ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercitazione su Javascript

F. Sovrano

Corsi di laurea in Informatica e Informatica per il Management
Alma Mater – Università di Bologna

Oggi vedremo…

- Un esempio di esercizio d'esame su Javascript.
- Alcuni Esercizi di Base
- Alcuni Esercizi Intermedi
- Veri Esercizi d'Esame dell'Anno Scorso
- Esercizi più Complessi

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Javascript: Un Esempio di Esercizio d'Esame degli Anni Passati

Esempio di Esercizio d'Esame

Implementare una funzione javascript. Nel farlo si ponga particolare enfasi nella gestione dei casi particolari, infatti le funzioni devono poter essere eseguite su qualunque pagina HTML senza fare assunzioni sul contenuto della pagina (a parte quelle esplicitate nel testo dell'esercizio, nel caso siano esplicitate).

- La funzione inserisca il contenuto di tutti gli elementi di classe «C1» e «C2» all'interno di un array associativo array_1 avente come chiavi id univoci a propria scelta.
- Successivamente si crei un altro array associativo array_2 in modo che abbia gli stessi valori di array_1 e anche le stesse chiavi però terminanti con la stringa «placeholder». Infine, la funzione ritorni sia array_1 e array_2.

Esempio di Esercizio d'Esame

```
function funzione1(){
   var c1elements = document.getElementsByClassName("C1");
   var c2elements = document.getElementsByClassName("C2");
   var array_1 = [];
   var array_2 = [];
   var array_index = 0;
   for(var i = 0; i < c1elements.length; i++){
      var chiave = 'key_'+ array_index;
      array_1[chiave] = c1elements[i].innerHTML;
      array_index++;
   }
   for(var i = 0; i < c2elements.length; i++){
      var chiave = 'key_'+ array_index;
      array_1[chiave] = c2elements[i].innerHTML;
      array_index++;
   }
   for (var key in array_1) {
      var chiave_new = key + 'placeholder';
      array_2[chiave_new] = array_1[key];
   }
   return [array_1, array_2];
}
```

Soluzione proposta da uno studente

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Javascript: Alcuni Esercizi di Base

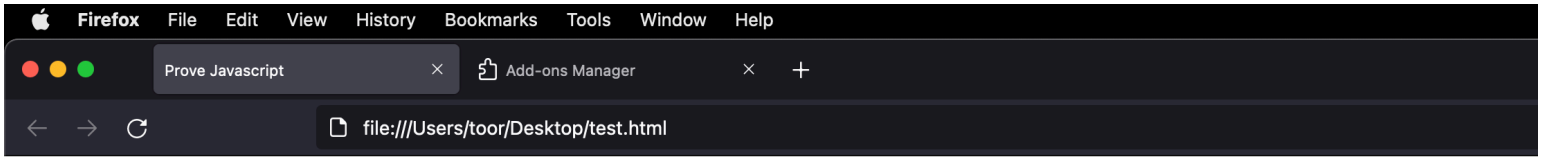
Esercizi

Usare il seguente HTML per i primi 5 esercizi (che, si noti, al momento restituisce un errore JavaScript):

```
<!DOCTYPE html> <html>

<head>
<title>Prove Javascript</title> <script type="text/javascript">

</script> </head>

<body>
<h1>Prove Javascript</h1>
<p id="para1">Un paragrafo di prova.</p>

<button type="button" onclick="onclick_fn()">Clicca qui</button> </body>

</html>
```

Dove vediamo gli errori??

Esercizi - Visualizzazione Errori

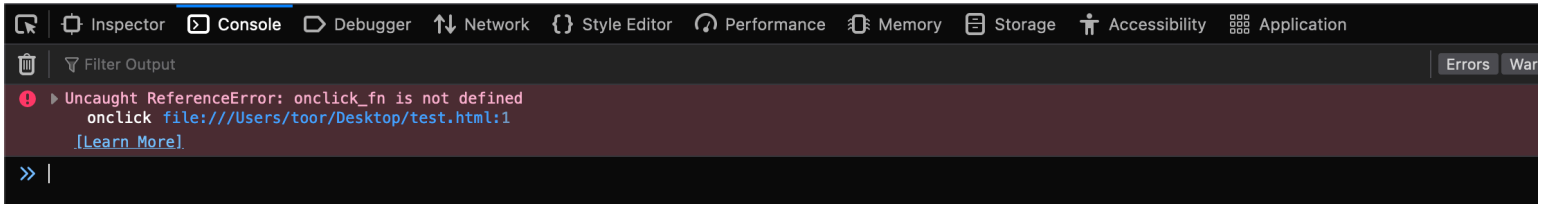


Prove Javascript

Un paragrafo di prova.

