Curso Informática para Internet

LTPW1

Capítulo 07

Múltiplas ações, variáveis e conteúdo dinâmico

Três assuntos em um mesmo capítulo? Pois é. Estamos avançando no JavaScript e precisamos ampliar nossos horizontes um pouquinho. Preciso te mostrar alguns conceitos importantes para prosseguirmos e para que nossos scripts fiquem mais interessantes. Vá com calma, se por acaso estiver cansado(a), faça uma pausa, descansa um pouco antes de continuar. Afinal, não adianta tentar correr para aprender programação. Paciência, dedicação e prática: essa é a receita para o sucesso!

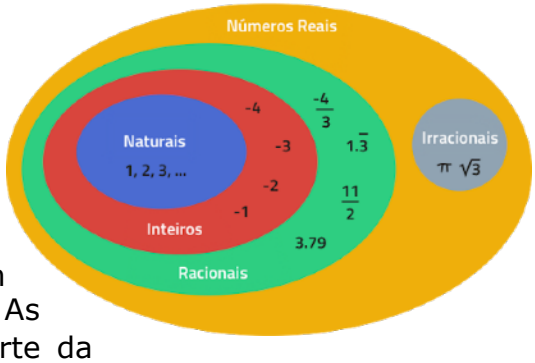
Vamos avançar um pouco com as variáveis

No capítulo 4, abordamos o conceito inicial de variáveis. Agora vamos nos aprofundar um pouco mais. Como você já deve saber, uma variável é um espaço na memória do computador, celular, tablet, TV, vídeo-game ou qualquer dispositivo que você use para acessar um site ou rodar um script JS.

IMPORTANTE: Provavelmente você está estudando hardware, arquitetura de computadores ou qualquer matéria similar. Pois saiba que todas as variáveis são armazenadas na **memória principal** do dispositivo (geralmente memória do tipo RAM).

Quando usamos comandos do tipo:

```
var a = 10
let b = 'JavaScript'
const c = 3.1415
```



A primeira linha cria uma variável chamada **a**, que guarda o **número inteiro** 10. A segunda linha também cria uma variável chamada **b**, que guarda uma **cadeia de caracteres** (também conhecida como ***String***) com o valor `JavaScript`. As aspas (simples, duplas ou crases) não fazem parte da String, apenas a delimitam.

Já a terceira linha declara uma **variável imutável** (também chamada de constante) com um **número real** com o valor 3.1415. Veja na imagem ao lado a relação existente entre números inteiros (Z) e reais (R).

RELEMBRANDO: Você deve se lembrar da época das suas aulas iniciais de Matemática, que **números inteiros** são aqueles que não possuem a vírgula (ponto flutuante). Já os **números reais** são aqueles que possuem parte fracionária separada por vírgula.

No JavaScript moderno, a distinção entre números inteiros e real está bem simplificada com o uso da função interna `Number()`. Antigamente usávamos bastante as funções `parseInt()` e `parseFloat()` para realizar a conversão para cada tipo específico.

Descobrindo o tipo de uma variável ou valor

A qualquer momento, podemos descobrir o tipo de uma variável ou valor, podemos usar a instrução typeof. Considere as linhas de exemplo acima e vamos analisar o tipo de cada uma delas através de comandos simples:

```
typeof a                // vai retornar “number”
typeof b                // vai retornar “string”
typeof c                // vai retornar “number”
```

Você pode usar o Console do seu DevTools para fazer esses testes. Inclusive, pode experimentar testes diretamente com valores ou expressões:

```
typeof 250              // vai retornar “number”
typeof '800'            // vai retornar “string”
typeof true             // vai retornar “boolean”
typeof ‘false'          // vai retornar “string”
typeof 200 * 5          // vai retornar “NaN”
typeof (500 / 3)        // vai retornar “number”
```

Na lista acima, temos algumas curiosidades:

- O valor `250` é considerado **number**, mas o valor `'800'` é uma **string**! Isso acontece porque o 800 está entre aspas e mesmo parecendo ser um número para nós, humanos, um computador sempre vai considerar qualquer coisa delimitada por aspas (simples, duplas ou crases) como uma cadeia de caracteres.
- O valor `true` é um valor lógico (só aceita verdadeiro ou falso), conhecidos pelo JS como o tipo **boolean**. A mesma linha de raciocínio feita anteriormente serve para o valor `'false'` testado logo em seguida. Mesmo parecendo um valor booleano, ele está entre aspas e vai ser considerado uma string.
- A penúltima linha retorna um termo que é novidade pra nós. Para o JavaScript, `NaN` significa NÃO É UM NÚMERO (*Not a Number*). O que a linguagem quer te dizer é que provavelmente você está tentando fazer uma conta ou mostrar um número, mas ele está gerando um resultado que não é numérico e ele não está conseguindo resolver.
- Já na última linha, a expressão pode ser resolvida e o resultado é um número. Sendo assim, se você quiser testar uma expressão usando `typeof`, coloque-a entre parênteses, ok?

Múltiplas ações em JavaScript

Até o momento, o que criamos aqui foram ações simples executadas em resposta a apenas um evento por vez. Somente um botão estava sendo monitorado. Agora vamos ver como podemos monitorar várias ações em um só código.

Abra seu diretório de exercícios JavaScript e crie uma pasta **ex010**. Nele, crie um arquivo chamado `index.html` e adicione as linhas básicas de um documento HTML5.

```
 9  <body>
10      <h1>JavaScript Externo</h1>
11      <button onclick="acao1()">Ação 1</button>
12      <button onclick="acao2()">Ação 2</button>
13      <button onclick="acao3()">Ação 3</button>
14      <button onclick="acao4()">Ação 4</button>
15      <section id="saida">
16          <p>Aqui vou registrar suas ações com os botões acima.</p>
17      </section>
18      <script src="acoes.js"></script>
19  </body>
```

Note que nas **linhas de 11 até 14**, criamos quatro botões diferentes e que cada um dispara uma ação diferente e não uma única funcionalidade.

Outra coisa que estamos fazendo pela primeira vez é a **linha 18**. Com ela, estamos evitando colocar um código JavaScript dentro de um arquivo HTML. Dessa maneira, nosso código fica muito mais enxuto e organizado, já que estamos separando o que é HTML do que é JS. Ao digitar a linha 18, segure a tecla `Ctrl` e clique sobre o nome do arquivo `acoes.js` (que até o momento não existe). Ao fazer isso, disparamos um pedido ao VS Code para que ele crie o arquivo automaticamente!

ANOTE AÍ: Sempre que você tiver uma ligação com outro arquivo externo em nossos código atual, basta usar o atalho `Ctrl+clique` em cima do nome do arquivo. Se o arquivo existir, o VS Code vai tentar abri-lo. Se ele não existir, vai tentar criá-lo. Essa é uma **MEGA DICA!**

No documento `acoes.js` que criamos, escreva apenas o código das funções.

```
1  function acao1() {
2      let resp = window.document.getElementById('saida')
3      resp.innerHTML += '<p>Clicou no primeiro botão</p>'
4  }
5
6  function acao2() {
7      let resp = window.document.getElementById('saida')
8      resp.innerHTML += '<p>Clicou no segundo botão</p>'
9  }
10
11 function acao3() {
12     let resp = window.document.getElementById('saida')
13     resp.innerHTML += '<p>Clicou no terceiro botão</p>'
14 }
15
16 function acao4() {
17     let resp = window.document.getElementById('saida')
18     resp.innerHTML += '<p>Clicou no quarto botão</p>'
19 }
```

escopo de resp

Se você analisar o código acima com cuidado, a **linha 2** está se repetindo várias vezes nas linhas 7, 12 e 17. Até o momento, isso é necessário porque toda função tem um bloco (delimitado por `{}`) e toda variável declarada dentro de um bloco só vai funcionar dentro desse bloco. **ANOTE MAIS ESSA DICA AÍ**!

