Promises

Uma promise é um objeto retornado por uma função assíncrona, contendo o estado atual da operação.

Ele pode ser criado pelo programador ou ser resultado de uma operação utilizando APIs nativas ou bibliotecas de terceiros no JavaScript.

Como operação assíncrona não é finalizada instantaneamente, o objeto da promise oferece métodos/funções para tratar o possível sucesso ou falha da operação no momento da sua finalização.

Promises – Exemplo 1

Para chamar uma api externa no JavaScript podemos utilizar o método fetch, que possui uma promise como retorno.

```
const promessa = fetch("https://mdn.github.io/learning-
area/javascript/apis/fetching-data/can-store/products.json")
console.log(promessa)
```

A operação de fetch inicialmente retorna: Promise {<pending>}, nos dizendo que no objeto Promise, a propriedade state no momento em que foi impresso tinha valor "pending" ("pendente").

O estado "pending" significa que a busca pela resposta ainda está ocorrendo.

Promises – Exemplo 1

Para conseguirmos um retorno após a conclusão da Promise, utilizamos a função then(), para quando a operação de busca tiver conclusão, recebermos um objeto com a resposta da API.

```
fetch("link de uma API")
.then(resposta => console.log(resposta))
```

Promises – Exemplo 1

No retorno da promessa, teremos uma propriedade body.
Para lermos o body, teremos que usar outra função: json(), que também é assíncrona, dessa forma teremos outra promessa:

```
fetch(“link-de-uma-API")
.then(respostaFetch => {
    return respostaFetch.json()
})
.then((respostaJson) => console.log(respostaJson))
```

Dessa forma, no primeiro then, iremos utilizar a resposta da primeira promessa (retorno de fetch) e dando um retorno com a função resposta.json() que é outra promessa. Com isso, teremos o segundo then que possuirá o resultado da segunda promessa.

Promises – Exemplo 1

Podemos deixar o return implícito para simplificar

```
fetch(“link-de-uma-API")
.then(respostaFetch => {
    return respostaFetch.json()
})
.then((respostaJson) => console.log(respostaJson))
```

```
fetch("link de uma API")
.then(resposta => resposta.json())
.then(respostaJson => console.log(respostaJson))
```

Promises – Exemplo 1

A API fetch() pode lançar um erro por vários motivos (por exemplo, porque não havia conexão de rede ou a API não foi encontrada).

Para lidarmos com esses erros, utilizaremos uma função além do then(), que é o catch(), servindo para realizarmos uma tratativa quando a operação assíncrona possui falha.

```
fetch("link de uma API")
.then(resposta => resposta.json())
.then(respostaJson => console.log(respostaJson))
.catch(err => console.error(`Erro no fetch ${JSON.stringify(err)}`))
```

Promises – Exemplo 1

Caso seja necessário dar um retorno após o then e o também o catch, independente de qual ocorrer, podemos utilizar a função finally ao final:

```
fetch("link de uma API")
.then(resposta => resposta.json())
.then(respostaJson => console.log(respostaJson))
.catch(err => console.error(`Erro no fetch ${JSON.stringify(err)}`))
.finally(() => console.log('Resposta final independente'))
```

Possíveis estados de uma Promise

pending: Promise foi criada e ainda não foi concluída.

fulfilled: a função assíncrona foi concluída com sucesso, podendo acionar a função then()

rejected: a função assíncrona falhou, podendo acionar a função catch()

settled: completa, independente de fulfilled ou rejected.

Async await

Async await é uma maneira mais legível de lidar com Promises.
Podemos usá-lo em uma função que utiliza o fetch, utilizando o termo async function em vez de só function, dessa forma é permitido receber os valores das promises sem o uso do then, utilizando a palavra await no lugar.

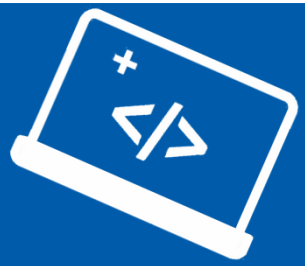
```
async function minhaFuncao() {
        const retornoFetch = await fetch("Link-API")
        const retornoJson = await retornoFetch.json()
        console.log(retornoJson)
}
minhaFuncao();
```

Async await

Para capturarmos erros, utilizamos o catch, porém precisamos incluir todo o código das Promises dentro de um bloco chamado try, que conterá a execução comum (caso não hajam erros)

```
async function minhaFuncao() {
    try {
        const retornoFetch = await fetch("Link-API")
        const retornoJson = await retornoFetch.json()
        console.log(retornoJson)
    } catch (error) => {
        console.error("Ocorreu um erro:", error))
    }
}
minhaFuncao();
```

Atividades (promises e async await)



- Use o fetch para buscar a lista de usuários (https://jsonplaceholder.typicode.com/users) e, em seguida, use o método map para armazenar os nomes dos usuários em um Array.

- Utilize o fetch para buscar uma lista de tarefas na API https://jsonplaceholder.typicode.com/todos
  Após isso, utilize o método filter para encontrar as tarefas incompletas. Em seguida, use o método length para contar quantas tarefas estão incompletas.

- Busque a lista de álbuns de fotos de um usuário na API https://jsonplaceholder.typicode.com/albums
  Use o método filter para encontrar os álbuns de um usuário de sua escolha pelo userId e depois map para exibir o título de cada álbum.

Métodos de Promises

Promise.all(): Quando é necessário que várias promises sejam cumpridas, mas elas não dependem umas das outras. Inicia todas juntas após o cumprimento de todas retorna um array com os retornos das promises em uma única promise (precisamos usar o then para acessar).

```
const fetchPromise1 = fetch("API 1")
const fetchPromise2 = fetch("API 2")
const fetchPromise3 = fetch("API 3")
Promise.all([fetchPromise1, fetchPromise2, fetchPromise3])
.then(respostas => console.log(respostas))
.catch(erro => console.error(`Falha ao buscar: ${erro}`))
```

Se alguma das promessas estiver com erro, o retorno será o catch com o erro da primeira promessa com erro.

Métodos de Promises

Promise.allSettled(): Semelhante ao Promise.all, porém ele nunca será rejeitado (catch).
No Promise.all, quando uma das promessas é rejeitada, a função catch é acionada, já no Promise.allSettled, é armazenado que o status da promessa foi rejeitada, mas o retorno continua sendo pela função then, no formato de array da seguinte forma:

```
[
        { status: "fulfilled", value: "valor da promessa 1"},
        { status: "fulfilled", value: "valor da promessa 2"},
        { status: "rejected", reason: "erro da promessa 3"}
]
```

Métodos de Promises

Promise.allSettled(): A sintaxe é semelhante a do Promise.all:

```
const fetchPromise1 = fetch("API 1")
const fetchPromise2 = fetch("API 2")
const fetchPromise3 = fetch("API 3")
Promise.allSettled([fetchPromise1, fetchPromise2, fetchPromise3])
.then(respostas => console.log(respostas))
.catch(erro => console.error(`Falha ao buscar: ${erro}`))
```

Métodos de Promises

Promise.any(): Quando é necessário captar o resultado de uma promessa qualquer de um conjunto, assim que houver a conclusão de uma promessa, irá retornar essa única conclusão como then ou catch

```
const fetchPromise1 = fetch("API 1")
const fetchPromise2 = fetch("API 2")
const fetchPromise3 = fetch("API 3")
Promise.any([fetchPromise1, fetchPromise2, fetchPromise3])
.then(respostas => console.log(respostas))
.catch(erro => console.error(`Falha ao buscar: ${erro}`))
```