Unidade I: Características do Javascript soulcode.com

**1 O que é o Javascript?**

JavaScript é uma linguagem de programação que permite implementar funci onalidades mais complexas em páginas web.

Sempre que uma página web faz mais do que apenas mostrar informações estáticas para você - ela mostra em tempo real conteúdos atualizados, mapas interativos, animações gráficas em 2D/3D, vídeos, etc. - você pode apostar que o Javascript provavelmente está envolvido.

**1.1 O que o Javascript pode fazer?**

O núcleo da linguagem JavaScript consiste em alguns benefícios comuns da programação que permite a você fazer coisas como:

• Armazena informações úteis em variáveis;

• Operações realizadas em pedaços de texto (strings);

• Executar códigos em resposta a uma determinada ação no documento HTML. Como o clique do usuário no botão;

• Dentre outras possibilidades.

**1.2 Como adicionar Javascript na página?**

**1.2.1 Javascript Interno**

Podemos inserir o Javascript da mesma maneira que fazemos com o CSS, há uma tag reservada para a escrita de códigos (**<script>**).

Basta adicionar essa tag antes do fechamento do body e escrever qualquer código JS, que ele será executado.

Listagem 1: Uso interno do JS

1 <script>

2 console.log("Hello, world!");

3 </script>

Observe, que nenhum código HTML é permitido dentro da tag <script>. O navegador ler o código HTML de cima para baixo e ao encontrar a tag ele executa o que estiver dentro dela.

Essa mensagem irá aparecer no console do navegador, você pode acessá lo usando o atalho F12 no navegador.

**1.3 Javascript Externo**

Podemos organizar melhor o Javascript em nossa aplicação criando arquivos externos, estes podem ser incorporados à nossa página HTML. Por exemplo: <script src="script.js"></script>

1

Unidade I: Características do Javascript soulcode.com

Dessa forma todo o código que estiver dentro do arquivo poderá ser exe cutado. Recomenda-se utilizar mais essa segunda forma, pois podemos or ganizar melhor os diversos scripts que uma página HTML pode possuir.

**Importante:** Iremos posteriormente discutir sobre a ordem das tags e o uso das propriedades **defer** e **async** e como elas impactam o carregamento da página.

**1.4 Comentários**

Como na maioria das linguagens, o Javascript suporta dois tipos de comentá rios:

• *// um comentário de linha* : Comentário de apenas uma linha; • */\* múltiplas linhas \*/* : Comentário que suporta quebra de linhas.

**2 Sintaxe do Javascript**

**2.1 Variáveis**

Uma variável é um container para um valor, como um número que podemos usar em uma operação de adição, ou uma sequência de texto que possamos usar como parte de uma frase.

Mas uma coisa especial a respeito das variáveis é que seu conteúdo pode mudar.

Um código sem variáveis seria impossível de construir pois elas são a base da programação, precisamos dela seja para salvar resultados ou dados inter mediários de algum processamento.

As variáveis são declaradas, ou seja, seu nome e conteúdo são explicita mente determinados, ao criar uma variável é como se um pequeno espaço na memória fosse reservado para o conteúdo dela.

Vale ressaltar, que as variáveis não são os valores em si, mas sim o con tainer para eles. Imagine como uma gaveta de um armário, nele eu guardo alguns itens úteis e posso abrir ela para retirar esse item ou fazer algo útil com ele, não significa que aquela gaveta é reservada apenas para aquele item, eu posso trocar o item que estiver dentro da gaveta por outro.

**2.1.1 Declarando variáveis**

Para declarar variáveis com Javascript, temos algumas formas, porém iremos dar início aos nossos estudos usando uma forma moderna e mais recomen dada: **let**.

Listagem 2: Declaração de variáveis via let

1 let nome;

2 let idade;

2

Unidade I: Características do Javascript soulcode.com

3 let nota;

No código acima, podemos observar a declaração de três variáveis seguindo a sintaxe:

let <nome da variável>;

Segue algumas regras para nomenclatura de variáveis:

1. **Comece com uma letra**: O nome da variável deve começar com uma letra (maiúscula ou minúscula) ou um sublinhado (\_). Não é possível começar o nome de uma variável com um número ou outros caracteres especiais;

2. **Use caracteres alfanuméricos**: Após o primeiro caractere, você pode usar letras (maiúsculas e minúsculas) e números em um nome de va riável. Evite usar caracteres especiais, como !, $ ou %;

3. **Evite palavras reservadas**: Não use palavras reservadas do JavaScript, como var, function, if, etc., como nomes de variáveis;

4. **Seja descritivo**: Escolha nomes de variáveis que sejam descritivos e sig nificativos para o propósito da variável. Isso ajudará a entender o código mais facilmente;

