

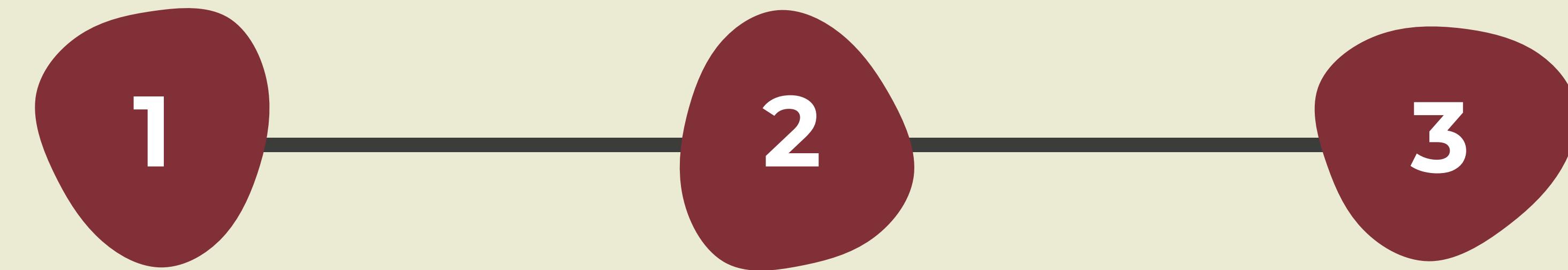
Lab Hours - Lecture 5

# Recap

+ Advanced Javascript

Contact: [ellepuntopi.98@gmail.com](mailto:ellepuntopi.98@gmail.com)

# Programma di oggi:



Correzione esercizio  
per casa

Esercizio d'esame  
insieme:  
HTML+CSS+JS

Esercizio d'esame in due parti:  
HTML+CSS  
HTML+CSS+JS+Ajax

# Esercizio d'esame da svolgere a casa

**Implementare una funzione javascript.** Nel farlo si ponga particolare enfasi alla 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 esplicite nel testo dell'esercizio, nel caso siano esplicite).

- La funzione inserisca il contenuto **testuale** di tutti gli elementi di classe «C1» e «C2» all'interno di un **array associativo (Object)** **array\_1** avente come chiavi id univoci di vostra 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.

# Esercizio d'esame - Correzione (1)

```
function raccogliEManipolaElementi() {  
    // Raccoglie tutti gli elementi con classe "C1" e "C2"  
    var c1elements = document.getElementsByClassName("C1");  
    var c2elements = document.getElementsByClassName("C2");  
  
    // Primo array associativo per conservare i contenuti  
    var array_1 = {};  
    // Secondo array associativo per conservare i contenuti modificati  
    var array_2 = {};
```

Come prima cosa, dopo aver definito la funzione, raccogliamo tutti gli elementi di classe **C1** e **C2**, poi definiamo gli Array Associativi (Objects) da popolare.

# Esercizio d'esame - Correzione (2)

```
// Funzione ausiliaria per popolare l'array associativo
function popolaArray(daElementi, array, indiceBase = 0) {
    for (var i = 0; i < daElementi.length; i++) {
        // Creazione di una chiave univoca per ogni elemento con un indice incrementale basato su indiceBase
        var chiave = 'key_' + (i + indiceBase) + '_';
        // Aggiunta del contenuto dell'elemento all'array associativo
        array[chiave] = daElementi[i].textContent;
    }
}
```

Successivamente andiamo a definire una funzione ausiliaria per popolare gli array associativi. Tale funzione ha lo scopo di definire il nome della chiave (diverso per ogni elemento), e di associare ad esso il valore **testuale** estratto dalla pagina HTML.

