

Linguagem de Programação



Prof. Renato Carioca Duarte



Unidade 1

Linguagem Javascript

- JavaScript é uma linguagem de programação de alto nível, interpretada e orientada a objetos.
- Ela foi originalmente desenvolvida pela Netscape em 1995, e desde então se tornou uma das linguagens mais populares para o desenvolvimento web.
- JavaScript foi criado por Brendan Eich, um engenheiro da Netscape, para adicionar interatividade a páginas web estáticas.
- Inicialmente, a linguagem foi chamada de "LiveScript", mas posteriormente foi renomeada para JavaScript para aproveitar o sucesso da linguagem Java.

Linguagem Javascript

- Ao longo dos anos, JavaScript evoluiu para além de sua utilização básica no front-end e agora é amplamente utilizado em uma variedade de ambientes, incluindo o desenvolvimento de aplicativos web, servidores, dispositivos móveis e até mesmo para criação de aplicativos de desktop.
- Além disso, frameworks populares como React, Angular e Vue.js são construídos em cima do JavaScript.
- Uma das características mais poderosas do JavaScript é sua capacidade de ser executado diretamente nos navegadores web, permitindo a manipulação dinâmica do conteúdo HTML, interação com o usuário, validação de formulários, animações, requisições assíncronas e muito mais.

Linguagem Javascript

- JavaScript é uma linguagem versátil e flexível, e sua popularidade cresceu significativamente com o passar dos anos.
- Atualmente, ela é suportada por todos os principais navegadores web e possui uma vasta comunidade de desenvolvedores, bibliotecas e frameworks que contribuem para sua constante evolução.

ECMAScript é o nome oficial da linguagem.

Linguagem ECMAScript

- A especificação da linguagem JavaScript é mantida pela organização Ecma International.
- Ecma International é uma associação de padronização sem fins lucrativos que se dedica à padronização de tecnologias, incluindo linguagens de programação.
- No caso do JavaScript, a especificação é conhecida como ECMAScript, que é o nome oficial da linguagem.
- A primeira edição do ECMAScript foi lançada em 1997, e desde então, houve várias revisões e atualizações para acompanhar o desenvolvimento contínuo da linguagem.

- A Ecma International, por meio do seu comitê técnico chamado TC39 (Technical Committee 39), é responsável por manter e evoluir a especificação do ECMAScript.
- O TC39 é composto por representantes de várias empresas e organizações relacionadas à tecnologia, como Mozilla, Google, Microsoft, Apple e muitas outras.
- Essas empresas desempenham um papel ativo no desenvolvimento e aprimoramento da linguagem JavaScript.
- Cada uma dessas organizações é responsável por implementar a especificação do ECMAScript em seus respectivos produtos e garantir a compatibilidade com a linguagem JavaScript.

1. Linguagem de programação interpretada:

- JavaScript é uma linguagem interpretada, o que significa que o código-fonte é executado diretamente por um interpretador em tempo de execução, sem a necessidade de compilação prévia.

2. Linguagem orientada a objetos:

- JavaScript possui suporte a programação orientada a objetos (POO), permitindo a criação de objetos, herança e encapsulamento.
- No entanto, seu modelo de objetos é baseado em protótipos em vez de classes, o que o diferencia de outras linguagens orientadas a objetos mais tradicionais.

3. Tipagem dinâmica:

- JavaScript é uma linguagem de tipagem dinâmica, o que significa que as variáveis não têm um tipo estático definido.
- Os tipos são verificados em tempo de execução, permitindo que as variáveis mudem de tipo durante a execução do programa.

4. Manipulação do Document Object Model (DOM):

- JavaScript é amplamente utilizado para interagir com o DOM, que é a representação em árvore do conteúdo HTML de uma página web.
- Ele permite que os desenvolvedores manipulem elementos HTML, adicionem, removam ou modifiquem o conteúdo e respondam a eventos do usuário.

5. Suporte a assincronismo:

- JavaScript possui recursos embutidos para trabalhar com operações assíncronas.
- Por exemplo, é possível fazer requisições HTTP assíncronas usando a API Fetch ou XMLHttpRequest, ou usar Promises e async/await para lidar com chamadas assíncronas de maneira mais legível e concisa.

6. Amplamente suportado nos navegadores:

- JavaScript é a linguagem de programação padrão dos navegadores web modernos.
- Isso significa que qualquer navegador moderno oferece suporte à execução de código JavaScript, permitindo que os desenvolvedores criem interações dinâmicas e ricas em conteúdo para páginas web.

