



++

45697096

# Introdução ao JavaScript

## Hybrid Mobile App Development

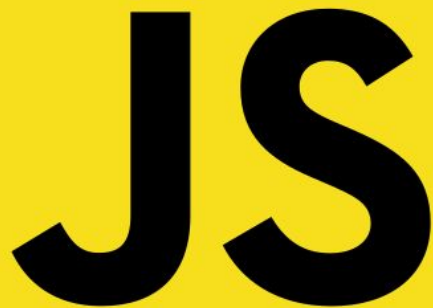
45697096

45697096

...



Prof. Alexandre dos Santos Mignon <[profalexandre.mignon@fiap.com.br](mailto:profalexandre.mignon@fiap.com.br)>



# **Introdução ao JavaScript**

- Linguagem Interpretada;
- Tipagem dinâmica fraca;
- Multi-paradigma;
- Inicialmente projetada para Web;
- Pode ser usada como Client-side & Server-side;
- Uma das linguagens mais utilizadas.

- Linguagem criada em 1995 por Brendan Eich (Netscape);
- Possui o nome JavaScript como forma de aproveitar a fama da linguagem Java na época de seu lançamento;
- No Internet Explorer 3, iniciou-se uma implementação modificada chamada JScript;
- Enviada para a European Computer Manufacturers Association para padronização da linguagem em 1997;
- Syntax semelhante com outras linguagens baseadas em C;
- ECMAScript é o nome oficial.

- Usada no desenvolvimento WEB (Client-side);
- Usada no desenvolvimento WEB (Server-side / Node.JS);
- Usada no desenvolvimento Desktop (Electron);
- Usada no desenvolvimento Mobile (Cordova, PhoneGap, Ionic, React-Native).

- Também chamado de ES6, ES2015 ou ECMAScript 2015, trás consigo algumas novidades:
  - Constants
  - Block Scope
  - Arrow Functions
  - Template Literals
  - De-structuring
  - Modules
  - Classes
  - Iterators
  - Collections
  - Promises

# Configuração do Ambiente

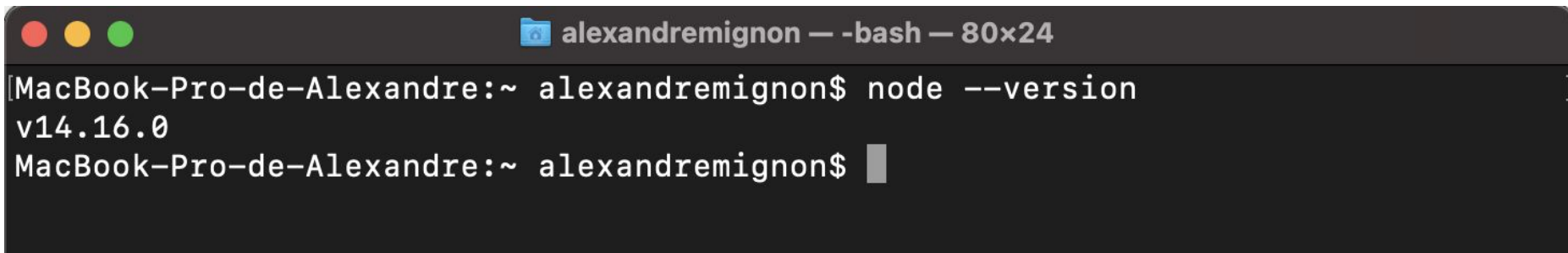
- Vamos inicialmente utilizar o NodeJS para aprendizado.
- Deve-se instalar a última versão estável.

<https://nodejs.org/en/>



- Após o NodeJS ser instalado, vamos usar o terminal (cmd, prompt, etc) para confirmar a versão instalada com o seguinte comando.

