

Ajax

Lecture 7

Programma di oggi

1

Correzione rapida esercizio per casa

```
// Riferimento al contenitore che mostrerà i prodotti aggiunti
const containerProdotti = document.getElementById('containerProdotti');

// Oggetto che mappa i pulsanti ai rispettivi prodotti
const productData = {
    btnP1: { name: 'Prodotto 1' }, // btnP1 corrisponde a "Prodotto 1"
    btnP2: { name: 'Prodotto 2' }, // btnP2 corrisponde a "Prodotto 2"
    btnP3: { name: 'Prodotto 3' }, // btnP3 corrisponde a "Prodotto 3"
};

/**
 * Funzione per aggiungere o aggiornare un prodotto nella lista
 * @param {string} buttonId - ID del pulsante che è stato cliccato
 */
function addOrUpdateProduct(buttonId) {
    // Ottieni le informazioni sul prodotto in base al pulsante premuto
    const product = productData[buttonId];

    // Verifica se il prodotto è già presente nel contenitore
    const existingProduct = Array.from(containerProdotti.children).find(
        (child) => child.getAttribute('data-product') === product.name
    );
}
```

2

Primo esercizio con Ajax (da simulazione d'esame)

```
function getEventi(){
    let request = new XMLHttpRequest();
    request.open("GET", "http://diorio.nws.cs.unibo.it/twe/07.06.2022a/api/index.php");
    request.send();
    request.onload = () => {
        console.log(request);
        if (request.status == 200) {
            var eventi = JSON.parse(request.response)
            console.log(eventi);
            var divElenco = document.getElementsByClassName('elenco')[0];
            for(var i=0; i < eventi.length; i++){
                if(eventi[i].evidenza){
                    var divRow = document.createElement('div');
                    divRow.setAttribute('class', 'evento row');

                    var divCol = document.createElement('div');
                    divCol.setAttribute('class', 'col-md-12 col-lg-3');
```

3

Secondo esercizio (individuale) con Ajax (da simulazione d'esame)

Correzione esercizio per casa

Vi veniva richiesto di creare una lista dei desideri (come quella in figura), e di curarne sia l'HTML, che il CSS, che il Javascript. L'esercizio è molto simile a quello che avevamo visto insieme sulla Lista di lettura, quindi andremo velocemente a vedere il nuovo esercizio, per poi spostarci subito su Ajax.

Liste dei desideri

Prodotti

Prodotto 1	<button>Aggiungi alla lista</button>
Prodotto 2	<button>Aggiungi alla lista</button>
Prodotto 3	<button>Aggiungi alla lista</button>

La tua lista dei desideri :

Prodotto 1 : 7
Prodotto 2 : 4
Prodotto 3 : 7

Metodo find → funzione freccia e funzione di callback (1)

```
const existingProduct = Array.from(containerProdotti.children).find(  
  (child) => child.getAttribute('data-product') === product.name  
);
```

In questo caso, stiamo verificando l'esistenza di un prodotto. Per farlo, oltre a convertire tutti i figli del container in un Array, andiamo a utilizzare un metodo degli Array, ovvero **find**.

Ma come funziona?

Il metodo **find** accetta una **funzione di callback** come **parametro**, che è (in questo caso) la funzione freccia definita all'interno del **find**. Tale metodo itera un array, e passa come **parametro** alla funzione di callback l'elemento corrente, andando perciò ad eseguire una funzione su ogni elemento (uno per uno) dell'array, finché uno di questi non restituirà **True**, a questo punto il **find** interrompe il ciclo e ritorna tale elemento. Se l'elemento non viene trovato, verrà ritornato **Undefined**.

```
(parametro) => { corpo della funzione }
```

Equivale a

```
function (parametro) {  
  return corpo della funzione;  
}
```

Metodo find → funzione freccia e funzione di callback (2)

Perciò, immaginiamo di avere il seguente HTML:

```
<div id="prodotti">
  <div data-product="apple">Apple</div>
  <div data-product="banana">Banana</div>
  <div data-product="cherry">Cherry</div>
</div>
```

