

Estrutura de Dados

Referências Bibliográficas:

GRONER, Loiane. **Estruturas de dados e algoritmos com JavaScript**: Escreva um código JavaScript complexo e eficaz usando a mais recente ECMAScript. 2ª ed. São Paulo: Novatec, 2019.



Discutiremos, inicialmente os seguintes assuntos:

- ✓ Configuração do ambiente e o básico sobre JavaScript;
- ✓ Variáveis, escopo de variáveis e operadores;
- ✓ Estruturas de controle e funções;
- ✓ Programação orientada a objetos em JavaScript;
- ✓ Depuração e ferramentas.



- Uma das vantagens da linguagem JavaScript, é que você não precisa instalar nem configurar um ambiente complicado para começar a usá-la:
 - ✓ O ambiente **mais simples** é usar o console do **Developer Tools** nos navegadores (Chrome, Firefox, Safari e Edge).
 - Tecla “F12” ou “ctrl + shift + i” no Chrome
- Experimente digitar os seguintes comandos no console do Chrome e teclar <Enter>:
 - ✓ `document.write('Hello, World!');`
 - ✓ `console.log('Boa noite Turma!');`
 - ✓ `alert('Vamos aprender Estrutura de Dados?');`

- Uma das vantagens é configurar um ambiente de desenvolvimento

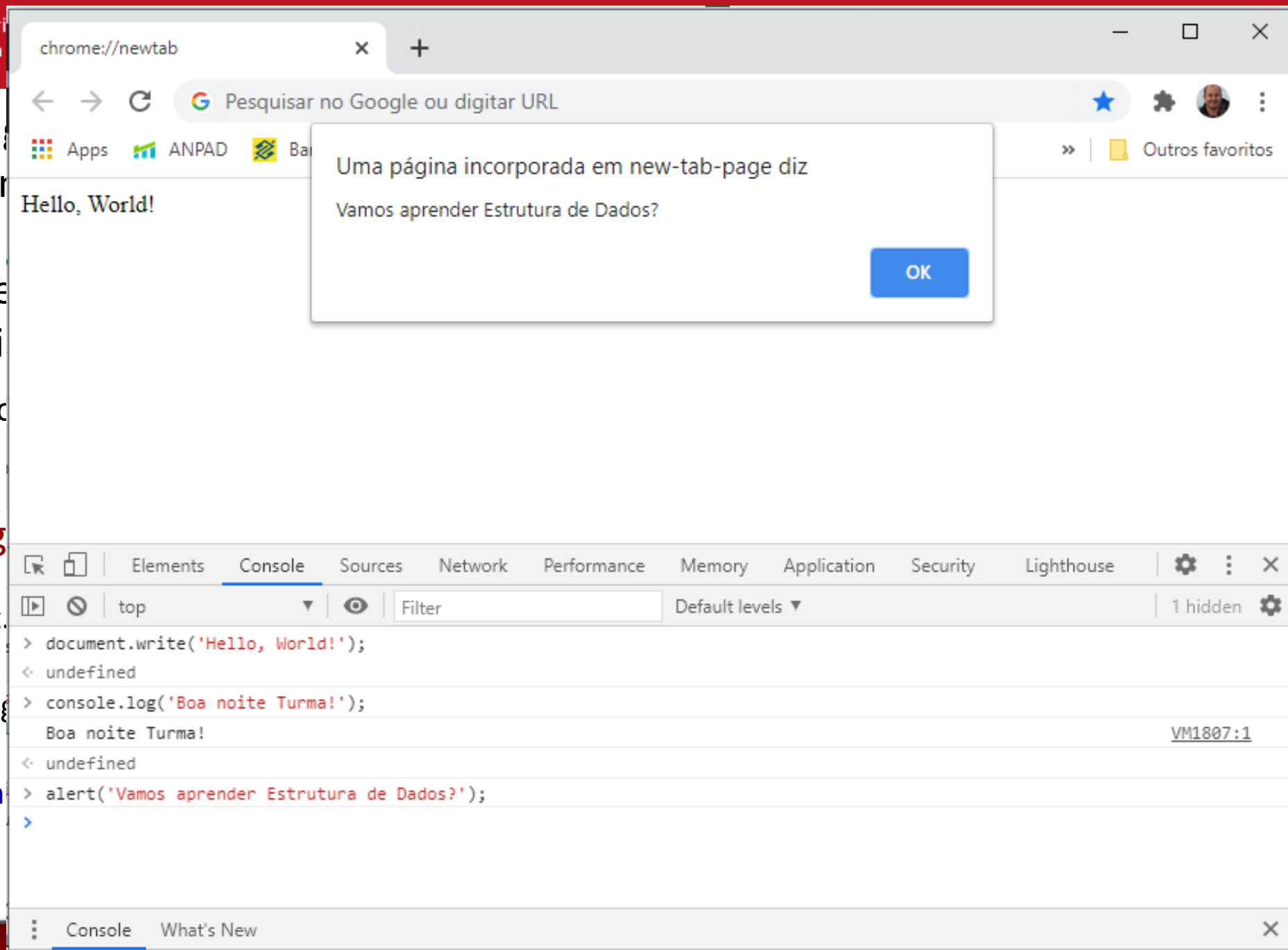
- ✓ O ambiente de desenvolvimento (Chrome, Firefox, VS Code)
 - Tecla “F12” para abrir o DevTools

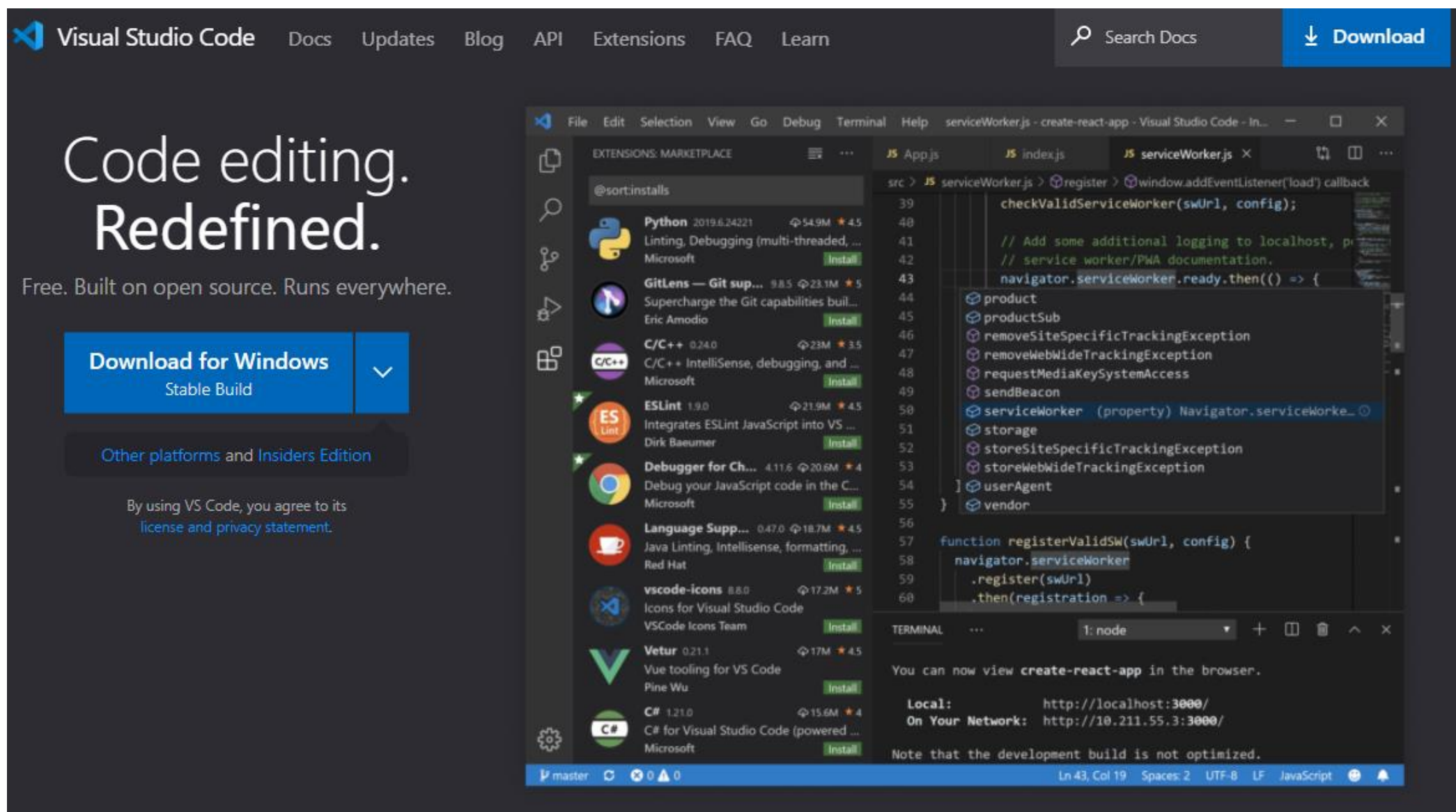
- Experimente digitar no console:

- ✓ `document.write('Hello, World!');`

- ✓ `console.log('Boa noite Turma!');`

- ✓ `alert('Vamos aprender Estrutura de Dados?');`





The image shows the Visual Studio Code website on the left and a screenshot of the Visual Studio Code interface on the right.

Visual Studio Code Website:

- Navigation: Visual Studio Code, Docs, Updates, Blog, API, Extensions, FAQ, Learn.
- Search: Search Docs
- Download: Download
- Headline: Code editing. Redefined.
- Text: Free. Built on open source. Runs everywhere.
- Buttons: Download for Windows (Stable Build), Other platforms and Insiders Edition.
- Text: By using VS Code, you agree to its [license and privacy statement](#).

