Programmazione per il Web

Sistemi Distribuiti, Parte 3b

Corso di Laurea in Ingegneria o per altri CDL

Claudio Badii, Michela Paolucci

Department of Information Engineering, DINFO

University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570

Lab: DISIT, Sistemi Distribuiti e Tecnologie Internet

http://www.disit.dinfo.unifi.it/

Paolo.nesi@unifi.it
http://www.disit.org/nesi

Sistemi Distribuiti

Corso di Laurea in Ingegneria

Programmare per il Web

1. Parte I: Espressioni Regolari
2. Parte II: Javascript / Approfondimenti

Espressioni regolari (1)

- Servono a definire la sintassi di sequenze di caratteri
- Si usano per trovare in una stringa la/le parti che hanno una sintassi particolare o per controllare se un testo ha la sintassi voluta
- Es:
  - Trovare in un testo tutte le email
  - Controllare se è stata inserita una email potenzialmente valida
  - Trovare in una pagina html tutti i link esterni (href="…")

Espressioni Regolari (2)

- Una espressione regolare ha una specifica sintassi:
  - '/***pattern***/', dove ***pattern*** contiene una sequenza di caratteri, alcuni di questi possono essere caratteri speciali ([], \, ., *, ?, +, ^, $, {})
  - Es: /abc/ indentifica i caratteri a,b e c in sequenza.
  - La / che indica inizio e fine del pattern può essere sostituita da un qualsiasi altro carattere (es. @ o |) basta che siano uguali, utile quando il carattere / fa parte del testo
- Semantica dei caratteri speciali:
  - […] e [^…]
    - Indica il possibile valore di un carattere in un elenco (o non nell'elenco con [^…])
      - ***[abc:]*** indica il carattere a o b o c o :
      - ***[a-z]*** indica una qualsiasi lettera minuscola
      - ***[0-9]*** indica una qualsiasi cifra
      - ***[a-zA-Z0-9]*** indica una qualsiasi lettera o cifra
      - ***[^0-9]*** indica un carattere che non è una cifra
      - Es: '/0[1-9]0/' identifica le sequenze di tre cifre fatte da 0, una cifra non 0 e poi 0.

Espressioni Regolari (3)

- Semantica caratteri speciali:
  - “.”
    - indica un qualsiasi carattere
      - Es: /a.c/ indica tre caratteri in cui la prima è a e ultima è c
  - “***e****”
    - Indica 0 o più occorrenze di ***e*** dove ***e*** è una espressione regolare o una lettera
      - ***ab**** indica il carattere a seguito da 0 o più b
      - ***x[a-z]**** indica il carattere x seguito da 0 o più lettere minuscole
      - ***[1-9][0-9]**** indica una cifra non 0 seguita da 0 o più cifre
      - ***.**** indica qualsiasi sequenza di caratteri
      - ***a *x*** indica la lettera a seguita da 0 o più spazi seguiti da una x.

Espressioni Regolari (4)

- Semantica caratteri speciali:
  - "***e***+"
    - 1 o piu' occorrenze di ***e*** dove ***e*** e' una espressione regolare o una lettera
      - ***ab+*** indica sequenze tipo ab, abb, abbb, …
  - "***e***{***n***,***m***}"
    - tra ***n*** e ***m*** occorrenze di ***e***
      - ***[a-z]{2,4}*** indica sequenze di 2, 3 o 4 lettere minuscole
  - "***e***{***n***,}"
    - almeno ***n*** occorrenze di ***e***
      - ***[a-z]{3,}*** indica sequenze di almeno 3 lettere minuscole
  - "***e***{,***m***}"
    - Al più ***m*** occorrenze di ***e***
      - ***x[a-z]{,3}*** indica x seguito da al massimo 3 lettere minuscole
  - "***e***?"
    - 0 o 1 occorrenza di ***e***
      - ***-?[1-9][0-9]**** indica un numero con un eventuale segno - inziale

Espressioni Regolari (5)

- Semantica caratteri speciali:
  - \...
    - Escape di un carattere speciale [,],.,*,+… oppure sequenze predefinite
    - *Es:* \++ indica una sequenza di uno o più caratteri +
    - \d indica una cifra (equivalente a [0-9])
    - \D indica non una cifra
    - \s indica uno spazio
    - \S indica un non spazio
    - …
  - "(…)"
    - Aggrega una sequenza di lettere e espressioni regolari
      - ***a(b[0-9])+*** indica una **a** seguita da una o più **b** seguita da una cifra (es: ab2b4b1)

Espressioni Regolari (6)

- Semantica caratteri speciali:
  - "^…"
    - Indica che ciò che segue ^ deve essere all'inizio della stringa da controllare
      - Es: /^[0-9]/ indica che la stringa deve iniziare con una cifra
  - "…$"
    - Indica che ciò che precede $ deve essere alla fine della stringa da controllare
      - Es: /[0-9]$/ indica che la stringa deve finire con una cifra
  - "^…$"
    - Indica che ciò che sta tra ^ e $ deve indicare completamente la stringa
      - Es: **/^[-\+]?\d+(\.\d*)?$/** indica che la stringa deve essere un numero decimale (es: 123, +23, -23.90, 0.99)

Espressioni Regolari (7)

- Esempi:
  - Un IP (es: 150.217.15.241):
    /([0-9]{1,3}\.){3}[0-9]{1,3}/
  - Un MAC addr (es: 12:a2:23:aB:19:90)
    /([0-9a-fA-F]{2}:){5}[0-9a-fA-F]{2}
  - Una data (es: 12/02/2008)
    @[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}@
  - Una e-mail (es: m.rossi@s.s-x.com)
    /[a-zA-Z\.\-0-9]+@[a-zA-Z0-9\.\-]+/

Espressioni Regolari (7)

- Funzioni
  - int preg_match($pattern, $testo [$matches])
    - Ritorna 1 se il pattern fa match con il testo (0 altrimenti) e in $matches viene inserito la parte del testo che ha fatto match.
    - Es:

      preg_match('/-?\d+/', 'abc -345 egf 23', $m)

      Ritorna 1 e $m==array('-345')
  - int preg_match_all($pattern, $testo, [$matches])
    - Ritorna il numero di parti del testo che fanno match con il pattern e in $matches vengono messi le parti del testo che fanno match.
    - Es:

      preg_match_all('/-?\d+/', 'abc -345 egf 23', $m)

      Ritorna 2 e $m==array(array('-345', '23'))

Espressioni Regolari (8)

- Funzioni:
  - string preg_replace($pattern, $replace, $testo)
    - Ritorna un nuovo testo dove tutte le occorrenze del pattern sono sostituite con il contenuto di $replace.
    - Es:

      preg_replace('/-?\d+/', 'N', 'abc -345 egf 23')

      Ritorna "abc N egf N"
    - Nella stringa $replace si possono indicare parti del pattern che ha fatto match usando \1, \2, \3 … che indicano le parti del pattern tra (…)
    - Es:

      preg_replace('/<(\d+)\.([a-z]*)>/', '(\2:\1)', 'abc<23.ab>defg<56.bf>')

      Ritorna: "abc(ab:23)defg(bf:56)"

      preg_replace('@([0-9]{1,2})/([0-9]{1,2})/([0-9]{4})@','\3-\2-\1', "abc 12/2/2004 cdef 26/8/2008 sed")

      Ritorna: "abc 2004-2-12 cdef 2008-8-26 sed"

Espressioni Regolari (9)

- Altre funzioni:
  - **preg_grep**, ricerca elementi di array che fanno match con un pattern
  - **preg_slice**, spezza una stringa in sottostringhe sulla base di un pattern

Espressioni Regolari (10)

- Modificatori del pattern:
  - Alla fine di un pattern si possono specificare una o più lettere che cambiano il comportamento del pattern: '/[a-z]+/**i**'
    - ***i*** : match case-insensitive
    - ***m*** : **^** e **$** fanno match anche dopo e prima di un a capo (multiline)
    - **x** : ignora gli spazi tra elementi del pattern
    - **e** : solo con replace interpreta la stringa da sostituire come codice PHP che produce la stringa da inserire
    - …

Espressioni Regolari (11)

- Match greedy (goloso)
  - Quando viene fatto il match di * o + (specialmente se usati con .) viene fatto il match con il numero maggiore possibile di caratteri.
  - Es:
    - pattern = @<a.*>.*@
    - testo = ‘abc**defghilm<a>**nopq****rst’
  - Per ovviare a questo si usa pattern:
    - @<a[^>]*>[^<]*@

    oppure
    - @<a.*?>.*?@

Sistemi Distribuiti

Corso di Laurea in Ingegneria

Programmare per il Web

1. Parte I: Espressioni Regolari
2. Parte II: Javascript / Approfondimenti

JavaScript

- Nasce nel 1995 con il nome ‘Mocha’ grazie a Brendan Eich (fondatore di Netscape)
- Negli anni ‘90 si parla di Dinamic HTML (DHTML)
- Nel ‘97 nasce lo standard internazionale ECMA-262 (ECMAScript), per regolamentare le specifiche javascript, http://www.ecma-international.org
- Nel 2005 Jesse James Garrett rilascia un white paper in cui conia il nome ‘Ajax’ per descrivere una serie di tecnologie per creare applicazioni web, tra cui JavaScript
- Nel 2009 si ha la versione ECMAScript 5
- Giugno 2015 ECMAScript 6, attuale versione
- Riferimenti:
  - https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript
  - https://www.w3.org/standards/webdesign/script

JavaScript

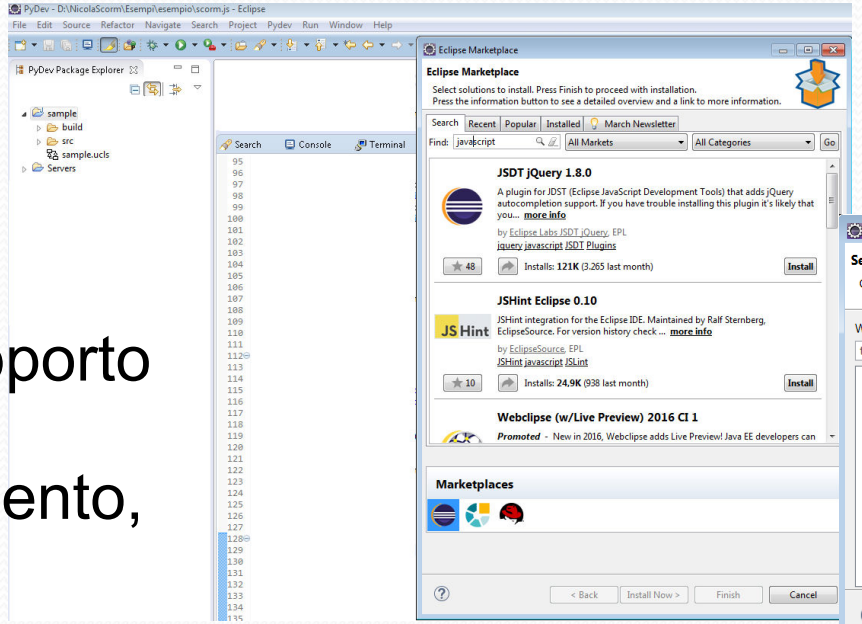
- E’ un linguaggio di scripting open source orientato agli oggetti e agli eventi
- Usato per:
  - Programmazione web sia lato server che, soprattutto lato client
  - Gestione azioni interattive e aggiunta di maggiore dinamicità alle pagine web
- Comunicazione sia sincrona che asincrona con il server