Allora, i due snippet qui sotto saranno equivalenti:

```
const containerProdotti = document.getElementById('prodotti');
const product = { name: 'banana' };

let existingProduct = undefined;
const childrenArray = Array.from(containerProdotti.children);

for (let i = 0; i < childrenArray.length; i++) {
  const child = childrenArray[i];
  if (child.getAttribute('data-product') === product.name) {
    existingProduct = child;
    break;
  }
}

console.log(existingProduct); // Stampa: <div data-product="banana">Banana</div>
```

Equivale a

```
const existingProduct = Array.from(containerProdotti.children).find(
  (child) => child.getAttribute('data-product') === product.name
);

console.log(existingProduct); // Stampa: <div data-product="banana">Banana</div>
```

Ordinamento dinamico di elementi nel DOM (1)

Ora, ispezioniamo un pò più nel dettaglio la funziona di ordinamento:

Cosa fa questo codice?

```
function sortProducts() {  
    // Ottiene una lista di tutti i figli del contenitore  
    const products = Array.from(containerProdotti.children);  
  
    // Ordina i prodotti in base al valore di 'data-product'  
    products.sort((a, b) =>  
        a.getAttribute('data-product').localeCompare(b.getAttribute('data-product'))  
    );  
  
    // Riaggiunge i prodotti ordinati al contenitore  
    // Questo non crea duplicati ma riposiziona gli elementi nell'ordine corretto  
    products.forEach((product) => containerProdotti.appendChild(product));  
}
```

1. Trasforma gli elementi figli in un array:

- `containerProdotti.children` restituisce un `HTMLCollection` (una lista simile a un array).
- `Array.from(...)` lo converte in un array per poter utilizzare i metodi degli array come `.sort()`.

2. Ordina gli elementi in base all'attributo `data-product`:

- `products.sort(...)` utilizza `localeCompare` per confrontare alfabeticamente i valori dell'attributo `data-product`.

3. Riposiziona gli elementi ordinati:

- Con `appendChild`, gli elementi vengono aggiunti nuovamente al contenitore nell'ordine corretto.
- **Importante:** Non vengono duplicati, ma semplicemente spostati.

Ordinamento dinamico di elementi nel DOM (2)

Perché `appendChild` non crea duplicati?

