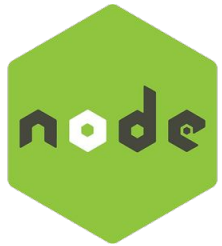
NodeJS



Prof. Celso Henrique Masotti

Necessidades Prévias

- Ter o node.JS instalado;
- Ter o gerenciador de pacotes (NPM) instalado;
- Ter um editor de códigos IDE(*), dê preferência ao “Visual Studio Code”
- Conhecer JavaScript básico.

(*) IDE, do inglês Integrated Development Environment ou Ambiente de Desenvolvimento Integrado, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo, como um editor de texto para a digitação de códigos.

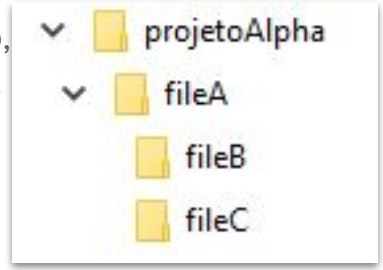
Você verá nesta aula:

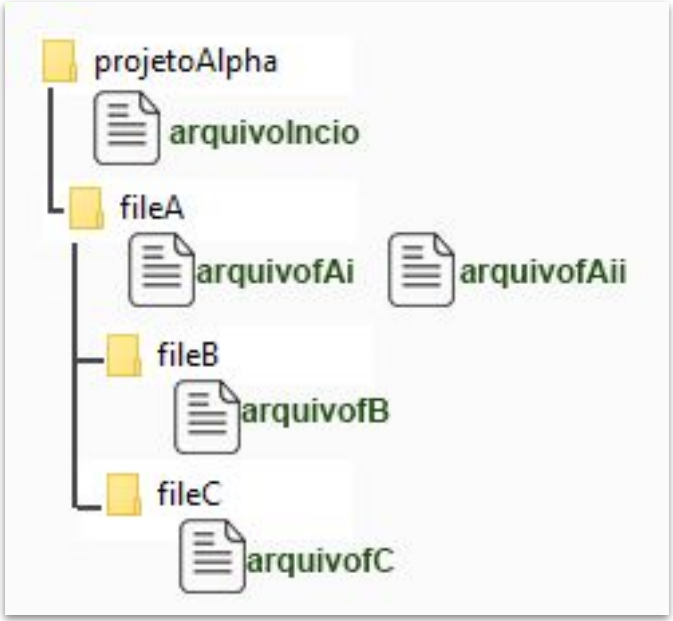
Objetivo: Saber usar endereçamentos de pastas, criar e utilizar arquivos em formato de módulos.

Endereçamento de Diretórios
Módulos
Praticando

Endereçamento de Diretórios

Muitas vezes precisamos incluir um arquivo em nosso programa e temos, portanto, de colocar seu endereço. Vejamos como fazer isso. Imagine quatro pastas, sendo primeira a do projeto.





Temos arquivos no interior destes diretórios/pastas. Vejam.

Endereçamento de Diretórios

- Mesmo Diretório: para informar a inclusão do arquivo “arquivofAii” no “arquivofAi” usamos o ponto barra (./), ficando “**./arquivofAii**”;
- Diretório Abaixo: para informar a inclusão do arquivo “arquivofB” no “arquivofAi” usamos o ponto barra (./), nome da pasta(fileB) e barra (/), ficando “**./fileB/arquivofB**”;
- Diretório Acima: para informar a inclusão do arquivo “arquivoInicio” no “arquivofAi” usamos o ponto ponto barra (../), ficando “**../arquivoInicio**”;

./	./pasta/	../
Sinal para partir da pasta atual	Partindo da pasta atual acessar o diretório “pasta”	Subir um diretório se aproximando do “root”

Módulos

O padrão de módulo é usado para imitar o conceito de classes (JavaScript não suporta nativamente o conceito de classes) assim, podemos armazenar dentro de um único objeto variáveis, métodos públicos e privados semelhante o conceito de classes que são usadas em outras linguagens de programação como Java ou Python. Isso nos permite criar uma API pública revestida com os métodos que queremos expor ao mundo, embora tudo isso seja encapsular variáveis particulares e métodos em um escopo de *closure* (algo como “fechamento”).

