

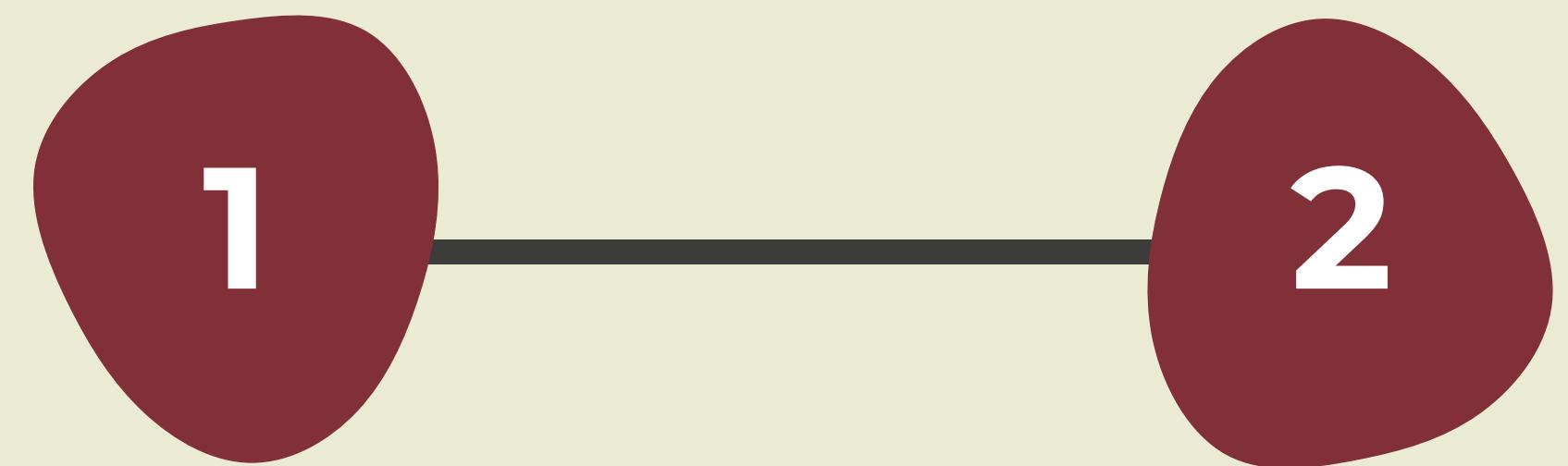
Lab Hours - Lecture EXTRA

Simulazione Esame

+ Advanced Concepts

Contact: **ellepuntopi.98@gmail.com**

Programma di oggi:



Simulazione
Esame
Completa

Esercizio Bonus
da una differente
simulazione

Simulazione Esame - Esercizio 1 - HTML+CSS - Specs

Scrivere il **codice HTML e CSS** per ottenere la visualizzazione mostrata in Figura (Prossima Pagina).

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

- la larghezza dell'area centrale, escluso un margine destro/sinistro arbitrario, è del 75% del viewport
- quest'area centrale non è equamente divisa tra la parte sinistra e quella destra: la parte sinistra (“Come prendersi cura di un coniglio nano”) occupa il 60% dell'area, e la parte destra (“Altri suggerimenti”) il 25% dell'area
- al passaggio del mouse su un link (in rosa), viene modificato il colore dello sfondo e diventa giallo
- cliccando su un link si apre la pagina corrispondente; creare un link arbitrario; NON è richiesto creare anche le pagine delle località
- Nel form, il “tipo di domanda” deve avere tre opzioni: “Alimentazione”, “Salute” ed “Altro”
- colori e dimensioni esatte di margini e padding non rilevanti, purché appropriate

Vincoli:

- non è ammesso usare attributi @id e @class nel sorgente HTML;
- non è ammesso usare tavole per organizzare il layout, ma sì per il contenuto su quale vegetali possono mangiare i conigli
- non è possibile usare Javascript (eventuali comportamenti dinamici vanno nell'esercizio successivo).

Risorse:

- Immagini disponibili su: <http://diiorio.nws.cs.unibo.it/twe/07.06.2022b/>

Simulazione Esame - Esercizio 1 - Immagine

Come prendersi cura di un coniglio nano

I conigli nani sono animali unici. E in quanto tali hanno bisogno di cura, amore, e pazienza. Vogliono che le persone spendono del tempo con loro giocando e prendendosene cura.