# Esercizio d'esame - Correzione (3)

```
// Popola array_1 con gli elementi di C1
popolaArray(c1elements, array_1);

// Calcola l'indice di base per gli elementi C2, che è uguale al numero di elementi C1
var indiceBaseC2 = c1elements.length; // Indice di base per gli elementi di C2
popolaArray(c2elements, array_1, indiceBaseC2);
```

Ora che abbiamo definito la funzione, andiamo ad utilizzarla per popolare gli array associativi. In particolare, iniziamo chiamando la funzione sugli elementi di classe **C1**, per i quali l'indice base è quello di default (0). Successivamente, andiamo a definire l'indice di base per gli elementi di **C2**, che partiranno dall'indice dell'ultimo elemento di **C1**. Per trovare tale indice, è sufficiente calcolare il numero di elementi con classe **C1**, per poi riprendere a contare da lì. Infine, richiamiamo la stessa funzione per **C2**.

# Esercizio d'esame - Correzione (4)

```
// Creazione di array_2 con le stesse chiavi di array_1 ma terminanti in "placeholder"
for (var key in array_1) {
    var chiaveModificata = key + "placeholder";
    array_2[chiaveModificata] = array_1[key];
}

// Ritorna entrambi gli array associativi
return [array_1, array_2];
```

Ora, come ultimo passo, dobbiamo popolare il secondo Array, e aggiungere la stringa “placeholder” alla fine di ogni chiave. Per farlo, basterà semplicemente iterare le chiavi in Array 1, per poi modificarle una ad una, e popolare con gli stessi valori il secondo array.

Infine, la funzione ritorna entrambi gli array associativi.

# Esercizio d'esame - Correzione (5)

JS

```
function raccogliEManipolaElementi() {
    // Raccoglie tutti gli elementi con classe "C1" e "C2"
    var c1elements = document.getElementsByClassName("C1");
    var c2elements = document.getElementsByClassName("C2");

    // Primo array associativo per conservare i contenuti
    var array_1 = {};
    // Secondo array associativo per conservare i contenuti modificati
    var array_2 = {};

    // Funzione ausiliaria per popolare l'array associativo
    function popolaArray(daElementi, array, indiceBase = 0) {
        for (var i = 0; i < daElementi.length; i++) {
            // Creazione di una chiave univoca per ogni elemento con un indice incrementale basato su indiceBase
            var chiave = 'key_' + (i + indiceBase) + '_';
            // Aggiunta del contenuto dell'elemento all'array associativo
            array[chiave] = daElementi[i].textContent;
        }
    }

    // Popola array_1 con gli elementi di C1
    popolaArray(c1elements, array_1);

    // Calcola l'indice di base per gli elementi C2, che è uguale al numero di elementi C1
    var indiceBaseC2 = c1elements.length; // Indice di base per gli elementi di C2
    popolaArray(c2elements, array_1, indiceBaseC2);

    // Creazione di array_2 con le stesse chiavi di array_1 ma terminanti in "placeholder"
    for (var key in array_1) {
        var chiaveModificata = key + "placeholder";
        array_2[chiaveModificata] = array_1[key];
    }

    // Ritorna entrambi gli array associativi
    return [array_1, array_2];
}

// Esempio di come chiamare la funzione e visualizzare il risultato
var risultati = raccogliEManipolaElementi();
console.log("Array 1:", risultati[0]);
console.log("Array 2:", risultati[1]);
```

Andando ora a chiamare la funzione, otterremo però due array vuoti...

Sapete perché?

```
var risultati = raccogliEManipolaElementi();
console.log("Array 1:", risultati[0]);
console.log("Array 2:", risultati[1]);
```



Array 1: - {}

Array 2: - {}

# Esercizio d'esame - Correzione (6)

Questo accade perché la funzione viene chiamata prima ancora che la pagina sia stata caricata. Dunque il DOM, non essendo ancora stato caricato, non contiene gli elementi di Classe C1 e C2!

Per risolvere il problema, ci basterà encapsulare la funzione all'interno di un *Event Listener*, il cui evento è appunto il caricamento del DOM.

```
document.addEventListener('DOMContentLoaded', (event) => { ...  
});
```

Perciò, la funzione, e la relativa chiamata ad essa, avverranno nel momento in cui il contenuto del DOM sarà stato caricato, risultando quindi in un corretto funzionamento.

