oggetti globali può portare a conflitti con altre librerie, con il codice dello sviluppatore o addirittura con le nuove versioni dei motori JavaScript.

Nascita di jQuery

- **Anno di nascita**: **2006**, presentato da **John Resig** al BarCamp di New York.
- **Contesto storico**: a metà anni 2000 il **JavaScript nativo** era molto frammentato:
    - Ogni browser (Internet Explorer, Firefox, Safari, Opera) implementava lo standard DOM in modo diverso.
    - Scrivere codice cross-browser era complicato e pieno di "if (browser) …".
- **Obiettivo iniziale**: semplificare il **DOM scripting**, rendere più semplice manipolare HTML, gestire eventi, fare animazioni e AJAX **con un'API unica e coerente**.

Perché si è reso necessario

1. **Compatibilità cross-browser**: scrivere codice una volta, funzionante ovunque.
2. **Sintassi più semplice**: il famoso selettore tipo $('#id') ha reso immediata l'interazione col DOM.
3. **Gestione eventi** unificata (mentre i browser usavano modelli diversi).
4. **AJAX facile**: $.ajax() e $.getJSON() hanno semplificato moltissimo le chiamate asincrone.
5. **Plugin ecosystem**: migliaia di plugin sviluppati dalla community per slider, validazioni, interfacce grafiche.

Utilizzo

Per anni (2006–2015 circa), **jQuery è stato lo standard de facto** nello sviluppo frontend:

- La maggior parte dei siti web lo includeva.
- Ha reso possibile lo sviluppo veloce e compatibile prima dell'esplosione dei framework moderni (React, Angular, Vue).
- È stato spesso distribuito assieme a **jQuery UI** (per componenti grafici) e **jQuery Mobile** (per web mobile).

Evoluzione: dalla prima all'ultima versione

- **jQuery 1.x (2006-2016)**
  - Focalizzato sulla compatibilità con **vecchi browser (anche IE6)**.
  - Molte funzioni "legacy" per supportare ambienti ormai obsoleti.
- **jQuery 2.x (2013-2018)**
  - Eliminato il supporto a IE6/7/8 → codice più snello e prestazioni migliori.
  - API quasi identiche alla 1.x per non rompere la compatibilità.
- **jQuery 3.x (2016-oggi)**
  - Allineato agli **standard moderni di JavaScript** (Promise per AJAX, compatibilità con ES6+).
  - Rimosse API deprecate e comportamenti obsoleti.
  - Migliore compatibilità con browser moderni (Edge, Chrome, Firefox, Safari).
  - Animazioni più fluide, correzioni di bug e miglior supporto a moduli JS.
- **Ultima versione: 3.7 (2023)**
  - È considerata una versione di **manutenzione e stabilità**.
  - Migliorate performance e gestione eventi.
  - Aggiustamenti per compatibilità con i browser moderni.
  - Non introduce novità "rivoluzionarie", ma consolida la libreria come **legacy stable** per progetti che ancora lo usano.

Situazione attuale

- Con la diffusione dei **framework moderni (React, Angular, Vue)** e con le API standardizzate (ES6+, querySelector, fetch, addEventListener), jQuery non è più indispensabile.
- Rimane però **largamente usato** in progetti legacy, CMS (WordPress, Drupal), e in tanti siti già in produzione.

WRITE LESS, DO MORE

A differenza di altre librerie nate nello stesso periodo, **jQuery**, si è basato su un'idea precisa: scrivere codice breve e leggibile, riducendo al minimo le modifiche agli oggetti globali di JavaScript per mantenere la massima compatibilità con altre librerie.

Questo approccio ha dato vita a uno strumento versatile che permette di:

- manipolare facilmente elementi HTML e stili CSS,
- creare animazioni ed effetti grafici,
- gestire chiamate AJAX in modo semplice e compatibile con i vari browser,
  il tutto senza alterare direttamente gli oggetti nativi di JavaScript.

Il cuore di jQuery: l'oggetto $

Chi inizia a lavorare con jQuery deve sapere che quasi tutto ruota attorno alla funzione/oggetto **$**, che è semplicemente un alias di **jQuery**.

Esempio:

```
$("#block");     // Sintassi abbreviata

jQuery("#block"); // Equivalente, ma più esplicito
```

Per capire il vantaggio, confrontiamo la stessa operazione – ottenere l'attributo href di un link – scritta in modi diversi:

```
<a id="myLink" href="http://www.jquery.com ">link</a>

document.getElementById("myLink ").href;      // JavaScript puro

$('myLink').readAttribute('href');          // Prototype

$("# myLink").attr("href");                 // jQuery
```

Da subito emergono due caratteristiche fondamentali:

1. **Sintassi più breve e chiara**.
2. Uso dei **selettori CSS** dentro $() per individuare gli elementi (come accade in Mootools con $$).