Ecco qui per te una panoramica su come prenderti cura del tuo **coniglio nano**. Per maggiori informazioni puoi andare alla [loro pagina Wikipedia](#).

Vegetale	Per il coniglio...
Radicchio	Va bene
Sedano	Va bene
Avocado	Non va bene
Broccoli	Non va bene
Carota	Va bene

Se hai domande, contattaci:

Il tuo email

Il tuo messaggio

Tipo di domanda

Alimentazione

Invia

Altri suggerimenti

Prendersi cura di un cane

September 1st, 2022

I cani sono animali molto...

[Read more](#)

Prendersi cura di un gato

April 1st, 2020

Alcuni dicono che i gati...

[Read more](#)



Simulazione Esame - Esercizio 1 - Svolgimento (1)

La pagina HTML è relativamente semplice da costruire. La cosa importante è essere in grado di strutturarla in maniera ordinata.

In particolare, non potendo utilizzare attributi **id** o **class**, è necessario dare una struttura facilmente comprensibile e richiamabile nel CSS.

Per farlo avremo la seguente:

- **body** (Giallo)
- **div-container** (Arancio)
- **div-1** (Verde)
- **div-2** (Azzurro)

Con il div container che incapsula div 1 e 2.

The screenshot shows a web page with the following structure:

- Header:** "Come prendersi cura di un coniglio nano" (How to take care of a nano rabbit).
- Text:** "I conigli sono animali unici. E in quanto tali hanno bisogno di cura, amore, e pazienda. Vogliono che le persone spendono del tempo con loro giocando a prendersene cura." (Rabbits are unique animals. As such, they need care, love, and attention. They want people to spend time with them playing.)
- Text:** "Ecco qui per te una panoramica su come prenderti cura del tuo coniglio nano. Per maggiori informazioni puoi andare alla [loro pagina Wikipedia](#). (Here is a summary of how to take care of your nano rabbit. For more information, you can go to their [Wikipedia page](#).)
- Table:** A table titled "Vegetale" (Vegetable) with columns "Vegetale" and "Per il coniglio..." (For the rabbit). The data is as follows:

Vegetale	Per il coniglio...
Radicchio	Va bene
Sedano	Va bene
Avocado	Non va bene
Broccoli	Non va bene
Carota	Va bene

- Contact Form:** "Se hai domande, contattaci:"
 - Il tuo email: [Redacted input field]
 - Il tuo messaggio: [Redacted input field]
 - Tipo di domanda: [Dropdown menu] Alimentazione
 - Invia [Green button]
- Sidebar:** "Altri suggerimenti" (Other suggestions)
 - [Prendersi cura di un cane](#) (September 1st, 2022)
I cani sono animali molto... [Read more](#)
 - [Prendersi cura di un gatto](#) (April 1st, 2020)
Alcuni dicono che i gatti... [Read more](#)
- Image:** A small image of a blue rabbit.



Simulazione Esame - Esercizio 1 - Svolgimento (2)

Ora che la struttura è definita, andiamo a popolare i due **div** (div-1 e div-2).

```
<body>
  <div>
    <div>...
    </div>

    <div>...
    </div>
  </div>
</body>
```

```
<div>
  <h2>Come prendersi cura di un coniglio nano</h2>
  <p>I conigli sono animali unici. E in quanto tali hanno...
  <p>Ecco qui per te una panoramica su come prenderti cura...

  <table>
    <tr>
      <th>Vegetale</th>
      <th>Per il coniglio...</th>
    </tr>
    <tr>
      <td>Radicchio</td>
      <td>Va bene</td>
    </tr>
    <tr>
      <td>Sedano</td>
      <td>Va bene</td>
    </tr>
    <tr>
      <td>Avocado</td>
      <td>Non va bene</td>
    </tr>
    <tr>
      <td>Broccoli</td>
      <td>Non va bene</td>
    </tr>
    <tr>
      <td>Carota</td>
      <td>Va bene</td>
    </tr>
  </table>

  <span>Se hai domande, contattaci:</span>

  <div>
    <span>Il tuo email</span>
    <input type="textarea">
    <span>Il tuo messaggio</span>
    <input type="textarea">
    <span>Tipo di domanda</span>
    <select>
      <option>Alimentazione</option>
      <option>Salute</option>
      <option>Altro</option>
    </select>
  </div>

  <button>Invia</button>
</div>
```

Div-1: Questo primo div, come è evidente dall'immagine, dovrà contenere:

- un titolo,
- un paragrafo,
- una tabella,
- un form,
- un bottone.