**node --version**



```
alexandremignon — -bash — 80x24  
[MacBook-Pro-de-Alexandre:~ alexandremignon$ node --version  
v14.16.0  
MacBook-Pro-de-Alexandre:~ alexandremignon$ ]
```

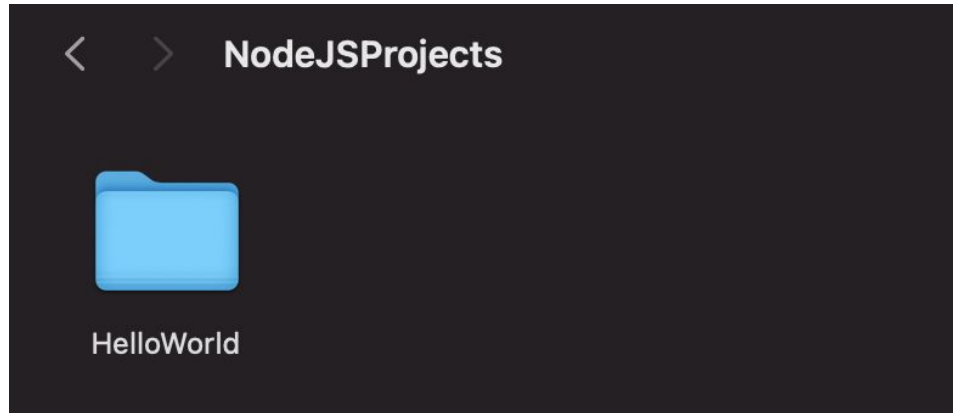
- Escolher um IDE para ser utilizado durante as aulas. Sugestões:

<https://code.visualstudio.com/> (Recomendado)

<http://brackets.io/>

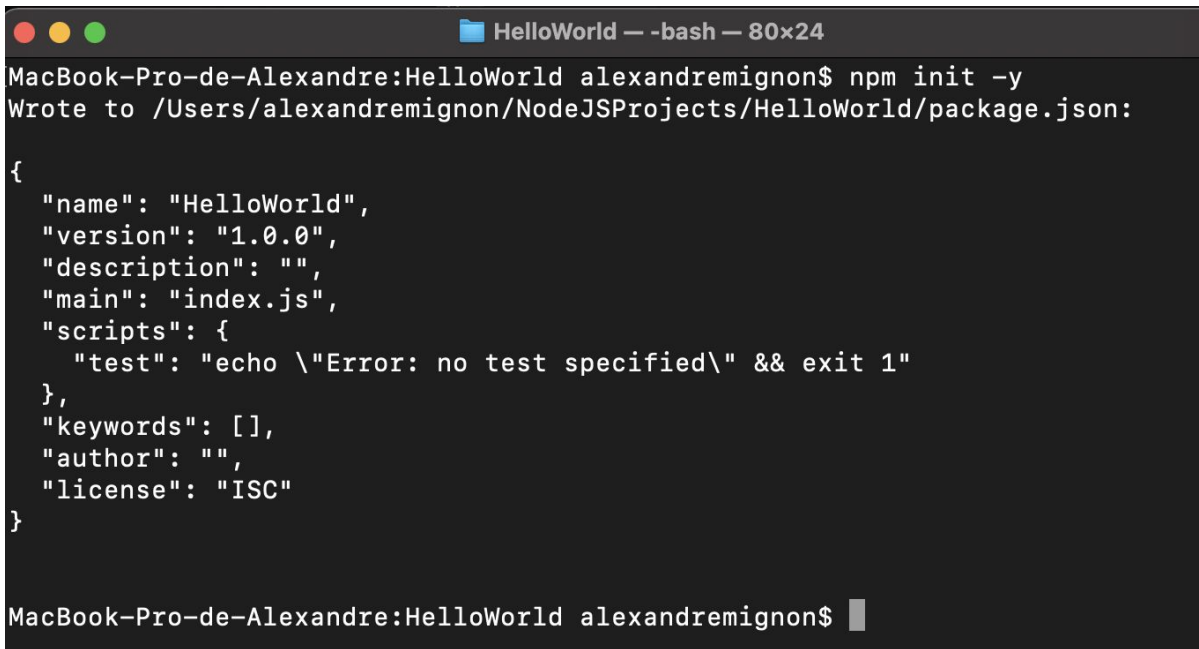
<https://atom.io/>

- Após tudo instalado, vamos criar nosso primeiro projeto em NodeJS.
- Quando instalado o NodeJS, ele instala uma ferramenta chamada NPM.
- O NPM funciona como um gerenciador de pacotes para NodeJS, ou seja, através dele é possível adicionar bibliotecas de terceiros em seu projeto.
- Crie uma pasta em seu computador chamada HelloWorld.