Clicca qui

Click Destro -> Inspect -> Console



Esercizio 1

Utilizzando il documento HTML specificato prima, e senza usare jQuery, fare in modo che, facendo click sul pulsante “Clicca qui”, compaia una finestra di alert con scritto “Hello World”.

Esercizio 1

```
onclick_fn = x=>alert('hello world');
```

Soluzione

Esercizio 2

Utilizzando il documento HTML specificato sopra, e senza usare jQuery, fare in modo che, facendo click sul pulsante “Clicca qui”, si prenda dal DOM del documento HTML tutto il contenuto del contenitore taggato “head” e lo si mostri all’interno del paragrafo con id “para1”.

Esercizio 2

```
var onclick_fn = function() {
  var inner_html = document.getElementsByTagName("head")
[0].innerHTML;
  var txt = document.getElementById("para1").textContent;
  txt = inner_html;
  var para1 = document.getElementById("para1");
  para1.textContent = inner_html;
};
```

Soluzione

Esercizio 3

Utilizzando il documento HTML specificato sopra, si faccia in modo che, facendo click sul pulsante “Clicca qui”:

1. Si renda non cliccabile il pulsante “Clicca qui”.
2. Si crei un timer che, nel paragrafo ”para1”, ogni secondo per non più di 10 secondi, stampi il numero di secondi passati dall’attivazione del timer.

Al termine del timer si renda nuovamente cliccabile il pulsante “Clicca qui”.

Esercizio 3

```
var onclick_fn = function() {
   var button = document.getElementsByTagName("button")[0];
   var para = document.getElementById("para1");

   button.disabled = true;
   var max_sec = sec = 10;
   para.textContent = 0;
   var timer = setInterval(function(){
      para.textContent = max_sec - (--sec);
      if (sec <= 0)
      {
         clearInterval(timer);
         button.disabled = false;
      }
   }, 1000);
};
```

Soluzione

Esercizio 4

Si salvi sul proprio spazio web il seguente documento JSON, come file:

```
[
	{
		"Name": "John",
		"Age": 40,
		"Height": 160
	}, {
		"Name": "Jane",
		"Age": 19,
		"Height": 170
	}, {
		"Name": "Judy",
		"Age": 38,
		"Height": 162
	}, {
		"Name": "Jessica",
		"Age": 43,
		"Height": 157
	}, {
		"Name": "Joe",
		"Age": 25,
		"Height": 181
	}
]
```

Utilizzando il documento HTML specificato alla prima pagina, si usi il meccanismo delle **promesse** per caricare la propria copia del file JSON e se ne visualizzi il contenuto sotto forma tabellare nel div del documento HTML (si può usare Bootstrap).

Facoltativo: si utilizzi un widget di data grid come http://www.jquery- bootgrid.com/ per ottenere una visualizzazione sofisticata della tabella con filtri, ordinamenti e flessibilità nella strutturazione delle colonne.

Esercizio 4

```
var get_request = (path) => {
   return new Promise((success, error) => {
      var xhr = new XMLHttpRequest();
      xhr.onreadystatechange = function()
      {
         if (xhr.readyState === XMLHttpRequest.DONE) {
            if (xhr.status === 200) {
               success(JSON.parse(xhr.responseText));
            } else {
               error(xhr);
            }
         }
      };
      xhr.open("GET", path, true);
      xhr.send();
   });
};
```

Soluzione - P1

Esercizio 4

```
function tableCreate(val) {
    var body = document.getElementsByTagName('body')[0];
    var tbl = document.createElement('table');
    tbl.style.width = '100%';
    tbl.setAttribute('border', '1');
    var tbdy = document.createElement('tbody');

    // head
    var tr = document.createElement('tr');
    for (var cell of Object.keys(val[0])) {
        var td = document.createElement('th');
        td.appendChild(document.createTextNode(cell))
        tr.appendChild(td)
    }
    tbdy.appendChild(tr);
```

