JavaScript Impressionador - Módulo 9 - JavaScript Assíncrono

Exercícios

Callbacks:

Exercício 1: Callback Simples
Escreva uma função ***imprimirMensagem*** que aceita uma mensagem e uma função de retorno de chamada (callback). A função deve imprimir a mensagem no console e, em seguida, invocar a função de retorno de chamada.

```
//resposta:
const imprimirMensagem = (mensagem, callback) => {
  console.log(mensagem);
  callback();
};

imprimirMensagem('mensagem de teste', () =>
  console.log('essa mensagem é resultado de um callback executado')
);
```

Exercício 2: Leitura de Arquivo com Callback
Escreva uma função ***lerArquivo*** que aceita um nome de arquivo e uma função de retorno de chamada (callback). A função deve ler o conteúdo do arquivo e passá-lo para a função de retorno de chamada. Por fim, chame a função lerArquivo() usando um callback que faça o conteúdo do arquivo ser exibido no console.
Dica: disponibilizamos um arquivo de texto para esse exercício. Coloque-o na mesma pasta do seu código JavaScript pra facilitar a resolução.

Obs: Para esse exercício, aqui vai um trecho de código que mostra como disponibilizar no seu código as funcionalidades de uso de arquivos com JavaScript/NodeJS. O Trecho abaixo deve ser utilizado para a leitura do arquivo.

```
const fs = require('fs');

/*o método readFile lê o conteúdo de um arquivo identificado
por filename (coloque o nome do arquivo e o caminho dele,
 ex: './textoExercicio2.txt' ) e entrega esse conteúdo dentro do objeto "data"
 o segundo parâmetro do método readFile, aqui definido como "utf-8", informa que o
conteúdo do arquivo deve ser entendido como texto.*/

fs.readFile(filename, 'utf-8', (err, data) => {
  if (err) {
    //Aqui escrevemos um tratamento para um erro na tentativa de leitura do aquivo
    return;
  }
```

```
  //Aqui escrevemos o que deve ser feito com o objeto "data"
});
```

```
//resposta:
const fs = require('fs');

function lerArquivo(arquivo, callback) {
  fs.readFile(arquivo, 'utf-8', (err, data) => {
    if (err) {
      console.error('erro ao tentar acessar o arquivo: ', err.message);
      return;
    }
    callback(data);
  });
}

lerArquivo('./textoExercicio2.txt', (conteudo) => console.log(conteudo));
```

Promises:

Exercício 3: Conceitos Básicos de Promises

Crie uma nova Promise que seja resolvida após um atraso de 1 segundo e retorne a string "Promise resolvida". Use .then() para lidar com o valor resolvido e imprimir no console.

```
//resposta:
const p1 = new Promise((resolve) => {
  setTimeout(() => {
    resolve('Promise resolvida');
  }, 1000);
});

p1.then(console.log);
```

Exercício 4: Encadeamento de Promises

Crie duas Promises: uma que seja resolvida para "Olá" após um atraso de 1 segundo e outra que seja resolvida para "Mundo" após um atraso de 2 segundos. Encadeie essas Promises usando .then() para imprimir "Olá, Mundo" no console.

```
//resposta:
const p1 = new Promise((resolve) => {
  setTimeout(() => resolve('Olá'), 1000);
});

const p2 = new Promise((resolve) => {
  setTimeout(() => resolve('Mundo'), 2000);
```

```
});

p1.then((result1) =>
  p2.then((result2) => console.log(`${result1}, ${result2}`))
);
```

Exercício 5: Tratamento de Erros com Promises

Crie uma Promise que seja rejeitada com um erro após um atraso de 1 segundo. Use .catch() para lidar com o erro e imprimir no console.

```
//resposta:
const p = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject('Falha ao cumprir sua "Promise"');
  });
});

p.then(
  console.log /*.then aqui não é necessário para a resolução do exercício, porém alguém que espera uma promise deve se preparar para o melhor e o pior cenário.*/
).catch(console.error /* ou (erro) => console.error(erro)*/);
```