Div-2: Il secondo div invece conterrà:

- Paragrafi
- link
- immagine

```
<div> <!-- div-2 // Parte Destra -->
  <h2>Altri suggerimenti</h2>
  <p>
    <span>Prendersi cura di un cane</span>
    <span>Semptember 1st, 2022</span>
    <span>I cani sono animali molto...</span>
    <a href="#">linkarbitario.html>Read more</a>
  </p>
  <p>
    <span>Prendersi cura di un gatto</span>
    <span>April 1st, 2020</span>
    <span>Alcuni dicono che i gatti...</span>
    <a href="#">linkarbitario.html>Read more</a>
  </p>
  bunny.jpg">
</div>
```

Simulazione Esame - Esercizio 1 - Svolgimento (3)

Ora, sarà necessario aggiungere lo stile al documento. Non potendo utilizzare classi e id, dovremo necessariamente fare un massiccio uso di selettori CSS di annidamento.

```
/* Parte sinistra */
body>div>div:first-child{ ...
}

body>div>div:first-child>p{ ...
}
```

```
/* Parte destra */
body>div>div:nth-child(2){ ...
}

body>div>div:nth-child(2)>p{ ...
}

body>div>div:nth-child(2)>p>span:first-child{ ...
}

body>div>div:nth-child(2)>p>span:nth-child(2){ ...
}

body>div>div:nth-child(2)>p>span:nth-child(3){ ...
}
```

Come quelli che vedete di seguito.
Una volta selezionati gli elementi,
fornirgli lo stile specifico è molto
semplice.

Vediamo insieme il codice...

Simulazione Esame - Esercizio 2 - HTML+JS - Specs

Scrivere il **codice HTML e CSS e Javascript** per realizzare un conto alla rovescia in secondi come mostrato di seguito:

Parti da:  secondi.

Avvia

3

Note:

- il campo di testo permette di inserire **valori interi positivi**
- al click sul bottone ‘Avvia’ viene avviato il conto alla rovescia e **dopo ogni secondo viene aggiornato il testo in basso**
- **allo scadere del tempo viene mostrato** il messaggio “Fatto!”
- i dettagli di presentazione (dimensioni, font, margini, ecc.) non sono rilevanti

Vincoli:

- NON è ammesso l’uso di jQuery o altri framework

Simulazione Esame - Esercizio 2 - Svolgimento (1)

Questo esercizio può essere fatto in almeno due modi: **con o senza listener**.

Vediamo entrambe le possibilità:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Esercizio2 – No Listener</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div id="contenitore">
      <span>Parti da: <input type="number" step="1" min="1" value="10" id="input"> secondi.</span>
      <button onclick="avvia()">Avvia</button>
      <span id="output"></span>
    </div>
    <script src="script.js"></script>
  </body>
</html>
```

Senza listener, dobbiamo caricare lo script a fine HTML, e aggiungere la funzione **avvia()** al click del bottone.

Simulazione Esame - Esercizio 2 - Svolgimento (2)

Volendo invece aggiungere i listener, caricheremo lo script subito, per poi eseguire tutte le azioni all'interno del file JS.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Esercizio2 – Listener</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <script src="script.js" defer></script>
  </head>
  <body>
    <div id="contenitore">
      <span>Parti da: <input type="number" step="1" min="1" value="10" id="input"> secondi.</span>
      <button id="avviaButton">Avvia</button>
      <span id="output"></span>
    </div>
  </body>
</html>
```

Come vedete, lo script viene caricato subito, e la funzione **avvia()** non è più presente.

Simulazione Esame - Esercizio 2 - Svolgimento (3)

```
*{  
    font-size: large;  
}  
  
#contenitore{  
    display: flex;  
    flex-direction: column;  
}  
button, input{  
    width: min-content;  
    padding: 5px;  
}  
  
button{  
    margin-top: 2vh;  
    margin-bottom: 5vh;  
}  
  
#output{  
    font-size: 40px;  
}
```

Il foglio di stile è lo stesso in entrambe le versioni, e come potete vedere è molto semplice.