Soluzione - P2

Esercizio 4

```
    for (var row of val) {
        var tr = document.createElement('tr');
        for (var cell of Object.values(row)) {
                var td = document.createElement('td');
                td.appendChild(document.createTextNode(cell))
                tr.appendChild(td)
        }
        tbdy.appendChild(tr);
    }
    tbl.appendChild(tbdy);
    body.appendChild(tbl)
}

var onclick_fn = () => {
    const promise = get_request('http://localhost:8080/test.json');
    promise.then( val => {
            console.log("asynchronous logging has val:",val);
            tableCreate(val);
    });
};
```

Soluzione - P3

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Javascript: Altri Esercizi

Esercizio A - Vista con CSS

Lista Eventi a Bologna

Grande Parata - via Pyongyang

1 Maggio 2100 - 21:00

Musica in festa - Piazza Maggiore

2 Ottobre 1700 - 20:30

Warhol&Friends - Palazzo Albergati

29 Settembre 2019 - 06:00

What The Fish - Via Da Qui

29 Settembre 2019 - 06:00

Esercizio A - API

- Definite API REST per i seguenti servizi:
  - **LoadEvents**: che restituisca l'elenco degli eventi in programma.
  - **SelectEvent**: che restituisca i dati del singolo evento registrati nel database: la data dell'evento, la città in cui si svolge l'evento, il numero partecipanti, il nome partecipanti, l'immagine dei partecipanti.

Esercizio A - Javascript (1/2)

- Scrivere:
  - Uno o più script Javascript che, subito prima di visualizzare la pagina all'utente, invochino il servizio LoadEvents e popolino correttamente e nel giusto ordine la lista degli eventi.

Esercizio A - Soluzione (1/2)

```
window.onload = function(){
     $.ajax({
          url: API_SERVER_URL+'/events',
          // async: false,
          method: 'GET',
          success: function(event_list) {
               for (var i in event_list)
               {
                    var event = event_list[i];
                    $(`#event-${i}`).html(
                         `<div class="item-events">
                         <img src="${event.image_url}">
                      </div>

                      <div class="tab-events">
                         <div class="upper-tab-event">
                             <p>${event.title}</p>
                         </div>
                         <div class="lower-tab-event">
                             <p>${event.date}</p>
                         </div>
                      </div>`
                    );
               }
               add_menu();
          },
     });
};
```

(Usiamo AJAX)

Esercizio A - Javascript (2/2)

- Scrivere:
  - Un metodo Javascript che viene invocato dopo il caricamento della lista degli eventi e aggiunge un menù a tendina (prima della lista). Il menù mostra quattro opzioni: “terminati” “live” “futuri” “città”.

Esercizio A - Soluzione (2/2)

```
function add_menu() {
      document.getElementById('event-0').insertAdjacentHTML('
      beforebegin',
          `<select id="menu" name="menu">
        <option value="" selected>Please choose</option>
        <option value="terminati">terminati</option>
        <option value="live">live</option>
        <option value="futuri">futuri</option>
        <option value="città">città</option>
    </select>`);
}
```

(***)

*** In JQuery si scrive: $(`#event-0`).before(…)

Esercizio B

La struttura dell'esercizio JS del compito è sempre la stessa

- Viene fornita (o suggerita implicitamente) una pagina HTML con dei widget rilevanti
- Viene descritta l'API di un servizio server-side con URI, metodi HTTP e strutture XML o JSON più o meno complesse.
- In alternativa, viene richiesto di descrivere un'API di tali servizi descritti a parole
- Bisogna associare agli eventi di alcuni widget la interrogazione asincrona ai servizi.
- Bisogna sempre gestire il caso di successo della comunicazione (che tipicamente va a riempire dei div della pagina HTML). Bisogna spesso gestire il caso d'errore (tipicamente un alert informativo)
- Spesso bisogna arricchire altri widget con eventi associati ad altre chiamate AJAX.

