wantsome
The friendly IT Academy

Saptamana 13

Partea 1

Programare Front-End

1. JavaScript Promise

What's "Promise" ?

- reprezinta eventuala completare cu succes sau nu a unei operatii asincrone, alaturi de rezultatul acesteia
- un **`Promise`** este un proxy pentru o valoare care nu este neaparat cunoscuta in momentul initializarii acestuia
- permite specificarea unui handler care sa trateze fie cazul de eroare sau succes al unei operatii asincrone
- permite metodelor asincrone sa returneze valori in aceeasi modalitate precum metodele sincrone
  - in loc ca valoarea finala sa fie imediat returnata, metoda asincrona va returna un **`Promise`** care va oferi valoarea ce trebuie returnata la un moment dat in viitor

Promise Example

```
const promiseExample = new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('foo');
  }, 300);
});

promiseExample.then(function(value) {
  console.log(value);
  // expected output: "foo"
});

console.log(promiseExample);
// expected output: [object Promise]
```

Promise syntax

```
new Promise(executor);
```

`executor`

- o functie care accepta ca si argumente alte doua functii, **`resolve`** si **`reject,`** folosite pentru a finaliza cu **success** sau **fail** ***promise***-ul

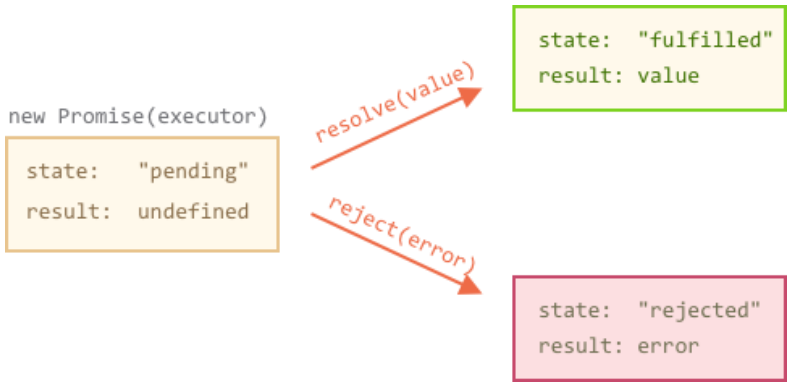
- in general, initiaza o operatie asincrona care odata ce este finalizata fie apeleaza **`resolve`** pentru a completa *promise*-ul cu succes, fie apeleaza **`reject`** in cazul in care apare o eroare; daca se “arunca” o eroare in corpul *promise*-ului, acesta va fi *rejected*, iar valoarea returnata va fi ignorata (**`return`**)

1.1 Promise States

Promise states

Un **`Promise`** se poate afla in una dintre urmatoarele stari:

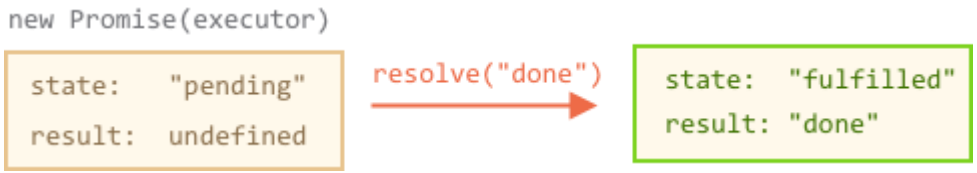
- ***pending***: stare initiala - Promise-ul nu este nici **completat**, nici **respins**
- ***fulfilled***: operatia a fost **completata** cu **succes**
- ***rejected***: operatia a **esuat** - a fost **respinsa**



Promise completion

```
let promise = new Promise(function(resolve, reject) {
  // the function is executed automatically when the promise is constructed

  // after 1 second signal that the job is done with the result "done"
  setTimeout(() => resolve("done"), 1000);
});
```



Promise rejection

```
let promise = new Promise(function(resolve, reject) {
  // after 1 second signal that the job is finished with an error
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});
```

1.1 Promise Consumers

Promise consumers - **then**

then

- cel mai des intalnit ***consumer***

```
promise.then(
  function(result) { /* handle a successful result */ },
  function(error) { /* handle an error */ }
);
```

Promise consumers - **then**

then - success

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => resolve("done!"), 1000);
});

// resolve runs the first function in .then
promise.then(
  result => alert(result), // shows "done!" after 1 second
  error => alert(error) // doesn't run
 );
```

Promise consumers - **then**

then - failure

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// reject runs the second function in .then
promise.then(
  result => alert(result), // doesn't run
  error => alert(error) // shows "Error: Whoops!" after 1 second
);
```

Promise consumers - **catch**

catch

- reactioneaza doar in cazul erorilor

```
let promise = new Promise((resolve, reject) => {
                setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// .catch(f) is the same as promise.then(null, f)
promise.catch(alert); // shows "Error: Whoops!" after 1 second
```