Visual Studio Code Interface:

- File Explorer: Extensions: Marketplace.
- Search Bar: @sortinstalls.
- Extension List:
 - Python 2019.6.24221 54.9M 4.5 (Linting, Debugging (multi-threaded, ... Microsoft) Install)
 - GitLens — Git sup... 9.8.5 23.1M 5 (Supercharge the Git capabilities built... Eric Amodio Install)
 - C/C++ 0.24.0 23M 3.5 (C/C++ IntelliSense, debugging, and ... Microsoft Install)
 - ESLint 1.9.0 21.9M 4.5 (Integrates ESLint JavaScript into VS ... Dirk Baumer Install)
 - Debugger for Ch... 4.11.6 20.6M 4 (Debug your JavaScript code in the C... Microsoft Install)
 - Language Supp... 0.47.0 18.7M 4.5 (Java Linting, Intellisense, formatting, ... Red Hat Install)
 - vscode-icons 8.8.0 17.2M 5 (Icons for Visual Studio Code VSCo Icons Team Install)
 - Vetur 0.21.1 17M 4.5 (Vue tooling for VS Code Pine Wu Install)
 - C# 1.21.0 15.6M 4 (C# for Visual Studio Code (powered ... Microsoft Install)
- Code Editor: JS serviceWorker.js. Content:


```

src > JS serviceWorker.js > register > window.addEventListener('load') callback
39
40
41
42 // Add some additional logging to localhost, p
43 // service worker/PWA documentation.
44
45 navigator.serviceWorker.ready.then(() => {
46   product
47   productSub
48   removeSiteSpecificTrackingException
49   removeWebWideTrackingException
50   requestMediaKeySystemAccess
51   sendBeacon
52   serviceWorker (property) Navigator.serviceWorke...
53   storage
54   storeSiteSpecificTrackingException
55   storeWebWideTrackingException
56   userAgent
57   vendor
58
59 function registerValidSW(swUrl, config) {
60   navigator.serviceWorker
61     .register(swUrl)
62     .then(registration => {

```
- Terminal: 1: node. Output:


```

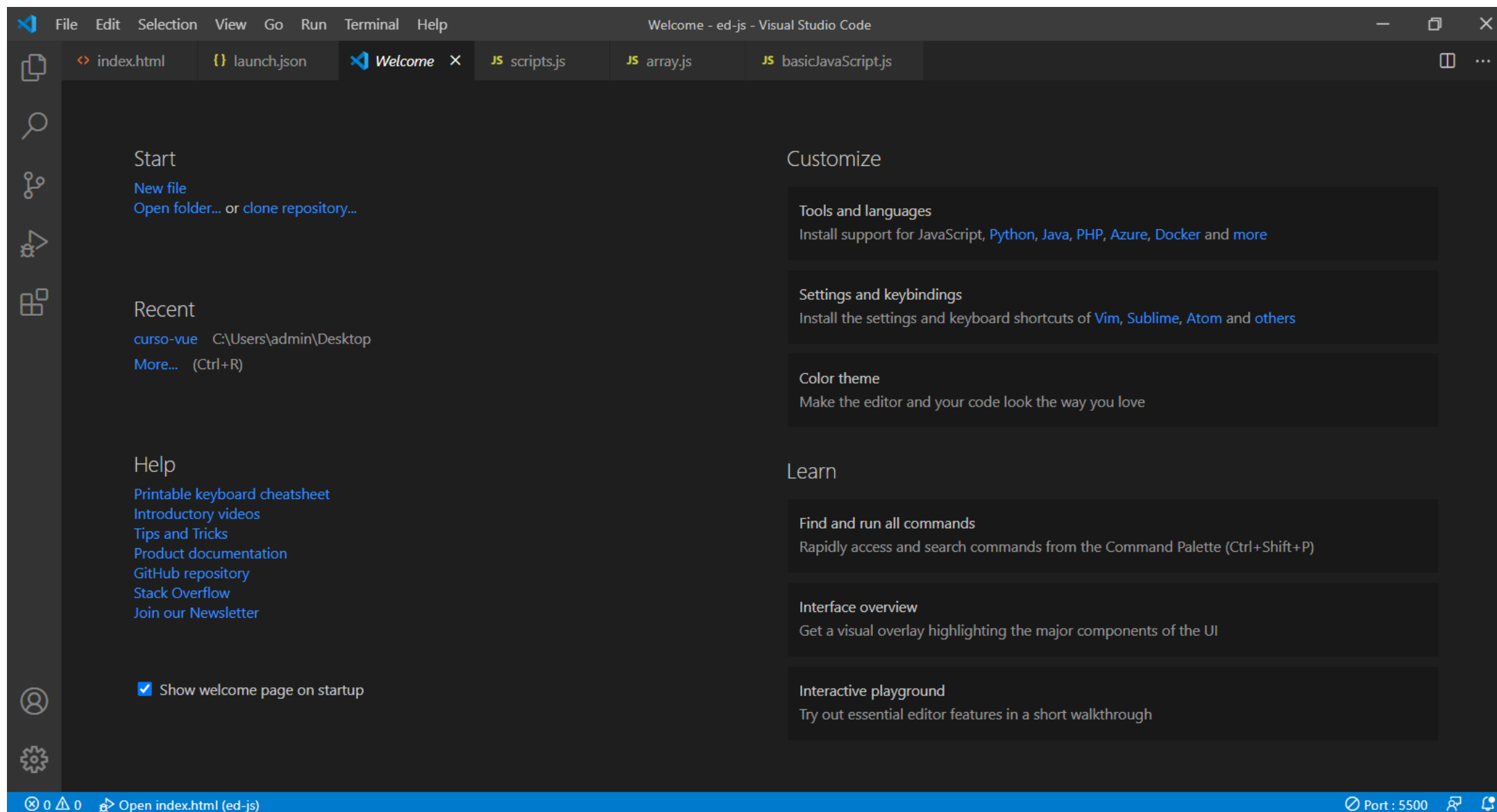
You can now view create-react-app in the browser.

Local:      http://localhost:3000/
On Your Network: http://10.211.55.3:3000/

Note that the development build is not optimized.

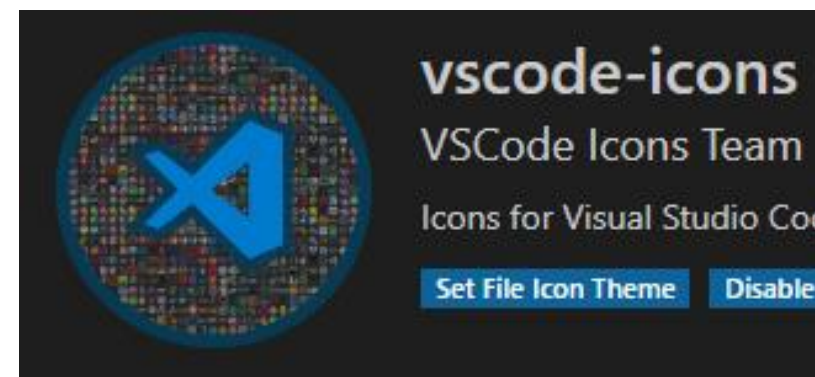
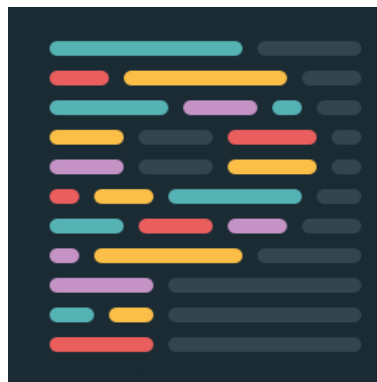
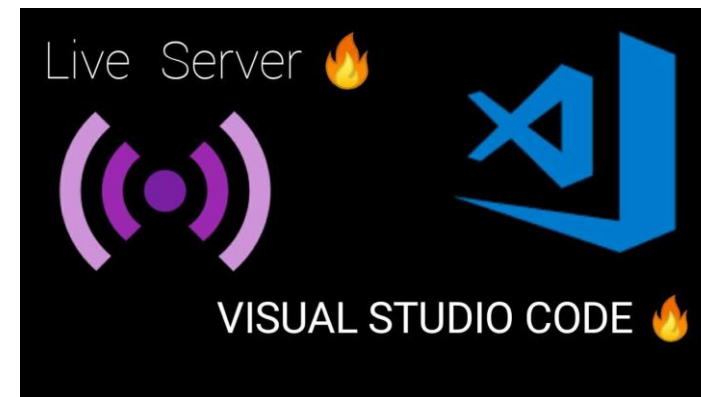
```
- Status Bar: master, 0 errors, 0 warnings, 0 info. Ln 43, Col 19. Spaces: 2. UTF-8. LF. JavaScript.

Primeira página do VsCode



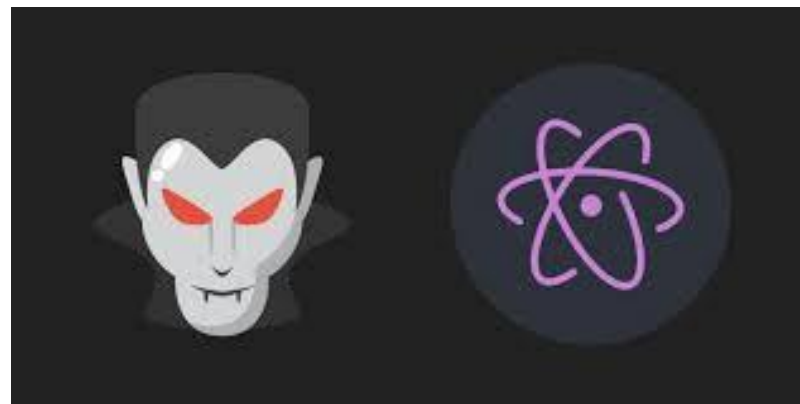
Plugins essenciais do Vscode

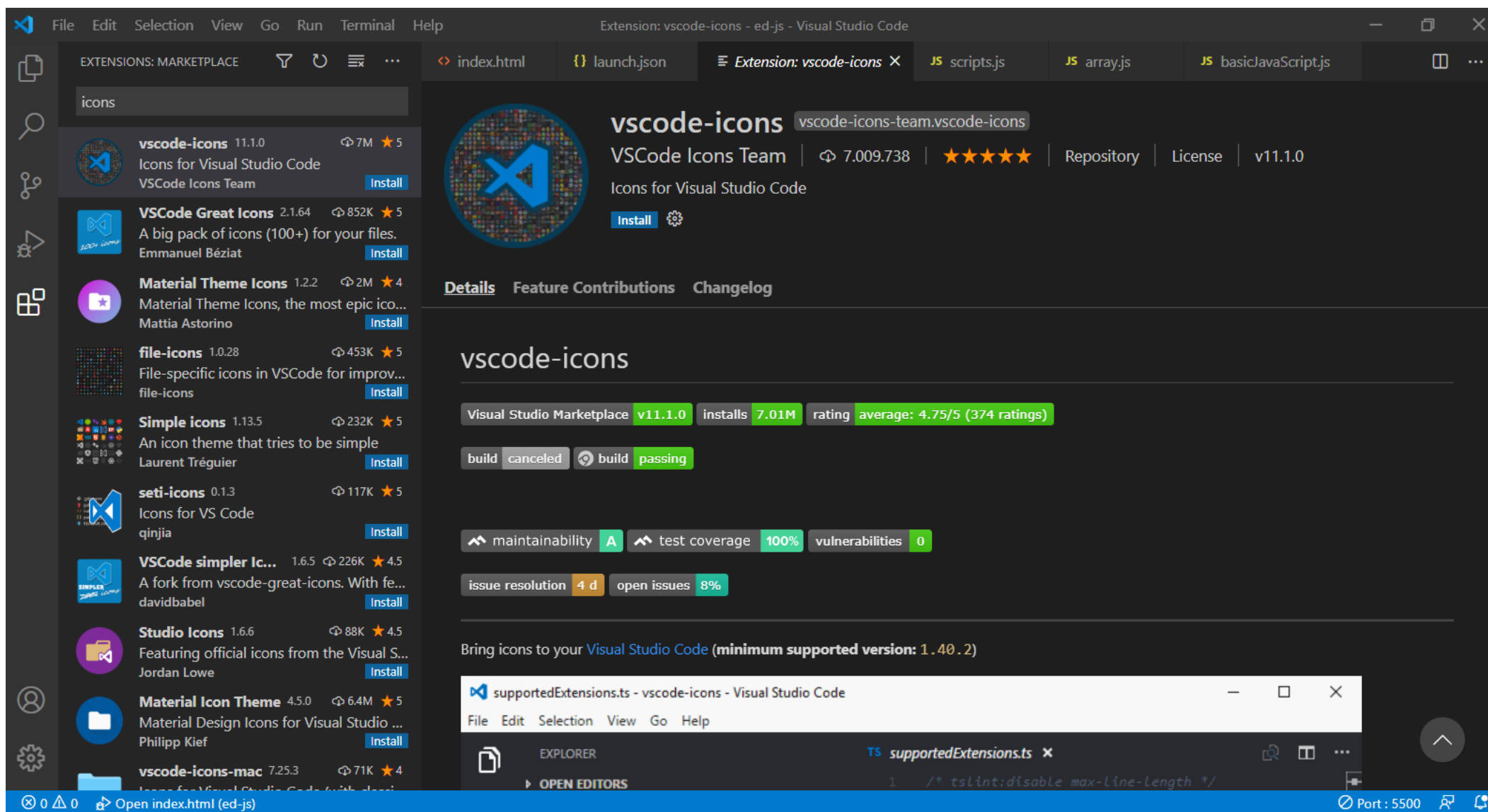
- Live Server – servidor local de execução
- ESLint – aponta erros no código
- Prettier – deixa o código mais bonito e legível
- Vscode-icons