La parte fondamentale, come è evidente, risiede dunque nel JS. Vediamo la prima variante, quella senza Listener.

La funzione non dovrebbe risultarvi troppo oscura, abbiamo già visto un timer e tutti gli elementi presenti in essa.

L'unico richiamo che faccio è sull'esecuzione, che avviene ogni 1000ms (1sec), come specificato nel **setInterval()**.

```
function avvia(){  
    var n = document.getElementById("input").value;  
    console.log(n);  
    var spanOutput = document.getElementById("output");  
    var avviaButton = document.querySelector('button');  
    avviaButton.disabled = true;  
  
    var timer = setInterval(function(){  
        spanOutput.innerHTML = n;  
  
        if(n > 0){  
            n--;  
        }else{  
            spanOutput.innerHTML = "Fatto!";  
            clearInterval(timer);  
            avviaButton.disabled = false;  
        }  
    }, 1000);  
}
```

Simulazione Esame - Esercizio 2 - Svolgimento (4)

```
document.addEventListener('DOMContentLoaded', function() {
  var avviaButton = document.getElementById('avviaButton');
  avviaButton.addEventListener('click', avvia);

  function avvia() {
    var n = document.getElementById("input").value;
    console.log(n);
    var spanOutput = document.getElementById("output");
    avviaButton.disabled = true;

    var timer = setInterval(function() {
      spanOutput.innerHTML = n;

      if (n > 0) {
        n--;
      } else {
        spanOutput.innerHTML = "Fatto!";
        clearInterval(timer);
        avviaButton.disabled = false;
      }
    }, 1000);
  }
});
```

Vediamo ora l'esecuzione **con Listener**. Anche questa è stata vista nelle sue dinamiche di funzionamento.

La funzione rimane dunque la stessa.

L'unica differenza è che abbiamo aggiunto un listener al documento per il caricamento del DOM, e un listener al bottone per il click.

Una volta cliccato il bottone, viene eseguita la stessa identica funzione di prima.

Piccolo esperimento: il fatto che il bottone venga disabilitato non è richiesto nelle specifiche dell'esercizio. Provate a casa a rimuovere il disabled, per poi vedere come si sovrappongono i vari timer. Una volta visto questo, provate a trovare una soluzione diversa al problema.

Simulazione Esame - Esercizio 3 - API - Specs

Progettare un **API REST (parziale) per la gestione di una collezione di serie televisive.**

La collezione è organizzata in **Serie TV**, ognuna individuata da un **ID**, un **titolo** e una **data di pubblicazione**; ogni serie include **un insieme di episodi numerati** (valore intero) e **dotati di un titolo**; inoltre ogni serie TV appartiene ad **un genere tra ‘Animazione’, ‘Avventura’, ‘Commedia’, ‘Azione’, ‘Documentario’**.

Scrivere un file in formato JSON o YAML.

L'API permette di:

- ottenere l'elenco di tutte le serie TV di un dato genere X e pubblicate dopo una data Y
- aggiungere un nuovo episodio ad una serie TV
- modificare il titolo di uno specifico episodio di una serie TV

Specificare:

- URL di accesso, metodi HTTP, parametri e risposte con esempi. L'API restituisce un errore, con codice 400, se i parametri in input non sono corretti.

Note:

- Non è richiesto includere le sezioni host, schemes, servers, tags
- Non è richiesto gestire autenticazione

Simulazione Esame - Esercizio 3 - Svolgimento (1)

Come al solito, iniziamo dalle specifiche.

```
swagger: '2.0'  
info:  
  version: 0.0.1  
  title: "Esercizio 3"  
  description: "API per la gestione di serie tv"
```

Poi iniziamo con i paths. In questo caso ci vengono richieste 3 possibili operazioni:

- ottenere un elenco (**GET**)
- aggiungere un item (**POST**)
- modificare un item (**PUT**)

Vediamo dunque il primo metodo (a destra), che è il primo ad essere encapsulato nella definizione dei **paths** (in giallo).