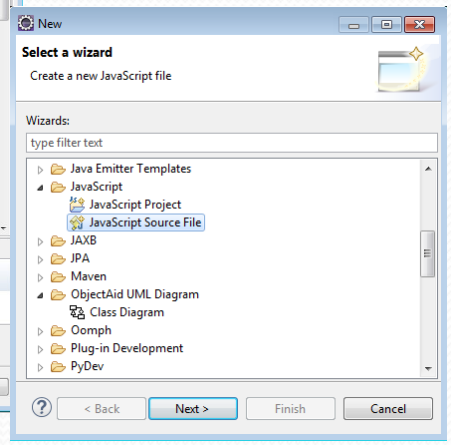
Concetto di script lato client

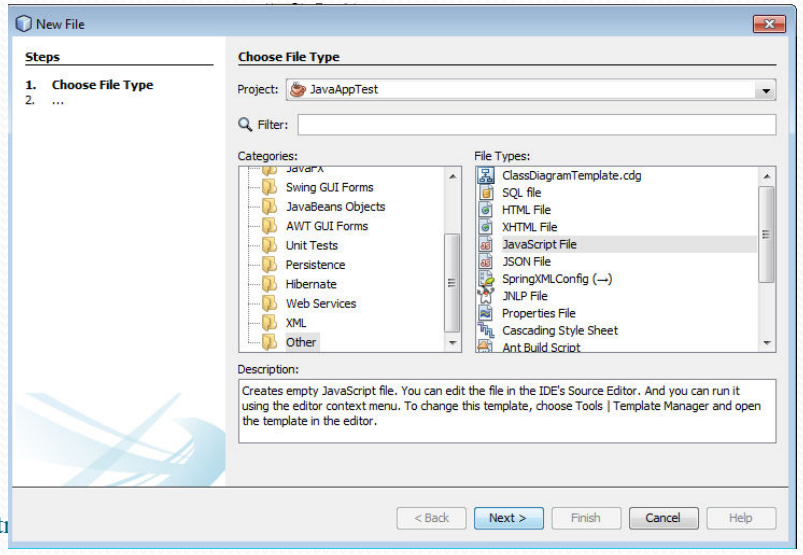
- Uno script lato client è un programma che affianca un documento HTML (embedded)
- Viene eseguito, sulla macchina del client, quando questo effettua il load della pagina HTML
- Azioni effettuate dagli script:
  - Modifica dinamica dei contenuti visualizzati
  - Controllo dinamico dei valori di input nei form
  - Possono essere attivati in base ad eventi effettuati dall'utente sulla pagina web (download, upload, movimenti del mouse, etc.)
  - Possono produrre elementi grafici
  - etc.
- Tipologie di Script:
  - Eseguiti una volta nel momento in cui l'utente carica la pagina
  - Eseguiti in base alle azioni effettuate dagli utenti (nel momento in cui si verificano)

Strumenti di sviluppo

- Editor di testo
- Strumenti che forniscono il supporto a javascript (autocompletamento, etc.):
  - Eclipse (pacchetto «Eclipse JavaScript Development Tools»)
  - Netbeans
  - …

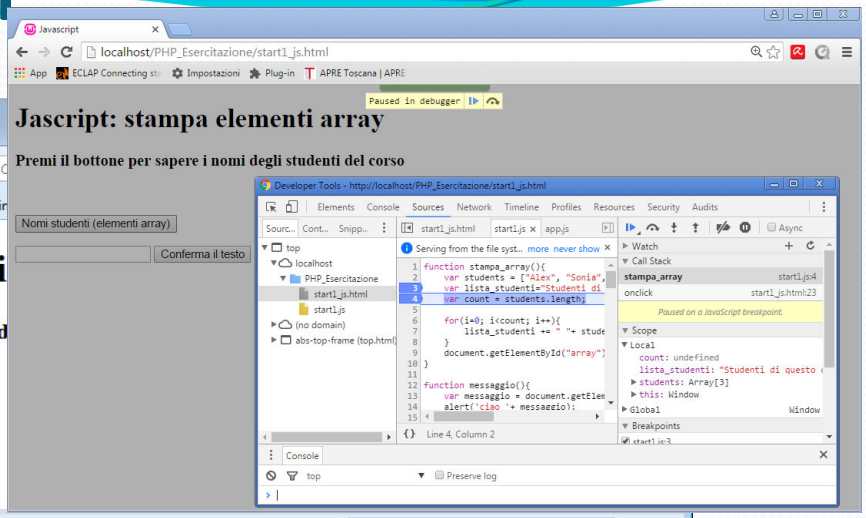


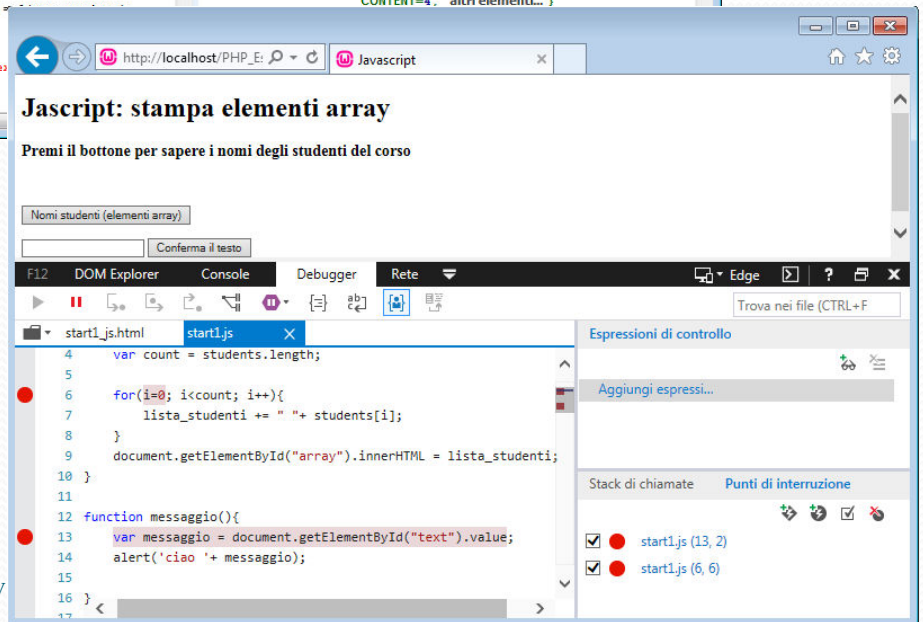




Strumenti di sviluppo nei browser

- Usare i Browser come strumenti per fare debug:
  - Crome
  - Firefox
  - Internet Explorer





Script in HTML

- E' necessario comunicare allo User Agent che tipo di linguaggio si sta usando per lo script:
  - <META http-equiv="Content-Script-Type" content="type">

  CON 'type' = {"text/html", "image/png", "image/gif", "video/mpeg", "text/css", "audio/basic", etc.}
- Può essere incluso si internamente al codice HTML che esternamente (link ad un file):

file esterno

Codice js interno alla pagina html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>A document with SCRIPT</TITLE>
<META http-equiv="Content-Script-Type" content="text/tcl">
<SCRIPT type="text/vbscript" src="http://someplace.com/progs/vbcalc">
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
...some JavaScript...
</SCRIPT>
</BODY>
</HTML>
```

Esempio: Javascript in HTML ....

```
<!DOCTYPE>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
    <title>Form2</title>
    <script src="form4.js"></script>
  </head>
  <body >
     <h1>Form e jascript</h1>
     <script>alert('Benvenuto! Immetti i tuo dati!')</script>
     <form action="action4_js.php" method="POST">
       <table align="left">
        <tr>
          <td> <a href="javascript:alert('Scrivi qui il tuo Nome!')"> Il tuo Nome: </td>
          <td> <input type="text" name="name" value="" /> </td>
        </tr>
        <tr>
          <td> Il tuo Cognome: </td>
          <td> <input type="text" name="surname" value="" /> </td>
        </tr>
        <tr>
          <td> La tua e-mail: </td>
          <td> <input type="email" name="email" value ="" />  </td>
        </tr>
        <tr>
          <td> <input type="submit" value="Invia" onclick="alert('Ci stai inviando...!')" > </td>
           <td> <button type="button" onclick="conferma()">Informazioni...</button> </td>
        </tr>
       </table>
     </form>
  </body>
</html>
```

