

CAPÍTULO 2 – Módulos e NPM

Neste capítulo você vai ver:

- Os recursos dos principais módulos nativos (core modules) do Node.js
- O gerenciador de pacotes NPM
- Como criar, exportar e importar módulos (criar códigos reutilizáveis)
- Construir um servidor HTTP apenas utilizando core modules (sem bibliotecas)

Atividade 1 – Tornando o script síncrono

Essa atividade tem como objetivo:

- Ler entradas a partir do console de forma assíncrona;
- Diminuir a complexidade utilizando módulos e bibliotecas;
- Instalar globalmente um modulo do Node.js por meio do NPM;

Comandos:

- var ou let - Declaração de variáveis
- npm install prompt-sync -g – Instalação de módulos
- require prompt-sync – Importação de módulos

Instalar o primeiro modulo global e executar o script de forma síncrona

1. Dentro do diretório “NODEJS_EXERCÍCIOS”, crie um novo diretório chamado “Cap02” e acesse o diretório via terminal com o comando “cd cap02”.
2. Vamos agora **instalar de forma global o modulo** externo **prompt-sync** para receber entradas de console de forma síncrona. Os módulos dos Node.js podem ser instalados por meio do seu gerenciador de pacotes o NPM. Para instalar um pacote utilizando o NPM utiliza-se o comando com a sintaxe “npm install <nome_do_pacote> -parametro”. Neste exemplo, utilize o comando “**npm install prompt-sync -g**” o parâmetro -g significa que a instalação é em modo global.
3. Como trata-se de uma instalação global, o modulo prompt-sync precisa ser “lincado” ao nosso script. O comando tem a sintaxe “npm link <nome_do_pacote>”, neste exemplo utilize o comando “**npm link prompt-sync**”;
4. Após a conclusão, no diretório “Cap02” - crie um arquivo com o nome “at01-OláMundo3.js”;
5. Insira o seguinte código no arquivo:

```
//prompt-sync - modulo que recebe entradas de console de modo síncrono
let prompt = require('prompt-sync')
prompt = prompt({sigint:true})
console.log('Olá Mundo!\r\n')
// var e let são argumentos para declarar variáveis
let nome= prompt('Qual é seu nome? ')
console.log(`Olá ${nome}!\r\n`)
console.log('Olá node!')
```

6. Rode o script e note que retornou um erro indicando que o módulo 'prompt-sync' não foi encontrado. Isto é devido ao módulo ter sido instalado no escopo global, por isso é necessário ser linkado ao projeto. Utilize o comando **"npm link prompt-sync"**.
7. Rode o script novamente e ele será executado normalmente. Repare também e veja que desta vez foi executado de maneira síncrona, ou veja, esperou a entrada do dado.

```
JS at05-OlaMundo3.js X
JS at05-OlaMundo3.js > (e) nome
1 //prompt-sync - modulo que recebe entradas de console de modo assi
2 let prompt = require('prompt-sync')
3 prompt = prompt({sigint:true})
4 console.log('Olá Mundo!\n')
5 // var e let são argumentos para declarar variáveis
6 let nome= prompt('Qual é seu nome? ')
7 console.log(`Olá ${nome}!\n`)
8 console.log('Olá node!')
9
10

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

PS C:\Users\NodeJs\Desktop\NodeJs_Exercícios\Cap01> node .\at05-OlaMundo3.js
Olá Mundo!
Qual é seu nome? Joaquim
Olá Joaquim!
Olá node!
PS C:\Users\NodeJs\Desktop\NodeJs_Exercícios\Cap01>
```

Obs.: O módulo prompt-sync não suporta o charset UTF-8, devido este exemplo ser apenas para fins didáticos iremos lidar com esta questão apenas em atividades

Atividade 2 – Conhecer o módulo “os” e variáveis de ambiente

Essa atividade tem como objetivo:

- Conhecer os recursos do core module “os” que tem acesso direto a informações e recursos do sistema operacional.
- Conhecer as variáveis globais de ambiente do Node.js

Comandos:

- os.cpus(); os.freemem(); os.homedir(); os.type()
- process.env

Importar o core module “os” e exibir variáveis globais de ambiente do Node.js

Primeiramente vamos importar e utilizar alguns recursos do modulo “os”. Ele é um modulo interno do Node.js e por isso não precisa ser instalado pelo NPM, basta apenas importar com o require.

