Manipulação de dados em arquivo

Nem todos os problemas consistem em entrada de dados fornecidas sequencialmente pelos usuários. Existem muitos problemas em que os dados a serem processados encontram-se armazenados em arquivos (de texto, por exemplo).

A linguagem Javascript fornece dois modos de leitura de dados em arquivo: modo **SÍNCRONO** e modo **ASSÍNCRONO**.

Modo síncrono

No modo síncrono, o arquivo é acessado e o programa interrompe seu processamento enquanto os dados estão sendo lidos. Esse é o modo típico de um diálogo por ligação telefônica convencional: enquanto um fala, o outro ouve e vice-versa.

Para leitura síncrona, Javascript disponibiliza a função pre-definida `readFileSync` . Essa função possui o caminho do arquivo como parâmetro de entrada e retorna o seu conteúdo.

[EXEMPLO] Programa que lê o conteúdo de um arquivo de texto, exibe esse conteúdo no terminal e, A SEGUIR, realiza a próxima operação programada (no exemplo, a soma de dois valores quaisquer).

```
1  //Bibliotecas para ler arquivos
2  const fs = require('fs')
3  const path = require('path')
4
5  //Especifica o nome do arquivo
6  const caminho = path.join(__dirname, 'dados.txt')
7
8  //Fica esperando a leitura do arquivo para avançar no código.
9  console.log('=== INICIO da leitura do arquivo...')
10 const conteudo = fs.readFileSync(caminho)
11 console.log(conteudo.toString())
12 console.log('=== FIM da leitura.\n')
13
14 console.log('=== INICIO de uma outra operação qualquer...')
15 const exec = (fn, a, b) => fn(a, b)
16 const somarNoTerminal = (x, y) => console.log(`Resultado: ${x + y}.`)
17 exec(somarNoTerminal, 56, 38)
18 console.log('=== FIM da outra operação.\n')
```

Modo assíncrono

Como visto acima, a execução de uma função precisa esperar o término da anterior. Mas em alguns casos, podemos ganhar tempo de execução realizando uma tarefa de maneira assíncrona, como acontece em um bate papo por mensageiros instantâneos (ex: Whatsapp).

Para leitura assíncrona, Javascript disponibiliza a função pre-definida `readFile`. Além do caminho do arquivo, essa função exige um segundo parâmetro de entrada: uma função responsável por realizar o processamento desejado assim que o conteúdo do arquivo tiver sido completamente lido. Ou seja, é como se mandássemos executar alguma coisa, mas essa coisa só será executada em um **momento futuro**. Existe uma nomenclatura especial para essa função que será executada em um momento futuro: **CALLBACK**.

[EXEMPLO] Programa que lê o conteúdo de um arquivo de texto no modo assíncrono, com *callback*.

```
1  //Bibliotecas Node para ler arquivos
2  const fs = require('fs')
3  const path = require('path')
4
5  const caminho = path.join(__dirname, 'dados.txt')
6
7  //Callback
8  const exibirConteudo = (_, conteudo) =>
9      console.log(conteudo.toString())
10
11 console.log('=== INICIO da leitura do arquivo...')
12 fs.readFile(caminho, exibirConteudo)
13 console.log('=== FIM da leitura.\n')
14
15 console.log('=== INICIO de uma outra operação qualquer...')
16 const exec = (fn, a, b) => fn(a, b)
17 const somarNoTerminal = (x, y) => console.log(`Resultado: ${x + y}.`)
18 exec(somarNoTerminal, 56, 38)
19 console.log('=== FIM da outra operação.\n')
```

Assim, a função `readFile` é uma função assíncrona que usa o *callback* `exibirConteudo` para que seja executado no momento (futuro) em que o conteúdo do arquivo `dados.txt` tiver sido completamente lido.