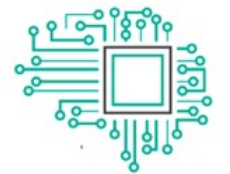


JAVASCRIPT

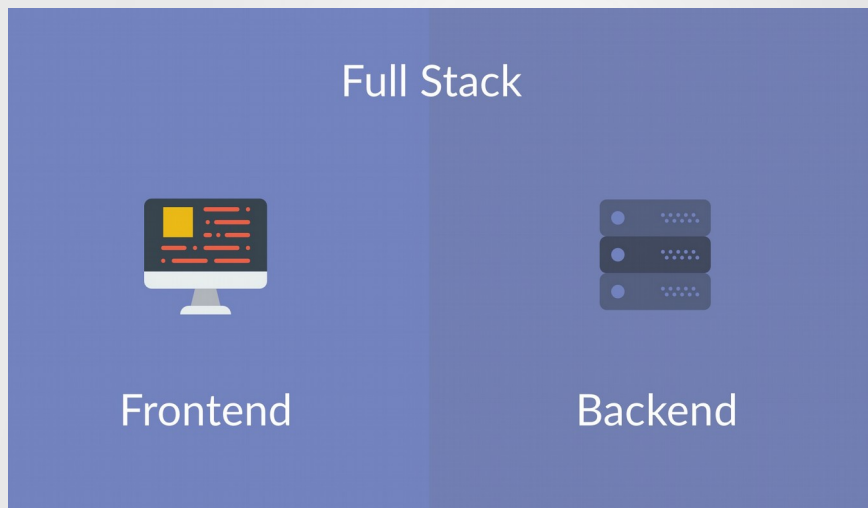
Pedro Moritz de Carvalho Neto



JCAVI

Observações

- Os conceitos deste curso serão abordados de maneira não-linear.
- JavaScript pode ser executado tanto no frontend quando no backend.




Observações

- Você pode utilizar qualquer IDE (Integrated Development Environment) durante o curso. Exemplos:
 - <https://www.sublimetext.com/>
 - <https://code.visualstudio.com/>
 - <https://atom.io/>
 - <https://www.jetbrains.com/webstorm/>
 - <https://codepen.io/pen>
 - <https://jsfiddle.net/>



Conteúdo Programático

- **Introdução ao JavaScript**
 - **Variáveis**
 - **Escopo**
 - **Funções**
 - **Closures**
 - **Tipos**
 - **Tipagem dinâmica**
 - **Operador typeof**
 - **Operadores aritméticos, lógicos e de comparação**
 - **Operador ternário**
 - **Arrays**
 - **Objetos**
 - **Eventos**
 - **Temporizadores**
 - **Ajax**
 - **DOM**
 - **Prototype**
- 



Introdução ao JavaScript

Introdução ao JavaScript

- Nasceu em 1995 com o nome de “Mocha” no Netscape Navigator.
- Em seguida, a linguagem foi renomeada para “LiveScript” e logo depois para “JavaScript”.
- O JavaScript não tem relação nenhuma com o Java, mas este último era uma linguagem extremamente promissora na época.



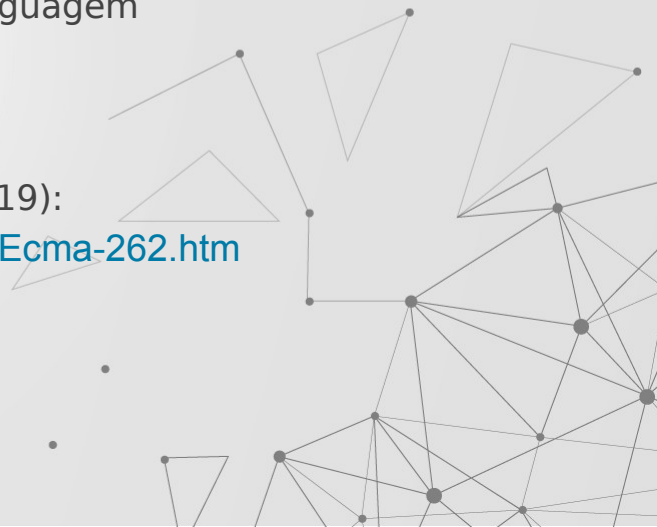
Introdução ao JavaScript

- A Microsoft fez uma engenharia reversa do JavaScript e criou sua própria implementação, denominada “JScript”, lançada em 1996.
- O JScript se comportava de maneira muito diferente do JavaScript, causando inúmeros problemas de compatibilidade.