Depois vamos escrever no script uma que exibe as variáveis de ambiente do Node.js no console e conversar sobre as suas utilidades.

Siga os passos seguintes:

1. No diretório “Cap02” **crie um arquivo** com o nome “**at02-OsProcessEnv.js**”.
2. Implemente o seguinte código no arquivo:

```
//Importação do modulo "os"
const os = require('os')
console.log('Processador(es): ', os.cpus())
console.log('Qtde memória livre: ', os.freemem())
console.log('Diretório do usuário: ', os.homedir())
console.log('Familia de S.O.: ', os.type())
```

3. Execute o script com o comando “**node at02-osProcessEnv.js**” e observe as saídas.
4. Agora o primeiro teste, adicione a seguinte linha no fim do script:

```
//Lista todas as variáveis de ambiente
console.log(process.env)
```

5. Observe as saídas e veja que retornou muitas informações do Sistema Operacional que podem ser uteis no desenvolvimento e na “automação” de várias necessidades que dependa do sistema operacional;

Atividade 3 – Conhecer o modulo “path”

Essa atividade tem como objetivo:

- Conhecer os recursos do core module “path” que tem como principal função fornecer informações de caminho, nome e extensão de arquivos;

Comandos:

- path.extname();
- path.basename();
- path.dirname();
- path.resolve();

Importar o modulo “path” e exibir informações de caminho e nome de arquivos

Vamos importar o modulo “path” e utilizar alguns de seus recursos. O “path” também é um modulo interno do Node.js, necessitando apenas a importação com o require. Siga os passos seguintes:

1. No diretório “Cap02” **crie um arquivo** com o nome “at03-Path.js”.
2. Implemente o seguinte código:

```
const path =require('path')
let arquivo='./at03-Path.js'

console.log('Extensão: ', path.extname(arquivo))
console.log('nome completo: ', path.basename(arquivo))
console.log('Unidade Base: ', path.dirname(arquivo))
console.log('Caminho Absoluto: ', path.resolve(arquivo))
```

3. Execute o script e observe as saídas do console.

Atividade 4 – Conhecer o modulo “url”

Essa atividade tem como objetivo:

- Apresentar alguns recursos do core module “url” que tem como principal função extrair informações das strings no formato de url. Será posteriormente utilizada em conjunto com o serviço HTTP para definir rotas;

Comandos:

- partUrl.host; partUrl.pathname;
- partUrl.search;
- partUrl.searchParams;
- partUrl.searchParams.get;

Importar o modulo “url” e exibir informações de uma string no formato de url

Vamos importar o modulo “url” e utilizar alguns de seus recursos. O “url” também é um modulo interno do Node.js, necessitando apenas a importação com o require. Siga os passos seguintes:

1. No diretório “Cap02” **crie um arquivo** com o nome “at04-Url.js”.
2. Implemente o seguinte código:

```
const url = require('url')
let uri = 'https://www.google.com/search?q=node+js&rlz=1C1BNSD_pt-
BRBR946BR946'
let partUrl = new url.URL(uri)
console.log('Domínio: ', partUrl.host)
console.log('Caminho ou Rota: ', partUrl.pathname)
console.log('Query String: ', partUrl.search)
console.log('Parâmetros: ', partUrl.searchParams)
console.log('Valor do parâmetro q: ', partUrl.searchParams.get('q'))
console.log('Valor do parâmetro rlz: ', partUrl.searchParams.get('rlz'))
```

3. Execute o script e observe as saídas do console.

Atividade 5 – NPM: Iniciar projetos, instalar e remover módulos.