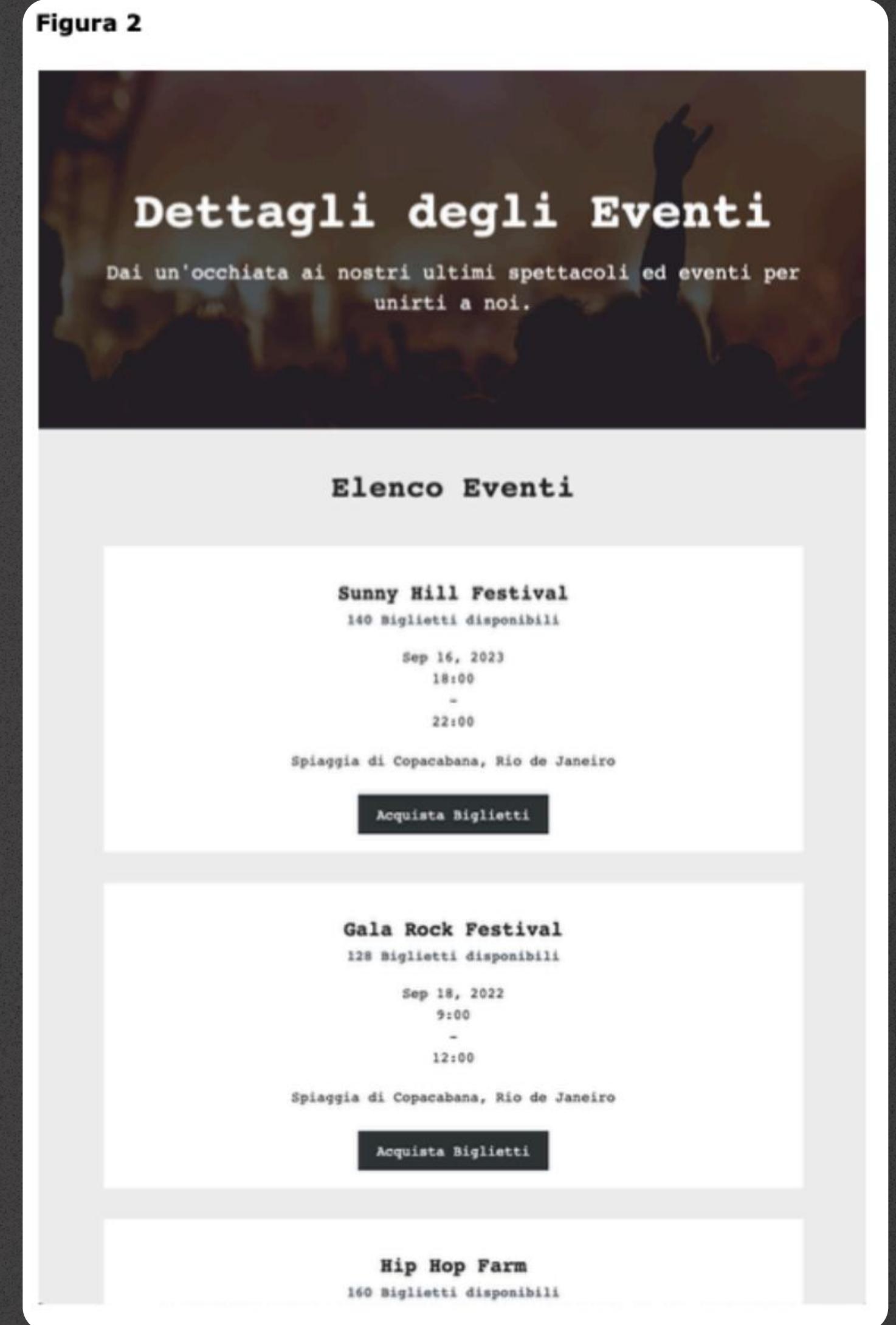
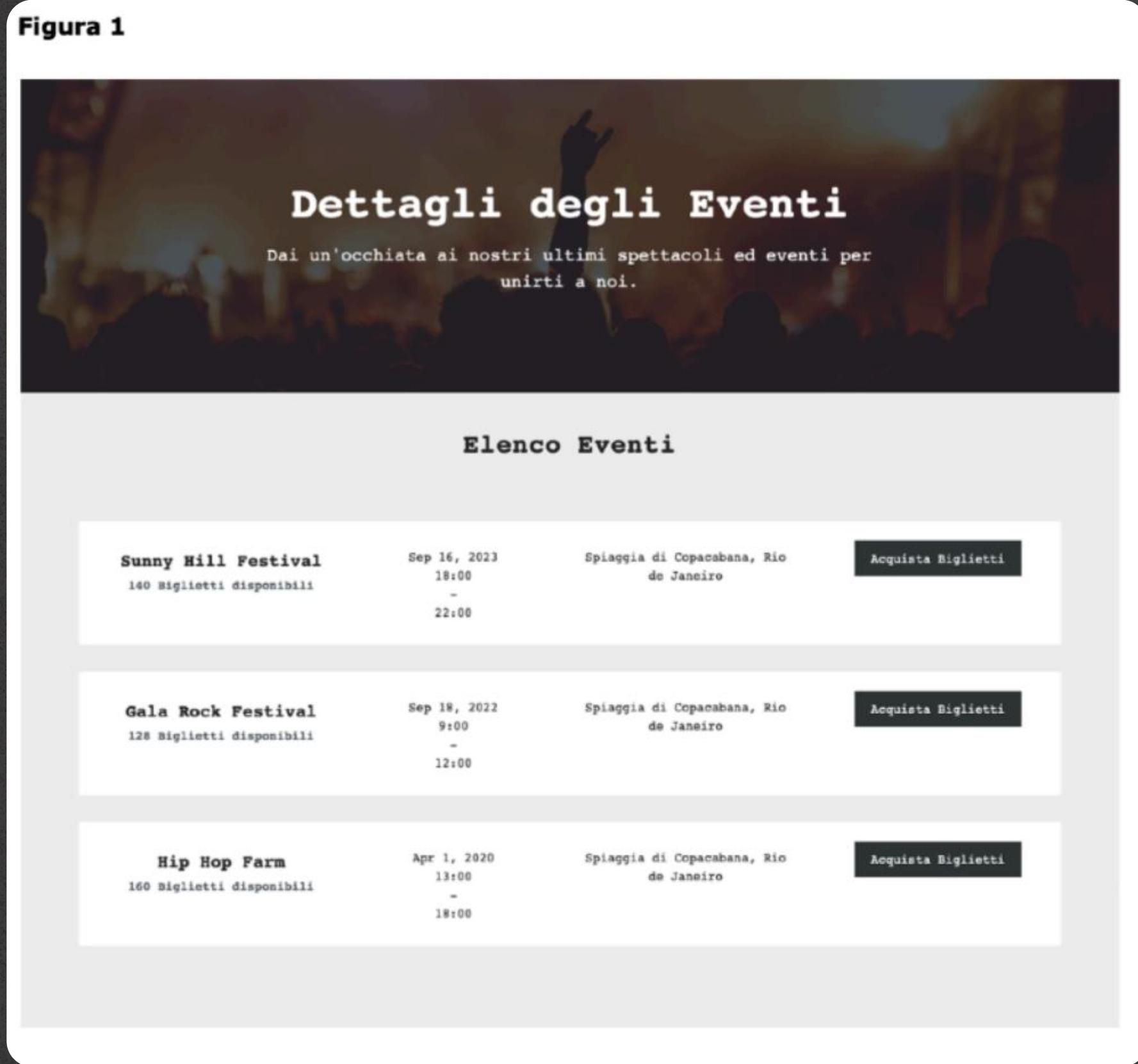
- Funzionamento di `appendChild`:
 - Se un elemento esiste già nel DOM, `appendChild` non lo duplica.
 - L'elemento viene spostato dalla posizione originale e aggiunto come ultimo figlio del contenitore specificato.
 - In questo caso, il codice riordina solo gli elementi esistenti senza creare copie.
- In caso di nuovi elementi:
 - Se un elemento è nuovo (non ancora nel DOM), `appendChild` lo aggiunge normalmente al contenitore.
 - Questo comportamento rende `appendChild` utile sia per riordinare elementi esistenti sia per aggiungerne di nuovi.