# Esercizio d'esame: HTML + CSS + JS

JS

Scrivere il codice HTML, CSS e Javascript per realizzare una piccola applicazione che permetta di aggiungere prodotti ad una lista dei desideri, come mostrato di seguito:

## Lista dei desideri

### Prodotti

Prodotto 1 Aggiungi alla lista

Prodotto 2 Aggiungi alla lista

Prodotto 3 Aggiungi alla lista

**La tua lista dei desideri :**

Prodotto 1 : 5  
Prodotto 2 : 4  
Prodotto 3 : 10

# Esercizio d'esame: HTML + CSS + JS (2) - Specifiche

JS

Specifiche:

- L'elenco prodotto è predefinito e include 3 prodotti. Ogni prodotto ha 10 pezzi in magazzino.
- Ogni prodotto ha un bottone corrispondente: "Aggiungi alla lista"
- Al click su questo bottone, il prodotto verrà aggiunto alla lista dei desideri (nella parte inferiore della pagina), che verrà quindi aggiornata e visualizzata.
- Alla lista vengono aggiunti non solo i nomi dei prodotti, ma anche il numero di pezzi che sono stati aggiunti (click cinque volte sul pulsante per aggiungere Prodotto 1 -> nella lista dei desideri verrà visualizzato "Prodotto 1: 5").
- Se un prodotto viene cliccato 10 volte, il pulsante "Aggiungi alla lista" per quel prodotto viene disattivato.
- L'applicazione è completamente lato client e non memorizziamo alcun dato, quindi se l'utente aggiorna la pagina la wishlist verrà svuotata.
- La lista dei desideri deve inoltre essere ordinata in ordine alfabetico. Ogni volta che un prodotto viene aggiunto all'elenco, viene verificato l'ordine alfabetico
- I dettagli di presentazione (dimensioni, font, margini, ecc.) non sono rilevanti
- I nomi dei prodotti possono essere arbitrari
- I bottoni devono diventare più scuri nel momento in cui il mouse va in hover
- Il puntatore del mouse, quando i bottoni sono in hover, dovrà diventare di tipo pointer

# Esercizio d'esame: HTML + CSS + JS - Svolgimento (1)

JS

Partiamo dal definire la struttura di base della pagina HTML.

```
<!DOCTYPE html>
<html>
<head>
    <title>Lista dei Desideri</title>
    <link type="text/css" rel="stylesheet" href="style.css">
    <script type="text/javascript" src="script.js"></script>
</head>
<body>
    <h1> Lista dei desideri </h1>
    <h2> Prodotti </h2>

    <div>
        <ul>
            <li> <span> Prodotto 1 </span> <input type="button" id="btnP1" value="Aggiungi alla lista" > </li>
            <li> <span> Prodotto 2 </span> <input type="button" id="btnP2" value="Aggiungi alla lista" > </li>
            <li> <span> Prodotto 3 </span> <input type="button" id="btnP3" value="Aggiungi alla lista" > </li>
        </ul>
    </div>

    <h2> La tua lista dei desideri : </h2>
    <div>
        <div id="containerProdotti">
            <p style="visibility:hidden" id="container1"> Prodotto 1 : <span id="countP1">0</span> </p>
            <p style="visibility:hidden" id="container2"> Prodotto 2 : <span id="countP2">0</span></p>
            <p style="visibility:hidden" id="container3"> Prodotto 3 : <span id="countP3">0</span> </p>
        </div>
    </div>
</body>
</html>
```

# Esercizio d'esame: HTML + CSS + JS - Svolgimento (2)

JS

```
html, body {  
    max-height: 100vh !important;  
    margin: 0;  
    padding: 0;  
}  
  
body {  
    font-family: 'Arial', sans-serif;  
    background: #f4f4f4;  
    display: flex;  
    flex-direction: column;  
}  
  
h1, h2 {  
    color: #5D5C61;  
    margin: 0 20px;  
}  
  
ul {  
    list-style: none;  
    padding: 0;  
    margin: 0;  
}  
  
li {  
    background: #fff;  
    margin-bottom: 10px;  
    padding: 15px;  
    border-radius: 5px;  
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
}  
  
span {  
    font-size: 18px;  
}  
  
input[type="button"] {  
    padding: 10px 20px;  
    border: none;  
    border-radius: 5px;  
    background-color: #77DD77;  
    color: white;  
    cursor: pointer;  
    font-size: 16px;  
    transition: background-color 0.3s ease;  
}  
  
input[type="button"]:hover {  
    background-color: #5CAB7D;  
}  
  
#containerProdotti {  
    background: #ffffff;  
    padding: 15px;  
    border-radius: 5px;  
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);  
    margin: 20px;  
    overflow: auto;  
}  
  
#containerProdotti p {  
    color: #333;  
    font-size: 16px;  
    margin: 5px 0;  
}  
  
#containerProdotti p span {  
    font-weight: bold;  
}
```