7. Ecossistema robusto:

- JavaScript possui um ecossistema vibrante e rico, com uma ampla variedade de bibliotecas, frameworks e ferramentas disponíveis.
- Isso inclui bibliotecas populares como React, Vue.js e Angular, bem como uma infinidade de módulos e pacotes disponíveis no gerenciador de pacotes npm.
- Essas são apenas algumas das características técnicas mais notáveis do JavaScript.
- A linguagem continua a evoluir rapidamente, com atualizações regulares da especificação ECMAScript e a introdução de novos recursos e melhorias.

Sintaxe Básica do Javascript

- A sintaxe básica do JavaScript é a forma como você escreve o código JavaScript, seguindo uma série de regras e convenções.
- A sintaxe correta é crucial para que o código seja interpretado corretamente e executado sem erros.
- Aqui estão alguns conceitos e exemplos da sintaxe básica do JavaScript:

Declaração de variáveis

- Utilize a palavra-chave ``let`` ou ``const`` para declarar uma variável.

Declaração de Variáveis

- Em JavaScript, existem três palavras-chave para declarar variáveis: ``var``, ``let`` e ``const``.
- Cada uma delas tem comportamentos e escopos diferentes:

Declaração de Variáveis

1. `var`:

- Escopo de função: Variáveis declaradas com `var` têm escopo de função, o que significa que são visíveis dentro da função em que são declaradas.
- Hoisting: As declarações de variáveis com `var` são movidas para o topo do escopo em que estão, permitindo que você use uma variável antes de declará-la. No entanto, seu valor inicial será `undefined`.
- Reatribuição e redeclaração: Variáveis declaradas com `var` podem ser reatribuídas e redeclaradas no mesmo escopo.

Declaração de Variáveis

```
var idade = 25;  
console.log(idade); // Saída: 25  
  
var nome = "Maria";  
console.log(nome); // Saída: Maria  
  
idade = 30; // Reatribuição  
console.log(idade); // Saída: 30  
  
var cidade; // Declaração sem atribuição  
console.log(cidade); // Saída: undefined  
cidade = "São Paulo";  
console.log(cidade); // Saída: São Paulo
```

Declaração de Variáveis

2. `let`:

- Escopo de bloco: Variáveis declaradas com `let` têm escopo de bloco, o que significa que são visíveis apenas dentro do bloco em que são declaradas, seja um bloco de função, um bloco de loop ou um bloco condicional.
- Não há hoisting: Diferente de `var`, as declarações de variáveis com `let` não são movidas para o topo do escopo. Você deve declarar uma variável antes de usá-la.
- Reatribuição: Variáveis declaradas com `let` podem ser reatribuídas (ter seu valor alterado), mas não podem ser redeclaradas no mesmo escopo.

Declaração de Variáveis

```
let idade = 25;  
console.log(idade); // Saída: 25  
  
let nome = "João";  
console.log(nome); // Saída: João  
  
idade = 30; // Reatribuição  
console.log(idade); // Saída: 30  
  
let cidade; // Declaração sem atribuição  
console.log(cidade); // Saída: undefined  
cidade = "Rio de Janeiro";  
console.log(cidade); // Saída: Rio de Janeiro
```

Declaração de Variáveis

3. `const`:

- Escopo de bloco: Variáveis declaradas com `const` também têm escopo de bloco.
- Não há hoisting: Assim como `let`, as declarações de variáveis com `const` não são movidas para o topo do escopo.
- Valor constante: Uma vez atribuído um valor, ele não pode ser alterado. A variável `const` é apenas para leitura e não pode ser reatribuída ou redeclarada no mesmo escopo.
- Constantes de referência: Se a variável `const` contiver um objeto ou um array, o valor do objeto ou array não pode ser alterado, mas suas propriedades ou elementos individuais podem ser modificados.

Declaração de Variáveis

```
const PI = 3.14;  
console.log(PI); // Saída: 3.14  
  
const nome = "Ana";  
console.log(nome); // Saída: Ana  
  
// Tentativa de reatribuição (gerará um erro)  
PI = 3.14159;  
  
const cidade = "Porto Alegre";  
console.log(cidade); // Saída: Porto Alegre
```

Declaração de Variáveis

- É recomendado utilizar ``let`` para variáveis que precisam de reatribuição
- É recomendado ``const`` para constantes ou valores que não devem ser alterados.
- O uso de ``var`` é menos comum atualmente, mas pode ser necessário em casos específicos de compatibilidade com versões mais antigas do JavaScript ou em situações em que o hoisting é desejado.

