Aula 2 - Sintaxe da Linguagem JavaScript

aTip Learn - Lógica de Programação com JavaScript

Objetivos da Aula

- Conhecer a sintaxe da linguagem JavaScript.
- Conhecer diferentes formas de declarar variáveis.
- Conhecer os principais operadores da linguagem JavaScript.

Sintaxe da Linguagem JavaScript

O que é sintaxe?

Uma linguagem de programação, assim como uma linguagem natural, possui um conjunto de regras que determina como as palavras e símbolos devem ser organizados para formar frases e expressões que carreguem algum sentido ou significado. A essas regras damos o nome de sintaxe, ou regras sintáticas.

Enquanto na linguagem natural podemos nos expressar de diferentes formas, nas linguagens de programação temos um conjunto de regras mais rígidas que precisam ser seguidas para que o código seja interpretado corretamente pelo computador.

Comentários

Comentários são partes do código-fonte que são ignoradas pelo interpretador da linguagem. Dessa forma os comentários são geralmente utilizados para documentar o código explicando o que ele faz, o motivo de determinadas decisões, ou qualquer outra informação relevante para quem precisar ler e entender o código no futuro.

A linguagem JavaScript possui dois tipos de comentários:

1. **Comentários de Linha Única**: São comentários que ocupam apenas uma linha do código e são iniciados com `//`. Tudo que vem após `//` na mesma linha é considerado um comentário e é ignorado pelo interpretador.

```
// Este é um comentário de linha única
console.log('Olá Mundo!'); // O código no início da linha é executado, porém tudo que vier depois do //
```

2. **Comentários de Múltiplas Linhas**: São comentários que podem ocupar várias linhas do código e são iniciados com `/*` e finalizados com `*/`. Tudo que estiver entre `/*` e `*/` é considerado um comentário e é ignorado pelo interpretador.

```
console.log('Essa mensagem é exibida!');
/*
Este é um comentário de múltiplas linhas
Tudo que está entre os delimitadores é ignorado pelo interpretador
*/
console.log('Essa mensagem também é exibida!');
```

- Documentação sobre Comentários

Ponto e Vírgula

O ponto e vírgula é utilizado para separar instruções em JavaScript, e geralmente é utilizado no final de cada linha de código. Utilizar o ponto e vírgula no final de cada linha é uma boa prática da linguagem, porém, seu uso não é obrigatório, pois o interpretador da linguagem JavaScript é capaz de inferir o final de uma instrução mesmo sem o uso do ponto e vírgula. Entretanto, existem casos em que a ausência do ponto e vírgula pode gerar erros de interpretação, além de dificultar o entendimento do código, por isso seu uso é recomendado e visto como uma boa prática.

```
let idade = 30 // Instrução sem ponto e vírgula
let nome = 'João'; // Instrução com ponto e vírgula
```

- Documentação sobre Ponto e Vírgula

Letras Maiúsculas e Minúsculas

JavaScript é uma linguagem case-sensitive, ou seja, ela faz distinção entre letras maiúsculas e minúsculas. Isso quer dizer que `nome`, `Nome` e `NOME` são considerados identificadores diferentes em JavaScript. Por conta disso, é importante manter a consistência na escrita do código, utilizando sempre o mesmo padrão de escrita para os identificadores.

```
let nome = 'João'; // Variável nome
let Nome = 'Maria'; // Variável Nome
let NOME = 'José'; // Variável NOME
```

Apesar de não existir uma regra para a escolha do padrão de escrita, a maioria dos programadores que utilizam a linguagem JavaScript adotam o padrão camelCase para nomear variáveis e funções. No padrão camelCase, a primeira palavra é escrita com letra minúscula, e as palavras seguintes são escritas com a primeira letra em maiúscula, sem espaços ou caracteres especiais entre as palavras.

```
let nomeCompleto = 'João Silva'; // Variável nomeCompleto
function exibirMensagem() { // Função exibirMensagem
    ...
}
```

Os programados de JavaScript também fazem uso do padrão PascalCase para nomear classes (veremos mais adiante o que são classes). No padrão PascalCase, todas as palavras são escritas com a primeira letra em maiúscula, sem espaços ou caracteres especiais entre as palavras.

```
class Usuario { // Classe Usuario
    ...
}
```

- Documentação sobre Convenções de Estilo (em Inglês)

Indentação

Indentação é a prática de adicionar espaços ou tabulações no início de uma linha de código para tornar o código mais legível. A indentação é utilizada para organizar o código, facilitando a leitura e entendimento do código, além de destacar a estrutura do código, como blocos de código, funções, estruturas de controle, etc. Algumas linguagens, como Python, utilizam a tabulação para definir blocos de código, enquanto outras, como JavaScript, utilizam a tabulação apenas para fins de organização e legibilidade, ou seja, escrever um código sem indentação não irá gerar um erro na linguagem, porém, irá tornar seu código mais díficil de ler e entender.

