Università di Bologna
Scuola di Ingegneria

Tecnologie Web T
A.A. 2023 – 2024

Esercitazione 04
Eventi, DOM e Javascript

Su Virtuale:
Versione 1 pagina per foglio = L.04.Javascript.pdf
Versione 2 pagine per foglio = L.04.Javascript-2p.pdf

Agenda

- Alcune premesse “pratiche”
  - progetto d'esempio
  - risorse statiche e cache dei browser
- Javascript e le applicazioni Web
  - semplici esempi di utilizzi ricorrenti
  - alcuni esercizi da svolgere in autonomia
  - accesso alle stesse risorse tramite Web Server
  - esempi di utilizzo delle API JQuery come alternativa al linguaggio nativo JavaScript
  - esercizio guidato tratto dall’appello d’esame del 26 gennaio 2018: HTML, CSS e Javascript

Progetto di esempio

- All'interno del file ***04_TecWeb.zip*** trovate lo scheletro di un semplice progetto di esempio
  - creato con Eclipse, contiene già tutti i descrittori necessari a essere riconosciuto e configurato correttamente
- Non importare il progetto in Eclipse, ma scompattare il file zip direttamente su filesystem
  - nella directory *web/pages* le pagine html con codice Javascript da testare
  - nella directory *web/scripts* un po' di codice Javascript

Prima di iniziare

- I browser "fanno cache" del contenuto statico
  - pagine html, script js, fogli di stile css, immagini gif/png/jpeg/jpg/...
  - infatti, normalmente non accade che tali risorse cambino tra due invocazioni successive
- Quando si sviluppa una applicazione, questo comportamento è invece fastidioso e bisogna cercare di ovviare
  - aggiungere i seguenti meta attributi nell'header delle pagine che si stanno modificando:

    ```
    <meta http-equiv="Pragma" content="no-cache"/>
    <meta http-equiv="Expires" content="-1"/>
    ```
  - tenersi sempre pronti a svuotare la cache del browser quando non si osservano gli effetti o le modifiche sperati (anche se si usano i meta attributi)
  - impostare (se possibile) il browser affinché NON usi la versione in cache delle risorse già scaricate

Definizione e invocazione di funzioni - *helloworld.html*

- Invocazione di funzioni man mano che il browser processa, effettua il rendering e interpreta il sorgente della pagina
  - le funzioni invocate, `alert()` e `confirm()`, sono predefinite
  - la loro definizione e implementazione è fornita dall'interprete Javascript presente nel browser
- La definizione di una funzione non ne comporta l'immediata esecuzione, ma la rende disponibile nel proseguimento della pagina, ai fini dell'invocazione:
  - dall'interno di uno script *in linea* nella pagina
  - dall'interno di altro codice Javascript
  - associata ad eventi DHTML
- in evidenza:
  - i messaggi popup interrompono il rendering
    - diversi browser, tuttavia, tentano di schedulare il rendering e l'esecuzione di codice Javascript come processi paralleli, quando possibile
    - provate (a casa su) Firefox per farvi un'idea delle differenze di comportamento che emergono
  - l'uso dell'alternanza tra apice singolo ' e virgolette “ permette di scrivere HTML ben formato anche all'interno del codice Javascript, senza utilizzare escaping
    - apice singolo e virgolette sono entrambi delimitatori di stringa validi in Javascript e **all'interno di una coppia di delimitatori di un tipo si possono usare quelli dell'altro tipo** senza bisogno di escaping

Interazione tra Javascript e il DOM - *javascript-dom.html*

- L'esecuzione di una funzione che interagisce con il contenuto visualizzato deve...

attendere il completamento

del caricamento del contenuto

...da parte del browser

- Sembra banale, eppure...

