PROGRAMAÇÃO WEB II

IFCE Campus Cedro - Sistemas de Informação

PROF. ZÉ OLINDA

JAVASCRIPT

Fundamentos da Linguagem

Parte 01

CONTEÚDO

1. Estrutura do Código

A primeira coisa que estudaremos são os blocos de construção do código.

O inferno é não entender meu próprio código.

Kyle Simpson

INSTRUÇÕES

- As instruções são construções de sintáticas e comandos que executam ações.
- Podemos ter quantas instruções em nosso código quisermos. As instruções **podem** ser separadas por ponto e vírgula.
- Normalmente, as instruções são escritas em linhas separadas para tornar o código mais legível.

PONTO-E-VÍRGULA

- Recomenda-se o uso do ponto-e-vírgula para encerrar cada instrução.
- Pode ser omitido **na maioria dos casos** quando existe uma quebra de linha.
- Existem casos em que uma nova linha não significa um ponto e vírgula.

PONTO-E-VÍRGULA

```
// Funciona
// Adiciona automáticamente um ponto-e-vírgula ao final da linha
alert('Hello')
alert('World')

// Funciona
// Não adiciona ponto-e-vírgula ao final da linha
// Entende que a instrução só termina após o parênteses
alert(3 +
1
+ 2)
```

PONTO-E-VÍRGULA

```
// Não Funciona
// Não identifica o final da instrução após o parênteses
// Une a linha seguinte e gera um erro de sintaxe
alert("Hello")

[1, 2].forEach(alert);
```

PONTO-E-VÍRGULA

```
// Funciona
alert("Hello");

[1, 2].forEach(alert);
```

COMENTÁRIOS

- Conforme o tempo passa, os programas tornam-se cada vez mais complexos. Torna-se necessário adicionar comentários que descrevam o que o código faz e por quê.
- Os comentários podem ser colocados em qualquer lugar de um script. Eles não afetam sua execução porque o javascript simplesmente os ignora.

COMENTÁRIOS

- Os comentários de uma linha começam com dois caracteres de barra //.
- Após // o resto da linha é um comentário. Pode ocupar uma linha inteira ou a partir de um determinado ponto da instrução.
- Os comentários de várias linhas começam com uma barra e um asterisco /* e terminam com um asterisco e uma barra */.
- Todo conteúdo entre /* e */ será completamente ignorado.
- **Comentários aninhados não são suportados! Pode não haver /*...*/ dentro de outro /*...*/**

2. Modo moderno: ‘*use strict*’

Modificações tragas pela especificação ECMAScript 5 trazem modernidade a linguagem

Por muito tempo, o JavaScript evoluiu sem problemas de compatibilidade. Novos recursos foram adicionados a linguagem, enquanto as funcionalidades anteriores não mudaram.

‘USE STRICT’

- O javascript mantém compatibilidade com as versões anteriores. Isso teve a vantagem de nunca quebrar o código existente. Mas a desvantagem era que qualquer erro ou decisão imperfeita dos criadores do JavaScript ficava presa na linguagem para sempre.
- O ECMAScript 5 (ES5) adicionou novos recursos a linguagem e modificou alguns dos existentes.
- Para manter o código antigo funcionando, a maioria dessas modificações está desativada por padrão.
- Você precisa ativá-los explicitamente com uma directiva especial: "**use strict**"

‘USE STRICT’

- A diretiva se parece com uma string: "**use strict**"ou '**use strict**'.
- Quando está localizado no topo de um script, todo o script funciona da maneira “moderna”.
- ~~"use strict" pode ser colocada no início de uma função. Isso ativa o modo estrito apenas nessa função.~~ (não recomendo)
- Certifique-se de que "use strict"está na parte superior de seus scripts, **caso contrário, o modo estrito pode não ser habilitado**.
- Ao estudar a linguagem, veremos as diferenças do modo padrão e o modo estrito. Todas as diferenças estão listadas **aqui**.