```
function conferma() {
    if (confirm("Vuoi proseguire?")) {
        alert("Controlla i dati e clicca su 'Invia'");
    }else {
        alert('Grazie per averci contattato...')
        window.location="http://www.disit.org"
    }
}
```

file esterno (riferimento assoluto o relativo)

Blocco di codice nella pagina HTML

1

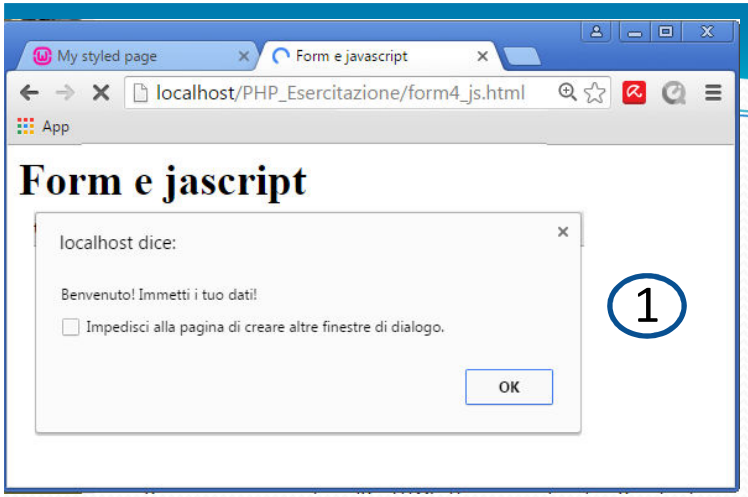
2a

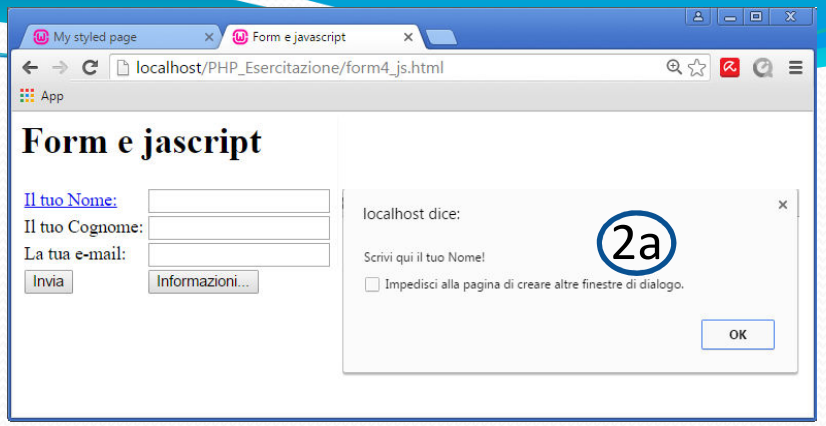
inline

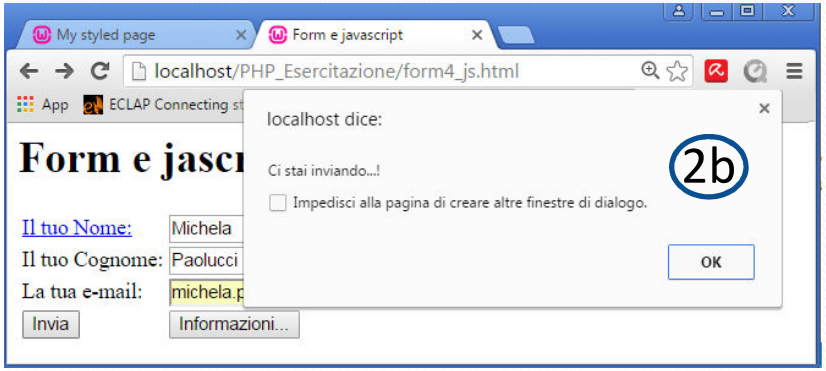
2b

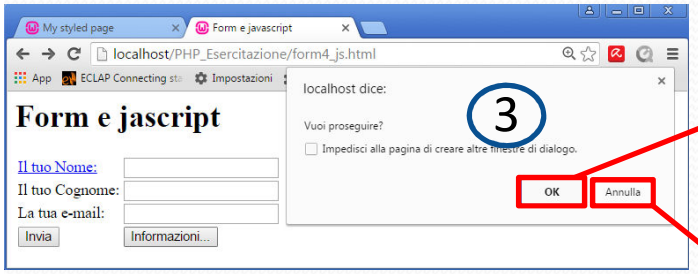
3

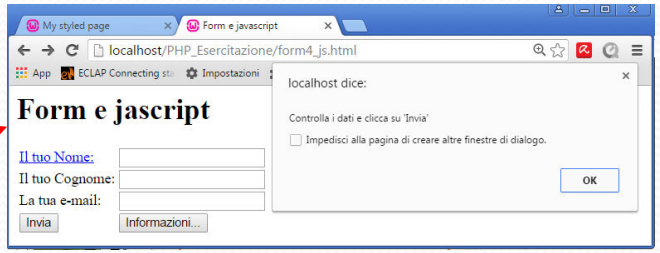
Funzione 'conferma()' definita nel file esterno

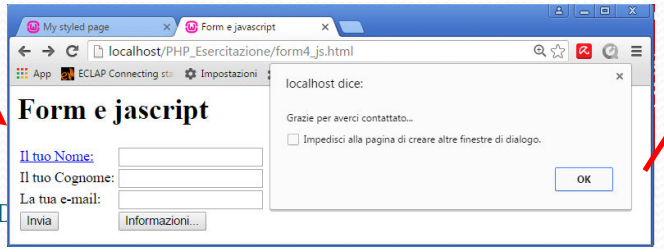

My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
Form e jascript
localhost dice:
Benvenuto! Immetti i tuo dati!
Impedisci alla pagina di creare altre finestre di dialogo.
OK
1


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
Form e jascript
Il tuo Nome:
Il tuo Cognome:
La tua e-mail:
Invia
Informazioni...
localhost dice:
Scrivi qui il tuo Nome!
Impedisci alla pagina di creare altre finestre di dialogo.
OK
2a


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
ECLAP Connecting st
Form e jasci
Il tuo Nome:
Michela
Il tuo Cognome:
Paolucci
La tua e-mail:
Invia
Informazioni...
localhost dice:
Ci stai inviando...!
Impedisci alla pagina di creare altre finestre di dialogo.
OK
2b


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
ECLAP Connecting st
Impostazioni
Form e jascript
Il tuo Nome:
Il tuo Cognome:
La tua e-mail:
Invia
Informazioni...
localhost dice:
Vuoi proseguire?
Impedisci alla pagina di creare altre finestre di dialogo.
OK
Annulla
3


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
ECLAP Connecting st
Impostazioni
Form e jascript
Il tuo Nome:
Il tuo Cognome:
La tua e-mail:
Invia
Informazioni...
localhost dice:
Controlla i dati e clicca su 'Invia'
Impedisci alla pagina di creare altre finestre di dialogo.
OK


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
ECLAP Connecting st
Impostazioni
Form e jascript
Il tuo Nome:
Il tuo Cognome:
La tua e-mail:
Invia
Informazioni...
localhost dice:
Grazie per averci contattato...
Impedisci alla pagina di creare altre finestre di dialogo.
OK


http://www.disit.dinfo.unifi.it
DISIT LAB OVERVIEW
Sii-Mobility

Commenti

- Esistono i seguenti metodi:
  - Commento in line
    //ecco un commento
  - Commento su più righe
    /*Commento
      su più righe
      …*/

Variabili

- Nomi delle variabili:
  - sono ‘case sensitive’
  - Si possono usare le lettere (A .. Z, a .. z), I numeri (0 .. 9), il ‘_’ (underbar, non come carattere iniziale
  - non possono contenere gli altri caratteri speciali:
    - Spazio, trattino (-), punto (.), punto interrogativo (?), dollaro ($), etc.
- Sintassi:
  - Dichiarazione (esplicita): **var** x = 10;
  - Dichiarazione implicita: x = 10;

  NOTA: se si abilita **lo strict mode,** si riceve una segnalazione nel caso in cui si usino variabili non dichiarate:
    - "use strict"
- Esempi validi:
  - **var** var1
  - **var** _variabile

spazi bianchi, ‘;’, costanti

- Il ‘;’ serve per determinare la fine di una espressione. Non è obbligatorio:
  - **var** x = 10;
  - **var** x = 10 /* questa dichiarazione è equivalente alla precedente*/
- Spazi bianchi:
  - Assumono un significato solo all’interno delle stringhe
- Costanti:
  - Non sono previste, si ricorre all’uso delle variabili
  - Dalla versione 6:
    - const PIGRECO = 3.14;

Tipi di dati

- Boolean
- Null
- Undefined
- Numeri
- Stringhe
- Array
- Object

Tipi di Dati: Boolean/Null/Undefined

- Il **Boolean** è il tipo di dato più semplice, può assumere due valori: True o False:
  - **var** myVariable = ***true***;
  - **var** myVariable = ***false***;

- Il tipo di dato Null prevede la notazione:
  - **var** x= ***null***;

- Il tipo di dato Undefined rappresenta un valore inesistente e prevede la notazione
  - **undefined**

Tipi di Dati: Numeri

- Esiste un unico tipo di dato:
  - Intero (se nn è specificata la parte decimale):
    - **var** negativo = -10;
    - **var** positivo = 596;
  - Decimale:
    - **var** decimale = -0.10;
  - Notazione scientifica:
    - **var** decimale 13e4;
  - Notazione Ottale/Esadecimale:
    - **var** ottale = 0134;
    - **var** esadecimale = 0x123;
  - Valori speciali:
    - Infinity | -Infinity

Tipi di Dati: Stringhe

- Tipo di variabile che contiene testo.
- Si hanno due modalità:
  - Tra apici (‘testo’), in questo caso se si vuole inserire un apice nella stringa, è necessario farlo precedere da backslash (\’) , il carattere backslash può essere inserito raddoppiandolo(\\)
    - **var** stringa = ‘testo dentro ad un file \’javascript\’… ’;
  - Tra virgolette ("testo"), in questo caso si possono usare i caratteri speciali del linguaggio di programmazione C (\n,\r,\\,\t, …) e si può includere il contenuto di altre stringhe:
    - **var** stringa = "metto un ritorno a capo \\nNel testo ";

Tipi di Dati: Array (1)

- Un array, contiene una serie di valori accessibili tramite un indice
- Definire un array, sintassi:
  - **var** studenti = [‘nome1’, ‘nome2’];
  - **var** studenti = **new Array()**;
    studenti = [‘Anna’, ‘Claudio’, ’Simone’];
  - **var** studenti = **new Array(**‘Anna’, ‘Claudio’, ’Simone’**)**;
- Esempi:
  - **var** studenti = [,‘Anna’, ‘Claudio’,];
    - si crea un array di quattro elementi in cui il primo e l’ultimo sono di tipo ‘undefined’
  - **var** array_tipi_diversi = [‘stringa’, 123, true, null];
  - **var** array_in_array = [‘stringa’, [1, 23, 4], null];
    - **var** elemento = array_in_array[2][3]; // vale 4
  - **var** matrice = [[1,2,3],[4,5,6],[7,8,9]]; //matrice 3x3
    - **var** elem = matrice [2][3]; //elem ha valore 6;

Manipolazione degli array: (1)

- Alcune proprietà/metodi:
  - (P) *length* - Restituisce la dimensione dell'array
    - **var** studenti = ['Anna', 'Claudio', 'Simone'];
    - **var** dimensione = studenti.length;
    - // risultato: dimensione=3;
  - (M) *concat* – Esegue la concatenazioni di due array
    - **var** studenti = ['Anna', 'Claudio', 'Simone'];
    - **var** nuovi = ['Mauro', 'Lara'];
    - **var** tutti = studenti.concat(nuovi);
    - //risultato: tutti= ['Anna', 'Claudio', 'Simone', 'Mauro', 'Lara']

Manipolazione degli array: (1)

- (M) *push* – Aggiunge un elemento in coda all'array
  - **var** studenti = ['Anna', 'Claudio', 'Simone'];
  - studenti.push('Lara');
  - //risultato: studenti = ['Anna', 'Claudio', 'Simone', 'Lara'];

- (M) *pop* – Rimuove un elemento dalla coda
  - **var** studenti = ['Anna', 'Claudio', 'Simone'];
  - studenti.pop();
  - //risultato: studenti = ['Anna', 'Claudio'];

Manipolazione degli array: (2)

- (M) *shift* - Rimuove il primo elemento dell'array
  - **var** studenti = ['Anna', 'Claudio', 'Simone'];
  - studenti.shift();
  - //risultato: studenti = ['Claudio', 'Simone'];
- (M) *reverse* – Inverte l'ordine degli elementi di un array
  - **var** studenti = ['Anna', 'Claudio', 'Simone'];
  - studenti.reverse();
  - //risultato: studenti = ['Simone', 'Claudio', 'Anna', 'Mauro', 'Fabio'];
- (M) *slice* – Seleziona gli elementi di un array in base alla posizione
  - **var** studenti = ['Simone', 'Claudio', 'Anna', 'Mauro', 'Fabio'];
  - studenti.slice(1,4);
  - //risultato: studenti = ['Claudio', 'Anna', 'Mauro'];

Tipi di Dati: Object (1)

- Anche gli oggetti sono variabili
- Sintassi:
  - var object_void = {};
  - var object = {proprietà1: "valore1, …, proprietàN: "valoreN" };
- Nomenclatura:
  - I nomi delle proprietà SE racchiusi tra doppi apici NON hanno restrizioni come quelli delle variabili
- Esempi:
  - var studente = {nome: "Andrea"; corso: "Sistemi Distribuiti"; AA: "2016"};
  - var stud = {"nome-stud" : "Andrea", "corso.ing" : "Sistemi Distribuiti", "AA/" : "2016"};

Tipi di Dati: Object (2)

- Si possono creare oggetti annidati:
  - var persona= {
    nome: "Andrea",
    cognome: "Rossi",
    indirizzo: { //oggetto composto da altre proprietà
    via: "S. Marta",
    numero: "3",
    città: "Firenze"
    }
    }

- A differenza degli array, gli indici sono le proprietà:
  - var nome_persona = persona["nome"];
  - var cognome_persona = persona.cognome;

Tipi di Dati: Object (3)

- Si possono creare metodi relativi agli oggetti:
  - var persona= {

