**Eventi**

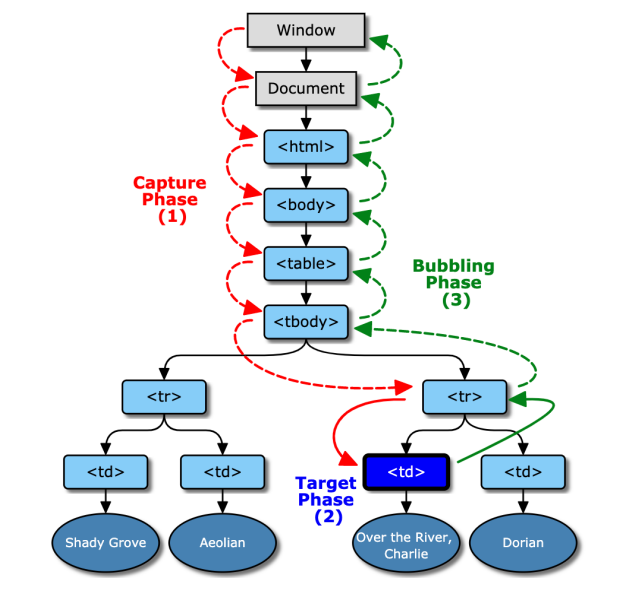
Usare onclick è considerato una bad practice perché si tende ad accoppiare troppo il markup e javascript.

Meglio usare **addEventListener(tipoEvento, callback)**

*document.querySelector('div’).addEventListener('click', function() { alert('hello!') })*

esiste un terzo parametro che è capture, che è un booleano e se è true viene chiamato solo in fase di capturing.

Gli eventi hanno 3 fasi:



* **Capturing** - l’evento si propaga dal contenitore verso i propri discendenti. L’evento si sposta dalla root del document fino all’elemento target chiamando gli handlers assegnati con addEventListeners(…, true)
* **At target** - l’evento è esattamente sull’oggetto su cui è stato richiesto. Tutti gli handlers assegnati all’elemento target vengono chiamati
* **Bubbling** - l’evento si propaga dal nodo su cui è stato agganciato fino alla radice del documento. L’evento risale dall’elemento target su fino alla root del documento, chiamando gli handlers assegnati con on, attributi HTML o addEventListener

Immagine che contiene testo

Descrizione generata automaticamente

Si possono avere due handler uno per il capturing e l’altro per il bubbling. Il primo si setta ponendo a true capture l’altro non mettendo niente.

Onclick e tutti gli eventi html sono eseguiti in fase di bubbling.

L’evento focus non ha bubbling, anche altri elementi non hanno bubbling

Il capturing è usato raramente.

Se si rimuovono elementi dal dom è buona cosa togliere anche i listener.

**Event delegation**

Consiste nel collegare l’handler di un evento a un solo elemento invece che ai discendenti

Immagine che contiene testo

Descrizione generata automaticamente

Event.target ha le informazioni dell’elemento che richiama l’evento.

Per usare il dataset il tag deve essere ***data-nomeCustom*** in JS lo richiamo tramite event.target.dataset.nomeCustom

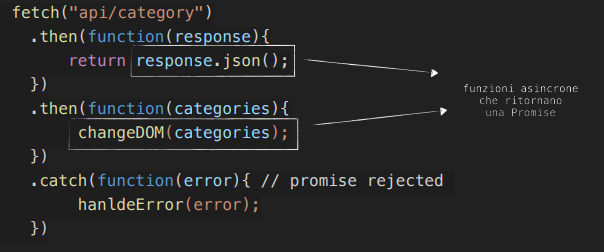
L’evento DEVE essere in bubbling o in capturing.

**Programmazione asincrona**

**Promises** sono un modo per chiamare un codice di lunga esecuzione e attendere il ritorno di un risultato.

Stati di una richiesta:

* **pending**: stato iniziale, sospeso: ne soddisfatta, ne rifiutata ancora.
* **fulfilled**: l’operazione è avvenuta con successo completamente.
* **rejected**: l’operazione è fallita.



Una Promise in sospeso può essere fulfilled con un valore o rejected con un motivo (errore). Quando si verifica una di queste opzioni, viene chiamata la callback associata ai metodi then/catch della Promise.

Promise.all() accetta un array di oggetti Promise come input e restituisce una

Promise. La Promise restituita verrà rifiutata se una delle Promises viene

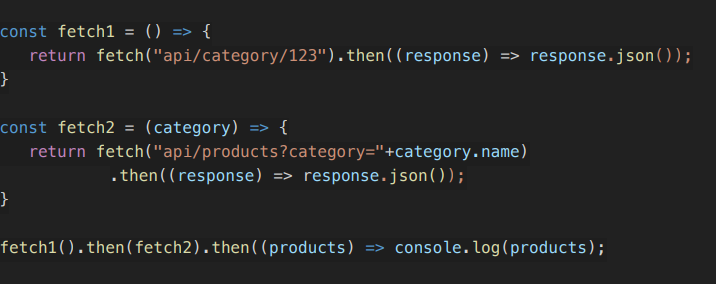
rejected. In caso contrario, verrà fulfilled con un array dei valori ritornati da

ciascuna Promise di input.

Immagine che contiene testo

Descrizione generata automaticamente

Fetch sequenziali, di può eseguire una fetch dopo un altra



* Le funzioni async e la keyword await permettono di rendere il codice più leggibile quando abbiamo una concatenazione di Promises
* Rendono il codice asincrono più simile al codice sincrono/procedurale, semplificando l’uso delle Promises

Immagine che contiene testo

Descrizione generata automaticamente

È importate capire che await non causa il blocco del programma e non fa nulla fino a quando la Promise non è risolta (o va in eccezione). Il codice rimane asincrono e ciò significa che qualsiasi codice che utilizza await è anch’esso asincrono.

Immagine che contiene testo

Descrizione generata automaticamente

**è possibile utilizzare la parola chiave await solo all'interno di funzioni dichiarate con la parola chiave async.**



Con quest’ultima riga si ferma la cascata del async, chi chiama async **deve** essere a sua volta async