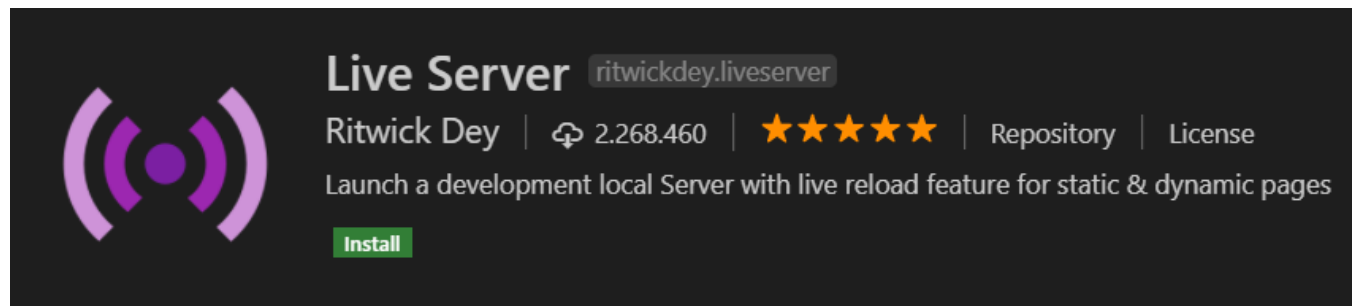
Plugins opcionais do Vscode

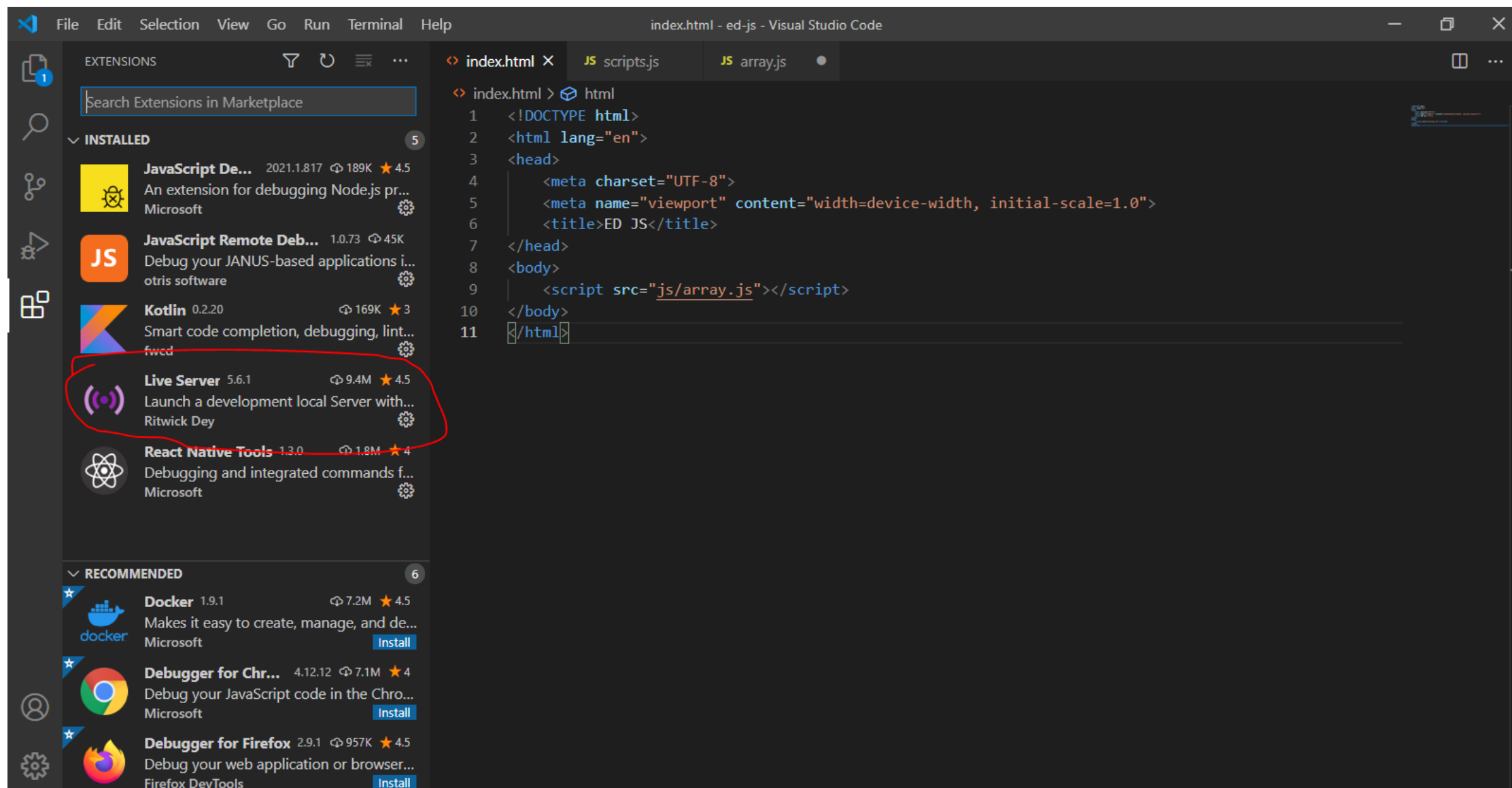
- Bracket Pair Colorizer – colore os pares de parêntesis, chaves e colchetes
- Material Icon Theme – Tema de ícones para arquivos e pastas
- Dracula Theme – Tema bacana demais





- A segunda opção de ambiente que você pode instalar em seu computador também é simples, mas exige a instalação de um servidor web.
- Se um arquivo HTML contiver somente código JavaScript simples, ele poderá ser executado no navegador clicando com o botão direito do mouse no arquivo **HTML** e selecionando a opção **Open With** + <nome do servidor web Local>.
- O código que desenvolveremos é simples e poderá ser executado usando essa abordagem.
- No entanto, é sempre bom ter um servidor web instalado.
- Existe um plug-in como extensão de servidor web local ([Live Server](#)), que pode ser instalado no editor **VsCode**.





http-server do Node.js

- A terceira opção é ter um ambiente **100% JavaScript!**
- Pra instalar acesse <http://nodejs.org>, faça o download e instale o **Node.js**.
- O Node **é um ambiente de execução fora do browser**. *Opte pela Versão LTS (Long Term Support)*
- Depois de instalar, abra um terminal ou Prompt (se estiver no Windows) e execute o seguinte comando:
`npm install http-server -g`
- O comando instalará o http-server, um servidor JavaScript

nodejs.org/en/download/

IPAD Banco do Brasil Para você - Nacion... Teste ANPAD Cidadão 81 launch pad Siga Aceite Docker: Ferramenta... Siga Preprod (183) Workshop de... (18)

node

HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS

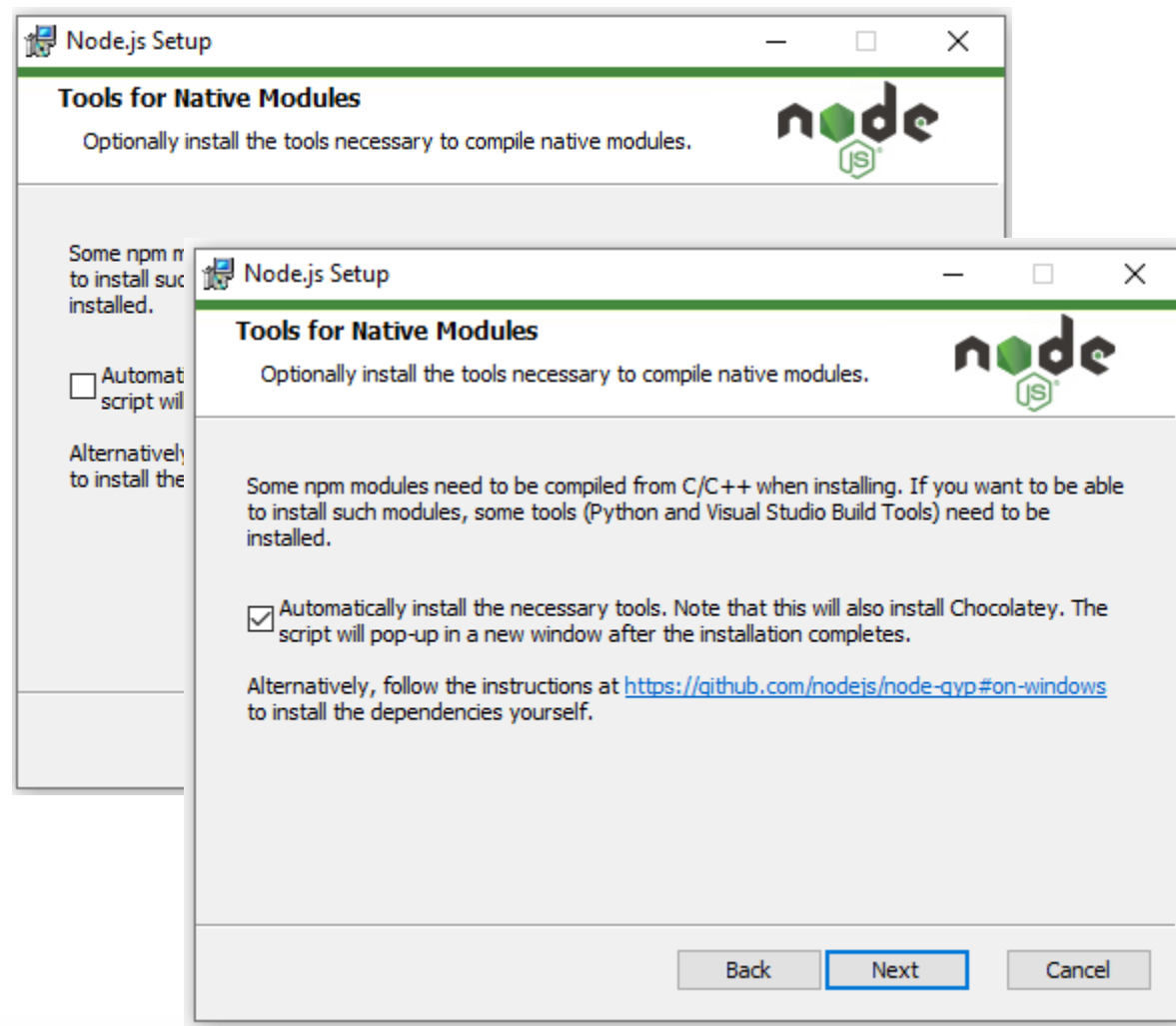
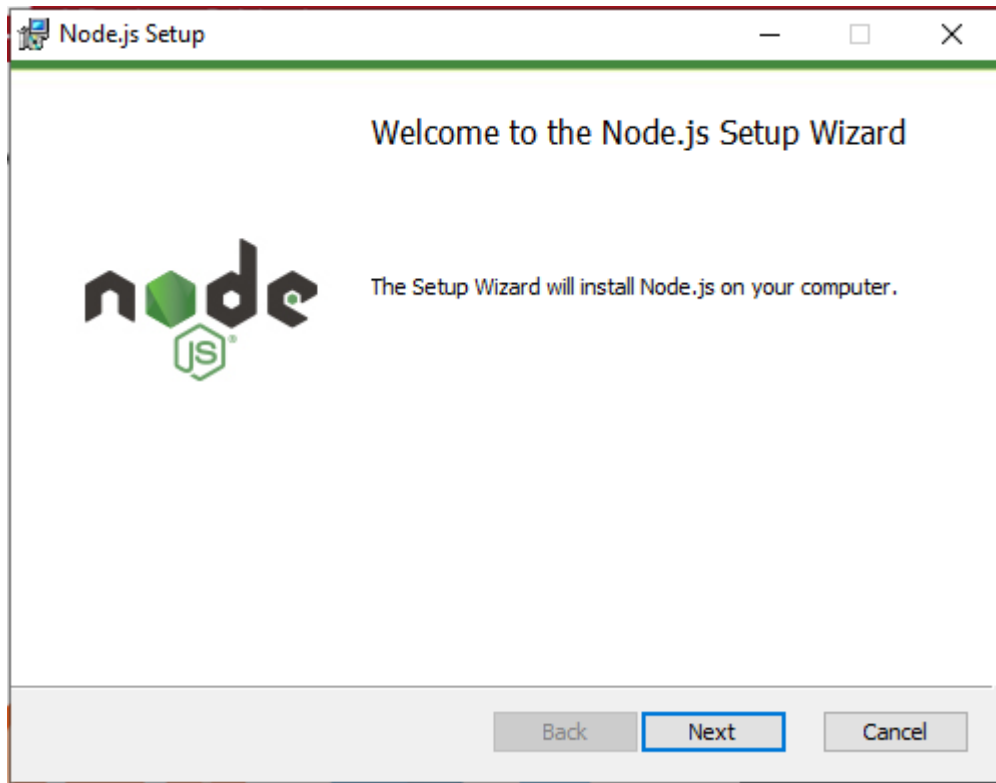
Downloads