5. **Use camelCase**: A convenção mais comum em JavaScript é usar camel Case para nomes de variáveis compostos por várias palavras. Comece cada palavra após a primeira com uma letra maiúscula, sem espaços ou sublinhados. Por exemplo: meuNome, quantidadeTotal, listaDeCom pras;

6. **Evite abreviações excessivas**: Embora seja bom ser conciso, evite abre viações excessivas que possam tornar o nome da variável difícil de en tender. Equilíbrio entre a clareza e a brevidade;

7. **Seja consistente**: Mantenha uma convenção consistente ao nomear va riáveis em todo o seu código para facilitar a leitura e a manutenção. Por exemplo, se você usar camelCase para nomear suas variáveis, continue usando-o em todo o código.

**2.1.2 Inicializando uma variável**

Para definir um valor para uma variável, podemos fazer isso ao mesmo tempo que declaramos, ou posteriormente no código. É importante entender que se uma variável não possui valor definido, o Javascript vai considerar que aquela variável possui valor **undefined**.

Listagem 3: Definição de valores

1 let nome;

2 let idade;

3 let nota;

4 ...

5 nome = "Pedro Alvares Cabral";

3

Unidade I: Características do Javascript soulcode.com

6 idade = 23;

7 nota = 9.5;

Para atribuir valores, utilizamos o operador de atribuição (**=**). Podemos tam bém declarar a variável e atribuir o valor em uma única linha: let telefone = "(00) 00000-0000";

**2.1.3 Atualizando uma variável**

Usando o mesmo operador de atribuição podemos atualizar o valor de uma variável posteriormente:

Listagem 4: Atualização de valores

1 let nome = "Pedro Alves";

2 let idade = 29;

3 ...

4 nome = "Pedro Alves Cabral";

5 idade = 30;

**2.1.4 Impedindo a atualização de valor**

Essa característica das variáveis do Javascript, permite um dinamismo enorme ao programa elaborado. Porém, existem variáveis com valores que não que remos que sejam modificados. Para isso podemos fazer uso da palavra reser vada **const**.

Listagem 5: Declaração de variáveis com const

1 const PI = 3.14;

2 PI = 10.0; // erro

Ao invés de usar **let**, utilizamos **const**, isso impedirá qualquer tentativa de atualizar o valor da variável *PI*.

Além da forma **let** e **const**, temos a palavra reservada **var** para declaração de variáveis, essa em questão é uma forma antiga e desencorajada de utili zar (veremos mais adiante algumas características de escopo de variáveis, na seção extra 3).

**2.2 Tipagem**

O JavaScript é uma linguagem de programação de tipagem dinâmica, o que significa que as variáveis não têm um tipo fixo e podem armazenar diferentes tipos de valores ao longo do tempo.

Isso difere de linguagens de tipagem estática, como Java ou C, em que as variáveis têm um tipo definido e restrito desde o momento da declaração. Em JavaScript, você pode atribuir um valor a uma variável sem declarar explicitamente o tipo da variável. O tipo é inferido automaticamente com base no valor atribuído. Vale ressaltar que as variáveis podem mudar de tipo durante a execução do programa:

4

Unidade I: Características do Javascript soulcode.com

Listagem 6: Tipagem dinâmica do JS

1 let nome = "Maria Antonieta";

2 nome = 32;

3 nome = 9.5;

Essa flexibilidade pode ser útil em alguns casos, mas também pode levar a erros difíceis de detectar. Para evitar erros de tipo inesperados, é importante estar ciente do tipo de dado que uma variável contém em um determinado momento.

No geral, a tipagem dinâmica do JavaScript oferece flexibilidade, mas tam bém requer atenção e cuidado na manipulação de tipos de dados para evitar erros inesperados.

**2.2.1 Tipos embutidos (built-in)**

Além disso, o JavaScript possui alguns tipos de dados primitivos embutidos, como **number** (números), **string** (sequências de caracteres), **boolean** (valores booleanos), **null** (valor nulo) e **undefined** (valor indefinido).

Existem também outros tipos de dados complexos, como **object** (objetos), **array** (matrizes) e **function** (funções).

Listagem 7: Tipos embutidos

1 let idade = 25; // number

2 let nome = "Joao"; // string

3 let ativo = true; // boolean

4 let semValor = null; // nulo

5 let indefinido = undefined; // undefined

6 let lista = [1, 2, 3, 4]; // array

7 let pessoa = {nome: "Carlos", idade: 20}; // object

**2.3 Operadores**

**2.3.1 Operadores aritméticos**

Segue os operadores aritméticos que auxiliam em diversas operações mate máticas:

Listagem 8: Operadores aritméticos

1 let a = 5 + 3; // Resultado: 8

2 let b = "Ola" + "Mundo"; // Resultado: "OlaMundo"

3 let a = 10 - 5; // Resultado: 5

4 let a = 2 \* 3; // Resultado: 6

5 let a = 10 / 2; // Resultado: 5

6 let a = 10 % 3; // Resultado: 1 (resto da divisao de 10 por 3 eh 1) 5

Unidade I: Características do Javascript soulcode.com

**2.3.2 Operadores de incremento e decremento**

Você pode usar os operadores de incremento e decremento para aumentar ou diminuir o valor de uma variável em 1.