Reazione ad eventi DHTML - *javascript-events.html*

- Nella pagina *javascript-events.html*
  - l'evento ***onload*** è sfruttato per agganciare funzioni Javascript a parti del DOM, in corrispondenza degli eventi DHTML da essi supportati
  - un'altra funzione (definita nella pagina stessa) viene associata ad un evento DHTML direttamente nel tag interessato

- In evidenza
  - all'interno del codice di una funzione Javascript è possibile definire nuove funzioni ed assegnarle

In Javascript, infatti, le funzioni sono oggetti di prima classe

  - non c'è alcuna differenza in Javascript tra un numero, una stringa e una funzione
  - tutti questi tipi di dati possono essere passati come argomento, memorizzati in variabili e restituiti come valore di ritorno di una funzione

Primi problemi (1) - *rollover.html*

- L'effetto di evidenziazione al passaggio del mouse non sempre è ottenuto modificando il colore di sfondo e/o del font
  - spesso si usano piccole immagini, sostituite tra loro in corrispondenza degli eventi relativi al mouse
  - l'effetto prende il nome di 'roll over'

- Nella pagina rollover.html sono presenti 4 modi diversi di ottenere lo stesso effetto
  - tramite CSS e pseudoclassi
  - tramite DOM e DHTML
  - tramite funzioni Javascript

- ma.... l'ultima funzione non va!
  - a differenza del codice JSP (compilato), quello Javascript è interpretato e la presenza di errori si scopre solo al momento di eseguire il codice stesso
  - infatti, non ci siamo accorti del problema finché non abbiamo eseguito (o, almeno, provato ad eseguire) la funzione incriminata
  - inoltre, il Servlet Container fallisce la traduzione e compilazione in servlet e ci restituisce un errore (500) e il relativo stackTrace dove guardare: il browser, invece, prosegue “a testa bassa” se qualcosa non va e semplicemente "inibisce" le parti di codice non corrette

Primi problemi (2) - *rollover.html*

- Sì, ma... e adesso? un indizio sul problema?
  - ore e ore a spulciare il codice
  - uso di milioni di `alert()` per mostrare popup in cui effettuare il log dello stato di variabili e oggetti per capire dove nasce il problema
  - usare un tool di sviluppo più avanzato!
- È il momento di Chrome Developer Tools (…o Firefox for Developer):
  - logging dei messaggi di errore Javascript su una specie di console di stderr
  - esecuzione del codice Javascript in modalità debug
  - ...e molto di più
- Ctrl + Shift + I per attivare Chrome Developer Tools
  - interagire con la pagina per lanciare l'esecuzione della funzione sbagliata e leggere i messaggi di errore nella console
  - cosa correggere?
  - una volta che la funzione esegue, collocare breakpoint al suo interno e seguirne l'esecuzione in modalità debug
    - Per collocare un breakpoint nello script Javascript è necessario accedere al codice tramite la scheda “Sources” di Chrome Developer Tools

Validazione di form lato client - *validate.html*

- Tipico utilizzo di Javascript per evitare di...
  - formulare HTTP request (magari anche pesanti) destinate a fallire
  - evitare di consumare banda
  - scrivere pagine più “responsive” e migliorare l'esperienza dell'utente
- Accesso agli elementi di un form via DOM e verifica di correttezza: seguiamo insieme l'esecuzione in modalità debug
  - apertura di Chrome Developer Tools, tab “Sources”, selezione del sorgente Javascript che interessa
  - collocazione dei breakpoint
  - interazione con la pagina per scatenare l'invocazione del codice Javascript (click sul pulsante, nel nostro caso)
- In evidenza
  - "esternalizzazione" delle funzioni Javascript in appositi file ***.js***, richiamabili da più pagine
  - è possibile (e anzi è prassi comune) utilizzare i valori booleani restituiti dalle funzioni Javascript per bloccare o permettere l'effettiva submission di un form

Modifica della visibilità del contenuto - *hideAndSeek.html*

Un tipico effetto grafico per ottenere contenuto dinamico all'interno di una pagina

- tutto il contenuto informativo è scaricato all'atto della richiesta, ma solo una parte è immediatamente visibile
- attraverso gli eventi DHTML scatenati dalle azioni dell'utente si aggancia l'esecuzione di codice Javascript che modifica gli attributi di stile degli elementi interessati
  - *display: block;* → l'elemento è mostrato E occupa spazio nella pagina
  - *display: none;* → l'elemento NON è mostrato E NON occupa spazio
- intervenendo sull'attributo
  - *visibility: hidden;*

  si sarebbe invece ottenuto l'effetto di nascondere l'elemento, ma lasciare vuoto lo spazio che esso avrebbe occupato