```
nome: "Andrea",
cognome: "Rossi",
indirizzo: { //oggetto composto da altre proprietà
            via: "S. Marta",
            numero: "3",
            città: "Firenze"
} ,
nomeCognome: function(){
            return "nome e cognome: "  + persona.nome +
            persona.cognome;
}
}
```

- Richiamo il metodo:
  - var nome_cognome = persona.nomeCognome();

Tipi di Dati: Object (4)

- Costruttore:

```
function person(firstName, lastName, age, eyeColor) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.eyeColor = eyeColor;
        this.changeName = function (name) {
                this.firstName = name;
        }
}
```

- Richiamo il metodo:
  - `var myMother = new person('Maria','Tirro', '55', 'blu');`
    `myMother.changeName("Doe");`
    `console.log(myMother.firstName); // scrivo su console`

Tipi di Dati: Object (5)

- Costruttore:
  - var persona= {
    nome: "Andrea",
    cognome: "Rossi",
    saluta: function(){
    alert("Benvenuto " + nome + " " + cognome);
    }
    }

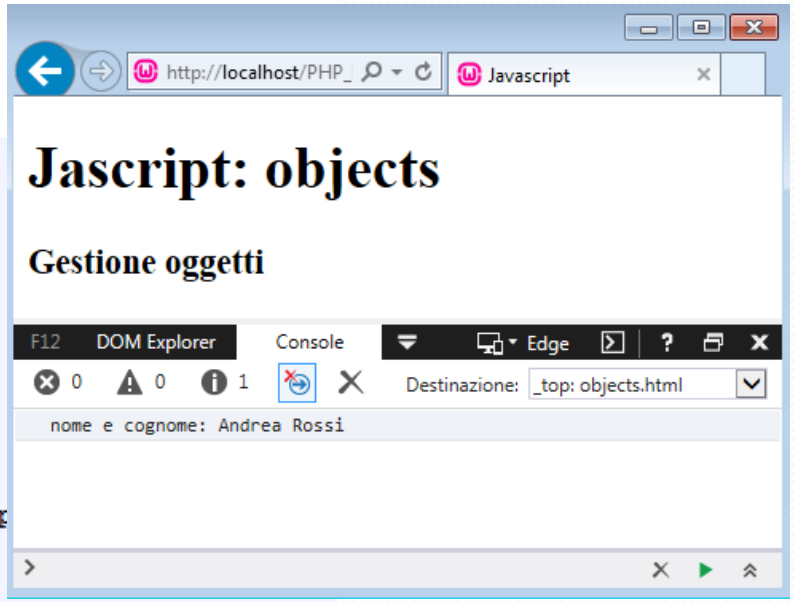
- Richiamo il metodo:
  - document.getElementById("pulsante").addEventListener("click", persona.saluta);

  //contesto DOM: lo associa al click dell'utente su un bottone

Tipi di Dati: Object (6)

E

```
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    </head>
    <body>
    <h1>Jascript: objects</h1>
    <h3>Gestione oggetti</h3>
    <script>
        var persona = {
            nome: "Andrea",
            cognome: "Rossi",
            indirizzo: { //oggetto composto da altre proprietà
                via: "S. Marta",
                numero: "3",
                città: "Firenze"
            },
            nomeCognome: function(){
                return 'nome e cognome: '+persona.nome + ' '+ p
            }
        }
        console.log(persona.nomeCognome());

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



Tipizzazione

- Se si dichiara una variabile senza specificarne il valore, essa assume il valore di default 'undefined'
- Il tipo di dato relativo ad una variabile si può cambiare tramite assegnazioni successive
  - **var** variabile;
  - variabile = 596;
  - variabile = "stringa";
  - variabile = true;
  - variabile = null;
- Dichiarazione di tipo iniziale:
  - **var** variabile2 = true; // tipo: boolen
- typeof: serve per verificare il tipo delle variabili:
  - "string", "boolean", "number", "function", "object", "undefined", "xml"

Operatori

- Aritmetici
- Logici
- Assegnazione
- Per Stringhe
- Relazionali
- …

Operatori Aritmetici

- Operatori binari

Operatore	Descrizione
+	addizione
-	sottrazione
/	divisione
*	moltiplicazione
%	Modulo o resto

- Operatori unari

Operatore	Descrizione
-	negazione
++	incremento
--	decremento

Operatori Logici

Operatore	Descrizione
&&	and
||	or
!	not

Operatori di Assegnazione

Operatore	Descrizione/esempio
=	x = 3+7, x assume il valore della espressione ‘3+7’
… ? ... : …	ternario x = *condizione* ? *val2* : *val3* x vale *val2* SE *condizione* è true, vale *val3*

Forma compatta	Descrizione
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Operatori per stringhe

- Ci sono due operatori per stringhe:
  - Il primo è l'operatore di concatenazione (‘+'), che restituisce la concatenazione dei suoi argomenti a destra e a sinistra:

```
var a = "Ciao ";
var b = a + "Mondo!"; // ora b contiene "Ciao Mondo!"
```

  - Il secondo è l'operatore di assegnazione concatenata (‘+='), che aggiunge alla fine dell'argomento sul lato destro l'argomento sul lato sinistro:

```
a = "Ciao ";
a += "Mondo!"; // ora a contiene "Ciao Mondo!"
```

Operatori & Tipi di dato (1)

- Conversioni implicite nel caso di conversione a partire da un tipo di dato a Boolean:

Tipo di dato	**Boolean**
undefined	false
null	false
numero	false se 0 o NaN (Not a Number), true in tutti gli altri casi
stringa	False se la stringa è vuota, true in tutti gli altri casi

Operatori Relazionali

Operatore	descrizione
<	minore
<=	minore o uguale
>	maggiore
>=	maggiore o uguale
==	uguale
!=	diverso
===	strettamente uguale
!==	strettamente uguale

Operatori & Tipi di dato (2)

- Conversioni implicite nel caso di conversione a partire da un tipo di dato a Numero:

Tipo di dato	**Numero**
undefined	NaN
null	0
boolean	1 se true 0 se false
stringa	intero, decimale, zero o NaN, in base alla stringa in esame

Operatori & Tipi di dato (3)

- Conversioni implicite nel caso di conversione a partire da un tipo di dato a Stringa:

Tipo di dato	**Numero**
undefined	"undefined"
null	"null"
boolean	"true" se true "false" se false
numero	"NaN" se NaN, "Infinity" se Infinity, "stringa" che rappresenta il numero negli altri casi

Istruzioni

- Uno script Javascript è costituito da una serie di istruzioni
- Una istruzione può essere una assegnazione, una chiamata di funzione, un ciclo, …
- Le istruzioni terminano con un punto e virgola
- Le istruzioni si possono raggruppare in blocchi di istruzioni racchiudendole tra parentesi graffe
- Un gruppo di istruzioni è, a sua volta, una istruzione

Funzioni

- Una funzione è un blocco di codice che può richiedere uno o più parametri in ingresso e può fornire un valore di uscita
- Javascript mette a disposizione numerose funzioni predefinite

Creare Funzioni

```
function <nome_funzione>(<parametri>) {
        <lista di azioni>
 }
```

- Esempio:

```
function messaggio(){
  alert('Stampo Messaggio… !');
 }
```

- Le variabili definite in una funzione sono variabili locali, per accedere a variabili globali si usa l'istruzione const:
  - **const** nome = 'Chiara';

Usare le Funzioni

- Per utilizzare una funzione bisogna connetterla ad un evento nella pagina web e richiamarla (o invocarla)
- Tipi di eventi:
  - Page load
  - Interazioni con gli oggetti (elementi html, DOM) della pagina (Click su bottoni o link, movimenti del mouse, etc.)
  - …

Alcuni eventi HTML

- onload
- onclick
- onchange
- onmousemove, onmouseover, onmouseout, …
- onsubmit
- …

ESEMPIO

file.html

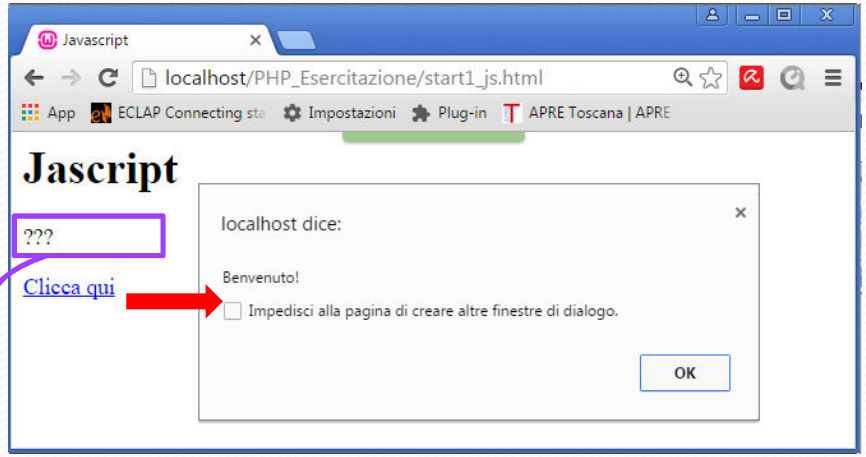
```
<!DOCTYPE>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    <script src="start1.js"></script>
    </head>
    <body >
    <h1>Jascript</h1>
    <p id="demo" onmouseover="stampa()" onmouseout="cancella()" > ??? </p>
    <div> <a href="javascript:alert('Benvenuto!')"/> Clicca qui</div>
  </body>
</html>
```

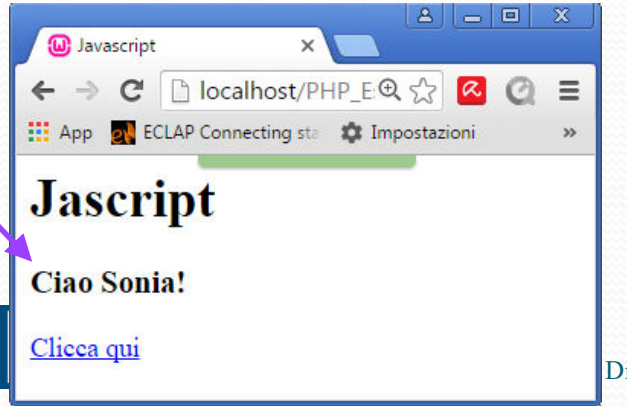
Eventi:

- onmouseover
- onmouseout

start1.js

```
function stampa(){
    var students = ["Alex", "Sonia", "Fabio"];
    document.getElementById("demo").innerHTML
    = "Ciao "+ students[1]+"!";
}

function cancella(){
    document.getElementById("demo").innerHTML = "???";
}
```