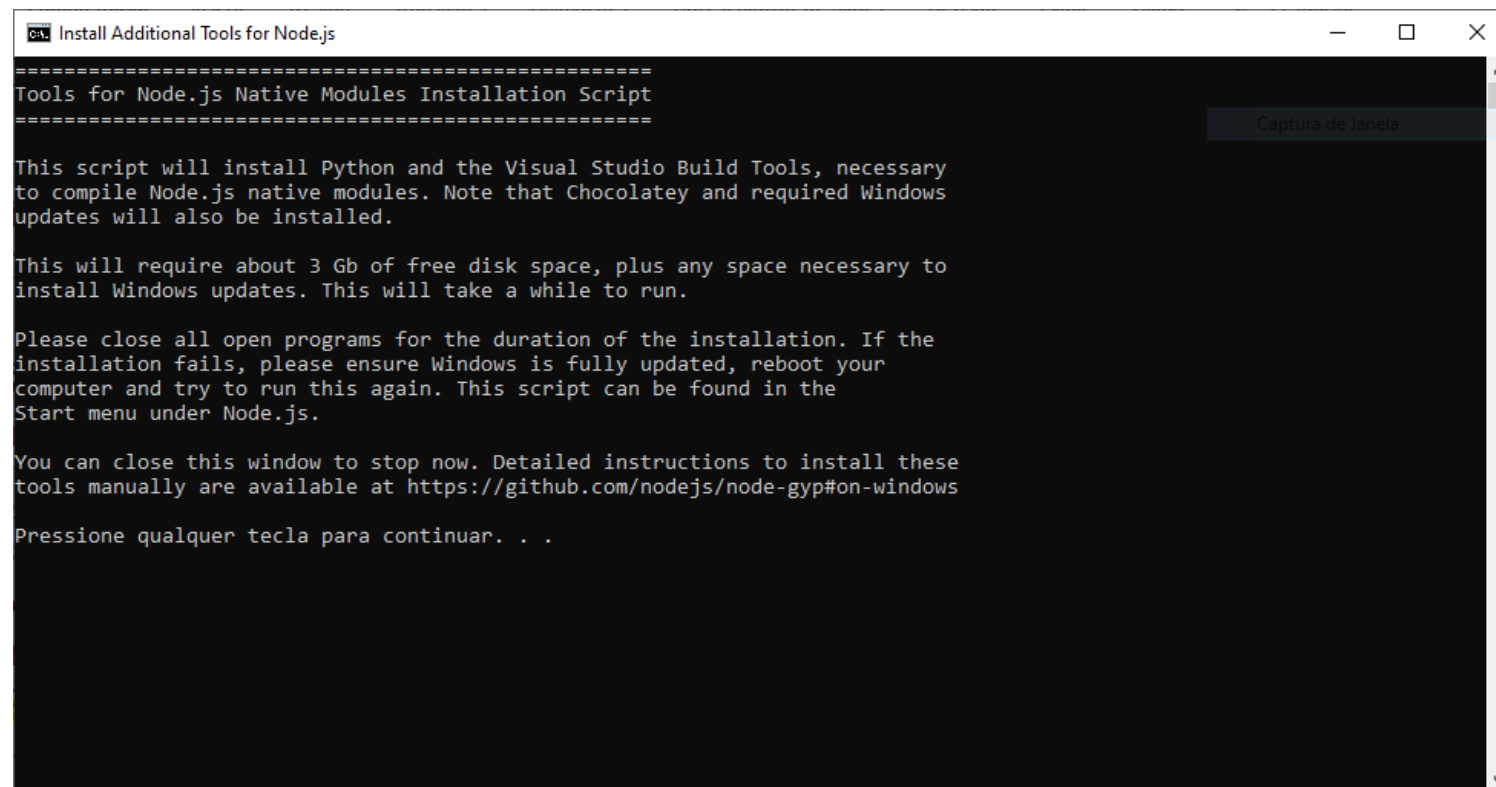
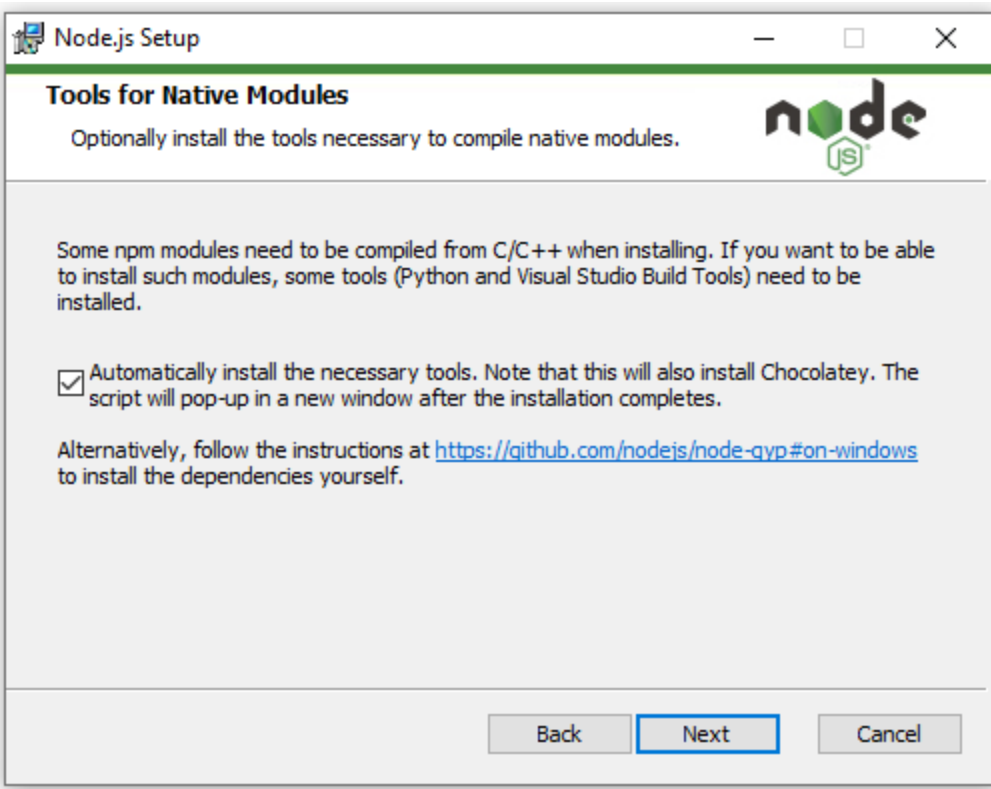
Latest LTS Version: **14.15.4** (includes npm 6.14.10)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

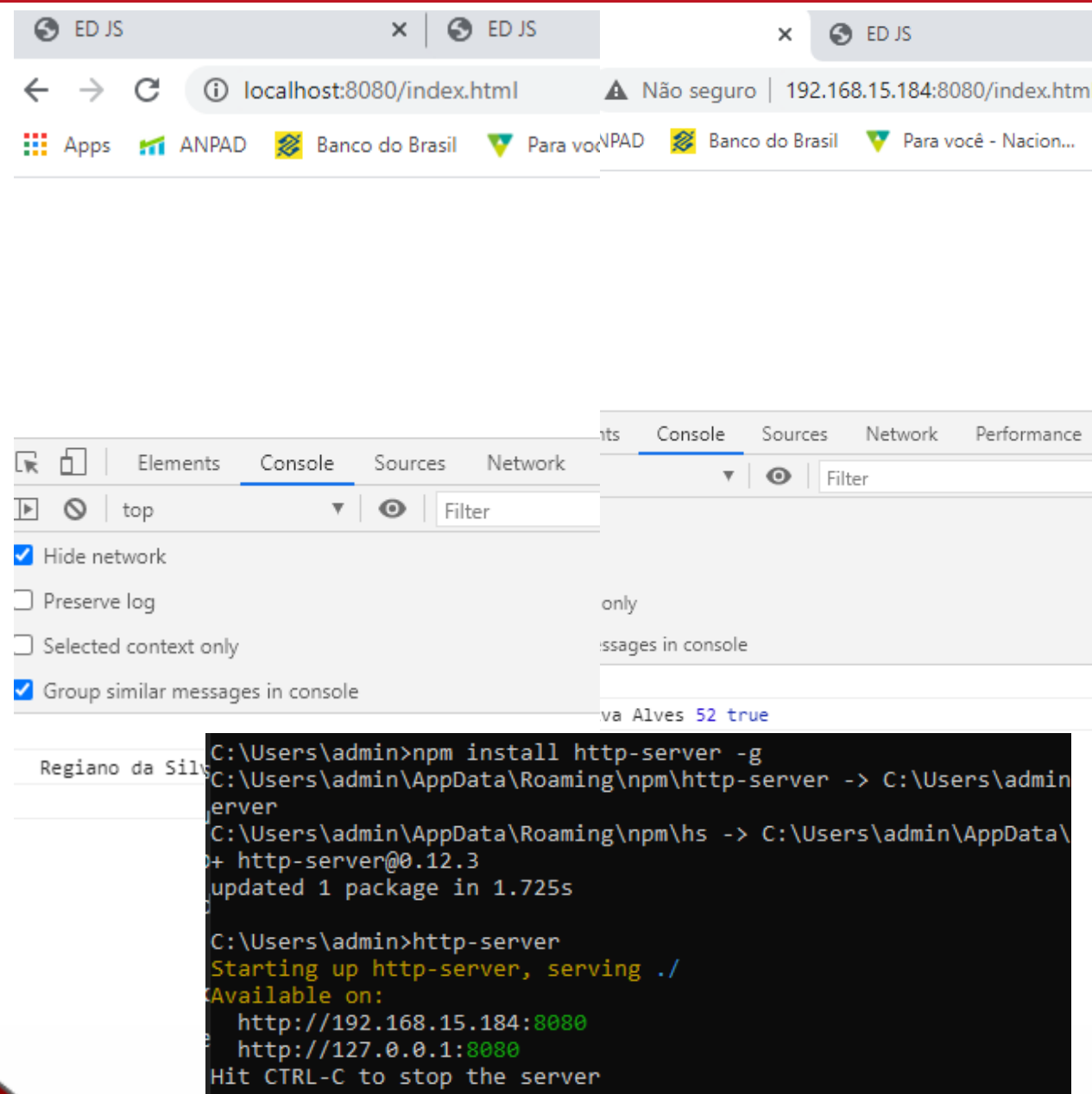
LTS Recommended For Most Users	Current Latest Features	
 Windows Installer node-v14.15.4-x64.msi	 macOS Installer node-v14.15.4.pkg	 Source Code node-v14.15.4.tar.gz