- Abra o terminal, vá até a pasta **HelloWorld** criada e digite o seguinte comando:

**npm init -y**



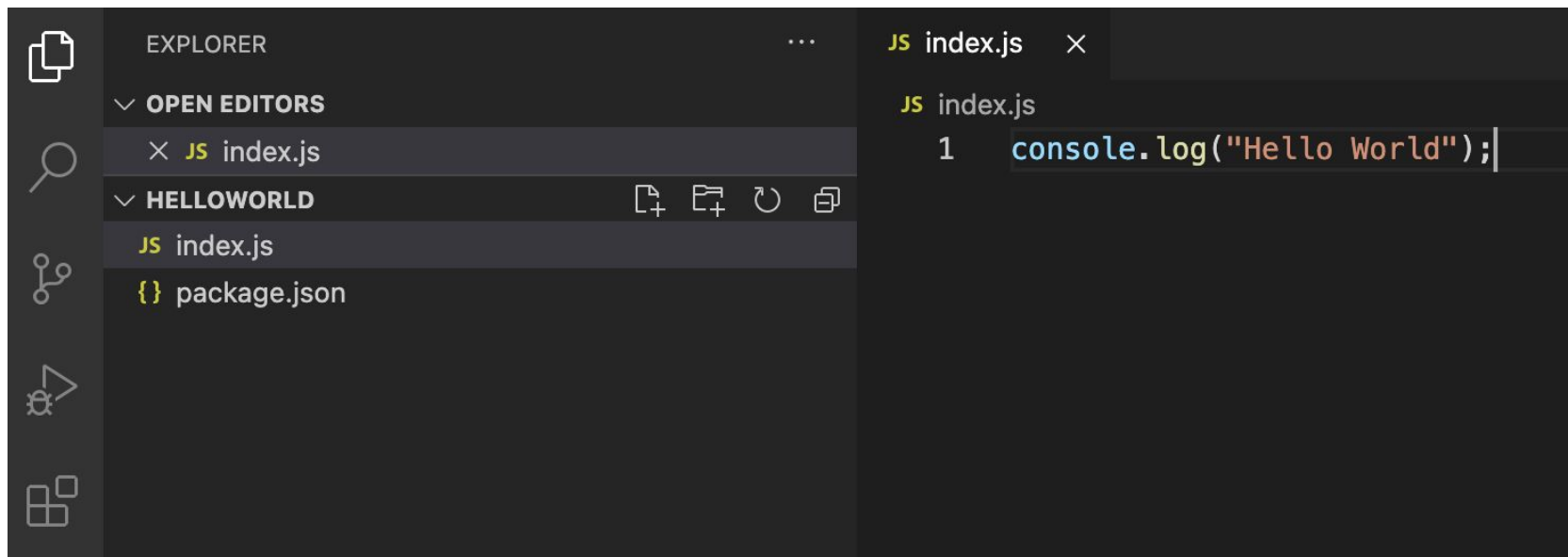
```
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$ npm init -y
Wrote to /Users/alexandremignon/NodeJSProjects/HelloWorld/package.json:

{
  "name": "HelloWorld",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$
```

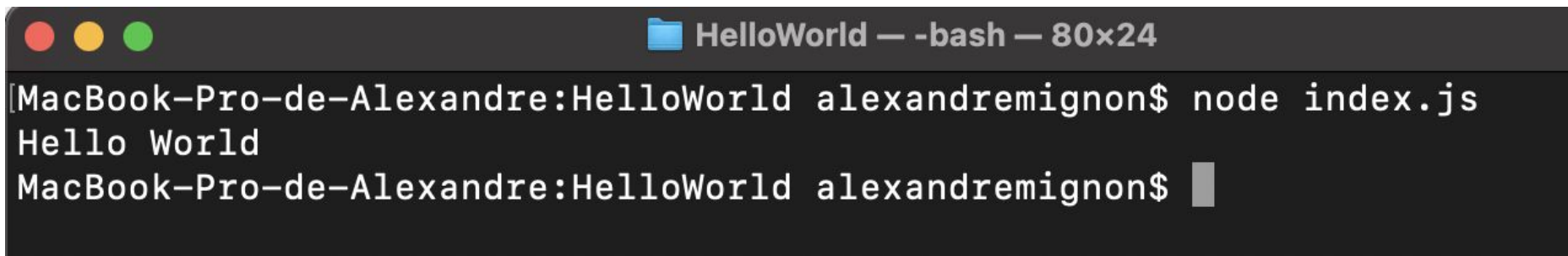
- Abra a pasta do projeto em seu editor preferido e crie um arquivo **index.js** com o seguinte comando:

```
console.log("Hello World");
```