Hoisting

- Hoisting é um comportamento específico do JavaScript em relação à declaração de variáveis e funções.
- O termo "hoist" significa "elevar" ou "levantar", e o hoisting ocorre quando as declarações de variáveis e funções são movidas para o topo do escopo em que estão.

Hoisting de Variáveis

```
console.log(nome); // Saída: undefined  
var nome = "João";  
console.log(nome); // Saída: João
```

No exemplo acima, mesmo que a variável `nome` seja usada antes da sua declaração, não ocorre um erro.

Isso acontece porque a declaração da variável `nome` é "elevada" para o topo do escopo em que está, mas o valor ainda não é definido.

Portanto, a primeira impressão retorna `undefined`.

A atribuição ocorre na ordem em que o código é escrito, então a segunda impressão retorna o valor "João".

Hoisting de Funções

```
imprimirMensagem(); // Saída: "Olá!"  
function imprimirMensagem() {  
  console.log("Olá!");  
}
```

Nesse exemplo, a função `imprimirMensagem()` é chamada antes da sua declaração.

No entanto, o hoisting faz com que a declaração da função seja movida para o topo do escopo, tornando-a acessível antes mesmo da sua declaração.

Portanto, a função é executada sem erros, resultando na impressão "Olá!".

Hoisting

- É importante destacar que apenas a declaração de variáveis e funções é elevada (“hoistada”) , não as atribuições ou definições de valor.
- Portanto, é recomendado sempre declarar variáveis e funções antes de usá-las para evitar comportamentos inesperados e tornar o código mais legível.
- Embora o hoisting seja um recurso do JavaScript, é uma boa prática de programação declarar as variáveis e funções antes de utilizá-las, para evitar confusões e garantir a clareza do código.

Comentários

- Utilize `//` para comentários de uma única linha.
- Utilize `/* ... */` para comentários de várias linhas.

```
// Isso é um comentário de uma única linha
```

```
/*  
Isso é um comentário  
de várias linhas  
*/
```

Operadores

Grupo	Operador	Descrição
Aritméticos	+	Adição
	-	Subtração
	*	Multiplicação
	/	Divisão
	%	Resto da divisão
	**	Exponenciação
	++	Decremento
	--	Maior ou igual

Operadores

Grupo	Operador	Descrição
Atribuição	=	Atribuição simples
	+=	Atribuição de adição
	-=	Atribuição de subtração
	*=	Atribuição de multiplicação
	/=	Atribuição de divisão
	%=	Atribuição de resto
	**=	Atribuição de exponenciação

Grupo	Operador	Descrição
Relacional	<code>==</code>	Igual
	<code>===</code>	Exatamente igual (conteúdo e tipo de dado)
	<code>!=</code>	Diferente
	<code>!==</code>	Exatamente diferente (conteúdo e tipo de dado)
	<code><</code>	Menor
	<code><=</code>	Menor ou igual
	<code>></code>	Maior
	<code>>=</code>	Maior ou igual

Grupo	Operador	Descrição
Lógicos	& &	E (AND)
		OU (OR)
	!	NÃO (NOT)

Condição

- Uma condição, em programação, é uma expressão que pode ser avaliada como verdadeira ou falsa.
- As estruturas condicionais são usadas para executar diferentes blocos de código com base no resultado dessa avaliação.
- O `if` é uma estrutura de controle condicional que permite executar um bloco de código se uma determinada condição for verdadeira.
- Ele possui a seguinte sintaxe:

```
if (condição) {  
    // bloco de código a ser executado se a condição for verdadeira  
}
```

Condição

- Nesse exemplo, se a variável `idade` for maior ou igual a 18, a mensagem "Você é maior de idade" será exibida no console.

```
let idade = 18;  
  
if (idade >= 18) {  
  console.log("Você é maior de idade.");  
}
```

Condição

- A estrutura if-else é uma extensão da estrutura if e permite definir um bloco de código a ser executado quando uma condição é verdadeira e outro bloco de código a ser executado quando a condição é falsa.
- Isso permite que você faça uma escolha entre dois caminhos diferentes de execução com base no resultado da condição.
- Ele possui a seguinte sintaxe:

```
if (condição) {  
    // bloco de código a ser executado se a condição for verdadeira  
} else {  
    // bloco de código a ser executado se a condição for falsa  
}
```


Condição

- Neste caso, se a variável temperatura for maior que 30, a mensagem "Está muito quente!" será exibida.
- Caso contrário, a mensagem "A temperatura está agradável." será exibida.

```
let temperatura = 25;

if (temperatura > 30) {
  console.log("Está muito quente!");
} else {
  console.log("A temperatura está agradável.");
}
```