Introdução ao JavaScript

- Em 1996 a Netscape submeteu o JavaScript ao ECMA (European Computer Manufacturers Association) para que fosse criado um padrão a ser seguido por todos os fabricantes de browsers.
- Em 1997 foi criado o padrão ECMA-262, que define a linguagem denominada “ECMAScript”.
- Sua atual versão é o ECMAScript 2019 (ou apenas ES2019):
<https://www.ecma-international.org/publications/standards/Ecma-262.htm>



Introdução ao JavaScript

- Algumas linguagens que seguem o padrão ECMAScript:
 - JavaScript
 - Jscript
 - ActionScript
 - QtScript
 - Google Apps Script



Introdução ao JavaScript

- Cada browser possui seu próprio interpretador de ECMAScript:
 - Chrome: v8
 - Firefox: SpiderMonkey
 - Safari: JavaScriptCore
 - IE e Edge: Chakra
- Existem implementações de interpretador de ECMAScript para o backend:
 - Node.js



Introdução ao JavaScript

- Existem muitas bibliotecas e plataformas que utilizam o ECMAScript:
 - jQuery
 - Bootstrap
 - AngularJS
 - React.js
 - Vue.js
 - Ember.js
 - ...



Introdução ao JavaScript

- Muitos produtos e tecnologias utilizam internamente a linguagem JavaScript:
 - MongoDB
 - Apache CouchDB
 - JSON
 - ...



Variáveis



Variáveis

- Variáveis são nomes simbólicos que referenciam valores
- Para se declarar uma variável utilizamos o comando **let**
- Para se atribuir um valor à uma variável utilizamos o operador **=**
- Variáveis no JavaScript podem ou não serem declaradas para serem usadas



Variáveis

- Regras para criação de nomes de variáveis no JavaScript:
 - Devem começar com uma letra, sublinhado (`_`), ou cifrão (`$`)
 - Os caracteres subsequentes podem também ser dígitos (0-9)
 - As letras podem ser minúsculas ou maiúsculas
 - O nome das variáveis, assim como em todos os elementos no JavaScript, são “case-sensitive”