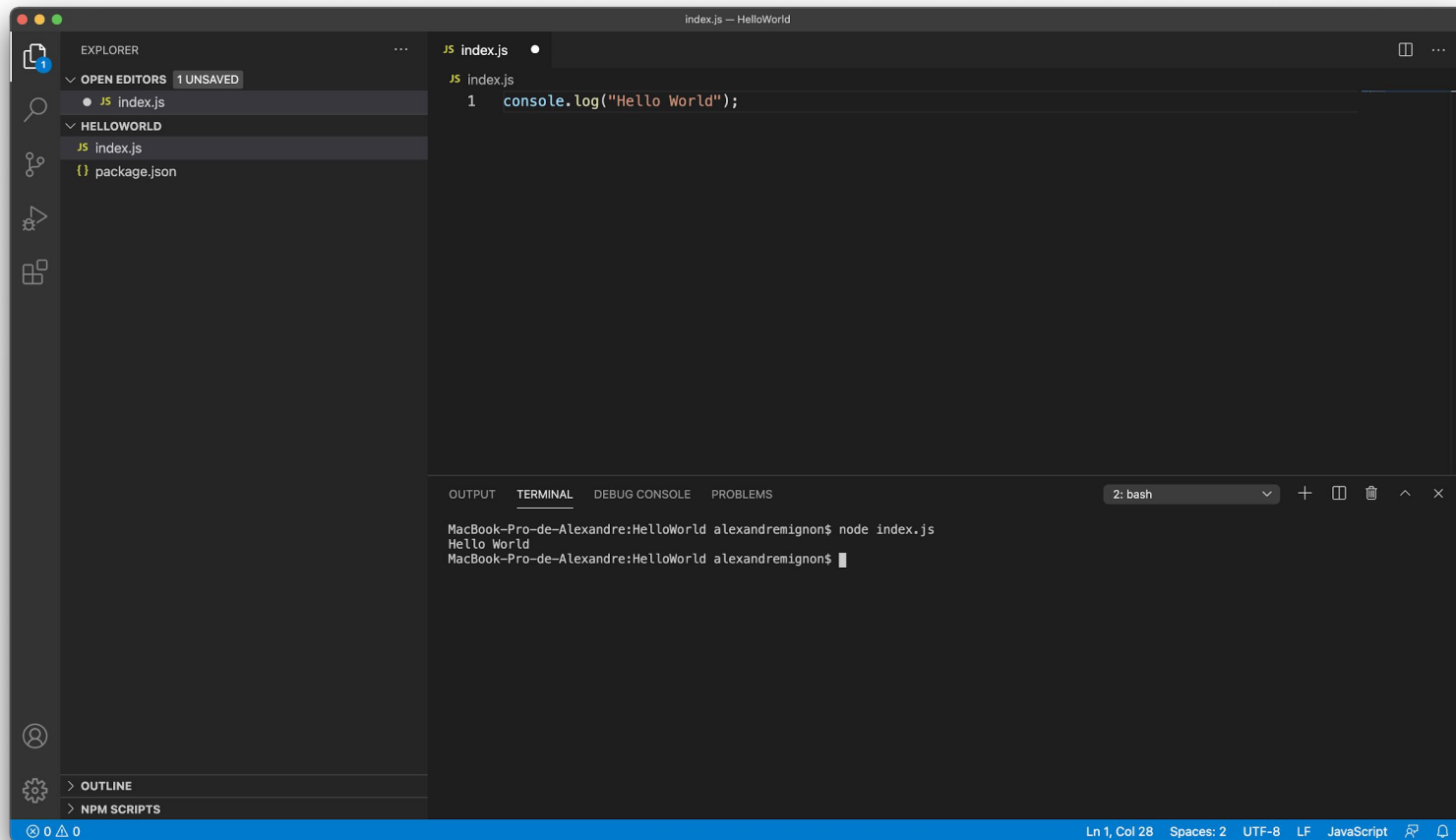
- Execute o **index.js** através do comando no terminal:

**node index.js**



```
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$ node index.js
Hello World
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$
```

# Configuração do Ambiente - Visual Studio Code



*BABEL*



- A sintaxe do ES6 que vamos utilizar em nossos projetos em React Native não é compatível com navegadores antigos caso queira usar para a Web.
- O Node.js <= v12 também não entende alguns novos comandos da linguagem, necessitando para estes casos citados o uso do **Babel**.
- O Babel irá converter a sintaxe do ES6+ para o ES5.

<https://babeljs.io/>

- Para a instalação do Babel em seu projeto, use o comando dentro da pasta que criamos:

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

- Após a instalação das bibliotecas mencionadas, seu arquivo **package.json** deverá estar parecido com o seguinte arquivo:

```
{ } package.json > ...
1  {
2    "name": "HelloWorld",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "devDependencies": {
13     "@babel/cli": "^7.13.10",
14     "@babel/core": "^7.13.10",
15     "@babel/preset-env": "^7.13.12"
16   }
17 }
18
```

- Crie um arquivo com o nome **babel.config.json** na raiz de seu projeto com o seguinte conteúdo:

```
B babel.config.json > ...
```

```
1  {
```

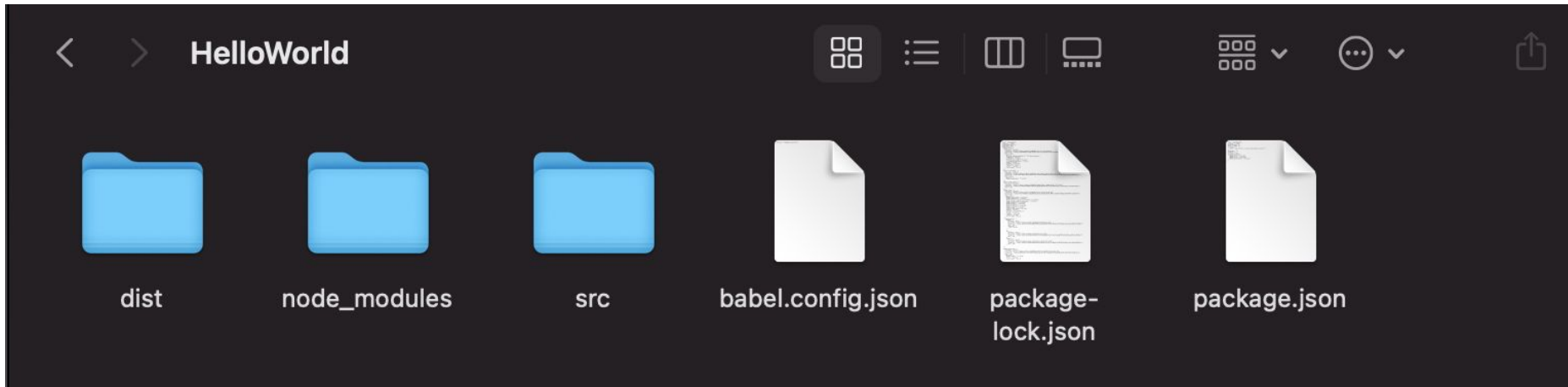
```
2    |   "presets": ["@babel/preset-env"]
```

```
3  }
```

- Adicione em seu projeto dois diretórios: **src** e **dist**.
- Adicione a linha destacada abaixo em seu arquivo **package.json**.

```
{ } package.json > { } scripts
1  {
2    "name": "HelloWorld",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "build": "babel src -d dist",
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC",
13   "devDependencies": {
14     "@babel/cli": "^7.13.10",
15     "@babel/core": "^7.13.10",
16     "@babel/preset-env": "^7.13.12"
17   }
18 }
```