Esercizio B (1/4)

- Nel sistema informativo di una mensa scolastica si vuole sperimentare un sistema che permetta ai singoli scolari di scegliere il proprio menù del pranzo.
- Alla mattina lo scolaro o un genitore accede al sistema informativo, inserisce le proprie credenziali e si trova di fronte ad un'interfaccia che gli permette di scegliere tra i vari piatti offerti quel giorno.
- Ogni menù è composto da primo, secondo con contorno e dessert (frutta o dolce).
- Il menù è interrogabile tramite metodo GET all'indirizzo http://www.menuscolastico.it/mandamenu.php?date=[XXX], dove [XXX] è una data in formato YYYYMMDD.

Esercizio B (2/4)

Data la molteplicità di etnie presenti nelle scuole pubbliche italiane e l'incremento di allergie e intolleranze alimentari nei bambini, alcuni cibi debbono essere esclusi perché non adatti alla cultura, alla religione o alla salute del singolo studente.

Poiché non si può richiedere che sia il bambino stesso ad evitare di scegliere i cibi inadatti (potrebbe ignorare i dettami della propria dieta o volerli sfidare), è necessario che il sistema escluda automaticamente i cibi inadatti.

In caso di successo dell'operazione di login, dunque, il sistema fornisce un oggetto JSON (non modificabile, stabilito in fase di registrazione dello studente) come il seguente:

```
{
        nome: 'Andrea',
        cognome: 'Rossi',
        classe: '3',
        sezione: 'A',
        restrizioni: ['vegetariano','celiaco']
}
```

Esercizio B (3/4)

Dopo tale richiesta il sistema spedisce un JSON come segue:

```
{
        "giorno": "2018-05-15",
        "menu": {
                "primi": [{
                        "nome": "Tagliatelle al ragù",
                        "ingr": ["pasta all'uovo", "manzo", "salsa pomodoro",
"spezie"]
                  },{
                        "nome": "Pasta al pomodoro",
                        "ingr": ["pasta secca", "salsa pomodoro", "spezie"]
                  }, ... ],
                "secondi": [{
                        "nome": "Scaloppina ai funghi",
                        "ingr": ["manzo", "funghi secchi", "besciamella"]
                  },{
                        "nome": "Stracchino e grissini",
                        "ingr": ["formaggio fresco", "grissini"]
                  }, ... ],
                "dessert": [{
                        "nome": "fiordilatte",
                        "ingr": ["latte", "zucchero", "aromi naturali"]
                },{
                        "nome": "Kiwi",
                        "ingredienti": ["kiwi"]
                }, ... ]
        }
}
```

Esercizio B (4/4)

Si assuma che esista già fornita una funzione **`puòMangiare(R,I)`**, che restituisce True o False a seconda che una persona con la restrizione R possa consumare o meno l'ingrediente I, per ogni compatibilità possibile, per ogni tipo di restrizione ed ogni ingrediente. Scrivere:

a) La funzione **`chiediMenu(D)`** che richiede il menu della data D (YYYYMMDD)
b) La funzione **`restringiMenu(AR,M)`**, che restituisce in formato JSON il menù risultante dall'eliminazione dei cibi che contengono ingredienti incompatibili con le restrizioni, dove AR è un array di restrizioni e M è un menu in JSON.
c) La funzione **`mostraMenu(M)`**, che va a creare un form per visualizzare i dati del menu M già ristretto come segue: per ogni categoria (primo, secondo, dessert): visualizza un menu a tendina con un elemento per ogni cibo. Di ogni cibo viene mostrato il nome e tra parentesi gli ingredienti.
d) La funzione **`spedisciMenu()`**, che si collega con collegamento Ajax e metodo POST all'indirizzo **`http://www.menuscolastico.it/mandamenu`** spedendo i valori di tutti i campi del form, e mostrando in caso di successo il messaggio "Indicazioni del menù ricevute correttamente", e in caso di insuccesso il messaggio "C'è stato un errore nella spedizione delle indicazioni di menù. Per cortesia riprova."

Esercizio B - Soluzione

- La funzione **chiediMenu(D)** è un wrapper di chiamata Ajax. In caso di successo bisogna popolare il form. Quindi la funzione successo deve prima restringere il menu e poi mostrare le scelte. Non si parla di insuccesso.
- La funzione **restringiMenu(AR,M)** è un confronto tra array. Bisogna pensare ad un algoritmo.
- La funzione **mostraMenu(M)** è un modificatore della pagina visualizzata. Bisogna selezionare l'elemento giusto, e creare del DOM al suo interno per tre stringhe o tre menu a tendina con relativa etichetta più il pulsante di submit.
- La funzione **spedisciMenu()** è un wrapper di chiamata Ajax associata al click del pulsante di submit. Attenzione che è un POST, e non un GET.

