

JS

Lab Hours - Lecture 4

Javascript

Base

Contact: ellepuntopi.98@gmail.com

Programma di oggi:



WarmUp Step - Familiarizzare con JS

Oggi seguiremo una struttura un pò diversa rispetto alle scorse volte. La lezione sarà anticipata da una breve sessione di *WarmUp*, in cui vi verranno dati dei semplici esercizi da eseguire individualmente, utili per richiamare alcuni concetti (che verranno successivamente approfonditi) e familiarizzare con la sintassi JS.

Gli esercizi saranno presi da **W3Schools**. Nel caso non doveste essere in grado di risolverne alcuni, c'è la possibilità di vedere immediatamente la soluzione.

WarmUp

Prendiamo un pò di familiarità con alcuni elementi base.

Accedendo tramite codice QR, risolvete tutti gli esercizi delle sezioni, a partire da **Variables** fino ad **Array Sort**.

Avete 15/20 minuti di tempo circa, buon lavoro.



Esercizi Individuali

Per gli esercizi individuali di questa lezione utilizzeremo il seguente codice HTML (lo trovate, come al solito, su GitHub).

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Lab4: Esercizi Base Javascript</title>
5          <script type="text/javascript">
6              //Inserire elementi JS
7
8              /* Come al solito, sarebbe preferibile utilizzare
9                 file esterni, per avere un maggiore controllo:
10                e.g. <script type="text/javascript" src="percorso/al/tuo/script.js"> */
11         </script>
12     </head>
13     <body>
14         <h1>Prove Javascript</h1>
15         <p id="para1">Un paragrafo di prova.</p>
16         <button type="button" onclick="onclick_fn()">Clicca qui</button>
17     </body>
18 </html>
```

Esercizio 1: Funzioni e sintassi

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”.

Successivamente, scrivete la stessa identica funzione, stavolta però utilizzate la **sintassi di funzione a freccia**.

Esercizio 1: Funzioni e sintassi - Soluzioni

Per quanto riguarda la sintassi tradizionale, semplicemente definiamo la funzione come segue.

```
1  function onclick_fn(){
2      |     alert("Hello World");
3      }
```

Mentre, la sintassi a freccia, ci permette di definire la stessa funzione nel modo sotto, ma ci sono anche altre possibilità...

```
1  let onclick_fn = () => alert('hello world');
```

Approfondimento: Sintassi a freccia

JS

Possiamo definire la funzione di prima in tre modi differenti, sempre utilizzando la sintassi a freccia.

```
1 let onclick_fn = () => alert('hello world');
2 let onclick_fn = () => (alert('hello world'));
3 let onclick_fn = () => {alert('hello world')};
```

1. La prima sintassi è la più concisa, la migliore per definire una funzione del genere. Infatti, è una forma abbreviata che può essere usata quando la funzione freccia esegue una sola espressione, e viene spesso impiegata per le funzioni di callback semplici.
2. Questa sintassi, con parentesi tonde, è comunemente usata per restituire un valore da una funzione freccia senza dover scrivere return. In questo caso, funziona lo stesso, ma non è necessario.
3. Questa sintassi, che utilizza parentesi graffe per delimitare il corpo della funzione, è la forma più versatile, e permette di definire l'esecuzione di più istruzioni al loro interno.

Esercizio 2: Manipolazione DOM

Utilizzando il documento HTML specificato prima, 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 primo “h1”, e lo si mostri all’interno del paragrafo con id “para1”.

Esercizio 2: Manipolazione DOM - Soluzione (1)

```
1 function onclick_fn(){
2     var inner_html = document.getElementsByTagName('h1')[0].innerHTML;
3     var para1 = document.getElementById('para1');
4     para1.textContent = inner_html;
5 }
```

La prima riga interna alla funzione cerca tutti gli **h1** all'interno del documento. Il metodo **getElementsByTagName()** restituisce un array di elementi, e, dopo aver selezionato il primo elemento **<h1>** (idx=0), estrae il suo contenuto HTML (cioè il testo e qualsiasi markup interno) e lo assegna alla variabile **inner_html**.

Esercizio 2: Manipolazione DOM - Soluzione (2)

JS

La seconda riga interna alla funzione cerca nella pagina un elemento con l'ID specifico "**para1**". Essendo l'ID unico all'interno di una pagina HTML, la funzione **getElementById()** restituirà un singolo elemento, che viene poi assegnato alla variabile **para1**.

Infine, l'ultima linea di codice imposta il contenuto testuale (**textContent**) dell'elemento **<p>** selezionato in precedenza (e ora referenziato dalla variabile **para1**) al valore della variabile **inner_html**, che è il contenuto HTML del primo elemento **<h1>** trovato nella pagina.

È importante notare che, assegnando a `textContent`, qualsiasi markup HTML contenuto in `inner_html` viene trattato come testo semplice, e non come HTML. Ciò significa che se `inner_html` contiene tag HTML, questi verranno visualizzati come testo piuttosto che essere interpretati come elementi HTML.

Per approfondire la differenza tra **innerHTML** e **TextContent** (che non abbiamo tempo di vedere oggi) visitate il seguente link: <https://www.geeksforgeeks.org/difference-between-textcontent-and-innerhtml/>

Esercizio 3: Timer

Utilizzando il documento HTML specificato prima, 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 ”paral”, 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: Timer - Soluzione (1)

```
1  var onclick_fn = function() {  
2      var button = document.getElementsByTagName("button")[0];  
3      var para = document.getElementById("para1");  
4      button.disabled = true;  
5      var max_sec = sec = 10;  
6      para.textContent = 0;  
7      var timer = setInterval(function(){  
8          para.textContent = max_sec - (--sec);  
9          if (sec <=0){  
10              clearInterval(timer);  
11              button.disabled = false;  
12          }  
13      }, 1000);  
14  };
```

Esercizio 3: Timer - Soluzione (2)

```
var button = document.getElementsByTagName("button")[0];
```

Questa istruzione seleziona il primo elemento **<button>** trovato nella pagina e lo assegna alla variabile **button**.