- O diretório do seu projeto deve estar da seguinte maneira:



- No diretório **src** vamos colocar o nosso código fonte com sintaxe do ES6.
- O diretório **dist** é o diretório onde o Babel transcreve para a sintaxe do ES5.
- Mova se arquivo **index.js** para dentro de **src** e altere para:

```
src > JS index.js > ...  
1   const msg = "Hello World";  
2   console.log(msg);
```

- Vamos converter o arquivo executando na raiz do projeto o seguinte comando no terminal:

```
npm run build
```

- Repare que estamos chamando o *script* que colocamos no arquivo **package.json**.
- Compare o arquivo original e o convertido, respectivamente:

```
src > JS index.js > ...  
1  const msg = "Hello World";  
2  console.log(msg);
```

```
dist > JS index.js > ...  
1  "use strict";  
2  
3  var msg = "Hello World";  
4  console.log(msg);
```

- Execute o arquivo `dist/index.js` através do node: **node dist/index.js**



# ES6 Syntax

- O valor atribuído em uma variável declarada usando **var** tem escopo da função ou escopo global.

```
JS index.js  X
JS index.js > ...
1  var x = 1;
2
3  if (x === 1) {
4      var x = 2;
5
6      console.log(x);
7      // saída esperada: 2
8  }
9
10 console.log(x);
11 // saída esperada: 2
```

OUTPUT   TERMINAL   DEBUG CONSOLE   PROBLEMS

```
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$ node index.js
2
2
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$
```

- O valor atribuído em uma variável declarada usando **let** existirá somente dentro de um bloco.

```
JS index.js  X
JS index.js > ...
1  let x = 1;
2
3  if (x === 1) {
4      let x = 2;
5
6      console.log(x);
7      // saída esperada: 2
8  }
9
10 console.log(x);
11 // saída esperada: 1
```

OUTPUT   TERMINAL   DEBUG CONSOLE   PROBLEMS

```
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$ node index.js
2
1
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$
```

- Constante não permite ter seu valor alterado após sua inicialização;
- O valor deverá ser informado obrigatoriamente ao declarar uma constante .

```
JS index.js > ...
1  const naoPossoMudar = 10;
2  naoPossoMudar = 100; // irá causar um erro
3  console.log(naoPossoMudar)
```

OUTPUT   TERMINAL   DEBUG CONSOLE   PROBLEMS

```
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$ node index.js
/Users/alexandremignon/NodeJSProjects/HelloWorld/index.js:2
naoPossoMudar = 100; // irá causar um erro
               ^
TypeError: Assignment to constant variable.
    at Object.<anonymous> (/Users/alexandremignon/NodeJSProjects/HelloWorld/index.js:2:15)
    at Module._compile (internal/modules/cjs/loader.js:1063:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1092:10)
    at Module.load (internal/modules/cjs/loader.js:928:32)
    at Function.Module._load (internal/modules/cjs/loader.js:769:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:72:12)
    at internal/main/run_main_module.js:17:47
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$
```

- Template strings permite interpolar as Strings de forma fácil.

```
JS index.js > ...  
1   let a = 10;  
2   let b = 20;  
3  
4   // Sem Template String  
5   let msg1 = "A = " + a + " B = " + b;  
6   console.log(msg1);  
7  
8   // Com Template String  
9   let msg2 = `A = ${a} B = ${b}`;  
10  console.log(msg2);
```

```
JS index.js > ...  
1  function soma(a, b) {  
2    |   return a + b;  
3  }  
4  
5  console.log( soma(3, 4) );
```

- Na declaração de funções, é possível declarar valores padrão (*default*) para cada parâmetro.

```
JS index.js > ...  
1  // com valor default para o parâmetro b  
2  function soma(a, b = 0) {  
3    |   return a + b;  
4  }  
5  
6  console.log( soma(3) );
```

JS index.js > ...

```
1 let digaOla = function() {  
2   | console.log("Ola");  
3 }  
4 digaOla();
```

