Atividade de Participação 3 - Módulos e Promises

Prof. Luiz Gustavo D. de O. Véras
Desenvolvimento de Sistemas Web (DSWI6)
Tecnologia em Análise e Desenvolvimento de Sistemas

Atividade 1. Considere o programa em javascript abaixo que faz cáculos estatíticos (stats) de média (*mean*) e desvio padrão (*standard deviation - stddev*).

```
// Funções de um possível módulo de estatística

const soma = (x, y) => x + y;
const quadrado = x => x * x;
function media(dados) {
    return dados.reduce(soma) / dados.length;
}

function desvioPadrão(dados) {
    let médiaCalculada = media(dados);
    return Math.sqrt(
        dados.map(x => x - médiaCalculada).map(quadrado).reduce(soma) /
(dados.length - 1)
    );
}

// Este código deverá se manter funcionando
console.log(media([1, 3, 5, 7, 9]));  // => 5
console.log(desvioPadrão([1, 3, 5, 7, 9]));  // => Math.sqrt(10)
```

Transforme todo o código do objeto stats em um módulo chamado `stats.js` e o importe em um script chamado `index.js` com as duas últimas linhas do código acima. Execute para verificar se funciona. Use a notação de módulos CommonJS ou ES6.

Atividade 2. Faça um programa em javascript que utilize os seguintes módulos:

- **express**: é um módulo de terceiros, e deve ser importado utilizando o comando `npm install express`
- **user.js**: um módulo local que deve exportar a classe User e funções como segue:

```
    class User {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
        printName() {
        console.log(this.name);
        }
```

```
        printAge() {
        console.log(this.name);
        }
    }
    function printName(user) {
    console.log(`User's name is ${user}`);
    }
    function printAge(age) {
    console.log(`User's age is ${age}`);
    }
```

- **server.js**: script principal. Deve importar a classe e as funções de user.js, importar o módulo express e criar um servidor web simples para realizar um `get`. O caminho `get` deve ser acessível para o caminho `localhost:8080/user_data` e receber como retorno a mensagem de texto `User's name is Amazing and User's age is 100`;

Depois, inicialize o servidor com o comando `node server.js` e realize um teste utilizando o navegador da sua preferência.

Atividade 3. Em Javascript, os *named imports* permitem que você importe funções, classes ou variáveis específicas de um módulo. Em vez de importar todo o módulo, você seleciona apenas o que precisa, como em `const { func1, func2 } = require('./funcoes.js');` no padrão CommonJS e `import { func1, func2 } from './funcoes.js';` no padrão ES6.

Com base nisso, crie um módulo chamado `conversor.js` que exporte as seguintes funções:

1. `quilogramaParaLibra(quilogramas)`: Recebe um valor em quilogramas e retorna o equivalente em libras (1 quilograma = 2,20462 libras).
2. `metroParaPe(metros)`: Recebe um valor em metros e retorna o equivalente em pés (1 metro = 3,28084 pés).
3. `celsiusParaFahrenheit(celsius)`: Recebe uma temperatura em graus Celsius e retorna o equivalente em graus Fahrenheit (fórmula: °F = (°C × 9/5) + 32).

Em seguida crie um arquivo `main.js` que importa usando **named imports** essas funções do módulo `conversor.js` e as utilize para realizar as seguintes conversões:

```
// main.js

// Importe as funções do módulo conversor.js
// Dica: Utilize a sintaxe de named imports

// Realize as seguintes conversões:
// 1. Converta 5 quilogramas para libras
// 2. Converta 10 metros para pés
// 3. Converta 25 graus Celsius para graus Fahrenheit

// Exiba os resultados no console.

// Saída esperada:
// Quilogramas para Libras: 5 kg = 11.0231 lb
```

```
// Metros para Pés: 10 m = 32.8084 ft
// Celsius para Fahrenheit: 25°C = 77°F
```

Atividade 4. Crie dois scripts no Node para ler um mesmo arquivo. Pode ser um arquivo html ou csv, por exemplo. Em cada script, leia e imprima o arquivo utilizando os modos “de uma só vez” (numa função async) e “em fluxo (stream)”, respectivamente. Execute cada script separadamente e perceba a diferença de tempo para a saída dos dados. Como dica, utilize as instruções abaixo para verificar a diferença de tempo. Como a leitura do stream é assíncrona, use algum evento para calcular o tempo final. Veja em https://nodejs.org/api/stream.html Obs: Quando ler um arquivo com readFileSync use o método split(“\n”) para separar em linhas o arquivo. Split retorna um array de Strings.

```
let init = Date.now();
// **** Computação *****
let end = Date.now();
console.log(end - init) //Saída em ms
```

Atividade 5. Utilize um dos códigos de exemplo de leitura de arquivo apresentados nos slidades da "Aula 3 - Módulos Javascript" e o transforme em módulo.

Atividade 6. Crie um programa javascript que receba como argumento no comando “node meuScript.js meuarquivo.txt”. O programa deve ler o arquivo linha por linha a apresentá-las no console.

Dicas: Pesquisa sobre o módulo “readline”. Você pode integrá-lo com o módulo “fs” para ler linha à linha.

Atividade 7. Utilize o exemplo do módulo local logger.js dos slides da aula "Aula 3 - Módulos Javascript" e incorpore debuglog e inspect da biblioteca “util” para tornar as mensagens coloridas. Configura da seguinte forma:

```
const {
    error
    warning,
    info
} = require("loggercolor.js");

error();
warning();
info();
```

NODE_DEV=INFO -> Mensagens em verde NODE_DEV=WARNING -> Mensagens em amarelo NODE_DEV=ERROR-> Mensagens em vermelho

Use o app.js a seguir para utilizar como base.

```
// app.js
const {
    error
```

```
    warning,
    info
} = require("loggercolor.js");

error();
warning();
info();
```