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit	
macOS Binary (.tar.gz)	64-bit	
Linux Binaries (x64)	64-bit	
Linux Binaries (ARM)	64-bit	
Source Code	ARMv7	ARMv8
	node-v14.15.4.tar.gz	





- ✓ Depois de instalar, abra um terminal ou Prompt (se estiver no Windows) e execute o seguinte comando:
`npm install http-server -g`
- ✓ O comando instalará o **http-server**, um servidor JavaScript.
- ✓ Para iniciar um servidor e executar os exemplos de código na aplicação do Terminal, mude o diretório para a pasta que contém o código-fonte e digite **http-server**, como mostra o prompt.
- ✓ Para executar os exemplos, abra o **navegador** e acesse o **localhost** na porta especificada pelo comando http-server.
- ✓ Clique no navegador **F12** ou **Ctrl+Shift+I** para visualizar a console nas Ferramentas do Desenvolvedor.



Executando JavaScript com o server runtime Node JS

- Para iniciar um servidor e executar os exemplos de código, mude o diretório para a pasta que contém o código fonte e digite **http-server**
- Para executar um script, basta digitar no prompt da pasta p comando **node + o nome do arquivo** com extensão .js
- Experimente criar um arquivo com o nome **array.js** dentro do diretório **/js**
- Digite as linhas de código da figura;
- Salve o arquivo (**ctrl + s**);
- Clique na aba Terminal do VsCode;
- Digite o comando **node + array.js**
- Caso não apresente o caminho da pasta onde está o arquivo, você pode indicar o caminho da pasta, ou direcionar com o comando **cd + nome_da_pasta**.

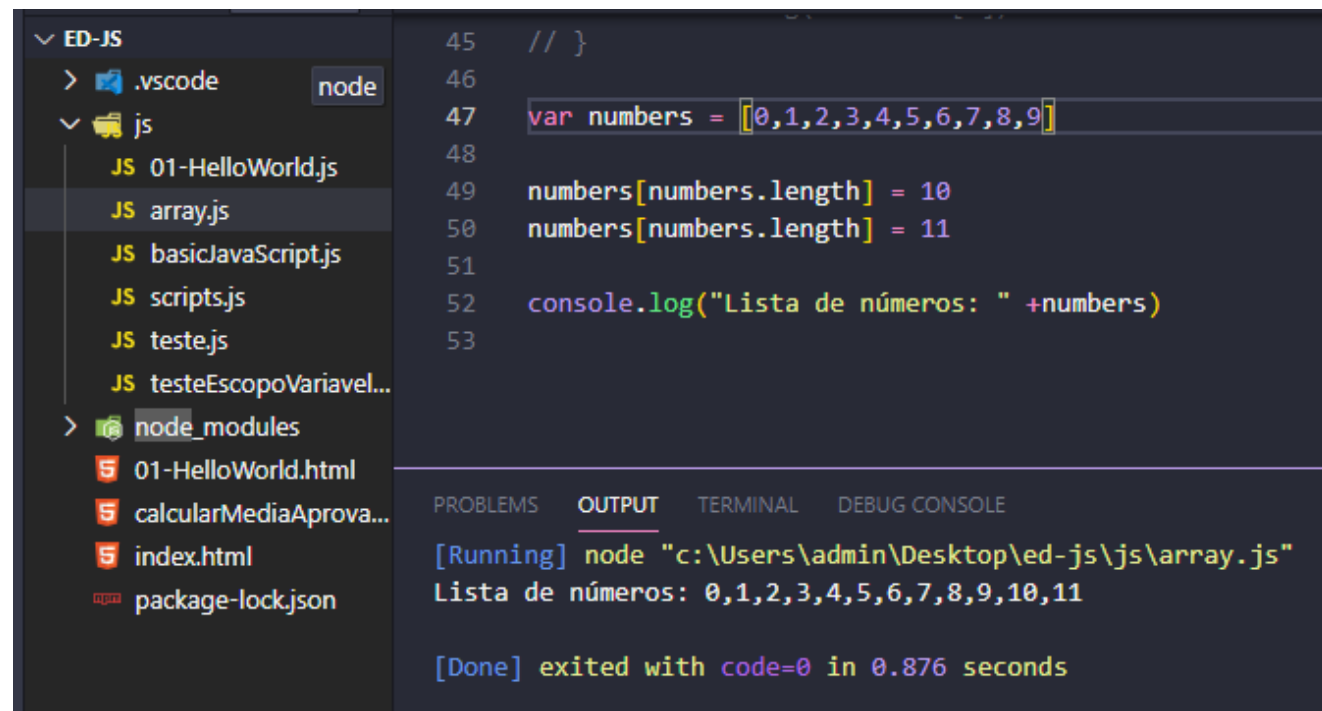
```
47  var numbers = [0,1,2,3,4,5,6,7,8,9]
48
49  numbers[numbers.length] = 10
50  numbers[numbers.length] = 11
51
52  console.log("Lista de números: " +numbers)
53
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\admin\Desktop\ed-js\js> node array.js
Lista de números: 0,1,2,3,4,5,6,7,8,9,10,11
PS C:\Users\admin\Desktop\ed-js\js> |
```

Executando JavaScript com o server runtime Node JS

- Outro modo simples de executar um script de comandos JavaScript, é somente clicando com o botão direito no arquivo.js e selecionar a opção **Run Code**, ou digitando juntas as teclas Ctrl + Alt + N
- O resultado irá aparecer na aba **Output**, veja na figura:
- Foi criado uma lista (array) de números com 10 elementos, depois foi inserido mais dois números na lista, e apresentado o conteúdo da lista com o método console.log()



```
45 // }
46
47 var numbers = [0,1,2,3,4,5,6,7,8,9]
48
49 numbers[numbers.length] = 10
50 numbers[numbers.length] = 11
51
52 console.log("Lista de números: " +numbers)
53
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

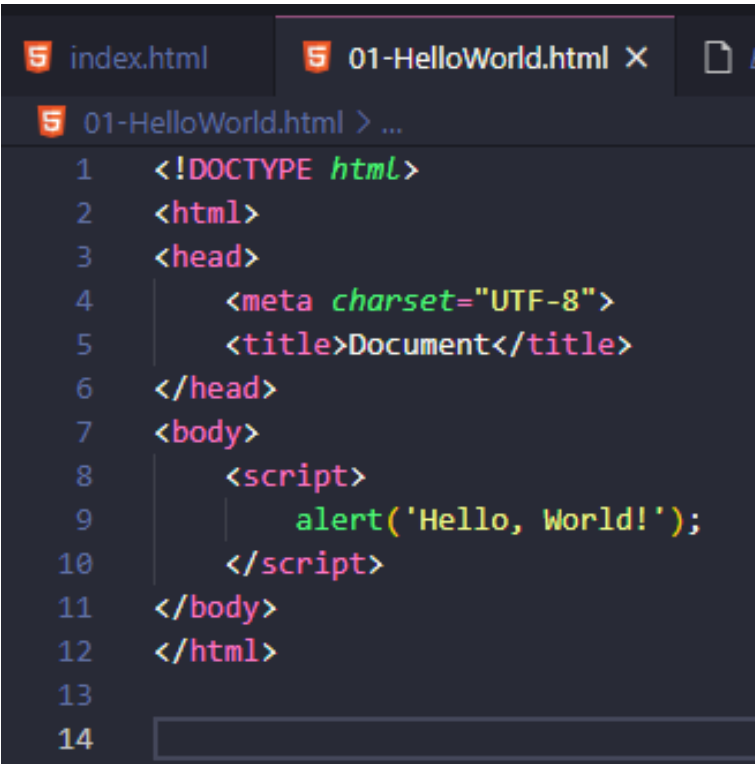
[Running] node "c:\Users\admin\Desktop\ed-js\js\array.js"

Lista de números: 0,1,2,3,4,5,6,7,8,9,10,11

[Done] exited with code=0 in 0.876 seconds

Executando meu primeiro comando JavaScript

- ✓ Crie um arquivo no VsCode com o nome 01-HelloWorld.html
- ✓ Depois digite o atalho “**sinal de exclamação + tab** (! + tab)” que ele gera uma estrutura básica de uma página.
- ✓ Insira na tag body, uma nova tag `<script></script>` e inclua o método `alert('Hello World!');`
- ✓ Para executar é só abrir em um browser
- ✓ Ou executar com plugin do servidor local Live-Server, acionando com o botão direito.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <title>Document</title>
6 </head>
7 <body>
8   <script>
9     alert('Hello, World!');
10  </script>
11 </body>
12 </html>
13
14
```