JS index.js > ...

```
1 let idade = 16;  
2 let bemVindo;  
3 √ if (idade < 18) {  
4   | bemVindo = function() {  
5     | console.log("Olá");  
6     | }  
7   | } else {  
8     | bemVindo = function() {  
9       | console.log("Saudações");  
10    | }  
11    | }  
12 bemVindo();
```



```
JS index.js > ...
1  let soma = (a, b) => a + b;
2  console.log( soma(3, 4) );
3
4  let dobro = n => n * 2;
5  console.log( dobro(5) );
6
7  let digaOla = () => console.log("Olá");
8  digaOla();
9
10 let digaHello = _ => console.log("Hello");
11 digaHello();
12
13 // multiplas linhas
14 ✓ let subtracao = (a, b) => {
15   |   let resultado = a - b;
16   |   return resultado;
17   | }
18 console.log( subtracao(10, 7) );
```

- O Spread Operator é representado por três pontos ... e converte um array em elementos individuais.

```
JS index.js > ...  
1  const meuArray = [1, 2, 3];  
2  
3  console.log(meuArray);  
4  
5  console.log(...meuArray);  
6  
7  console.log([...meuArray, 4, 5, 6]);  
  
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS  
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$ node index.js  
[ 1, 2, 3 ]  
1 2 3  
[ 1, 2, 3, 4, 5, 6 ]  
MacBook-Pro-de-Alexandre:HelloWorld alexandremignon$
```

- A técnica de Destructuring permite a quebra de estrutura complexas em partes únicas.
- É possível aplicá-la em Arrays e Objetos:

```
src > JS index.js > ...  
1  const meuArray = ["azul", "vermelho", "verde"];  
2  const [cor1, cor2, cor3] = meuArray;  
3  
4  console.log(cor1);  
5  console.log(cor2);  
6  console.log(cor3);  
7  
8  const meuObjeto = {x : 10, y : 20, z : 30};  
9  const {x, y} = meuObjeto;  
10 console.log(x);  
11 console.log(y);
```

# JavaScript Orientado a Objetos

- Objetos em JavaScript podem ser criados sem o uso de classes, alimentando estruturas em tempo de execução.

JS index.js > ...

```
1  var carro1 = {marca: "Fiat", modelo: "500", cor: "branco"};  
2  var carro2 = {marca: "Reanult", modelo: "Sander", cor: "preto"};  
3  
4  console.log(carro1);  
5  console.log(carro1.marca);  
6  console.log(carro1.modelo);  
7  console.log(carro1.cor);  
8
```

OUTPUT   TERMINAL   DEBUG CONSOLE   PROBLEMS

```
MacBook-Pro-de-Alexandre:Teste alexandremignon$ node index.js  
{ marca: 'Fiat', modelo: '500', cor: 'branco' }  
Fiat  
500  
branco
```

- Definindo objetos com declaração literal.

```
JS index.js > ...  
1  var pessoa = {  
2      nome : "Mario",  
3      sobrenome : "de Andrade",  
4      id : 5566,  
5      nomeCompleto : function() {  
6          return this.nome + " " + this.sobrenome;  
7      }  
8  };  
9  
10 console.log(pessoa);  
11 console.log(pessoa.nomeCompleto());
```

- Instância de objetos a partir de implementação vazia.

```
JS index.js > ...
1  function criarNovaPessoa(nome) {
2      // obj vazio
3      var obj = {};
4
5      // atributo
6      obj.nome = nome;
7
8      // método
9      obj.saudacao = function() {
10         console.log("Olá, " + obj.nome);
11     };
12
13     return obj;
14 }
15
16 var pessoa = criarNovaPessoa("Ale");
17 pessoa.saudacao();
```

- Instância de objetos utilizando função que representa uma classe e o operador **new**.

```
JS index.js > ...  
1  function Pessoa(nome) {  
2      this.nome = nome;  
3      this.saudacao = function() {  
4          console.log("Ola, " + this.nome);  
5      };  
6  }  
7  
8  var pessoa = new Pessoa("Ale");  
9  pessoa.saudacao();  
10
```



- No ES6 é possível escrever códigos orientados a objetos de forma clara, com sintaxe parecida com outras linguagens de programação.

```
JS index.js > ...
1  class Pessoa {
2    // Construtor
3    constructor() {
4      this.nome = "";
5      this.email = "";
6    }
7  }
8
9  const pessoa = new Pessoa();
10 pessoa.nome = "João";
11
12 console.log(pessoa.nome);
```