```
var para = document.getElementById("para1");
```

Viene selezionato l'elemento con l'ID specificato "**para1**" e assegnato alla variabile **para**. Questo elemento verrà utilizzato per visualizzare il conto alla rovescia.

```
button.disabled = true;
```

Questa istruzione rende il pulsante selezionato non cliccabile dall'utente, impostando la sua proprietà **disabled** su **true**.

Esercizio 3: Timer - Soluzione (3)

```
var max_sec = sec = 10;
```

Vengono inizializzate **due variabili**, **max_sec** e **sec**, entrambe con valore 10. Questo indica che il conto alla rovescia partì da 10 secondi.

```
para.textContent = 0;
```

Il contenuto testuale dell'elemento selezionato precedentemente (indicato con "para") viene inizializzato a 0. Questo è inteso come “un reset” in preparazione del conto alla rovescia.

Esercizio 3: Timer - Soluzione (4)

```
var timer = setInterval(function(){...}, 1000);
```

Viene creato un timer che esegue una funzione ogni 1000 millisecondi (1 secondo). All'interno di questa funzione anonima, avvengono le seguenti operazioni:

```
para.textContent = max_sec - (--sec);
```

Ad ogni tick del timer, il valore di sec viene decrementato (decremento prefisso, con il `--` si diminuisce di 1) e il risultato della sottrazione tra `max_sec` e il nuovo valore di sec viene assegnato come contenuto testuale dell'elemento para, aggiornando così il conteggio alla rovescia visualizzato all'utente.

```
if (sec <=0){...}
```

Se il valore di sec raggiunge 0, o un valore inferiore, il timer viene interrotto con `clearInterval(timer);` e il pulsante viene riabilitato (`button.disabled = false;`), permettendo nuovamente all'utente di cliccarlo.

Esercizio 3: Timer - Approfondimento Decrementi

- **Decremento Prefisso (--variabile):** Diminuisce il valore della variabile di 1 **prima di restituire il valore**. Questo significa che se utilizzate **--variabile** in un'espressione, state usando il valore già decrementato della variabile in quella espressione.
- **Decremento Postfisso (variabile--):** Diminuisce il valore della variabile di 1 **dopo aver restituito il suo valore corrente**. Se utilizzate **variabile--** in un'espressione, state usando il valore originale della variabile prima che venga decrementato, e il decremento avverrà subito dopo.

Ora, provate ad utilizzare il decremento postfisso nella funzione del timer e vedete che succede.

Esercizio d'esame

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 d'esame - Soluzione

Il risultato sarà la seguente **stringa**: "3,8,3,5"

Vediamo gli step:

- Pre-ForLoop: b="3"
- Iterazione 1 (i = 1): a[1] = "," -> b= "3," **[Notare che siamo partiti da idx=1]**
- Iterazione 2 (i = 2): a[2] = "8" -> b = "3,8"
- Iterazione 3 (i = 3): a[3] = "," -> b = "3,8,"
- Iterazione 4 (i = 4): a[4] = "3" -> b = "3,8,3"
- Iterazione 5 (i = 5): a[5] = "," -> b = "3,8,3,"
- Iterazione 6 (i = 6): a[6] = "5" -> b = "3,8,3,5"
- L'iterazione si ferma qui, perché la condizione del ciclo for specifica **i < 7**. La funzione restituisce dunque b = "3,8,3,5".

Esercizi d'esame: Approfondimento DOM loading

L'esercizio che abbiamo appena visto era molto basilare. Richiedeva la comprensione di For Loop e Data Types, niente di più.

La prossima volta però vedremo qualche esercizio d'esame più complesso, in cui vi verrà richiesto di implementare direttamente alcune funzioni JS. Inoltre, ci saranno esercizi che vi richiederanno di prendere alcuni contenuti dalla pagina HTML (ad esempio, tutti gli elementi di una data classe).

Per poter prendere elementi da una pagina, dovrete aspettare che la pagina sia caricata completamente prima che la funzione JS venga eseguita.

In situazioni del genere, è importante encapsulare la funzione all'interno di un Listener. Un esempio è il seguente:

```
document.addEventListener('DOMContentLoaded', (event) => {  
    // Funzione(i) JavaScript ed esecuzione qui dentro  
});
```

Esercizio d'esame Individuale - Per Casa

JS

Ora vi lascio un esercizio da svolgere autonomamente a casa. Questo è un altro esercizio di esami passati, quindi prestate particolare attenzione. Il codice HTML per testare la vostra soluzione è lo stesso che abbiamo appena utilizzato. **Attenzione:** all'esame non vi verrà fornito codice HTML in esercizi del genere.

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.

Per svolgere bene l'esercizio dovete approfondire: array associativi, DOM Loading e Listener, la differenza tra TextContent ed InnerHTML.

JS

Fine