Imagina que eu queria criar um módulo que faz só uma coisa, por exemplo, eu tenho um módulo que faz uma função específica e isso acontece muito. Eu posso resetar o module fazendo assim:

```
module.exports = function(x, y) {
    return x * y;
}
```

Esse módulo ele faz exportar diretamente uma função aqui então se eu importar ele no app.js eu posso fazer:

```
const multiplicacacao = require('./modulo1.js');
```

Essa minha variável é multiplicação contém a minha função assim então já posso chamada como minha função:

```
console.log(multiplicacao(2, 2));
```

```
4
```

Ou seja a minha função faz algo muito similar a aquele **default** que eu faço no ES6 Eu estou importando só uma coisa, eu só quero essa coisa e o programa está pegando. Porém, nem dentro do module eu consigo executar essa função porque ela é anônima se eu quisesse utilizar eu teria que usar:

```
console.log(module.exports(2, 2));
```

Eu zerei a variável então essa função não foi criada para ser usada no arquivo modulo1.js, mas se eu quiser por algum motivo eu posso. Isso vai funcionar com Arrow Functions, variáveis, funções etc.

```
const number = require('./modulo1.js');

console.log(number);
```

```
module.exports = 19;
```

Estou exportando o número 19. Se eu quisesse que isso fosse uma classe:

```
module.exports = class Cachorro {
    constructor(name) {
        this.name = name;
    }
```

```
    latir() {
        console.log(`${this.name} está fazendo au au.`);
    }
}
```

```
const cachorro = require('./modulo1.js');

const dog = new cachorro('Golden Retriever');
dog.latir();
```

```
Golden Retriever está fazendo au au.
```

Então eu estou fazendo aqui um módulo exportar uma classe. Eu exportei uma classe direto no module.exports e no app.js eu já puxo direto a classe para a variável cachorro e daí eu trato ela como uma variável. Eu poderia utilizar a classe assim como eu usei nas aulas anteriores a diferença que ela está vindo de um módulo.

No arquivo modulo1 e modulo2.js eu faço a exportação e no app.js eu faço a importação.

```
const cachorro = require('./example4/modulo2.js');

const c1 = new cachorro('Golden Retriever');
c1.latir();
```

```
const cachorro = require('../exemple1/exemple2/exemple3/modulo1.js');

module.exports = cachorro; // exportando a classe
```

```
// ./example1/arquivo.js
// ./arquivo.js
// ../nomepasta/nomepasta/nomearquivo...
```

A gente tem duas variáveis internas dentro do módulo que eu posso utilizar são elas as variáveis:

```
console.log(__filename);
console.log(__dirname);
```

Elas são muito convenientes e eu vou saber quando eu vou utilizar qualquer coisa relacionado com o caminho, quando eu precisar buscar um caminho, um arquivo alguma coisa assim. **dirname** é o nome da pasta atual e **filename** é o nome do

arquivo atual,  mas esse é o caminho absoluto, ou seja, ele vai pegar lá da raiz do meu computador dependendo do sistema operacional no caso do Windows: C/D… até chegar no arquivo. No meu caso é:

```
d:\JavaScript\node\class2\app.js
d:\JavaScript\node\class2
```

E daí para manipular caminhos o Luiz Otávio o professor gosta muito de uma função de um módulo que está disponível por padrão no node o **path**. Vamos testá-lo novamente:

```
const path = require('path');
console.log(__dirname);
```

Imagina que eu queira navegar mas eu não sei em qual sistema o meu sistema, o meu programa vai rodar em um computador Windows, servidor Windows (raro, mas acontece), então eu não sei na onde ele vai rodar (por exemplo) e daí eu gostaria que ele resolvesse como que ele vai colocar a / para frente ou para trás, ele vai se virar e vai criar um caminho correto pra mim:

```
const path = require('path');
console.log(path.resolve(__dirname));
```

No resolve eu mando o caminho da pasta aonde eu estou **__dirname**. Ele vai pegar exatamente isso:

```
d:\JavaScript\node\class2
```

Agora imagine que nessa função eu posso colocar uma vírgula e falar: “Desse local aqui que eu estou eu quero voltar duas pastas então ele vai excluir a pasta da class2 e do node:

```
const path = require('path');
console.log(path.resolve(__dirname, '..', '..'));
```

```
d:\JavaScript
```

E imagina que daqui do JavaScript eu queira ir numa pasta que chama, por exemplo, pasta tempo e dentro dela tem a pasta imagens eu faço desta maneira:

```
const path = require('path');
console.log(__dirname);
console.log(path.resolve(__dirname, '..', '..', 'tempo', 'imagens'));
```

```
d:\JavaScript\tempo\imagens
```

Eu falei pra ele voltar duas pastas e entrar na pasta tempo e imagens então agora eu estou na pasta imagens e ele pode voltar. Se eu colocar 1 ponto ao invés de 2 acontece o seguinte:

```
d:\JavaScript\node\tempo\imagens
```

O caminho é diferente porque eu continuei na mesma pasta porque ele resolveu isso somente incluindo na página, ou seja, ele ignorou os pontos porque esses 2 pontos representam a pasta atual na qual eu estou.

Em módulos a gente sabe que a gente pode colocar ou não a extensão *js*, mas vamos confirmar isso:

```
const cachorro = require('./example4/modulo2.js');

const c1 = new cachorro('Border Collie');
c1.latir();
```

```
Border Collie está fazendo au au.
```

Não mudou nada!

Geralmente nós não vamos colocar a extensão *js* quando o arquivo for um arquivo js porque pode ter arquivos CSS que a gente importa, além de imagens então quando não for um arquivo js eu coloco a extensão pro meu programa saber o que ele tem que exportar porque neste caso aqui ele só sabe que é arquivo JS, porque ele sabe que é JavaScript.