- O ES6 não possui modificadores de visibilidade: público, privado ou protegido. Para deixar claro que algo é privado, usamos o `_` (underline) antes do nome do atributo e manipulamos seu valor através de métodos `get` e `set`.

```
JS index.js > ...
1  class Pessoa {
2    constructor() {
3      this._nome = ""; // "privado"
4    }
5
6    get nome() {
7      return this._nome;
8    }
9
10   set nome(value) {
11     this._nome = value;
12   }
13 }
14
15 const pessoa = new Pessoa();
16 pessoa.nome = "João"; // sem a sintaxe de método
17
18 console.log(pessoa.nome);
```

- Definindo métodos estáticos.

```
JS index.js > ...
1  class Pessoa {
2    constructor(nome) {
3      |   this._nome = nome;
4    }
5
6    get nome() {
7      |   return this._nome;
8    }
9
10   static qualClasseSou() {
11     |   console.log("Sou a classe Pessoa.");
12   }
13 }
14
15 let pessoa = new Pessoa("Ale");
16 console.log(pessoa.nome);
17 Pessoa.qualClasseSou();
```

```
JS index.js > [❌] suv
1  class Carro {
2    |   constructor(marca) {
3    |     |   this.marca = marca;
4    |   }
5  }
6
7  class SUV extends Carro {
8    |   constructor(marca, tipoCambio) {
9    |     |   super(marca);
10   |     |   this.tipoCambio = tipoCambio;
11   |   }
12 }
13
14 var suv = new SUV('Honda', 'Automatico');
15 console.log(suv.marca);
16 console.log(suv.tipoCambio);
```

- Módulos são pedaços de códigos JavaScript escritos em um arquivo de modo que seja possível usar apenas um trecho deste arquivo dentro de outro arquivo.
- Eles podem ser **exportados** de forma **nomeada** ou forma **default**.
- Dentro de outro arquivo, eles serão importados, podendo usar a técnica de **Destructuring**.
- Isso facilita a componentização do código, deixando-o mais organizado e legível.

- Classes, funções e variáveis presentes em um arquivo podem ser exportadas de forma nomeada. Ex: arquivo **MeuModulo.js**.

```
JS MeuModulo.js > ...
1  class MinhaClasse {
2    show() {
3      console.log("Um método dentro da classe.");
4    }
5  }
6
7  let minhaFuncao = _ => console.log("Uma função");
8  let minhaVariavel = 10;
9
10 export {MinhaClasse};
11 export {minhaFuncao};
12 export {minhaVariavel};
```

- Para importar somente a **MinhaClasse** do arquivo **MeuModulo.js** dentro do **index.js**:

```
JS index.js > ...  
1  import {MinhaClasse} from './MeuModulo.js'  
2  
3  let a = new MinhaClasse();  
4  a.show();
```

- Não esqueça de usar o **Babel** para conseguir testar este código no Node.JS!!!

- Para executar usando o NodeJS, sem o uso do Babel, adicione a linha destacada abaixo no arquivo **package.json**.

```
{ } package.json > ...
1  {
2    "name": "Teste",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "type": "module",
   ▶ Debug
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC"
13 }
```



- Seu módulo poderá ter um **default export**. Isso é útil quando o arquivo possui apenas uma classe ou método para ser exportado, não precisando fazer o Destructuring para importar.

```
JS MeuModulo.js > ...  
1  export default class MinhaClasse {  
2    show() {  
3      console.log("Um método dentro da classe.");  
4    }  
5  }
```

```
JS index.js > ...  
1  import MinhaClasse from './MeuModulo.js'  
2  
3  let a = new MinhaClasse();  
4  a.show();
```

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>
- <https://javascript.info/>
- <https://jsfiddle.net/>
- <https://codepen.io/>

++

45697096

# Dúvidas?

◇◇◇

45697096

45697096

■ ■ ■



Copyright © 2020 Prof. Alexandre dos Santos Mignon <[profalexandre.mignon@fiap.com.br](mailto:profalexandre.mignon@fiap.com.br)>

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).