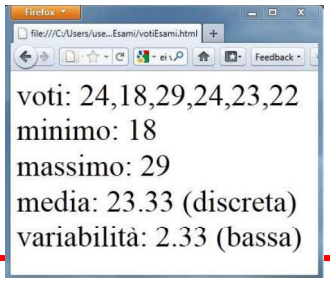
Stringhe ed interi - *calculator.html*

- Simile all'esempio presente nelle slide di teoria, serve qui a imprimere il concetto che **il browser e l'interprete Javascript che ne legge il DOM ragionano entrambi a stringhe**
  - non è specificato che un campo di input testuale sia destinato all'immissione di numeri
  - se non viene forzato a “valutare” l'espressione rappresentata dall'input testuale, l'interprete Javascript ne considera il valore come stringa
- Come sistemare?
  - sbirciate nel codice...
  - ovviamente non aprendo i file, ma con Chrome Developer Tools

Calcolo media e variabilità voto esami

Realizzare una pagina HTML dinamica basata su Javascript che, data una sequenza di voti di lunghezza non nota a priori, ne restituisca la media e la variabilità

- Ciascun voto deve essere un numero compreso tra 18 e 33: controllare opportunamente i dati in input
- La pagina restituita deve specificare
  - elenco dei voti inseriti
  - voto minimo e voto massimo
  - media aritmetica e giudizio sintetico (6 fasce tra 18 e 33)
  - variabilità come "errore medio" e giudizio sintetico (4 fasce tra 0 e 7.5)
- N.B. La soluzione deve basarsi sulla definizione di un oggetto *Statistica* contenente proprietà e metodi necessari alla computazione richiesta