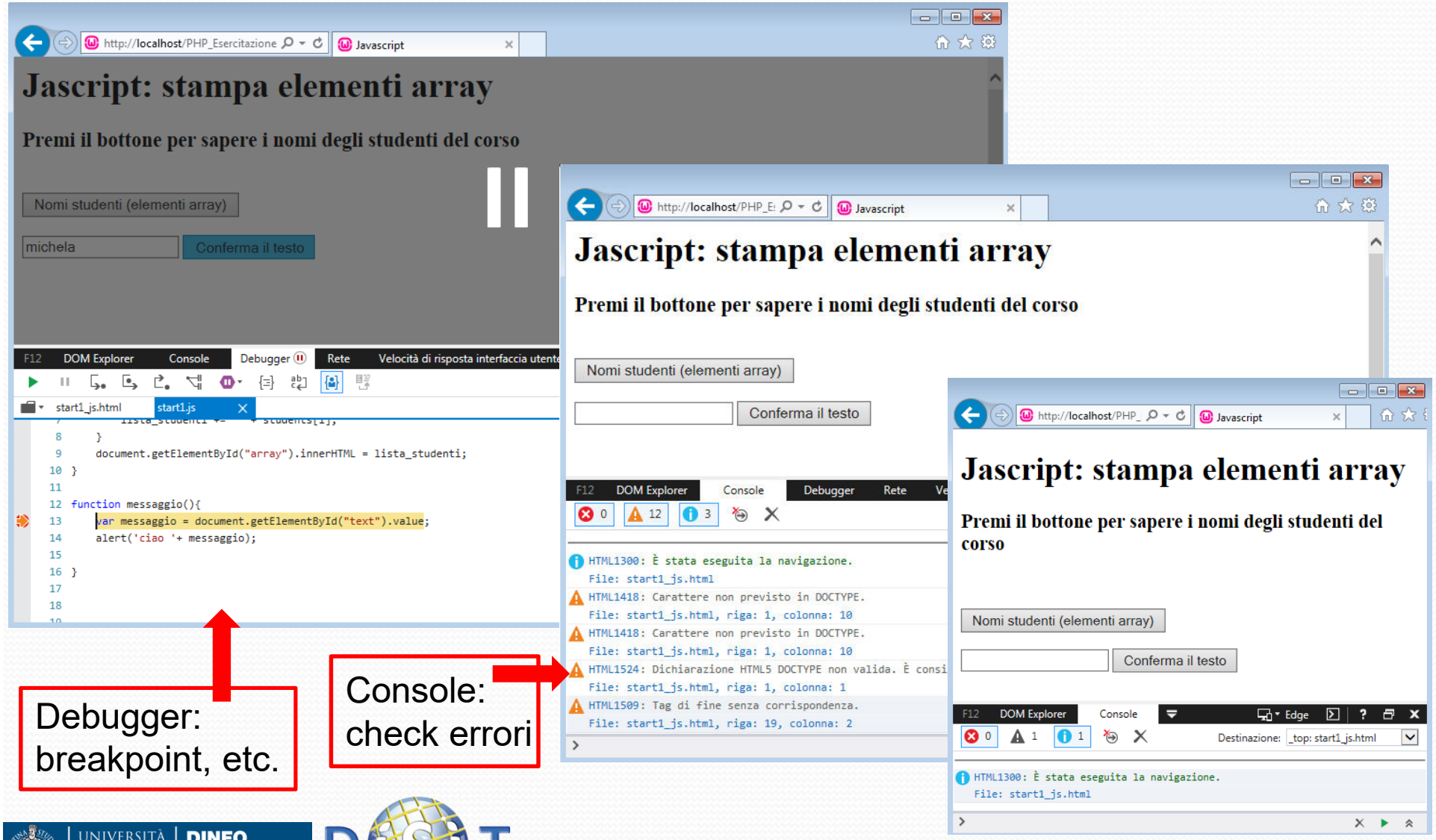




- Selettore:
  - document.getElementbyId("id")
- Proprietà:
  - innerHTML

NOTA: l'oggetto 'document', fa parte del DOM (Document Object Model) il modello a oggetti delle pagine HTML

ESEMPIO… usando Internet Explorer



Debugger: breakpoint, etc.

Console: check errori

Istruzioni: if else

- **Sintassi:**

```
if (condizione) {
  azione1 da effettuare;
  azione2;
}
else { //se la condizione non e'verificata
  altre azioni;
}
```

- Note:
  - Le parentesi graffe servono per raggruppare una serie di azioni
  - La clausola else { } è facoltativa, va usata nel caso ci sia una alternativa se if non soddisfa la condizione indicata fra le parentesi tonde
- Esempio:

```
if (a==b){
  alert('sono uguali');
}
else{
  alert('sono diversi');
}
```

elseif

- Sintassi:

```
if (a=='cond1'){
    alert('a vale: cond1');
}
elseif ('cond2') {
    alert('a vale: cond2');
}
...
else {
 alert('a vale: altro…');
}
```

NOTE:
Si tratta di un'altra istruzione IF all'interno di un IF. Il server controlla se il primo *if* è vero, se è falso va sul *elseif*, se è falso anche questo continua con gli *elseif* fino a quando non trova una alternativa oppure l'istruzione finale *else* (non obbligatoria)

Cicli: for

Sintassi:

```
for (espressione iniziale; condizione; aggiornamento) {
        lista azioni;
}
```

Esempio:

```
for(i=0; i<students.length; i++){
        lista_studenti += " "+ students[i];
}
```

NOTA: Se la variabile non raggiunge la condizione inserita dentro il ciclo si crea un loop infinito.

Cicli: for-in (array)

Sintassi:

```
var nome_array = …
var indice;
for (indice in nome_array) {
        < azioni>
}
```

Esempio:

```
var valori = [10, 20, 30, 40]
var indice;
var totale = 0;
for(indice in valori)
        totale += valori[indice];
}
```

Cicli: for-of (array)

Sintassi:

```
var nome_array = …
var indice;
for (indice in nome_array) {
        <azioni>
}
```

Esempio:

```
var numeri = [12, 35, 20, 7, 4, 2];
var tot_num = 0;
var valore;

for (valore of numeri) {
        tot_num +=   valore;
}
```

Stampare elementi di un array in un div

- Cicli: for o while
- Eventi:
  - onmouseover
  - onmouseout
  - onclick (<button>)
  - …
- Selettore/proprietà:
  - document.getElementbyId("id").innerHTML = ‘stringa’ + ‘!’;

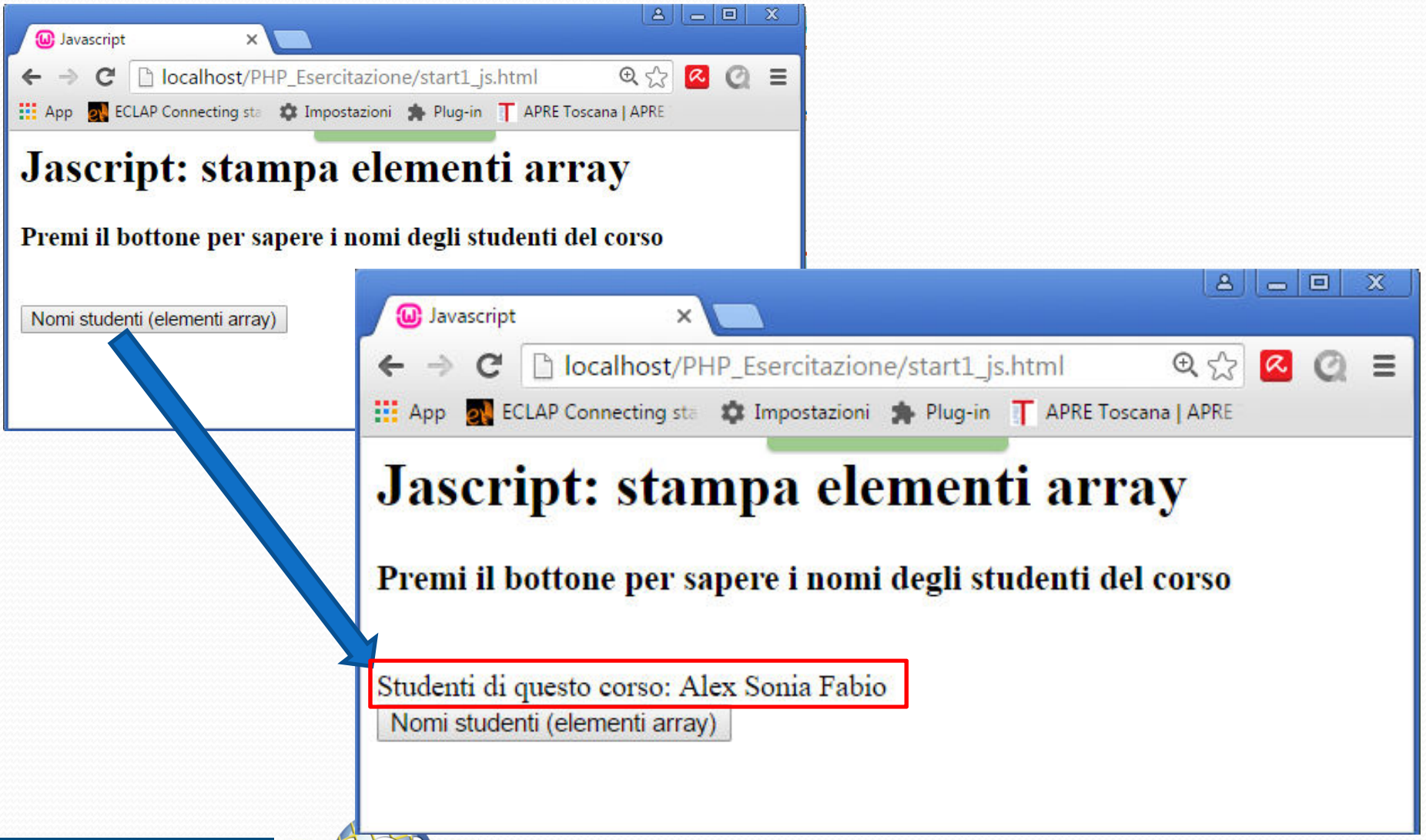
Stampare elementi di un array in un div (2)

```
<!DOCTYPE>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    <script src="start1.js"></script>
    </head>
    <body >
    <h1>Jascript: stampa elementi array</h1>
    <h3 > Premi il bottone per sapere i nomi degli studenti del corso   </h3>
    <br>
    <div id="array"> </div>
    <button onclick="stampa_array()">Nomi studenti (elementi array)</button>
  </body>
</html>
```