Attenzione però: scrivere $('myLink') in jQuery non funziona, perché la libreria interpreterebbe mioLink come un tag inesistente, e non come un id. Il vero punto di forza, infatti, è il **motore di selezione interno**, veloce e potente.

Metodi concatenabili

Una delle peculiarità di jQuery è la possibilità di **concatenare metodi**:

```
$("#mioLink").text("Nuovo testo").css("color", "red");
```

In un'unica riga cambiamo sia il contenuto che lo stile del link, rendendo il codice più compatto e leggibile.

Perché usare jQuery

Oltre alla semplicità di scrittura, ci sono diversi motivi per cui jQuery si è imposto:

- **Compatibilità garantita**: grazie al metodo jQuery.noConflict(), è possibile evitare conflitti con altre librerie che usano il simbolo $. Ad esempio:
    - Mootools può usare $('myId')
    - jQuery rimane accessibile con jQuery("#myId").
- **Sistema di plugin**: la libreria può essere estesa facilmente con nuove funzioni sviluppate da terzi.
- **Community attiva**: attorno al progetto si è creata una vasta rete di sviluppatori che condividono plugin, guide e supporto.
- **Efficienza e sintesi**: il motto "*Write less, do more*" riflette perfettamente la filosofia di jQuery: meno codice, più funzionalità.

Come ottenere jQuery

Dal sito ufficiale (**jquery.com**) è possibile scaricare il framework in due versioni:

- **Development** (non compresso, utile per il debug – circa 120KB).
- **Production** (minificato, ottimizzato per l'uso online – circa 59KB, riducibili a ~19KB con GZip lato server).

Per includerlo in una pagina HTML basta aggiungere:

```
<script src="jquery-3.7.1.min.js"></script>
```

Un'alternativa molto diffusa è usare la versione ospitata sui **CDN di Google**:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
```

Questa scelta ha due vantaggi:

1. **Caricamento più rapido** se il file è già presente nella cache del browser dell'utente.
2. Riduzione del peso sul server del sito.

Lo svantaggio, però, è la dipendenza dalla disponibilità del servizio esterno.

L’oggetto $ come selettore

Il fulcro di jQuery è la funzione/oggetto **$**, che nella maggior parte dei casi viene utilizzata come un **selettore CSS**.

Esempio:

var elementi = $("#myId");

In questo caso, la variabile elementi rappresenta un oggetto jQuery che fa riferimento al nodo del DOM con id myId.

La stringa passata a $() segue esattamente le regole dei selettori CSS:

$(".miaClasse"); // Seleziona tutti gli elementi con class="miaClasse"

Come nei fogli di stile, si possono usare più selettori o combinazioni complesse:

$(".miaClasse, #myId, ul#mioMenu li");

// Cerca: elementi con classe "miaClasse", l’elemento con id "myId"

// e tutti i <li> dentro la lista con id "mioMenu"

Selettori con contesto

È anche possibile restringere la ricerca a un ambito specifico, aggiungendo un **secondo parametro** che rappresenta il contesto:

$(".miaClasse", document.getElementById("myId"));

// Tutti gli elementi con classe "miaClasse" contenuti dentro #myId

Questa caratteristica è molto utile quando si lavora su un insieme di elementi e si vuole iterare solo su quelli contenuti in una determinata sezione.

Uso diretto di oggetti DOM

La funzione $() accetta come argomento non solo stringhe, ma anche riferimenti diretti ad elementi del DOM:

$(document.getElementById("myId"));

$(window);

Creare nuovi elementi

Per mantenere la filosofia di sintesi, $() può essere usata anche per **creare nodi HTML** al volo.

Esempi:

$(" ");

$(" "); // sintassi abbreviata

È possibile creare strutture più complesse direttamente come stringa:

var blocco = $(" Esempio di elementi nidificati ");

Su questi nuovi elementi possiamo applicare metodi jQuery come su qualsiasi altro oggetto:

blocco.addClass("nuovaClasse"); // aggiunge una classe

E per inserirli nel documento:

blocco.appendTo(document.body); // dentro il body

blocco.appendTo("#contenitore"); // dentro un elemento con id contenitore

---

Funzioni al caricamento del DOM

Una particolarità di $() è che può accettare come parametro una **funzione**. In quel caso, il codice verrà eseguito automaticamente **quando il DOM è pronto** (evento *DOM ready*):