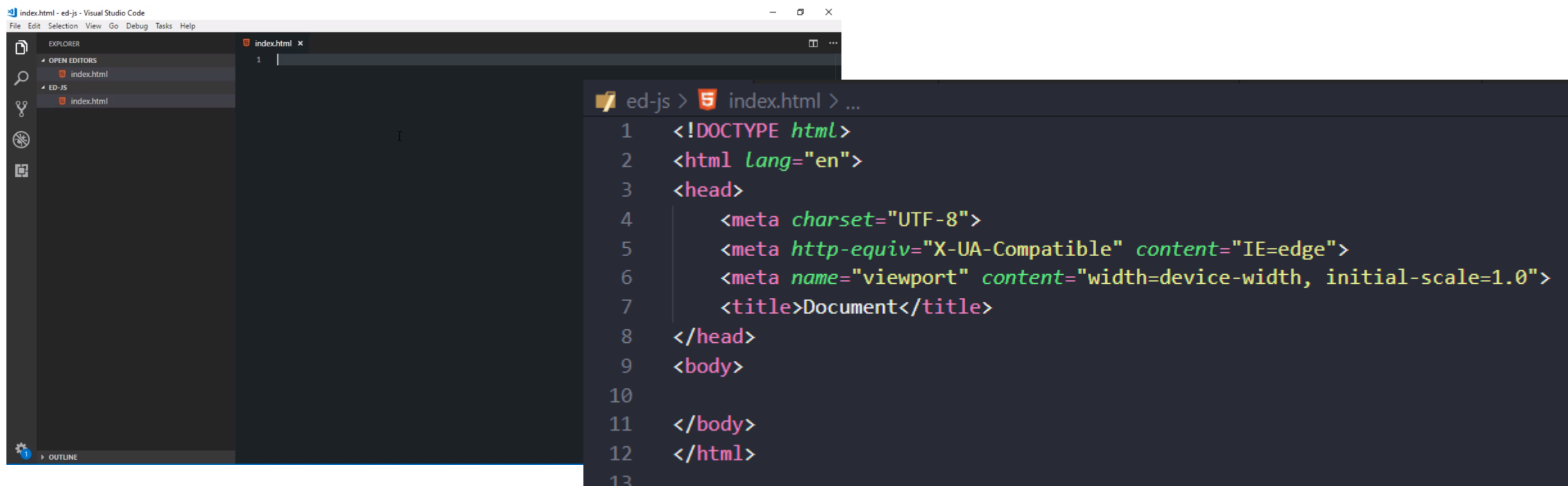
Executando a partir de um arquivo .js

- ✓ Crie uma pasta chamada 'js' e depois a arraste para dentro do VsCode
- ✓ Agora crie um arquivo dentro dela com o nome 01-HelloWorld.js
- ✓ Depois insira neste arquivo o método `alert('Hello World!');`
- ✓ Insira na tag body, uma atalho para o arquivo na tag `<script src="js/01-HelloWorld.js"></script>`
- ✓ Para executar é só abrir em um browser
- ✓ Ou executar o arquivo .html com plugin do servidor local **Live-Server**, acionando com o botão direito.

```
js > JS 01-HelloWorld.js
1
2  alert('Hello, World!');
3
```

```
5 01-HelloWorld.html > ...
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>Document</title>
6  </head>
7  <body>
8      <script src="js/01-HelloWorld.js"></script>
9  </body>
10 </html>
11
```

- Novamente, vamos treinar mais.
- Agora crie um novo arquivo `index.html`
- Se você clicar na primeira linha `<exclamação(!) + tab>` ele vai criar uma estrutura mínima para uma página HTML



```
index.html - ed-js - Visual Studio Code
File Edit Selection View Go Debug Tasks Help

EXPLORER
├─ OPEN EDITORS
│  └─ index.html
├─ ED-JS
│  └─ index.html
└─ OUTLINE

index.html x
1 |
2 |
3 |
4 |
5 |
6 |
7 |
8 |
9 |
10 |
11 |
12 |
13 |

ed-js > index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8  </head>
9  <body>
10
11 </body>
12 </html>
13
```

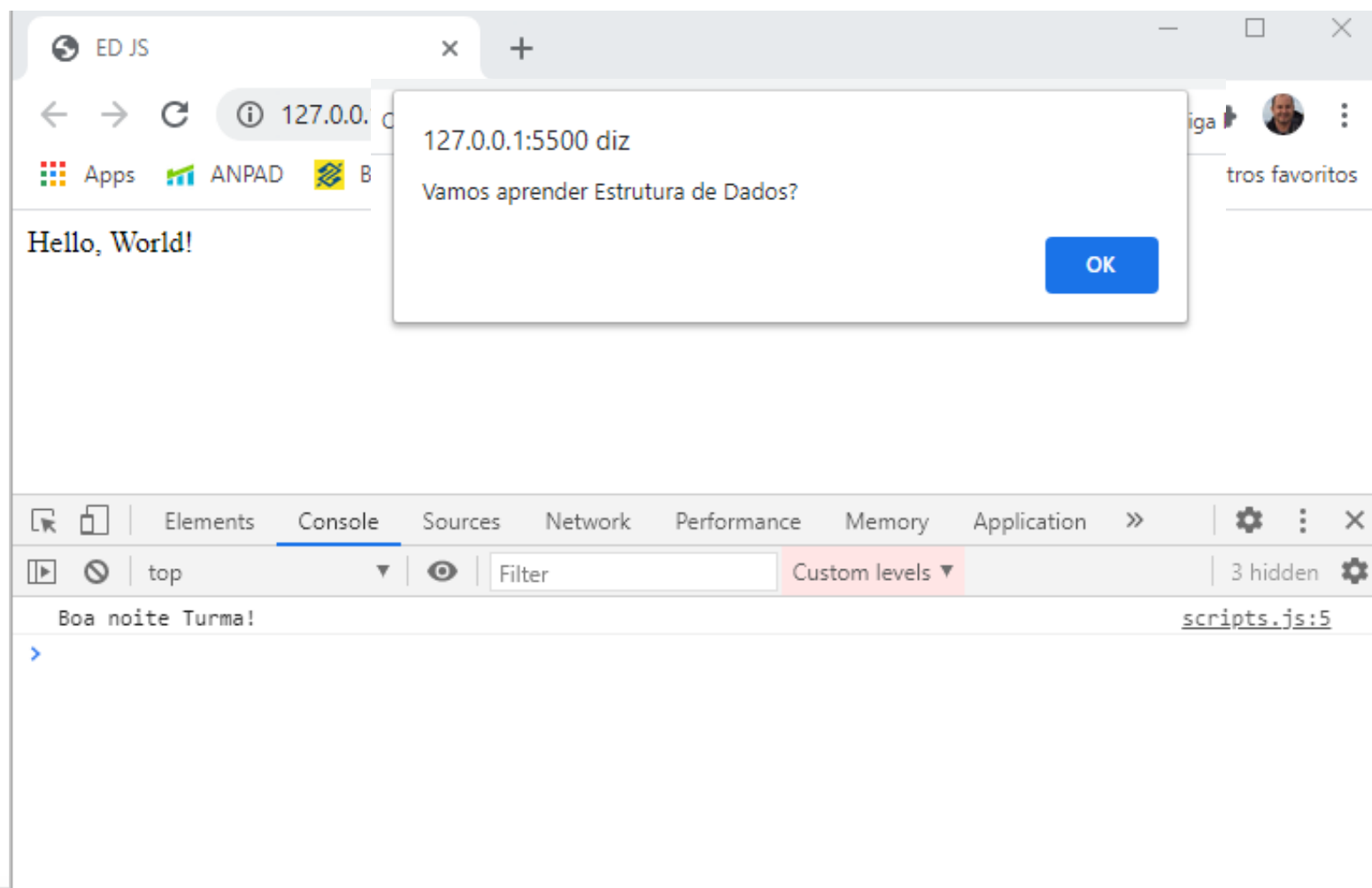

- ✓ Agora, só pra ficar organizado dentro da pasta **js** crie um novo arquivo com o nome **scripts.js** onde iremos codificar alguns comandos JavaScript.
- ✓ Dentro da tag **<body>** insira a seguinte tag: **<script src="js/script.js"></script>**
- ✓ Assim quando você abrir o arquivo **index.html** em um browser, os comandos inseridos no arquivo **script.js** serão executados.

```
index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>ED JS</title>
7  </head>
8  <body>
9      <script src="js/scripts.js"></script></i></b>
10 </body>
11 </html>
```

- Assim quando você abrir o arquivo `index.html` em um browser, os comandos inseridos no arquivo `script.js` serão executados.
- Experimente esses comandos simples em JavaScript:

```
ed-js > js > JS scripts.js
1 // imprime diretamente na página do navegador
2 document.write('Hello, World!');
3
4 // imprimir na console do navegador, na ferramenta do desenvolvedor (F12)
5 console.log('Boa noite Turma!');
6
7 // imprime em uma popup
8 alert('Vamos aprender Estrutura de Dados?');
9
```

- Assim quando você abrir o arquivo `index.html` em um browser, os comandos inseridos no arquivo `script.js` serão executados.



O que são **variáveis**?

Uma variável guarda um dado

Uma variável é utilizada para salvar um dado na memória do seu computador. Na programação precisamos de variáveis para **manipular dados**.

```
js > JS teste.js > ...
1  const nome = 'Regiano Alves';
2  const idade = 52;
3  const sabeJavaScript = true;
4
5  console.log(nome, idade, sabeJavaScript);
6
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\admin\Desktop\ed-js\js> node teste.js
Regiano Alves 52 true
PS C:\Users\admin\Desktop\ed-js\js> 
```

```
js > JS teste.js > ...
1  let nome = 'Regiano Alves';
2  const idade = 52;
3  const sabeJavaScript = true;
4  console.log(nome, idade, sabeJavaScript);
5
6  nome = 'Regiano da Silva Alves';
7  console.log(nome);
8
9
10
11
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\admin\Desktop\ed-js\js> node teste.js
Regiano Alves 52 true
Regiano da Silva Alves
PS C:\Users\admin\Desktop\ed-js\js> 
```

- As variáveis armazenam dados que podem ser **definidos**, **atualizados** e **recuperados** sempre que necessário.
- Os valores atribuídos a uma variável têm um tipo (domínio).
- Em JavaScript, os tipos disponíveis são: **number**, **string**, **boolean**, **function** e **object**.
- Também temos **undefined** e **null**, junto com **arrays**, **datas** e **expressões** regulares.

Groner, 2019, pág. 31

```
js > JS scripts.js > ...  
1  
2 // A seguir, apresentamos um exemplo de uso de variáveis em JavaScript  
3 var num = 1; //{1}  
4 num = 3; //{2}  
5 var price = 1.5 // {3}  
6 var myName = 'Regiano Alves'; //{4}  
7 var trueValue = true; //{5}  
8 var nullVar = null; //{6}  
9 var und; //{7}  
10  
11
```



```
js > JS scripts.js > ...
```

```

1
2 // A seguir, apresentamos um exemplo de uso de variáveis em JavaScript
3 var num = 1; //{1}
4 num = 3; //{2}
5 var price = 1.5 // {3}
6 var myName = 'Regiano Alves'; //{4}
7 var trueValue = true; //{5}
8 var nullVar = null; //{6}
9 var und; //{7}
10
11 /**
12  * {1} Na linha 3, temos um exemplo de como declarar uma variável numérica em JavaScript
13  * {2} Na linha 4, atualizamos uma variável existente. A linguagem JavaScript não é
14  *       fortemente tipada. Isso significa que podemos declarar uma variável, inicializá-la
15  *       com um número e depois atualizá-la com uma string ou com qualquer outro tipo de dado.
16  * {3} Na linha 5, declaramos igualmente um número, porém, dessa vez, é um número decimal
17  *       de ponto flutuante.
18  * {4} Na linha 6, declaramos uma string.
19  * {5} Na linha 7, declaramos um booleano.
20  * {6} Na linha 8, declaramos um valor null e,
21  * {7} Na linha 9, declaramos uma variável undefined (indefinido)
22  * Um valor null quer dizer sem valor, e undefined significa uma variável que foi declarada,
23  * mas que ainda não recebeu nenhum valor.
24  */
25