Descriviamo ora lo stile della nostra pagina.

Le cose a cui prestare attenzione non sono troppe. Sarà sufficiente definire un display di tipo flex, in particolare per orientare la disposizione degli elementi; l'hover per scurire il colore dei bottoni; il cursore di tipo pointer.

Per il resto, ho aggiunto padding, margin, box-shadow, ecc... giusto per farvi avere un'ulteriore reference durante lo studio. Però, se tali dettagli non sono presenti nelle specifiche, allora non è necessario aggiungerli.

# Esercizio d'esame: HTML + CSS + JS - Svolgimento (3)

JS

```
1  document.addEventListener('DOMContentLoaded', function() {
2      function addProduct(containerId, countId, buttonId) {
3          var container = document.getElementById(containerId);
4          var countList = document.getElementById(countId);
5          var countProdotto = parseInt(countList.textContent, 10);
6          countProdotto++;
7
8          if (countProdotto === 1) {
9              container.style.visibility = 'visible';
10         }
11         if (countProdotto === 10) {
12             document.getElementById(buttonId).disabled = true;
13         }
14
15         countList.textContent = countProdotto;
16     }
17
18     document.getElementById('btnP1').addEventListener('click', function() {
19         addProduct('container1', 'countP1', 'btnP1');
20     });
21     document.getElementById('btnP2').addEventListener('click', function() {
22         addProduct('container2', 'countP2', 'btnP2');
23     });
24     document.getElementById('btnP3').addEventListener('click', function() {
25         addProduct('container3', 'countP3', 'btnP3');
26     });
27 });

});
```

Come potete vedere, il file di script è concettualmente “diviso in due parti”. Il tutto è inglobato da un **listener**, che attende il DOM sia caricato completamente.

La prima parte contiene la funzione principale, quella che, quando invocata, esegue le seguenti operazioni:

1. Prende elementi da IDs
2. Trasforma il testo che “tiene il conto” in un intero (**parseInt**: prende il testo e lo converte in numero intero, con base 10, come specificato dopo la virgola)
3. Nel momento in cui (primo **if**) il conteggio diventa almeno uguale ad 1, il container diventa visibile
4. Quando il conteggio diventa 10, il bottone viene disabilitato.

# Esercizio d'esame: HTML + CSS + JS - Svolgimento (4)

JS

```
18  document.getElementById('btnP1').addEventListener('click', function() {  
19      addProduct('container1', 'countP1', 'btnP1');  
20  } );  
21  document.getElementById('btnP2').addEventListener('click', function() {  
22      addProduct('container2', 'countP2', 'btnP2');  
23  } );  
24  document.getElementById('btnP3').addEventListener('click', function() {  
25      addProduct('container3', 'countP3', 'btnP3');  
26  } );
```

La seconda parte dello script ha lo scopo di chiamare la funzione definita prima a seguito del click dei bottoni.

Una volta fatto, la funzione si attiva con i giusti input, e il funzionamento dell'applicazione è completo.

# Esercizio d'esame: HTML + CSS + JS + Ajax

## Specifiche Parte 1 (Solo HTML + CSS)

JS

Scrivere il codice HTML e CSS, in un file esterno, per ottenere le visualizzazioni mostrate in Figura 1 e Figura 2 (vedi pagina successiva) quando il documento è caricato in un browser.

In particolare, se lo schermo ha larghezza di almeno 992px il layout è quello in Figura 1 altrimenti in Figura 2.

