```
//DESVANTANGEM DE UMA CALLBACK HELL SOBRE UM PROMISE:

//Para mostrarmos como um promise é muito mais vantajoso do que utilizar callbacks uma encadeada sobre a outra somente para trazer um resultado, é muito mais vantajosa.
//Abaixo temos 3 URLS que trazem detalhes sobre 3 turmas de uma escola, as turmas A, B e C:

//Endereço do arquivo JSON para Turma A: http://files.cod3r.com.br/curso-js/turmaA.json
//Endereço do arquivo JSON para Turma A: http://files.cod3r.com.br/curso-js/turmaB.json
//Endereço do arquivo JSON para Turma A: http://files.cod3r.com.br/curso-js/turmaC.json

//Queremos gerar uma função que retorne somente os nomes dos alunos de cada turma em um único array...

//EXEMPLO COM CALLBACKS (Callback Hell>:C):
//Primeiro temos que importar uma biblioteca para conexão com o arquivo por através de protocolo "http"...
const http = require('http') //Estamos usando o módulo interno "http" do node invez de usar o "axios" por que o axios já é baseado em promise, então não faria sentido fazer um exercício com promise numa biblioteca que já utiliza "promise", por isso estamos usando a biblioteca "http" que é mais genérica.

//Abaixo temos a função que captura o arquivo JSON de acordo com a letra da turma...
const getTurma = (letra, callback) => { //Já que o único parâmetro que diferencia as URLS das turmas é uma letra, nossa função vai receber uma letra que será fundida a url de busca por através de um template string. E como 2º parâmetro, vai receber uma callback para o tratamento dos valores posteriormente...
    const url = `http://files.cod3r.com.br/curso-js/turma${letra}.json` //Aqui juntamos a letra com a URL...
    http.get(url, res => { //Usando o método get da biblioteca "http" buscamos uma resolução para a nossa url e também executamos uma função callback que recebe como parâmetro os dados que estiverem no endereço da URL em formato de memória em buffer...
        let resultado = '' //Temos uma variável que vai receber os dados...

        res.on('data', dados => { //a variável vai receber os dados JSON dentro de um array com os dados da URL graças ao "on('data')""
            resultado += dados
        })

        res.on('end', () => { //Aqui temos um evento que finaliza a execução por chamar a callback que irá tratar os dados recebidos, mas antes irá transformar esses dados para o formato JSON...
            callback(JSON.parse(resultado))
        })
```

```
    })
}

let nomes = [] //Aqui temos a variável que vai receber somente os nomes de todos os alunos de todas as turmas...
getTurma('A', alunos => { //Perceba que chamamos a função "getTurma" primeiramente passando como parâmetro a letra "A", para pegar os alunos somente da turma "A", e usamos uma callback que mapeia somente os nomes e os atribuí a variável nomes...
    nomes = nomes.concat(alunos.map(a => `A: ${a.nome}`))
    getTurma('B', alunos => { //Em seguida temos outra chamada para a função "getTurma" dentro da chamada da para a turma "A", mas agora chamando a turma "B", só para não ter que chamar a turma B numa segunda chamada separada...
        nomes = nomes.concat(alunos.map(a => `B: ${a.nome}`))
        getTurma('C', alunos => { //E outra para a turma "C"
            nomes = nomes.concat(alunos.map(a => `C: ${a.nome}`))
            console.log(nomes) //Como podemos ver, todas as chamadas são atribuídas para a mesma variável "nomes"...
        })
    })
})



//REFATORANDO A FUNÇÃO ACIMA USANDO PROMISE USANDO PROMISE:
const getTurmaPromise = letra => { //Perceba que agora, a função recebe somente 1 parâmetro que é a letra correspondente a turma...
    const url = `http://files.cod3r.com.br/curso-js/turma${letra}.json` //Aqui juntamos a letra com a URL...
    return new Promise((resolve, reject) => { //Perceba que agora estamos gerando um promise no retorno da função...
        http.get(url, res => {
            let resultado = '' //Temos uma variável que vai receber os dados...

            res.on('data', dados => { //a variável vai receber os dados JSON dentro de um array com os dados da URL graças ao "on('data')""
                resultado += dados
            })

            res.on('end', () => { //o evento "end" só ativado quando o evento anterior termina, depois que resultado colhe todos os dados, ele é ativado
                try { //Mas nessa vez estamos usando um try e catch no evento "end", onde, caso o evento dê certo ele transforma os dados em JSON, caso dê errado ele retorna a mensagem de erro por através de catch...
```

```
                    resolve(JSON.parse(resultado))
                } catch(e){
                    reject(e)
                }
            })
        })
    })

}

Promise.all([getTurmaPromise('A'), getTurmaPromise('B'), getTurmaPromise('C')]) //Usando o objeto "promise" temos o método "all()", esse método recebe um array de funções para serem executadas, ele vai executar todas as funções e só quando terminar as execuções ele vai dar seguimento ao código...
    .then(turmas => [].concat(...turmas)) //Perceba que o then() pega todos os resultados das execuções e os concatena num array...
    .then(alunos => alunos.map(alunos => alunos.nome)) //O array com os resultados é mapeado e são recolhidos só os nomes em um novo array...
    .then(nomes => console.log('\nRESULTADO COM PROMISSE: VEJA QUE É IGUAL AO RESULTADO DA CALLBACK HELL, E É ATÉ EXECUTADA ANTES POR SER MAIS LEVE...', nomes)) //Finalmente o array somente com os nomes é devolvido...

//Veja o resultado se der erro...
getTurmaPromise('D').catch(e => console.log('\nERRO!!!: ', e.message)) //Usamos a letra para um turma que não existe...
```

RESULTADO NO CONSOLE...

```
[Running] node "c:\Users\Almoxarifado\Documents\javascript\arquivos_das_aulas\156-Promise_Vendo_a_Vantagem_do_Promise_Sobre_CallBacks_Comuns.js"

RESULTADO COM PROMISSE: VEJA QUE É IGUAL AO RESULTADO DA CALLBACK HELL, E É ATÉ EXECUTADA ANTES POR SER MAIS LEVE... [
  'Kellia',   'Hi',     'Inge',
  'Myrle',    'Doreen', 'Pennie',
  'Faye',     'Leena',  'Taylor',
  'Juieta',   'Rossie', 'Mary',
  'Dionysus', 'Myca',   'Sharlene',
  'Meghan',   'Perice', 'Micheil',
  'Nat',      'Bone',   'Kellina',
  'Barrie',   'Darda',  'Rainer',
```

```
  'Joan',     'Kasper', 'Sammie',
  'Scott',    'Kiel',   'Dell'
]

ERRO!!!:  Unexpected token < in JSON at position 0
[
  'A: Kellia',   'A: Hi',     'A: Inge',
  'A: Myrle',    'A: Doreen', 'A: Pennie',
  'A: Faye',     'A: Leena',  'A: Taylor',
  'A: Juieta',   'B: Rossie', 'B: Mary',
  'B: Dionysus', 'B: Myca',   'B: Sharlene',
  'B: Meghan',   'B: Perice', 'B: Micheil',
  'B: Nat',      'B: Bone',   'C: Kellina',
  'C: Barrie',   'C: Darda',  'C: Rainer',
  'C: Joan',     'C: Kasper', 'C: Sammie',
  'C: Scott',    'C: Kiel',   'C: Dell'
]

[Done] exited with code=0 in 1.835 seconds
```