```
$(function () {
    alert("DOM caricato!");
});
```

Questo comportamento è diverso da window.onload:

```
window.onload = function () {
    alert("Caricato!");
};
```

- window.onload → attende il caricamento completo della pagina, comprese immagini e fogli di stile.
- $(document).ready() (o la forma abbreviata sopra) → scatta appena la struttura HTML è stata caricata, rendendo già disponibili gli elementi della pagina.

Il motore di selezione di jQuery

Uno dei punti di forza principali di jQuery è il suo **motore di selezione del DOM**: permette di individuare in modo rapido e preciso gli elementi della pagina, sui quali poi applicare i vari metodi della libreria.

Il funzionamento è semplice: basta indicare un selettore (lo stesso che useremmo nei CSS) e passarlo alla funzione $().

Esempio:

```
// Seleziono tutti gli <li> dentro il menu con id "menu"
$("ul#menu li");
```

Oltre ai selettori CSS3, jQuery introduce anche **filtri speciali** che rendono ancora più immediata la ricerca degli elementi.

Selettori di base

I selettori più comuni sono quelli che richiamano tag, classi e id:

```
$("a");          // tutti i link della pagina
$("#blocco");    // elemento con id="blocco"
$("a.external"); // link che hanno la classe "external"
```

Come nei fogli di stile, è possibile combinarne più di uno:

```
$("#blocco, a.external");
```

È persino possibile selezionare **tutti** gli elementi del documento:

```
$("*");
```

Nota: più specifico è il selettore, più veloce sarà l'elaborazione e minore l'impatto sulle prestazioni dello script.

Selettori gerarchici

jQuery supporta pienamente anche i selettori **basati sulla gerarchia** degli elementi HTML.

1. Spazio vuoto (Parent Child)

Seleziona tutti gli elementi contenuti all'interno di un altro.

```
// Tutti i link (<a>) dentro #blocco
$("#blocco a");
```

---

2. > (Parent > Child)

Limita la selezione solo ai **figli diretti**, escludendo i discendenti più interni.

```
$("ul > li"); // solo i <li> direttamente dentro <ul>
```

Esempio pratico:

```
<ul>
  <li>Selezionato</li>
  <li>
    <ol>
      <li>Non selezionato</li>
    </ol>
  </li>
</ul>
```

---

3. + (Prev + Next)

Seleziona l'elemento che si trova **immediatamente dopo** un altro.

```
$('div + span');
```

Esempio:

```
<div>Questo</div><span>✓ Sì</span>
<div>Questo</div><img /><span>✗ No</span>
```

Uso pratico:

```
$("label + input"); // input subito dopo un'etichetta
```

---

4. ~ (Prev ~ Next)

Seleziona **tutti i fratelli successivi**, non solo quello adiacente.

```
$('div ~ span');
```

Esempio:

* Questo ✓ Sì*
* Questo <img />✓ Anche*

Selettori basati sugli attributi

Un'altra caratteristica molto potente di jQuery è la possibilità di selezionare elementi in base ai **loro attributi**, utilizzando una sintassi identica a quella dei CSS.

Esempio semplice:

$("a[target='_self']"); // tutti i link che si aprono nella stessa finestra

In questo caso abbiamo filtrato i link che hanno target="_self".

Ricerca parziale nei valori

Non è necessario specificare l'intero valore di un attributo: possiamo cercare anche solo una parte di esso.

$("img[alt^='icone']"); // immagini con attributo alt che inizia con "icone"

$("p[class$='-testo']"); // paragrafi con classi che terminano con "-testo"

$("input[name='email']"); // input il cui name contiene "email"*

Esclusioni con attributi

Possiamo anche escludere determinati elementi specificando i valori da non considerare:

$("a[target!='_self']"); // tutti i link che NON si aprono nella stessa finestra

---

Selettori per presenza di attributi

Non serve sempre un valore: è sufficiente verificare la presenza di un attributo per intercettare un gruppo di elementi.

$("button[disabled]"); // tutti i pulsanti disabilitati

Questa tecnica è molto utile, ad esempio, per sostituire o arricchire funzioni native del browser con script personalizzati.