**Figura 1**

The screenshot shows a mobile application interface. At the top, there is a dark banner with the text "Dettagli degli Eventi" and a subtitle "Dai un'occhiata ai nostri ultimi spettacoli ed eventi per unirti a noi." Below the banner, the title "Elenco Eventi" is displayed. Three event cards are listed:

- Sunny Hill Festival**  
140 Biglietti disponibili  
Sep 16, 2023  
18:00 - 22:00  
Spiaggia di Copacabana, Rio de Janeiro  
[Acquista Biglietti](#)
- Gala Rock Festival**  
128 Biglietti disponibili  
Sep 18, 2022  
9:00 - 12:00  
Spiaggia di Copacabana, Rio de Janeiro  
[Acquista Biglietti](#)
- Hip Hop Farm**  
160 Biglietti disponibili  
Apr 1, 2020  
13:00 - 18:00  
Spiaggia di Copacabana, Rio de Janeiro  
[Acquista Biglietti](#)

**Figura 2**

The screenshot shows a desktop browser window displaying a dark-themed event details page. At the top, there is a dark banner with the text "Dettagli degli Eventi" and a subtitle "Dai un'occhiata ai nostri ultimi spettacoli ed eventi per unirti a noi." Below the banner, the title "Elenco Eventi" is displayed. Two event cards are visible:

- Sunny Hill Festival**  
140 Biglietti disponibili  
Sep 16, 2023  
18:00 - 22:00  
Spiaggia di Copacabana, Rio de Janeiro  
[Acquista Biglietti](#)
- Gala Rock Festival**  
128 Biglietti disponibili  
Sep 18, 2022  
9:00 - 12:00  
Spiaggia di Copacabana, Rio de Janeiro  
[Acquista Biglietti](#)

At the bottom of the page, there is a partial view of another event card for "Hip Hop Farm".

Oltre alle caratteristiche tipografiche già evidenti in figura, si tenga presente che:

- Lo sfondo del banner "Dettagli degli Eventi" è completamente ricoperto da un'immagine,  ed ha un padding di 110px in alto e in basso e di 0px sui lati.
- Nel primo caso (Figura 1), ogni evento occupa un'area equamente divisa in 4 parti per ogni dettaglio dell'evento (titolo, orario, luogo, biglietti)
- nel secondo caso (Figura 2), i dettagli di ogni evento sono uno sopra l'altro e sono tutti centrati
- cliccando sul titolo di ogni evento si apre l'articolo corrispondente; è sufficiente creare un link arbitrario; NON è richiesto creare anche le pagine dei vari articoli
- cliccando su "acquista biglietto" di ogni evento si apre la pagina "ticket-details.html", NON è richiesto creare anche questa pagina
- colori e dimensioni esatte di margini e padding non rilevanti, purché appropriate

Vincoli:

- non è ammesso usare attributi @id nel sorgente HTML;
- non è possibile usare Javascript (eventuali comportamenti dinamici vanno nell'esercizio successivo).
- È possibile utilizzare Bootstrap

**Risorse:** Immagine disponibile su: <http://diiorio.nws.cs.unibo.it/twe/07.06.2022a/>

**Bootstrap:** Collegare al seguente Bootstrap:

<http://diiorio.nws.cs.unibo.it/twe/lib/bootstrap-4.0.0-dist/css/bootstrap.min.css>

# Esercizio d'esame: HTML + CSS + JS +Ajax

## Specifiche Parte 1 - Soluzione (1)

JS

```
<head>
  <title>Esercizio HTML_CSS_JS_AJAX PT1</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" type="text/css" href="style.css">
  <link rel="stylesheet" type="text/css" href="http://diorio.nws.cs.unibo.it/twe/lib/bootstrap-4.0.0-dist/css/bootstrap.min.css">
</head>
```

Come prima cosa, oltre a definire il collegamento al file CSS esterno (che vedremo tra poco), colleghiamo la pagina Bootstrap come richiesto (riquadro rosso).

```
<div class="title">
  <h1>Dettagli degli eventi</h1>
  <p>Dai un'occhiata ai nostri ultimi spettacoli ed eventi per unirti a noi.</p>
</div>
```

Successivamente, definiamo il DIV contenente il titolo e il sottotitolo.