```

- JavaScript não é fortemente tipada como C/C++, C# e Java.
- Em linguagens fortemente tipadas, devemos definir o tipo da variável em sua declaração (**em Java, por exemplo: `int num = 1;`**)
- Em JavaScript basta **`var num = 1;`** (por isso ela é fracamente tipada)
- Fracamente tipada, significa tipada dinamicamente.

```
27  /**
28   * Se quisermos ver o valor de cada variável que declaramos,
29   * podemos usar console.log para isso, conforme listado
30   * no trecho de código a seguir:
31   *
32   */
33  console.log('num: ' + num);
34  console.log('Meu nome: ' + myName)
35  console.log('trueValue: ' + trueValue);
36  console.log('price: ' + price);
37  console.log('nullVar: ' + nullVar);
38  console.log('und: ' + und);
39
```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

[Running] node "c:\Users\admin\Desktop\ed-js\js\scripts.js"

num: 3
Meu nome: Regiano Alves
trueValue: true
price: 1.5
nullVar: null
und: undefined

```
js > JS scripts.js > ...
```

```
1
2 // A seguir, apresentamos um exemplo de uso de variáveis.
3 var num = 1; //{1}
4 num = 3; //{2}
5 var price = 1.5 // {3}
6 var myName = 'Regiano Alves'; //{4}
7 var trueValue = true; //{5}
8 var nullVar = null; //{6}
9 var und; //{7}
10
11 /**
12  * {1} Na linha 3, temos um exemplo de como declarar um
13  * {2} Na linha 4, atualizamos uma variável existente.
14  *       fortemente tipada. Isso significa que podemos declarar
15  *       com um número e depois atualizá-la com uma string.
16  * {3} Na linha 5, declaramos igualmente um número, porém com
17  *       de ponto flutuante.
18  * {4} Na linha 6, declaramos uma string.
19  * {5} Na linha 7, declaramos um booleano.
20  * {6} Na linha 8, declaramos um valor null e,
21  * {7} Na linha 9, declaramos uma variável undefined (que não
22  *       Um valor null quer dizer sem valor, e undefined significa
23  *       mas que ainda não recebeu nenhum valor.
24  */
```

```
26
27 /**
28  * Se quisermos ver o valor de cada variável que declaramos,
29  * podemos usar console.log para isso, conforme listado
30  * no trecho de código a seguir:
31  *
32  */
33 console.log('num: ' + num);
34 console.log('Meu nome: ' + myName)
35 console.log('trueValue: ' + trueValue);
36 console.log('price: ' + price);
37 console.log('nullVar: ' + nullVar);
38 console.log('und: ' + und);
39
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\admin\Desktop\ed-js\js> node scripts.js
num: 3
Meu nome: Regiano Alves
trueValue: true
price: 1.5
nullVar: null
und: undefined
PS C:\Users\admin\Desktop\ed-js\js> 
```

- O **escopo** se refere ao local em que podemos acessar a variável no algoritmo.
- Também pode ser uma função quando trabalhamos com **escopos de função**.
- As variáveis podem ser **locais** ou **globais**.
- Procure evitar as variáveis globais.

```
ed-js > js > JS escopoVariaveis.js > ...
1  //vamos observar um exemplo:
2  var myVariable = 'global';
3  myOtherVariable = 'global';
4
5  function myFunction(){
6      var myVariable = 'local';
7      return myVariable;
8  }
9
10 function myOtherFunction(){
11     myOtherVariable = 'local';
12     return myOtherVariable;
13 }
14
```

ed-js > js > JS escopoVariaveis.js > ...

```
1 //vamos observar um exemplo:
2 var myVariable = 'global';
3 myOtherVariable = 'global';
4
5 function myFunction(){
6     var myVariable = 'local';
7     return myVariable;
8 }
9
10 function myOtherFunction(){
11     myOtherVariable = 'local';
12     return myOtherVariable;
13 }
14
```

```
14
15 console.log(myVariable) // {1}
16 console.log(myFunction()); //{2}
17 console.log(myOtherFunction()); //{4}
18 console.log(myOtherVariable); // {5}
19 /**
20  * - A linha {1} exibirá global porque estamos referenciando uma variável global.
21  * - A linha {2} exibirá local porque declaramos a variável myVariable dentro da
22  *   função myFunction como uma variável local, portanto o escopo está apenas no
23  *   interior de myFunction
24  * - A linha {3} exibirá global porque estamos referenciando a variável global
25  *   chamada myOtherVariable, inicializada na segunda linha do exemplo.
26  * - A linha {4} exibirá local. Na função myOtherFunction, referenciamos a variável
27  *   global myOtherVariable e lhe atribuímos o valor local, pois não declaramos a
28  *   variável usando a palavra reservada var.
29  * - por esse motivo, a linha {5} exibirá local (pois alteramos o valor da variável
30  *   em myOtherFunction)
31  */
```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

[Running] node "c:\Users\admin\Desktop\ed-js\js\escopoVariaveis.js"

global

local

local

local

A linguagem JavaScript também tem operadores:

- aritméticos
- de atribuição
- de comparação
- lógicos
- bit a bit (**bitwise**), e
- unários (entre outros)

```
ed-js > js > JS operadores.js > ...
1 // Vamos observar esses operadores
2 var num = 0; // {1}
3 // operadores aritméticos
4 num = num + 2;
5 num = num * 3;
6 num = num / 2;
7 // operadores de atribuição
8 num++;
9 num--;
10 num += 1; //{2}
11 num -= 2;
12 num *= 3;
13 num /= 2;
14 num % 3;
15 // Visualizando no console os valores de cada operação
16 console.log('num: ' + num);
17 // operadores de comparação
18 console.log('num == 1: ' + (num == 1)); // {3}
19 console.log('num === 1: ' + (num === 1));
20 console.log('num != 1: ' + (num != 1));
21 console.log('num > 1: ' + (num > 1));
22 console.log('num < 1: ' + (num < 1));
23 console.log('num >= 1: ' + (num >= 1));
24 console.log('num <= 1: ' + (num <= 1));
25 console.log('true && false: ' + (true && false)); // {4}
26 console.log('true || false: ' + (true || false));
27 console.log('!true: ' + (!true));
28
```


Operadores aritméticos

Operador aritmético	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto de uma divisão)
++	Incremento
--	Decremento

```

ed-js > js > JS operadores.js > ...
15 // Visualizando no console os valores de cada operação
16 console.log('num: ' + num);
17 // operadores de comparação
18 console.log('num == 1: ' + (num == 1)); // {3}
19 console.log('num === 1: ' + (num === 1));
20 console.log('num != 1: ' + (num != 1));
21 console.log('num > 1: ' + (num > 1));
22 console.log('num < 1: ' + (num < 1));
23 console.log('num >= 1: ' + (num >= 1));
24 console.log('num <= 1: ' + (num <= 1));
25 console.log('true && false: ' + (true && false)); // {4}
26 console.log('true || false: ' + (true || false));
27 console.log('!true: ' + (!true));
28

```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

```

[Running] node "c:\Users\admin\Desktop\ed-js\js\operadores.js"
num: 3
num == 1: false
num === 1: false
num != 1: true
num > 1: true
num < 1: false
num >= 1: true
num <= 1: false
true && false: false
true || false: true
!true: false

```

Operador de atribuição	Descrição
=	Atribuição
+=	Atribuição de soma $(x += y) == (x = x + y)$
-=	Atribuição de subtração $(x -= y) == (x = x - y)$
*=	Atribuição de multiplicação $(x *= y) == (x = x * y)$
/=	Atribuição de divisão $(x /= y) ==$
%=	Atribuição de resto $(x %= y) == (x = x \% y)$

Operador lógico	Descrição
&&	E
	Ou
!	Negação

Operador de comparação	Descrição
==	Igual a
===	Igual a (tanto o valor quanto o tipo do objeto)
!=	Diferente de
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a

Operador bit a bit (bitwise)	Descrição
&	E
	Ou
~	Negação

Operador bit a bit (bitwise)	Descrição
&	E
	Ou
~	Negação

Operador bit a bit (bitwise)	Descrição
^	Ou exclusivo (Xor)
<<	Deslocamento para a esquerda (left shift)
>>	Deslocamento para a direita (right shift)

```

29 // operadores bit a bit (bitwise)
30 console.log('5 & 1: ', (5 & 1));
31 console.log('5 | 1: ', (5 | 1));
32 console.log('~5: ', (~5));
33 console.log('5 ^ 1:', (5 ^ 1));
34 console.log('5 << 1:', (5 << 1));
35 console.log('5 >> 1:', (5 >> 1));

```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

```

[Running] node "c:\Users\admin\Desktop\ed-
5 & 1: 1
5 | 1: 5
~5: -6
5 ^ 1: 4
5 << 1: 10
5 >> 1: 2

```

De acordo com a especificação, há **dois** tipos de dados em JavaScript:

- **Tipos de dados primitivos:** **null** (nulo), **undefined** (indefinido), **string**, **number** (número), **boolean** (booleano) e **symbol** (símbolo);
- **Tipos de dados derivados/objetos:** **objetos** JavaScript, incluindo **funções**, **arrays** e **expressões regulares**.
- O operador **typeof** devolve o tipo da variável

```
37 // O operador typeof devolve o tipo da variável ou expressão
38 var num = 0;
39 console.log('typeof num: ', typeof num);
40 console.log('typeof Texto String: ', typeof 'Texto String');
41 console.log('typeof true: ', typeof true);
42 console.log('typeof [1,2,3]: ', typeof [1,2,3]);
43 console.log('typeof {name: Pedro}', typeof {name: 'Pedro'});
```

PROBLEMS 2

OUTPUT

TERMINAL

DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
typeof num: number
typeof Texto String: string
typeof true: boolean
typeof [1,2,3]: object
typeof {name: Pedro} object
```

- Em JavaScript, **true** e **false** são um pouco complicados.
- Na maioria das linguagens os valores booleanos **true** e **false** representam os resultados verdadeiro/falso.
- Em JavaScript, uma string “Hello World” é avaliada como **true**.

Tipo do valor	Resultado
Undefined	false
Null	false
Boolean	Verdadeiro é true e falso é false
Number	O resultado é false para +0, -0 ou NaN; caso contrário, é true
String	O resultado é false se a string for vazia (o tamanho é 0); caso contrário, é true (tamanho ≥ 1)
Object	true

- Em JavaScript, **true** e **false** são um pouco complicados.

- Na maioria das vezes, o valor verdadeiro é considerado verdadeiro.

- Em JavaScript, o valor verdadeiro é considerado verdadeiro.