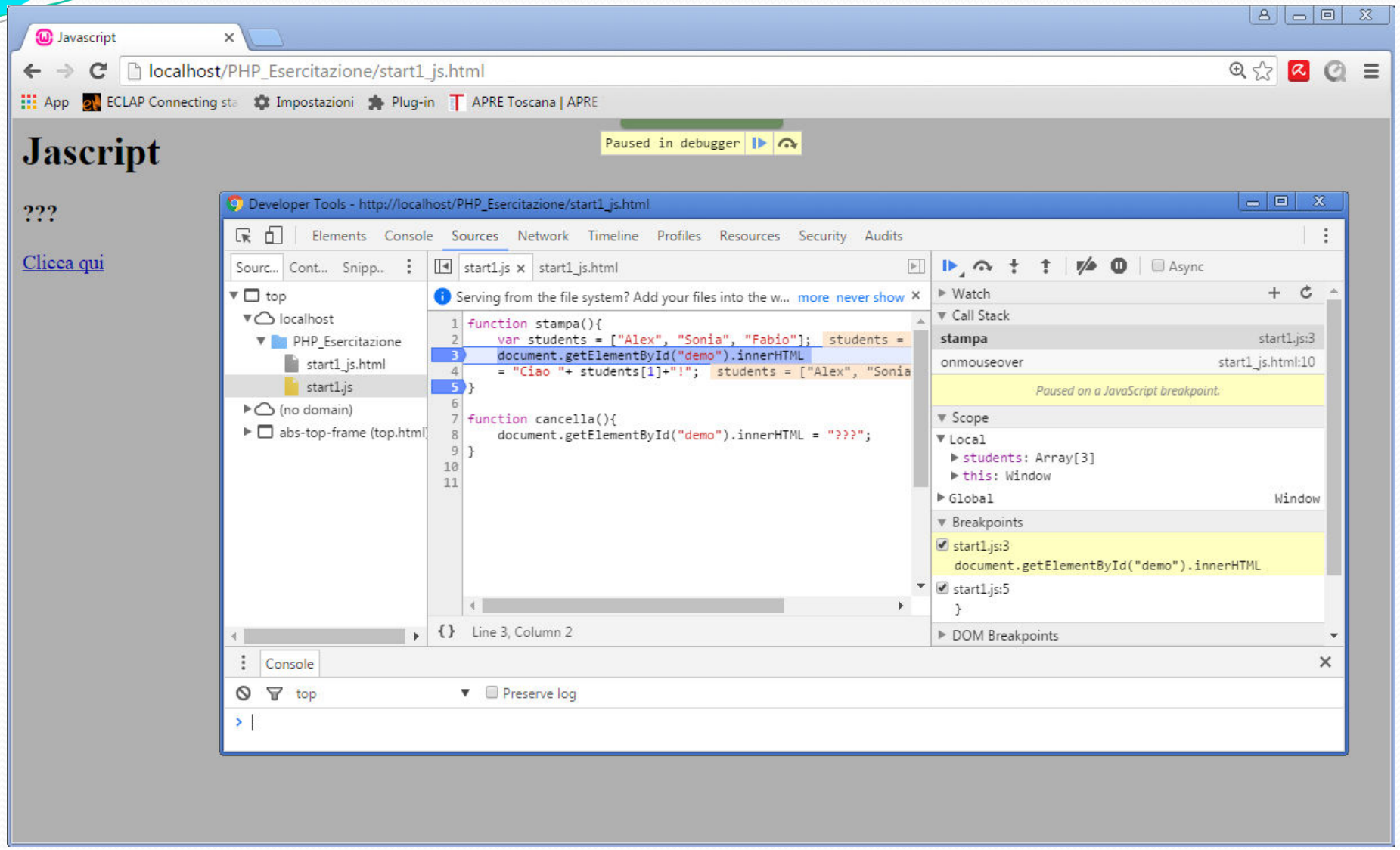
```
function stampa_array(){
    var students = ["Alex", "Sonia", "Fabio"];
    var lista_studenti="Studenti di questo corso: ";
    var count = students.length;

    for(i=0; i<count; i++){
        lista_studenti += " "+ students[i];
    }
    document.getElementById("array").innerHTML = lista_studenti;
}
```

Stampare elementi di un array in un div (3)

E

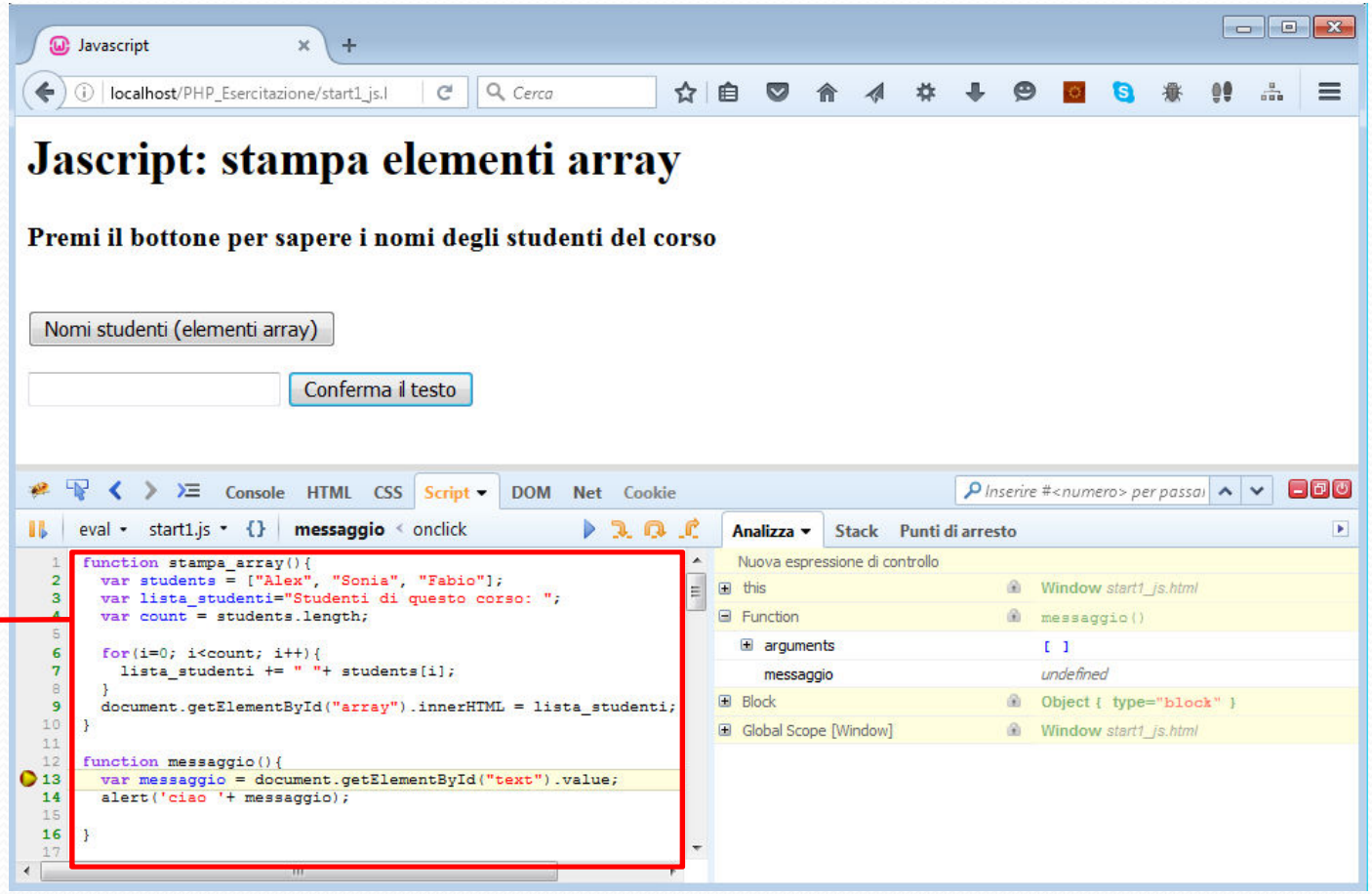


ESEMPIO… usando Crome



ESEMPIO… usando Firefox

- Attivazione del plugin ‘Firebug’
- javascript
- Debugger: breakpoint, etc.



Cicli: While

- Sintassi:

```
while(condizione) {
        azione1;
        azione2;
        //azione per far variare la condizione
}
```

NOTE: Il ciclo while dura fino a quando la condizione è vera. Per far questo dobbiamo necessariamente far variare la condizione all'interno del ciclo

- Esempio:

```
var lista_numeri = '';
var numero = 1;
while(numero <= 10){
        lista_numeri +=  ' '+ numero;
        numero +=1;
}
```

In questo caso il ciclo while continua fino a quando numero non raggiunge il valore 10

Cicli: do while

- È' simile al ciclo *while* MA mentre il ciclo *while* può non essere eseguito, il ciclo *do while* esegue sempre, almeno per una volta. Questo perché il ciclo *do while* inserisce prima le azioni da fare e dopo la condizione. Il server esegue le prime istruzioni, poi legge la condizione e se è sempre vera esegue nuovamente le istruzioni

- **Sintassi:**

```
do {
  azione1;
  azione2;
  //azione per far variare la condizione
}
while(condizione)
```

Interrompere un ciclo: break

- **Sintassi:**

```
while{
    azioni;
    if(condizione) break;
}
```

- **Esempio:**

```
var x= 0;
while (x < 10) {
    x++;
    if (x > 5) continue;
          console.log(x);// se x è maggiore di
                         // 5, il log non viene
                         // più eseguito
  }
```

Interrompere un ciclo: continue

- **Sintassi:**

```
while{
    azioni;
    if(condizione) continue;
}
```

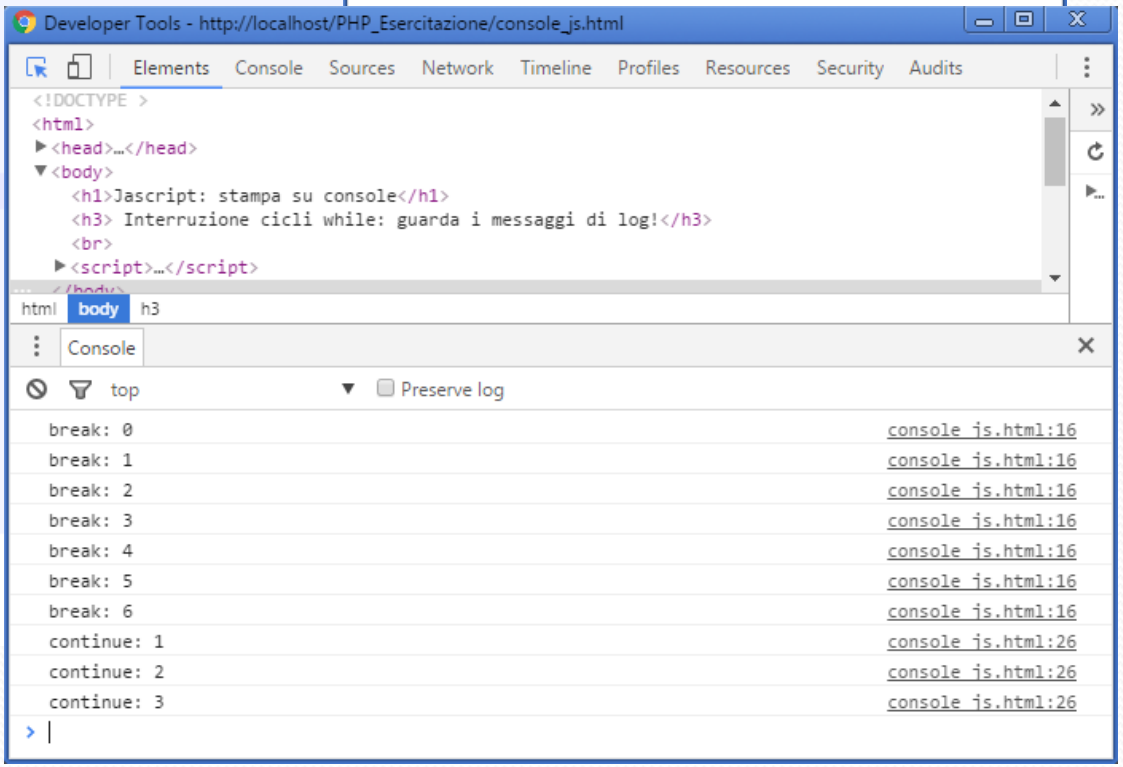
- **Esempio:**

```
var x= 0;
while (true) {
  console.log(x);
  // condizione di uscita
  if (x > 20) break;
  x++;
}
```

Esempio: Interruzione di ciclo e scrittura su console (Crome)

E

```
<!DOCTYPE>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-88
    <title>Javascript</title>
    </head>
    <body>
    <h1>Jascript: stampa su console</h1>
    <h3> Interruzione cicli while: guarda i messaggi di log!</h3>
    <script>
    var x= 0;
    while (true) {
        console.log('break: ' + x);
        // condizione di uscita
        if (x > 5) break;
        x++;
    }
    var x= 0;
    while (x < 10) {
        x++;
        if (x > 3) continue;
            // se x è maggiore di 3,
            //il log non viene più eseguito
            console.log('continue: '+ x);
    }
    </script>
  </body>
</html>
```