# Esercizio d'esame: HTML + CSS + JS + Ajax

JS

## Specifiche Parte 1 - Soluzione (2)

```
<div class="elenco">
  <h1>Elenco Eventi</h1>
  <div class="evento row">
    <div class="col-md-12 col-lg-3">
      <a href="articolo/"><h4>Sunny Hill Festival</h4></a>
      <p>140 Biglietti disponibili</p>
    </div>
    <div class="col-md-12 col-lg-3">
      <p>Sep 16, 2023</p>
      <p>18:00</p>
      <p>--</p>
      <p>22:00</p>
    </div>
    <div class="col-md-12 col-lg-3">
      <p>Spiaggia Copacabana, Rio de Janeiro</p>
    </div>
    <div class="col-md-12 col-lg-3">
      <a href="ticket-detail.com" class="btn btn-primary">Acquista biglietti</a>
    </div>
  </div>
  <div class="evento row">
    <div class="col-md-12 col-lg-3">
      <a href="articolo/"><h4>Sunny Hill Festival</h4></a>
      <p>140 Biglietti disponibili</p>
    </div>
    <div class="col-md-12 col-lg-3">
      <p>Sep 16, 2023</p>
      <p>18:00</p>
      <p>--</p>
      <p>22:00</p>
    </div>
    <div class="col-md-12 col-lg-3">
      <p>Spiaggia Copacabana, Rio de Janeiro</p>
    </div>
    <div class="col-md-12 col-lg-3">
      <a href="ticket-detail.com" class="btn btn-primary">Acquista biglietti</a>
    </div>
  </div>
```

Di seguito, definiamo un DIV con classe “elenco”, utile a contenere i nostri eventi.

Ogni singolo evento (quadrato verde) sarà definito allo stesso modo, ciò che cambia è solo l'informazione testuale contenuta al suo interno.

Quei **col-md-12**, **col-lg-3**, ecc. sono classi di utilità fornite da **Bootstrap**, un framework di front-end molto popolare per lo sviluppo web. Queste classi sono utilizzate per creare un layout responsive attraverso il sistema di griglia di Bootstrap.

A seguito, ogni evento verrà appunto definito allo stesso modo.

# Esercizio d'esame: HTML + CSS + JS + Ajax

JS

## Specifiche Parte 1 - Soluzione (3)

```
body {  
    margin: 0px;  
}  
  
.title{  
    background-image: url('http://diiorio.nws.cs.unibo.it/twe/07.06.2022a/shows_events_bg.jpg');  
    background-size: cover;  
    width: 100vw;  
    height: auto;  
    padding: 100px 0px;  
}  
  
.title h1, .title p {  
    text-align: center;  
    color: white;  
}  
  
.evento.row{  
    background-color: white;  
    padding: 30px;  
    margin: 20px 40px;  
}  
  
h2, h4, p {  
    text-align: center;  
    font-family: "Courier";  
}
```

```
h4 {  
    color: black !important;  
}  
  
.elenco h1 {  
    text-align: center;  
    padding: 30px 0px 50px 0px;  
}  
  
.elenco {  
    background-color: lightgray;  
    height: 100%;  
    padding-bottom: 100px;  
}  
  
.col-lg-1 {  
    text-align: center;  
}  
  
.btn.btn-primary {  
    background-color: black;  
    border-color: black;  
    font-family: "Courier";  
    color: white  
}  
  
.col-md-12.col-lg-3 a {  
    text-decoration: none;  
    color: white;  
}
```

# Esercizio d'esame: HTML + CSS + JS + Ajax

JS

## Specifiche Parte 2 (Ajax)

Data la pagina HTML dell'esercizio precedente, in cui il DIV principale con l'elenco degli eventi è inizialmente vuoto, si realizzino alcuni comportamenti dinamici in Javascript.