Condição

- Nesse exemplo, a mensagem "Bom dia!" será exibida se a variável hora for menor que 12.
- Se hora for menor que 18, a mensagem "Boa tarde!" será exibida.
- Caso contrário, a mensagem "Boa noite!" será exibida.

```
let hora = 14;

if (hora < 12) {
  console.log("Bom dia!");
} else if (hora < 18) {
  console.log("Boa tarde!");
} else {
  console.log("Boa noite!");
}
```

Repetição

- As estruturas de repetição, também conhecidas como loops, são recursos em programação que permitem executar repetidamente um bloco de código com base em uma condição ou um número específico de iterações.
- Elas são usadas para automatizar tarefas repetitivas e economizar código, evitando a repetição de instruções semelhantes.
- Existem três principais estruturas de repetição em JavaScript:
 1. For
 2. While
 3. Do While

Repetição - For

- A estrutura `for` permite especificar um loop com uma inicialização, uma condição de continuação e um incremento.
- É uma estrutura de repetição mais compacta e comumente usada quando o número de iterações é conhecido.

Repetição - For

- A estrutura `for` permite especificar um loop com uma inicialização, uma condição de continuação e um incremento.
- É uma estrutura de repetição mais compacta e comumente usada quando o número de iterações é conhecido.
- Sintaxe:

```
for (inicialização; condição; incremento) {  
    // bloco de código a ser repetido  
}
```

Repetição - For

- Exemplo:

```
for (let i = 0; i < 5; i++) {  
  console.log(i);  
}
```

- Nesse exemplo, o loop `for` é configurado para iniciar com `i = 0`, continuar enquanto `i` for menor que 5 e incrementar `i` em 1 a cada iteração.
- O bloco de código dentro do `for` é executado repetidamente até que a condição seja falsa.

Repetição - WHILE

- A estrutura `for` permite especificar um loop com uma inicialização, uma condição de continuação e um incremento.
- É uma estrutura de repetição mais compacta e comumente usada quando o número de iterações é conhecido.

Repetição - WHILE

- A estrutura `while` executa repetidamente um bloco de código enquanto uma condição especificada for verdadeira.
- Antes de cada execução do bloco, a condição é verificada.
- Sintaxe:

```
while (condição) {  
    // bloco de código a ser repetido  
}
```


Repetição - WHILE

- Exemplo:

```
let contador = 0;
while (contador < 5) {
  console.log(contador);
  contador++;
}
```

- Nesse exemplo, o bloco de código dentro do `while` será executado repetidamente enquanto o valor da variável `contador` for menor que 5.
- A cada iteração, o valor do `contador` é impresso no console e incrementado em 1.

Repetição – DO WHILE

- A estrutura `do-while` é semelhante à estrutura `while`, mas a condição é verificada após a execução do bloco de código.
- Isso garante que o bloco seja executado pelo menos uma vez, independentemente da condição.

Repetição – DO WHILE

- A estrutura do-while é semelhante à estrutura while, mas a condição é verificada após a execução do bloco de código.
- Isso garante que o bloco seja executado pelo menos uma vez, independentemente da condição.
- Sintaxe:

```
do {  
    // bloco de código a ser repetido  
} while (condição);
```

Repetição – DO WHILE

- Exemplo:

```
let contador = 0;  
do {  
  console.log(contador);  
  contador++;  
} while (contador < 5);
```

- Nesse exemplo, o bloco de código dentro do `do` é executado pelo menos uma vez, pois a condição é verificada após a primeira execução.
- A partir da segunda iteração, a condição é verificada e, se for verdadeira, o bloco continua sendo executado

Funções

- Uma função é um bloco de código que pode ser chamado repetidamente.
- As funções são usadas para agrupar código relacionado e torná-lo mais reutilizável.
- Elas também podem ser usadas para passar argumentos para o código e retornar valores.
- Em JavaScript, as funções são declaradas usando a palavra-chave function.

Funções

- A sintaxe para declarar uma função é a seguinte:

```
function nomeDaFuncao(parametros) {  
    // corpo da função  
}
```

- O **nome da função** é o nome que será usado para chamar a função.
- Os **parâmetros** são as variáveis que serão passadas para a função.
- O **corpo da função** é o código que será executado quando a função for chamada.

Funções

- Para chamar uma função, use o nome da função seguido dos argumentos entre parênteses.
- Por exemplo:

```
nomeDaFuncao(argumento1, argumento2);
```

Funções

- As funções podem retornar valores. Para retornar um valor, use a palavra-chave return.
- Por exemplo:

```
function nomeDaFuncao(parametros) {  
    // corpo da função  
    return valor;  
}
```

- O valor retornado pela função será o valor que será retornado para o código que chamou a função.