Switch

- Si usa se ci sono più alternative e non si vogliono inserire più *if* annidati.

```
switch (a) {
      case <espressione>:
            <lista azioni>
          ...
      default: //entra qui se nessuna condizione è verificata
            <lista azioni>
}
```

- Si usa se si deve gestire una variabile in maniera diversa, in base al valore che assume: con l'istruzione if dovremmo scrivere due if annidati, con switch ne basta uno:

```
switch(stringa){
      case 'ciao':
            alert("Ci vediamo presto");
            break;
      case 'addio':
            alert("Non torni più?");
            break;
      default:
            alert("Forse tornerai");
            break;
}
```

Try catch throw finally (1)

- try {
  <lista azioni>
  }
  catch(err) {
  <lista azioni da eseguire in caso di errore (stampa err)>
  }

Esempio:

```
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    </head>
    <body >
    <h1>Jascript: gestione errori</h1>
    <h3 > Funziona tutto correttamente? </h3>
    <p id="demo"></p>
    <script>
        try {
            AAAAalert("Welcome guest!"); //errore voluto...
        }
        catch(err) {//scrive nel paragrafo con id='demo'
            document.getElementById("demo").innerHTML = err.message;
        }
    </script>
  </body>
</html>
```

try catch throw finally (2)

- try {
  <lista azioni>
  if(condizione) throw <eccezione>
  /*lista di condizioni che sollevano eccezioni personalizzate*/
  }
  catch(err) {
  <lista azioni da eseguire in caso di errore (stampa err)>
  }
  finally{
  <lista azioni da eseguire in caso>
  /* indipendentemente dagli errori sollevati */
  }

Esempio:
uso di try catch throw finally

```html
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    <script src="errori_javascript.js"></script>
    <style>#error {color: red;} #finally{color: green;} </style>
    </head>
    <body >
    <h1>Jascript: gestione errori</h1>
    <h3>Inserisci un numero da 1 a 10:</h3>
    <br>
    <input type ="text" id="text" />
    <button type ="submit" onclick="validazione()"> Conferma </button>
    <br>
    <p id="error"></p> <!-- per messagg
    <p id="message"></p> <!-- per messa
    <p id="finally"></p> <!-- sempre --
  </body>
</html>
```

```
function validazione(){
    var x;
    x = document.getElementById("text").value;
    try {
        if(x>=1 && x<=10){//ci passa se va a buon fine
            document.getElementById("message").innerHTML =
                                    "Hai inserito correttamente: " + x;
            document.getElementById("error").innerHTML = "";
            }
        else{
            document.getElementById("message").innerHTML = "";
            if(x == "") throw "campo vuoto, inserisci un numero da 1 a 10!";
            if(isNaN(x)) throw "hai inserito una stringa, non un numero! Riprova!";
            x = Number(x);
            if(x < 1) throw "numero troppo piccolo! Riprova!";
            if(x > 10) throw "numero troppo grande! Riprova!";
        }
    }
    catch(err) {//ci passa se viene sollevato un errore
        document.getElementById("error").innerHTML = "Errore: " + err;
        //alert(err);
    }
    finally{// ci passa sempre
        document.getElementById("finally").innerHTML =
                                "Passo da finally: eseguo in ogni caso... ";
    }
}
```

Esempio: uso di try catch throw finally File.html

E

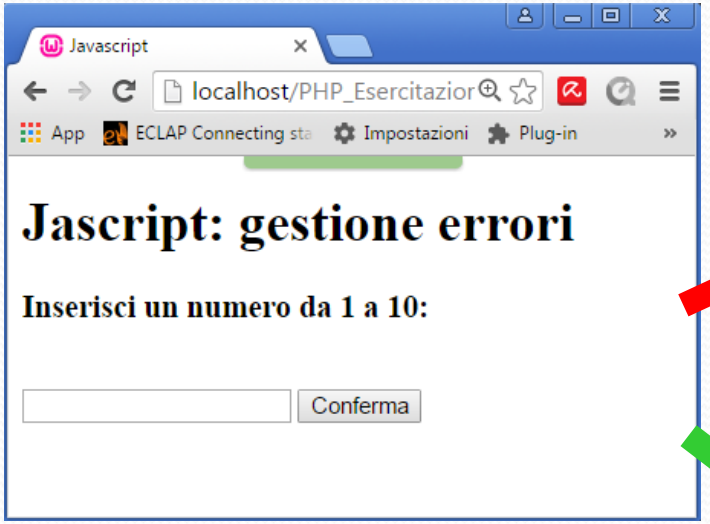
```
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    <script src="errori_javascript.js"></script>
    <style>#error {color: red;} #finally{color: green;} </style>
    </head>
    <body >
    <h1>Jascript: gestione errori</h1>
    <h3>Inserisci un numero da 1 a 10:</h3>
    <br>
    <input type ="text" id="text" />
    <button type ="submit" onclick="validazione()"> Conferma </button>
    <br>
    <p id="error"></p> <!-- per messaggi errore -->
    <p id="message"></p> <!-- per messaggi inserimento corretto -->
    <p id="finally"></p> <!-- sempre -->
  </body>
</html>
```

Esempio: uso di try catch throw finally

E

File.js

```
function validazione(){
    var x;
    x = document.getElementById("text").value;
    try {
        if(x>=1 && x<=10){//ci passa se va a buon fine
            document.getElementById("message").innerHTML =
                                    "Hai inserito correttamente: " + x;
            document.getElementById("error").innerHTML = "";
            }
        else{
            document.getElementById("message").innerHTML = "";
            if(x == "") throw "campo vuoto, inserisci un numero da 1 a 10!";
            if(isNaN(x)) throw "hai inserito una stringa, non un numero! Riprova!";
            x = Number(x);
            if(x < 1) throw "numero troppo piccolo! Riprova!";
            if(x > 10) throw "numero troppo grande! Riprova!";
        }
    }
    catch(err) {//ci passa se viene sollevato un errore
        document.getElementById("error").innerHTML = "Errore: " + err;
        //alert(err);
    }
    finally{// ci passa sempre
        document.getElementById("finally").innerHTML =
                                "Passo da finally: eseguo in ogni caso... ";
    }
}
```



- Uno dei possibili errori gestiti

- Inserimento andato a buon fine

Approfondimento funzioni… (1)

- parseInt() – Fa il parsing di una stringa e restituisce un intero
  - var a = parseInt("24.00"); //a = 24
  - var b = parseInt("24.35"); //b = 24
  - var c = parseInt("24 34 6"); //c = 24
  - var d = parseInt(" 24 "); //d = 24
  - var e = parseInt("24 ieri"); //e = 24
  - var f = parseInt("ieri era il 24 "); //f = NaN
  - …

Approfondimento funzioni… (2)

- parseFloat() – Fa il parsing di una stringa e restituisce un float
  - var a = parseFloat("24.00"); //a = 24
  - var b = parseFloat("24.35"); //b = 24.35
  - var c = parseFloat("24 34 6"); //c = 24
  - var d = parseFloat(" 24 "); //d = 24
  - var e = parseFloat("24 ieri"); //e = 24
  - var f = parseFloat("ieri era il 24 "); //f = NaN
  - …

Approfondimento funzioni… (3)

- isNaN(val) - Restituisce true/false in base al parametro in ingresso
  - isNaN(0) //false
  - isNaN('stringa') //false
  - isNaN('12/04/2016') //true
  - isNaN('true') //false
  - isNaN('NaN') //true
  - isNaN(undefined) //true
  - isNaN(NaN) //true
- isFinite() – Controlla se un numero è finito o meno
  - var isFinite(24-3) //true
  - var isFinite('stringa') || isFinite('12/04/2016') //false

Approfondimento funzioni… (4)

- escape() – Effettua la codifica delle stringhe (codifica esadecimale di ogni carattere preceduta dal %), di tutti i caratteri tranne: * @ - _ + . /
  - var str = escape("Questa è una stringa!");
  - //str = ‘Questa%20%E8%20una%20stringa%21’;

- unescape() - Esegue il procedimento contrario rispetto alla escape(). Converte le codifiche esadecimali nei corrispondenti caratteri ASCII

Approfondimento funzioni… (5)

- encodeURI() – Effettua la codifica di un URI. Caratteri NON codificati: , / ? : @ & = + $ #
- decodeURI() – Esegue il procedimento contrario rispetto alla encodeUri()
- encodeURIComponent() - Effettua la codifica di un URI (usato per la codifica dei parametri)
- decodeURIComponent() – Effettua il procedimento inverso rispetto alla encodeUriComponent()

Approfondimento funzioni… (6)

- indexOf(stringa) – Rende l'indice della stringa in ingresso
  - var stringa = 'scrivo una stringa';
  - var indice = indexOf('stringa') //index = 11;

- replace() – Sostituisce una stringa con un'altra
  - var stringa = 'scrivo una stringa';
  - var res = str.replace("stringa", "riga di codice");
  - // res= 'scrivo una riga di codice';

Approfondimento funzioni… (7)

- substr(c1,c2) – Estrae una sottostringa da una stringa dal carattere c1 (compreso) al carattere c2 (compreso)
  - var str = "Buongiorno!";
  - var res = str.substr(1, 4); //res = ‘uong’;
- substring() – Estrae una sottostringa da una stringa dal carattere c1 (compreso) al carattere c2 (NON compreso)
  - var str = "Buongiorno!";
  - var res = str.substr(1, 4); //res = ‘uon’;

Approfondimento funzioni… (8)

- toLowerCase(str) – Converte la stringa str in caratteri minuscoli
- toUppercase() - Converte la stringa str in caratteri maiuscoli
- trim() – Toglie gli spazi vuoti all'inizio e alla fine di una stringa
  - var email = trim(‘ nome@google.it ’);
  - //email = ‘nome@google.it’
- startsWith(str) / endsWith – restituisce true o false se una stringa inizia/finisce o meno con str
  - var stringa = ‘Benvenuto sul nostro portale!’;
  - var inizia = stringa.startWith(‘Benvenuto’); //init = true;

Espressioni Regolari (1)

- JavaScript fornisce un supporto nativo per le **espressioni regolari:** si basa sull'oggetto **RegExp**.
- var x = new RegExp("[a-z]");
- Esistono proprietà e metodi predefiniti che consentono di gestire testi, individuare e/o sostituire stringhe all'interno di altre, etc.:
  - exec(e) – Restituisce l'espressione regolare cercata nella stringa SE la trova
    - var str = 'Benvenuto sul nostro portale!';
    - var pattern = new RegExp("en");
    - var res = pattern.exec(str); //res = 'en'

Espressioni Regolari (2)

- split(‘c’) – Fa la trasposizione della stringa in un array, a partire dal carattere di separazione c
  - var str = ‘Benvenuto sul nostro portale!’;
  - var array = str.split(‘ ’);
  - //array = [‘Benvenuto, ‘sul’, ‘nostro’, ‘portale’]
- match(e) – Ricerca una sottostringa, espressa tramite una espressione regolare e, dentro ad una stringa
  - var str = ‘Ciao! Ci fa piacere … ’;
  - var substr = str.match(/Ci/g);
  - //substr = [Ci,Ci]

Serializzare gli oggetti in JavaScript

- Si parla di:
  - Serializzazione: processo di trasformazione di un oggetto/dato/informazione/… in un formato facilmente memorizzabile e/o trasmissibile
  - Deserializzazione: processo inverso