Cosa succede al DOM dopo l'ordinamento?

1. Il contenitore (`containerProdotti`) viene aggiornato con gli elementi nell'ordine specificato.
2. Non ci sono duplicati, perché ogni elemento esiste una sola volta nel DOM.
3. I nuovi elementi possono essere aggiunti senza interferire con l'ordine già stabilito.

Primo esercizio con Ajax

Scrivere il codice HTML e CSS, in un file esterno, per ottenere le visualizzazioni mostrate in Figura 1 e Figura 2 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.



Primo esercizio con Ajax - Specs

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>

Introduzione al Sistema a Griglia di Bootstrap

(anche se *in teoria* dovreste già conoscerlo)

Cos'è Bootstrap?

- Bootstrap è un framework CSS che semplifica la creazione di layout responsive e componenti UI.
- Il suo cuore è il sistema a griglia, che divide lo spazio disponibile in 12 colonne e consente di creare layout flessibili.

Sistema a Griglia di Bootstrap

1. Concetto base:

- Ogni riga (**row**) contiene un massimo di **12 colonne**.
- Le colonne (**col-...**) possono avere una larghezza specificata in numero di colonne (ad esempio, **col-6** occupa metà dello spazio disponibile).

2. Classi di larghezza:

- Bootstrap usa classi come **col-sm-**, **col-md-**, **col-lg-**, **col-xl-** per definire il comportamento su diversi breakpoint:
 - **col-sm**: Schermi piccoli ($\geq 576\text{px}$)
 - **col-md**: Schermi medi ($\geq 768\text{px}$)
 - **col-lg**: Schermi grandi ($\geq 992\text{px}$)
 - **col-xl**: Schermi extra-large ($\geq 1200\text{px}$)

3. Comportamento responsive:

- Se non specificato, le colonne si espandono automaticamente per riempire la riga.
- Puoi specificare larghezze diverse per breakpoint:

```
<div class="col-md-6 col-lg-4"></div>
```

- Su schermi medi (**md**), occupa 6 colonne (50%).
- Su schermi grandi (**lg**), occupa 4 colonne (33.33%).

