# PROGRAMMAZIONE ASINCRONA

Javascript di default è sincrono ma può avere un comportamento asincrono.

Esempio di codice asincrono:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente setTimeout aspetta (in parallelo al programma) **circa** 2 secondi e poi chiama la callback. Il programma non rimane bloccato.

**Callback** sono uno dei modi principali per scrivere codice asincrono

Asincronicità possibile grazie a due elementi:

* Execution environment (*node, javascript nel browser*)
* Event loop

*Query database, richieste http… vengono ottimizzati da asincronicità*

### Immagine che contiene testo, schermata, Carattere Descrizione generata automaticamente**Callback asincrone**

Modo base per fare codice asincrono, eseguite in modo non bloccante.

* Gestire azioni utente (click bottone)
* Gestire operazioni di I/O (fetch di un documento)
* Gestire intervalli di tempo (timers)
* Interfacciarsi con il database

#### Immagine che contiene testo, Carattere, schermata Descrizione generata automaticamente**TIMERS**

* **setTimeout()** → chiama una funzione dopo un certo tempo (max 231-1 ms)
  + callback
  + secondi
  + Immagine che contiene testo, Carattere, schermata

    Descrizione generata automaticamenteparametri della callback
* **setInterval()** → chiama una funzione ciclicamente ogni intervallo di tempo
* **clearInterval(id) →** ferma esecuzione della setInterval con id id

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**Gestione errori callback**, default:

* primo parametro contiene l’errore
* secondo parametro è il contenitore per i risultati, per i dati

### **DATABASE**

In SqlLite, il database è contenuto interamente dentro un file (positivo per app piccole, no per app grosse)

Queries:

const sql=”SELECT …”;

**db.all()** → esegue e ritorna tutte le righe della callback

* se **err**=true → c’è stato un errore, altrimenti rows contiene un risultato
* **rows** è un array, ogni item contiene i campi del risultato

**db.get()** → prende il primo risultato in risposta ad una query

* se c’è un errore sintatico nella query o database inaccessibile → err pieno
* se la query è sintatticamente corretta ma non produce risultati → row=undefined

**db.each()** → esegue la callback per ogni risultato

**db.run()** → per statement che non ritornano un valore

* create table
* insert
* update
* funzioni callback
  + this.changes → quante righe sono state modificate dalla nostra operazione
  + this.lastID → ultimo ID assegnato all’ultima riga inserita (vale solo per insert)

NOTA: con db.run bisogna usare una function normale perché un arrow function andrebbe a sovrascrivere i valori this.

#### PARAMETRI NELLA QUERY

****Per passare un parametro alla query, si usa il **?** nella query e [] nella chiamata

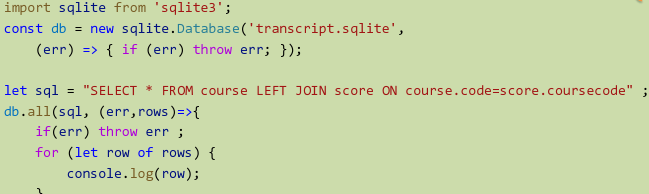


Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteImportare librerie

Collegarsi a database

Immagine che contiene testo, Carattere, schermata, calligrafia

Descrizione generata automaticamenteScrittura query

Db.all con stampa risultato

*Esempi di query e come non farle tenendo conto che vengono eseguite in maniera asincrona.*

### PROMISES

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteOggetto che rappresenta il **possibile completamento** di un’operazione asincrona. Contiene la promessa che questo oggetto verrà servito e quindi si potranno effettuare operazioni. → mi permette di evitare callback infinite.  
Quando una callback viene letta può essere in **3 stati:**

* **pending** → sono ancora una promessa, non ancora soddisfatta
* **fulfilled, resolve →** soddisfattacon successo
* **rejected →** non andata a buon fine, rifiutata

**Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente**

**Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente**

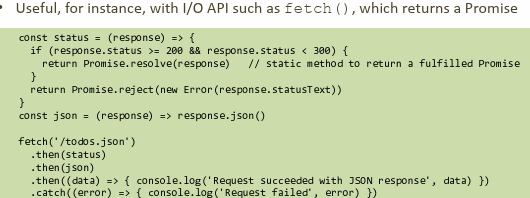
**Consumare la promise:**

* **then:** se resolve
* **catch:** se rejected (si può omettere se non utile, tipo se si causa errore direttamente)
* Immagine che contiene testo, schermata, Carattere, linea

  Descrizione generata automaticamente**finally:** si esegue in ogni caso

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente **NOTA:** le promise sono **concatenabili** in modo tale da eseguire delle funzioni in sequenza facendo in modo che si succedano in caso di risultato valido



Le promise possono anche essere eseguite in parallelo (ad esempio posso far eseguire 2 promise in parallelo e attendere la fine di entrambe per andare avanti)

* **Promise.all**()
  + Prende **tutte le promise** inserite come argomento
  + Se tutte le promise sono state fulfilled, ritorna un array con tutti i valori delle promise risolte
  + Se almeno 1 non viene risolta, nessuna viene risolta
  + Argomento di queste promise.all può anche non essere una promise (non farà mai rejected, sempre valida)
* **Promise.race**()
  + Ritorna una promise che è fulfilled o rejected quando **la prima delle promise** nell’array è fulfilled o rejected (NOTA: se c’è una funzione non promise, sicuro torna fulfilled)

### ASYNC/AWAIT

Keyword per semplificare la scrittura di codice asincrono usando promise.

* **Async** prima di una funzione → indica che la funzione ritornerà una promise
* Immagine che contiene testo, schermata, Carattere

  Descrizione generata automaticamente**Awayt** prima di una funzione chiamata → indica che la funzione che sto chiamando, ritorna una promise e che quindi devo aspettare che quella promise sia fulfilled o rejected.
  + Blocca esecuzione finchè non viene risolta la promise
    - In caso di errore → throw
  + **Nota: una awayt può essere usata solo in una funzione async**

#### Promise vs async-aways

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

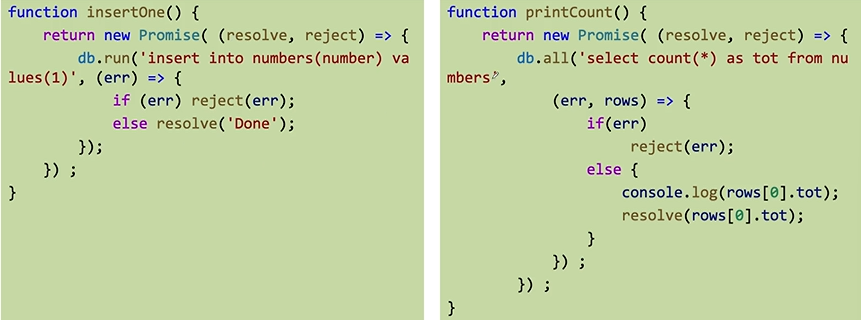
Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

* Se l’output della funzione 2, dipende dall’output della funzione 1 → **await**
* Se due funzioni possono girare in parallelo → due funzioni async che runnano in // con Promise.all
  + *Async-awayt non ha un meccanismo per fare promise.all() o promise.race() e quindi in quel caso bisogna usare le* ***promise****.*
* **Immagine che contiene testo, schermata, Carattere, calligrafia

  Descrizione generata automaticamente**Generalmente è più comodo usare async-await ma a volte (sqlite3) non è possibile farlo.

**NOTA**: **Resolve** → ritorno promise fuori da funzione

NOTA: Essendo esecuzione asincrona, nel caso barrato, si chiude il database prima che sia effettivamente usato o durante l’utilizzo.