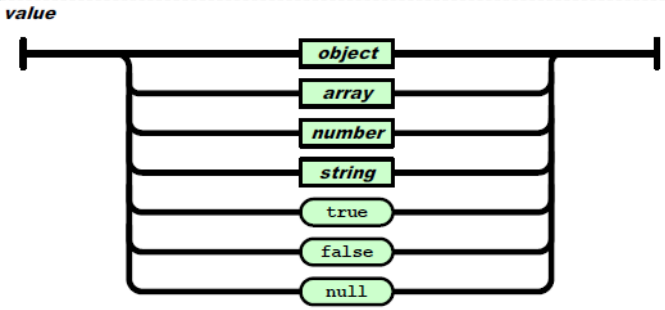
- La serializzazione in javascript avviene attraverso la la rappresentazione JSON, Javascript Object Notation

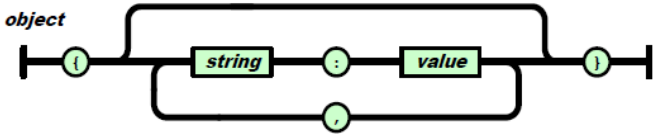
JSON (Javascript Object Notation)

- Nasce per memorizzare dati, trasferire informazioni, rappresentare i dati in maniera da poterli trasferire tra programmi anche diversi
- Nasce dalla modalità di rappresentazioni degli oggetti in javascript
- E' una alternativa a XML per la trasimissione delle informazioni
- E' diventato uno standard: http://www.json.org
  - RFC: https://tools.ietf.org/html/draft-zyp-json-schema-03
- ESEMPIO :
  - **Javascript (oggetto)**: {nome: "Mario", cognome: "Rossi "}
  - **JSON (stringa di car. UNICODE)**: '{nome: "Mario", cognome: "Rossi"}'
  - **XML:** <nome>Mario</nome><cognome>Rossi</cognome>

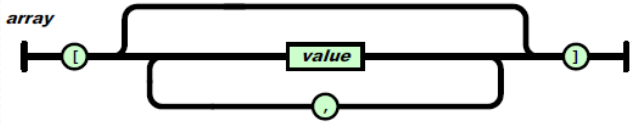
JSON – Sintassi (cenni)

- JSON values:
  - {*object*, *array*, *number*, *string*, true, false,null}
- Oggetto:
  - Racchiuso tra graffe
  - Contiene una serie di coppie ***nome: valore*** separate da virgola:
    - Il ***nome*** è un stringa
    - Il ***valore*** puo' essere una stringa o a sua volta un oggetto
- Array:
  - Racchiuso tra quadre
  - Contiene una collezione di elementi separati da virgola






```
{
   nome: "Mario",
   cognome: "Rossi"
}
```




```
"phoneNumber":
[
    {
      "type": "home",
      "number": "055 111111"
    },
    {
      "type": "fax",
      "number": "055 111111"
    }
]
```

Esempio JSON

JSON

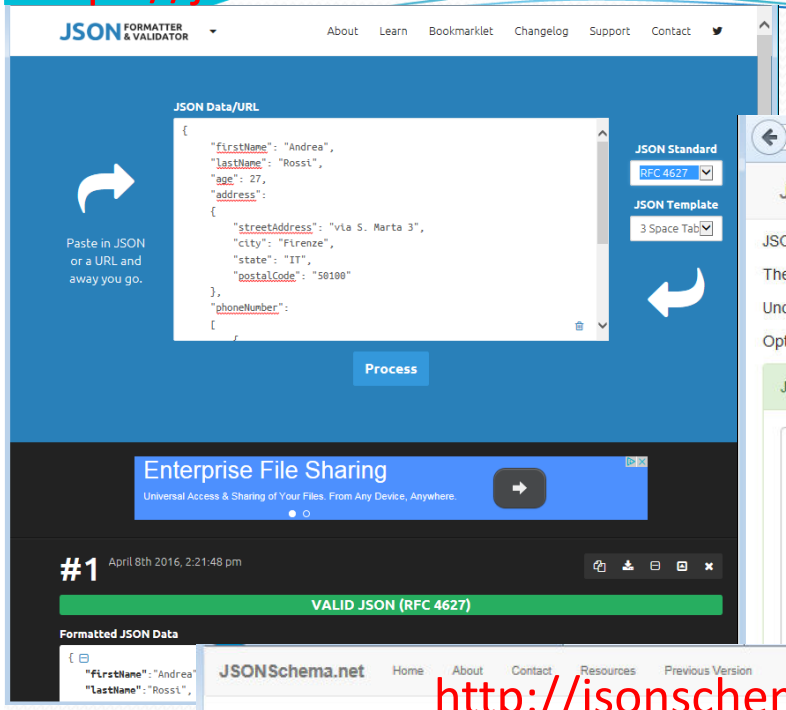
```
{
    "firstName": "Andrea",
    "lastName": "Rossi",
    "age": 27,
    "address":
    {
        "streetAddress": "via S. Marta 3",
        "city": "Firenze",
        "state": "IT",
        "postalCode": "50100"
    },
    "phoneNumber":
    [
        {
          "type": "home",
          "number": "055 111111"
        },
        {
          "type": "fax",
          "number": "055 111111"
        }
    ]
}
```

JSON Schema

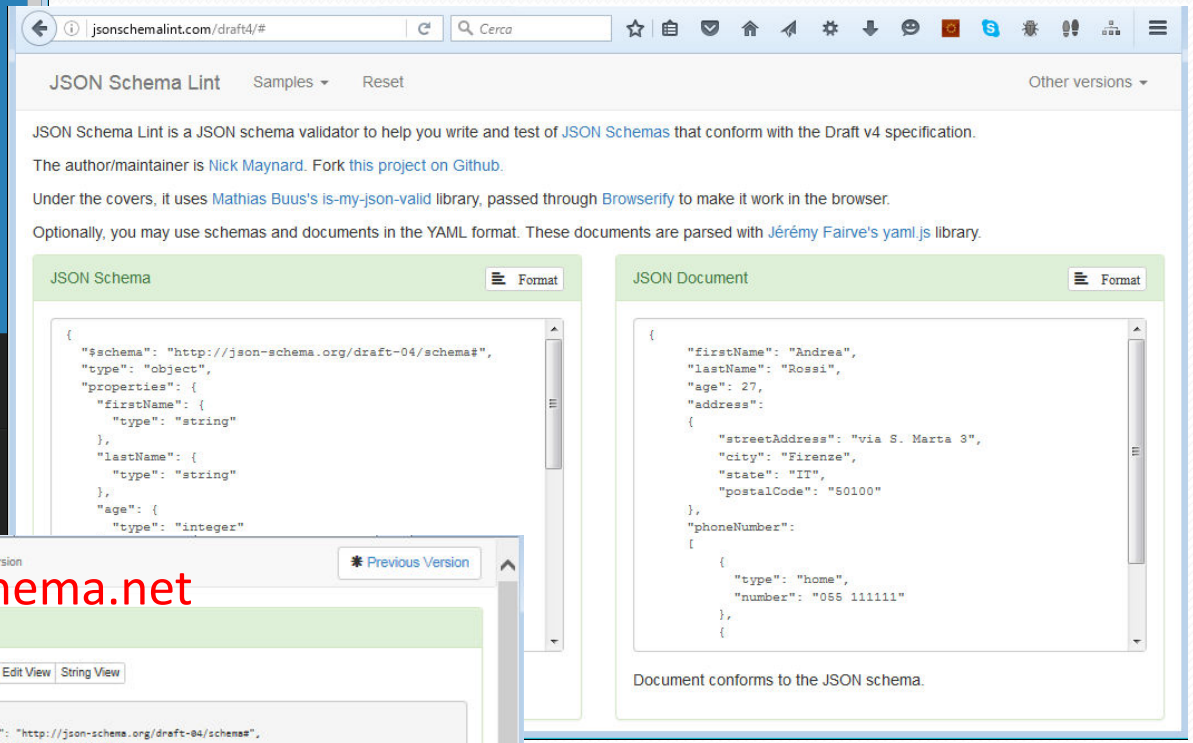
```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string"
    },
    "lastName": {
      "type": "string"
    },
    "age": {
      "type": "integer"
    },
    "address": {
      "type": "object",
      "properties": {
        "streetAddress": {
          "type": "string"
        },
        "city": {
          "type": "string"
        },
        "state": {
          "type": "string"
        },
        "postalCode": {
          "type": "string"
        }
      }
    }
  },
  "required": [
    "firstName",
    "lastName",
    "age",
    "address"
  ]
}
```

JSON Validators

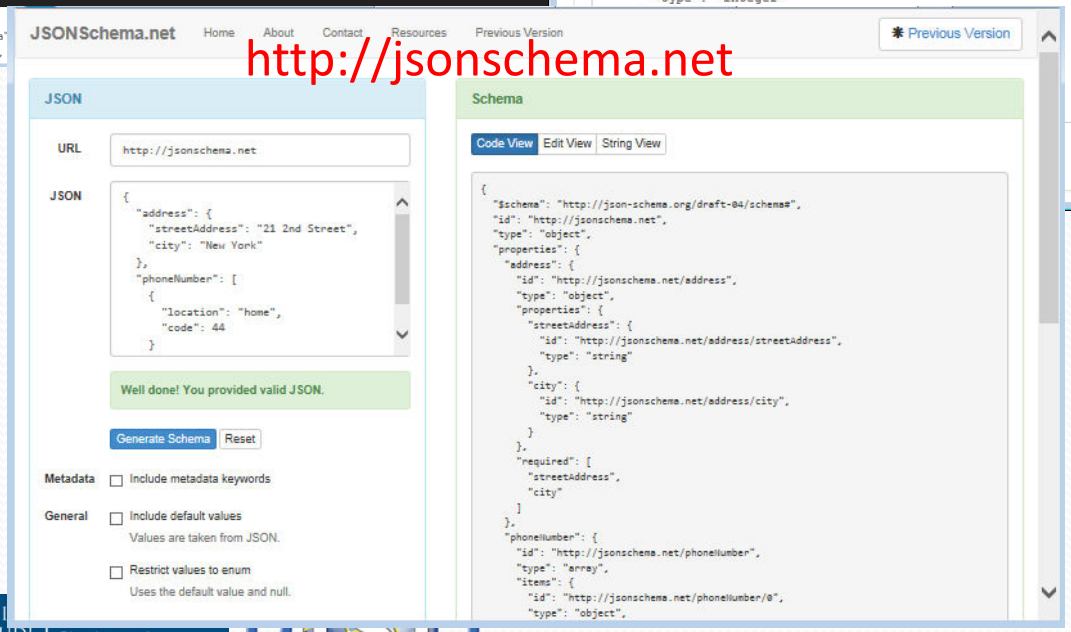
https://jsonformatter.curiousconcept.com



http://jsonschemalint.com/draft4/



http://jsonschema.net



Alcuni esempi:

- Verifica buona formazione JSON
- Validazione schema – istanza JSON
- Schema generator

Gestire i JSON

- parse() – Prende in ingresso una stringa e genera il corrispondente JSON
  - **var** andreaRossi = JSON.parse('{nome: "Andrea", cognome: "Rossi"}');
- stringify() – Genera la rappresentazione JSON dell'oggetto passato come argomento
  - **var** jsonMarioRossi = JSON.stringify({nome: "Andrea", cognome: "Rossi"});

Riferimenti / Approfondimenti

- http://www.ecma-international.org
- https://www.w3.org/standards/webdesign/script
- JSON, http://www.json.org
- JQuery, http://jquery.com