Vincoli:

- è ammesso l'uso di jQuery
- è ammesso usare attributi @id e @class nel sorgente HTML
- è ammesso (e consigliato!) usare CSS dell'esercizio precedente

Nota:

- Scrivere il codice HTML in un file indexPt2.html diverso dal precedente, e il JS in un file esterno script.js

```
[  
 {  
 "id":1,  
 "title":"Sunny Hill Festival",  
 "n_biglietti":"140",  
 "time":"Sep 16 2023",  
 "ora_min":"18",  
 "ora_max":"22",  
 "place":"Spiaggia di Copacabana, Rio de Janeiro",  
 "evidenza":false  
 },  
 {  
 "id":2,  
 "title":"Festa dei Cani e dei Gatti",  
 "n_biglietti":"330",  
 "time":"Nov 3 2023",  
 "ora_min":"9",  
 "ora_max":"22",  
 "place":"Boston, Stati Uniti",  
 "evidenza":true  
 },  
 {  
 "id":3,  
 "title":"Festa della spiaggia",  
 "n_biglietti":"10",  
 "time":"Dic 12 2020",  
 "ora_min":"16",  
 "ora_max":"18",  
 "place":"Trani, Italia",  
 "evidenza":true  
 },  
 {  
 "id":4,  
 "title":"Festival Rock di Gala",  
 "n_biglietti":"128",  
 "time":"18 settembre 2021",  
 "ora_min":"18",  
 "ora_max":"22",  
 "place":"Spiaggia di Copacabana, Rio de Janeiro",  
 "evidenza":false  
 },  
 {  
 "id":5,  
 "title":"Hip Hop Farm",  
 "n_biglietti":"160",  
 "time":"Apr 1 2020",  
 "ora_min":"13",  
 "ora_max":"18",  
 "place":"Spiaggia di Copacabana, Rio de Janeiro",  
 "evidenza":true  
 }]
```

In particolare si realizzi:

- Al caricamento, la pagina accede asincronamente in GET al servizio web [htto://diforio.nws.cs.unibo.it/twe/07.06.2022a/abii/index.php](http://diforio.nws.cs.unibo.it/twe/07.06.2022a/abii/index.php), ottenendo un JSON (come quello a fianco) con i dati relativi alle notizie e li visualizza in modo appropriato.

Note sulla visualizzazione:

- riprodurre la visualizzazione dell'esercizio precedente;
- gli orari degli eventi sono mostrati anche con i minuti (es. 18:00), anche se non lo sono nel file JSON (es. 18).
- il link delle pagine relative ad ogni evento sono calcolati dall'ID dell'evento
- il file JSON non contiene solo gli eventi che dovrebbero essere mostrati ma anche altri. Vanno mostrati solo gli eventi con proprietà "evidenza" uguale a true.
- i dati JSON di esempio includono alcuni eventi ma il codice deve funzionare anche per un numero diverso di eventi;

# Esercizio d'esame: HTML + CSS + JS +Ajax

## Specifiche Parte 2 - Soluzione (1)

JS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Esercizio HTML_CSS_JS_AJAX PT2</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="style.css">
    <link rel="stylesheet" type="text/css" href="http://diorio.nws.cs.unibo.it/twe/lib/bootstrap-4.0.0-dist/css/bootstrap.min.css">
    <script src="script.js"></script>
  </head>
  <body onload="getEventi()">
    <div class="title">
      <h1>Dettagli degli eventi</h1>
      <p>Dai un'occhiata ai nostri ultimi spettacoli ed eventi per unirti a noi.</p>
    </div>
    <div class="elenco">
      <h1>Elenco Eventi</h1>
    </div>
  </body>
</html>
```

Come vedete, questa volta il file HTML è sostanzialmente vuoto, contiene un collegamento ad un file di script JS, oltre ai due di prima al bootstrap e al CSS.

La differenza principale risiede nella funzione **getEventi()** che si avvia al caricamento della pagina. Essendo il CSS lo stesso di prima, andiamo ora a vedere tale funzione.

# Esercizio d'esame: HTML + CSS + JS + Ajax

## Specifiche Parte 2 - Soluzione (2)

JS

La funzione `getEventi()` è un esempio di come utilizzare Ajax con JavaScript per recuperare dati da un server, e poi manipolare il DOM (Document Object Model) per visualizzare questi dati nella pagina web. Suddividiamola in componenti, e andiamo a vederne il funzionamento nel dettaglio.

### Step 1: Creazione e Invio della Richiesta Ajax

- 1. Creazione di un oggetto XMLHttpRequest:** `let request = new XMLHttpRequest();` crea un nuovo oggetto di tipo `XMLHttpRequest`, che è usato per interagire con il server via HTTP senza dover ricaricare la pagina.
- 2. Apertura della richiesta:** `request.open("GET", "http://diiorio.nws.cs.unibo.it/twe/07.06.2022a/api/index.php");` configura il tipo di richiesta HTTP che si vuole fare (GET in questo caso) e l'URL a cui la richiesta deve essere inviata.
- 3. Invio della richiesta:** `request.send();` invia la richiesta al server configurato con il metodo `open()`.