O nome que damos a essa característica é **ESCOPO**. De forma resumida, o escopo de uma variável é a área onde ela existe e funciona (o que chamamos de **escopo local**). Na imagem acima, marquei o escopo da variável `resp` que está dentro da `function acao1()`. A minha variável `resp` do acao1() não vai funcionar fora daquela linha pontilhada. Por isso tivemos que criar várias versões da mesma variável, cada uma dentro de cada ação.

Porém, essa não é a melhor maneira de resolver esse problema. Vamos ver uma nova versão, bem melhor.

escopo global de `resp`

```
1  let resp = window.document.getElementById('saida')
2
3  function acao1() {
4      resp.innerHTML += '<p>Clicou no primeiro botão</p>'
5  }
6
7  function acao2() {
8      resp.innerHTML += '<p>Clicou no segundo botão</p>'
9  }
10
11 function acao3() {
12     resp.innerHTML += '<p>Clicou no terceiro botão</p>'
13 }
14
15 function acao4() {
16     resp.innerHTML += '<p>Clicou no quarto botão</p>'
17 }
```

Olha só o que aconteceu! Transferi a linha que declara a variável `resp` para fora das funções, lá no início, na linha imediatamente acima da primeira (não pode ser no final). Agora estamos dizendo ao JS que a variável criada tem **escopo global**, o que significa que ela vai funcionar dentro de todas as funções, é só olhar a nova linha pontilhada do segundo caso.

Conteúdo dinâmico

Outra característica do código acima é o seu comportamento dinâmico. Conforme você usa o site e vai clicando nos botões, ele vai adicionando conteúdo ao final da página sem apagar o conteúdo anterior, que era o que estava acontecendo antes (veja na imagem ao lado).

Isso tudo está acontecendo, simplesmente porque estamos usando o operador `+=` no lugar de usar o a atribuição simples `=` nas linhas 4, 8, 12 e 16.

Se você não entendeu o funcionamento do `+=`, volte para o capítulo anterior, onde explicamos os diversos tipos de operadores de atribuição. Faça aí uma experiência, modificando uma das ações e substituindo o `+=` pelo `=` e veja qual foi o resultado prático.

ATENÇÃO: Nesse caso, o operador `+` não funciona como uma soma aritmética. Na verdade, ele também pode funcionar como **concatenação**, juntando uma string em outra. Sendo assim, quando usamos `+=` com strings, estamos pedindo para o JavaScript não apagar o conteúdo anterior e adicionar o novo conteúdo no final.

Hora de exercitar

Agora chegou a hora de praticar. Acesse agora mesmo o endereço https://gustavoguanabara.github.io/javascript/exercicios/ e execute o **exercício 009** no seu computador e tente atingir esse mesmo resultado em casa, sem copiar o código que eu criei. Nesse momento, a prática é algo que você mais precisa. Se por acaso ficar difícil, pode acessar o repositório público de HTML e CSS e dar uma olhada nos comandos, mas **EVITE COPIAR**.

Tenho uns desafios pra você!

Lá no repositório, além do material em PDF e dos códigos dos exercícios 100% disponíveis, também disponibilizamos alguns **desafios** que devem ser resolvidos. Esses desafios não incluem o código original e você deve tentar chegar à resposta sem copiar nenhum código.

Com todo o conteúdo que vimos até essa aula, você já pode resolver todos os seguintes desafios, do **d001** até o **d010**.

Acesse o repositório público, abra a área do curso de JavaScript e clique no link de acesso aos desafios. Manda ver! Só não fica pedindo a resposta! Você consegue resolver isso sozinho(a)!

Repositório em: https://gustavoguanabara.github.io

Eu já falei sobre isso no YouTube?

Eu sei que às vezes as pessoas gostam mais de assistir vídeos do que ler livros, e é por isso que eu lanço há anos materiais no canal Curso em Vídeo no YouTube. O link que vou compartilhar contigo vai diretamente para a playlist completa do curso que já está totalmente disponível e mostra todos os procedimentos passo-a-passo. Acesse agora mesmo!

Curso em Vídeo: https://www.youtube.com/playlist?list=PLHz_AreHm4dlsK3Nr9GVvXCbpQyHQl1o1