```
$("a[title]").each(function(){
   // applico un tooltip personalizzato
});
```

Combinazioni di selettori

Come per i selettori di base, anche quelli sugli attributi possono essere **concatenati** per affinare ulteriormente la ricerca:

```
$("a[target='_blank'][rel='noopener']");
// link che aprono una nuova finestra e hanno rel="noopener"
```

Perché tutto è un metodo in jQuery

In **JavaScript nativo**, quando vogliamo manipolare un elemento del DOM, dobbiamo prima **trovarlo** e poi applicare un'operazione. Per esempio:

document.getElementById("titolo").style.color = "red";

In **jQuery**, invece, ogni volta che usiamo $() otteniamo un **oggetto jQuery**, che rappresenta uno o più elementi del DOM. Su questo oggetto possiamo chiamare direttamente dei **metodi**.

Esempio equivalente in jQuery:

$("#titolo").css("color", "red");

Differenze chiave:

- In JavaScript puro otteniamo un **nodo DOM** e usiamo proprietà/metodi nativi.
- In jQuery otteniamo un **oggetto jQuery** che mette a disposizione una **serie di metodi già pronti**, più brevi e uniformi (compatibili cross-browser).
- I metodi jQuery sono spesso **concatenabili**:

  $("#titolo").css("color", "red").text("Nuovo Titolo");

METODI DI EVENTO IN JQUERY 3.7

I metodi di evento servono per collegare comportamenti a ciò che fa l'utente (click, tastiera, form, ecc.).

- click(): Si verifica quando un elemento viene cliccato.
- dblclick(): Si verifica quando un elemento viene cliccato due volte.
- mouseenter(): Si verifica quando il mouse entra nell'area di un elemento.
- mouseleave(): Si verifica quando il mouse esce dall'area di un elemento.
- mousedown(): Si verifica quando un pulsante del mouse viene premuto su un elemento.
- mouseup(): Si verifica quando un pulsante del mouse viene rilasciato su un elemento.
- hover(): Combina mouseenter() e mouseleave().
- focus(): Si verifica quando un elemento ottiene il focus.
- blur(): Si verifica quando un elemento perde il focus.
- keydown(): Si verifica quando un tasto viene premuto.
- keyup(): Si verifica quando un tasto viene rilasciato.
- submit(): Si verifica quando un form viene inviato.
- change(): Si verifica quando il valore di un elemento form cambia.

- **click()**
  - Esegue una funzione quando un elemento viene cliccato.

```
$("#btn").click(function() {
  alert("Hai cliccato il pulsante!");
});
```

- **dblclick()**
  - Attiva la funzione quando l'utente fa doppio clic.

```
$("#box").dblclick(function() {
  $(this).toggleClass("evidenziato");
});
```

- **mouseenter() / mouseleave()**
  - Scattano quando il cursore entra o esce dall'area di un elemento.

```
$("#box").mouseenter(function() {
  $(this).css("background", "lightblue");
}).mouseleave(function() {
  $(this).css("background", "white");
});
```

- **mousedown() / mouseup()**
  - Mousedown quando il mouse viene premuto, mouseup quando viene rilasciato.

```
$("#btn").mousedown(function() {
  $(this).text("Premuto...");
}).mouseup(function() {
  $(this).text("Rilasciato!");
});
```

- **hover()**
  - Combinazione di mouseenter e mouseleave.

```
$("#menu").hover(
  function() { $(this).css("color", "red"); },
  function() { $(this).css("color", "black"); }
);
```

- **focus() / blur()**
  - Focus si attiva quando un elemento (es. input) riceve il cursore, blur quando lo perde.

```
$("#nome").focus(function() {
  $(this).css("border", "2px solid green");
}).blur(function() {
  $(this).css("border", "1px solid gray");
});
```

- **keydown() / keyup()**
  - Intercettano la pressione (keydown) e il rilascio (keyup) di un tasto.

```
$(document).keydown(function(e) {
  console.log("Tasto premuto: " + e.key);
});
```

- **submit()**
  - Intercetta l'invio di un form.

```
$("#mioForm").submit(function(event) {
  event.preventDefault(); // blocca l'invio
  alert("Form inviato (ma non spedito davvero)!");
});
```

- **change()**
  - Si attiva quando cambia il valore di un input, select o textarea.