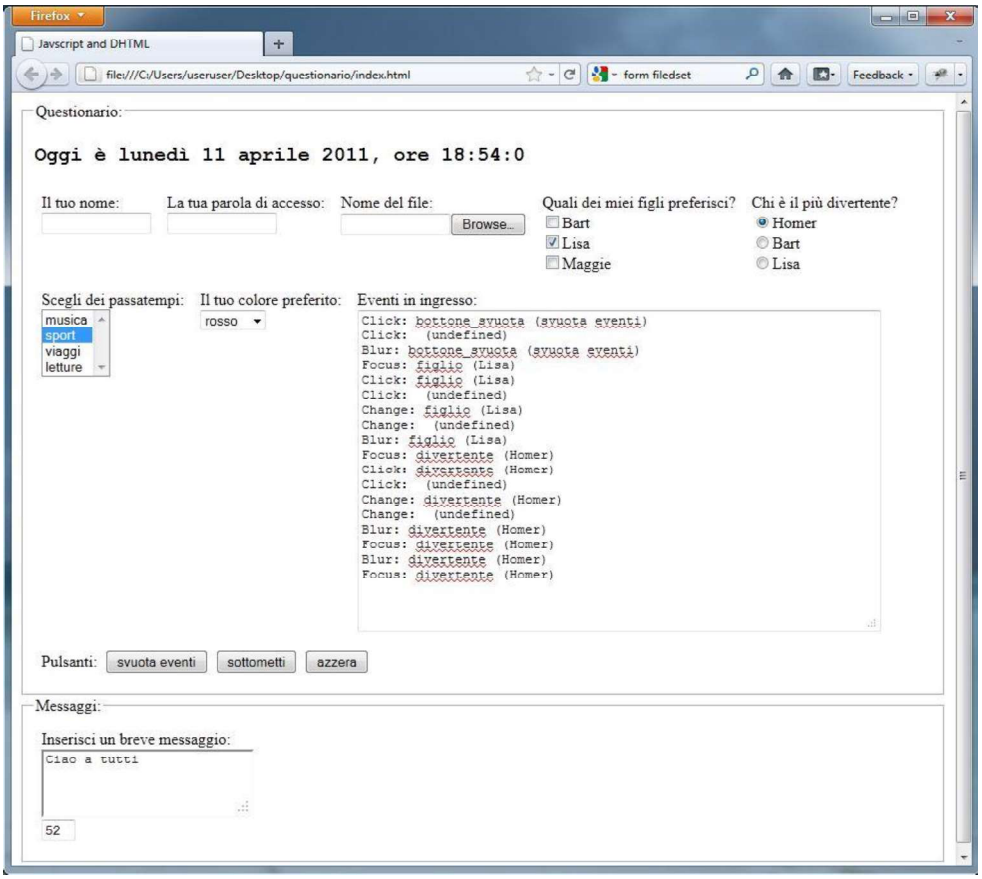


Gestione dinamica componenti HTML

Realizzare una pagina HTML + Javascript che presenti

- In alto **l'orario** che si aggiorna una volta al secondo
  - *onLoad()* per invocare ogni 1000ms una funzione di aggiornamento
- In mezzo un **form** coi seguenti elementi HTML: input di testo, password, file, checkbox, radio, tendina scelta multipla, tendina scelta singola, button/submit/reset
  - realizzare una funzione Javascript che visualizzi in un'area di testo informazioni relative agli eventi associati ai precedenti elementi
    1) recuperare **dinamicamente** l'elenco degli elementi del form
    2) per ciascun elemento registrare una funzione agli eventi di tipo *onclick*, *onchange*, *onfocus*, *onblur*, *onselect* e *ondblclick*
- In basso un'**area di testo** con al più 64 caratteri
  - il colore dell'area cambia quando il mouse ci passa sopra (*onmouseover*/*onmouseout*)
  - un altro campo di testo specifica i caratteri ancora a disposizione (*onKeyUp*)
  - se si supera il limite, i caratteri in eccesso vengono eliminati

Gestione dinamica componenti HTML: esempio



Deployment su Tomcat

- Come dicevamo... all'interno del file ***04_TecWeb.zip*** trovate lo scheletro di un semplice progetto di esempio
  - creato con Eclipse, contiene già tutti i descrittori necessari a essere riconosciuto e configurato correttamente
- Ora importare il progetto come visto nelle precedenti esercitazioni
  - *File → Import → General → Existing Projects into Workspace → Next → Select archive file*
- Eseguirne il deployment su Tomcat ed accedere all'applicazione
  - *http:/localhost:8080/04_TecWeb/*

Pagina di benvenuto – *welcome.jsp*

- Struttura a *frame*...
  - giusto per sapere che esistono: tutti i maggiori browser li supportano, **ma il loro uso è deprecato**
  - non così l'uso di ***iframe***, invece, data la necessità di poter eseguire richieste cross-domain

– Il codice di *welcome.jsp* comporta la visualizzazione di alcuni fragment ed un menù con le pagine viste fin ora

Differenze con gli esempi precedenti

- Ora lavoriamo sul progetto d'esempio utilizzando il **browser**, il **Web Server** e il protocollo **HTTP**

- Necessario in quanto
  - ...alcune pagine del progetto (i menu, la pagina con i frameset, ...) sono basate su tecnologia JSP e richiedono un Servlet Container per funzionare
  - ...ma **le pagine oggetto degli esempi Javascript** (cioè quelle all'interno della directory *pages*) **richiedono soltanto la presenza dell'interprete Javascript** per poter essere visualizzate
    - sono infatti risorse statiche con estensione **.html**
    - richiamano altre risorse statiche (immagini **.gif/.png**, script **.js**, fogli di stile **.css**) attraverso path relativi da cui il browser ottiene URL completi basandosi sull'URL della pagina visualizzata
    - **ecco perché potete aprire ciascuna di esse direttamente nel browser**, utilizzandone l'URL su file system (file://...), senza lanciare il Web Server e accedere via protocollo http (http://...)