```
47 // Vamos considerar alguns exemplos de Verdadeiro/Falso e observar suas saídas:
48 function testTruthy(val){
49     return val ? console.log('true') : console.log('false');
50 }
51 testTruthy(true); // true
52 testTruthy(false); // false
53 testTruthy(new Boolean(false)); // true (objeto é sempre true)
54 testTruthy(''); // false
55 testTruthy('Bom dia!'); // true
56 testTruthy(new String('')); // true (objeto é sempre true)
57 testTruthy(1); // true
58 testTruthy(-1); // true
59 //A propriedade global NaN é um valor especial que significa Not-A-Number (não é um número)
60 testTruthy(NaN); // false
61 testTruthy(new Number(NaN)); // true (objeto é sempre true)
62 var obj = { name: 'João' };
63 testTruthy(obj); // true
64 testTruthy(obj.name); // true
65 testTruthy(obj.age); // age (propriedade não existe)
66
```

Operadores de igualdade (== e ===)

- Os dois operadores de **igualdade** aceitos em JavaScript podem causar um pouco de confusão.
- Quando **==** é usado, os valores poderão ser considerados iguais (mesmo se forem de tipos diferentes).
- Se **x** e **y** forem do mesmo tipo, então JavaScript usará o método **equals** para comparar os dois valores ou objetos.

Type(x)	Type(y)	Resultado
null	undefined	True
undefined	null	true
Number	String	X == toNumber(y)
String	Number	toNumber(x) == y
Boolean	Any	toNumber(x) == y
Any	Boolean	X == toNumber(y)
String ou Number	Object	X == toPrimitive(y)
Object	String ou Number	toPrimitive(x) == y

Operadores de igualdade (== e ===)

- Quando === é usado, se estivermos comparando dois valores de tipos diferentes, o resultado será sempre **false**.
- Se forem do mesmo tipo, eles serão comparados de acordo com a seguinte tabela:

Type(x)	Valores	Resultado
Number	x tem o mesmo valor que y (mas não é NaN)	true
String	X e y têm caracteres idênticos	true
Boolean	x e y são ambos true ou são ambos false	true
Object	x e y referenciam o mesmo objeto	true

- Se x e y forem de tipos diferentes, o resultado será **false**. Veja os exemplos:

```
68 // Operador de igualdade (===)
69 console.log('Joaquim' === true); // false
70 console.log('Maria' === 'Maria'); // true
71 var person1 = {name: 'Pedro'};
72 var person2 = {name: 'Pedro'};
73 console.log(person1 === person2); // false, objetos diferentes
74
```

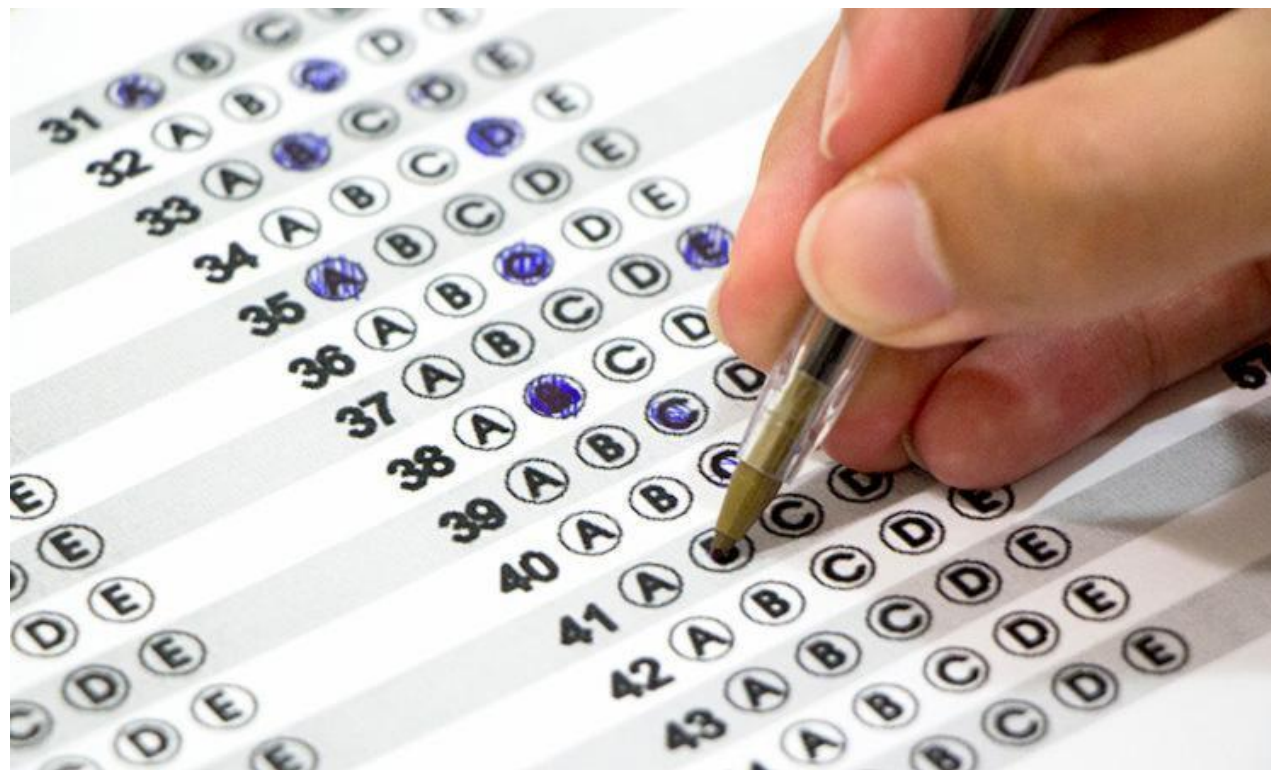
PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
false
true
false
```


Ano: 2019 Banca: VUNESP Órgão: Câmara de Piracicaba - SP Prova: SP – Programador (TI)

Na linguagem JavaScript, o operador === (três sinais de igualdade) realiza a comparação apenas do:

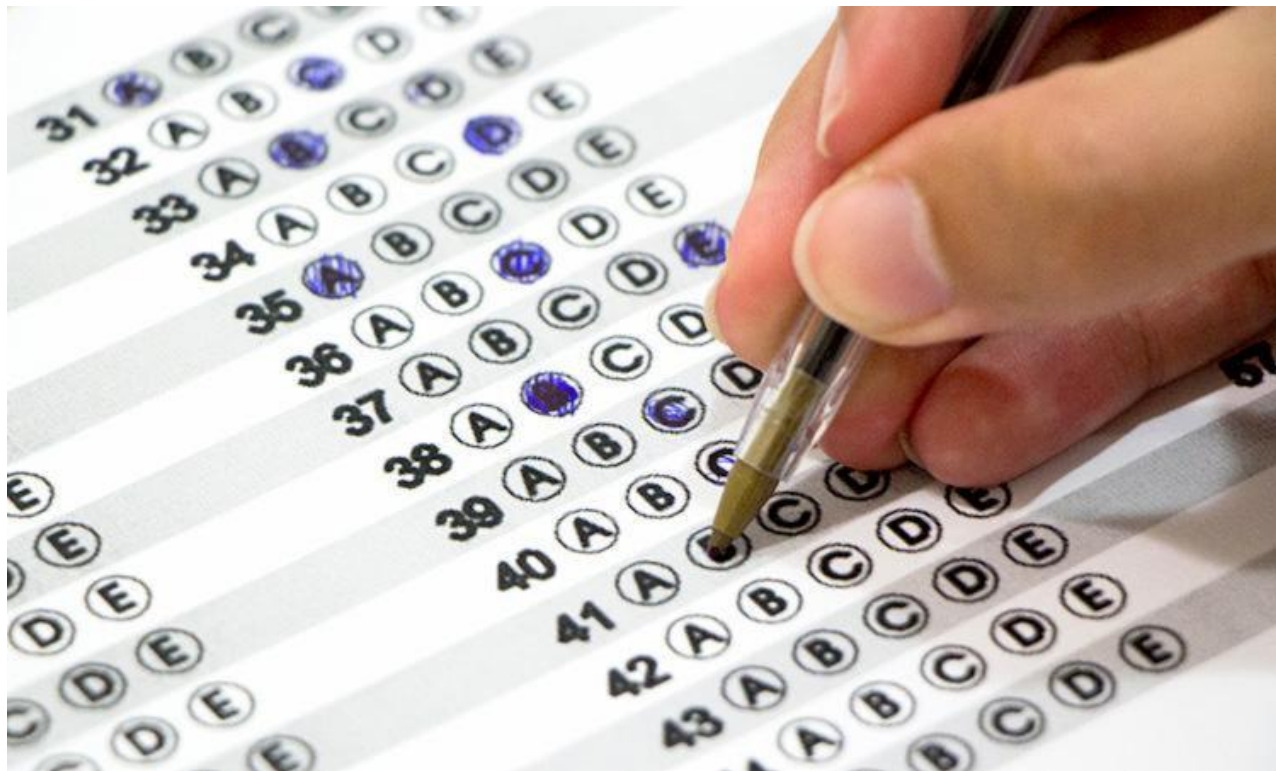
- a) tipo dos operandos.
- b) conteúdo dos operandos.
- c) valor dos operandos.
- d) valor lógico dos operandos.
- e) valor e do tipo dos operandos.



Ano: 2019 Banca: VUNESP Órgão: Câmara de Piracicaba - SP Prova: SP – Programador (TI)

Na linguagem JavaScript, o operador === (três sinais de igualdade) realiza a comparação apenas do:

- a) tipo dos operandos.
- b) conteúdo dos operandos.
- c) valor dos operandos.
- d) valor lógico dos operandos.
- ➡ e) **valor e do tipo dos operandos.**



Ano: 2013 Banca: VUNESP Órgão: MPE-ES Prova: VUNESP - 2013 - MPE-ES - Agente Técnico – Webdesigner

Em páginas HTML, as rotinas JavaScript são delimitadas pelos *tags*:

- A. <html> e </html>
- B. <script> e </script>
- C. <body> e </body>
- D. <head> e </head>
- E. <p> e </p>



Ano: 2013 Banca: VUNESP Órgão: MPE-ES Prova: VUNESP - 2013 - MPE-ES - Agente Técnico – Webdesigner

Em páginas HTML, as rotinas JavaScript são delimitadas pelos *tags*:

- A. <html> e </html>
- ➔ B. <script> e </script>
- C. <body> e </body>
- D. <head> e </head>
- E. <p> e </p>



Ano: 2020 Banca: COMPERVE Órgão: TJ-RN Prova: Administrador de Sites (Web Master) 2020

No javascript é possível interagir com o console dos navegadores. O comando para imprimir o texto 'TJ-RN' no console é:

- A. `console.dump('TJ-RN');`
- B. `console.print('TJ-RN');`
- C. `console.log('TJ-RN');`
- D. `console.echo('TJ-RN');`



Ano: 2020 Banca: COMPERVE Órgão: TJ-RN Prova: Administrador de Sites (Web Master) 2020

No javascript é possível interagir com o console dos navegadores. O comando para imprimir o texto 'TJ-RN' no console é:

- A. `console.dump('TJ-RN');`
- B. `console.print('TJ-RN');`
- ➔ C. `console.log('TJ-RN');`
- D. `console.echo('TJ-RN');`



Ano: 2018 Banca: CESPE Órgão: FUB Prova: Técnico de Tecnologia da Informação

A fim de melhor organizar um código em JavaScript e facilitar a sua manutenção, é possível utilizar módulos que estejam implementados em arquivos distintos do arquivo onde está o código que o invoca; nesse caso, é necessário que o módulo seja explicitamente declarado como passível de exportação no seu arquivo de origem:

- A. Errado
- B. Certo;



Ano: 2018 Banca: CESPE Órgão: FUB Prova: Técnico de Tecnologia da Informação

A fim de melhor organizar um código em JavaScript e facilitar a sua manutenção, é possível utilizar módulos que estejam implementados em arquivos distintos do arquivo onde está o código que o invoca; nesse caso, é necessário que o módulo seja explicitamente declarado como passível de exportação no seu arquivo de origem:

A. Errado

➡ B. Certo