Esercizio B - Soluzione (1/5)

Assumo che il form abbia **id="formMenu"**. Ho due variabili globali:

```
var url = "http://www.menuscolastico.it/mandamenu.php
var utente = {
        nome: 'Andrea',
        cognome: 'Rossi',
        classe: '3',
        sezione: 'A',
        restrizioni: ['vegetariano','celiaco']
}
```

Esercizio B - Soluzione (2/5)

Domanda a)

(Usiamo javascript nativo)

```
function httpGet(theUrl)
{
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open( "GET", theUrl, false ); // false for synchronous request
    xmlHttp.send( null );
    return xmlHttp.responseText;
}
```

```
function chiediMenu(D) {
    var datiStringa = httpGet(url + “?date=“+D.format(“YYYYMMDD")); (*)
    var dati = JSON.parse(datiStringa);
    var menuRistretto = restringMenu(utente.restrizioni, dati.menu);
    mostraMenu(menuRistretto);
}
```

(*): E' accettabile descrivere a parole Date.format(formato).

Esercizio B - Soluzione (3/5)

Domanda b)

```
function restringiMenu(restrizioni,menu) {
  for (var i in menu) {              //cioè primo, secondo, dessert
    var categoria = menu[i]
    for (var j=0; j<categoria.length; j++) {
      var ricetta = categoria[j]
      for (var k=0; k<ricetta.ingredienti.length; k++) {
        var ingrediente = ricetta.ingredienti[k]
        for (var l=0; l<restrizioni.length; l++) {
          var restrizione = restrizioni[l]
          if (!puoMangiare(restrizione,ingrediente)) {
            remove(categoria,j)        (**)
        }}
    }}}
  return menu
}
```

(**): E' accettabile descrivere a parole remove(array,posizione).

Esercizio B - Soluzione (4/5)

Domanda c)

```
function mostraMenu(menu) {
  var html = ""
  for (var i in menu) { //cioè 'primo', 'secondo', 'dessert'
    html += "<div class='menu'><h2>"+i+"</h2><select>" ;
    for (var j=0; j<menu[i].length; j++) {
      html += "<option>"+menu[i][j].nome ;
      html += " ("+menu[i][j].ingr.join(', ')+")" ;
      html += "</option>"
    }
    html += "</select>"
  }
  html += "</div><button onclick='spedisciMenu()'>Spedisci</
button>"
   document.getElementById("formMenu").innerHTML=html;
}
```

Esercizio B - Soluzione (5/5)

Domanda d)

```
function spedisciMenu() {
  var formData = $(“#formMenu").getAllData()          (***)
  $.ajax({    (Usiamo AJAX)
    url: url,
    method: 'POST',
    data: formData,
    success: function(datiString, status, richiesta){
      alert("Indicazioni del menù ricevute correttamente")
    },
    error: function(err) { alert("C'è stato un errore. Per
cortesia riprova") }
  })
}
```

(***): E' accettabile descrivere a parole getAllData().

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Javascript: Veri Esercizi d'Esame dell'Anno Scorso

Esercizio 1

Qual è il contenuto della variabile c dopo l'esecuzione dello script seguente?

```
function mutate(a) {
    var b = "3";
    for (var i = 1; i < 7; i=i+1) {
        b = b+a[i];
    }
    return b;
}
var c = mutate("0,8,3,5,1,9,7,2,4,6") ;
```

Esercizio 1 - Soluzione

Il contenuto sarà: “3,8,3,5”

Esercizio 2

Implementare una funzione javascript che faccia due cose:

1. Per prima cosa inserisca una select, per 3 tipi diversi di lavatrici, al posto il terzo titolo con classe "elemento_1". Se suddetto terzo titolo non esiste, allora la funzione lo deve creare.
2. Per seconda cosa legga il contenuto HTML del blocco con id "element_2" e lo stampi in tutti i paragrafi con classe "parahtml", controllando anche che "element_2" contenga almeno 4 elementi con tag b.

