30 - Usando o Babel

1. Vamos começar inicializando um novo projeto npm:

```
npm init -y
```

2. Depois, precisamos instalar os módulos que compõem o Babel:

```
npm install --save-dev @babel/core @babel/cli @babel/preset-env
```

3. Com eles instalados, vamos copiar o conteúdo da resolução do exercício 14, onde utilizando diversos recursos modernos do javascript, para um arquivo “index.js”:

```
const average = (...numbers) => { const sum = numbers.reduce((accum, num) => accum + num, 0) return sum / numbers.length } console.log(`Média Simples: ${average(3, 6, 10, 9)}`) const weightedAverage = (...entries) => { const sum = entries.reduce((accum, { number, weight }) => accum + (number * weight), 0) const weightSum = entries.reduce((accum, entry) => accum + entry.weight, 0) return sum / weightSum } console.log(`Média Ponderada: ${weightedAverage( { number: 9, weight: 3 }, { number: 7, weight: 1 }, { number: 10, weight: 1 }, )}`) const median = (...numbers) => { const orderedNumbers = [...numbers].sort((a, b) => a - b) const middle = Math.floor(orderedNumbers.length / 2) if (orderedNumbers.length % 2 !== 0) { return orderedNumbers[middle] } const firstMedian = orderedNumbers[middle - 1] const secondMedian = orderedNumbers[middle] return average(firstMedian, secondMedian) } console.log(`Mediana: ${median(2, 5, 99, 4, 42, 7)}`) console.log(`Mediana: ${median(15, 14, 8, 7, 3)}`) const mode = (...numbers) => { const quantities = numbers.map(number => [ number, numbers.filter(n => number === n).length ]) quantities.sort((a, b) => b[1] - a[1]) return quantities[0][0] } console.log(`Moda: ${mode(1, 1, 5, 4, 9, 7, 4, 3, 5, 2, 4, 0, 4)}`)
```

4. Agora que temos esse arquivo, podemos executar o comando do Babel e ver que ele irá compilar o nosso código e exibir o resultado no próprio terminal:

   Obs.: Repare que aqui ele não fez nenhuma conversão de compatibilidade, pois nós ainda não a especificamos. Ele apenas fez correções de código, como a inclusão do ; no fim de cada linha e outros pequenos detalhes que podem gerar algum problema para o nosso código.

```
npx babel index.js
```

5. Além de exibir o resultado no terminal, o que não é muito útil, podemos salvar esse resultado em um arquivo com a flag --out-dir:

```
npx babel index.js --out-file dist.js
```

6. Agora vamos ver como podemos realmente deixar o nosso código compatível com o ES5. Vamos utilizar para isso o preset-env. No Babel, presets são configurações prontas para casos de uso comuns, o que nos poupa bastante trabalho de configuração. O @babel/preset-env que nós instalamos é um preset que fará essa conversão para ES5 por padrão, mas que também é altamente configurável.

   Por enquanto vamos utilizá-lo da forma padrão. Execute o comando passando a flag --presets:

   Obs.: Repare que todos aqueles recursos modernos que utilizamos foram modificados para serem compatíveis com as versões mais antigas.

```
npx babel index.js --out-file dist.js --presets=@babel/preset-env
```

7. Além de usar o Babel em um único arquivo também podemos utilizá-lo em pastas inteiras. Crie uma pasta “src” e divida o conteúdo do arquivo index.js em diferentes arquivos dentro dela.

   Feito isso, execute o comando com a flag --out-dir (ao invés de --out-file):

```
npx babel src --out-dir dist --presets=@babel/preset-env
```

8. Por fim, podemos até mesmo escrever um script para executar o Babel de forma mais ágil:

```
// package.json // ... "main": "index.js", "scripts": { "babel": "babel src --out-
dir dist --presets=@babel/preset-env" }, "keywords": [], // ...
```

9. Agora só precisamos executar o Babel com o comando:

```
npm run babel
```

10. Uma última observação é que nós também podemos configurar o Babel através de arquivos, assim não precisaremos utilizar as flags no comando. Ele possui dois tipos de arquivos de configuração: *.babelrc* e *babel.config.js*. O babel.config.js é o arquivo de configuração principal, com ele podemos definir as opções que serão usadas como padrão pelo Babel. Já o .babelrc permite escrever configurações específicas para diretórios ou partes específicas dos nossos projetos.

    Para vermos esse comportamento rapidamente vamos criar o arquivo "babel.config.js" na pasta do projeto e nele vamos especificar os presets passados através de flag:

    ```
    // babel.config.js module.exports = { presets: [ [ "@babel/preset-env" ] ] }
    ```

11. Agora podemos remover a flag e executar novamente o comando, e mesmo assim a saída será igual a que obtivemos antes:

    ```
    // package.json // ... "main": "index.js", "scripts": { "babel": "babel src --out-
    dir dist --presets=@babel/preset-env" }, "keywords": [], // ...
    ```

    ```
    npm run babel
    ```