Ecco un esempio di richiesta URL del metodo:

[http://editor.swagger.io/serie/?
dataPubb=2022-01-01&genere=animazione](http://editor.swagger.io/serie/?dataPubb=2022-01-01&genere=animazione)

```
paths:  
  /serie/:  
    get:  
      description: "Ottenere l'elenco di tutte le serie TV di un dato genere  
                   e pubblicate dopo una data specificata"  
      operationId: getEpisodeByGenereAndData  
      parameters:  
        - name: dataPubb  
          in: query  
          type: string  
          required: true  
        - name: genere  
          in: query  
          required: true  
          type: string  
          enum:  
            - "animazione"  
            - "avventura"  
            - "commedia"  
            - "azione"  
            - "documentario"  
      responses:  
        200:  
          description: "OK"  
          schema:  
            type: array  
            items:  
              $ref: "#/definitions/Serie"  
        400:  
          description: "i parametri in input non sono corretti"
```

Simulazione Esame - Esercizio 3 - Svolgimento (2)

Definiamo ora **POST** e **PUT**.

```
/serie/{idSerie}/episodi/:
  post:
    description: "Aggiungere un nuovo episodio ad una serie TV"
    operationId: addEpisodeById
    parameters:
      - name: idSerie
        in: path
        type: string
        required: true
      - in: body
        name: episodio
        required: true
        schema:
          $ref: "#/definitions/Episodio"

    responses:
      200:
        description: "Episodio aggiunto"
      400:
        description: "I parametri in input non sono corretti"
```

```
/serie/{idSerie}/episodi/{numero}:
  put:
    description: "modificare il titolo di uno specifico episodio di una serie TV"
    operationId: setTitleEpisodeById
    parameters:
      - name: idSerie
        in: path
        type: string
        required: true
      - name: numero
        in: path
        type: integer
        required: true
      - in: body
        name: titolo
        required: true
        schema:
          type: object
          properties:
            titolo:
              type: string
    responses:
      200:
        description: "Titolo episodio modificato"
      400:
        description: "I parametri in input non sono corretti"
```

Simulazione Esame - Esercizio 3 -

Svolgimento (3)

L'ultima cosa da aggiungere sono le **definitions** (in giallo). Iniziamo con la definizione dell'oggetto **Serie** (a destra). Questo sarà composto da **id**, **titolo**, **data di pubblicazione**, **genere**, ed **episodi**.

```
Episodio:  
  type: object  
  required:  
    - numero  
    - titolo  
  properties:  
    numero:  
      type: integer  
      example: 10  
      description: "Numero della puntata"  
    titolo:  
      type: string  
      example: "Il ritorno in Italia"  
      description: "Titolo della puntata"
```

Per quanto riguarda id, titolo, data e genere, andiamo a definire le specifiche, mentre per quanto riguarda gli episodi andiamo invece a richiamare l'oggetto **Episodio** nelle definitions (sinistra).

Non dimenticate gli example!

```
definitions:  
  Serie:  
    type: object  
    required:  
      - id  
      - titolo  
      - dataPubb  
      - genere  
      - episodi  
    properties:  
      id:  
        type: string  
        example: "S0001"  
        description: "ID della serie tv"  
      titolo:  
        type: string  
        example: "Diavoli"  
        description: "il nome della serie tv"  
      dataPubb:  
        type: string  
        format: date  
        example: "2022-01-01"  
        description: "Data di pubblicazione della serie"  
      genere:  
        type: string  
        enum:  
          - "animazione"  
          - "avventura"  
          - "commedia"  
          - "azione"  
          - "documentario"  
        example: "animazione"  
        description: "Genere della serie"  
      episodi:  
        type: array  
        items:  
          $ref: "#/definitions/Episodio"
```

Simulazione Esame - Esercizio BONUS - AJAX - Specs

Scrivere il codice **HTML, CSS e Javascript** per realizzare una calcolatrice speciale come mostrata di seguito.

Importante: La lista “Numeri magici del giorno” inizialmente è vuota e viene popolata al caricamento della pagina, a partire dai dati recuperati (in formato string) via Ajax all’indirizzo: <http://diiorio.nws.cs.unibo.it/twe/14.07.2022b/api/index.php>

Note e vincoli:

- La calcolatrice consente all’utente di inserire due numeri e di moltiplicarli o dividerli. Il risultato viene mostrato dopo il paragrafo "Il risultato é: ".
- La pagina mostra anche l’elenco delle operazioni eseguite dall’utente; le operazioni sono mostrate dopo il testo "Elenco delle tue moltiplicazioni e divisioni" in un elenco aggiornato dopo ogni operazione (inizialmente vuoto). Ad esempio, se l’utente ha moltiplicato 2 e 4, l’elenco include "2 * 4 = 8".
- È importante sottolineare che se il risultato (in questo caso, 8) è uno dei numeri magici del giorno, questo numero viene anche aggiunto all’elenco in "Numeri magici trovati".
- La seconda figura mostra un esempio della pagina dopo che l’utente ha eseguito una serie di operazioni.