APPENDICE

(esempi guidati sull'uso delle API JQuery ☺)

JQuery in a nutshell

- Breve introduzione a JQuery
  - JQuery vs. JavaScript
  - JQuery: Oggetti
  - JQuery: Selettori
  - JQuery: Css, Eventi e modifica del DOM

- Esempi d'uso delle API JQuery
  - Color
  - TODO List
  - Events Delegation

JavaScript vs. JQuery

- …come già sappiamo dalla teoria, JQuery è:
  - una libreria JavaScript (API)
  - essendo scritto in JavaScript tutto quello che permette di fare JQuery è possibile farlo anche solo con JavaScript
  - favorisce la compatibilità cross-browser che spesso JavaScript fatica a trovare
  - permette la manipolazione di CSS, del DOM, l'aggiunta dinamica di eventi e tante altre cose…

- JQuery è finalizzato a:
  - Rendere più semplice lo sviluppo di interfacce grafiche
  - Sviluppo semplice di applicativi lato Client
  - Aggiungere con minimo sforzo effetti altrimenti complicati da ottenere
  - Snellire il codice JavaScript e renderlo più sintetico

Per un pugno di $

- In JQuery tutto ruota attorno all'oggetto/funzione **$**, abbreviazione o alias di ***JQuery***
  - $ è l'operatore principe di selezione, in particolare si fa quasi tutto attraverso i parametri della funzione **$()**
  - La caratteristica più utile ed apprezzata di JQuery è il suo motore di selezione
    - *$('nome_tag') restituisce un oggetto JQuery contenente tutti i tag di tipo nome_tag*
    - *$('.nome_class') restituisce un oggetto JQuery contenente tutti gli oggetti del DOM appartenenti alla classe nome_class*
    - *$('#nome_id') restituisce un oggetto JQuery contenente tutti gli oggetti del DOM con id nome_id*
    - *$('[nome_attr=val_attr]') restituisce un oggetto JQuery contenente tutti gli oggetti del DOM aventi l'attributo nome_attr che vale val_attr*
  - Come vedremo negli esempi, ci sono molti tipi di selettori, per gerarchie, componibili e combinabili, per posizione, ecc.. mentre in JavaScript solo document.getElementById, document.getElementByTagName, document.querySelector (recente)

JQuery: selettori (i)

Selettori principali offerti da JQuery

Selector	Example	Selects
*	$("*")	All elements
#id	$("#lastname")	The element with id="lastname"
.class	$(".intro")	All elements with class="intro"
.class,.class	$(".intro,.demo")	All elements with the class "intro" or "demo"
element	$("p")	All <p> elements
el1,el2,el3	$("h1,div,p")	All <h1>, and <p> elements

:first	$("p:first")	The first <p> element
:last	$("p:last")	The last <p> element
:even	$("tr:even")	All even <tr> elements
:odd	$("tr:odd")	All odd <tr> elements

JQuery: selettori (ii)

[attribute]	$("[href]")	All elements with a href attribute
[attribute=value]	$("[href='default.htm']")	All elements with a href attribute value equal to "default.htm"
[attribute!=value]	$("[href!='default.htm']")	All elements with a href attribute value not equal to "default.htm"
[attribute$=value]	$("[href$='.jpg']")	All elements with a href attribute value ending with ".jpg"
[attribute|=value]	$("[title|='Tomorrow']")	All elements with a title attribute value equal to 'Tomorrow', or starting with 'Tomorrow' followed by a hyphen
[attribute^=value]	$("[title^='Tom']")	All elements with a title attribute value starting with "Tom"
[attribute~=value]	$("[title~='hello']")	All elements with a title attribute value containing the specific word "hello"
[attribute*=value]	$("[title*='hello']")	All elements with a title attribute value containing the word "hello"

parent > child	$("div > p")	All <p> elements that are a direct child of a element
parent descendant	$("div p")	All <p> elements that are descendants of a element
element + next	$("div + p")	The <p> element that are next to each elements
element ~ siblings	$("div ~ p")	All <p> elements that are siblings of a element
:eq(index)	$("ul li:eq(3)")	The fourth element in a list (index starts at 0)
:gt(no)	$("ul li:gt(3)")	List elements with an index greater than 3
:lt(no)	$("ul li:lt(3)")	List elements with an index less than 3
:not(selector)	$("input:not(:empty)")	All input elements that are not empty