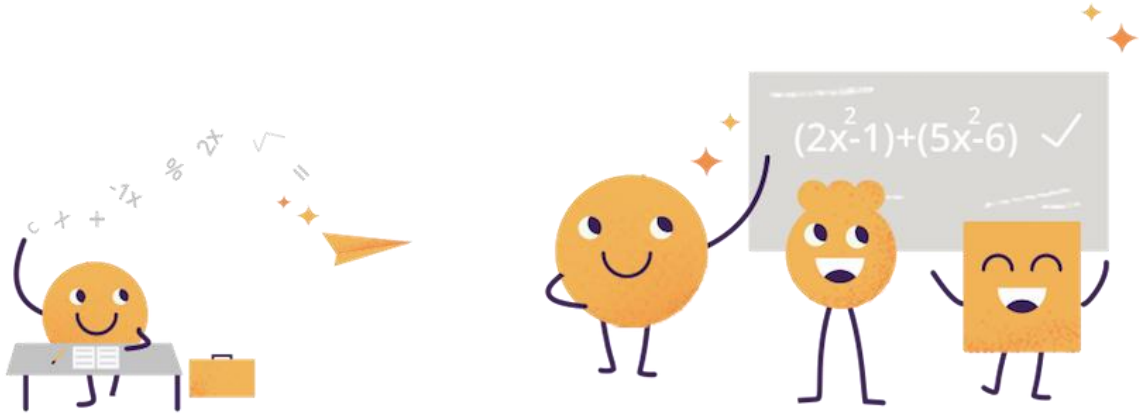
Promise consumers - **finally**

finally

- reactioneaza mereu atunci cand ***promise***-ul atunci cand operatia din cadrul ***promise***-ului a fost terminata, fie cu succes sau nu
- nu primeste argumente
- **`finally`** poate fi un handler potrivit pentru situatiile in care se intentioneaza o operatiun de “clean up”
    - ex: ascunderea unui ***loader*** avand in vedere ca nu mai este necesara afisarea lui, indiferent de rezultat

```
new Promise((resolve, reject) => {
  /* do something that takes time, and then call resolve/reject */
})
      // runs when the promise is settled, doesn't matter successfully or not
      .finally(() => stop loading indicator)
      .then(result => show result, err => show error)
```

PRACTICE: **JavaScript Promise**

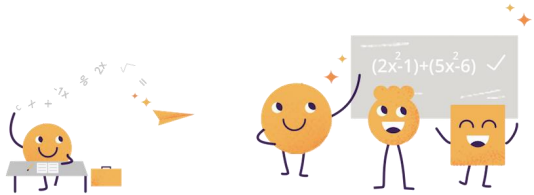


PRACTICE: **JavaScript Promise**

Cerinte:

1. Creati o functie care primeste un numar ca si argument si returneaza un **`Promise`** care testeaza daca valoarea este mai mica sau mai mare decat 10 - se va face **`reject`** / **`resolve`** in functie de valoarea de adevar a conditiei de comparatie.

   Apelati functia si folositi consumatorii **`then`** si **`catch`** pentru a trata ambele cazuri.

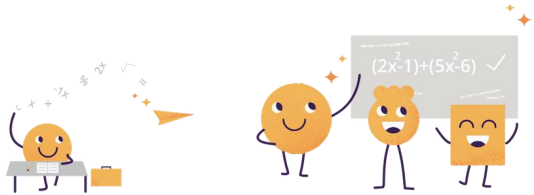


PRACTICE: **JavaScript Promise**

Cerinte:

2. Creati o functie care primeste un ***string*** ca si argument si returneaza un `Promise` care testeaza daca acesta contine sau nu cuvantul “promise” - se va face `reject` / `resolve` in functie de valoarea de adevar a conditiei specificate.

   Apelati functia si folositi consumatorii `then` si `catch` pentru a trata ambele cazuri.

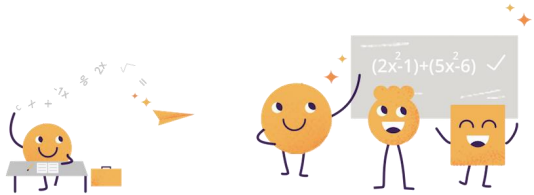


PRACTICE: **JavaScript Promise**

Cerinte:

3. Creati o functie care primeste un singur parametru si returneaza un `Promise`. Folosind `setTimeout`, dupa 500ms, acest `Promise` fie va face `resolve`, fie va face `reject`, in functie de urmatoarele cazuri: daca input-ul este un ***string***, `Promise`-ul se va rezolva cu rezultatul avand valoarea ***string***-ului ***uppercased***; daca input-ul nu este un ***string***, `Promise`-ul va face `reject` cu rezultatul avand valoarea ***string***-ului fara nicio modificare.

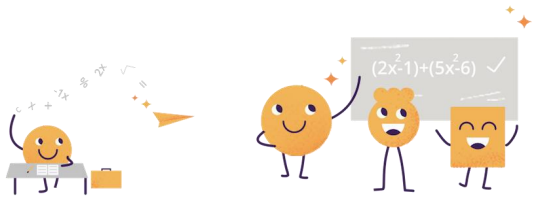
   Apelati functia si folositi consumatorii `then` si `catch` pentru a trata ambele cazuri.



PRACTICE: **JavaScript Promise**

Cerinte:

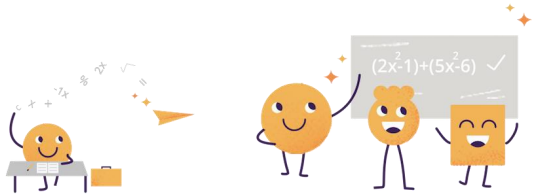
4. Creati doua functii care folosesc **Promises** pentru a putea face o ***inlantuire*** (**Promise chain**). Prima functie, `capitalizeWords()`, va primi ca si argument un ***array*** de cuvinte si va aplica o operatiune de ***capitalize*** pe acestea. A doua functie, `sortWords()`, va primi ca si argument rezultatul primului `Promise` si va sorta cuvintele in ordine alfabetica. In cazul in care ***array***-ul initial contine un element cu o valoare diferita de tipul ***string***, se va face `reject`.



wantsome
The friendly IT Academy

PRACTICE: **JavaScript Promise**

https://developers.google.com/web/fundamentals/primers/promises#promisifying_xmlhttprequest



PRACTICE: **JavaScript Promise**

Cerinte:

5. Implementati functionalitatea anterior prezentata si folositi-o pentru a apela mai multe ***API***-uri externe, la alegere (cel putin 3)