Esses operadores são úteis em várias situações, como iterar em loops, controlar contadores ou realizar cálculos numéricos.

Existem duas formas de usar esses operadores: pré-incremento/decremento e pós-incremento/decremento.

1. Pré-incremento e pré-decremento

O **pré-incremento (++x)** aumenta o valor da variável antes de qualquer outra operação ser realizada.

Listagem 9: Pré-incremento

1 let x = 5;

2 let y = ++x; // O valor de 'x' eh incrementado para 6 antes de ser atribuido a 'y'.

3 // Agora, 'x' eh igual a 6 e 'y' tambem eh igual a 6.

O **pré-decremento (–x)** diminui o valor da variável antes de qualquer outra operação ser realizada.

Listagem 10: Pré-decremento

1 let x = 5;

2 let y = --x; // O valor de 'x' eh decrementado para 4 antes de ser atribuido a 'y'.

3 // Agora, 'x' eh igual a 4 e 'y' tambem eh igual a 4.

2. Pós-incremento e pós-decremento

O **pós-incremento (x++)** aumenta o valor da variável após a conclusão de outras operações.

Listagem 11: Pós-incremento

1 let x = 5;

2 let y = x++; // O valor de 'x' eh atribuido a 'y' e, em seguida, 'x' eh incrementado para 6.

3 // Agora, 'x' eh igual a 6 e 'y' eh igual a 5.

O **pós-decremento (x–)** diminui o valor da variável após a conclusão de outras operações.

Listagem 12: Pós-decremento

1 let x = 5;

2 let y = x--; // O valor de 'x' eh atribuido a 'y' e, em seguida, 'x' eh decrementado para 4.

3 // Agora, 'x' eh igual a 4 e 'y' eh igual a 5.

6

Unidade I: Características do Javascript soulcode.com

**2.3.3 Operadores de atribuição**

Já utilizamos este tipo de operador anteriormente, porém existem alguns de talhes a mais sobre este operador.

1. **Atribuição com operação adicional (+=)**

Listagem 13: Atribuição com adição

1 let x = 5;

2 x += 3; // Equivalente a: x = x + 3

3 // Agora, 'x' eh igual a 8

2. **Atribuição com operação de subtração (-=)**

Listagem 14: Atribuição com subtração

1 let x = 5;

2 x -= 3; // Equivalente a: x = x - 3

3 // Agora, 'x' eh igual a 2

3. **Atribuição com operação de multiplicação(\*=)**

Listagem 15: Atribuição com multiplicação

1 let x = 5;

2 x \*= 3; // Equivalente a: x = x \* 3

3 // Agora, 'x' eh igual a 15

4. **Atribuição com operação de divisão (/=)**

Listagem 16: Atribuição com divisão

1 let x = 10;

2 x /= 2; // Equivalente a: x = x / 2

3 // Agora, 'x' eh igual a 5

Os operadores de atribuição são úteis para simplificar a atribuição de va lores e a realização de operações em uma única etapa. Eles podem melhorar a legibilidade e a concisão do código.

**2.3.4 Operadores de comparação**

Os operadores de comparação são usados para comparar valores e retornar um valor **booleano** (*true* ou *false*) com base na comparação. Eles são comu mente usados em estruturas condicionais, como declarações if e loops, para tomar decisões com base em determinadas condições.

1. **Igual (==)**: O operador de igualdade compara se dois valores são iguais, realizando a conversão de tipo, se necessário.

7

Unidade I: Características do Javascript soulcode.com

Listagem 17: Operadores de igualdade

1 let x = 5;

2 let y = "5";

3 console.log(x == y); // Retorna true, pois 5 eh igual a "5" apos a conversao de tipo

2. **Igual estrito (===)**: O operador de igualdade estrita compara se dois va lores são iguais, sem realizar a conversão de tipo.

Listagem 18: Operadores de igualdade restrita

1 let x = 5;

2 let y = "5";

3 console.log(x === y); // Retorna false, pois 5 nao eh estritamente igual a "5" devido ao tipo diferente

3. **Diferente (!=)**: O operador de desigualdade compara se dois valores são diferentes, realizando a conversão de tipo, se necessário.