JQuery: metodi … alcuni

- Essendo una libreria, JQuery offre metodi e proprietà
  - $(#menu li).size();
  - $(#menu li).length; (sintassi array, più veloce)
- Per ottenere elementi da una lista:
  - $("#menu li").get(0);
  - In JavaScript -> document.getElementById("menu").getElementsByTagName("li");
- Per ottenere html:
  - $("#menu li").get(0).innerHTML; // con JavaScript nativo
  - $("#menu li").eq(0).html(); // con metodo jQuery
- Per lavorare con gli attributi:
  - $("a#mioLink").attr("href"); restituisce il valore di href
  - $("a#mioLink").attr("href","http://www.html.it"); imposta il valore di href
  - $("a#mioLink").attr("href",function () { ... }); imposta il valore di href in base alla funzione
  - $("#menu li a").removeAttr("target"); rimuove un attributo
- Per lavorare con testi e html
  - $("p").text("Nuovo testo");
  - $("p").html("Nuovo testo con HTML");

JQuery: impostare CSS

- Con JQuery, una volta ottenuti degli oggetti, è possibile impostare il loro css, anche qui con la stessa semantica di getter e setter:
  - $("a").css("color"); restituisce il colore esadecimale del primo elemento link
  - $("a").css("color","#FF0000"); //imposta il colore dei link
  - $("a").css({
    "color" : "#FF0000", //imposta il colore
    "display" : "block" // imposta la visualizzazione
    });
- Impostare il colore di sfondo:
  - $("a").css("background-color","#FF0000");
  - $("a").css("backgroundColor","#FF0000");

Metodi	Descrizione
.width() .height()	permette di trovare o impostare larghezza e altezza complessiva di un elemento in pixel
.innerWidth() .innerHeight()	larghezza e altezza interne di un elemento (include il padding, non conta bordi e margini)
.outerWidth() .outerHeight()	larghezza e altezza esterne di un elemento (conta bordi e padding, opzionalmente i margini passando l'argomento `true`)
.offset()	ritorna un oggetto con la distanza da sinistra e dall'alto dell'emento rispetto al documento
.position()	come `.offset()` ma le distanze sono relative al contenitore più prossimo
.scrollTop() .scrollLeft()	trova o imposta scroll dell'elemento rispetto al documento

JQuery: binding e gestione eventi

JQuery permette ***associazione dinamica di eventi agli oggetti selezionati***

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

- Ci sono due modi per fare il binding di eventi:
  - $("p").click(function(){// action goes here!!});
  - $("p").on("evento", (function(){// action goes here!!});
- Esempio di *multiple event assignment*

```
$("p").on({
    mouseenter: function(){
        $(this).css("background-color", "lightgray");
    },
    mouseleave: function(){
        $(this).css("background-color", "lightblue");
    },
    click: function(){
        $(this).css("background-color", "yellow");
    }
});
```