Essa atividade tem como objetivo:

- Apresentar as funcionalidades do NPM;
- Iniciar um projeto;
- Conhecer o arquivo package.json;
- Entender como o node organiza seus arquivos;
- Instalar, atualizar e remover módulos;
- Utilizar um modulo instalado pelo NPM;

Comandos:

- npm init || npm init -y(forma rápida);
- npm install <pacote>;
- npm uninstall <pacote>

Instalar, importar e remover módulos externos com o NPM

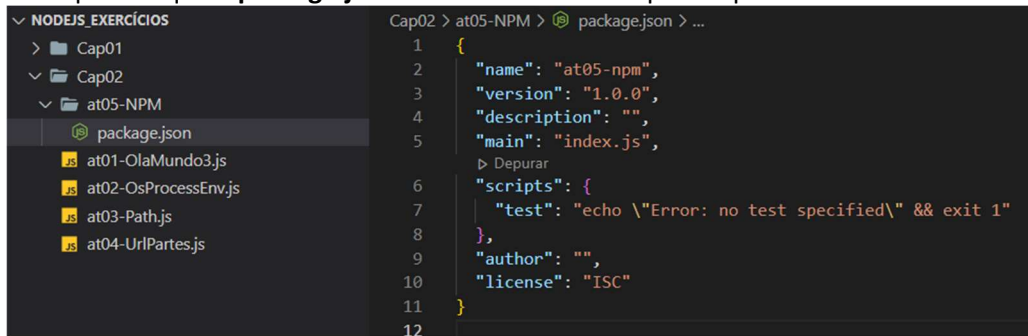
Já foi observado que o NPM é o gerenciador de pacotes do Node.js, então agora vamos ver como ele adicionar e remover módulos externos. Por meio do arquivo package.json (contém todas as informações sobre o projeto), o npm também pode gerenciar as dependências do projeto.

Vamos experimentar algumas funcionalidades do NPM.

Siga os seguintes passos:

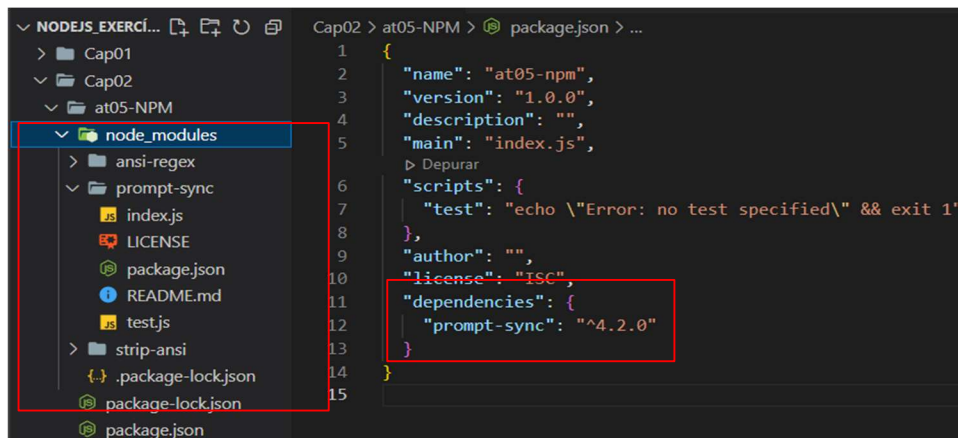
1. Dentro de Cap02 crie um subdiretório “**at05-NPM**” e acesse-o via terminal com o comando “**cd at05-NPM**”.