Exercício 6: Promise All

Crie três Promises: uma que seja resolvida para "Um" após um atraso de 1 segundo, outra que seja resolvida para "Dois" após um atraso de 2 segundos e uma terceira que seja resolvida para "Três" após um atraso de 3 segundos. Use Promise.all() para aguardar a resolução de todas as Promises e, em seguida, imprima o array de valores resolvidos no console.

```
//resposta:
const p1 = new Promise((resolve) => {
  setTimeout(() => {
    resolve('Um');
  }, 1000);
});

const p2 = new Promise((resolve) => {
  setTimeout(() => {
    resolve('Dois');
  }, 2000);
});

const p3 = new Promise((resolve) => {
  setTimeout(() => {
    resolve('Três');
  }, 3000);
});
```

```
const results = Promise.all([p1, p2, p3]);

results.then((data) => console.log(data));
```

Async/Await:

Exercício 7: Async/Await Básico

Escreva uma função assíncrona buscarDados que usa await para simular a busca de dados de uma API após um atraso de 2 segundos. Imprima os dados buscados no console.

```
//resposta:
function pegarDadosNaAPI(nomeDoUsuario) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({
        name: nomeDoUsuario,
        age: Math.floor(80 * Math.random()),
        favorite_programming_language: 'JavaScript',
      });
    }, 2000);
  });
}

async function buscarDados(nomeDoUsuario) {
  const resposta = await pegarDadosNaAPI(nomeDoUsuario);
  console.log(resposta);
}

buscarDados('Daniel');
```

Exercício 8: Tratamento de Erros com Async/Await

Escreva uma função assíncrona obterDados que usa await para buscar dados de uma API após um atraso de 1 segundo. Se ocorrer um erro, capture-o e imprima uma mensagem de erro no console.

```
//resposta:
function buscarDados() {
  return new Promise((resolve, reject) => {
    const numeroAleatorioEntre0e1 = Math.random();
    if (numeroAleatorioEntre0e1 > 0.5) {
      resolve({
        color: '#51f',
        shape: 'rectangle',
      }); /*Aqui eu escrevi um objeto com dados quaisquer a fim de imitar uma API
genérica*/
    }
    reject(new Error('Error obtaining data'));
  });
}
```

```
async function obterDados() {
  try {
    const data = await buscarDados();
    console.log(data);
  } catch (error) {
    console.error(error.message);
  }
}

obterDados();
```

Exercício 9: Async/Await com Promise.all()

Escreva uma função assíncrona buscarMultiplosDados que usa Promise.all() e await para buscar dados de várias APIs simultaneamente. Imprima o array de dados buscados no console.

```
//resposta:
async function buscarMultiplosDados() {
  const result = await Promise.all([
    new Promise((resolve) => {
      setTimeout(() => {
        resolve('dados1');
      }, 1000);
    }),
    new Promise((resolve) => {
      setTimeout(() => {
        resolve('dados2');
      }, 1500);
    }),
    new Promise((resolve) => {
      setTimeout(() => {
        resolve('dados3');
      }, 2000);
    }),
  ]);

  console.log(result);
}

buscarMultiplosDados();
```

Exercício 10: Async/Await com Tratamento de Erros

Escreva uma função assíncrona obterDadosComFallback que usa try...catch e await para buscar dados de uma API. Se ocorrer um erro, utilize um valor de fallback e imprima-o no console.

```
//resposta:
const times = [
  'Flamengo',
  'Fluminense',
  'Vasco',
```

```
  'Botafogo',
  'Corinthians',
  'Santos',
  'Internacional',
  'Grêmio',
  'São Paulo',
  'Palmeiras',
  'Cruzeiro',
  'Grêmio',
  'Atlético MG',
];

function descobrirCampeaoBrasileiro2023() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const numeroAleatorioEntre0e1 = Math.random();
      if (numeroAleatorioEntre0e1 < 0.2) {
        resolve(
          Math.floor(times.length * Math.random())
        ); /*Aqui eu gerei um índice aleatório pro meu array de times*/
      }
      reject(new Error('erro buscando o time campeão'));
    }, 1500); /*Aqui o setTimeout foi colocado pra simular a espera natural que
existe na comunicação com outros sistemas*/
  });
}

async function obterDadosComFallback() {
  try {
    const indice = await descobrirCampeaoBrasileiro2023();
    console.log(
      `O time campeão do Campeonato Brasileiro de 2023 será o ${times[indice]}`
    );
  } catch (error) {
    console.error(error.message);
    console.log(
      'Já que não conseguimos adivinhar o time, vamos fazer a aposta mais lógica.
O campeão será o Fluminense!!!'
    );
  }
}

obterDadosComFallback();
```