Listagem 19: Operadores de diferente

1 let x = 5;

2 let y = "5";

3 console.log(x != y); // Retorna false, pois 5 eh igual a "5" apos a conversao de tipo

4. **Diferente estrito (!==)**: O operador de desigualdade estrita compara se dois valores são diferentes, sem realizar a conversão de tipo.

Listagem 20: Operadores de diferença estrita

1 let x = 5;

2 let y = "5";

3 console.log(x !== y); // Retorna true, pois 5 nao eh estritamente igual a " 5" devido ao tipo diferente

5. **Maior que (>), Maior ou igual a (>=), Menor que (<), Menor ou igual a (<=)**: Esses operadores comparam se um valor é maior, maior ou igual, menor ou menor ou igual a outro valor, respectivamente.

Listagem 21: Operadores de comparação

1 let x = 5;

2 let y = 10;

3 console.log(x > y); // Retorna false

4 console.log(x >= y); // Retorna false

5 console.log(x < y); // Retorna true

6 console.log(x <= y); // Retorna true

**2.3.5 Operadores lógicos**

Os operadores lógicos são usados para combinar valores booleanos e retor nar um novo valor booleano com base na lógica das operações.

8

Unidade I: Características do Javascript soulcode.com

1. **AND lógico (&&)**: O operador **AND** lógico retorna true se ambos os ope randos forem true. Caso contrário, retorna false.

Listagem 22: Operador AND

1 let x = 5;

2 let y = 10;

3 console.log(x > 0 && y > 0); // Retorna true, pois ambos x e y sao maiores que 0

4 console.log(x > 0 && y < 0); // Retorna false, pois y nao eh maior que 0

2. **OR lógico (||)**: O operador **OR** lógico retorna true se pelo menos um dos operandos for true. Se ambos os operandos forem false, retorna false.

Listagem 23: Operador OR

1 let x = 5;

2 let y = 10;

3 console.log(x > 0 || y > 0); // Retorna true, pois pelo menos um dos operandos eh true

4 console.log(x < 0 || y < 0); // Retorna false, pois ambos os operandos sao false

3. **NOT lógico (!)**: O operador **NOT** lógico inverte o valor booleano do ope rando. Se o operando for true, retorna false. Se o operando for false, retorna true.

Listagem 24: Operador NOT

1 let x = 5;

2 let y = 10;

3 console.log(!(x > 0)); // Retorna false, pois x > 0 eh true, mas eh invertido pelo operador !

4 console.log(!(x < 0)); // Retorna true, pois x < 0 eh false, mas eh invertido pelo operador !

Esses operadores lógicos podem ser combinados para criar expressões ló gicas mais complexas. Você pode usar parênteses para agrupar as expressões e definir a precedência das operações.

Listagem 25: Expressão com operadores lógicos

1 let x = 5;

2 let y = 10;

3 let z = 15;

4 console.log((x > 0 && y > 0) || z > 0); // Retorna true, pois a expressao entre parenteses eh true

5 console.log(!(x > 0) && y > 0); // Retorna false, pois a primeira parte da expressao eh false

**3 Extra: Uso do var e hoisting**

No JavaScript, a palavra-chave **var** é usada para declarar variáveis. No en tanto, o **var** tem algumas características peculiares, como o hoisting. Hoisting

9

Unidade I: Características do Javascript soulcode.com

é o comportamento em JavaScript em que as declarações de variáveis (mas não as atribuições) são movidas para o topo do escopo em tempo de execu ção.

Isso significa que, independentemente de onde uma variável é declarada em um escopo, ela é automaticamente "elevada"para o topo do escopo du rante a fase de compilação (antes de executar).

Listagem 26: Demonstração do hoisting

1 console.log(x); // Retorna undefined

2 var x = 5;

3 console.log(x); // Retorna 5

No exemplo acima, mesmo que a variável "x"seja acessada antes de ser decla rada, não ocorre um erro. O JavaScript "eleva"a declaração da variável "x"para o topo, mas o valor atribuído a ela (5) é definido apenas quando a linha de código correspondente é alcançada.

É importante observar que o hoisting ocorre apenas com a declaração da variável, não com a atribuição. Isso significa que a variável é inicializada com o valor undefined até que a atribuição real seja encontrada.

No entanto, é importante destacar que o uso do **var** para declarar variáveis tem algumas limitações e desvantagens.

Por exemplo, o escopo do **var** é sempre a função em que é declarado, não respeitando blocos de controle de fluxo (veremos nas próximas unidades), o que pode levar a problemas de vazamento de variáveis e escopo indesejado.

Para evitar esses problemas, é recomendável usar as palavras-chave **let** e **const** demonstradas nas seções anteriores, que têm escopo de bloco mais restrito e comportamento mais previsível.

10