```
$("#scelta").change(function() {
  alert("Hai scelto: " + $(this).val());
});
```

METODI PRINCIPALI DI MANIPOLAZIONE IN JQUERY 3.7

- hide(): Nasconde l'elemento selezionato.
- show(): Mostra l'elemento selezionato.
- toggle(): Alterna tra nascondere e mostrare l'elemento selezionato.
- fadeIn(): Mostra l'elemento con un effetto di dissolvenza.
- fadeOut(): Nasconde l'elemento con un effetto di dissolvenza.
- slideUp(): Nasconde l'elemento con un effetto di scorrimento verso l'alto.
- slideDown(): Mostra l'elemento con un effetto di scorrimento verso il basso.
- animate(): Crea animazioni personalizzate.
- addClass(): Aggiunge una classe CSS all'elemento selezionato.
- removeClass(): Rimuove una classe CSS dall'elemento selezionato.
- toggleClass(): Alterna una classe CSS sull'elemento selezionato.
- attr(): Ottiene o imposta un attributo dell'elemento selezionato.
- html(): Ottiene o imposta il contenuto HTML dell'elemento selezionato.
- text(): Ottiene o imposta il contenuto testuale dell'elemento selezionato.
- val(): Ottiene o imposta il valore di un elemento form.
- append(): Aggiunge contenuto alla fine dell'elemento selezionato.
- prepend(): Aggiunge contenuto all'inizio dell'elemento selezionato.
- after(): Aggiunge contenuto dopo l'elemento selezionato.
- before(): Aggiunge contenuto prima dell'elemento selezionato.
- remove(): Rimuove l'elemento selezionato.
- empty(): Rimuove tutti i figli dell'elemento selezionato.
- css(): Ottiene o imposta una proprietà CSS dell'elemento selezionato.

- **hide()**
  - Nasconde gli elementi selezionati, impostando display: none.
  - Parametri:
    - **duration *(opzionale)*** → velocità dell'animazione ("slow", "fast", o millisecondi).
    - **easing *(opzionale)*** → tipo di transizione (es. "swing", "linear").
    - **callback *(opzionale)*** → funzione da eseguire dopo la fine dell'animazione.

```
$("#box").hide(); // Nasconde subito l'elemento
$("#box").hide(1000, function() { // Nasconde in 1 secondo, poi mostra un alert
  alert("Elemento nascosto!");
});
```

- **show()**
  - Mostra elementi precedentemente nascosti (con hide() o display:none).
  - Parametri: gli stessi di hide().

```
$("#box").show();// Mostra subito
$("#box").show("slow"); // Mostra con animazione lenta
```

- **toggle()**
  - Alterna la visibilità → se è visibile lo nasconde, se è nascosto lo mostra.
  - Parametri: identici a hide() e show().
    - Se si passa una duration, l'alternanza avverrà con animazione.

```
$("#box").toggle(); // Alterna subito
$("#box").toggle(500, function() { // Alterna con effetto e callback
  console.log("Stato cambiato!");
});
```

EFFETTI DI DISSOLVENZA

- **fadeIn()**
  - Mostra un elemento nascosto variando gradualmente l'opacità da 0 a 1.
  - Parametri:
    - **duration** *(opzionale)* → velocità ("slow", "fast", o millisecondi).
    - **easing** *(opzionale)* → tipo di transizione ("swing", "linear" o plugin extra).
    - **callback** *(opzionale)* → funzione eseguita al termine dell'animazione.

```
$("#box").fadeIn("slow"); // Mostra con dissolvenza lenta
$("#box").fadeIn(2000, function() { // Mostra in 2 secondi, poi cambia colore al testo
  $(this).css("color", "blue");
});
```

- **fadeOut()**
  - Nasconde un elemento riducendo l'opacità da 1 a 0.
  - Parametri: uguali a fadeIn().

```
$("#box").fadeOut(); // Nasconde gradualmente
$("#box").fadeOut(1000, function() { // Nasconde in 1 secondo e poi rimuove del tutto l'elemento dal DOM
  $(this).remove();
});
```

EFFETTI DI SCORRIMENTO VERTICALE

- **slideDown()**
  - Mostra un elemento nascosto facendo scorrere il contenuto verso il basso.
  - Parametri:
    - **duration, easing, callback** (come per fadeIn).

```
$("#menu").slideDown(600); // Espande il menu in 600 ms
$("#menu").slideDown("slow", function() { // Espande e logga un messaggio
  console.log("Menu aperto!");
});
```

- **slideUp()**
  - Nasconde un elemento facendo scorrere il contenuto verso l'alto.
  - Parametri: identici a slideDown().

```
$("#menu").slideUp(); // Nasconde il menu
$("#menu").slideUp(1000, function() { // Nasconde in 1 secondo e aggiunge una classe
  $(this).addClass("chiuso");
});
```

- **animate()**
  - Crea animazioni personalizzate sulle proprietà CSS numeriche.