Convencionou-se que cada novo bloco de código deve ser indentado com um número fixo de espaços ou tabulações, geralmente 2 ou 4 espaços. A escolha do número de espaços é uma questão de preferência pessoal, ou da equipe de desenvolvimento, porém, é importante manter a consistência na escrita do código, utilizando sempre o mesmo número de espaços para indentar o código. A maioria dos editores de código modernos possuem funcionalidades que facilitam a indentação do código, identando automaticamente o código à medida que você escreve. Além disso existem ferramentas, como o utilitário Prettier, que são capazes de formatar o código automaticamente, seguindo um padrão de indentação pré-definido.

```
if (condicao) {
    // Bloco de código indentado com 4 espaços
    console.log('Primeira Mensagem');
    if (outraCondicao) {
        // Bloco de código indentado com 4 espaços
```

```
        console.log('Segunda Mensagem');
    }
}
```

Palavras-Chave

As palavras-chave são termos reservados da linguagem JavaScript que possuem um significado específico e não podem ser utilizados como nomes de variáveis, funções ou qualquer outro identificador da linguagem, são exemplos de palavras-chave `var`, `let` e `const` que utilizamos para declarar variáveis.

Você pode encontrar uma lista completa de palavras-chave da linguagem JavaScript no site da Mozilla Developer Network.

Declaração de Variáveis

Como vimos na aula anterior, as variáveis são utilizadas para identificar e referenciar dados armazenados na memória do computador, quando declaramos uma variável estamos dando um nome para um determinado espaço na memória do computador, podemos então usar esse nome para armazenar ou ler informações da memória.

A linguagem JavaScript permite que declaremos variáveis de três formas diferentes, cada forma é identificada por uma palavra-chave da linguagem:

1. **var**: É a forma mais antiga de declarar variáveis em JavaScript. Com a palavra-chave `var` criamos variáveis que podem ser lidas em qualquer parte do código, até mesmo antes da sua declaração, ou seja, o JavaScript permite que você utilize a variável em linhas de código que estão antes da sua declaração, uma funcionalidade conhecida como **hoisting**. Apesar de facilitar a escrita do código, o hoisting também pode ser motivo de confusão, e por conta disso nas versões mais recentes do JavaScript esse tipo de declaração caiu em desuso em favor das próximas duas formas de declaração.
2. **let**: A palavra-chave `let` foi introduzida no JavaScript a partir da versão ES6 (ECMAScript 2015) e é a forma mais recomendada de declarar variáveis em JavaScript. A declaração de variáveis com `let` não permite o hoisting, ou seja, você não pode utilizar a variável antes de declará-la. Isso torna o código mais fácil de entender e menos propenso a erros. Além disso, as variáveis declaradas com `let` são acessíveis apenas no bloco de código onde foram declaradas, ou seja, se você declarar uma variável dentro de um bloco de código, ela não poderá ser acessada fora desse bloco. Você entenderá melhor o que isso quer dizer quando estudarmos estruturas de controle e funções.
3. **const**: A palavra-chave `const` também foi introduzida no JavaScript a partir da versão ES6 e é utilizada para declarar constantes. Se funcionamento é parecido com o da palavra chave `let`, porém, uma vez que uma constante é declarada, seu valor não pode ser alterado. Isso significa que você não pode atribuir um novo valor a uma constante depois de declará-la.

Veja abaixo a sintaxe para declarar variáveis com cada uma das palavras-chave:

```
var nome = 'João';
let idade = 30;
const PI = 3.1415;
```

Diferente de `let`e `var`, a palavra-chave `const` exige que a variável seja inicializada no momento da declaração, ou seja, você não pode declarar uma constante sem atribuir um valor a ela, além disso não é possível alterar seu valor, por isso os códigos abaixo resultariam em erros:

```
const PI;
PI = 3.1415; // Erro: SyntaxError: Missing initializer in const declaration

const X = 10;
X = 20; // Erro: TypeError: Assignment to constant variable.
```

Em JavaScript podemos ainda declarar diversas variáveis em uma única linha, separando-as por vírgula:

```
let nome = 'João', idade = 30, cidade = 'São Paulo';
```

Como JavaScript é uma linguagem de tipagem dinâmica, não precisamos informar o tipo de dado que a variável irá armazenar, o próprio JavaScript irá inferir o tipo de dado com base no valor que atribuímos à variável. Além disso, é possível alterar o tipo de dado armazenado em uma variável a qualquer momento, por exemplo:

```
let nome = 'João';
nome = 30; // Agora a variável nome armazena um número
```

Apesar de ser possível, alterar o tipo de dado de uma variável pode tornar o seu código confuso e mais propenso a erros, por isso é recomendado que você mantenha o tipo de dado armazenado em uma variável sempre o mesmo.

- Mais Informações sobre Variáveis

Operadores

Operadores são símbolos que representam diferentes tipos de operações que podem ser realizadas em JavaScript, como operações matemáticas, lógicas e de comparação. Os operadores são classificados de acordo com o número de operandos que eles recebem:

1. **Operadores Unários**: São operadores que recebem apenas um operando. Um exemplo de operador unário é o operador de negação `!`, que inverte o valor de uma expressão lógica (ex. `!true` é igual a `false`, e `!false` é igual a `true`).
2. **Operadores Binários**: São operadores que recebem dois operandos. Um exemplo de operador binário é o operador de adição `+`, que soma dois valores (ex. `2 + 3` é igual a `5`).
3. **Operadores Ternários**: São operadores que recebem três operandos. O único operador ternário em JavaScript é o operador condicional `? :`, que é utilizado para criar expressões condicionais (ex. `condição ? valor1 : valor2`). Nós exploraremos o operador condicional com mais detalhes quando estudarmos estruturas de controle.