Simulazione Esame - Esercizio BONUS - AJAX - Figure

Calcolatore: la moltiplicazione e la divisione di due numeri

Basta inserire due numeri e premere un pulsante!

1° numero:

2° numero:

Il risultato è:

Numeri magici del giorno:

- 2
- 8
- 20
- 28
- 50
- 82
- 126

Numeri magici trovati:

Elenco delle tue moltiplicazioni e divisioni:

Calcolatore: la moltiplicazione e la divisione di due numeri

Basta inserire due numeri e premere un pulsante!

1° numero: 126

2° numero: 1

Il risultato è:

126

Numeri magici del giorno:

- 2
- 8
- 20
- 28
- 50
- 82
- 126

Numeri magici trovati:

- 8
- 50
- 126

Elenco delle tue moltiplicazioni e divisioni:

- $2 * 3 = 6$
- $2 * 4 = 8$
- $2 * 25 = 50$
- $2 / 25 = 0.08$
- $126 * 1 = 126$

Simulazione Esame - Esercizio BONUS - AJAX

Fate voi...

HTML e CSS sono molto semplici, ora li vediamo direttamente dal codice.

La cosa complessa è la struttura dello script, che è diviso in 3 parti principali.

1. La prima parte serve ad aggiungere i vari Listener.
2. La seconda parte è una funzione che fa una chiamata Ajax al server. Tale chiamata restituisce una specifica struttura. Per vederla, basterà collegarsi all'URL fornito.
3. La terza parte è invece una funzione utile a popolare il documento.

Provate a scriverlo voi, poi lo correggiamo. Vi fornisco HTML e CSS.

N.B.: vi sto dando io HTML e CSS, quando questo esercizio è stato proposto all'esame, anche HTML e CSS erano da scrivere!

Simulazione Esame - Esercizio BONUS - Svolgimento (1)

Come dicevamo, la prima parte è quella che aggiunge i vari listener.

Nulla di nuovo, come al solito ne mettiamo uno per aspettare il caricamento del DOM, e due per l'esecuzione di funzioni al click sui bottoni. Infine, una volta caricato il DOM, andiamo anche a prendere dal server la lista di numeri magici.

```
document.addEventListener('DOMContentLoaded', function() {
    fetchMagicNumbers();
    document.getElementById('multiply').addEventListener('click', function() { calculate('*'); });
    document.getElementById('divide').addEventListener('click', function() { calculate('/'); });
});
```

```
{
  "source": "https://it.wikipedia.org/wiki/Numero_magico_(fisica)",
  "numeri_magici": [
    2,
    8,
    20,
    28,
    50,
    82,
    126
  ],
  "giorno": "2022-07-14"
}
```

Andiamo a vedere velocemente anche la risposta del server (a sinistra).

Come vedete questa contiene 3 chiavi: **source**, **numeri_magici**, e **giorno**.

Simulazione Esame - Esercizio BONUS - Svolgimento (2)

```
function fetchMagicNumbers() {  
    const xhr = new XMLHttpRequest();  
    xhr.open('GET', 'http://diorio.nws.cs.unibo.it/twe/14.07.2022b/api/index.php', true);  
  
    xhr.onload = function() {  
        if (xhr.status === 200) {  
            const data = JSON.parse(xhr.responseText);  
            const magicList = document.getElementById('magicNumbers');  
            data.numeri_magici.forEach(function(number) {  
                const li = document.createElement('li');  
                li.textContent = number;  
                magicList.appendChild(li);  
            });  
        } else {  
            console.error('Errore durante il caricamento dei numeri magici:', xhr.statusText);  
        }  
    };  
  
    xhr.onerror = function() {  
        console.error('Errore di rete durante il tentativo di recupero dei numeri magici');  
    };  
  
    xhr.send();  
}
```

Questa funzione rappresenta la parte 2, e serve a riprendere i numeri magici con Ajax. La struttura è la stessa dell'ultima volta, il terzo parametro del metodo **open**, ovvero **true**, specifica che la richiesta va eseguita in modo asincrono, il che significa che l'esecuzione del codice JS non si blocca in attesa che la richiesta HTTP completi. Invece, il browser invia la richiesta e continua con l'esecuzione del codice successivo, e la funzione di callback associata a **onload** viene eseguita solo dopo che la risposta è stata completamente ricevuta.