Esercizio 2.1 - Soluzione

```
~~var titoli = $('h1 .elemento_1’);~~
var titoli = $('h1.elemento_1');
// controllo se ci sono meno di 3 elementi e in tal caso creo tutti quelli che mancano
if (titoli.length < 3) {
   for (i=0; i<titoli.length; i++) {
      if (!titoli[i]) {
         $('body').append('<h1 class = "elemento_1"> Titolo </h1>');
      }
   }
}
~~var titoli = $('h1 .elemento_1’);~~
var titoli = $('h1.elemento_1');
titoli[2].replaceWith(' <select name="lavatrici" id="lavatrici">
                                    <option value="lav1"> Lavatrice tipo 1 </option>
                                    <option value="lav2"> Lavatrice tipo 2 </option>
                                    <option value="lav3"> Lavatrice tipo 3 </option>
                                     </select>');
```

Esercizio 2.2 - Soluzione

```
if ($('#element_2').find('b').length >= 4) {
    var htmlContent  = $(‘#element_2’).html();
    $(‘.parahtml’).html(htmlContent);
    $(‘.parahtml’).each(x=>x.html(htmlContent));
}
```

Javascript: Esercizi Senza Soluzioni

Esercizio I

Creare una classe javascript chiamata Algoritmo. Un algoritmo ha un nome, una complessità asintotica, un corpo (cioè una funzione javascript) e una descrizione degli input e degli output del corpo.

Istanziare 3 diversi oggetti della classe Algoritmo che rappresentino rispettivamente 3 diverse funzioni di tua scelta (eg. la funzione costante che ritorna sempre lo stesso valore, la funzione che ritorna valori casuali, ecc..).

Utilizzando il documento HTML specificato nella prima pagina, si aggiungano due campi numerici e si faccia in modo che, facendo click sul pulsante “Clicca qui”, si apra un pop-up in cui si chieda di scegliere quale dei 3 diversi algoritmi mostrare. Nel paragrafo “para1” si mostrino nome, descrizione, complessità asintotica e codice sorgente dell’algoritmo scelto.

Questo esercizio potrebbe richiedere di studiare cose non viste a lezione.

Esercizio II (1/2)

Definire le seguenti classi: **Algoritmo**, **Goal**, **Codice**, **Risorsa**.

Un **Algoritmo** ha:

- Un nome
- Una descrizione
- Un **Goal**
- Una complessità asintotica
- Un **Codice**
- Una o più **Risorse** in input
- Una o più **Risorse** in output

Questo esercizio potrebbe richiedere di studiare cose non viste a lezione.

Un **Goal** ha:

- Una descrizione
- Una classe di appartenenza (eg. Computer Vision, Natural Language Processing, etc..)
- Una sotto-classe di appartenenza (eg. Text summarization, Object recognition, Text generation, ecc..)

Esercizio II (2/2)

Un **Codice** ha:

- Un sorgente
- Un linguaggio
- Una licenza
- Un creatore

Questo esercizio potrebbe richiedere di studiare cose non viste a lezione.

Una **Risorsa** ha:

- Un formato (eg. PDF, MP4, DOC, ecc..)
- Un contenuto definito da un insieme di file
- Una descrizione
- Una licenza

Successivamente, creare due view separate che supportino:

- di creare un'interfaccia che permetta di creare un nuovo algoritmo.
- di creare un'interfaccia che permetta di visualizzare il codice sorgente di un algoritmo, in base ai seguenti step:

    1. L'utente specifica un tipo di risorsa in input.
    2. L'utente riceve dal sistema l'elenco di algoritmi che accettano il tipo di risorsa specificato. Tale elenco è raggruppato in base a classe e sotto-classe di appartenenza dell'oggetto goal associato all'algoritmo. L'utente può esplorare l'elenco delle soluzioni.
    3. L'utente sceglie un algoritmo. L'utente deve poter visualizzare TUTTE le informazioni sull'algoritmo scelto, incluso il codice sorgente.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Speaker: **Francesco Sovrano**

Dipartimento di Informatica – Scienze e Ingegneria
Alma mater – Università di Bologna

Website: *unibo.it/sitoweb/francesco.sovrano2*
E-mail: *francesco.sovrano2@unibo.it*