Funções

```
function somar(numero1, numero2) {  
    return numero1 + numero2;  
}  
  
const resultado = somar(5, 6);  
console.log(resultado); // 11
```

- No exemplo acima, a função somar é declarada com dois parâmetros, numero1 e numero2.
- A função retorna a soma dos dois números.
- A função somar é então chamada com os argumentos 5 e 6.
- O resultado da função é armazenado na variável resultado e então impresso no console.

Escopo em JavaScript

- O escopo em JavaScript se refere à acessibilidade e visibilidade de variáveis e funções em diferentes partes do código.
- Existem dois tipos principais de escopos em JavaScript:
 - a) Escopo global: Variáveis e funções declaradas fora de qualquer função ou bloco de código têm escopo global e podem ser acessadas de qualquer lugar no código, incluindo outras funções.
 - a) Escopo local: Variáveis declaradas dentro de uma função têm escopo local e só podem ser acessadas dentro dessa função ou de funções aninhadas.

Escopo em JavaScript

```
// Escopo global
const globalVar = 'Eu sou uma variável global';

function exemploEscopo() {
  // Escopo local da função
  const localVar = 'Eu sou uma variável local';

  console.log(globalVar); // Acessando variável global dentro da função
  console.log(localVar);  // Acessando variável local dentro da função
}

exemploEscopo();
console.log(globalVar); // Acessando variável global fora da função
console.log(localVar);  // Resultará em um erro, pois localVar não é acessível fora da função
```

Escopo em JavaScript

- No exemplo anterior, `globalVar` tem escopo global e pode ser acessada tanto dentro como fora da função `exemploEscopo`.
 - Por outro lado, `localVar` tem escopo local e só pode ser acessada dentro da função `exemploEscopo`.
 - Tentar acessá-la fora dessa função resultará em um erro.
-
- Escopo local também se aplica a parâmetros de função.
 - Os parâmetros são considerados variáveis locais e só podem ser acessados dentro da própria função.

Entrada e Saída

- Programas Javascript escritos com Node.js para servidores não requerem interação com o usuário, sendo executados à nível de console, sem ambiente gráfico;
- No entanto, para programas Javascript client-side interface HTML é requerida juntamente com o uso do DOM (Document Object Model);
- Programas Javascript também podem efetuar interação com o usuário por meio de caixas de diálogo (dialog boxes);

Função alert()

```
1
2 <meta charset="UTF-8">
3 <script>
4     alert("Hello, World!")
5     window.alert("Hello, World novamente !!!");
6
7     alert(10 * 3);
8     alert(true);
9
10     alert("texto A", "texto B"); // somente texto A será exibido!
11 </script>
```

Função alert()

```
1  
2 <meta charset="UTF-8">  
3 <script>  
4  
5  
6  
7  
8  
9  
10 alert("texto A"  
11 </script>
```

Essa página diz

Hello, World!

OK

Essa página diz

Hello, World novamente !!!

OK

Essa página diz

30

OK

Essa página diz

true

OK

Essa página diz

texto A

OK

Função confirm()

- É um método que aceita um parâmetro opcional (mensagem) e exibe dois botões (OK e Cancel);
- Dependendo do botão pressionado pelo usuário, o método retorna um valor booleano;
- true se for pressionado OK e false caso Cancel.

Função confirm()

```
1
2 <meta charset="UTF-8">
3 <script>
4
5     let remove = confirm("Remover todos os dados ?");
6     let mensagem = remove ? "Dados serão deletados!" : "Operação cancelada!";
7
8     console.log(mensagem);
9     alert(mensagem);
10 </script>
```

Essa página diz
Remover todos os dados ?

OK

Cancelar

Essa página diz
Dados serão deletados!

OK

Função prompt()

- Ao se executar a função `prompt()` uma caixa de diálogo é exibida ao usuário com a mensagem especificada dentro das aspas;
- Se o usuário clicar em "OK" e inserir um valor, esse valor será atribuído à variável resposta.
- Se o usuário clicar em "Cancelar" ou deixar o campo em branco, a variável resposta receberá o valor `null`;
- A função `prompt()` é frequentemente usada para obter entrada interativa do usuário.

Função prompt()

```
1
2 <meta charset="UTF-8">
3 <script>
4
5     let resposta = prompt("Entre com seu time preferido:");
6     alert("Meu time preferido: " + resposta);
7
8 </script>
```

Essa página diz

Entre com seu time preferido:

OK Cancelar

Essa página diz

Meu time preferido: Juventos

OK



JS

Dúvidas

??????