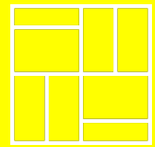
PROMISES

Uma Promise (de "promessa") representa uma operação que ainda não foi completada.

É um objeto usado para processamento assíncrono e representa um valor que pode estar disponível agora, no futuro ou nunca.



Sincronismo vs Assincronismo

SINCRONISMO

Normalmente, o código de um programa é executado de forma direta, com uma coisa a acontecer de cada vez.

Se uma função depende do resultado de outra função, ela tem que esperar o retorno do resultado, e até que isso aconteça, o programa inteiro praticamente para de funcionar na perspectiva do utilizador.

É importante relembrar que o javascript corre apenas usando uma thread, daí que bloquear um programa tem consequências nefastas na experiência do utilizador.

O javascript é apenas um dos concorrentes à thread, na qual ocorrem processos de atualização de estilos, parsing de DOM ou escutar e reagir a eventos do utilizador.

ASSINCRONISMO

Nós, podemos dizer que somos multi-thread. Com mais ou menos facilidade nós conseguimos executar mais de uma ação ao mesmo tempo, mas também temos as nossas tarefas bloqueantes, como espirrar por exemplo.

Para isso foram criadas maneiras de o utilizador ter responsividade enquanto o programa corre no “background”.

Estados de uma Promise

Uma Promise está num destes estados:

PENDING
: Promise pendente: estado inicial, que não foi realizada nem rejeitada.

FULFILLED
: Promise realizada: sucesso na operação.

REJECTED
: Promise rejeitada: falha na operação.

Uma Promise pendente pode se tornar realizada com um valor ou rejeitada por um motivo (erro).

Criar uma Promise

Exemplo de uma Promise:

```
var promise = new Promise(function (resolve, reject) {
    try {
        resolve('Tudo funcionou!');
    } catch (e) {
        reject(e);
    }
});
```

RESOLVE

Função chamada quando queremos retornar o resultado final.

REJECT

Função chamada quando queremos retornar um erro.

Consumir uma Promise

Existem 3 formas de consumir uma Promise. As três são bastante similares e não há uma que seja mais correcta do que outra:

BLOCO .then()

```
promise
.then(function(result){
        console.log(result);
}, function(error){
        console.log(error);
})
```

Neste caso estamos a passar dois parâmetros para a função *.then()*.
O primeiro parâmetro refere-se a uma Promise realizada; o segundo parâmetro refere-se a uma Promise rejeitada.

Consumir uma Promise

BLOCO .then().catch()

```
promise
.then(function(result){
        console.log(result);
})
.catch(function(error){
        console.log(error);
})
```

Neste caso estamos a passar apenas o primeiro parâmetro para a função *.then()*, que se refere a uma Promise realizada.
Para lidarmos com uma Promise rejeitada usamos o método *.catch()* após o método *.then()*.

Consumir uma Promise

BLOCO .then() E BLOCO .catch()

```
promise.then(function(result){
        console.log(result);
});
```

```
promise.catch(function(error){
        console.log(error);
});
```

Neste caso estamos a chamar a função *.then()* e *catch()* separadamente.
Funcionalmente mantém tudo igual: *.then()* refere-se a uma Promise realizada e *.catch()* a uma Promise rejeitada.

Exercício

PROMISES

- Cria um ficheiro html;
- Cria uma tag script para o código js;
- Cria uma Promise com as seguintes características:
  - Ter uma função que faça a soma de 2 valores, recebidos por parâmetros;
  - Se a soma for par, deverá ficar resolvida;
  - Se a soma for ímpar, deverá ser rejeitada.

Métodos de Promise

PROMISE.RESOLVE(valor)

Retorna um objeto Promise que é resolvido com o valor passado. Se o valor for uma promise, o objeto será o resultado da chamada Promise.resolve; do contrário a promise será realizada com o valor.

```
Promise.resolve(5);
var promise = new Promise(function(resolve){ resolve('texto') });
Promise.resolve('Teste de resolve').then(function (data) {});
```

PROMISE.REJECT(motivo)

Retorna um objeto Promise que é rejeitada com um dado motivo.

```
var p1 = Promise.reject('Teste de reject);
p1.catch(function (motivo) {});

Promise.reject('Teste de reject').then().catch(function (motivo) {});
```

Métodos de Promise

PROMISE.ALL(lista)

Retorna uma única Promise que resolve quando todas as promises no argumento iterável forem resolvidas ou quando o iterável passado como argumento não contém promises.
É rejeitado com o motivo da primeira promise que foi rejeitada.

```
Promise.all([promise1, promise2, promise3]).then((result) => {
    console.log(result);
});
```

PROMISE.RACE(lista)

Retorna uma Promise que resolve ou rejeita assim que uma das Promise no iterável resolver ou rejeitar, com o valor ou razão daquela promise.

```
Promise.race([promise1, promise2]).then((result) => {
    console.log(result); // resultado da promise mais rápida
});
```

Métodos de Promise

PROMISE.ALLSETTLED(lista)

Retorna uma promise que é resolvida quando todas as promises no argumento lista forem resolvidas ou rejeitadas.
Este método pode ser útil para agregar resultados de múltiplas promises.
Todos os valores seguem a seguinte estrutura:

Fullfilled: *{ status: 'fulfilled', value: 99}*
Rejected: *{status: 'rejected', reason: Error: an error}*

```
var promise = new Promise(function (resolve) {
        resolve('done');
});
var promise2 = new Promise(function (resolve, reject) {
        reject('failed');
});

Promise.allSettled([promise, promise2]).then(function (data) {
        console.log(data);
});
```

Async/await

O padrão async/await é um recurso que permite que uma função assíncrona e sem bloqueio seja estruturada de maneira semelhante a uma função síncrona comum.
Quando a função assíncrona retorna um valor, a Promise será resolvida com o valor retornado. Quando a função assíncrona lança uma exceção ou algum valor, a Promise será rejeitada com o valor lançado.

```
function resolveAfter2Seconds() {
        return new Promise((resolve) => {
                setTimeout(() => { resolve('resolved') }, 2000);
        });
}

async function asyncCall() {
        console.log('calling');
        const result = await resolveAfter2Seconds();
        console.log(result); // "resolved"
}
```

Uma função assíncrona pode conter uma expressão await, que pausa a execução da função assíncrona e espera pela resolução da Promise passada, e depois retoma a execução da função assíncrona e retorna o valor resolvido.

Exercício

PROMISES

- Cria um ficheiro html;
- Cria uma tag script para o código js;
- É o mesmo exercício da Promise, mas usando async/await.
- Cria uma função assíncrona com as seguintes características:
  - Ter uma função que faça a soma de 2 valores, recebidos por parâmetros;
  - Se a soma for par, deverá ficar resolvida;
  - Se a soma for ímpar, deverá ser rejeitada.