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

Specifiche Parte 1 - Soluzione (1)

```
<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

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.

Successivamente, ogni evento sarà appunto definito allo stesso modo.

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

Specifiche Parte 1 - Soluzione (3)

```
body {  
    margin: 0px;  
}  
  
.title{  
    background-image: url('http://diorio.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;  
}
```

Definiamo ora lo stile della pagina.

Come specificato prima, il design responsive è fornito da Bootstrap, dunque a noi non resta che definire elementi di stile semplici, e aggiungere lo sfondo (riquadro giallo).

Aggiungiamo dunque Ajax (1)

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`

Aggiungiamo dunque Ajax (2)

```
[  
  {  
    "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 <http://diiorio.nws.cs.unibo.it/twe/07.06.2022a/api/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)

```
<!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://diiorio.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 questa funzione.

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

Specifiche Parte 2 - Soluzione (2)

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://diorio.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://diorio.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)

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

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.

Secondo esercizio con Ajax

Esercizio Individuale: Piante e Amore

Data la pagina HTML dell'esercizio **Piante e Amore** (visto nell'esercitazione aggiuntiva su HTML e CSS), in cui il DIV principale con l'elenco delle piante 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 esercizio2.html diverso dal precedente, e il JS, in un file esterno esercizio2.js

In particolare si realizzi:

1. Al caricamento, la pagina accede asincronamente in GET al servizio web <http://diiorio.nws.cs.unibo.it/twe/15.09.2022/api/index.php>, ottenendo un JSON con i dati relativi alle piante e li visualizza in modo appropriato.

Note sulla visualizzazione:

- riprodurre la visualizzazione dell'esercizio precedente; NON è necessario gestire entrambi i layout e le differenti larghezze dello schermo, ammesso anche un solo layout
- il link delle pagine relative ad ogni pianta sono calcolati dall'ID della pianta
- il file JSON non contiene solo le piante che dovrebbero essere mostrate ma anche altre. Vanno mostrate solo le piante con proprietà "evidenza" uguale a true e con proprietà "n_piante" uguale o maggiore a 1.
- Per le piante con proprietà "n_piante" uguale a 1, sotto il bottone "Acquista" compare un testo in rosso che dice "Solo 1 in magazzino!"
- i dati JSON di esempio includono alcune piante, ma il codice deve funzionare anche per un numero diverso di piante

Secondo esercizio con Ajax

Esercizio Individuale: Piante e Amore

Data la pagina HTML dell'esercizio **Piante e Amore** (visto nell'esercitazione aggiuntiva su HTML e CSS), in cui il DIV principale con l'elenco delle piante 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 esercizio2.html diverso dal precedente, e il JS, in un file esterno esercizio2.js

In particolare si realizzi:

1. Al caricamento, la pagina accede asincronamente in GET al servizio web <http://diiorio.nws.cs.unibo.it/twe/15.09.2022/api/index.php>, ottenendo un JSON con i dati relativi alle piante e li visualizza in modo appropriato.

Note sulla visualizzazione:

- riprodurre la visualizzazione dell'esercizio precedente; **NON** è necessario gestire entrambi i layout e le differenti larghezze dello schermo, ammesso anche un solo layout
- il link delle pagine relative ad ogni pianta sono calcolati dall'ID della pianta
- il file JSON non contiene solo le piante che dovrebbero essere mostrate ma anche altre. Vanno mostrate solo le piante con proprietà "evidenza" uguale a true e con proprietà "n_piante" uguale o maggiore a 1.
- Per le piante con proprietà "n_piante" uguale a 1, sotto il bottone "Acquista" compare un testo in rosso che dice "Solo 1 in magazzino!"
- i dati JSON di esempio includono alcune piante, ma il codice deve funzionare anche per un numero diverso di piante