Simulazione Esame - Esercizio BONUS - Svolgimento (3)

La terza parte è la funzione che permette di calcolare i risultati delle operazioni effettuate, e di aggiornare le varie liste.

Come vedete, anche questa funzione è relativamente semplice. Ispezioniamo ora le parti più complesse.

```
function calculate(op) {
    const num1 = document.getElementById('number1').value;
    const num2 = document.getElementById('number2').value;
    if (num1 === '' || num2 === '' || (op === '/' && num2 === '0')) {
        alert('Per favore, inserisci dei numeri validi (non è possibile dividere per zero).');
        return;
    }

    const result = op === '*' ? num1 * num2 : num1 / num2;
    document.getElementById('result').textContent = result;

    // Aggiorna l'elenco delle operazioni
    const operationText = `${num1} ${op} ${num2} = ${result}`;
    const newOpLi = document.createElement('li');
    newOpLi.textContent = operationText;
    document.getElementById('operationsList').appendChild(newOpLi);

    // Verifica se il risultato è un numero magico
    const magicNumbers = Array.from(document.getElementById('magicNumbers').children).map(li => li.textContent);
    if (magicNumbers.includes(String(result))) {
        const newMagicLi = document.createElement('li');
        newMagicLi.textContent = result;
        document.getElementById('magicFound').appendChild(newMagicLi);
    }
}
```

Simulazione Esame - Esercizio BONUS - Svolgimento (4)

Quello che vedete sotto è un operatore ternario, sostanzialmente un **if** compresso. Controlla se l'elemento **op** è uguale a * o ad altro (**else**), e a seconda di ciò calcola **result** come moltiplicazione o divisione dei due numeri.

```
const result = op === '*' ? num1 * num2 : num1 / num2;
```

La parte sotto invece contiene dei \$, questi servono ad inserire in una stringa degli elementi del codice (una variabile, una costante...), che vengono poi formattati come stringa.

```
const operationText = `${num1} ${op} ${num2} = ${result}`;
```

Sotto invece vedete la creazione di **magicNumbers**. Questo avviene prendendo tutto ciò che è contenuto nell'elemento con ID specificato (che è un ****), dunque avremo una serie di elementi **** che si ottengono mediante il metodo **.children**. Otteniamo perciò una **HTMLCollection** (un oggetto non array), che viene convertito in array grazie ad **Array.from()**, sul quale viene poi chiamato il metodo **map**, che in questo caso prende il **textContent** di tutti gli elementi **** dell'array, e lo utilizza per popolare un nuovo array, che sarà perciò il risultato della riga.

```
const magicNumbers = Array.from(document.getElementById('magicNumbers').children).map(li => li.textContent);
```

Simulazione Esame - Esercizio BONUS - Svolgimento (5)

Vi ho aggiunto anche uno script alternativo sul github. Questo rimpiazza la funzione Ajax con la funzione **.fetch()**, utilizzata per fare richieste asincrone. Approfondite il funzionamento di **.fetch()**, poi se avete domande mandatemi una mail o chiedetemi un colloquio (Teams).

```
function fetchMagicNumbers() {
    fetch('http://diiorio.nws.cs.unibo.it/twe/14.07.2022b/api/index.php')
        .then(response => response.json())
        .then(data => {
            const magicList = document.getElementById('magicNumbers');
            data.numeri_magici.forEach(number => {
                const li = document.createElement('li');
                li.textContent = number;
                magicList.appendChild(li);
            });
        })
        .catch(error => console.error('Errore durante il caricamento dei numeri magici:', error));
}
```

Fine

Abbiamo finito le lezioni di laboratorio.

Grazie mille per l'attenzione e la partecipazione.

Rimango disponibile via mail e su Telegram per qualsiasi domanda/dubbio. Se dovreste aver bisogno di un ricevimento potete inviarmi una mail al solito indirizzo.

Vi auguro un buon proseguimento, e in bocca al lupo per l'esame!