A seguir, apresentamos os principais operadores da linguagem JavaScript, por categoria:

Operadores Aritméticos

Operador	Exemplo	Nome	Descrição
`+`	`2 + 3`	Adição	Realiza a adição entre dois valores
`-`	`5 - 3`	Subtração	Realiza a subtração entre dois valores
`*`	`2 * 3`	Multiplicação	Realiza a multiplicação entre dois valores
`/`	`6 / 2`	Divisão	Realiza a divisão entre dois valores
`%`	`5 % 2`	Resto da divisão	Retorna o resto da divisão entre dois valores
`++`	`x++`	Incremento	Incrementa o valor de uma variável em 1
`--`	`x--`	Decremento	Decrementa o valor de uma variável em 1

Operadores Lógicos

Operador	Exemplo	Nome	Descrição
`&&`	`true && false`	E	Retorna `true` se ambos os operandos forem verdadeiros
`||`	`true || false`	OU	Retorna `true` se pelo menos um dos operandos for verdadeiro
`!`	`!true`	Negação	Inverte o valor de uma expressão lógica

Operadores de Comparação

Operador	Exemplo	Nome	Descrição
`==`	`2 == 2`	Igualdade	Retorna `true` se os operandos forem iguais
`!=`	`2 != 3`	Diferença	Retorna `true` se os operandos forem diferentes
`===`	`2 === '2'`	Igualdade Estrita	Retorna `true` se os operandos forem iguais e do mesmo tipo
`!==`	`2 !== '2'`	Diferença Estrita	Retorna `true` se os operandos forem diferentes ou de tipos diferentes
`>`	`5 > 3`	Maior que	Retorna `true` se o operando da esquerda for maior que o operando da direita
`<`	`3 < 5`	Menor que	Retorna `true` se o operando da esquerda for menor que o operando da direita
`>=`	`5 >= 5`	Maior ou igual	Retorna `true` se o operando da esquerda for maior ou igual ao operando da direita
`<=`	`3 <= 5`	Menor ou igual	Retorna `true` se o operando da esquerda for menor ou igual ao operando da direita

Igualdade vs Igualdade Estrita: Em JavaScript, temos dois operadores de igualdade (e de diferença), o operador `==` e o operador `===`. O operador `==` realiza uma comparação de igualdade entre dois valores, porém, ele não leva em consideração o tipo de dado dos valores, ou seja, ele converte os valores para o mesmo tipo antes de realizar a comparação. Já o operador `===` realiza uma comparação de igualdade estrita, ou seja, ele compara os valores e os tipos dos valores, retornando `true` apenas se os valores forem iguais e do mesmo tipo.

```
2 == '2'; // true
2 === '2'; // false
```

```
2 != '2'; // false
2 !== '2'; // true
```

Para evitar erros de comparação, é recomendável utilizar os operadores de igualdade e diferença estrita (`===` e `!==`) sempre que possível.

Operadores de Atribuição

Operador	Exemplo	Nome	Descrição
`=`	`x = 10`	Atribuição	Atribui um valor a uma variável
`+=`	`x += 5`	Atribuição com Adição	Adiciona um valor a uma variável
`-=`	`x -= 5`	Atribuição com Subtração	Subtrai um valor de uma variável
`*=`	`x *= 5`	Atribuição com Multiplicação	Multiplica uma variável por um valor
`/=`	`x /= 5`	Atribuição com Divisão	Divide uma variável por um valor
`%=`	`x %= 5`	Atribuição com Resto da Divisão	Atribui o resto da divisão de uma variável por um valor

Operador de Concatenação

O operador `+`, além de ser utilizado para realizar operações matemáticas de adição, também é utilizado para concatenar strings em JavaScript. Quando utilizamos o operador `+` para concatenar strings, ele irá unir os valores das strings em uma única string.

```
let nome = 'João';
let sobrenome = 'Silva';
let nomeCompleto = nome + ' ' + sobrenome; // 'João Silva'
```

Assim, sempre que um dos operandos do operador `+` for uma string, o JavaScript irá realizar a concatenação dos valores ao invés da soma, isso quer dizer que no exemplo abaixo o valor da variável `resultado` será a string `'25'` e não o número `7`.

```
let numero = 2;
let resultado = numero + '5'; // '25'
```

Outros Operadores e Precedência de Operadores

A linguagem JavaScript possui diversos outros operadores que são utilizados para realizar diferentes tipos de operações. Além disso a linguagem possui regras de precedência que determinam a ordem em que os operadores são avaliados em uma expressão. Você pode encontrar uma lista completa de operadores e a ordem de precedência deles no site da Mozilla Developer Network.