3. Variáveis

As variáveis permite armazenar, alterar e processar informações.

VARIÁVEIS

- Uma variável é um “armazenamento nomeado” para dados.
- Podemos usar variáveis para armazenar texto, número, total de visitantes e outros dados.
- Para criar uma variável em JavaScript, use a palavra-chave **let**

```
let messagem;
messagem = 'Oi!';

alert(messagem); // Mostra a variável
```

VARIÁVEIS

- Também podemos declarar várias variáveis em uma linha.
- Pode parecer mais curto, mas não o recomendamos. Para uma melhor legibilidade, use uma única linha por variável.

```
// Linha única. Não recomendado
let user = 'John', age = 25, message = 'Hello';

// Um variável por linha. Forma ideal
let user = 'John';
let age = 25;
let message = 'Hello';
```

CHAMA O VAR!!!

- Em scripts mais antigos, você também pode encontrar outra palavra-chave: **var** em vez de **let**.
- var é quase o mesmo que let. Também declara uma variável, mas de uma forma um pouco diferente, “à moda antiga”.
- A palavra var é semelhante a let. Na maioria das vezes, podemos substituir let por var ou vice-versa e esperar que as coisas funcionem.
- Var não tem escopo de bloco.
- Var aceita redeclaração.

var vs let

```
if (true) {
  var test = true; // A variável test existirá fora do IF
}

alert(test); // true
```

```
if (true) {
  let test = true; // test só existe dentro do bloco if
}

alert(test); // ReferenceError: test is not defined
```

var vs let

```
let user;
let user; // SyntaxError: 'user' has already been declared
```

```
var user = "Peter";

var user = "John"; // substitui a variável anterior

alert(user); // John
```

CONSTANTES

- Como o nome sugere, constante não podem ter seu valor alterado, nem redeclarado.
- Sem que um dado armazenado em uma variável não for passível de alteração, deve ser armazenado como constante.
- Cria-se constante usando a palavra reservada **const**.
- Sempre que possível, use const.

CONSTANTES

```
const myBirthday = '18.04.1982';

myBirthday = '01.01.2001';
// erro, não é posspivel mudar o valor de uma constante
```

CONSTANTES

- Uma prática comum é definir constante usando letras maísculas. Contudo, costumamos usar constantes para armazenar referências/ponteiros para objetos e funções. Neste casos, não é necessário usar maiúsculas.

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

REGRAS DE NOMES

- O nome deve conter apenas letras, dígitos ou os símbolos $ e _.
- O primeiro caractere não deve ser um número.
- JS é case-sensitive.
- Não pode usar palavras reservadas da linguagem. Confira a lista **aqui**.

4. Hoisting

Basicamente, todas as suas variáveis são içadas/elevadas para o topo do escopo (global, função ou bloco)

HOISTING

Hoist em inglês significa levantar ou suspender algo através de um aparato mecânico. Em bom português, significa usar o guindaste para elevar um objeto.

E é isto o que acontece em JavaScript quando declaramos uma variável ou função. Sua ***declaração*** *é "elevada" para o topo do escopo.*

```
//Exemplo 1 - Não eleva (hoist)
var x = 1; // Inicializa x
console.log(x + " " + y); // '1 undefined'
var y = 2; // Initialize y
//Isso não funcionará, pois o JavaScript apenas eleva declarações

//Example 2 - Hoists
var num1 = 3; //Declara e inicializa num1
num2 = 4; //Inicializa num2
console.log(num1 + " " + num2); //'3 4'
var num2; //Declara num2 para hoisting

//Example 3 - Hoists
a = 'Cran'; //Inicializa a
b = 'berry'; //Inicializa b
console.log(a + "" + b); // 'Cranberry'
var a, b; //Declara ambos a & b para hoisting
```

5. Tipos de Dados

Embora fracamente tipada, é necessário conhecer os tipos primitivos e compostos da linguagem JS.