2. No diretório “at05-NPM” emita o comando “**npm init**” ou “**npm init -y**”(forma rápida) para iniciar um projeto Node.js;
3. Perceba que o arquivo **package.json** foi criado. Abra o arquivo e perceba seus atributos:



```
1 {
2   "name": "at05-npm",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC"
11 }
12
```

4. Agora, um modulo externo utilizando o NPM. A sintaxe do comando para a instalação de módulos é “**npm install <nome_do_modulo>**”. Para o nosso exemplo, vamos instalar o modulo **prompt-sync**, desta vez de forma local, para isso emita o comando “**npm install prompt-sync**”.
5. Note que um subdiretório chamado **node_modules** foi criado. Duas coisas são importantes serem observadas: 1) dentro de **node_modules** temos uma pasta contendo os módulos com o código do **prompt.sync**; e 2) um atributo “**dependencies**” foi adicionado no **package.json** e contém a dependência do modulo instalado bem como a indicação de sua versão.



```
1 {
2   "name": "at05-npm",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "prompt-sync": "^4.2.0"
13  }
14 }
15
```

6. Agora vamos realizar a instalação de diversos pacotes simultaneamente, utilize o comando “**npm i express mysql2 consign cors sequelize**”(o argumento **install** pode ser utilizado da forma abreviada “**i**”). Após a conclusão da instalação, perceba que a lista de dependências no **package.json** cresceu:



```
1 {
2   "name": "at05-npm",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "consign": "^0.1.6",
13    "cors": "^2.8.5",
14    "express": "^4.18.1",
15    "mysql2": "^2.3.3",
16    "prompt-sync": "^4.2.0",
17    "sequelize": "^6.23.0"
18  }
19 }
```

- Com o objetivo de exercitar as funcionalidades do NPM, agora vamos remover os módulos. A sintaxe do comando para remover um pacote é “npm uninstall <nome_do_pacote>”. Como exemplo vamos utilizar o comando “**npm uninstall prompt-sync**”.
- Os arquivos e a dependência do prompt-sync foram removidos. Para exercitar, remova os demais módulos instalados com o comando “**npm uninstall express mysql2 consign cors sequelize**”.
- Verifique que todos as pastas do node_modules e as dependências do package.json foram removidos:



- Agora vamos instalar e utilizar um modulo. Para o exemplo vamos utilizar o modulo lodash. Utilize o comando “**npm i lodash**”.
- Após a instalação podemos importar o modulo normalmente com o comando require. Crie um arquivo com o nome at05-lodash.js e insira código a seguir:

```
const _ = require('lodash'); //Importação da biblioteca lodash
//Função que gera números aleatórios
const randomNum = ()=> Math.trunc(Math.random()*100);

console.log('----Usando o lodash---')
let numRandoms = _.times(10, randomNum)//Executa uma função X vezes
//Exibição do array - jeito convencional
console.log(typeof(numRandoms)) //Typeof exibe o tipo da variável
console.log(numRandoms)
console.log('Soma dos elementos: ', numRandoms.reduce((s, acc)=> acc+=s))
console.log(`Primeiro Elemento: ${numRandoms[0]}`)
console.log(`Último Elemento: ${numRandoms[numRandoms.length-1]}\r\n`)
// Exibição do array - utilizando o lodash
console.log('----Usando o lodash---')
console.log('Soma dos elementos: ', _.sumBy(numRandoms))
console.log(`Primeiro c/ lodash: ${_.first(numRandoms)}`)
console.log(`Último c/ lodash: ${_.last(numRandoms)}\r\n`)
```

- Execute o código e observe as saídas no console.

Atividade 6 – Implementar e utilizar módulos internos próprios.

Essa atividade tem como objetivo:

- Implementar, Importar, exportar e reutilizar funções em módulos próprios;
- Transformar functions em arrow functions.

Comandos:

- exports
- module.exports

Vamos aprender a criar nossos próprios módulos, exportá-los para serem reutilizados por outros códigos. Vamos também aproveitar e revisar a sintaxe de functions(funções em javaScript) e arrow functions (funções de seta).