Módulos

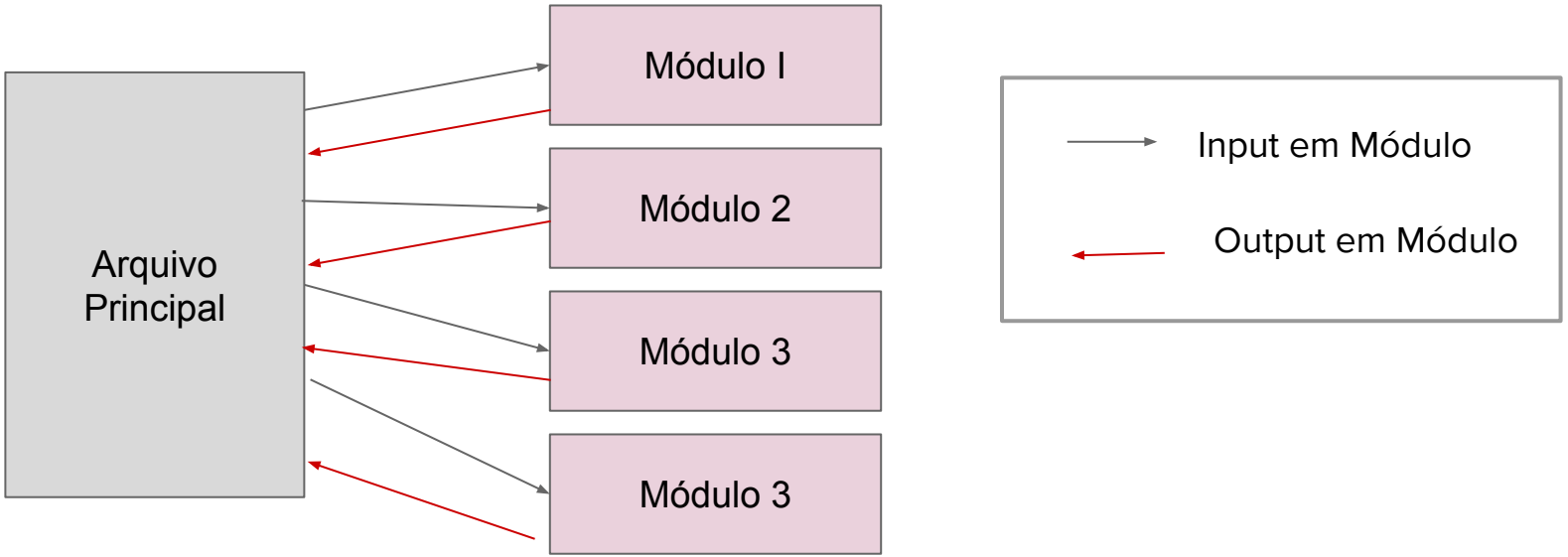
Existem uma série de benefícios ao usar módulos em favor de uma extensa base de códigos interdependentes. Os dois mais importantes, são:

1) **Manutenibilidade**: Por definição, um módulo é auto-suficiente, quando bem projetado visa diminuir as dependências em partes da base de código, tanto quanto possível, de modo que ele possa crescer e melhorar de forma independente. Atualizando um único módulo é muito mas fácil quando o mesmo é dissociado de outros pedaços de códigos.

2) **Reusabilidade**: módulos podem ser reutilizados sempre que desejarmos.

Módulos

Módulos são partes separadas de algoritmos que entram em atividade somente quando solicitamos.



Módulos

Em javascript um módulo nasce como função e é inserida em uma variável e que deve, obrigatoriamente ficar recluso a um arquivo exclusivo. Para que a função possa ser enviada à quem solicitar o arquivo deve possuir o comando “module.exports”.

```
function somar(a,b){
        return a + b;
}
```

Função somar realiza a soma de dois valores e devolve o resultado. Esta pode estar em um arquivo com outras ações em códigos.

```
var somar = function (a,b){
        return a + b;
}
```

Uma variável conterá uma função sem nome e terá de ficar em um arquivo específico.

```
var somar = function (a,b){
        return a + b;
}
module.exports = somar;
```

A última linha de comando do arquivo recebe o “module” com a propriedade “exports” que torna possível a exportação da função para quem solicitar

Módulos

Para que possamos fazer uso de um módulo devemos armazenar a função que desejamos chamando o módulo correspondente através do comando “include” seguido do endereçamento de pasta e nome de arquivo onde o módulo se encontra.

```
var somar = function (a,b){

        return a + b;

}

module.exports = somar;
```

Este módulo se encontra no arquivo “somar.js”

Arquivos de javascript não precisam exibir a extensão.

```
var somaFunc = require("./somar");
console.log(somaFunc(6,9));
```

Requisitando o arquivo “somar.js” no arquivo que será executado pelo node.js. A função foi inserida na variável “somaFunc” e posteriormente ativada pelo “console.log()”

```
>node arquivoInicial.js
>15
```

Executando o arquivo no node.JS

Praticando

1 - Em uma pasta crie três arquivos denominados “inicial.js”, “area.js” (que calcula a área de um terreno) e “perimetro.js” (que calcula o perímetro de um terreno). Transforme as duas últimas em módulos. Inclua estes arquivos no “inicial.js”.

2 - Crie uma pasta chamada “Calculadora” e em seu interior outra pasta denominada “operacoes”. Na pasta “Calculadora” crie um arquivo javascript que realizará todas as operações que se encontrarão nos arquivos modulares da pasta “operacoes” (soma.js, divisao.js, multiplicacao.js, subtracao.js e exponenciacao.js).

Obrigado

Profº. Celso Henrique Masotti