Promise

PROMISE

- Le promise offrono una alternativa al sistema di callback.
- La promise è un oggetto che avvia una task al momento della creazione. In se rappresenta una task avviata e che sarà completata in futuro.

```
new Promise((resolve, reject) => {
// Corpo della "task asincrona"
});
```

PROMISE

- Una promise può avere tre stati: "pending", "fullfilled" e "rejected". Al momento dell'istanziazione lo stato è "pending".
- La funzione che viene passata al costruttore della promise ha due parametri: `resolve` e `reject` (a loro volta funzioni).
- Nel momento in cui `resolve` viene invocata la promise passa allo stato "fullfilled", mentre se viene chiamata `reject` passa allo stato "rejected".

PROMISE

- Nel prototype dell'oggetto Promise sono presenti le funzioni `then`, `catch` e `finally`.

  Ognuna di queste prende come parametri una funzione che verrà chiamata al momento di cambio di stato.

  - "resolved" - `then` e `finally` vengono chiamate e ricevono come input i valori passati a `resolve`.
  - "rejected" - `catch` e `finally` vengono chiamate e ricevono come input i valori passati a `reject`.

PROMISE

```
let myFirstPromise = new Promise((resolve, reject) => {
  setTimeout(function() {
    resolve("Success!");
  }, 250);
});

myFirstPromise.then((successMessage) => {
  console.log(successMessage);
});
```

PROMISE

Cosa succede se il `then` viene settato dopo la risoluzione della promise?

```
let myFirstPromise = new Promise((resolve, reject) => {
  setTimeout(function() {
    console.log("Resolving...");
    resolve("Success!");
  }, 250);
});

setTimeout(function() {
  console.log("Setting a then...");
  myFirstPromise.then((successMessage) => {
    console.log(successMessage);
  });
}, 1000);

// Resolving...
// Setting a then...
// Success!
```

PROMISE

Supponiamo di avere una lista di path a delle risorse che vogliamo caricare. Possiamo creare una funzione per recupera questi file che, anzichè bloccare l'esecuzione in attesa del caricamento, restituisce una *Promise*.

```
const paths = [
  "/path/to/resource.png",
  "/other/image.png"
];

function loadFile(path) {
  return new Promise((resolve, reject) => {
    // qui dentro verrà fatto il caricamento
    if (error) {
        reject(error);
    }

    resolve(file);
  });
}
```

PROMISE

Per poter parallelizzare le task di caricamento possiamo creare una promise invocando `loadFile` per ogni path presente nell'array. Per fare ciò potremmo usare `Array.prototype.foreach`.

Quindi

```
paths.forEach(path => loadFile(path));
// O in maniera più concisa
paths.forEach(loadFile);
```

Come possiamo però eseguire delle azioni in seguito alla risoluzione di tutte le promise?

PROMISE

`Promise.all` è una funzione che riceve come argomento un array di *Promise* e restituisce una *Promise*. Verrà chiamata la funzione passata a `then` qualora tutte le promise passate passino in stato *fullfilled*, `catch` qualora anche solo una promise vada in stato *rejected*.

Quindi nel nostro esempio possiamo eseguire delle azioni in seguito al caricamento di tutti i file nel seguente modo.

```
const promiseArray = paths.map(path => loadFile(path));
// O in maniera più concisa
const promiseArray = paths.map(loadFile);

Promise
  .all(promiseArray)
  .then(files => {
    // Fai qualcosa con i risultati
    files.forEach(notifyUser)
  })
  .catch(error => {
    // Gestisci l'errore
  });
```