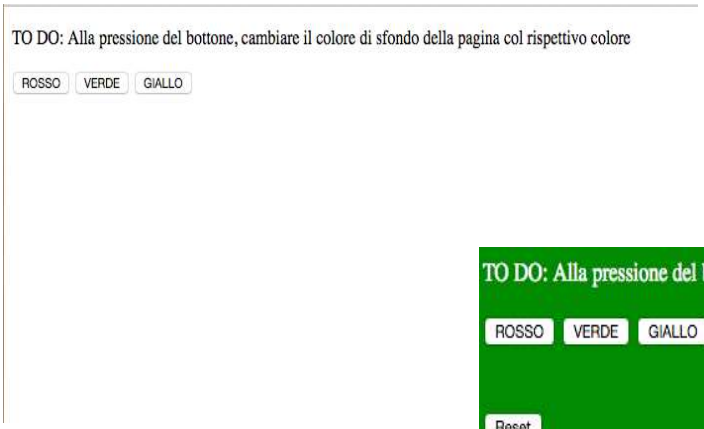

JQuery: modifica del DOM ed Event Delegation

- JQuery permette di modificare il DOM
  - $('body').append('<p ">Testo a casa '); fa l'append del tag html al body
  - Esiste anche prepend che inserisce il contenuto prima dell'oggetto selezionato
- Altri metodi per la modifica del DOM
  - $(" Nuovo paragrafo "); *crea un nuovo elemento nell'oggetto jquery*
  - $("p").html("Testo del paragrafo"); fa la stessa cosa
  - $("
lista
").appendTo("#menu"); inserisco un nuovo nodo nell'elemento selezionato
  - ce ne sono molti altri…
- JQuery permette di fare una cosa molto comoda quando si aggiungono dinamicamente gli oggetti al DOM, ovvero la «delegazione di eventi»
- Si assegna un evento ad un nodo padre, ma lo si scatena solo quando l'utente interagisce con un nodo figlio. Esempio:
  - $('body').on('click', 'button', function(e){}); assegno la funzione di gestione dell'evento click al body, ma la scateno quando l'utente clicca bottoni contenuti nel body

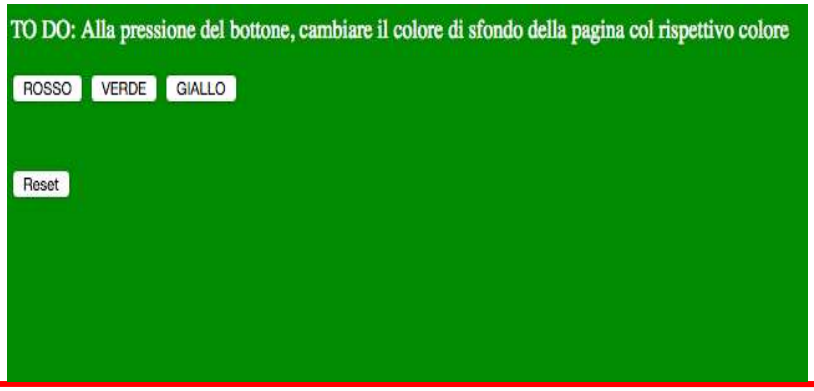
```
$('body').on('click', '[mio_attr="attr_val"]:first-of-type', function(e){
    // fai qualcosa
});
```

Esempio 1: «Colorize me»

- Il file color.html contiene un paragrafo con le istruzioni da implementare e 3 bottoni
- Utilizzando JQuery, alla pressione di uno di questi bottoni colorare lo sfondo del Body con il colore indicato da questi.

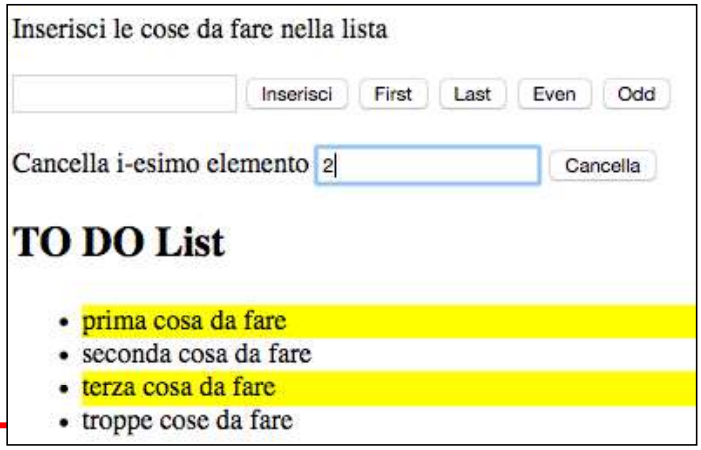


- Il risultato…



Esempio 2: «TODO list»

- File todo.html: creare una TODO list dinamica
- Attraverso JQuery inserire dinamicamente nella lista sottostante le cose da fare
- Aggiungere dei bottoni per la selezione:
  - del primo elemento della lista
  - degli elementi pari
  - degli elementi dispari
  - dell'ultimo elemento
- Infine inserire nel secondo campo di testo l'indice dell'elemento da eliminare ed eliminarlo
- Il risultato…



Esempio 3: «Randomize Color»

- File index.html: inserire dinamicamente alla pressione del bottone un paragrafo con scritto ‘Click’
- Al click si uno di questi paragrafi generare un colore casuale ed impostarlo come colore di sfondo
- Per generare un colore casuale
  - '#'+Math.floor(Math.random()*16777215).toString(16);

- ! ATTENZIONE: «Event Delegation»
- Il risultato…