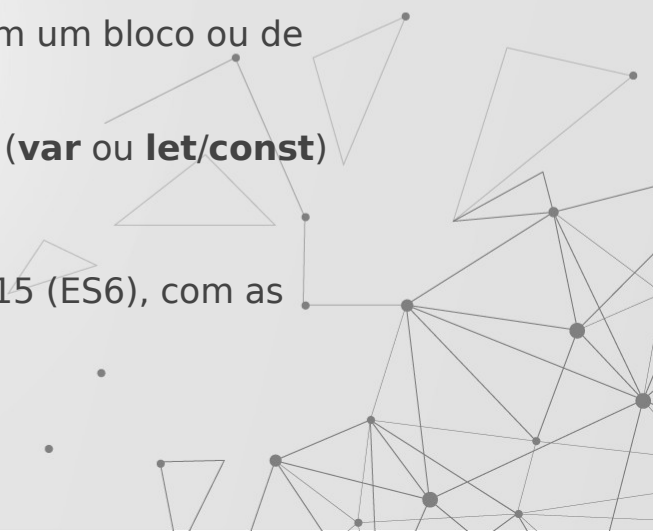


Escopo



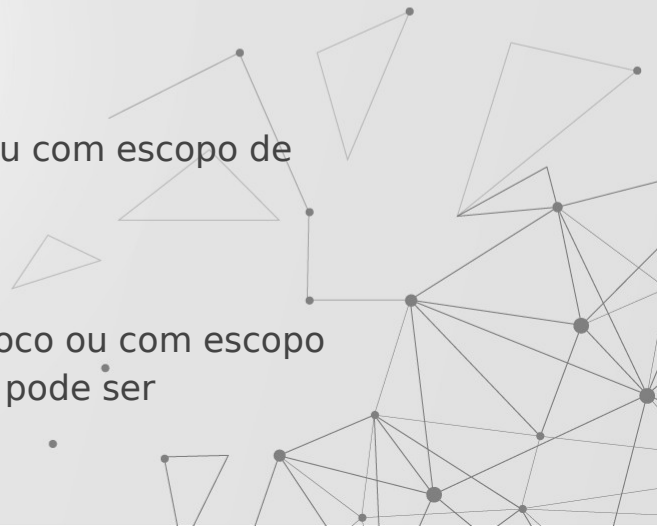
Escopo

- O escopo de uma variável é o espaço a partir do qual esta variável pode ser acessada
- O escopo de uma variável é definido por dois fatores:
 - pelo local onde ela é declarada (em uma função, em um bloco ou de maneira global)
 - pela instrução utilizada para declarar esta variável (**var** ou **let/const**)
- O conceito de escopo de bloco surgiu na ECMAScript 2015 (ES6), com as instruções **let** e **const**



Escopo

- Nas declarações de variável em nível de função, o escopo fica restrito à função
- A instrução **var** declara variáveis com escopo global ou com escopo de função
- A instrução **let** declara variáveis com escopo de bloco ou com escopo de função
- A instrução **const** declara constantes com escopo de bloco ou com escopo de função (não podem ser redeclaradas e seu valor não pode ser modificado)



Escopo

| keyword | const | let | var |
|-------------------|-------|-----|-----|
| global scope | NO | NO | YES |
| function scope | YES | YES | YES |
| block scope | YES | YES | NO |
| can be reassigned | NO | YES | YES |

<http://www.constletvar.com>

Funções



Funções

- Uma função é um bloco de código que realiza uma tarefa específica
- É semelhante ao conceito de **procedure** ou **subrotina**, existente em outras linguagens de programação
- Funções sempre retornam um valor
- Se a instrução de retorno (**return**) não for utilizada dentro da função, ela retornará **undefined**



Funções

- Valores podem ser passados para uma função como parâmetros ou usados dentro da função se estiverem dentro do seu escopo
- Uma vez que a função executa o seu código de retorno, mais nenhum código declarado dentro da função é executado
- A função envia o resultado do seu retorno para a instrução que a invocou



Funções

- Funções possuem as seguintes características (antes do ECMAScript 2015):
 - É definida pela instrução **function**
 - Seu nome segue as mesmas regras de nomes de variáveis
 - Pode possuir parâmetros (listados dentro de parênteses)
 - O código a ser executado é declarado dentro de chaves (obrigatório)

```
function teste(parameter1, parameter2, parameter3) {  
    /* */  
}
```



Funções

- Uma função é executada quando é invocada dentro do algoritmo
- Podem ser invocadas imediatamente após sua criação (*Immediately-invoked Function Expression* ou **IIFE**):

```
(function() {  
    /**/  
})();
```

- Uma IIFE pode possuir um nome ou não:

```
(function teste() {  
    /**/  
})();
```



Funções

- Declaração de função:

O código a seguir é uma declaração de função. Esta função existe a partir do momento em que o script é carregado no browser.

```
function teste(parameter1, parameter2, parameter3) {  
    /**/  
}
```



Funções

- Expressão de função:

O código a seguir é uma expressão de função. Esta função só existe quando a linha onde está a expressão é executada.

```
let teste = function(parameter1, parameter2, parameter3) {  
  /**/  
}
```



Funções

- Funções declaradas podem ser executadas antes da declaração. Isto é possível devido ao mecanismo denominado *hoisting*
- Este mecanismo coloca as declarações de variáveis e funções no topo do código antes de sua execução, conforme o exemplo:

```
let output = teste(2, 4);  
console.log(output);
```

```
function teste(x, y) {  
  return x * y;  
}
```



Funções

- ECMAScript 2015 (ES6) também introduziu uma nova forma de criar funções: Arrow functions.

```
function teste(x, y) {  
  return x * y;  
}
```

```
let teste = function(x, y) {  
  return x * y;  
}
```

```
let teste = (x, y) => {  
  return x * y;  
}
```



Funções

- Se a arrow function não possui parametros, pode ser representada da seguinte forma:

```
let teste = () => {  
  return "Olá";  
}
```



Funções

- Se a arrow function possui apenas um parâmetro, pode ser representada assim:

```
let teste = x => {  
  return x * 2;  
}
```



Funções

- Se a arrow function possui apenas uma declaração, e esta retorna um valor, pode ser representada como no seguinte exemplo:

```
let teste = () => "Olá";
```