Siga os passos seguintes:

1. Dentro do diretório Cap02 crie um arquivo chamado “cd at06-ModulosProprios” e o acesse via terminal com o comando “**cd at06-ModulosProprios**”.
2. Crie um arquivo com o nome “at06-Calc.js” e insira o seguinte código:

```
function soma (a, b){
    return parseInt(a)+parseInt(b)
}
function sub (a, b){
    return parseInt(a)-parseInt(b)
}
function mult (a, b){
    return parseInt(a)*parseInt(b)
}
function div (a, b){
    if (parseInt(b)>0){
        return parseInt(a)/parseInt(b)
    }
    return "Não é possível dividir por zero."
}
console.log(soma(10,10))
console.log(sub(50,10))
console.log(mult(9,8))
console.log(div(100,4))
console.log(div(10,0))
```

3. Execute o código e observe a saída.
4. Note as funções foram escritas na sua forma mais verbosa. Vamos reescrever as utilizando um modulo externo e simplificá-las utilizando arrow function. Para isso, crie um diretório chamado “at06-CalcMod1” e o acesse via terminal com o comando “**cd at06-CalcMod1**”.
5. Crie um arquivo no chamado “**calculadora.js**” e insira o seguinte código:


```
soma=(a, b)=>{return parseInt(a)+parseInt(b)}
exports.soma = soma

sub=(a, b)=>{return parseInt(a)-parseInt(b)}
exports.sub = sub

mult=(a, b)=>{return parseInt(a)*parseInt(b)}
exports.mult = mult

div=(a, b)=>{return parseInt(b)>0? parseInt(a)/parseInt(b) : "Não é
possível dividir por zero."}
exports.div = div
```

6. Perceba que na escrita das funções foi utilizada a sintaxe de arrow function e a escrita das funções ficaram menos verbosas. Note também que foi utilizado argumento “**exports**” que tem como funcionalidade tornar uma função ou variável disponível para serem reutilizadas em outros módulos.
7. Para utilizarmos as funções do modulo “calculadora.js” crie um novo arquivo no diretório “at06-CalcMod1” com o nome “at06-Calc.js” e insira o seguinte código:

```
const calculadora = require('./calculadora')
console.log(calculadora.soma(10,10))
console.log(calculadora.sub(50,10))
console.log(calculadora.mult(9,8))
console.log(calculadora.div(100,4))
console.log(calculadora.div(10,0))
```

8. Execute o código e analise as saídas. Perceba que a funcionalidade do programa é a mesma do código anterior.
9. Agora vamos ver uma forma mais simplificada de exportar as funções de um modulo e simplificar ainda mais a escrita das funções utilizando a sintaxe de arrow function. Para isso **duplique** o diretório “at06-CalcMod1” e **renomeie a cópia** para “at06-CalcMod2”.
10. Substitua o código do modulo “calculadora.js” pelo seguinte código:

```
const calculadora = {
  soma:(a, b)=> parseInt(a)+parseInt(b),
  sub: (a, b)=>parseInt(a)-parseInt(b),
  mult:(a, b)=>parseInt(a)*parseInt(b),
  div:(a, b)=> parseInt(b)>0? parseInt(a)/parseInt(b) : "Não é
    possível dividir por zero."
}
module.exports=calculadora
```

11. Note que as funções foram simplificadas ainda mais com a sintaxe de arrow function. Mas o mais importante é perceber que agora as funções foram encapsuladas em um objeto chamado calculadora e foram exportadas todas de uma vez com o argumento `"module.exports"`.
12. Execute o script do arquivo `"at06-Calc.js"`, observe a saída e perceba que a funcionalidade dos três códigos são as mesmas.

Exercícios extras

Estas atividades extras têm como objetivo revisar e aprimorar fundamentos da linguagem JavaScript que serão utilizados nas atividades deste curso.

1. **Pesquise e implemente** exemplos de utilização de **arrays** no JavaScript;
2. **Pesquise e implemente** exemplos das seguintes funções de **manipulação de arrays**:
`Array.push()`; `Array.pop()`; `Array.unshift()`; `Array.shift()`