```
let request = new XMLHttpRequest();
request.open("GET", "http://diiorio.nws.cs.unibo.it/twe/07.06.2022a/api/index.php");
request.send();
```

# Esercizio d'esame: HTML + CSS + JS + Ajax

## Specifiche Parte 2 - Soluzione (3)

JS

### Step 2: Gestione della Risposta

- 1. Assegnazione di un event handler per l'evento onload:** `request.onload=()=>{...}` definisce una funzione che verrà eseguita una volta che la richiesta riceve una risposta. Questo event handler gestisce la risposta del server.
- 2. Controllo dello stato della richiesta:** All'interno dell'event handler, `if (request.status == 200) {...}` controlla se la richiesta è stata completata con successo (status 200 significa "OK"). Se la richiesta è fallita, esegue un log dell'errore.

```
request.onload = () => {
  console.log(request);
  if (request.status == 200) {
    ...
  } else {
    ...
  }
}
```

# Esercizio d'esame: HTML + CSS + JS + Ajax

JS

## Specifiche Parte 2 - Soluzione (4)

### Step 3: Elaborazione e Visualizzazione dei Dati

- 1. Parsing dei dati JSON:** `var eventi = JSON.parse(request.response)` converte la stringa JSON ricevuta dal server in un oggetto JavaScript utilizzabile.
- 2. Selezione del contenitore nel DOM:** `var divElenco = document.getElementsByClassName('elenco')[0]`; seleziona il primo elemento del DOM con la classe elenco, dove verranno visualizzati gli eventi.
- 3. Iterazione sugli eventi ricevuti:** `for(var i=0; i < eventi.length; i++){...}` itera su ogni evento ricevuto dal server.
- 4. Condizione per l'inserimento nel DOM:** `if(eventi[i].evidenza){...}` verifica se l'evento deve essere mostrato a display (**evidenza = True -> l'evento viene mostrato**).
- 5. Creazione degli elementi DOM per ogni evento:** All'interno del ciclo for, vengono creati diversi elementi **div**, **h4**, **p** e **a** che rappresentano diversi aspetti dell'evento come titolo, orario, luogo, e link per acquistare i biglietti.
- 6. Aggiunta degli elementi al DOM:** `divElenco.appendChild(divRow)`; aggiunge ogni **divRow** creato al div elenco già esistente nella pagina, rendendo così visibili i dati degli eventi nella pagina web.

# Fine

Prima di concludere, volevo dirvi che c'è la possibilità di fare un'ulteriore esercitazione.

Noi abbiamo concluso il programma di laboratorio, ma **se dovreste avere dei dubbi, vi prego di accordarvi e di farmi sapere su cosa vi piacerebbe fare l'esercitazione extra** (volete rivedere le API, altro JS, Ajax, solo esercizi d'esame...). Potremmo anche fare un approfondimento sulle **Promesse**, sta a voi.

Se invece non è necessaria, allora nessun problema. Io **resto comunque sempre disponibile via mail per qualsiasi dubbio, correzione, o chiarimento.**

Lo so, non ho ancora caricato l'ultimo esercizio della parte sulle API, prometto verrà caricato la prossima settimana.

Vi auguro un buon lavoro, e in bocca al lupo per l'esame.  
Grazie a tutti.