NUNCA ESQUEÇA

- QUASE TUDO É OBJETO
- Em JavaScript, os objetos são reis. Se você entender objetos, você entenderá JavaScript.
- Orientação a objetos aqui não é uma opção, é uma necessidade.

Quais são os objetos do JS?

Sempre serão objetos

- Date
- Math
- Regular Expression
- Array
- Function
- Object

Às vezes serão objetos*

- Boolean
- Number
- String

Nunca serão objetos

- undefined
- null

*Serão objetos quanto for usado NEW em sua definição

VALORES PRIMITIVOS

- Um valor primitivo não possui propriedades ou métodos. Um tipo de dado primitivo é aquele que possui um valor primitivo.
- São 5 os tipos primitivos no JS:
  - string
  - number
  - boolean
  - null
  - undefined

TIPOS DE DADOS

- Existem 8 tipos de dados básicos em JavaScript.
  - **Number** para números de qualquer tipo: inteiro ou ponto flutuante, os inteiros são limitados por .±(2^53-1)
  - **bigint** é para números inteiros de comprimento arbitrário.
  - **String** para texto. Uma string pode ter zero ou mais caracteres, não existe um tipo de caractere único separado (char).
  - **Boolean** para **true**/ **false**.
  - **Null** para valores desconhecidos - um tipo autônomo que possui um único valor **null**.

TIPOS DE DADOS (cont.)

- **undefined** para valores não atribuídos - um tipo autônomo que possui um único valor **undefined**.
- **object** para estruturas de dados mais complexas.
- **symbol** para identificadores exclusivos.

TYPEOF

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof 10n // "bigint"

typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof Math // "object"  (1)

typeof null // "object"  (2)

typeof alert // "function"  (3)
```

As últimas três linhas podem precisar de explicação adicional:

1. **Math** é um objeto embutido que fornece operações matemáticas. Vamos aprender no capítulo Números . Aqui, ele serve apenas como um exemplo de um objeto.
2. O resultado de typeof null é "object". Esse é um erro de typeof, comportamento oficialmente reconhecido, vindo dos primeiros dias do JavaScript e mantido para compatibilidade. Definitivamente, null não é um objeto. É um valor especial com um tipo próprio separado.
3. O resultado de typeof alert é "function", porque alert é uma função. As funções pertencem ao tipo de objeto. Mas os typeof trata de forma diferente, voltando "function". Isso também vem desde os primeiros dias do JavaScript. Tecnicamente, esse comportamento não é correto, mas pode ser conveniente na prática.

NUMBER

- O tipo de número representa números inteiros e de ponto flutuante.
- Além de números regulares, existem os chamados “valores numéricos especiais” que também pertencem a este tipo de dados: **Infinity**, **-Infinity**, **NaN**.
- **Infinity** representa o infinito matemático ∞. É um valor especial maior do que qualquer número.
- **-Infinity** representa o infinito matemático negativo -∞. É um valor especial menor do que qualquer número.

NUMBER

- **Infinity** representa o infinito matemático ∞. É um valor especial maior do que qualquer número.
- **-Infinity** representa o infinito matemático negativo -∞. É um valor especial menor do que qualquer número.
- **NaN** (not a number) representa um erro computacional. É o resultado de uma operação matemática incorreta ou indefinida.
- NaN é pegajoso. Qualquer outra operação de NaN retorna NaN. ortanto, se houver NaN em algum lugar em uma expressão matemática, ele se propagará para todo o resultado.

BIGINT

- Em JavaScript, o tipo “number” não pode representar valores inteiros maiores que (2^53-1) (use seja 9007199254740991), ou menores -(2^53-1) para negativos. É uma **limitação técnica** causada por sua representação interna.

Operações matemáticas seguras

- Fazer matemática é “**seguro**” em JavaScript. Podemos fazer qualquer coisa: dividir por zero, tratar strings não numéricas como números, etc.
- O script nunca irá parar com um erro fatal. Na pior das hipóteses, obteremos NaN como resultado.

CONTINUA.