```
$("#box").animate({ width: "300px", opacity: 0.5 }, 1000);
```

- **addClass() / removeClass() / toggleClass()**
  - Gestiscono le classi CSS di un elemento.

```
$("#box").toggleClass("attivo");
```

- **attr()**
  - Ottiene o modifica un attributo.

```
$("#link").attr("href", "https://openai.com");
```

- **html()**
  - **Senza parametri** → legge il contenuto HTML del primo elemento selezionato.
  - **Con un parametro** → sostituisce il contenuto con quello specificato (accetta stringhe HTML).

let contenuto = $("#box").html(); // Legge l'HTML dentro #box

console.log(contenuto); // es. " Testo con stile "

$("#box").html("Nuovo contenuto"); // Sostituisce il contenuto con nuovo HTML

- **text()**
  - **Senza parametri** → legge solo il testo (senza interpretare tag HTML).
  - **Con un parametro** → sostituisce il contenuto testuale (il testo viene inserito così com'è, senza trasformarsi in HTML).

let soloTesto = $("#box").text(); // Legge solo il testo (senza markup)

console.log(soloTesto); // es. "Testo con stile"

$("#box").text("Nuovo testo"); // Sostituisce il contenuto con puro testo

// Risultato visibile: Nuovo testo (come testo normale, non HTML)

Differenza chiave html() e text()

html() ***Mantiene e interpreta HTML***

text() ***Restituisce o inserisce solo testo, ignorando i tag***

- **val()**
  - Ottiene o imposta il valore di un input o select.

  $("#nome").val("Mario Rossi");

- **append() / prepend()**
  - Aggiungono contenuto rispettivamente in coda o in testa all'interno di un elemento.

  *$("#lista").append("
Nuovo elemento
");*

- **after() / before()**
  - Inseriscono contenuto immediatamente dopo o prima dell'elemento selezionato.

  $("#titolo").before("<h2>Sottotitolo</h2>");

- **remove()**
  - Elimina l’elemento stesso dal DOM insieme a tutto il suo contenuto.
  - Parametri: Può accettare un selettore per rimuovere solo gli elementi che lo rispettano.

$("#box").remove(); // Rimuove completamente #box dal DOM

$("div").remove(".attivo"); // Rimuove solo i div con classe "attivo"

- **empty()**
  - Elimina solo i contenuti interni (figli) dell’elemento selezionato, lasciando però l’elemento contenitore.
  - Parametri: nessuno.

$("#box").empty(); // Svuota il contenuto di #box, ma il div resta

// Dopo questo esempio:

*// *

Differenza chiave remove() e empty()

remove() **Rimuove completamente l’elemento dal DOM**

empty() **Mantiene l’elemento, ma lo svuota del suo contenuto**

- **css()**

Il metodo .css() serve per ottenere o impostare proprietà CSS sugli elementi selezionati. È molto versatile perché accetta diversi tipi di parametri.

---

1. Leggere una singola proprietà CSS

Se usato con un solo argomento stringa, restituisce il valore della proprietà del primo elemento selezionato.

```
let colore = $("#box").css("color"); // Legge il colore del testo dentro #box

console.log(colore); // es. "rgb(0, 0, 0)"
```

---

2. Impostare una singola proprietà

Se passiamo due argomenti (nomeProprietà, valore), aggiorna la proprietà CSS di tutti gli elementi selezionati.

```
$("#box").css("color", "red"); // Cambia il colore del testo

$("#box").css("height", "200px"); // Imposta l'altezza a 200px
```

---

3. Impostare più proprietà insieme

Possiamo passare un oggetto {chiave: valore} con più proprietà.

```
// Applica più stili contemporaneamente

$("#box").css({

  "color": "white",

  "background-color": "black",

  "padding": "10px"

});
```

---

4. Usare una funzione per calcolare il valore

Se il valore è una funzione, riceve come parametri l'indice dell'elemento e il valore corrente, così possiamo calcolare dinamicamente.

```
// Incrementa la larghezza di ogni div del 50%

$("div").css("width", function(index, value) {

  return (parseInt(value) + 50) + "px";

});
```

RIEPILOGO CSS()

- *.css("prop") → legge una proprietà*
- *.css("prop", "valore") → imposta una proprietà*
- *.css({prop: valore, ...}) → imposta più proprietà insieme*
- *.css("prop", funzione) → calcola dinamicamente il valore*